Rubik's Cube Solver

Embedded System Design
CSEE W4840
Final Report

Advisor: Prof. Stephen Edwards
May 16, 2013

Zongheng Wang zw2223

Ifeoma Okereke iro2103

Yin-Chieh Wang yw2491

Heather Fisher hlf2119

Table of Contents

1. Background …..... 3
2. Design …..... 3
3. Implementation …..... 4
    3.1 Software
        3.1.1 The Thistlewaite's Algorithm
        3.1.2 RCS
    3.2 Hardware
        3.2.1 Acceleration Blocks
        3.2.2 VGA Module
4. Results …..... 15
5. Contributions and Teamwork …..... 15
6. Milestone Report …..... 16
7. Challenges and Lessons Learnt …..... 16
8. Future Work …..... 17
9. Conclusion …..... 17
10. References …..... 17
11. Source Code …..... 17

1. Background

The Rubik's Cube. A source of great frustration to many puzzle enthusiasts around the world. But we aim to put an end to those exasperating attempts to solve this colorful cube of misery. Perhaps misery is a bit strong, but they can be annoying to try to solve. We have developed a Rubik's Cube Solver (RCS) that will solve a standard Rubik's Cube using the Thistlewaite's Algorithm. We set out to speed up the solve time of the Thistlewaite's Algorithm by utilizing the FPGA to accelerate selected elements of the algorithm. To make it as easy as possible for the user, we have a pseudo-3D image of a cube that lets the user input the color positions from the cube they are trying to solve. They will then be able to run the solver, which will show step-by-step instructions and color changes on the display to walk the user through solving their cube. With RCS, solving that pesky Rubik's Cube is as simple as pressing buttons on your keyboard.

2. Design

The aim of this project is to develop a Rubik's Cube Solver using the Thistlethwaite's algorithm. In this project, we set out to solve the Rubik's Cube in the shortest possible time taking advantage of the FPGA's speed. We used pure software to test the time it takes to get a Rubik's Cube solution and compare the execution time with software/hardware hybrid approach. This helped us determine the efficiency of using the FPGA to solve the Rubik's Cube. We also designed a user-friendly user interface so that one could input the color configuration of a Rubik's Cube from a PS/2 keyboard and the corresponding solution will be shown on the VGA screen in separate steps.

The VGA part is one of the main design components of this project. How to make sure the user is be able to understand cube representation at the first sight is a difficult challenge for us. With pseudo-3D cube design we can achieve this goal. The user can see the front three faces with ease. Furthermore, by the assistance of our extending lines and projecting faces, the user can control the back three faces that are invisible. The user will not get confused with this user interface.

Keyboard is also a major component for our design. In a Rubik's Cube, each cubie on a face is equivalent to 1-9 on a number pad. Players can keep pressing the number pad to change the color until it fits the physical Rubik's Cube. Furthermore, players can switch between each side by using the Tab key on a keyboard. After that, press the Enter key to start solving a cube. Our system will generate instructions to solve the cube. By pressing the right or left arrow key players can acquire every step of instructions. When players want to restart the game, they just have to press Esc and Rubik's Cube will start again.

The most important of all components; the idea of hardware acceleration is also implemented in our design. There are three most frequently used functions, cycle, twist, and permtonum in our software part. Thus, we use hardware to redesign these three functions. On the FPGA board, performance of hardware is

much better than performance of software. we can save a lot of time with the assistance of hardware acceleration.

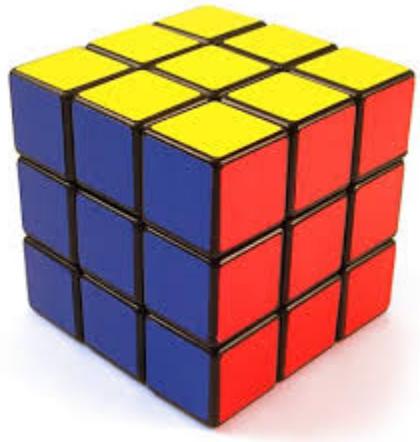

Figure 1: Rubik's Cube

3. Implementation

3.1 Software

3.1.1 The Thistlewaite's Algorithm

The Thistlewaite's Algorithm is based on a mathematical theory called group theory, which studies the algebraic structures known as groups. Group theory is one of the most popular ways of solving Rubik's Cube related math problems [1]. In general, the Thistlewaite's Algorithm solves a cube in four phases. In each phase, certain rotations are restricted so that they cannot be performed in the following phases. For example, rotating the top face clockwise might be legal in phase one, but restricted later. The objective of each phase is to move some target cubies to their expected positions. After the four phases, the cube is solved.

The software of our project is based on the third prize of the short Tom Rokicki's Cube contest [2]. In the original code, the input for the program is the same as that of Mike Reid's cube solver. A solved cube is represented by the output:

UF UR UB UL DF DR DB DL FR FL BR BL UFR URB UBL ULF DRF DFL DLB DBR

Therefore, we need to transform the input received from the user interface into this format so that we could feed it to the program. In our software, the function void inputFormatTransform(int colorConfig[6][9]) does this. After receiving the input, the program starts generating a huge look-up table before solving the cube. This process takes about one third of the entire execution time. Therefore, we

generate this table beforehand and store it in SRAM, since it takes only 30KB and the software would use it frequently. In this way, a bunch of functions in the original code can be eliminated. Next, we used the gprof command for the program and got the following table:

Table 1: Profile data generated from command gprof

% Time	Cumulative Seconds	Self Seconds	Calls	Self ms/Call	Total ms/ Call	name
64.41	0.18	0.18	2646168	0.00	0.00	cycle
14.31	0.22	0.04	794597	0.00	0.00	permtonum
10.73	0.25	0.03	2675292	0.00	0.00	twist
7.16	0.27	0.02	477227	0.00	0.00	getposition
3.58	0.28	0.01	42624	0.00	0.00	numtoperm
0.00	0.28	0.00	661542	0.00	0.00	domove
0.00	0.28	0.00	19629	0.00	0.00	reset
0.00	0.28	0.00	19621	0.00	0.00	setposition
0.00	0.28	0.00	33	0.00	2.31	searchphase
0.00	0.28	0.00	8	0.00	25.53	filltable

From table 1, we find that the top three functions cycle, permtonum and twist are invoked millions of times and take more than 90% of the entire execution time. Therefore, we decided to design hardware to accelerate these three functions. Details of this design are provided in the hardware section.

3.1.2 RCS

Our software guides the flow of this project. Figure 2 shows the flowchart of the RCS program. Upon starting the system, some initialization will be performed such as for the PS2 keyboard and showing the welcome message. Then the program keeps scanning the user input. If the signals are from the tab key and the numpad, the corresponding face will be selected and the color data will be changed both in software and hardware. If the enter key is pressed, then the program starts checking whether the current cube is valid or not. The checking for now is only based on whether there are exactly nine cubies having one same

color and whether the color of the six center cubies have the same color. There is still chance that our checking cannot detect an invalid cube, but this should not happen as long as the user doesn't do that on purpose. Since a precise checking algorithm is much more complicated and is not the goal of this project, we just leave the checking algorithm to what we have now. When the checking fails, an error message will be displayed on the screen and the user can keep on inputting or modifying the color. If the checking passes, the inputFormatTransform function and the solver will be invoked sequentially. After the solution is successfully generated(an invalid cube will lead to an infinite loop of the solver), a simple solution shrinker will be called since in some cases, the solution will include … U1, U2 …, which means rotating the top face clockwise by 90 degrees and then 180 degrees. This will be shrinked into U3, which means rotating the top face counter-clockwise. Eventually, the user enters the demoing stage, where he/she can press left and right arrows to navigate the solution demo at the speed he/she desires. Pressing the escape key here will restart the system from the beginning.

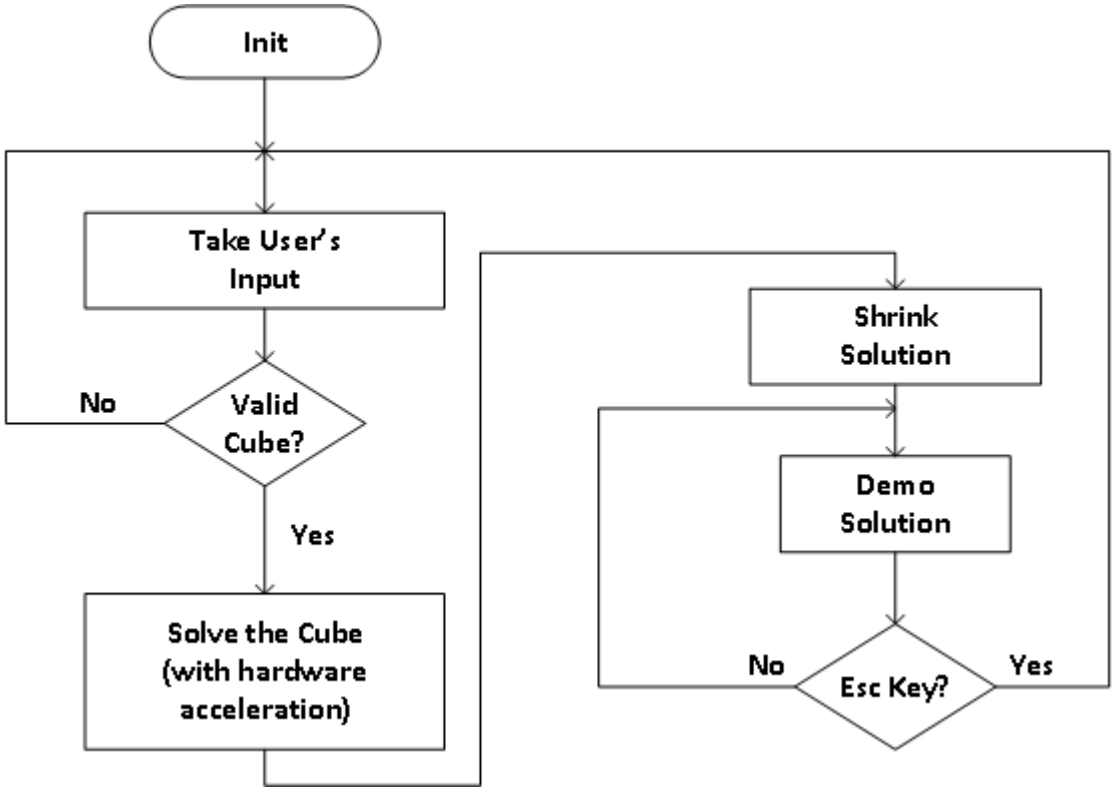

Figure 2: RCS Flowchart

3.2 Hardware

The following chart shows the high-level design of our project. For the PS2 controller we used the keyboard controller that was implemented in lab 2. Therefore we will not go into details of that controller in this report. The SRAM is the memory needed for the NIOS II Processor. In this section we will go into details about the acceleration blocks and the VGA controller..

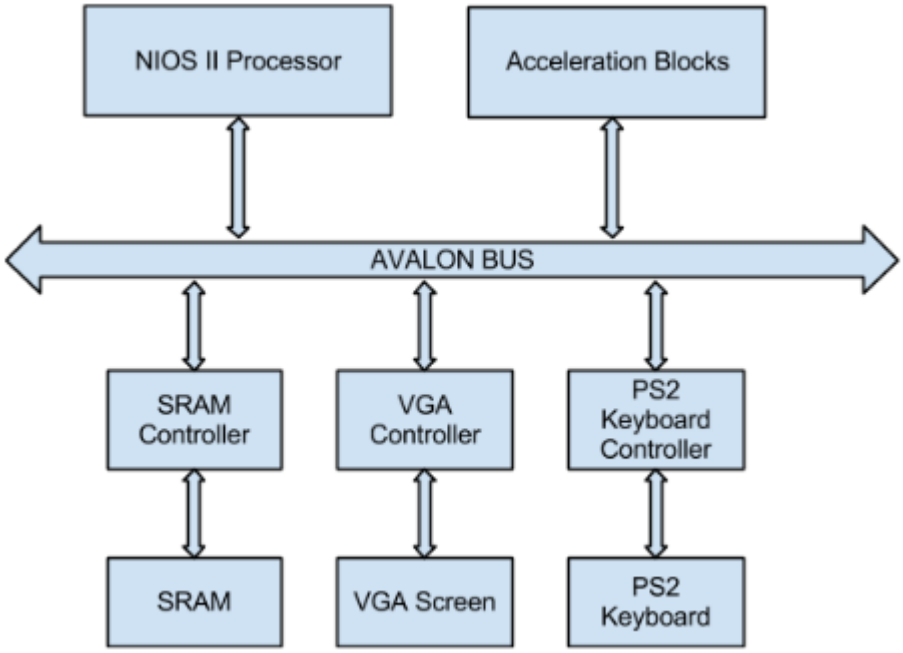

Figure 3: High-level Hardware Overview

3.2.1 Acceleration Blocks

The target functions to accelerate(cycle, permtonum and twist) are mentioned in the software section. The code for these functions are listed below.

```
// Use very ugly and unsafe macro to swap items instead of classic routine with
// pointers for the sole reason of brevity
#define SWAP(a,b) TEMP=a;a=b;b=TEMP;
// number 65='A' is often subtracted to convert char ABC... to number 0,1,2,...
#define CHAROFFSET 65

// Cycles 4 pieces in array p, the piece indices given by a[0..3].
void cycle(char*p,char*a) {
        SWAP(p[*a-CHAROFFSET],p[a[1]-CHAROFFSET]);
        SWAP(p[*a-CHAROFFSET],p[a[2]-CHAROFFSET]);
        SWAP(p[*a-CHAROFFSET],p[a[3]-CHAROFFSET]);
}

// twists i-th piece a+1 times.
void twist(int i,int a){
        i-=CHAROFFSET;
        ori[i]=(ori[i]+a+1)%val[i];
```

```
}

// convert permutation of 4 chars to a number in range 0..23
int permtonum(char* p){
        int n=0;
        int a, b;
        for (a=0; a<4; a++) {
                n*=4-a;
                for(b=a; ++b<4; )
                        if (p[b]<p[a]) n++;
        }
        return n;
}
```

The cycle function basically swaps data from A, B, C, D to D, A, B, C. The twist function can be viewed as bit manipulations. Both these functions are feasible to implement in hardware. Furthermore, these two functions only show up in one another function called domove, which is listed below. To easily understand what these functions do, we should first learn the two global variables: pos[20] and ori[20]. Even though there are 54 colors on the surface of a cube, but there are only 20 structural pieces, 12 side pieces and 8 corner pieces, the center pieces do not count as they cannot be moved. The range of pos is from 0 to 19 each index indicating a different structural piece. The range for ori is from 0 to 1 for side pieces and from 0 to 2 for corner pieces since there are only two possible orientations for side pieces and three possible orientations for corner pieces. The domove function, as explained in the comment, performs a clockwise quarter turn, where the parameter m indicates which side you are rotating now. When executing this function, the corresponding pos and ori variable need to be updated. Therefore, instead of designing acceleration blocks for both the cycle and the twist function, we decided to design hardware for the domove function directly.

```
//do a clockwise quarter turn cube move
void domove(int m){
        char *p=perm+8*m;
        int i=8;

        //cycle the edges
        cycle(pos,p);
        cycle(ori,p);
        //cycle the corners
        cycle(pos,p+4);
        cycle(ori,p+4);

        //twist corners if RLFB
```

```
        if(m<4)
                for(;--i>3;) twist(p[i],i&1);
        //flip edges if FB
        if(m<2)
                for(i=4;i--;) twist(p[i],0);
}
```

The permtonum function converts permutation of 4 chars to a number in range from 0 to 23. It is hard to understand at the first glance. Therefore, we unroll the loops and get the code as follows. It does some comparison and update n based on the result.

```
int permtonum(char* p){
        int n=0;
        if (p[1] < p[0]) n++;
        if (p[2] < p[0]) n++;
        if (p[3] < p[0]) n++;
        n *= 3;

        if (p[2] < p[1]) n++;
        if (p[3] < p[1]) n++;
        n *= 2;

        if (p[3] < p[2]) n++;

        return n;
}
```

Figure 4 is a high level block diagram of the acceleration blocks. The functionality of each component, which is also a separate vhd file in the project directory, is explained below. The only functionality that is not implemented in a component is cycle. As we explained before, cycle swaps data from A, B, C, D to D, A, B C. Therefore, we just 1) read D, 2) read C, write D into register, 3) write C to D, 4) read B, 5) write B to C, 6) read A, 7) write A to B and 8) write D to A. Overall, it takes eight clock cycles to perform a cycle. From the software domove function, there are four cycles in all. One pair for pos and ori at position P and the other pair for pos and ori at position P+4. Since they are independent from each other, we decided to use a true dual port ram with a single clock so that we could perform the two cycles simultaneously.

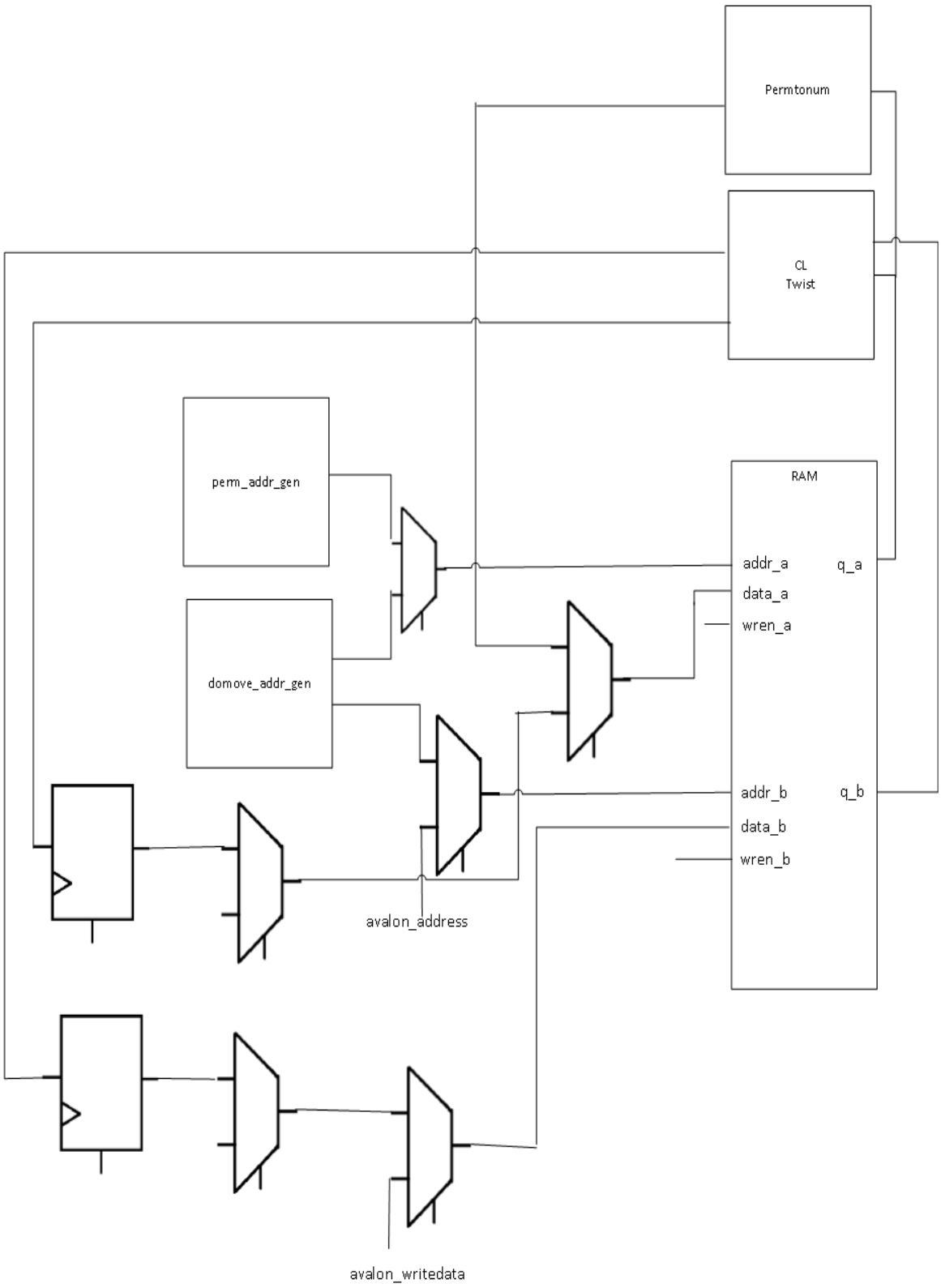

Figure 4: Block Diagram for the Acceleration Blocks

true_dual_port_ram_single_clock

This is where the pos and ori variables are stored. As we explained previously, there are 20 possibilities for the pos and 2 or 3 possibilities for the ori depending on whether it is a side piece or corner piece. Therefore, pos takes 5 bits and ori takes 2 bits, and thus we need a 32x7 bit RAM to store these two variables(first 5 bits for pos and last 2 bits for ori). There are two ports for the RAM and each port can be used to read or write independently. Since reading from and writing to the same port won't happen in our project, we leave it "don't care" while generating this component from the MegaWizard in Quartus II.

twist

This is a combinational logic component to perform the twist function described above. It is put right before where the domove hardware can write back data to the dual port RAM. Designing this way increases the efficiency since the twist hardware is now integrated within the cycle function. In other words, when the hardware tries to perform the swapping function of cycle, it modifies the data before it is fed back to the RAM according to the code of twist. In the software domove function, the parameter m is checked before invoking twist, and thus the value in the corresponding m register has to be the input of the twist block. The loop for invoking twist is unrolled in software and converted to combinational logic.

domove_addr_gen

This is a RAM to store the addresses that the domove function needs to perform cycle(and twist as well, since they are integrated) for different m parameters. Since the cycle swaps four values and each value is a 5-bit address, then all the six faces, a 32x5 bit RAM is needed.

permtonum

After the domove function is finished, the permtonum hardware controlled by the moore state machine will start running automatically. The block is composed of four registers, six comparators and one look-up ROM. The four registers latches the four numbers(e.g. p[0], p[1], p[2] and p[3] in the software, loop unrolled version). The six comparators are used for the six different comparisons. The ROM is used to generate result without calculation on n.

perm_addr_gen

This is a RAM to store the addresses that the permtonum function needs to store the result back to the dual port RAM for pos and ori when it generates a result. Since the the address is also 5-bit and there are five different parameters for permtonum, then a 8x5 bit RAM is needed.

moore_state_machine

The moore state machine controls the entire datapath and generates mux selection signals and write enables signals etc. Figure 5 shows the state diagram. State S0 is the idle state so that the acceleration blocks do nothing. When m is updated to a value other than 7, then it goes to state S1 to S7 where domove is performed. At this time, the avalon interface is disabled temporarily. Then the permtonum starts and iterates until all the five possibilities for permtonum are finished. Then it goes back to the idle state. In addition, this iteration can be interrupted anytime the software wants to perform another domove function. The un-finished permtonum and the corresponding data can be safely discarded since when the new domove function terminates, the permtonum function will start all over again.

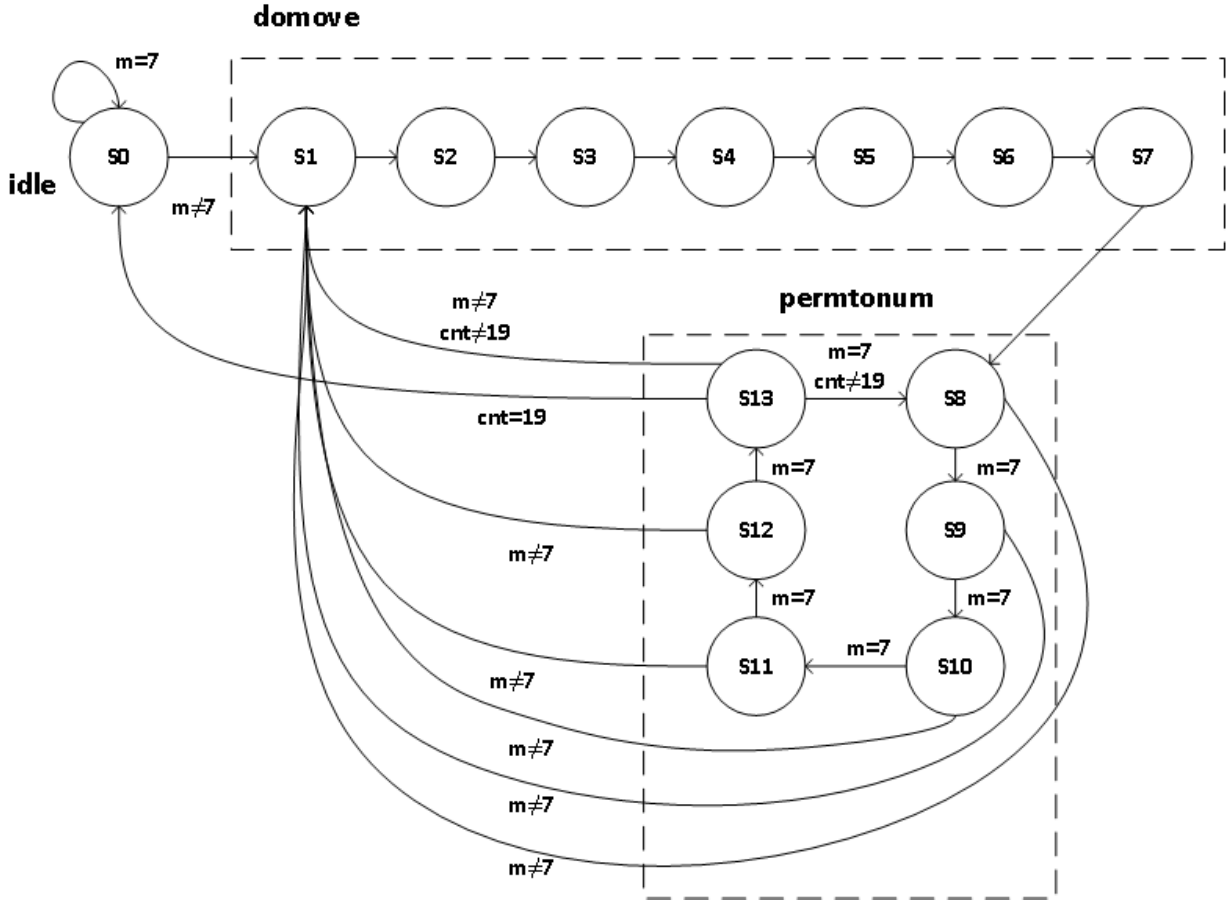

Figure 5: State Diagram for the Acceleration Block

3.2.2 VGA Module

The Rubik's Cube Solver was displayed on the VGA Screen using the active regions of the horizontal and vertical synchronization VGA signals. The 25Mhz clock frequency for the VGA Screen was internally generated in the VGA Controller.

Figure 6 shows the Display Design for the Rubik's Cube Solver. Each face of the Rubik's Cube was displayed using the intersection of the Horizontal and Vertical regions shown in figure 6. For example, to display the top-left face in figure 6, the intersection between the V4Region and HRegion(5 to 3) is detected and the face is displayed. To display the cubie with position number 0 on this face, we detect whether the VGA signals are in the intersection between V4Region(1) and HRegion(5). The black lines are displayed when a region change between cubie locations is sensed. The color of each cubie in a face is stored in a 64x3 bit RAM using the address [(9*face) + position]. Position represents the cubie location on the face as shown in figure 7. This made it easy to read from or write color information for each cubie to the RAM whenever it was needed.

In order to display the selection border for each face, we detect whether the border falls into the vertical and horizontal regions around each face. For example, to display the selection border on the top left face in figure 6, we detect whether the horizontal and vertical VGA signals are around the intersection between the V4Region and the HRegion, but not inside the intersection between these two regions.

To display text on the VGA Screen, we used a character bitmap that contained 8x8 pixel-sized characters. The 640x480 pixels of the VGA screen were divided into 80x60 tiles to enable us display the 8x8 characters easily. The text information was stored in a 256x6 bit RAM. Three regions on the VGA Screen were defined for text display. Whenever there was text available to display and the horizontal and vertical VGA active signals were in the Text Display region, text was displayed on the VGA screen.

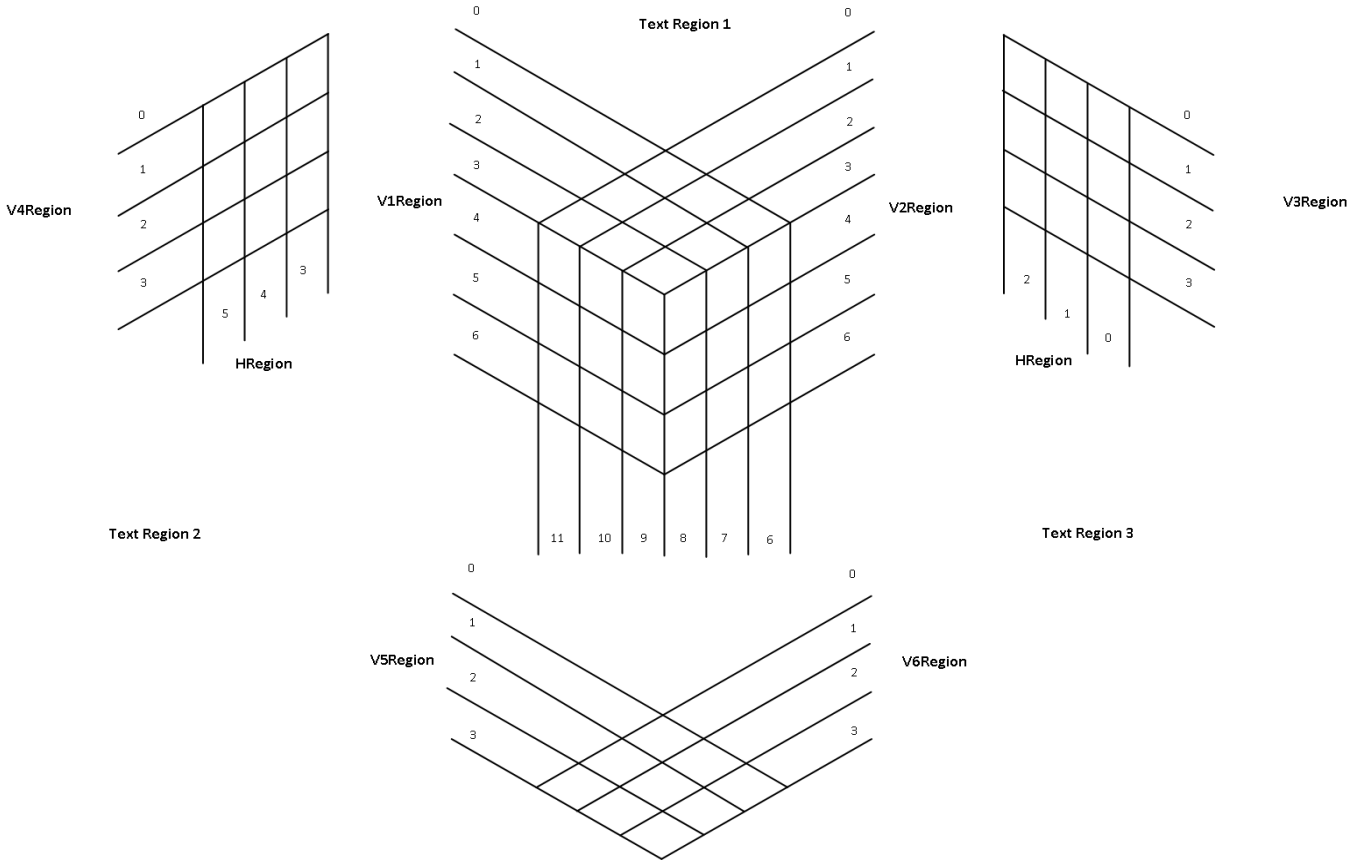

Figure 6: Rubik's Cube Solver VGA Display Design

0	1	2
3	4	5
6	7	8

Figure 7: Diagram showing the position numbers for the Cubies on a Rubik's Cube Face

```
000  : 00011000 ; %     **      %
001  : 00111100 ; %    ****     %
002  : 01100110 ; %   **  **    %
003  : 01111110 ; %   ******    %
004  : 01100110 ; %   **  **    %
005  : 01100110 ; %   **  **    %
006  : 01100110 ; %   **  **    %
007  : 00000000 ; %             %
```

Figure 8: Snapshot of the Bitmap File showing the character ‘A’

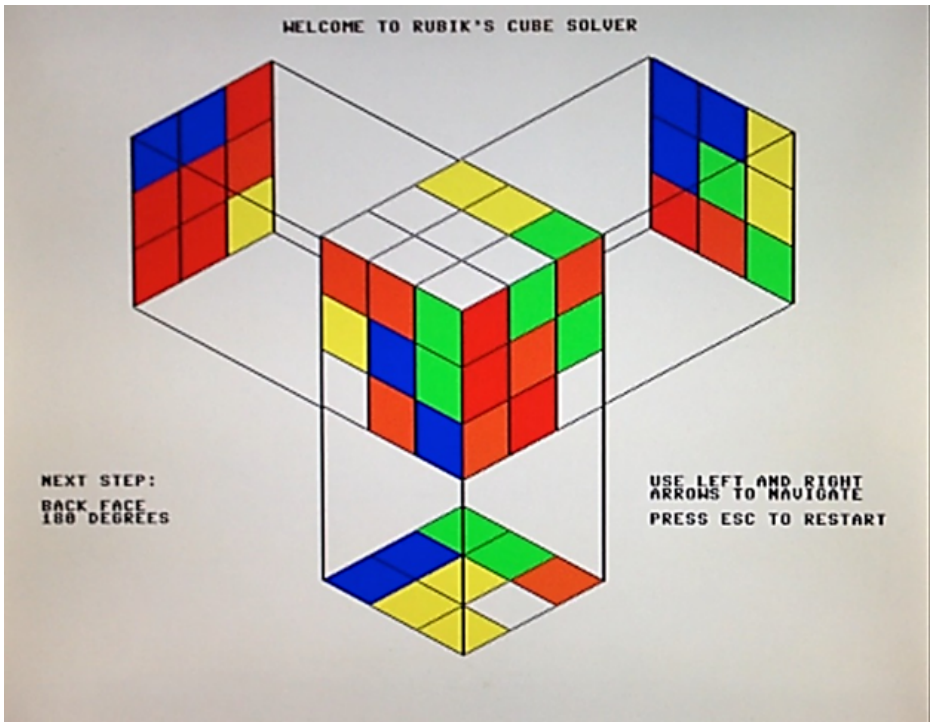

Figure 9: Final Version of the Rubik’s Cube Solver

4. Results

To determine the success of our acceleration blocks, we tested the algorithm on the FPGA without acceleration and on the FPGA with acceleration. For comparison we also timed how long the algorithm took to run on a laptop. The time in which the algorithm solved the cube is also based on the difficulty of the cube. Intuitively, a cube with a simple puzzle configuration will be solved faster than a cube that has a more complex puzzle configuration. The following table shows the results we obtained:

Table 2: Results of Rubik’s Cube Solution Execution Time Test:

FPGA Without Acceleration Blocks	80-100 sec
FPGA with Acceleration Blocks	5-15 sec
Laptop	10-20 msec

5. Contributions and Teamwork

Zongheng Wang	Acceleration blocks, VGA Controller version 3, Software
Ifeoma Okereke	VGA Controller version 1, VGA text display
Heather Fisher	VGA Controller version 2, Software
Yin-Chieh Wang	VGA Controller version 2, PS2 Keyboard

6. Milestone Report

Milestone	Date	Goal	Accomplished
Milestone 1	Apr 6	- Hardware: Design the layout of the user interface Implement the VGA controller for the user input part Integrate the system with the SRAM and PS/2 keyboard controllers - Software: Write the algorithm for transforming the input from the user into Mike Reid's Cube Solver format Test the Thistlethwaite's algorithm on Altera DE2 Board with only NIOS II processor and SRAM	The Thistlewaite's Algorithm works well on FPGA without acceleration blocks. It takes about 90 seconds to solve a cube. Six individual faces of the cube are displayed on the screen.
Milestone 2	Apr 16	- Hardware: Finish the acceleration block Implement the Keyboard controller part that receives input color configurations from the user Integrate the system with all the components so that the modified algorithm can run - Software: Modify and optimize the Thistlethwaite's algorithm and show the solution through printf Integrate the algorithm with acceleration blocks Compare the speed of solving a cube between pure software systems and hybrid software and hardware system	First version of the acceleration blocks take too many hardware resources. Keyboard doesn't work. The Thistlewaite's Algorithm is accelerated and solving a cube takes about 15 seconds. Changed display to show two pseudo 3D cubes.
Milestone 3	Apr 30	- Hardware: Implement the VGA controller for solution demonstration part - Software: Implement the algorithm that coordinates with hardware to show solution demonstration	The new version of the acceleration blocks took much less hardware resources and gave equivalent performance. Changed display to show a single pseudo-3D cube with projection..
Deadline	May 16	Complete project. Prepare for presentation. Write report.	As planned

7. Challenges and Lessons Learnt

The most challenging aspect of this project was that we had to redesign many of the hardware parts of the project multiple times for optimization purpose. We had to redesign the acceleration blocks 3 times, the VGA controller 4 times, and the software portion twice. The end result was that our code became simple, efficient and flexible. Especially in hardware acceleration, the VHDL code not only became simpler, but also saved a lot of hardware logic elements. For an ideal design, to make a trade off between performance and materials is important because minimizing components used can also save budget and power consumption in the hardware. Another challenge we faced was that sometimes the printf function would not work on the NIOS II IDE when we ran the FPGA. We also had to deal with ELF errors when trying to download the project onto the FPGA. When this error occurs it is not possible to download the file to the FPGA because there are errors that prevent communication with the Avalon bus.

There were many lessons that we learned while working on this project. The first important lesson we learned was that VHDL is not like the typical programming languages we are used to working with. You cannot treat it like C or Java and try something and see what works. It is necessary to design your project before you start coding. VHDL is a kind of description programming languages. Describing behavior of logic gates is the main feature for this language. This includes having both block diagrams and timing diagrams. Additionally we learned that if your VHDL file has thousands of lines of code something is very wrong. VHDL code should be short and concise.

8. Future Work

There are several things we would like to add to our project design in the future that we would not have had time to implement in this class. Among those things would include support for non-traditional Rubik's Cube variations. We would like to add animation to show the solution to the cube rather than just changing colors on the screen to show what the next cube state should look like. In addition we would like to use color sensors to allow the user to enter the color inputs of the cubies without having to use the keyboard to enter values in individually.

9. Conclusion

We were able to successfully implement a hardware accelerated Rubik's Cube Solver on the FPGA. Our accelerated version was significantly faster than running the Thistlewaite's Algorithm on the FPGA without the acceleration blocks. After trying different a few different VGA display options we settled on a VGA display that we believe is the most user friendly to make it as easy as possible for the user to enter the inputs.

10. References

[1] Group (mathematics), Wikipedia, [Online] Available: http://en.wikipedia.org/wiki/Group_(mathematics)

[2] Short Cube Program Contest [Online] Available: http://tomas.rokicki.com/cubecontest/

11. Source Code

Note: The drive software for PS2 keyboard is included since there is a little modification. But the sram_controller.vhd and all the hardware for PS2 keyboard are not included since they are all available on the course website.

Source Code – Software

```
------------------------------------------------------------------
main.h
------------------------------------------------------------------

#include <io.h>
#include <system.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <alt_types.h>
#include "alt_up_ps2_port.h"
#include "ps2_keyboard.h"

alt_u8 key = 0;
KB_CODE_TYPE decode_mode;
int status = 0;

unsigned long numOfDomove;
unsigned long numOfPermtonum;

void SET_COLOR(int face, int position, int color)
{
    IOWR_8DIRECT(VGA_RASTER_BASE, face*9+position, color);
}

void SET_BORDER(int face)
{
    IOWR_8DIRECT(VGA_RASTER_BASE, 54, (1<<face));
}

void CLEAR_BORDER(void)
```

```
{
    IOWR_8DIRECT(VGA_RASTER_BASE, 54, 0);
}

char get_char_addr(char ch)
{
    if(ch >= 'A' && ch <= 'Z')
      ch = ch - 'A';
    else if(ch >= '0' && ch <= '9')
      ch = ch - '0' + 26;
    else if(ch == '\n')
      ch = 48;
    else if(ch == ' ')
      ch = 36;
    else if(ch == ':')
      ch = 37;
    else if(ch == '.')
      ch = 38;
    else if(ch == '/')
      ch = 39;
    else if(ch == '-')
      ch = 40;
    else if(ch == '&')
      ch = 43;
    else if(ch == '!')
      ch = 44;
    else if(ch == '\'')
      ch = 45;
    else if(ch == '(')
      ch = 46;
    else if(ch == ')')
      ch = 47;
    else
      ch = 36;
    return ch;
}

void SET_TEXT(int region, const char* text)
{
    int i, j, k, flag;
    char ch;
    if(region == 1) {
      flag = 0;
      for(i=0;i<30;i++) {
            if(flag) {
                  IOWR_8DIRECT(VGA_RASTER_BASE, i+64, 36);
                  continue;
            }
            ch = get_char_addr(text[i]);
            if(ch == 48) {
```

```
                flag = 1;
                IOWR_8DIRECT(VGA_RASTER_BASE, i+64, 36);
            }
            else
                IOWR_8DIRECT(VGA_RASTER_BASE, i+64, ch);
        }
    }
    else if(region == 2) {
        k = 0;
        for(j=0;j<4;j++) {
            flag = 0;
            for(i=0;i<20;i++) {
                if(flag) {
                    IOWR_8DIRECT(VGA_RASTER_BASE, j*20+i+94, 36);
                    continue;
                }
                ch = get_char_addr(text[k++]);
                if(ch == 48) {
                    flag = 1;
                    IOWR_8DIRECT(VGA_RASTER_BASE, j*20+i+94, 36);
                }
                else
                    IOWR_8DIRECT(VGA_RASTER_BASE, j*20+i+94, ch);
            }
        }
    }
    else if(region == 3) {
        k = 0;
        for(j=0;j<4;j++) {
            flag = 0;
            for(i=0;i<20;i++) {
                if(flag) {
                    IOWR_8DIRECT(VGA_RASTER_BASE, j*20+i+174, 36);
                    continue;
                }
                ch = get_char_addr(text[k++]);
                if(ch == 48) {
                    flag = 1;
                    IOWR_8DIRECT(VGA_RASTER_BASE, j*20+i+174, 36);
                }
                else
                    IOWR_8DIRECT(VGA_RASTER_BASE, j*20+i+174, ch);
            }
        }
    }
}

int GET_POS(int i)
{
    return IORD_8DIRECT(ACCELERATION_BASE, i) >> 2;
```

```
}

int GET_ORI(int i)
{
    return IORD_8DIRECT(ACCELERATION_BASE, i) & 3;
}

void SET_POS(int i, int val)
{
    int tmp = GET_ORI(i);
    IOWR_8DIRECT(ACCELERATION_BASE, i, (val << 2) + tmp);
}

void SET_ORI(int i, int val)
{
    int tmp = GET_POS(i);
    IOWR_8DIRECT(ACCELERATION_BASE, i, (tmp << 2) + val);
}

void DOMOVE(int m)
{
    numOfDomove++;
    IOWR_8DIRECT(ACCELERATION_BASE, 20, m);
}

#define SET_PERMTONUM0(i) \
  IOWR_8DIRECT(ACCELERATION_BASE, 21, i)

#define SET_PERMTONUM1(i) \
  IOWR_8DIRECT(ACCELERATION_BASE, 22, i)

#define SET_PERMTONUM2(i) \
  IOWR_8DIRECT(ACCELERATION_BASE, 23, i)

#define SET_PERMTONUM3(i) \
  IOWR_8DIRECT(ACCELERATION_BASE, 24, i)

#define SET_PERMTONUM4(i) \
  IOWR_8DIRECT(ACCELERATION_BASE, 25, i)

#define PERMTONUM0() \
  IORD_8DIRECT(ACCELERATION_BASE, 21)

#define PERMTONUM1() \
  IORD_8DIRECT(ACCELERATION_BASE, 22)

#define PERMTONUM2() \
  IORD_8DIRECT(ACCELERATION_BASE, 23)

#define PERMTONUM3() \
```

```
  IORD_8DIRECT(ACCELERATION_BASE, 24)

#define PERMTONUM4() \
  IORD_8DIRECT(ACCELERATION_BASE, 25)

//A solved cube format:
//UF UR UB UL DF DR DB DL FR FL BR BL UFR URB UBL ULF DRF DFL DLB DBR
const char CENTER_COLOR_POSITION[6] = {'F', 'R', 'U', 'B', 'L', 'D'};
const int SIDE_INDEX_1[12] = {2,2,2,2,5,5,5,5,0,0,3,3};
const int SIDE_INDEX_2[12] = {7,5,1,3,3,1,5,7,5,3,3,5};
const int SIDE_INDEX_3[12] = {0,1,3,4,0,1,3,4,1,4,1,4};
const int SIDE_INDEX_4[12] = {1,1,1,1,7,7,7,7,3,5,5,3};
const int CORNER_INDEX_1[8] = {2,2,2,2,5,5,5,5};
const int CORNER_INDEX_2[8] = {8,2,0,6,0,6,8,2};
const int CORNER_INDEX_3[8] = {0,1,3,4,1,0,4,3};
const int CORNER_INDEX_4[8] = {2,2,2,2,6,6,6,6};
const int CORNER_INDEX_5[8] = {1,3,4,0,0,4,3,1};
const int CORNER_INDEX_6[8] = {0,0,0,0,8,8,8,8};

int colorConfig[6][9];
int colorConfigOld[6][9];

char transformedInputFormat[20][3];
char solution[80][3];

char
    // RLFBUD is the face order used for input, so that a correctly oriented
    // piece in the input has its 'highest value' facelet first. The rest of
the
    // program uses moves in FBRLUD order.
    *faces="RLFBUD",
    // I use char arrays here cause they can be initialised with a string
    // which is shorter than initialising other arrays.
    // Internally cube uses slightly different ordering to the input so that
    //  orbits of stage 4 are contiguous. Note also that the two corner
orbits
    //  are diametrically opposite each other.
    //input:  UF UR UB UL  DF DR DB DL  FR FL BR BL  UFR URB UBL ULF   DRF
DFL DLB DBR
    //        A  B  C  D   E  F  G  H   I  J  K  L   M   N   O   P     Q   R
S   T
    //        A  E  C  G   B  F  D  H   I  J  K  L   M   S   N   T     R   O
Q   P
    //intrnl: UF DF UB DB  UR DR UL DL  FR FL BR BL  UFR UBL DFL DBR   DLB
DRF URB ULF
    *order="AECGBFDHIJKLMSNTROQP",
    //To quickly recognise the pieces, I construct an integer by setting a
bit for each
    // facelet. The unique result is then found on the list below to map it
to the correct
```

```
    // cubelet of the cube.
    //intrnl: UF DF UB DB  UR DR UL DL  FR FL BR BL  UFR UBL DFL DBR   DLB
DRF URB ULF
   //bithash:20,36,24,40, 17,33,18,34, 5, 6, 9, 10, 21, 26, 38, 41,   42, 37,
25, 22
    *bithash="TdXhQaRbEFIJUZfijeYV",
    //Each move consists of two 4-cycles. This string contains these in
FBRLUD order.
    //intrnl: UF DF UB DB  UR DR UL DL  FR FL BR BL  UFR UBL DFL DBR   DLB
DRF URB ULF
    //        A  B  C  D   E  F  G  H   I  J  K  L   M   N   O   P     Q   R
S   T
    //*perm="AIBJTMROCLDKSNQPEKFIMSPRGJHLNTOQAGCEMTNSBFDHORPQ",   //size: 48

    // current cube position
    //pos[20],ori[20],
    val[20],
    // temporary variable used in swap macro
    //TEMP,
    // pruning tables, 2 for each phase
    *tables[8];
    // current phase solution
int move[20],moveamount[20],
    // current phase being searched (0,2,4,6 for phases 1 to 4)
    phase;
    // Length of pruning tables. (one dummy in phase 1);
    //tablesize[]={1,4096,  6561,4096,  256,1536,  13824,576};   //30946
bytes

char table0[] =
{1};
char table1[] =
{1, 0,     0,    5,    0,    6,    5,    0,    0,    5,    6,    0,    5,    0,   0,
    7,     0,    4,    4,    0,
4,  0,     0,    5,    4,    0,    0,    6,    0,    5,    6,    0,    0,    4,   4,
    0,     4,    0,    0,    6,
4,  0,     0,    5,    0,    6,    5,    0,    5,    0,    0,    5,    0,    6,   5,
    0,     0,    5,    6,    0,
5,  0,     0,    7,    0,    4,    4,    0,    4,    0,    0,    5,    4,    0,   0,
    6,     0,    5,    6,    0,
5,  0,     0,    4,    0,    6,    5,    0,    0,    5,    6,    0,    4,    0,   0,
    7,     5,    0,    0,    4,
0,  6,     5,    0,    0,    5,    6,    0,    4,    0,    0,    7,    0,    5,   5,
    0,     5,    0,    0,    6,
5,  0,     0,    6,    0,    6,    6,    0,    0,    4,    4,    0,    4,    0,   0,
    6,     4,    0,    0,    5,
0,  6,     5,    0,    5,    0,    0,    4,    0,    6,    5,    0,    0,    5,   6,
    0,     4,    0,    0,    7,
5,  0,     0,    4,    0,    6,    5,    0,    0,    5,    6,    0,    4,    0,   0,
    7,     0,    5,    5,    0,
```

```
5,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,  0,  5,  0,  0,
    5,  0,  6,  5,  0,
0,  5,  6,  0,  5,  0,  0,  7,  0,  5,  5,  0,  5,  0,  0,
    6,  5,  0,  0,  6,
0,  6,  6,  0,  0,  5,  5,  0,  5,  0,  0,  6,  5,  0,  0,
    6,  0,  6,  6,  0,
7,  0,  0,  7,  0,  7,  7,  0,  0,  7,  7,  0,  7,  0,  0,
    5,  0,  5,  5,  0,
4,  0,  0,  5,  4,  0,  0,  5,  0,  6,  6,  0,  4,  0,  0,
    4,  0,  5,  4,  0,
0,  5,  5,  0,  5,  0,  0,  6,  4,  0,  0,  4,  0,  5,  5,
    0,  0,  4,  5,  0,
5,  0,  0,  6,  0,  5,  5,  0,  5,  0,  0,  6,  5,  0,  0,
    6,  0,  6,  6,  0,
4,  0,  0,  3,  0,  5,  4,  0,  0,  4,  5,  0,  4,  0,  0,
    6,  0,  5,  4,  0,
5,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,  0,  0,  4,  5,
    0,  5,  0,  0,  6,
5,  0,  0,  6,  0,  6,  6,  0,  5,  0,  0,  6,  0,  7,  7,
    0,  0,  7,  7,  0,
7,  0,  0,  6,  4,  0,  0,  3,  0,  5,  4,  0,  0,  4,  5,
    0,  4,  0,  0,  6,
0,  5,  4,  0,  5,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,
    0,  0,  4,  5,  0,
5,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,  0,  5,  0,  0,
    6,  0,  7,  7,  0,
0,  7,  7,  0,  7,  0,  0,  6,  0,  4,  4,  0,  5,  0,  0,
    5,  5,  0,  0,  5,
0,  5,  5,  0,  5,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  5,
5,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,
    5,  0,  7,  7,  0,
7,  0,  0,  6,  7,  0,  0,  6,  0,  6,  6,  0,  0,  5,  5,
    0,  4,  0,  0,  5,
4,  0,  0,  5,  0,  6,  6,  0,  4,  0,  0,  3,  0,  5,  4,
    0,  0,  4,  5,  0,
4,  0,  0,  6,  4,  0,  0,  3,  0,  5,  4,  0,  0,  4,  5,
    0,  4,  0,  0,  6,
0,  4,  4,  0,  5,  0,  0,  5,  5,  0,  0,  5,  0,  5,  5,
    0,  4,  0,  0,  4,
0,  5,  4,  0,  0,  5,  5,  0,  5,  0,  0,  6,  0,  5,  4,
    0,  5,  0,  0,  6,
5,  0,  0,  6,  0,  6,  6,  0,  0,  5,  4,  0,  5,  0,  0,
    6,  5,  0,  0,  6,
0,  6,  6,  0,  5,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  5,
4,  0,  0,  4,  0,  5,  5,  0,  0,  4,  5,  0,  5,  0,  0,
    6,  0,  4,  5,  0,
5,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,  0,  0,  4,  5,
    0,  5,  0,  0,  6,
```

```
5,  0,  0,  6,  0,  6,  6,  0,  5,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  6,  0,
6,  0,  0,  5,  0,  5,  5,  0,  5,  0,  0,  6,  5,  0,  0,
    6,  0,  6,  6,  0,
5,  0,  0,  6,  0,  7,  7,  0,  0,  7,  7,  0,  7,  0,  0,
    6,  5,  0,  0,  6,
0,  7,  7,  0,  0,  7,  7,  0,  7,  0,  0,  6,  0,  7,  7,
    0,  7,  0,  0,  6,
7,  0,  0,  6,  0,  6,  6,  0,  5,  0,  0,  2,  0,  4,  3,
    0,  0,  3,  4,  0,
4,  0,  0,  6,  0,  4,  3,  0,  5,  0,  0,  5,  4,  0,  0,
    5,  0,  5,  5,  0,
0,  3,  4,  0,  4,  0,  0,  5,  5,  0,  0,  5,  0,  5,  5,
    0,  4,  0,  0,  6,
0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,  6,  0,  4,  3,
    0,  5,  0,  0,  5,
4,  0,  0,  5,  0,  5,  5,  0,  5,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  6,  0,
6,  0,  0,  5,  4,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,
    0,  7,  0,  0,  5,
0,  6,  6,  0,  7,  0,  0,  6,  6,  0,  0,  6,  0,  7,  6,
    0,  0,  3,  4,  0,
4,  0,  0,  5,  5,  0,  0,  5,  0,  5,  5,  0,  4,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  6,  0,  7,  0,  0,  5,  5,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  6,  0,
6,  0,  0,  5,  0,  6,  6,  0,  6,  0,  0,  6,  7,  0,  0,
    6,  0,  6,  7,  0,
4,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,
    6,  0,  6,  6,  0,
7,  0,  0,  6,  6,  0,  0,  6,  0,  7,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  6,
7,  0,  0,  6,  0,  6,  7,  0,  7,  0,  0,  7,  0,  6,  7,
    0,  0,  7,  6,  0,
7,  0,  0,  8,  0,  4,  4,  0,  5,  0,  0,  6,  5,  0,  0,
    6,  0,  5,  5,  0,
4,  0,  0,  5,  0,  5,  5,  0,  0,  4,  5,  0,  4,  0,  0,
    6,  4,  0,  0,  5,
0,  5,  4,  0,  0,  5,  5,  0,  4,  0,  0,  6,  0,  5,  5,
    0,  5,  0,  0,  6,
5,  0,  0,  6,  0,  6,  6,  0,  4,  0,  0,  4,  0,  5,  4,
    0,  0,  4,  5,  0,
3,  0,  0,  6,  0,  5,  5,  0,  5,  0,  0,  6,  4,  0,  0,
    6,  0,  6,  6,  0,
0,  5,  5,  0,  4,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,
    0,  5,  0,  0,  7,
0,  7,  7,  0,  0,  7,  7,  0,  6,  0,  0,  6,  4,  0,  0,
    4,  0,  5,  4,  0,
0,  4,  5,  0,  3,  0,  0,  6,  0,  5,  5,  0,  5,  0,  0,
    6,  4,  0,  0,  6,
```

```
0,  6,  6,  0,  0,  5,  5,  0,  4,  0,  0,  6,  5,  0,  0,
    6,  0,  6,  6,  0,
5,  0,  0,  7,  0,  7,  7,  0,  0,  7,  7,  0,  6,  0,  0,
    6,  0,  5,  5,  0,
4,  0,  0,  5,  4,  0,  0,  5,  0,  5,  5,  0,  5,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  6,  0,  6,  0,  0,  5,  5,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  6,  0,
6,  0,  0,  5,  0,  7,  7,  0,  7,  0,  0,  6,  7,  0,  0,
    6,  0,  6,  6,  0,
6,  0,  0,  4,  0,  5,  5,  0,  0,  5,  5,  0,  4,  0,  0,
    6,  0,  5,  5,  0,
5,  0,  0,  6,  5,  0,  0,  5,  0,  6,  5,  0,  0,  5,  5,
    0,  5,  0,  0,  5,
5,  0,  0,  6,  0,  5,  6,  0,  6,  0,  0,  6,  0,  7,  7,
    0,  0,  7,  7,  0,
6,  0,  0,  6,  0,  5,  5,  0,  5,  0,  0,  5,  5,  0,  0,
    5,  0,  5,  5,  0,
6,  0,  0,  6,  0,  7,  6,  0,  0,  6,  6,  0,  6,  0,  0,
    6,  6,  0,  0,  6,
0,  6,  6,  0,  0,  6,  7,  0,  6,  0,  0,  5,  0,  7,  6,
    0,  7,  0,  0,  7,
6,  0,  0,  6,  0,  7,  6,  0,  0,  5,  5,  0,  5,  0,  0,
    5,  5,  0,  0,  5,
0,  5,  5,  0,  6,  0,  0,  6,  0,  7,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  5,
6,  0,  0,  6,  0,  6,  6,  0,  0,  6,  7,  0,  6,  0,  0,
    6,  0,  6,  7,  0,
6,  0,  0,  6,  7,  0,  0,  7,  0,  6,  7,  0,  6,  0,  0,
    6,  0,  6,  7,  0,
0,  7,  6,  0,  6,  0,  0,  4,  0,  7,  6,  0,  7,  0,  0,
    6,  6,  0,  0,  5,
0,  6,  5,  0,  0,  6,  7,  0,  6,  0,  0,  5,  7,  0,  0,
    6,  0,  5,  6,  0,
7,  0,  0,  6,  0,  7,  6,  0,  0,  6,  7,  0,  6,  0,  0,
    7,  5,  0,  0,  3,
0,  5,  4,  0,  0,  4,  5,  0,  3,  0,  0,  6,  0,  4,  4,
    0,  5,  0,  0,  6,
4,  0,  0,  5,  0,  5,  5,  0,  0,  4,  4,  0,  4,  0,  0,
    5,  5,  0,  0,  6,
0,  5,  5,  0,  5,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  5,
0,  5,  4,  0,  4,  0,  0,  5,  4,  0,  0,  5,  0,  6,  5,
    0,  5,  0,  0,  6,
0,  7,  6,  0,  0,  6,  7,  0,  6,  0,  0,  6,  5,  0,  0,
    6,  0,  6,  7,  0,
0,  7,  6,  0,  6,  0,  0,  6,  0,  7,  6,  0,  7,  0,  0,
    6,  6,  0,  0,  6,
0,  7,  6,  0,  0,  4,  5,  0,  4,  0,  0,  5,  4,  0,  0,
    5,  0,  5,  6,  0,
```

```
5,  0,  0,  6,  0,  6,  7,  0,  0,  7,  6,  0,  6,  0,  0,
    6,  5,  0,  0,  6,
0,  7,  6,  0,  0,  6,  7,  0,  6,  0,  0,  6,  0,  6,  7,
    0,  6,  0,  0,  6,
7,  0,  0,  6,  0,  6,  7,  0,  5,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  6,  0,
6,  0,  0,  5,  0,  7,  6,  0,  7,  0,  0,  7,  6,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  7,  0,  6,  0,  0,  6,  7,  0,  0,  7,  0,  6,  6,
    0,  7,  0,  0,  7,
0,  6,  7,  0,  0,  7,  6,  0,  7,  0,  0,  7,  0,  5,  5,
    0,  6,  0,  0,  5,
6,  0,  0,  5,  0,  4,  4,  0,  5,  0,  0,  5,  0,  6,  6,
    0,  0,  5,  5,  0,
5,  0,  0,  5,  5,  0,  0,  5,  0,  5,  5,  0,  0,  6,  6,
    0,  5,  0,  0,  5,
0,  6,  6,  0,  6,  0,  0,  6,  6,  0,  0,  6,  0,  6,  6,
    0,  6,  0,  0,  5,
0,  5,  5,  0,  0,  5,  5,  0,  5,  0,  0,  4,  0,  6,  5,
    0,  6,  0,  0,  6,
5,  0,  0,  5,  0,  6,  5,  0,  0,  6,  6,  0,  6,  0,  0,
    5,  6,  0,  0,  6,
0,  6,  6,  0,  6,  0,  0,  6,  0,  7,  6,  0,  0,  7,  6,
    0,  7,  0,  0,  7,
6,  0,  0,  5,  0,  5,  5,  0,  0,  5,  5,  0,  5,  0,  0,
    4,  0,  6,  6,  0,
6,  0,  0,  6,  6,  0,  0,  5,  0,  6,  6,  0,  0,  5,  6,
    0,  5,  0,  0,  5,
6,  0,  0,  6,  0,  5,  6,  0,  6,  0,  0,  6,  0,  6,  7,
    0,  0,  6,  7,  0,
7,  0,  0,  7,  0,  6,  6,  0,  6,  0,  0,  5,  6,  0,  0,
    5,  0,  5,  5,  0,
6,  0,  0,  6,  0,  6,  6,  0,  0,  6,  5,  0,  6,  0,  0,
    7,  6,  0,  0,  6,
0,  5,  6,  0,  0,  6,  6,  0,  6,  0,  0,  7,  0,  6,  6,
    0,  6,  0,  0,  8,
6,  0,  0,  8,  0,  7,  7,  0,  0,  4,  4,  0,  5,  0,  0,
    6,  5,  0,  0,  6,
0,  5,  5,  0,  4,  0,  0,  4,  0,  5,  4,  0,  0,  4,  5,
    0,  3,  0,  0,  6,
4,  0,  0,  4,  0,  5,  4,  0,  0,  4,  5,  0,  3,  0,  0,
    6,  0,  5,  5,  0,
4,  0,  0,  5,  4,  0,  0,  5,  0,  5,  5,  0,  4,  0,  0,
    5,  0,  5,  5,  0,
0,  4,  5,  0,  4,  0,  0,  6,  0,  5,  5,  0,  5,  0,  0,
    6,  4,  0,  0,  6,
0,  6,  6,  0,  0,  5,  5,  0,  5,  0,  0,  6,  4,  0,  0,
    6,  0,  6,  6,  0,
5,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,
    5,  4,  0,  0,  5,
```

```
0,  5,  4,  0,  0,  5,  5,  0,  4,  0,  0,  6,  0,  5,  5,
    0,  4,  0,  0,  6,
5,  0,  0,  6,  0,  6,  6,  0,  0,  5,  5,  0,  4,  0,  0,
    6,  5,  0,  0,  6,
0,  6,  6,  0,  5,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  5,
0,  5,  5,  0,  5,  0,  0,  6,  5,  0,  0,  6,  0,  6,  6,
    0,  5,  0,  0,  7,
0,  7,  7,  0,  0,  7,  7,  0,  6,  0,  0,  6,  5,  0,  0,
    7,  0,  7,  7,  0,
0,  7,  7,  0,  6,  0,  0,  6,  0,  7,  7,  0,  7,  0,  0,
    6,  7,  0,  0,  6,
0,  6,  6,  0,  5,  0,  0,  3,  0,  5,  4,  0,  0,  4,  5,
    0,  3,  0,  0,  6,
0,  5,  4,  0,  4,  0,  0,  5,  4,  0,  0,  5,  0,  6,  5,
    0,  0,  4,  5,  0,
4,  0,  0,  5,  4,  0,  0,  5,  0,  5,  6,  0,  5,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  6,  0,  6,  0,  0,  5,  0,  4,  4,  0,  5,  0,  0,
    6,  4,  0,  0,  5,
0,  5,  5,  0,  5,  0,  0,  6,  0,  7,  6,  0,  0,  6,  7,
    0,  6,  0,  0,  6,
5,  0,  0,  6,  0,  6,  7,  0,  0,  7,  6,  0,  6,  0,  0,
    6,  0,  7,  6,  0,
7,  0,  0,  7,  6,  0,  0,  6,  0,  6,  6,  0,  0,  4,  4,
    0,  4,  0,  0,  5,
5,  0,  0,  6,  0,  5,  5,  0,  5,  0,  0,  6,  0,  6,  7,
    0,  0,  7,  6,  0,
6,  0,  0,  6,  5,  0,  0,  6,  0,  7,  6,  0,  0,  6,  7,
    0,  6,  0,  0,  6,
0,  6,  7,  0,  6,  0,  0,  6,  7,  0,  0,  7,  0,  6,  6,
    0,  5,  0,  0,  6,
0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,  5,  0,  7,  6,
    0,  7,  0,  0,  6,
6,  0,  0,  6,  0,  7,  6,  0,  0,  6,  7,  0,  6,  0,  0,
    6,  7,  0,  0,  6,
0,  6,  7,  0,  7,  0,  0,  7,  0,  6,  7,  0,  0,  7,  6,
    0,  7,  0,  0,  7,
6,  0,  0,  4,  0,  5,  5,  0,  0,  5,  5,  0,  4,  0,  0,
    6,  0,  5,  5,  0,
5,  0,  0,  5,  5,  0,  0,  5,  0,  5,  5,  0,  0,  5,  5,
    0,  5,  0,  0,  5,
5,  0,  0,  5,  0,  5,  5,  0,  6,  0,  0,  6,  0,  6,  7,
    0,  0,  7,  6,  0,
6,  0,  0,  4,  0,  5,  5,  0,  5,  0,  0,  6,  5,  0,  0,
    5,  0,  6,  5,  0,
6,  0,  0,  6,  0,  7,  6,  0,  0,  6,  6,  0,  6,  0,  0,
    6,  6,  0,  0,  6,
0,  7,  6,  0,  0,  6,  6,  0,  6,  0,  0,  5,  0,  7,  6,
    0,  7,  0,  0,  6,
```

```
6,  0,  0,  5,  0,  6,  5,  0,  0,  5,  5,  0,  5,  0,  0,
    5,  5,  0,  0,  6,
0,  5,  6,  0,  6,  0,  0,  6,  0,  6,  6,  0,  0,  6,  7,
    0,  6,  0,  0,  5,
6,  0,  0,  6,  0,  6,  6,  0,  0,  6,  7,  0,  6,  0,  0,
    6,  0,  6,  7,  0,
6,  0,  0,  5,  7,  0,  0,  6,  0,  5,  6,  0,  6,  0,  0,
    6,  0,  7,  7,  0,
0,  7,  7,  0,  6,  0,  0,  6,  0,  7,  6,  0,  7,  0,  0,
    7,  6,  0,  0,  6,
0,  7,  6,  0,  0,  6,  7,  0,  6,  0,  0,  6,  7,  0,  0,
    7,  0,  6,  7,  0,
7,  0,  0,  6,  0,  7,  6,  0,  0,  6,  7,  0,  6,  0,  0,
    7,  0,  5,  5,  0,
6,  0,  0,  5,  6,  0,  0,  5,  0,  4,  4,  0,  6,  0,  0,
    5,  0,  5,  5,  0,
0,  5,  5,  0,  5,  0,  0,  4,  6,  0,  0,  5,  0,  5,  5,
    0,  0,  5,  5,  0,
5,  0,  0,  4,  0,  6,  6,  0,  6,  0,  0,  5,  6,  0,  0,
    5,  0,  5,  5,  0,
5,  0,  0,  5,  0,  6,  6,  0,  0,  5,  5,  0,  5,  0,  0,
    5,  0,  6,  5,  0,
6,  0,  0,  6,  5,  0,  0,  5,  0,  6,  5,  0,  0,  6,  6,
    0,  6,  0,  0,  6,
6,  0,  0,  5,  0,  6,  6,  0,  6,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  5,  0,
6,  0,  0,  7,  5,  0,  0,  5,  0,  5,  5,  0,  0,  6,  6,
    0,  5,  0,  0,  5,
0,  6,  6,  0,  6,  0,  0,  5,  6,  0,  0,  6,  0,  6,  6,
    0,  0,  5,  6,  0,
5,  0,  0,  5,  6,  0,  0,  6,  0,  5,  6,  0,  6,  0,  0,
    6,  0,  5,  6,  0,
0,  6,  6,  0,  6,  0,  0,  7,  0,  6,  6,  0,  6,  0,  0,
    6,  6,  0,  0,  6,
0,  6,  6,  0,  6,  0,  0,  6,  0,  7,  6,  0,  0,  7,  6,
    0,  7,  0,  0,  7,
6,  0,  0,  6,  0,  6,  7,  0,  0,  6,  7,  0,  7,  0,  0,
    7,  0,  6,  6,  0,
6,  0,  0,  8,  6,  0,  0,  8,  0,  7,  7,  0,  5,  0,  0,
    4,  0,  4,  3,  0,
0,  3,  4,  0,  2,  0,  0,  6,  0,  5,  4,  0,  4,  0,  0,
    5,  3,  0,  0,  5,
0,  5,  5,  0,  0,  4,  5,  0,  3,  0,  0,  5,  4,  0,  0,
    5,  0,  5,  5,  0,
4,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,
    6,  0,  5,  4,  0,
4,  0,  0,  5,  3,  0,  0,  5,  0,  5,  5,  0,  5,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  6,  0,  6,  0,  0,  5,  4,  0,  0,  7,  0,  6,  6,
    0,  0,  6,  6,  0,
```

```
6,  0,  0,  5,  0,  7,  6,  0,  6,  0,  0,  7,  6,  0,  0,
    6,  0,  6,  6,  0,
0,  4,  5,  0,  3,  0,  0,  5,  4,  0,  0,  5,  0,  5,  5,
    0,  4,  0,  0,  7,
0,  6,  6,  0,  0,  6,  6,  0,  6,  0,  0,  5,  5,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  6,  0,  6,  0,  0,  5,  0,  6,  7,  0,  6,  0,  0,
    6,  6,  0,  0,  7,
0,  6,  6,  0,  4,  0,  0,  6,  0,  6,  6,  0,  0,  6,  6,
    0,  6,  0,  0,  6,
0,  7,  6,  0,  6,  0,  0,  7,  6,  0,  0,  6,  0,  6,  6,
    0,  0,  6,  7,  0,
6,  0,  0,  6,  6,  0,  0,  7,  0,  6,  6,  0,  7,  0,  0,
    7,  0,  6,  7,  0,
0,  7,  6,  0,  7,  0,  0,  8,  0,  6,  6,  0,  5,  0,  0,
    4,  5,  0,  0,  4,
0,  5,  5,  0,  5,  0,  0,  5,  0,  6,  5,  0,  0,  6,  5,
    0,  5,  0,  0,  5,
5,  0,  0,  5,  0,  5,  6,  0,  0,  5,  6,  0,  5,  0,  0,
    5,  0,  6,  6,  0,
6,  0,  0,  6,  6,  0,  0,  6,  0,  6,  6,  0,  6,  0,  0,
    5,  0,  5,  5,  0,
0,  5,  5,  0,  5,  0,  0,  4,  0,  6,  5,  0,  6,  0,  0,
    6,  5,  0,  0,  5,
0,  6,  5,  0,  0,  6,  6,  0,  6,  0,  0,  6,  6,  0,  0,
    6,  0,  5,  6,  0,
6,  0,  0,  7,  0,  7,  7,  0,  0,  6,  6,  0,  6,  0,  0,
    7,  6,  0,  0,  5,
0,  5,  5,  0,  0,  5,  5,  0,  5,  0,  0,  4,  0,  6,  6,
    0,  6,  0,  0,  6,
6,  0,  0,  6,  0,  6,  5,  0,  0,  5,  6,  0,  5,  0,  0,
    5,  6,  0,  0,  6,
0,  5,  6,  0,  6,  0,  0,  7,  0,  6,  6,  0,  0,  7,  7,
    0,  6,  0,  0,  7,
0,  6,  6,  0,  6,  0,  0,  5,  6,  0,  0,  5,  0,  5,  5,
    0,  6,  0,  0,  6,
0,  6,  6,  0,  0,  6,  5,  0,  6,  0,  0,  7,  6,  0,  0,
    6,  0,  5,  6,  0,
0,  6,  6,  0,  6,  0,  0,  7,  0,  6,  6,  0,  6,  0,  0,
    7,  6,  0,  0,  7,
0,  8,  8,  0,  0,  6,  6,  0,  5,  0,  0,  4,  5,  0,  0,
    4,  0,  5,  5,  0,
6,  0,  0,  5,  0,  5,  5,  0,  0,  5,  5,  0,  5,  0,  0,
    4,  6,  0,  0,  5,
0,  5,  5,  0,  0,  5,  5,  0,  5,  0,  0,  4,  0,  6,  6,
    0,  6,  0,  0,  5,
6,  0,  0,  5,  0,  5,  5,  0,  5,  0,  0,  5,  0,  6,  5,
    0,  0,  6,  5,  0,
5,  0,  0,  5,  0,  6,  5,  0,  6,  0,  0,  6,  5,  0,  0,
    5,  0,  6,  5,  0,
```

```
0,  6,     6,    0,    6,    0,    0,    6,    6,    0,    0,    6,    0,    6,   5,
    0,     6,    0,    0,    6,
0,  6,     6,    0,    0,    6,    5,    0,    6,    0,    0,    7,    5,    0,   0,
    5,     0,    5,    6,    0,
0,  5,     6,    0,    5,    0,    0,    5,    0,    6,    6,    0,    6,    0,   0,
    6,     6,    0,    0,    6,
0,  5,     6,    0,    0,    5,    6,    0,    5,    0,    0,    5,    6,    0,   0,
    6,     0,    5,    6,    0,
6,  0,     0,    6,    0,    5,    6,    0,    0,    6,    6,    0,    6,    0,   0,
    7,     0,    6,    6,    0,
6,  0,     0,    6,    6,    0,    0,    6,    0,    6,    6,    0,    6,    0,   0,
    7,     0,    7,    7,    0,
0,  6,     6,    0,    6,    0,    0,    7,    6,    0,    0,    7,    0,    6,   6,
    0,     0,    7,    7,    0,
6,  0,     0,    7,    0,    6,    6,    0,    6,    0,    0,    7,    6,    0,   0,
    7,     0,    8,    8,    0,
7,  0,     0,    6,    0,    6,    6,    0,    0,    6,    6,    0,    6,    0,   0,
    3,     0,    6,    6,    0,
6,  0,     0,    5,    6,    0,    0,    4,    0,    5,    4,    0,    0,    6,   6,
    0,     6,    0,    0,    4,
6,  0,     0,    5,    0,    4,    5,    0,    7,    0,    0,    5,    0,    6,   6,
    0,     0,    6,    6,    0,
5,  0,     0,    7,    0,    6,    6,    0,    6,    0,    0,    5,    6,    0,   0,
    4,     0,    5,    4,    0,
7,  0,     0,    6,    0,    6,    5,    0,    0,    5,    4,    0,    6,    0,   0,
    7,     7,    0,    0,    5,
0,  5,     6,    0,    0,    6,    5,    0,    5,    0,    0,    7,    0,    6,   5,
    0,     6,    0,    0,    7,
5,  0,     0,    7,    0,    7,    7,    0,    0,    6,    6,    0,    6,    0,   0,
    4,     6,    0,    0,    5,
0,  4,     5,    0,    7,    0,    0,    5,    0,    5,    6,    0,    0,    6,   5,
    0,     5,    0,    0,    7,
7,  0,     0,    6,    0,    4,    5,    0,    0,    5,    6,    0,    6,    0,   0,
    7,     0,    5,    6,    0,
5,  0,     0,    7,    6,    0,    0,    7,    0,    7,    7,    0,    7,    0,   0,
    5,     0,    6,    6,    0,
0,  6,     6,    0,    5,    0,    0,    7,    0,    6,    5,    0,    6,    0,   0,
    7,     5,    0,    0,    7,
0,  7,     7,    0,    0,    5,    6,    0,    5,    0,    0,    7,    6,    0,   0,
    7,     0,    7,    7,    0,
5,  0,     0,    8,    0,    7,    7,    0,    0,    7,    7,    0,    8,    0,   0,
    8};
char table2[] =
{1, 0,     0,    0,    0,    6,    0,    6,    0,    0,    0,    6,    0,    5,   0,
    7,     0,    0,    0,    6,
0,  7,     0,    0,    0,    0,    5,    0,    0,    7,    0,    5,    0,    6,   0,
    0,     0,    5,    0,    5,
0,  0,     0,    0,    4,    6,    0,    0,    0,    0,    5,    0,    5,    0,   0,
    7,     0,    6,    0,    0,
```

```
0,  0,  5,  6,  0,  0,  0,  0,  5,  0,  5,  0,  0,  0,  5,
    0,  4,  0,  5,  0,
0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  0,  6,  0,  6,  0,
    0,  0,  0,  6,  7,
0,  0,  0,  0,  6,  0,  7,  0,  0,  5,  0,  5,  0,  0,  0,
    0,  7,  6,  0,  0,
0,  0,  5,  0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,
    7,  0,  0,  0,  0,
6,  0,  6,  0,  0,  0,  5,  0,  5,  0,  5,  0,  0,  0,  5,
    0,  5,  0,  0,  0,
0,  7,  0,  6,  0,  7,  0,  0,  0,  0,  4,  7,  0,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  5,  0,  7,  0,  5,  0,  0,  6,  0,  0,  0,  0,  6,
    0,  6,  0,  0,  0,
5,  0,  6,  0,  5,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,
    7,  0,  0,  5,  0,
6,  0,  4,  0,  0,  0,  5,  0,  5,  0,  0,  0,  0,  6,  6,
    0,  0,  0,  0,  6,
0,  6,  0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  0,  5,  0,
    5,  0,  0,  0,  0,
7,  7,  0,  0,  0,  0,  6,  0,  6,  0,  0,  6,  0,  6,  0,
    0,  0,  0,  6,  6,
0,  0,  0,  0,  6,  0,  5,  0,  0,  0,  5,  0,  5,  0,  5,
    0,  0,  7,  0,  0,
0,  0,  7,  0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,
    0,  5,  0,  5,  0,
0,  0,  0,  7,  0,  4,  0,  4,  0,  0,  0,  0,  5,  6,  0,
    0,  0,  0,  6,  0,
6,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,  6,  0,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  8,  0,  6,  0,  8,  0,  0,  0,  6,  0,  7,  0,  0,
    0,  0,  7,  0,  0,
6,  0,  7,  0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,
    7,  5,  0,  0,  0,
0,  7,  0,  7,  0,  6,  0,  0,  0,  0,  6,  0,  6,  0,  0,
    0,  5,  0,  6,  0,
4,  0,  0,  0,  4,  0,  5,  0,  0,  0,  0,  6,  0,  0,  6,
    0,  7,  0,  5,  0,
0,  0,  6,  0,  6,  0,  0,  0,  0,  7,  7,  0,  0,  0,  0,
    7,  0,  7,  0,  0,
5,  0,  6,  0,  0,  0,  0,  7,  7,  0,  0,  0,  0,  7,  0,
    7,  0,  0,  0,  6,
0,  7,  0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  4,
    6,  0,  0,  0,  0,
6,  0,  6,  0,  0,  0,  4,  0,  6,  0,  5,  0,  0,  7,  0,
    0,  0,  0,  6,  0,
6,  0,  0,  0,  5,  0,  6,  0,  5,  0,  0,  0,  5,  0,  5,
    0,  0,  0,  0,  6,
0,  0,  5,  0,  7,  0,  5,  0,  0,  0,  6,  0,  6,  0,  0,
    0,  0,  7,  6,  0,
```

```
0,  0,    0,   6,   0,   6,   0,   6,   0,   0,   0,   0,   6,   0,   6,
    0,    0,   0,   5,   0,
7,  0,    6,   0,   0,   0,   6,   0,   5,   0,   0,   0,   0,   7,   0,
    0,    4,   0,   6,   0,
5,  0,    0,   0,   6,   0,   6,   0,   0,   0,   0,   7,   7,   0,   0,
    0,    0,   7,   0,   7,
0,  0,    5,   0,   4,   0,   0,   0,   0,   6,   7,   0,   0,   0,   0,
    7,    0,   7,   0,   0,
0,  6,    0,   7,   0,   6,   0,   0,   0,   0,   6,   0,   3,   0,   6,
    0,    0,   0,   4,   0,
5,  0,    0,   0,   0,   7,   6,   0,   0,   0,   0,   5,   0,   6,   0,
    0,    5,   0,   4,   0,
0,  0,    0,   7,   4,   0,   0,   0,   0,   6,   0,   6,   0,   0,   0,
    5,    0,   7,   0,   6,
0,  0,    6,   0,   0,   0,   0,   6,   0,   5,   0,   0,   0,   6,   0,
    7,    0,   5,   0,   0,
0,  7,    0,   7,   0,   0,   0,   0,   6,   0,   0,   7,   0,   6,   0,
    7,    0,   0,   0,   6,
0,  5,    0,   0,   0,   0,   5,   7,   0,   0,   0,   0,   6,   0,   5,
    0,    0,   6,   0,   5,
0,  0,    0,   0,   5,   6,   0,   0,   0,   0,   6,   0,   5,   0,   0,
    0,    6,   0,   5,   0,
6,  0,    0,   7,   0,   0,   0,   0,   6,   0,   5,   0,   0,   0,   7,
    0,    6,   0,   6,   0,
0,  0,    6,   0,   7,   0,   0,   0,   0,   7,   0,   5,   0,   4,   0,
    0,    0,   0,   6,   5,
0,  0,    0,   0,   6,   0,   7,   0,   0,   0,   6,   0,   7,   0,   6,
    0,    0,   4,   0,   0,
0,  0,    7,   0,   7,   0,   0,   0,   7,   0,   6,   0,   7,   0,   0,
    0,    6,   0,   7,   0,
0,  0,    0,   7,   0,   0,   6,   0,   7,   0,   4,   0,   0,   0,   7,
    0,    7,   0,   0,   0,
0,  6,    6,   0,   0,   0,   0,   7,   0,   6,   0,   6,   0,   0,   0,
    0,    6,   0,   4,   0,
0,  0,    6,   0,   6,   0,   3,   0,   0,   0,   7,   0,   6,   0,   0,
    0,    0,   7,   0,   0,
5,  0,    6,   0,   6,   0,   0,   0,   6,   0,   6,   0,   0,   0,   0,
    7,    5,   0,   0,   0,
0,  8,    0,   6,   0,   0,   6,   0,   6,   0,   0,   0,   0,   7,   5,
    0,    0,   0,   0,   6,
0,  7,    0,   0,   0,   6,   0,   7,   0,   6,   0,   0,   0,   5,   0,
    5,    0,   0,   0,   0,
6,  5,    0,   0,   0,   0,   7,   0,   7,   0,   0,   0,   6,   0,   7,
    0,    4,   0,   0,   5,
0,  0,    0,   0,   7,   0,   6,   0,   0,   0,   7,   0,   6,   0,   7,
    0,    0,   0,   6,   0,
7,  0,    0,   0,   0,   6,   0,   0,   6,   0,   7,   0,   6,   0,   0,
    0,    7,   0,   7,   0,
0,  0,    0,   7,   6,   0,   0,   0,   0,   7,   0,   6,   0,   5,   0,
    0,    0,   0,   7,   0,
```

```
6,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,  0,  7,  0,  7,
    0,  0,  0,  0,  6,
0,  0,  6,  0,  6,  0,  6,  0,  0,  0,  6,  0,  7,  0,  0,
    0,  0,  7,  6,  0,
0,  0,  0,  7,  0,  6,  0,  0,  7,  0,  7,  0,  0,  0,  0,
    7,  6,  0,  0,  0,
0,  7,  0,  6,  0,  0,  0,  7,  0,  6,  0,  6,  0,  0,  0,
    0,  7,  0,  6,  0,
6,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  5,  0,  0,
    0,  0,  7,  0,  7,
0,  0,  6,  0,  6,  0,  0,  0,  0,  7,  6,  0,  0,  0,  0,
    6,  0,  5,  0,  0,
0,  7,  0,  7,  0,  7,  0,  0,  6,  0,  0,  0,  0,  8,  0,
    6,  0,  0,  0,  7,
0,  7,  0,  7,  0,  0,  0,  7,  0,  7,  0,  0,  0,  0,  6,
    7,  0,  0,  0,  0,
5,  0,  5,  0,  0,  0,  5,  0,  6,  0,  6,  0,  0,  0,  6,
    0,  7,  0,  0,  0,
0,  7,  0,  0,  6,  0,  6,  0,  3,  0,  0,  0,  6,  0,  7,
    0,  0,  0,  0,  7,
4,  0,  0,  0,  0,  6,  0,  6,  0,  0,  7,  0,  6,  0,  0,
    0,  0,  7,  6,  0,
0,  0,  0,  8,  0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,
    0,  0,  0,  6,  0,
5,  0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  6,
    0,  0,  0,  0,  8,
0,  7,  0,  0,  6,  0,  7,  0,  0,  0,  0,  5,  5,  0,  0,
    0,  0,  6,  0,  5,
0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  5,  0,  0,  0,  0,
    6,  0,  7,  0,  0,
0,  7,  0,  6,  0,  7,  0,  0,  0,  6,  0,  7,  0,  0,  0,
    0,  6,  0,  7,  0,
6,  0,  0,  0,  0,  7,  5,  0,  0,  0,  0,  6,  0,  6,  0,
    0,  0,  7,  0,  7,
0,  7,  0,  0,  5,  0,  0,  0,  0,  6,  0,  5,  0,  0,  0,
    6,  0,  5,  0,  7,
0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  0,  0,  7,  0,
    7,  0,  7,  0,  0,
0,  7,  0,  7,  0,  0,  0,  0,  7,  8,  0,  0,  0,  0,  7,
    0,  6,  0,  0,  7,
0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  6,  0,  5,
    0,  0,  0,  5,  0,
5,  0,  5,  0,  0,  7,  0,  0,  0,  0,  6,  0,  5,  0,  0,
    0,  7,  0,  6,  0,
6,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  0,  0,  5,
    0,  5,  0,  4,  0,
0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  4,  0,  0,  0,  0,
    7,  0,  6,  0,  6,
0,  0,  0,  0,  5,  0,  5,  0,  0,  0,  7,  0,  6,  0,  6,
    0,  0,  0,  6,  0,
```

```
5,  0,  0,  0,  0,  6,  0,  0,  6,  0,  6,  0,  6,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  0,  7,  6,  0,  0,  0,  0,  8,  0,  6,  0,  0,  3,
    0,  6,  0,  0,  0,
0,  6,  4,  0,  0,  0,  0,  6,  0,  7,  0,  0,  0,  7,  0,
    7,  0,  7,  0,  0,
0,  0,  6,  0,  4,  0,  5,  0,  0,  0,  3,  0,  5,  0,  0,
    0,  0,  5,  6,  0,
0,  0,  0,  7,  0,  6,  0,  0,  5,  0,  4,  0,  0,  0,  0,
    5,  4,  0,  0,  0,
0,  7,  0,  6,  0,  0,  0,  7,  0,  7,  0,  6,  0,  0,  5,
    0,  0,  0,  0,  6,
0,  7,  0,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  5,  0,
    5,  0,  0,  0,  0,
6,  7,  0,  0,  0,  0,  6,  0,  4,  0,  0,  0,  6,  0,  7,
    0,  7,  0,  0,  0,
3,  0,  6,  0,  0,  0,  0,  6,  0,  0,  7,  0,  6,  0,  6,
    0,  0,  0,  6,  0,
6,  0,  0,  0,  0,  6,  5,  0,  0,  0,  0,  6,  0,  6,  0,
    0,  6,  0,  5,  0,
0,  0,  0,  6,  5,  0,  0,  0,  0,  8,  0,  6,  0,  0,  0,
    7,  0,  7,  0,  7,
0,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  0,  5,  0,  6,
    0,  0,  0,  0,  7,
5,  0,  0,  0,  0,  7,  0,  6,  0,  0,  6,  0,  5,  0,  0,
    0,  0,  7,  7,  0,
0,  0,  0,  7,  0,  6,  0,  0,  0,  6,  0,  6,  0,  7,  0,
    0,  5,  0,  0,  0,
0,  7,  0,  5,  0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  0,
    7,  0,  7,  0,  0,
0,  0,  5,  0,  6,  0,  6,  0,  0,  0,  0,  6,  5,  0,  0,
    0,  0,  6,  0,  7,
0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  6,  0,  0,  0,  0,
    7,  0,  5,  0,  0,
0,  7,  0,  6,  0,  6,  0,  0,  0,  6,  0,  7,  0,  0,  0,
    0,  7,  0,  0,  7,
0,  6,  0,  7,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  7,
    7,  0,  0,  0,  0,
6,  0,  5,  0,  0,  0,  6,  0,  5,  0,  5,  0,  0,  0,  5,
    0,  5,  0,  0,  0,
0,  6,  5,  0,  0,  0,  0,  6,  0,  6,  0,  0,  2,  0,  5,
    0,  0,  0,  0,  6,
5,  0,  0,  0,  0,  6,  0,  6,  0,  0,  0,  6,  0,  6,  0,
    6,  0,  0,  6,  0,
0,  0,  0,  6,  0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,
    0,  0,  6,  0,  6,
0,  0,  0,  0,  5,  0,  6,  0,  6,  0,  0,  0,  0,  6,  6,
    0,  0,  0,  0,  7,
0,  6,  0,  0,  0,  6,  0,  6,  0,  7,  0,  0,  5,  0,  0,
    0,  0,  5,  0,  7,
```

```
0,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  7,  0,  6,  0,
    0,  0,  0,  6,  0,
0,  6,  0,  5,  0,  7,  0,  0,  0,  6,  0,  7,  0,  0,  0,
    0,  7,  6,  0,  0,
0,  0,  6,  0,  5,  0,  5,  0,  0,  0,  0,  7,  0,  6,  0,
    0,  0,  6,  0,  6,
0,  7,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  0,  0,
    6,  0,  7,  0,  6,
0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  4,  6,  0,  0,  0,
    0,  6,  0,  5,  0,
0,  6,  0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  5,
    0,  4,  0,  0,  0,
6,  0,  6,  0,  7,  0,  0,  0,  0,  7,  0,  6,  0,  7,  0,
    0,  0,  5,  0,  6,
0,  0,  0,  0,  5,  7,  0,  0,  0,  0,  5,  0,  6,  0,  0,
    5,  0,  6,  0,  0,
0,  0,  5,  6,  0,  0,  0,  0,  5,  0,  6,  0,  0,  0,  6,
    0,  6,  0,  7,  0,
0,  7,  0,  0,  0,  0,  5,  0,  6,  0,  0,  0,  6,  0,  5,
    0,  6,  0,  0,  0,
7,  0,  6,  0,  0,  0,  0,  7,  0,  5,  0,  5,  0,  0,  0,
    0,  6,  5,  0,  0,
0,  0,  6,  0,  7,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,
    5,  0,  0,  0,  0,
7,  0,  7,  0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  0,  7,
    0,  7,  0,  0,  0,
0,  7,  0,  0,  4,  0,  7,  0,  6,  0,  0,  0,  7,  0,  6,
    0,  0,  0,  0,  6,
6,  0,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  0,  0,  5,
    0,  5,  0,  0,  0,
3,  0,  6,  0,  6,  0,  0,  0,  6,  0,  7,  0,  0,  0,  0,
    7,  0,  0,  6,  0,
6,  0,  5,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  6,
    0,  0,  0,  0,  6,
0,  8,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  4,  0,  0,
    0,  0,  6,  0,  6,
0,  0,  0,  6,  0,  6,  0,  6,  0,  0,  0,  4,  0,  5,  0,
    0,  0,  0,  6,  4,
0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  4,  0,  7,  0,  6,
    0,  0,  5,  0,  0,
0,  0,  7,  0,  6,  0,  0,  0,  7,  0,  6,  0,  7,  0,  0,
    0,  7,  0,  7,  0,
0,  0,  0,  6,  0,  0,  6,  0,  7,  0,  6,  0,  0,  0,  7,
    0,  6,  0,  0,  0,
0,  7,  6,  0,  0,  0,  0,  6,  0,  7,  0,  5,  0,  0,  0,
    0,  6,  0,  7,  0,
0,  0,  6,  0,  6,  0,  6,  0,  0,  0,  7,  0,  7,  0,  0,
    0,  0,  7,  0,  0,
6,  0,  6,  0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,
    7,  6,  0,  0,  0,
```

```
0,  6,  0,  7,  0,  0,  7,  0,  7,  0,  0,  0,  0,  6,  6,
    0,  0,  0,  0,  6,
0,  7,  0,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  0,  6,
    0,  5,  0,  6,  0,
0,  0,  7,  0,  6,  0,  0,  0,  0,  5,  5,  0,  0,  0,  0,
    7,  0,  6,  0,  0,
6,  0,  7,  0,  0,  0,  0,  7,  5,  0,  0,  0,  0,  5,  0,
    6,  0,  0,  0,  7,
0,  6,  0,  7,  0,  0,  6,  0,  0,  0,  0,  7,  0,  8,  0,
    0,  0,  7,  0,  6,
0,  7,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  6,  6,  0,
    0,  0,  0,  4,  0,
6,  0,  0,  0,  6,  0,  6,  0,  5,  0,  0,  0,  6,  0,  6,
    0,  0,  0,  0,  7,
0,  0,  3,  0,  6,  0,  6,  0,  0,  0,  6,  0,  6,  0,  0,
    0,  0,  7,  5,  0,
0,  0,  0,  7,  0,  6,  0,  0,  6,  0,  7,  0,  0,  0,  0,
    7,  5,  0,  0,  0,
0,  6,  0,  8,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,  0,
    0,  6,  0,  6,  0,
7,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  7,  6,  0,  0,
    0,  0,  6,  0,  8,
0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,
    5,  0,  6,  0,  0,
0,  7,  0,  7,  0,  7,  0,  0,  5,  0,  0,  0,  0,  7,  0,
    7,  0,  0,  0,  7,
0,  7,  0,  7,  0,  0,  0,  7,  0,  7,  0,  0,  0,  0,  6,
    0,  6,  0,  7,  0,
0,  0,  0,  7,  5,  0,  0,  0,  0,  5,  0,  6,  0,  0,  0,
    7,  0,  7,  0,  7,
0,  0,  5,  0,  0,  0,  0,  6,  0,  6,  0,  0,  0,  7,  0,
    5,  0,  6,  0,  0,
0,  7,  0,  7,  0,  0,  0,  0,  7,  0,  0,  7,  0,  7,  0,
    7,  0,  0,  0,  6,
0,  6,  0,  0,  0,  0,  6,  8,  0,  0,  0,  0,  6,  0,  7,
    0,  0,  6,  0,  6,
0,  0,  0,  0,  5,  4,  0,  0,  0,  0,  6,  0,  7,  0,  0,
    0,  6,  0,  8,  0,
6,  0,  0,  4,  0,  0,  0,  0,  7,  0,  6,  0,  0,  0,  7,
    0,  6,  0,  7,  0,
0,  0,  6,  0,  6,  0,  0,  0,  0,  7,  0,  0,  6,  0,  8,
    0,  6,  0,  0,  0,
6,  0,  6,  0,  0,  0,  0,  7,  5,  0,  0,  0,  0,  7,  0,
    7,  0,  5,  0,  0,
0,  0,  6,  0,  6,  0,  0,  0,  7,  0,  6,  0,  7,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  0,  7,  0,  0,  7,  0,  7,  0,  7,  0,  0,  0,  6,
    0,  6,  0,  0,  0,
0,  6,  7,  0,  0,  0,  0,  7,  0,  7,  0,  0,  6,  0,  6,
    0,  0,  0,  0,  6,
```

```
6,  0,  0,  0,  0,  6,  0,  6,  0,  0,  0,  6,  0,  7,  0,
    7,  0,  0,  0,  0,
5,  0,  7,  0,  5,  0,  0,  0,  5,  0,  6,  0,  0,  0,  0,
    7,  6,  0,  0,  0,
0,  6,  0,  7,  0,  0,  6,  0,  5,  0,  0,  0,  0,  7,  7,
    0,  0,  0,  0,  6,
0,  6,  0,  0,  0,  7,  0,  7,  0,  7,  0,  0,  7,  0,  0,
    0,  0,  7,  0,  7,
0,  0,  0,  6,  0,  6,  0,  5,  0,  0,  0,  7,  0,  7,  0,
    0,  0,  0,  5,  5,
0,  0,  0,  0,  6,  0,  6,  0,  0,  0,  7,  0,  7,  0,  7,
    0,  0,  0,  6,  0,
6,  0,  0,  0,  0,  6,  0,  0,  7,  0,  6,  0,  7,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  0,  6,  6,  0,  0,  0,  0,  6,  0,  6,  0,  0,  6,
    0,  6,  0,  0,  0,
0,  7,  7,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  7,  0,
    7,  0,  6,  0,  0,
0,  0,  7,  0,  6,  0,  7,  0,  0,  0,  6,  0,  5,  0,  0,
    0,  0,  5,  7,  0,
0,  0,  0,  6,  0,  6,  0,  0,  5,  0,  6,  0,  0,  0,  0,
    5,  6,  0,  0,  0,
0,  7,  0,  7,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,  7,
    0,  0,  0,  0,  6,
0,  6,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,  0,  5,  0,
    5,  0,  0,  0,  0,
5,  0,  6,  0,  6,  0,  0,  0,  0,  7,  6,  0,  0,  0,  0,
    5,  0,  5,  0,  0,
0,  6,  0,  6,  0,  6,  0,  0,  7,  0,  0,  0,  0,  6,  0,
    6,  0,  0,  0,  7,
0,  7,  0,  6,  0,  0,  0,  7,  0,  7,  0,  0,  0,  0,  6,
    0,  0,  7,  0,  7,
0,  7,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  7,  0,
    0,  0,  0,  7,  0,
6,  0,  0,  0,  5,  0,  7,  0,  5,  0,  0,  0,  5,  0,  6,
    0,  0,  0,  0,  7,
7,  0,  0,  0,  0,  7,  0,  7,  0,  0,  6,  0,  5,  0,  0,
    0,  0,  7,  7,  0,
0,  0,  0,  6,  0,  6,  0,  0,  0,  5,  0,  6,  0,  6,  0,
    0,  6,  0,  0,  0,
0,  7,  0,  6,  0,  0,  0,  7,  0,  7,  0,  7,  0,  0,  0,
    7,  0,  7,  0,  0,
0,  0,  5,  0,  6,  0,  6,  0,  0,  0,  0,  7,  7,  0,  0,
    0,  0,  6,  0,  6,
0,  0,  0,  7,  0,  7,  0,  7,  0,  0,  6,  0,  0,  0,  0,
    5,  0,  5,  0,  0,
0,  6,  0,  7,  0,  7,  0,  0,  0,  6,  0,  7,  0,  0,  0,
    0,  7,  0,  0,  6,
0,  6,  0,  6,  0,  0,  0,  7,  0,  7,  0,  0,  0,  0,  6,
    7,  0,  0,  0,  0,
```

```
6,  0,  7,  0,  4,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,
    7,  0,  6,  0,  7,
0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  6,  0,  0,  7,  0,
    6,  0,  7,  0,  0,
0,  4,  0,  4,  0,  0,  0,  0,  4,  7,  0,  0,  0,  0,  7,
    0,  6,  0,  0,  6,
0,  7,  0,  0,  0,  0,  6,  7,  0,  0,  0,  0,  6,  0,  7,
    0,  0,  0,  7,  0,
7,  0,  7,  0,  0,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,
    0,  4,  0,  6,  0,
5,  0,  0,  0,  5,  0,  6,  0,  0,  0,  0,  6,  0,  0,  5,
    0,  6,  0,  6,  0,
0,  0,  6,  0,  7,  0,  0,  0,  0,  7,  6,  0,  0,  0,  0,
    7,  0,  7,  0,  0,
4,  0,  5,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  7,  0,
    7,  0,  0,  0,  6,
0,  7,  0,  7,  0,  0,  0,  0,  5,  0,  6,  0,  6,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  0,  7,  7,  0,  0,  0,  0,  7,  0,  7,  0,  0,  6,
    0,  6,  0,  0,  0,
0,  7,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  7,  0,
    6,  0,  8,  0,  0,
6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  7,  0,  5,  0,
    6,  0,  0,  0,  6,
0,  7,  0,  0,  0,  0,  6,  0,  5,  0,  6,  0,  0,  0,  0,
    7,  6,  0,  0,  0,
0,  6,  0,  6,  0,  0,  0,  7,  0,  7,  0,  7,  0,  0,  6,
    0,  0,  0,  0,  7,
0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,  0,  6,  0,
    7,  0,  0,  0,  0,
6,  0,  0,  7,  0,  6,  0,  7,  0,  0,  0,  7,  0,  5,  0,
    0,  0,  0,  5,  6,
0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  5,  0,  5,  0,  6,
    0,  0,  0,  7,  0,
6,  0,  0,  0,  0,  7,  6,  0,  0,  0,  0,  7,  0,  7,  0,
    0,  6,  0,  7,  0,
0,  0,  0,  6,  5,  0,  0,  0,  0,  6,  0,  6,  0,  0,  0,
    7,  0,  5,  0,  7,
0,  0,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  6,  0,
    6,  0,  8,  0,  0,
0,  5,  0,  7,  0,  0,  0,  0,  6,  0,  6,  0,  7,  0,  0,
    0,  0,  7,  6,  0,
0,  0,  0,  7,  0,  7,  0,  0,  0,  6,  0,  7,  0,  7,  0,
    0,  6,  0,  0,  0,
0,  7,  0,  6,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  0,
    5,  0,  6,  0,  0,
0,  0,  7,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  5,  0,
    6,  0,  0,  0,  0,
6,  7,  0,  0,  0,  0,  6,  0,  7,  0,  5,  0,  0,  0,  0,
    7,  0,  7,  0,  0,
```

```
0,  7,  0,  4,  0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,
    0,  6,  0,  0,  6,
0,  6,  0,  8,  0,  0,  0,  4,  0,  6,  0,  0,  0,  0,  7,
    7,  0,  0,  0,  0,
6,  0,  6,  0,  0,  6,  0,  8,  0,  0,  0,  0,  7,  7,  0,
    0,  0,  0,  6,  0,
5,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  0,  5,  0,  6,
    0,  0,  0,  0,  6,
6,  0,  0,  0,  0,  7,  0,  6,  0,  0,  0,  7,  0,  7,  0,
    7,  0,  0,  6,  0,
0,  0,  0,  6,  0,  7,  0,  0,  0,  5,  0,  6,  0,  6,  0,
    0,  0,  7,  0,  6,
0,  0,  0,  0,  5,  0,  0,  6,  0,  6,  0,  6,  0,  0,  0,
    5,  0,  7,  0,  0,
0,  0,  6,  6,  0,  0,  0,  0,  7,  0,  6,  0,  5,  0,  0,
    0,  0,  7,  0,  7,
0,  0,  0,  6,  0,  5,  0,  8,  0,  0,  0,  6,  0,  8,  0,
    0,  0,  0,  7,  0,
0,  6,  0,  5,  0,  6,  0,  0,  0,  5,  0,  5,  0,  0,  0,
    0,  6,  7,  0,  0,
0,  0,  6,  0,  6,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,
    7,  0,  0,  0,  0,
5,  0,  6,  0,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  0,
    5,  0,  7,  0,  6,
0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,
    0,  6,  0,  7,  0,
0,  6,  0,  7,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  7,
    0,  7,  0,  0,  0,
5,  0,  7,  0,  6,  0,  0,  5,  0,  0,  0,  0,  7,  0,  6,
    0,  0,  0,  5,  0,
7,  0,  6,  0,  0,  0,  6,  0,  7,  0,  0,  0,  0,  7,  0,
    6,  0,  7,  0,  0,
0,  0,  6,  7,  0,  0,  0,  0,  5,  0,  6,  0,  0,  0,  4,
    0,  5,  0,  5,  0,
0,  6,  0,  0,  0,  0,  5,  0,  6,  0,  0,  0,  6,  0,  6,
    0,  7,  0,  0,  0,
6,  0,  6,  0,  0,  0,  0,  6,  0,  0,  5,  0,  5,  0,  5,
    0,  0,  0,  6,  0,
6,  0,  0,  0,  0,  6,  4,  0,  0,  0,  0,  6,  0,  7,  0,
    7,  0,  0,  0,  0,
4,  0,  6,  0,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  5,
    0,  6,  0,  0,  0,
0,  6,  0,  0,  7,  0,  7,  0,  6,  0,  0,  0,  6,  0,  6,
    0,  0,  0,  0,  6,
5,  0,  0,  0,  0,  6,  0,  8,  0,  0,  6,  0,  3,  0,  0,
    0,  0,  6,  5,  0,
0,  0,  0,  6,  0,  6,  0,  0,  0,  7,  0,  7,  0,  7,  0,
    0,  0,  0,  5,  0,
5,  0,  6,  0,  0,  0,  5,  0,  2,  0,  0,  0,  0,  6,  6,
    0,  0,  0,  0,  6,
```

```
0,  6,  0,  0,  5,  0,  5,  0,  0,  0,  0,  6,  5,  0,  0,
    0,  0,  6,  0,  6,
0,  0,  0,  6,  0,  6,  0,  6,  0,  0,  5,  0,  0,  0,  0,
    6,  0,  6,  0,  0,
0,  6,  0,  6,  0,  6,  0,  0,  0,  6,  0,  6,  0,  0,  0,
    0,  5,  6,  0,  0,
0,  0,  5,  0,  5,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,
    0,  6,  0,  3,  0,
0,  0,  0,  6,  0,  0,  6,  0,  6,  0,  7,  0,  0,  0,  6,
    0,  6,  0,  0,  0,
0,  7,  4,  0,  0,  0,  0,  7,  0,  6,  0,  0,  5,  0,  6,
    0,  0,  0,  0,  6,
6,  0,  0,  0,  0,  6,  0,  8,  0,  0,  0,  7,  0,  7,  0,
    7,  0,  0,  0,  0,
7,  0,  7,  0,  6,  0,  0,  0,  5,  0,  6,  0,  0,  0,  0,
    7,  5,  0,  0,  0,
0,  5,  0,  7,  0,  0,  6,  0,  5,  0,  0,  0,  0,  7,  7,
    0,  0,  0,  0,  6,
0,  7,  0,  0,  0,  7,  0,  6,  0,  7,  0,  0,  5,  0,  0,
    0,  0,  6,  0,  7,
0,  0,  0,  7,  0,  6,  0,  6,  0,  0,  0,  7,  0,  7,  0,
    0,  0,  0,  5,  0,
6,  0,  6,  0,  0,  0,  0,  6,  5,  0,  0,  0,  0,  7,  0,
    5,  0,  0,  0,  7,
0,  5,  0,  6,  0,  0,  6,  0,  0,  0,  0,  6,  0,  7,  0,
    0,  0,  7,  0,  6,
0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  0,  0,
    7,  0,  6,  0,  6,
0,  0,  0,  6,  0,  7,  0,  0,  0,  0,  6,  6,  0,  0,  0,
    0,  5,  0,  6,  0,
0,  0,  5,  0,  4,  0,  6,  0,  0,  0,  4,  0,  5,  0,  0,
    0,  0,  5,  5,  0,
0,  0,  0,  7,  0,  6,  0,  0,  5,  0,  3,  0,  0,  0,  0,
    5,  4,  0,  0,  0,
0,  6,  0,  7,  0,  0,  0,  7,  0,  6,  0,  6,  0,  0,  6,
    0,  0,  0,  0,  6,
0,  7,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  0,  5,  0,
    5,  0,  0,  0,  0,
6,  0,  6,  0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,
    5,  0,  7,  0,  0,
0,  7,  0,  6,  0,  7,  0,  0,  5,  0,  0,  0,  0,  7,  0,
    6,  0,  0,  0,  6,
0,  6,  0,  7,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  7,
    0,  0,  7,  0,  6,
0,  7,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  7,  0,
    0,  0,  0,  5,  0,
6,  0,  5,  0,  0,  0,  0,  6,  0,  7,  0,  0,  0,  6,  0,
    7,  0,  6,  0,  0,
0,  6,  0,  6,  0,  0,  0,  0,  6,  0,  0,  7,  0,  6,  0,
    6,  0,  0,  0,  6,
```

```
0,  7,  0,  0,  0,  0,  4,  6,  0,  0,  0,  0,  4,  0,  5,
    0,  0,  6,  0,  6,
0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  5,  0,  6,  0,  0,
    0,  7,  0,  6,  0,
6,  0,  0,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  6,
    0,  6,  0,  5,  0,
0,  0,  5,  0,  4,  0,  0,  0,  0,  6,  0,  0,  5,  0,  6,
    0,  4,  0,  0,  0,
7,  0,  6,  0,  0,  0,  0,  7,  6,  0,  0,  0,  0,  7,  0,
    7,  0,  0,  6,  0,
5,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  7,  0,  7,  0,
    0,  0,  7,  0,  7,
0,  6,  0,  0,  0,  0,  6,  0,  5,  0,  5,  0,  0,  0,  7,
    0,  6,  0,  0,  0,
0,  6,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  6,  0,  7,
    0,  0,  0,  0,  7,
5,  0,  0,  0,  0,  6,  0,  6,  0,  0,  0,  8,  0,  6,  0,
    6,  0,  0,  6,  0,
0,  0,  0,  7,  0,  7,  0,  0,  0,  7,  0,  5,  0,  7,  0,
    0,  0,  7,  0,  5,
0,  0,  0,  0,  6,  0,  6,  0,  5,  0,  0,  0,  0,  6,  6,
    0,  0,  0,  0,  7,
0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,  6,  0,  0,
    0,  0,  6,  0,  7,
0,  0,  0,  6,  0,  6,  0,  5,  0,  0,  0,  7,  0,  5,  0,
    0,  0,  0,  6,  0,
0,  7,  0,  7,  0,  7,  0,  0,  0,  6,  0,  7,  0,  0,  0,
    0,  5,  6,  0,  0,
0,  0,  6,  0,  7,  0,  0,  0,  6,  0,  6,  0,  5,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  0,  7,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  6,
    0,  6,  0,  0,  0,
0,  7,  6,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  6,  0,
    5,  0,  7,  0,  0,
7,  0,  0,  0,  0,  7,  0,  7,  0,  0,  0,  8,  0,  6,  0,
    7,  0,  0,  0,  7,
0,  6,  0,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,  0,  0,
    7,  6,  0,  0,  0,
0,  6,  0,  7,  0,  0,  0,  7,  0,  6,  0,  6,  0,  0,  6,
    0,  0,  0,  0,  7,
0,  7,  0,  0,  0,  7,  0,  7,  0,  6,  0,  0,  0,  6,  0,
    5,  0,  0,  0,  0,
6,  0,  0,  7,  0,  7,  0,  6,  0,  0,  0,  6,  0,  5,  0,
    0,  0,  0,  7,  7,
0,  0,  0,  0,  7,  0,  6,  0,  5,  0,  0,  0,  0,  7,  0,
    7,  0,  0,  0,  6,
0,  5,  0,  6,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,
    0,  0,  8,  0,  5,
0,  6,  0,  0,  0,  5,  0,  5,  0,  0,  0,  0,  6,  7,  0,
    0,  0,  0,  6,  0,
```

```
5,  0,  0,  8,  0,  6,  0,  0,  0,  0,  7,  7,  0,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  7,  0,  6,  0,  6,  0,  0,  0,  6,  0,  5,  0,  0,
    0,  0,  7,  6,  0,
0,  0,  0,  6,  0,  7,  0,  0,  0,  7,  0,  6,  0,  7,  0,
    0,  6,  0,  0,  0,
0,  6,  0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,  0,  0,
    5,  0,  7,  0,  0,
0,  0,  5,  0,  0,  7,  0,  7,  0,  7,  0,  0,  0,  7,  0,
    6,  0,  0,  0,  0,
6,  6,  0,  0,  0,  0,  7,  0,  7,  0,  5,  0,  0,  0,  0,
    7,  0,  7,  0,  0,
0,  8,  0,  6,  0,  6,  0,  0,  0,  8,  0,  6,  0,  0,  0,
    0,  7,  0,  0,  6,
0,  4,  0,  7,  0,  0,  0,  6,  0,  4,  0,  0,  0,  0,  7,
    7,  0,  0,  0,  0,
5,  0,  6,  0,  0,  6,  0,  7,  0,  0,  0,  0,  6,  7,  0,
    0,  0,  0,  6,  0,
6,  0,  0,  0,  7,  0,  7,  0,  6,  0,  0,  0,  0,  6,  0,
    7,  0,  5,  0,  0,
0,  7,  0,  6,  0,  0,  0,  0,  6,  5,  0,  0,  0,  0,  6,
    0,  7,  0,  0,  6,
0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  7,  0,  7,
    0,  0,  0,  6,  0,
7,  0,  5,  0,  0,  6,  0,  0,  0,  0,  7,  0,  6,  0,  0,
    0,  6,  0,  7,  0,
5,  0,  0,  0,  7,  0,  6,  0,  0,  0,  0,  7,  0,  0,  6,
    0,  4,  0,  6,  0,
0,  0,  4,  0,  5,  0,  0,  0,  0,  7,  6,  0,  0,  0,  0,
    5,  0,  6,  0,  0,
5,  0,  4,  0,  0,  0,  0,  7,  3,  0,  0,  0,  0,  7,  0,
    7,  0,  0,  0,  5,
0,  6,  0,  6,  0,  0,  6,  0,  0,  0,  0,  6,  0,  5,  0,
    0,  0,  6,  0,  6,
0,  5,  0,  0,  0,  7,  0,  7,  0,  0,  0,  0,  6,  0,  5,
    0,  5,  0,  0,  0,
0,  8,  7,  0,  0,  0,  0,  6,  0,  7,  0,  0,  0,  7,  0,
    6,  0,  7,  0,  0,
6,  0,  0,  0,  0,  6,  0,  7,  0,  0,  0,  7,  0,  5,  0,
    7,  0,  0,  0,  6,
0,  6,  0,  0,  0,  0,  7,  0,  0,  7,  0,  7,  0,  7,  0,
    0,  0,  6,  0,  5,
0,  0,  0,  0,  6,  7,  0,  0,  0,  0,  6,  0,  7,  0,  5,
    0,  0,  0,  0,  6,
0,  6,  0,  0,  0,  6,  0,  7,  0,  6,  0,  0,  0,  6,  0,
    6,  0,  0,  0,  0,
6,  0,  0,  6,  0,  6,  0,  6,  0,  0,  0,  6,  0,  7,  0,
    0,  0,  0,  6,  7,
0,  0,  0,  0,  6,  0,  5,  0,  0,  7,  0,  6,  0,  0,  0,
    0,  7,  6,  0,  0,
```

```
0,  0,  5,  0,  4,  0,  0,  0,  6,  0,  4,  0,  6,  0,  0,
    0,  5,  0,  5,  0,
0,  0,  0,  8,  6,  0,  0,  0,  0,  7,  0,  6,  0,  0,  0,
    7,  0,  7,  0,  7,
0,  0,  7,  0,  0,  0,  0,  7,  0,  6,  0,  0,  0,  7,  0,
    5,  0,  7,  0,  0,
0,  5,  0,  6,  0,  0,  0,  0,  6,  0,  0,  7,  0,  6,  0,
    7,  0,  0,  0,  6,
0,  6,  0,  0,  0,  0,  7,  7,  0,  0,  0,  0,  7,  0,  6,
    0,  4,  0,  0,  0,
0,  7,  0,  7,  0,  0,  0,  7,  0,  4,  0,  6,  0,  0,  0,
    7,  0,  7,  0,  0,
0,  0,  7,  0,  0,  6,  0,  4,  0,  7,  0,  0,  0,  6,  0,
    6,  0,  0,  0,  0,
7,  7,  0,  0,  0,  0,  7,  0,  6,  0,  0,  7,  0,  7,  0,
    0,  0,  0,  7,  7,
0,  0,  0,  0,  6,  0,  7,  0,  0,  0,  6,  0,  4,  0,  6,
    0,  0,  0,  0,  6,
0,  6,  0,  5,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,
    5,  0,  0,  0,  0,
5,  0,  5,  0,  0,  6,  0,  7,  0,  0,  0,  0,  7,  7,  0,
    0,  0,  0,  7,  0,
7,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  6,  0,  0,  0,
    0,  6,  0,  6,  0,
0,  0,  7,  0,  7,  0,  6,  0,  0,  0,  6,  0,  6,  0,  0,
    0,  0,  7,  5,  0,
0,  0,  0,  6,  0,  6,  0,  0,  0,  6,  0,  6,  0,  6,  0,
    0,  0,  6,  0,  7,
0,  0,  0,  0,  7,  0,  0,  6,  0,  7,  0,  6,  0,  0,  0,
    7,  0,  6,  0,  0,
0,  0,  6,  6,  0,  0,  0,  0,  4,  0,  5,  0,  0,  6,  0,
    6,  0,  0,  0,  0,
6,  7,  0,  0,  0,  0,  5,  0,  6,  0,  0,  0,  6,  0,  4,
    0,  6,  0,  0,  0,
0,  5,  0,  6,  0,  6,  0,  0,  0,  7,  0,  6,  0,  0,  0,
    0,  7,  6,  0,  0,
0,  0,  6,  0,  6,  0,  0,  6,  0,  6,  0,  0,  0,  0,  6,
    7,  0,  0,  0,  0,
7,  0,  7,  0,  0,  0,  6,  0,  7,  0,  7,  0,  0,  5,  0,
    0,  0,  0,  5,  0,
5,  0,  0,  0,  7,  0,  7,  0,  6,  0,  0,  0,  6,  0,  6,
    0,  0,  0,  0,  7,
0,  6,  0,  6,  0,  0,  0,  0,  6,  6,  0,  0,  0,  0,  6,
    0,  6,  0,  0,  0,
6,  0,  5,  0,  6,  0,  0,  6,  0,  0,  0,  0,  6,  0,  6,
    0,  0,  0,  5,  0,
3,  0,  5,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  7,  0,
    0,  6,  0,  5,  0,
6,  0,  0,  0,  6,  0,  6,  0,  0,  0,  0,  7,  6,  0,  0,
    0,  0,  7,  0,  7,
```

```
0};
char table3[] =
{0, 0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    5,     0,    0,    0,    0,
0,  0,     0,    6,    0,    0,    0,    5,    0,    6,    5,    0,    0,    0,    0,
    0,     0,    0,    0,    5,
0,  0,     0,    6,    0,    5,    6,    0,    0,    0,    0,    5,    0,    6,    5,
    0,     0,    5,    6,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    6,    0,    0,    0,
    5,     0,    6,    5,    0,
0,  0,     0,    6,    0,    6,    5,    0,    0,    5,    4,    0,    6,    0,    0,
    0,     0,    0,    0,    6,
0,  5,     6,    0,    0,    6,    5,    0,    6,    0,    0,    0,    0,    5,    4,
    0,     5,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    5,     0,    0,    0,    6,
0,  5,     6,    0,    0,    0,    0,    6,    0,    5,    6,    0,    0,    6,    5,
    0,     6,    0,    0,    0,
0,  0,     0,    6,    0,    4,    5,    0,    0,    5,    6,    0,    6,    0,    0,
    0,     0,    4,    5,    0,
4,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    5,     0,    6,    5,    0,
0,  5,     6,    0,    5,    0,    0,    0,    0,    5,    4,    0,    5,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    0,    4,    5,    0,    4,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
3,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    6,    0,    0,    0,    6,    0,    6,    6,    0,    0,    0,    0,
    6,     0,    6,    6,    0,
0,  6,     5,    0,    6,    0,    0,    0,    0,    0,    0,    6,    0,    5,    6,
    0,     0,    6,    6,    0,
6,  0,     0,    0,    0,    5,    5,    0,    5,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    6,    0,    6,    6,    0,    0,    6,    5,    0,    6,    0,    0,
    0,     0,    5,    5,    0,
6,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    5,    5,
    0,     5,    0,    0,    0,
6,  0,     0,    0,    0,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    6,    0,    5,    6,    0,    0,    6,    6,
    0,     6,    0,    0,    0,
0,  5,     5,    0,    6,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    5,    5,    0,
5,  0,     0,    0,    6,    0,    0,    0,    0,    0,    0,    0,    4,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    5,    5,    0,    5,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
```

```
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    6,
0,  0,     0,    6,    0,    6,    6,    0,    0,    0,    0,    6,    0,    6,    6,
    0,     0,    6,    5,    0,
6,  0,     0,    0,    0,    0,    0,    6,    0,    5,    6,    0,    0,    6,    6,
    0,     6,    0,    0,    0,
0,  5,     5,    0,    5,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    6,
0,  6,     6,    0,    0,    6,    5,    0,    6,    0,    0,    0,    0,    5,    5,
    0,     6,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    5,    5,    0,    6,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    6,    0,    5,    6,    0,    0,    6,    6,    0,    6,    0,    0,
    0,     0,    5,    5,    0,
5,  0,     0,    0,    6,    0,    0,    0,    0,    0,    0,    0,    0,    5,    5,
    0,     5,    0,    0,    0,
6,  0,     0,    0,    0,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    5,    5,    0,    5,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    6,    0,    6,    6,
    0,     0,    6,    6,    0,
6,  0,     0,    0,    0,    5,    5,    0,    5,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
0,  5,     5,    0,    5,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    5,    5,
    0,     5,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    5,    5,    0,
5,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
```

```
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    6,    0,    0,    0,
    6,     0,    6,    6,    0,
0,  0,     0,    6,    0,    6,    6,    0,    0,    6,    5,    0,    6,    0,    0,
    0,     0,    0,    0,    6,
0,  5,     6,    0,    0,    6,    6,    0,    6,    0,    0,    0,    0,    5,    5,
    0,     5,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    6,    0,    6,    6,
    0,     0,    6,    5,    0,
6,  0,     0,    0,    0,    6,    5,    0,    5,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
0,  5,     6,    0,    5,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    6,     0,    5,    6,    0,
0,  6,     6,    0,    6,    0,    0,    0,    0,    6,    5,    0,    5,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    0,    5,    6,    0,    5,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    5,    5,    0,
5,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    4,    0,    5,    4,    0,    0,    4,    5,    0,    4,    0,    0,
    0,     0,    5,    4,    0,
5,  0,     0,    0,    4,    0,    0,    0,    0,    0,    0,    0,    0,    4,    5,
    0,     4,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    4,    3,    0,    4,    0,    0,    0,    3,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     3,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    3,    4,    0,    3,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    3,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    2,    0,    0,
    0,     0,    0,    0,    0,
```

```
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    5,
0,  5,     5,    0,    0,    5,    5,    0,    5,    0,    0,    0,    0,    5,    4,
    0,     4,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    4,    5,    0,    4,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    3,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  4,     4,    0,    5,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    4,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    4,    4,    0,    4,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    3,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    4,    4,
    0,     4,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    4,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
```

```
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    6,     0,    0,    0,    6,
0,  6,     6,    0,    0,    0,    0,    6,    0,    6,    6,    0,    0,    6,    5,    5,
    0,     6,    0,    0,    0,
0,  0,     0,    6,    0,    5,    6,    0,    0,    6,    6,    0,    6,    0,    0,    0,
    0,     0,    5,    5,    0,
5,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    6,     0,    6,    6,    0,
0,  6,     5,    0,    6,    0,    0,    0,    0,    6,    5,    0,    5,    0,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    0,    6,    5,    0,    5,    0,    0,    0,    5,    0,    0,    0,
    0,     0,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    6,
0,  5,     6,    0,    0,    6,    6,    0,    6,    0,    0,    0,    0,    5,    6,    6,
    0,     5,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    5,    6,    0,    5,    0,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  5,     5,    0,    5,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    4,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    5,    0,    5,    5,    0,    0,    5,    5,    5,
    0,     5,    0,    0,    0,
0,  4,     4,    0,    5,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    4,    4,    0,
4,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    3,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    5,    4,    0,    4,    0,    0,    0,
    0,     4,    0,    0,    0,
0,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    4,    5,    5,
    0,     4,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    4,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    5,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
```

```
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     3,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    4,    0,    5,    4,    0,    0,    4,    5,    0,    4,    0,    0,
    0,     0,    4,    3,    0,
4,  0,     0,    0,    3,    0,    0,    0,    0,    0,    0,    0,    0,    3,    4,
    0,     3,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    2,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    5,    4,    0,    5,    0,    0,    0,    4,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     3,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    4,    5,    0,    4,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    3,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    4,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    4,    4,    0,
4,  0,     0,    0,    4,    0,    0,    0,    0,    0,    0,    0,    5,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    5,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
4,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    4,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
```

```
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    6,  0,  6,  6,  0,
0,  6,  6,  0,  6,  0,  0,  0,  0,  5,  5,  0,  5,  0,  0,
    0,  5,  0,  0,  0,
0,  0,  0,  0,  0,  5,  5,  0,  5,  0,  0,  0,  5,  0,  0,
    0,  0,  0,  0,  0,
4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  5,  5,  0,
5,  0,  0,  0,  5,  0,  0,  0,  0,  0,  0,  0,  5,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  5,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  5,  5,  0,  5,  0,  0,  0,  5,  0,  0,  0,  0,  0,  0,
    0,  5,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  5,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  4,  4,  0,  4,  0,  0,
    0,  4,  0,  0,  0,
0,  0,  0,  0,  4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
4,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  5,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  5,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,  0,  0,  0,  0,
```

```
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  4,  4,  0,  4,  0,  0,  0,  4,  0,  0,
    0,   0,  0,  0,  0,
5,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   5,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  4,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  4,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
1,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
0,  0,   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
    0,   0,  0,  0,  0,
```

```
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0};
char table4[] =
{0, 0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    3,     0,    0,    0,    0,
0,  0,     0,    4,    0,    0,    0,    5,    0,    4,    5,    0,    0,    0,    0,
    0,     0,    0,    0,    5,
0,  0,     0,    4,    0,    5,    4,    0,    0,    0,    0,    5,    0,    4,    5,
    0,     0,    5,    4,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    4,    0,    0,    0,
    5,     0,    4,    5,    0,
0,  0,     0,    4,    0,    4,    3,    0,    0,    3,    2,    0,    4,    0,    0,
    0,     0,    0,    0,    5,
0,  3,     4,    0,    0,    4,    3,    0,    5,    0,    0,    0,    0,    4,    5,
    0,     4,    0,    0,    0,
5,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    5,     0,    0,    0,    4,
0,  5,     4,    0,    0,    0,    0,    5,    0,    3,    4,    0,    0,    4,    3,
    0,     5,    0,    0,    0,
0,  0,     0,    4,    0,    2,    3,    0,    0,    3,    4,    0,    4,    0,    0,
    0,     0,    5,    4,    0,
5,  0,     0,    0,    4,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    5,     0,    4,    5,    0,
0,  5,     4,    0,    5,    0,    0,    0,    0,    4,    5,    0,    4,    0,    0,
    0,     5,    0,    0,    0,
0,  0,     0,    0,    0,    5,    4,    0,    5,    0,    0,    0,    4,    0,    0,
    0,     0,    0,    0,    0,
1,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0};
char table5[] =
{0, 0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    3,    10,   11,   10,
    10,    11,   0,    0,    0,    0,
```

```
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    10,   9,
8,  7,     7,    8,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     9,    10,   7,    8,    8,    7,    0,    0,    0,    0,    0,    0,    6,    6,
    10,    10,   6,    12,   10,
9,  6,     6,    9,    9,    11,   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    9,    9,
    10,    7,    8,    8,    7,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    10,   9,
8,  7,     7,    8,    0,    0,    0,    0,    0,    0,    9,    6,    6,    9,    9,    9,
    11,    6,    10,   10,   6,
12, 10,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    9,    2,    10,   12,   12,   9,    0,    0,    0,    0,
    0,     0,    0,    9,    11,
5,  10,    10,   6,    11,   9,    10,   6,    5,    10,   0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     11,   9,    10,   5,    6,    10,   9,    11,   6,    10,   10,   5,    0,    0,
    0,     0,    0,    0,    0,
6,  5,     9,    10,   10,   9,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    6,    10,   10,   12,   6,    10,   0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    9,    6,    6,    9,    9,    9,
    11,    0,    0,    0,    0,
0,  0,     10,   8,    9,    7,    7,    8,    9,    7,    10,   8,    8,    7,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    9,    11,
5,  10,    10,   6,    0,    0,    0,    0,    0,    0,    7,    4,    8,    11,
    11,    9,    8,    10,   7,    11,
11, 4,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    8,    8,
    10,    7,    11,   11,   4,
8,  3,     8,    12,   12,   10,   0,    0,    0,    0,    0,    0,    9,    11,   6,    6,
    10,    10,   5,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
    11,    9,    10,   6,    5,    10,
```

```
0,  0,     0,    0,    0,    0,    8,    10,  7,    11,  11,  4,    7,    4,   8,
    11,    11,   9,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   8,   3,    8,    12,
    12,    10,   8,    10,   7,    11,
11, 4,     0,    0,    0,    0,    0,    0,   11,   9,   10,  6,    5,    10,  0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    10,   8,
9,  7,     7,    8,    9,    7,    10,   8,   8,    7,   0,   0,    0,    0,   0,
    0,     6,    10,   10,   6,
12, 10,    0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
9,  6,     6,    9,    9,    11,   0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   9,   6,    6,    9,   9,
    11,    0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   6,
    10,    10,   12,   6,    10,
0,  0,     0,    0,    0,    0,    9,    7,   10,   8,   8,   7,    10,   8,   9,
    7,     7,    8,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     11,   9,    10,   5,    6,    10,  0,    0,   0,   0,    0,    0,   8,
    10,    7,    11,   11,   4,
8,  3,     8,    12,   12,   10,   0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    7,    4,
8,  11,    11,   9,    8,    10,   7,    11,  11,   4,   0,   0,    0,    0,   0,
    0,     11,   9,    10,   5,
6,  10,    0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    9,    11,
6,  10,    10,   5,    0,    0,    0,    0,   0,    0,   8,   3,    8,    12,
    12,    10,   8,    10,   7,    11,
11, 4,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   8,
    10,    7,    11,   11,   4,
7,  4,     8,    11,   11,   9,    0,    0,   0,    0,   0,   0,    9,    11,  5,
    10,    10,   6,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     9,    7,    10,   8,    8,    7,   10,   8,   9,   7,    7,    8,   0,
    0,     0,    0,    0,    0,
9,  6,     6,    9,    9,    11,   0,    0,   0,    0,   0,   0,    0,    0,   0,
    0,     0,    0,    0,    0,
```

```
0,  0,     0,    0,    6,    10,   10,   6,    12,   10,   0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    6,    5,    9,    10,
    10,    9,    0,    0,    0,    0,
0,  0,     9,    11,   6,    10,   10,   5,    11,   9,    10,   6,    5,    10,  0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    11,   9,    10,   5,    6,    10,   9,    11,  5,
    10,    10,   6,    0,    0,
0,  0,     0,    0,    9,    2,    10,   12,   12,   9,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   6,
    10,    10,   12,   6,    10,
9,  6,     6,    9,    9,    11,   0,    0,    0,    0,    0,    0,    10,   9,   8,
    7,     7,    8,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     9,    10,   7,    8,
8,  7,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    9,    6,    6,    9,   9,
    11,    6,    10,   10,   12,
6,  10,    0,    0,    0,    0,    0,    0,    9,    10,   7,    8,    8,    7,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    10,   9,   8,
    7,     7,    8,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
1,  10,    11,   10,   10,   11,   0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0,     0,    0,    0,    0,
0,  0,     0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,   0,
    0};
char table6[] =
{1, 10,    10,   5,    5,    10,   10,   7,    5,    10,   10,   5,    5,    10,
    10,    5,    7,    10,   10,   5,
5,  10,    10,   7,    9,    4,    2,    9,    9,    6,    4,    11,   9,    6,   6,
    9,     9,    6,    6,    9,
```

```
11, 6,    6,   9,   9,   6,   6,   11,  9,   6,   8,   9,   9,   4,   6,
    11,   9,   8,   8,   9,
9,  8,    8,   9,   11,  8,   8,   9,   9,   8,   8,   11,  9,   8,   8,
    5,    7,   8,   8,   9,
7,  10,   10,  9,   9,   10,  10,  9,   9,   10,  10,  7,   9,   10,
    10,   9,   9,   8,   8,   7,
5,  8,    8,   9,   9,   10,  10,  7,   7,   10,  10,  9,   9,   10,
    10,   9,   9,   10,  10,  9,
7,  6,    8,   9,   9,   8,   10,  11,  9,   8,   10,  9,   9,   10,  8,
    9,    11,  10,  8,   9,
9,  8,    10,  11,  9,   4,   6,   9,   9,   6,   4,   11,  9,   6,   6,
    9,    9,   6,   6,   9,
11, 6,    6,   9,   9,   2,   6,   11,  7,   10,  10,  5,   5,   10,
    10,   5,   5,   10,  10,  5,
5,  10,   10,  5,   7,   10,  10,  5,   5,   10,  10,  3,   9,   8,
    10,   7,   9,   8,   8,   9,
9,  10,   10,  7,   9,   10,  10,  9,   9,   10,  10,  5,   7,   8,
    10,   9,   11,  10,  10,  9,
9,  8,    10,  9,   9,   8,   8,   9,   9,   8,   8,   9,   9,   10,  8,
    9,    9,   10,  6,   9,
11, 8,    8,   9,   9,   8,   8,   11,  9,   8,   8,   9,   9,   8,   8,
    9,    11,  8,   6,   9,
9,  8,    8,   9,   9,   8,   10,  9,   9,   10,  8,   9,   7,   10,
    10,   9,   9,   10,  8,   7,
9,  10,   10,  7,   5,   8,   10,  9,   9,   8,   10,  9,   7,   8,   8,
    9,    9,   10,  10,  5,
9,  10,   10,  7,   9,   10,  10,  7,   9,   8,   10,  9,   11,  8,   8,
    9,    9,   8,   8,   11,
9,  6,    8,   9,   9,   8,   8,   9,   11,  8,   8,   9,   9,   8,   8,
    9,    7,   10,  8,   9,
9,  8,    6,   11,  9,   8,   10,  9,   9,   10,  8,   9,   11,  10,  8,
    9,    9,   8,   10,  11,
9,  8,    8,   7,   9,   10,  8,   9,   5,   10,  10,  9,   7,   10,  8,
    9,    9,   10,  10,  9,
7,  10,   10,  9,   7,   10,  10,  9,   9,   10,  10,  7,   9,   10,
    10,   9,   9,   10,  10,  9,
9,  10,   10,  9,   9,   10,  10,  9,   11,  8,   8,   9,   9,   8,   8,
    11,   9,   8,   6,   9,
9,  8,    8,   9,   9,   8,   8,   9,   9,   8,   8,   11,  11,  10,
    10,   9,   9,   8,   10,  9,
9,  8,    8,   9,   9,   8,   8,   9,   9,   6,   8,   9,   9,   10,
    10,   9,   9,   8,   10,  9,
9,  10,   8,   9,   9,   10,  10,  7,   7,   10,  8,   5,   9,   10,
    10,   9,   7,   8,   10,  9,
9,  8,    8,   9,   7,   10,  8,   9,   7,   10,  10,  9,   5,   10,  8,
    7,    9,   10,  10,  9,
9,  10,   10,  9,   9,   6,   8,   9,   9,   8,   6,   11,  9,   8,   8,
    9,    9,   8,   4,   9,
11, 8,    8,   9,   9,   8,   8,   11,  11,  8,   8,   9,   9,   8,   8,
    11,   9,   8,   8,   9,
```

```
9,  6,     8,    9,    9,    8,    8,    9,    9,    8,    8,    11,   9,    10,
    10,    7,    7,    10,   10,   7,
7,  10,    10,   7,    7,    10,   10,   7,    5,    10,   10,   7,    7,    10,
    10,    9,    9,    4,    6,    9,
9,  2,     4,    11,   9,    6,    6,    9,    9,    6,    6,    9,    11,   6,   6,
    9,     9,    6,    6,    11,
7,  8,     8,    3,    3,    8,    10,   7,    5,    10,   10,   5,    5,    10,
    10,    7,    7,    8,    10,   5,
7,  10,    8,    7,    5,    8,    8,    7,    7,    10,   8,    9,    9,    10,
    10,    7,    9,    10,   12,   9,
9,  10,    10,   7,    9,    8,    10,   9,    7,    6,    6,    9,    7,    8,   8,
    9,     9,    8,    8,    9,
11, 8,     10,   9,    9,    8,    10,   9,    11,   8,    8,    9,    7,    6,   6,
    7,     9,    8,    8,    9,
11, 10,    8,    9,    9,    8,    10,   11,   9,    8,    8,    9,    9,    8,   8,
    9,     7,    10,   8,    7,
7,  6,     10,   7,    9,    10,   8,    9,    9,    8,    10,   9,    9,    10,
    10,    9,    9,    10,   10,   9,
7,  10,    10,   5,    5,    8,    8,    7,    7,    10,   10,   3,    7,    10,
    10,    5,    7,    8,    10,   3,
5,  8,     8,    7,    11,   6,    6,    9,    9,    6,    6,    11,   9,    4,   6,
    9,     9,    6,    8,    9,
11, 4,     4,    9,    9,    6,    4,    9,    7,    8,    8,    9,    9,    8,   6,
    9,     11,   10,   8,    7,
9,  8,     10,   11,   9,    8,    8,    9,    9,    6,    8,    9,    9,    10,  8,
    9,     9,    10,   10,   9,
7,  8,     8,    9,    9,    10,   8,    9,    7,    10,   8,    9,    7,    8,
    10,    7,    7,    10,   8,    7,
7,  10,    10,   7,    9,    10,   8,    7,    7,    8,    10,   9,    7,    8,
    10,    7,    7,    8,    8,    7,
9,  8,     8,    9,    9,    8,    8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     7,    8,    8,    7,
9,  8,     6,    9,    7,    8,    8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     11,   8,    10,   9,
9,  8,     10,   7,    11,   6,    8,    9,    7,    10,   8,    7,    7,    10,
    10,    7,    7,    10,   8,    7,
9,  8,     10,   7,    7,    8,    10,   7,    9,    8,    8,    7,    7,    10,
    10,    9,    9,    6,    10,   7,
9,  10,    8,    7,    9,    8,    10,   9,    9,    10,   10,   7,    9,    8,
    10,    9,    9,    8,    8,    7,
9,  8,     8,    9,    9,    8,    6,    9,    9,    8,    8,    9,    7,    8,   8,
    9,     9,    8,    6,    9,
11, 8,     8,    9,    9,    6,    8,    11,   9,    8,    8,    9,    9,    8,   8,
    9,     9,    8,    8,    9,
9,  8,     8,    11,   9,    8,    10,   7,    9,    10,   8,    9,    7,    10,
    10,    9,    9,    8,    8,    9,
9,  10,    10,   7,    7,    10,   8,    9,    9,    10,   8,    9,    9,    10,
    10,    9,    9,    8,    10,   9,
7,  8,     8,    7,    7,    10,   8,    9,    9,    8,    10,   7,    9,    8,   8,
    9,     9,    8,    8,    9,
```

9, 8, 8, 7, 9, 6, 8, 9, 7, 6, 8, 9, 9, 8, 8,
9, 9, 8, 8, 9,
7, 8, 8, 9, 9, 8, 8, 9, 9, 6, 8, 9, 7, 6, 8,
9, 9, 8, 8, 9,
9, 8, 10, 7, 7, 10, 8, 9, 7, 10, 8, 7, 7, 8,
10, 7, 5, 10, 10, 7,
7, 10, 10, 9, 9, 8, 10, 9, 7, 10, 8, 9, 9, 10, 8,
7, 7, 10, 8, 7,
9, 8, 10, 9, 9, 10, 10, 9, 11, 8, 8, 9, 9, 10, 8,
9, 9, 8, 8, 9,
9, 8, 6, 9, 11, 8, 8, 9, 9, 8, 8, 11, 9, 6, 4,
9, 9, 8, 6, 11,
9, 8, 8, 9, 9, 8, 8, 9, 11, 8, 8, 9, 9, 8, 8,
11, 5, 8, 10, 7,
7, 8, 8, 9, 7, 10, 10, 9, 7, 10, 8, 9, 9, 10,
10, 9, 9, 12, 10, 9,
9, 6, 6, 5, 5, 6, 8, 9, 7, 8, 8, 7, 7, 8, 8,
9, 9, 6, 8, 7,
9, 8, 6, 9, 5, 6, 8, 7, 5, 6, 8, 7, 7, 8,
10, 9, 7, 8, 8, 7,
7, 8, 8, 7, 9, 10, 8, 7, 5, 6, 8, 5, 7, 6, 8,
7, 7, 8, 8, 7,
7, 10, 8, 9, 7, 8, 8, 9, 7, 10, 8, 7, 9, 8, 4,
7, 7, 6, 8, 9,
9, 6, 8, 9, 9, 8, 8, 9, 9, 8, 6, 9, 9, 8, 8,
9, 9, 8, 10, 7,
7, 10, 8, 9, 7, 8, 10, 7, 7, 10, 10, 7, 9, 10, 8,
7, 7, 10, 10, 5,
11, 8, 10, 9, 9, 8, 8, 9, 9, 8, 8, 9, 9, 8, 8,
9, 11, 8, 8, 9,
9, 6, 8, 11, 7, 8, 8, 5, 9, 8, 8, 7, 9, 8, 8,
7, 7, 10, 8, 7,
7, 8, 6, 7, 7, 10, 6, 5, 9, 8, 8, 9, 9, 6, 8,
9, 9, 8, 6, 9,
9, 6, 6, 9, 9, 8, 6, 7, 7, 6, 8, 9, 9, 6, 8,
7, 9, 8, 6, 9,
7, 6, 8, 9, 9, 8, 8, 9, 9, 8, 8, 7, 7, 6, 6,
9, 7, 8, 8, 9,
7, 8, 8, 7, 5, 8, 10, 7, 9, 8, 8, 7, 7, 8, 6,
7, 7, 10, 6, 5,
7, 8, 8, 9, 5, 8, 8, 7, 7, 6, 10, 7, 9, 8, 8,
7, 7, 6, 8, 7,
7, 10, 8, 5, 9, 6, 8, 9, 7, 8, 6, 9, 9, 8, 8,
7, 7, 8, 8, 7,
9, 6, 6, 9, 9, 6, 8, 9, 9, 8, 4, 9, 9, 8, 8,
9, 7, 6, 8, 9,
7, 8, 6, 9, 9, 8, 6, 9, 9, 8, 8, 9, 5, 8, 8,
7, 7, 8, 6, 7,
7, 8, 8, 7, 5, 10, 6, 9, 7, 8, 8, 9, 7, 10, 8,
7, 11, 8, 6, 9,

```
9,  8,    8,   11,  9,   8,   8,   9,   9,   8,   8,   9,   11,  8,  8,
    9,    9,   8,   8,   9,
7,  10,   10,  7,   7,   8,   10,  7,   7,   8,   10,  7,   7,   10, 8,
    9,    7,   8,   8,   9,
7,  10,   8,   7,   9,   8,   8,   9,   9,   6,   8,   9,   9,   6,  6,
    7,    9,   6,   6,   7,
9,  8,    8,   9,   9,   6,   8,   9,   7,   8,   8,   7,   9,   8,  8,
    7,    9,   6,   8,   7,
5,  10,   8,   7,   7,   6,   8,   7,   7,   10,  8,   5,   5,   8,  8,
    7,    7,   8,   6,   7,
5,  8,    10,  9,   7,   8,   6,   7,   7,   8,   8,   7,   9,   10, 8,
    7,    9,   8,   6,   7,
7,  8,    6,   9,   5,   8,   8,   9,   5,   8,   6,   7,   9,   6,  8,
    9,    7,   8,   6,   9,
7,  10,   10,  7,   7,   8,   10,  7,   7,   8,   10,  9,   7,   10, 8,
    7,    7,   8,   8,   7,
9,  10,   8,   7,   11,  8,   8,   9,   9,   8,   8,   11,  9,   8,  6,
    9,    9,   6,   8,   9,
9,  6,    8,   9,   9,   8,   6,   11,  9,   8,   8,   7,   5,   8,  8,
    9,    9,   10,  10,  7,
7,  10,   10,  9,   9,   10,  10,  9,   9,   10,  10,  9,   7,   6,  8,
    7,    9,   6,   8,   9,
9,  8,    8,   9,   9,   10,  8,   11,  9,   8,   8,   11,  9,   10, 8,
    9,    5,   6,   6,   5,
7,    8,  8,   7,   9,   10,  8,   7,   7,   8,   10,  9,   7,   8,  8,
    7,    7,   8,   8,   7,
7,  8,    4,   7,   7,   4,   8,   9,   9,   6,   6,   9,   9,   6,  8,
    9,    7,   8,   6,   9,
9,  8,    8,   7,   7,   4,   6,   7,   9,   6,   8,   7,   9,   8,  6,
    9,    9,   8,   8,   11,
9,  8,    6,   9,   7,   8,   8,   9,   5,   8,   8,   7,   3,   8,
    10,   5,   7,   10,  8,   7,
7,  8,    10,  7,   7,   10,  8,   7,   7,   8,   10,  7,   9,   8,  8,
    9,    9,   8,   8,   9,
9,  8,    8,   7,   9,   8,   8,   9,   9,   6,   6,   9,   9,   8,  8,
    7,    9,   10,  10,  7,
9,  10,   10,  9,   7,   8,   10,  7,   9,   10,  10,  7,   9,   8,
    10,   7,   7,   8,   8,   9,
9,  6,    8,   7,   9,   6,   6,   9,   9,   8,   8,   9,   9,   6,  8,
    11,   7,   6,   6,   9,
7,  6,    6,   7,   7,   10,  8,   7,   7,   8,   10,  7,   5,   10, 8,
    7,    7,   8,   10,  7,
5,  10,   8,   5,   7,   8,   8,   5,   7,   6,   8,   7,   7,   8,  8,
    5,    9,   10,  6,   7,
7,  8,    10,  9,   7,   8,   8,   7,   5,   8,   8,   7,   9,   8,  6,
    9,    7,   6,   8,   7,
9,  8,    4,   9,   9,   8,   6,   9,   7,   8,   4,   9,   7,   6,  8,
    7,    7,   8,   6,   9,
9,  8,    8,   7,   9,   6,   8,   9,   7,   8,   8,   7,   9,   8,  6,
    9,    9,   6,   8,   7,
```

```
7,  8,  6,  7,  7,  10,  8,  7,  9,  8,  8,  5,  7,  8,  8,
    9,  7,  6,  10,  7,
7,  8,  8,  5,  7,  10,  8,  7,  5,  10,  10,  7,  7,  8,  8,
    7,  7,  8,  8,  7,
7,  8,  10,  7,  7,  8,  10,  7,  9,  6,  6,  9,  9,  6,  6,
    9,  9,  8,  6,  7,
7,  8,  8,  9,  7,  6,  6,  11,  9,  8,  6,  7,  9,  8,
    10,  9,  5,  10,  8,  9,
9,  10,  8,  7,  7,  10,  10,  9,  9,  10,  8,  9,  7,  10,
    10,  9,  9,  6,  8,  9,
9,  8,  8,  7,  9,  8,  8,  9,  9,  10,  8,  11,  9,  8,  6,
    11,  7,  10,  8,  9,
7,  10,  8,  7,  9,  10,  8,  7,  7,  8,  8,  7,  7,  8,  8,
    5,  7,  10,  10,  7,
7,  8,  10,  7,  7,  8,  8,  9,  9,  6,  8,  7,  9,  8,  8,
    7,  7,  6,  8,  9,
9,  4,  6,  11,  9,  6,  8,  9,  7,  8,  8,  9,  9,  6,  8,
    7,  9,  8,  6,  7,
9,  6,  6,  7,  7,  8,  8,  9,  9,  8,  8,  9,  7,  8,  8,
    7,  7,  10,  8,  7,
9,  8,  6,  7,  5,  8,  8,  9,  5,  6,  10,  7,  7,  8,  8,
    7,  9,  8,  8,  9,
9,  6,  8,  9,  9,  8,  8,  9,  7,  8,  8,  9,  7,  6,  8,
    9,  9,  8,  8,  9,
9,  10,  10,  7,  9,  10,  10,  9,  7,  10,  10,  9,  7,  8,  8,
    7,  9,  8,  10,  7,
7,  10,  8,  9,  9,  8,  8,  5,  7,  8,  8,  9,  7,  10,
    10,  9,  9,  10,  10,  9,
9,  10,  10,  7,  9,  10,  10,  9,  7,  6,  8,  9,  7,  6,  8,
    9,  9,  8,  10,  11,
9,  8,  8,  9,  9,  8,  8,  9,  11,  10,  8,  9,  5,  6,  6,
    7,  5,  8,  8,  7,
7,  8,  8,  7,  9,  8,  10,  7,  7,  8,  10,  7,  9,  8,  8,
    7,  7,  4,  6,  9,
7,  6,  8,  7,  9,  6,  8,  9,  9,  6,  8,  7,  9,  8,  8,
    9,  11,  8,  8,  9,
7,  8,  4,  7,  7,  4,  8,  9,  9,  6,  6,  9,  9,  6,  8,
    9,  7,  8,  6,  9,
9,  8,  8,  7,  5,  8,  8,  3,  7,  8,  10,  5,  7,  8,  8,
    7,  7,  8,  10,  7,
7,  10,  10,  7,  7,  8,  10,  7,  9,  8,  8,  9,  9,  8,  8,
    9,  9,  6,  8,  9,
9,  8,  8,  9,  9,  8,  8,  7,  9,  8,  6,  7,  9,  10,
    10,  9,  7,  10,  10,  9,
9,  10,  10,  7,  7,  10,  10,  7,  9,  8,  8,  7,  7,  8,  8,
    9,  7,  8,  6,  9,
9,  8,  8,  7,  7,  6,  8,  9,  9,  8,  8,  9,  9,  8,  6,
    9,  7,  6,  8,  7,
7,  10,  8,  9,  7,  10,  8,  7,  7,  10,  8,  7,  7,  8,  8,
    7,  7,  10,  8,  7,
```

```
5,  8,     10,  7,   7,   8,   6,   7,   7,   10,  8,   7,   7,   10,  8,
    7,     9,   8,   8,   7,
7,  8,     8,   5,   9,   8,   6,   5,   7,   8,   8,   9,   9,   6,   8,
    7,     7,   6,   6,   11,
9,  8,     8,   9,   9,   8,   8,   7,   9,   6,   4,   9,   9,   6,   8,
    9,     7,   6,   6,   9,
9,  6,     6,   9,   9,   8,   8,   7,   7,   6,   8,   9,   11,  6,   6,
    7,     7,   6,   8,   7,
7,  8,     8,   5,   7,   8,   8,   7,   9,   6,   10,  5,   7,   8,
    10,    7,   9,   8,   8,   7,
7,  10,    8,   5,   7,   10,  10,  7,   7,   10,  8,   7,   7,   8,   8,
    7,     7,   10,  8,   7,
7,  8,     8,   7,   7,   8,   8,   9,   9,   6,   8,   7,   9,   8,   6,
    9,     9,   6,   6,   9,
7,  8,     8,   7,   7,   8,   8,   9,   9,   8,   10,  5,   9,   10,  8,
    9,     7,   8,   10,  9,
9,  8,     10,  7,   9,   10,  10,  7,   9,   10,  10,  9,   9,   8,   8,
    9,     9,   6,   8,   9,
7,  8,     8,   9,   9,   8,   8,   9,   7,   8,   8,   9,   9,   8,   6,
    9,     7,   10,  8,   7,
7,  8,     10,  7,   7,   8,   8,   5,   5,   8,   10,  7,   5,   8,
    10,    7,   7,   8,   10,  5,
9,  8,     6,   7,   9,   6,   8,   7,   9,   4,   8,   9,   9,   4,   6,
    7,     7,   8,   8,   9,
9,  6,     8,   7,   9,   6,   6,   9,   9,   6,   6,   9,   7,   6,   8,
    11,    9,   6,   8,   9,
7,  6,     8,   7,   9,   8,   6,   7,   7,   8,   8,   7,   7,   10,  8,
    7,     5,   10,  8,   7,
9,  6,     8,   7,   5,   8,   8,   7,   9,   8,   6,   7,   9,   6,   8,
    9,     9,   8,   8,   7,
9,  6,     10,  11,  9,   8,   8,   7,   9,   8,   8,   9,   11,  10,  8,
    9,     9,   10,  10,  9,
7,  10,    10,  9,   7,   10,  8,   7,   7,   10,  8,   7,   9,   8,
    10,    9,   7,   10,  8,   9,
7,  6,     8,   9,   9,   8,   10,  11,  9,   10,  8,   9,   9,   8,   8,
    9,     11,  10,  10,  9,
9,  8,     10,  11,  7,   10,  6,   7,   7,   8,   10,  7,   9,   8,
    10,    9,   9,   10,  10,  9,
9,  10,    8,   9,   9,   10,  10,  9,   9,   8,   6,   7,   7,   4,   8,
    9,     9,   8,   6,   9,
9,  6,     8,   9,   9,   8,   8,   9,   9,   8,   8,   9,   5,   8,   8,
    3,     7,   8,   10,  5,
7,  8,     10,  7,   7,   8,   8,   7,   7,   10,  8,   7,   7,   10,
    10,    7,   5,   8,   8,   7,
3,  8,     10,  5,   7,   8,   8,   7,   7,   10,  8,   7,   7,   10,  8,
    7,     7,   10,  10,  7,
9,  2,     6,   9,   9,   6,   8,   9,   9,   6,   6,   9,   9,   6,   6,
    9,     11,  8,   6,   9,
9,  6,     8,   11,  7,   10,  10,  9,   9,   10,  10,  7,   9,   8,
    10,    9,   9,   10,  8,   7,
```

```
9,  10,    8,   9,   7,   6,   10,  9,   11,  10,  8,   9,   9,   10,  8,
    11,    9,   8,   8,   9,
9,  8,     10,  9,   11,  8,   8,   9,   9,   8,   8,   7,   7,   10,  8,
    7,     7,   8,   10,  7,
7,  8,     8,   7,   7,   10,  8,   7,   7,   10,  8,   5,   7,   10,  8,
    7,     9,   8,   6,   9,
9,  6,     6,   9,   9,   6,   6,   9,   9,   6,   6,   9,   9,   8,   6,
    9,     9,   6,   4,   9,
9,  8,     6,   9,   7,   8,   8,   9,   9,   8,   6,   9,   9,   8,   8,
    9,     9,   8,   4,   9,
7,  6,     8,   9,   5,   8,   10,  7,   7,   8,   10,  5,   7,   8,   8,
    7,     7,   10,  8,   7,
7,  10,    8,   7,   3,   8,   10,  7,   7,   10,  8,   7,   7,   8,
    10,    7,   7,   8,   10,  5,
7,  8,     8,   7,   7,   8,   8,   7,   7,   10,  10,  7,   9,   8,   6,
    7,     9,   8,   8,   9,
9,  4,     8,   9,   9,   6,   8,   7,   9,   8,   8,   9,   9,   6,   8,
    9,     11,  6,   8,   9,
9,  8,     6,   11,  9,   8,   8,   9,   9,   8,   8,   9,   9,   8,   8,
    9,     9,   8,   8,   9,
7,  10,    10,  7,   7,   8,   10,  7,   5,   8,   10,  7,   7,   8,   8,
    7,     7,   10,  8,   7,
7,  8,     8,   7,   11,  6,   8,   9,   9,   10,  10,  7,   9,   8,   8,
    9,     9,   8,   10,  9,
11, 10,    8,   9,   9,   8,   10,  11,  7,   10,  10,  9,   7,   8,
    10,    7,   9,   10,  6,   9,
9,  10,    8,   9,   9,   10,  8,   9,   7,   10,  10,  9,   9,   8,   6,
    9,     9,   6,   6,   9,
9,  6,     6,   9,   9,   6,   6,   9,   9,   4,   6,   9,   9,   6,   8,
    9,     5,   8,   10,  7,
7,  8,     10,  5,   7,   8,   10,  7,   7,   8,   8,   3,   7,   10,  8,
    7,     7,   8,   10,  7,
7,  10,    10,  7,   7,   8,   10,  7,   7,   8,   8,   7,   5,   10,  8,
    7,     7,   8,   8,   7,
7,  8,     10,  7,   9,   8,   8,   9,   9,   8,   8,   9,   9,   8,   6,
    9,     9,   6,   4,   7,
9,  8,     8,   9,   7,   6,   8,   9,   7,   10,  10,  7,   9,   8,
    10,    7,   9,   8,   10,  9,
9,  6,     8,   7,   9,   10,  10,  9,   9,   10,  10,  9,   11,  10,  8,
    9,     9,   8,   8,   11,
9,  10,    8,   9,   9,   8,   8,   9,   7,   8,   10,  9,   9,   8,   8,
    11,    9,   4,   6,   9,
9,  6,     4,   11,  9,   6,   6,   9,   9,   6,   2,   9,   11,  6,   6,
    9,     9,   6,   6,   11,
7,  10,    8,   5,   5,   10,  8,   7,   3,   10,  10,  7,   3,   10,  8,
    5,     7,   8,   10,  7,
5,  10,    8,   7,   9,   8,   10,  7,   7,   10,  8,   9,   7,   10,  8,
    7,     7,   8,   10,  7,
5,  10,    10,  7,   7,   10,  10,  9,   9,   8,   8,   9,   9,   8,   8,
    9,     7,   8,   8,   9,
```

```
9,  6,     8,    9,    7,    8,    8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     9,    8,    8,    9,
9,  8,     6,    9,    7,    8,    8,    9,    7,    6,    8,    9,    9,    8,   8,
    9,     7,    10,   10,   9,
9,  10,    10,   7,    9,    10,   8,    9,    9,    8,    6,    7,    9,    10,
    10,    9,    7,    8,    10,   9,
7,  8,     10,   7,    7,    10,   10,   7,    5,    10,   10,   5,    5,    10,  8,
    3,     7,    8,    10,   5,
3,  8,     8,    7,    11,   6,    6,    9,    9,    8,    6,    11,   9,    4,   6,
    9,     9,    6,    6,    9,
11, 4,     4,    9,    9,    6,    4,    9,    9,    8,    8,    9,    9,    8,   8,
    9,     9,    8,    8,    9,
9,  6,     8,    9,    7,    8,    8,    9,    7,    8,    6,    9,    9,    10,  8,
    7,     9,    8,    10,   9,
9,  8,     10,   9,    9,    8,    10,   9,    7,    10,   8,    7,    9,    8,
    10,    7,    7,    10,   8,    9,
7,  10,    10,   7,    7,    10,   8,    9,    7,    8,    10,   7,    7,    8,
    10,    7,    7,    8,    8,    7,
7,  6,     8,    11,   9,    10,   8,    9,    9,    10,   8,    11,   9,    8,   8,
    7,     9,    8,    8,    9,
9,  6,     8,    9,    9,    8,    8,    9,    9,    8,    8,    9,    9,    8,   6,
    9,     9,    8,    8,    7,
7,  6,     8,    9,    9,    8,    8,    9,    7,    10,   8,    7,    9,    10,
    10,    7,    7,    10,   8,    7,
7,  8,     10,   7,    7,    8,    10,   9,    7,    8,    8,    7,    7,    10,  8,
    9,     9,    10,   10,   7,
7,  10,    8,    9,    7,    8,    6,    9,    9,    10,   10,   9,    9,    10,
    10,    9,    7,    8,    6,    9,
11, 10,    6,    9,    9,    8,    8,    9,    7,    8,    8,    9,    9,    8,
    10,    11,   9,    8,    8,    9,
11, 8,     8,    9,    9,    8,    8,    11,   9,    8,    8,    9,    9,    8,   6,
    9,     9,    8,    8,    9,
9,  8,     8,    11,   9,    8,    10,   9,    7,    8,    8,    9,    9,    10,
    10,    7,    7,    8,    10,   7,
9,  8,     10,   9,    9,    10,   10,   9,    9,    10,   8,    9,    7,    8,
    10,    9,    9,    8,    8,    7,
9,  10,    10,   9,    7,    10,   8,    9,    9,    8,    10,   7,    7,    6,   8,
    9,     11,   10,   8,    9,
9,  8,     8,    9,    9,    8,    8,    9,    9,    8,    10,   11,   7,    6,   8,
    9,     7,    8,    6,    11,
9,  10,    6,    9,    7,    10,   8,    11,   9,    8,    8,    9,    9,    8,   8,
    9,     9,    8,    8,    9,
5,  8,     8,    9,    9,    12,   8,    9,    7,    10,   10,   9,    7,    10,
    10,    7,    9,    10,   10,   9,
7,  8,     10,   9,    9,    8,    10,   7,    9,    8,    8,    9,    7,    10,  8,
    9,     9,    10,   10,   9,
9,  10,    10,   7,    7,    10,   8,    9,    11,   8,    8,    9,    9,    6,   8,
    9,     9,    8,    8,    9,
9,  8,     10,   9,    11,   8,    8,    9,    9,    8,    8,    11,   7,    10,
    10,    5,    5,    10,   10,   5,
```

```
5,  10,    10,  5,   5,   10,  10,  5,   3,   10,  10,  5,   5,   10,
    10,    7,   11,  6,   6,   9,
9,  6,     6,   11,  9,   6,   4,   9,   9,   4,   6,   9,   9,   4,   6,
    9,     9,   8,   4,   11,
11, 8,     8,   9,   9,   10,  8,   9,   9,   8,   8,   9,   9,   8,   6,
    9,     11,  8,   8,   9,
9,  8,     8,   11,  9,   10,  10,  9,   7,   10,  10,  9,   7,   10,  8,
    7,     7,   10,  8,   7,
9,  8,     10,  9,   7,   10,  8,   9,   9,   10,  10,  7,   9,   10,
    10,    9,   7,   10,  10,  9,
7,  8,     8,   7,   9,   8,   10,  7,   7,   10,  8,   9,   11,  10,
    10,    9,   9,   8,   8,   11,
9,  8,     8,   9,   9,   8,   8,   9,   7,   8,   8,   9,   9,   10,  8,
    11,    11,  6,   8,   9,
9,  6,     6,   11,  9,   6,   4,   9,   9,   4,   6,   9,   9,   4,   6,
    9,     9,   6,   4,   11,
7,  10,    10,  5,   5,   10,  10,  5,   5,   10,  10,  5,   5,   10,
    10,    5,   5,   10,  10,  5,
5,  10,    10,  5,   9,   10,  10,  9,   7,   10,  10,  9,   7,   10,  8,
    7,     7,   10,  8,   7,
9,  8,     10,  9,   7,   10,  8,   9,   9,   8,   8,   9,   9,   8,   8,
    9,     9,   8,   10,  9,
9,  10,    8,   9,   11,  8,   8,   9,   9,   8,   10,  9,   11,  8,   8,
    9,     9,   6,   8,   11,
9,  8,     8,   9,   9,   8,   6,   9,   9,   6,   8,   9,   9,   8,   6,
    11,    9,   10,  10,  7,
7,  8,     10,  9,   9,   10,  10,  7,   7,   8,   10,  7,   9,   8,
    10,    7,   9,   10,  8,   9,
9,  10,    10,  7,   9,   10,  10,  9,   7,   10,  10,  9,   7,   8,   8,
    7,     9,   8,   10,  7,
7,  10,    8,   9,   11,  8,   8,   9,   9,   6,   8,   11,  9,   8,   8,
    9,     9,   8,   6,   9,
9,  6,     8,   9,   9,   8,   6,   11,  11,  8,   10,  9,   9,   8,
    10,    11,  9,   8,   8,   9,
9,  8,     8,   9,   7,   8,   8,   9,   9,   10,  8,   11,  9,   10,
    10,    7,   7,   8,   10,  9,
9,  10,    10,  7,   7,   8,   10,  7,   9,   8,   10,  7,   9,   10,  8,
    9,     7,   10,  10,  7,
7,  10,    10,  9,   7,   10,  10,  7,   7,   10,  10,  7,   5,   10,
    10,    7,   7,   10,  10,  9,
9,  8,     8,   9,   9,   8,   8,   11,  9,   8,   10,  9,   9,   6,   8,
    9,     11,  8,   8,   9,
9,  8,     8,   11,  9,   8,   8,   9,   9,   8,   8,   9,   9,   8,
    10,    9,   9,   10,  8,   9,
11, 10,    8,   9,   9,   8,   8,   9,   9,   10,  10,  7,   7,   8,
    10,    9,   7,   10,  8,   7,
9,  10,    10,  9,   9,   8,   10,  7,   7,   10,  8,   9,   9,   10,
    10,    7,   7,   8,   10,  9,
7,  10,    8,   7,   9,   10,  10,  9,   9,   8,   10,  7,   7,   10,  8,
    9,     11,  8,   8,   9,
```

```
9,  6,     8,    9,    9,    8,    8,    9,    9,    8,    10,   9,    11,   8,   8,
    9,     9,    8,    8,    11,
9,  8,     8,    9,    9,    8,    8,    11,   9,    8,    6,    9,    9,    10,  8,
    9,     11,   8,    8,    9,
9,  8,     8,    11,   7,    10,   10,   9,    9,    10,   10,   7,    9,    10,
    10,    9,    9,    10,   10,   9,
11, 10,    10,   9,    9,    10,   10,   9,    9,    8,    8,    9,    7,    10,  8,
    9,     7,    10,   10,   9,
5,  10,    8,    7,    9,    10,   10,   9,    9,    10,   10,   9,    7,    8,   8,
    9,     9,    8,    6,    9,
7,  8,     10,   11,   9,    8,    6,    9,    9,    8,    8,    9,    11,   10,  8,
    9,     7,    8,    8,    9,
5,  8,     8,    7,    7,    8,    8,    9,    7,    6,    10,   7,    5,    6,
    10,    7,    7,    8,    8,    7,
9,  6,     6,    9,    7,    8,    6,    9,    9,    8,    8,    9,    9,    6,   6,
    7,     7,    6,    6,    9,
11, 8,     6,    7,    7,    8,    8,    9,    9,    6,    8,    7,    9,    8,   6,
    7,     9,    6,    6,    7,
7,  8,     8,    9,    9,    8,    8,    9,    7,    10,   8,    7,    7,    8,
    10,    7,    7,    8,    8,    7,
5,  8,     10,   7,    7,    8,    10,   7,    7,    8,    10,   7,    9,    8,   8,
    9,     9,    8,    8,    9,
9,  8,     8,    9,    9,    8,    8,    7,    9,    6,    6,    9,    9,    8,   8,
    7,     9,    10,   10,   7,
9,  10,    10,   9,    7,    8,    10,   7,    9,    10,   10,   7,    9,    8,
    10,    7,    7,    8,    8,    9,
9,  8,     6,    9,    9,    6,    8,    7,    9,    8,    8,    9,    7,    4,   6,
    9,     7,    8,    4,    7,
9,  6,     8,    7,    7,    10,   8,    5,    7,    10,   10,   7,    7,    10,  8,
    7,     7,    8,    8,    7,
5,  10,    8,    7,    5,    8,    8,    5,    7,    8,    8,    9,    7,    10,  6,
    5,     7,    10,   8,    9,
7,  6,     8,    7,    7,    8,    8,    5,    7,    8,    8,    7,    9,    6,   8,
    9,     9,    8,    6,    9,
7,  8,     6,    11,   9,    8,    6,    9,    7,    6,    6,    7,    9,    6,   6,
    7,     7,    8,    8,    9,
7,  8,     8,    7,    9,    8,    6,    9,    9,    8,    6,    7,    9,    4,   6,
    9,     11,   6,    8,    9,
7,  8,     6,    9,    7,    8,    8,    7,    7,    8,    8,    9,    7,    8,
    10,    5,    7,    6,    10,   7,
7,  8,     8,    5,    5,    10,   8,    7,    7,    10,   8,    5,    7,    10,  8,
    7,     3,    8,    8,    7,
7,  10,    8,    7,    7,    8,    10,   7,    7,    8,    4,    9,    9,    8,   8,
    9,     7,    6,    6,    9,
7,  6,     4,    9,    7,    8,    6,    9,    9,    8,    8,    7,    9,    8,
    10,    9,    7,    10,   8,    9,
9,  10,    10,   9,    5,    8,    10,   7,    9,    10,   8,    7,    9,    10,
    10,    9,    9,    8,    8,    9,
7,  8,     8,    9,    9,    8,    8,    9,    9,    8,    6,    9,    7,    6,   8,
    9,     9,    8,    8,    9,
```

```
7,  8,    8,   7,   7,   8,   10,  7,   7,   8,   8,   5,   9,   8,
    10,   7,   7,   10,  10,  7,
7,  8,    10,  7,   7,   8,   6,   9,   7,   8,   8,   7,   9,   6,   8,
    7,    9,   8,   8,   9,
9,  8,    6,   9,   9,   6,   8,   7,   7,   8,   6,   9,   9,   8,   4,
    7,    7,   8,   8,   11,
9,  6,    6,   9,   9,   8,   6,   7,   9,   8,   8,   9,   5,   8,   6,
    9,    7,   10,  6,   7,
5,  10,   8,   9,   7,   8,   8,   7,   7,   8,   8,   7,   7,   8,   8,
    7,    9,   8,   8,   9,
9,  8,    6,   7,   9,   8,   10,  11,  9,   8,   8,   9,   9,   8,   6,
    7,    11,  10,  8,   9,
9,  10,   10,  7,   7,   8,   10,  9,   7,   10,  8,   7,   9,   10,
    10,   9,   9,   8,   10,  7,
7,  10,   8,   9,   11,  10,  8,   9,   9,   10,  10,  9,   9,   8,   8,
    9,    9,   8,   10,  9,
9,  6,    8,   9,   9,   8,   10,  9,   9,   10,  10,  9,   9,   8,
    10,   9,   9,   8,   8,   7,
9,  8,    8,   7,   7,   10,  10,  9,   9,   8,   10,  7,   9,   8,   6,
    9,    9,   8,   8,   9,
9,  6,    8,   9,   7,   6,   6,   7,   9,   8,   6,   9,   9,   6,   8,
    9,    7,   10,  8,   7,
7,  8,    10,  7,   7,   8,   10,  5,   5,   8,   8,   7,   5,   8,   8,
    7,    7,   10,  10,  5,
7,  10,   10,  7,   9,   8,   8,   7,   7,   8,   10,  7,   7,   8,   8,
    5,    7,   10,  8,   7,
7,  8,    10,  7,   9,   8,   6,   9,   9,   6,   6,   9,   9,   6,   6,
    9,    9,   6,   6,   9,
9,  4,    6,   9,   9,   6,   8,   9,   9,   10,  8,   9,   7,   8,
    10,   9,   9,   10,  8,   9,
7,  8,    8,   9,   7,   10,  8,   9,   9,   10,  10,  7,   9,   8,   8,
    9,    9,   8,   8,   9,
9,  10,   8,   9,   9,   8,   8,   9,   9,   10,  10,  9,   9,   8,   8,
    11,   7,   10,  10,  7,
5,  8,    10,  7,   7,   8,   10,  7,   7,   8,   8,   5,   5,   8,   8,
    7,    7,   8,   10,  5,
11, 6,    6,   9,   9,   6,   6,   11,  9,   6,   6,   9,   9,   6,   6,
    9,    9,   6,   6,   9,
9,  6,    6,   9,   9,   8,   8,   9,   9,   6,   8,   9,   9,   6,   6,
    7,    9,   6,   6,   7,
9,  8,    8,   9,   9,   6,   8,   9,   7,   10,  10,  7,   7,   8,
    10,   7,   7,   8,   8,   5,
5,  10,   8,   7,   5,   8,   8,   7,   7,   8,   10,  5,   7,   8,   8,
    7,    7,   8,   10,  7,
7,  8,    10,  9,   5,   8,   8,   7,   7,   10,  8,   7,   7,   10,
    10,   7,   9,   8,   6,   9,
7,  6,    8,   9,   9,   8,   6,   9,   7,   6,   6,   9,   9,   8,   6,
    9,    9,   8,   8,   9,
9,  6,    6,   9,   9,   6,   8,   9,   9,   6,   6,   9,   9,   6,   6,
    9,    9,   4,   6,   9,
```

```
9,  6,     8,    9,    7,    10,   8,    7,    5,    8,    10,   7,    7,    8,   8,
    7,     7,    10,   8,    5,
5,  8,     8,    7,    7,    10,   10,   5,    9,    10,   8,    9,    9,    8,
    10,    11,   9,    10,   8,    9,
9,  8,     8,    9,    9,    6,    10,   9,    9,    8,    10,   9,    9,    10,  8,
    9,     9,    10,   10,   9,
9,  8,     10,   9,    7,    8,    8,    7,    7,    10,   8,    9,    9,    8,
    10,    7,    9,    6,    8,    9,
9,  8,     6,    9,    9,    8,    8,    9,    9,    8,    8,    9,    11,   10,  8,
    9,     9,    8,    8,    11,
7,  10,    8,    7,    5,    8,    8,    7,    7,    8,    8,    7,    7,    10,  8,
    9,     7,    10,   8,    7,
7,  10,    10,   7,    7,    8,    10,   7,    7,    8,    10,   7,    7,    8,   8,
    5,     9,    10,   8,    7,
7,  10,    8,    7,    7,    8,    10,   7,    9,    8,    6,    9,    7,    6,   8,
    9,     9,    6,    6,    7,
9,  8,     8,    9,    9,    8,    6,    9,    9,    6,    8,    9,    9,    10,  8,
    9,     7,    8,    10,   9,
9,  8,     8,    7,    9,    10,   10,   9,    7,    10,   8,    9,    9,    8,
    10,    7,    9,    8,    8,    9,
9,  10,    8,    9,    9,    8,    8,    9,    9,    8,    10,   9,    11,   10,  8,
    9,     9,    8,    8,    9,
11, 8,     8,    9,    9,    8,    8,    11,   9,    8,    8,    9,    9,    6,   8,
    9,     9,    8,    8,    9,
9,  8,     8,    11,   7,    10,   10,   7,    7,    8,    10,   7,    7,    8,
    10,    9,    7,    10,   8,    7,
7,  8,     8,    7,    9,    10,   8,    7,    9,    6,    8,    9,    7,    8,   6,
    9,     9,    8,    6,    7,
7,  8,     6,    7,    9,    6,    8,    9,    9,    8,    8,    9,    7,    6,   8,
    7,     7,    8,    8,    5,
7,  6,     10,   9,    7,    8,    8,    5,    7,    8,    8,    7,    9,    10,  8,
    7,     7,    8,    10,   7,
7,  6,     8,    7,    7,    8,    10,   7,    5,    8,    8,    9,    5,    6,   8,
    9,     7,    8,    8,    7,
9,  8,     8,    7,    9,    6,    8,    9,    9,    6,    8,    9,    9,    4,   6,
    7,     9,    8,    8,    9,
9,  8,     8,    9,    7,    10,   10,   7,    9,    8,    10,   7,    9,    8,
    10,    7,    7,    10,   8,    7,
7,  8,     8,    7,    7,    10,   8,    7,    11,   8,    6,    9,    9,    8,   8,
    11,    9,    8,    8,    9,
9,  8,     8,    9,    11,   6,    8,    9,    9,    6,    6,    9,    7,    8,
    10,    7,    9,    8,    6,    5,
9,  8,     10,   7,    5,    8,    8,    7,    7,    8,    6,    7,    7,    8,   8,
    7,     9,    8,    6,    9,
9,  8,     8,    9,    7,    6,    6,    9,    9,    8,    6,    9,    9,    8,   6,
    9,     7,    6,    8,    9,
9,  8,     8,    7,    7,    8,    6,    9,    9,    8,    8,    9,    5,    6,   8,
    7,     9,    6,    6,    5,
7,  8,     6,    9,    7,    8,    8,    9,    7,    8,    8,    7,    7,    8,
    10,    7,    9,    8,    6,    7,
```

```
5,  8,     8,    5,    7,    10,   6,    7,    7,    8,    8,    9,    7,    6,   8,
    7,     7,    8,    10,   7,
9,  8,     8,    5,    5,    6,    8,    7,    7,    10,   8,    7,    9,    6,   8,
    5,     9,    8,    8,    9,
7,  6,     8,    7,    7,    6,    8,    5,    9,    6,    8,    7,    9,    8,   6,
    9,     9,    8,    8,    9,
9,  6,     8,    9,    9,    8,    8,    9,    7,    4,    6,    9,    9,    8,   6,
    7,     9,    8,    8,    9,
7,  8,     10,   5,    7,    8,    8,    7,    7,    8,    10,   7,    7,    8,   6,
    9,     5,    8,    8,    9,
7,  8,     6,    7,    11,   6,    8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     9,    4,    8,    9,
11, 8,     8,    9,    9,    8,    8,    11,   9,    8,    10,   7,    7,    8,   8,
    9,     7,    10,   10,   7,
7,  10,    8,    7,    5,    10,   10,   7,    7,    10,   10,   9,    9,    8,   6,
    9,     7,    6,    8,    9,
9,  6,     6,    7,    9,    8,    8,    9,    9,    8,    6,    9,    9,    6,   8,
    9,     7,    6,    10,   5,
9,  8,     8,    5,    9,    6,    10,   7,    7,    8,    8,    7,    7,    8,   8,
    7,     7,    8,    8,    7,
7,  8,     8,    7,    7,    8,    6,    5,    7,    8,    10,   9,    7,    8,   8,
    7,     7,    8,    6,    5,
9,  10,    8,    7,    9,    6,    8,    7,    9,    6,    6,    9,    7,    8,   6,
    9,     9,    8,    8,    9,
9,  8,     8,    7,    7,    8,    6,    9,    9,    8,    10,   7,    9,    10,  8,
    5,     9,    8,    12,   9,
7,  10,    10,   7,    9,    10,   8,    7,    9,    10,   10,   9,    9,    8,   8,
    9,     9,    8,    8,    11,
9,  8,     6,    9,    9,    10,   8,    9,    11,   8,    8,    9,    9,    8,   8,
    11,    9,    8,    10,   9,
9,  10,    8,    9,    9,    10,   10,   7,    7,    10,   8,    5,    9,    10,
    10,    9,    7,    8,    10,   9,
9,  8,     8,    9,    9,    8,    8,    9,    9,    6,    8,    9,    7,    8,   8,
    9,     9,    6,    8,    9,
9,  8,     8,    7,    7,    8,    8,    7,    9,    8,    8,    7,    7,    10,  8,
    5,     7,    6,    10,   7,
5,  6,     8,    9,    7,    8,    8,    7,    9,    8,    6,    7,    9,    6,   8,
    7,     9,    4,    8,    9,
9,  4,     6,    7,    7,    8,    8,    9,    9,    6,    8,    7,    7,    8,   6,
    9,     9,    8,    8,    7,
11, 6,     6,    7,    7,    8,    6,    9,    9,    4,    8,    9,    9,    8,   8,
    9,     5,    8,    8,    7,
7,  10,    10,   5,    7,    8,    8,    7,    7,    8,    8,    3,    7,    10,
    10,    7,    7,    8,    10,   7,
7,  6,     10,   9,    11,   8,    8,    9,    11,   8,    10,   9,    9,    8,   6,
    9,     9,    8,    8,    9,
7,  8,     8,    9,    9,    10,   8,    7,    7,    10,   10,   9,    7,    10,
    10,    9,    7,    10,   10,   9,
9,  8,     8,    7,    7,    10,   8,    9,    9,    6,    8,    9,    9,    8,   6,
    9,     11,   6,    8,    7,
```

```
7,  6,     6,    9,    7,    6,    8,    9,    9,    6,    6,    7,    7,    10,  8,
    7,     7,    10,   10,   7,
5,  8,     8,    7,    7,    8,    8,    7,    5,    10,   10,   5,    7,    8,   8,
    5,     7,    8,    8,    7,
9,  8,     8,    7,    7,    10,   8,    7,    5,    8,    10,   7,    7,    6,   8,
    9,     7,    6,    8,    5,
7,  8,     8,    9,    9,    8,    8,    9,    9,    6,    6,    9,    9,    6,   4,
    7,     7,    8,    6,    9,
7,  4,     8,    7,    7,    8,    8,    9,    7,    6,    8,    7,    9,    8,   6,
    9,     7,    6,    6,    9,
7,  8,     8,    9,    9,    8,    8,    9,    7,    6,    8,    5,    9,    10,  8,
    5,     7,    8,    8,    7,
7,  6,     8,    7,    7,    8,    10,   9,    7,    8,    8,    7,    7,    10,  8,
    7,     7,    8,    10,   7,
7,  10,    8,    7,    7,    8,    10,   5,    7,    8,    8,    7,    7,    8,
    10,    7,    9,    6,    6,    7,
11, 8,     6,    9,    9,    6,    6,    9,    9,    8,    6,    9,    7,    6,   8,
    9,     7,    8,    6,    7,
9,  8,     10,   7,    9,    10,   8,    9,    9,    8,    10,   7,    7,    8,
    10,    5,    9,    10,   10,   9,
9,  10,    10,   9,    9,    8,    8,    9,    9,    8,    8,    9,    9,    8,   8,
    7,     9,    8,    6,    9,
7,  6,     8,    9,    9,    8,    8,    9,    7,    10,   8,    7,    5,    8,   8,
    7,     7,    10,   8,    7,
7,  8,     10,   9,    7,    10,   8,    7,    7,    8,    10,   7,    7,    4,   8,
    7,     11,   8,    8,    7,
9,  6,     8,    9,    9,    6,    6,    9,    9,    8,    8,    9,    7,    6,   8,
    9,     7,    8,    6,    9,
7,  8,     8,    7,    9,    6,    8,    7,    9,    8,    8,    9,    9,    8,   6,
    9,     9,    6,    8,    7,
5,  6,     8,    7,    9,    10,   8,    7,    7,    8,    8,    7,    7,    8,   8,
    7,     7,    8,    10,   9,
5,  6,     8,    7,    9,    6,    10,   7,    11,   8,    8,    7,    11,   6,
    10,    9,    9,    8,    8,    9,
9,  8,     8,    9,    9,    8,    8,    9,    9,    10,   10,   7,    7,    8,
    10,    9,    7,    10,   8,    7,
9,  10,    10,   9,    9,    8,    10,   7,    7,    10,   8,    9,    9,    8,   8,
    7,     9,    10,   8,    9,
5,  10,    10,   9,    7,    10,   8,    9,    9,    10,   10,   9,    7,    10,
    10,    9,    7,    8,    8,    9,
9,  8,     6,    9,    9,    8,    8,    9,    7,    10,   6,    11,   9,    8,   8,
    11,    9,    10,   8,    9,
7,  8,     8,    5,    9,    8,    8,    7,    7,    10,   6,    7,    7,    8,
    10,    7,    5,    8,    8,    9,
7,  8,     6,    7,    7,    8,    8,    9,    9,    6,    8,    7,    9,    8,   6,
    9,     9,    6,    6,    9,
7,  8,     8,    7,    7,    8,    8,    9,    9,    6,    6,    7,    9,    8,   6,
    9,     9,    6,    6,    9,
9,  8,     6,    11,   7,    6,    8,    9,    7,    8,    6,    7,    7,    10,  8,
    7,     7,    8,    10,   7,
```

```
5,  10,   8,   7,   7,   8,   10,  7,   7,   10,  8,   7,   7,   8,   8,
    7,    9,   8,   8,   9,
9,  8,    8,   9,   9,   6,   8,   9,   9,   8,   8,   9,   9,   8,   8,
    9,    7,   8,   6,   7,
9,  10,   10,  9,   7,   10,  10,  9,   9,   10,  10,  7,   7,   10,
    10,   7,   9,   8,   8,   7,
7,  8,    8,   9,   7,   8,   8,   7,   9,   8,   8,   7,   9,   6,   8,
    9,    9,   6,   6,   11,
9,  8,    8,   9,   7,   6,   4,   9,   7,   8,   8,   7,   7,   8,
    10,   7,   9,   10,  8,   7,
7,  8,    10,  7,   7,   10,  8,   5,   7,   8,   10,  7,   7,   8,   6,
    7,    9,   8,   8,   7,
7,  10,   8,   7,   7,   8,   10,  7,   7,   8,   8,   9,   5,   8,   6,
    5,    7,   8,   6,   7,
9,  8,    8,   7,   9,   6,   8,   9,   9,   8,   8,   9,   9,   8,   6,
    7,    9,   6,   8,   7,
9,  8,    6,   9,   9,   6,   8,   7,   7,   4,   4,   7,   9,   8,   6,
    9,    7,   8,   8,   9,
9,  6,    8,   7,   7,   8,   8,   7,   9,   10,  6,   5,   7,   8,   6,
    5,    7,   8,   8,   7,
7,  8,    10,  9,   7,   8,   8,   7,   5,   10,  8,   7,   7,   10,  8,
    5,    3,   8,   8,   7,
7,  8,    8,   7,   7,   10,  10,  7,   7,   8,   10,  7,   7,   8,   6,
    9,    9,   8,   4,   7,
9,  6,    6,   7,   7,   8,   6,   9,   9,   8,   8,   11,  9,   8,   8,
    9,    9,   8,   10,  7,
9,  10,   8,   9,   5,   8,   8,   7,   9,   10,  10,  9,   9,   10,
    10,   9,   7,   10,  10,  9,
9,  8,    8,   9,   9,   8,   6,   7,   9,   6,   8,   7,   9,   10,  8,
    11,   9,   8,   8,   11,
9,  10,   8,   9,   7,   10,  8,   7,   5,   10,  10,  7,   7,   8,   8,
    7,    7,   8,   8,   5,
5,  8,    10,  7,   7,   8,   10,  5,   9,   6,   8,   9,   9,   8,   6,
    9,    9,   6,   8,   7,
7,  6,    6,   9,   7,   6,   8,   11,  9,   6,   6,   7,   7,   8,   4,
    9,    9,   8,   8,   9,
7,  6,    6,   9,   7,   6,   4,   9,   7,   8,   6,   9,   9,   8,   8,
    7,    5,   8,   6,   7,
9,  10,   6,   7,   7,   8,   8,   7,   5,   8,   8,   7,   7,   8,
    10,   9,   7,   8,   8,   7,
9,  8,    8,   7,   9,   8,   8,   9,   9,   8,   8,   9,   9,   8,   6,
    9,    7,   8,   8,   9,
9,  8,    6,   9,   9,   10,  10,  7,   7,   8,   10,  9,   9,   10,
    10,   7,   7,   8,   10,  7,
9,  8,    10,  7,   9,   10,  8,   9,   11,  8,   8,   9,   9,   8,   8,
    11,   9,   8,   6,   9,
9,  8,    8,   9,   9,   8,   8,   9,   9,   8,   8,   11,  7,   10,
    10,   7,   7,   8,   10,  7,
7,  8,    10,  7,   7,   10,  8,   9,   7,   8,   8,   9,   7,   10,  8,
    7,    9,   6,   8,   7,
```

```
9,  8,    6,   9,   7,   8,   8,   9,   9,   6,   6,   9,   9,   8,   8,
    7,    7,   8,   6,   9,
7,  8,    10,  7,   7,   6,   8,   7,   5,   8,   8,   9,   7,   10,  8,
    7,    5,   8,   8,   7,
9,  8,    6,   7,   7,   6,   8,   7,   7,   8,   8,   5,   7,   8,   8,
    7,    7,   10,  8,   9,
7,  8,    6,   9,   5,   10,  8,   7,   9,   8,   8,   9,   7,   6,   8,
    9,    9,   8,   4,   9,
9,  8,    6,   9,   9,   8,   6,   9,   7,   8,   8,   9,   7,   10,
    10,   9,   7,   8,   10,  7,
7,  8,    10,  7,   9,   10,  8,   7,   7,   8,   8,   7,   7,   10,  8,
    7,    11,  8,   6,   9,
9,  8,    8,   11,  9,   8,   8,   9,   9,   8,   8,   9,   11,  6,   8,
    9,    9,   6,   6,   9,
7,  8,    8,   7,   9,   6,   8,   7,   9,   8,   8,   7,   7,   10,  8,
    7,    5,   8,   8,   7,
5,  10,   6,   7,   9,   8,   6,   7,   9,   6,   8,   9,   9,   6,   8,
    9,    9,   6,   8,   9,
9,  8,    6,   7,   9,   6,   8,   9,   9,   6,   8,   9,   5,   8,   8,
    9,    7,   8,   6,   7,
7,  8,    8,   9,   9,   6,   6,   7,   5,   8,   6,   9,   7,   6,
    10,   9,   5,   8,   8,   5,
7,  8,    8,   7,   9,   10,  8,   7,   7,   8,   6,   7,   7,   8,   8,
    7,    7,   8,   10,  9,
7,  8,    6,   5,   5,   6,   8,   7,   9,   10,  8,   7,   7,   8,   8,
    7,    7,   8,   8,   7,
9,  8,    8,   7,   7,   8,   6,   9,   5,   6,   6,   5,   9,   8,   8,
    7,    9,   6,   8,   9,
7,  8,    6,   9,   9,   8,   8,   9,   9,   6,   8,   9,   7,   6,   4,
    7,    9,   8,   6,   9,
9,  8,    8,   9,   9,   8,   8,   9,   7,   8,   8,   7,   7,   8,   6,
    5,    7,   6,   8,   5,
7,  10,   8,   9,   7,   8,   8,   9,   7,   10,  8,   7,   11,  6,   8,
    9,    9,   8,   6,   9,
9,  8,    4,   9,   9,   8,   8,   9,   11,  8,   8,   9,   9,   8,   8,
    11,   9,   8,   10,  9,
7,  10,   8,   5,   7,   8,   10,  7,   9,   12,  10,  9,   9,   10,  8,
    9,    7,   10,  10,  9,
9,  8,    6,   9,   9,   8,   8,   9,   9,   6,   8,   9,   7,   6,   6,
    7,    9,   8,   6,   9,
9,  6,    8,   9,   7,   8,   8,   7,   9,   8,   8,   7,   9,   8,   8,
    5,    7,   10,  6,   7,
5,  6,    8,   7,   7,   10,  8,   7,   7,   8,   10,  7,   5,   8,   8,
    7,    7,   8,   8,   9,
7,  10,   6,   7,   5,   6,   8,   7,   9,   8,   8,   7,   9,   6,   8,
    9,    7,   6,   6,   9,
9,  8,    8,   7,   7,   6,   8,   7,   9,   6,   8,   9,   9,   8,   8,
    9,    9,   8,   10,  7,
7,  8,    8,   9,   7,   10,  10,  7,   7,   10,  8,   7,   5,   10,
    10,   7,   7,   10,  10,  9,
```

```
9,  8,     8,    9,    9,    8,    8,    11,   9,    8,    10,   9,    9,    6,   8,
    9,     11,   8,    8,    9,
9,  8,     8,    11,   7,    10,   8,    9,    9,    8,    6,    11,   9,    10,  8,
    9,     9,    8,    8,    9,
11, 10,    10,   9,    9,    8,    10,   11,   7,    10,   6,    9,    9,    10,
    10,    7,    7,    8,    10,   9,
7,  10,    8,    9,    9,    10,   8,    9,    9,    10,   10,   9,    9,    8,   8,
    9,     9,    4,    8,    9,
9,  8,     6,    7,    9,    6,    8,    9,    9,    8,    8,    7,    9,    6,   8,
    9,     7,    10,   10,   5,
7,  8,     10,   7,    7,    8,    8,    7,    7,    10,   8,    7,    7,    10,  8,
    7,     7,    8,    8,    7,
7,  10,    10,   7,    5,    8,    10,   7,    7,    8,    10,   7,    7,    8,   8,
    7,     7,    8,    8,    7,
7,  8,     10,   7,    11,   6,    8,    9,    9,    8,    6,    11,   9,    8,   8,
    9,     9,    8,    8,    9,
9,  8,     8,    9,    9,    8,    8,    9,    7,    10,   10,   9,    9,    8,
    10,    7,    9,    8,    10,   7,
9,  10,    10,   9,    9,    10,   8,    7,    9,    6,    10,   9,    11,   8,   8,
    9,     9,    10,   10,   11,
9,  8,     8,    9,    9,    8,    10,   9,    11,   8,    8,    9,    9,    8,   8,
    7,     5,    10,   10,   7,
7,  8,     8,    5,    7,    8,    10,   7,    7,    8,    8,    7,    7,    10,  8,
    3,     7,    8,    10,   7,
9,  6,     6,    9,    9,    6,    8,    9,    9,    6,    6,    9,    9,    6,   6,
    9,     9,    8,    6,    9,
9,  6,     4,    9,    9,    8,    6,    9,    9,    8,    8,    9,    9,    8,   8,
    9,     7,    6,    8,    9,
9,  8,     4,    7,    9,    6,    8,    9,    7,    10,   8,    7,    7,    8,
    10,    7,    7,    8,    10,   7,
7,  8,     8,    7,    7,    10,   8,    7,    5,    10,   8,    7,    5,    10,
    10,    7,    7,    8,    8,    5,
7,  8,     8,    3,    7,    10,   8,    7,    7,    10,   8,    7,    7,    8,
    10,    7,    9,    8,    6,    9,
9,  8,     8,    9,    7,    4,    6,    7,    9,    8,    8,    9,    9,    8,   8,
    9,     9,    6,    8,    9,
9,  8,     6,    9,    9,    6,    2,    9,    9,    6,    6,    9,    9,    6,   6,
    9,     11,   8,    6,    9,
9,  6,     8,    11,   5,    10,   8,    7,    7,    8,    8,    5,    3,    8,   8,
    7,     7,    10,   8,    7,
7,  10,    8,    7,    7,    10,   10,   7,    11,   10,   8,    9,    9,    10,  6,
    7,     9,    8,    8,    9,
9,  8,     10,   9,    11,   10,   8,    9,    9,    8,    10,   11,   7,    10,
    10,    9,    9,    8,    10,   7,
7,  8,     6,    7,    9,    10,   8,    9,    9,    10,   10,   9,    9,    10,
    10,    9,    9,    6,    6,    9,
9,  6,     8,    9,    9,    6,    6,    9,    9,    6,    6,    9,    9,    4,   6,
    9,     9,    6,    8,    9,
7,  10,    8,    7,    7,    8,    10,   7,    7,    8,    8,    7,    7,    10,  8,
    5,     7,    8,    8,    7,
```

```
7,  10,    10,  7,   5,   10,  8,   7,   7,   8,   8,   5,   7,   8,
    10,    7,   3,   8,   8,   7,
7,  10,    8,   7,   7,   10,  10,  7,   9,   8,   6,   9,   9,   8,   8,
    9,     7,   8,   6,   9,
7,  6,     4,   9,   9,   8,   8,   9,   9,   8,   8,   9,   7,   10,
    10,    9,   9,   8,   10,  7,
9,  10,    10,  9,   7,   6,   8,   9,   9,   10,  8,   7,   9,   10,
    10,    9,   11,  8,   8,   9,
9,  8,     10,  11,  9,   10,  8,   9,   9,   8,   8,   9,   7,   8,
    10,    9,   9,   8,   8,   11,
9,  8,     10,  9,   7,   8,   8,   9,   9,   10,  10,  5,   9,   10,
    10,    7,   9,   10,  10,  7,
9,  8,     10,  9,   9,   8,   8,   9,   7,   8,   8,   9,   9,   6,   8,
    9,     9,   8,   8,   9,
9,  6,     8,   9,   9,   8,   8,   7,   5,   8,   8,   7,   7,   8,   6,
    7,     7,   8,   8,   7,
9,  8,     10,  7,   7,   8,   10,  5,   9,   6,   8,   7,   9,   6,   6,
    9,     9,   6,   6,   9,
7,  6,     8,   9,   11,  6,   8,   9,   7,   6,   8,   7,   9,   8,   6,
    7,     7,   8,   6,   9,
9,  8,     8,   7,   9,   6,   8,   9,   7,   8,   8,   7,   9,   8,   6,
    9,     9,   6,   8,   7,
7,  10,    8,   7,   7,   10,  10,  7,   7,   10,  8,   5,   7,   8,   8,
    7,     7,   8,   8,   7,
7,  8,     10,  7,   7,   8,   10,  11,  9,   6,   6,   9,   9,   8,   8,
    9,     11,  10,  8,   9,
9,  8,     8,   7,   9,   8,   8,   9,   9,   10,  8,   7,   7,   10,
    10,    9,   7,   10,  10,  9,
7,  10,    10,  9,   9,   8,   8,   7,   7,   10,  8,   9,   7,   8,   8,
    9,     9,   4,   8,   9,
9,  6,     6,   7,   9,   6,   8,   9,   7,   8,   6,   7,   9,   4,   8,
    7,     7,   10,  8,   5,
7,  8,     10,  7,   7,   8,   8,   7,   7,   8,   10,  7,   5,   10,
    10,    7,   5,   8,   8,   5,
7,  8,     8,   7,   5,   10,  8,   7,   7,   10,  8,   7,   9,   8,   8,
    7,     7,   6,   8,   7,
9,  6,     8,   5,   9,   6,   8,   11,  7,   6,   6,   9,   9,   6,   6,
    9,     9,   8,   8,   7,
7,  6,     8,   9,   9,   6,   6,   7,   7,   8,   8,   9,   9,   6,   4,
    7,     7,   6,   6,   9,
11, 8,     8,   9,   9,   8,   8,   7,   9,   6,   8,   9,   7,   8,   8,
    7,     7,   8,   6,   5,
5,  8,     6,   7,   9,   8,   10,  7,   7,   8,   10,  7,   9,   8,   8,
    7,     5,   10,  8,   7,
7,  8,     8,   5,   7,   8,   8,   3,   7,   8,   10,  7,   7,   10,
    10,    7,   7,   8,   10,  7,
9,  8,     6,   9,   9,   6,   8,   7,   7,   4,   4,   7,   9,   8,   6,
    9,     7,   8,   8,   9,
9,  6,     8,   7,   9,   8,   10,  9,   7,   10,  8,   9,   7,   8,   8,
    5,     9,   10,  10,  7,
```

```
9,  10,    10,  9,   9,   10,  10,  9,   9,   8,   10,  11,  9,   8,  6,
    7,     7,   6,   8,   9,
11, 10,    8,   9,   9,   8,   8,   9,   9,   8,   8,   9,   7,   8,  8,
    7,     7,   10,  10,  7,
7,  10,    8,   9,   5,   8,   8,   7,   7,   10,  8,   7,   7,   8,
    10,    7,   7,   8,   8,   9,
7,  6,     8,   7,   9,   8,   6,   9,   7,   6,   6,   9,   7,   8,  8,
    9,     9,   8,   8,   9,
7,  8,     6,   11,  7,   6,   8,   7,   9,   6,   8,   9,   9,   6,  8,
    7,     9,   4,   8,   9,
9,  8,     8,   9,   7,   8,   8,   7,   7,   10,  8,   7,   7,   10, 6,
    7,     9,   8,   8,   5,
5,  6,     8,   7,   9,   8,   8,   7,   9,   8,   8,   9,   9,   6,  8,
    9,     9,   8,   8,   9,
9,  8,     8,   7,   7,   6,   8,   9,   9,   8,   8,   9,   9,   10,
    10,    7,   9,   10,  10,  9,
7,  10,    10,  9,   7,   8,   8,   7,   9,   8,   10,  7,   7,   10, 8,
    9,     7,   10,  10,  9,
9,  10,    10,  7,   9,   10,  10,  9,   9,   10,  10,  9,   9,   10,
    10,    9,   9,   10,  10,  9,
11, 8,     6,   9,   9,   8,   8,   11,  9,   8,   8,   9,   9,   8,  8,
    9,     11,  8,   8,   9,
9,  8,     8,   9,   11,  8,   8,   9,   9,   6,   8,   11,  9,   8,  8,
    9,     9,   8,   8,   9,
9,  8,     8,   9,   9,   8,   8,   11,  9,   8,   10,  5,   9,   10, 8,
    9,     7,   8,   10,  9,
9,  8,     10,  7,   9,   10,  10,  7,   9,   10,  10,  9,   9,   8,
    10,    9,   5,   10,  8,   9,
9,  10,    8,   7,   7,   10,  10,  9,   9,   10,  8,   9,   7,   10,
    10,    9,   11,  6,   10,  9,
9,  8,     10,  7,   9,   8,   8,   9,   9,   8,   8,   9,   11,  10, 8,
    9,     9,   10,  10,  11,
11, 8,     8,   9,   9,   8,   8,   11,  9,   8,   8,   9,   9,   8,  8,
    9,     11,  8,   8,   9,
9,  6,     8,   9,   7,   10,  10,  7,   7,   10,  10,  9,   7,   10,
    10,    7,   7,   10,  10,  7,
9,  10,    10,  7,   7,   10,  10,  5,   9,   8,   10,  7,   9,   10, 8,
    9,     9,   10,  10,  9,
7,  8,     10,  9,   9,   10,  8,   5,   7,   10,  10,  9,   9,   10, 8,
    9,     9,   8,   10,  11,
9,  8,     10,  9,   9,   10,  8,   9,   9,   10,  8,   9,   9,   8,  6,
    9,     11,  6,   8,   9,
9,  8,     6,   9,   9,   8,   8,   9,   9,   8,   8,   9,   11,  8,  4,
    9,     9,   8,   8,   11,
9,  8,     10,  9,   7,   10,  8,   9,   7,   10,  8,   9,   9,   10,
    10,    9,   9,   10,  8,   7,
5,  10,    10,  9,   9,   8,   10,  9,   7,   10,  8,   9,   7,   8,  8,
    5,     9,   10,  10,  7,
9,  10,    10,  9,   9,   10,  10,  9,   11,  6,   8,   9,   9,   8,  6,
    9,     9,   4,   8,   9,
```

```
9,  8,     8,    9,    11,   8,    8,    9,    9,    8,    8,    11,   11,   10,
    10,    9,    9,    8,    6,    7,
9,  8,     8,    9,    9,    8,    8,    9,    11,   10,   8,    9,    9,    10,
    10,    11,   9,    8,    10,   7,
9,  10,    8,    9,    5,    8,    8,    7,    9,    10,   10,   9,    9,    10,
    10,    9,    7,    10,   10,   9,
9,  10,    10,   7,    7,    10,   10,   3,    7,    10,   10,   7,    7,    10,
    10,    7,    9,    10,   10,   7,
7,  10,    10,   9,    11,   6,    8,    9,    9,    8,    6,    9,    9,    8,   4,
    9,     9,    8,    8,    9,
11, 8,     8,    9,    9,    8,    8,    11,   9,    10,   8,    9,    9,    8,
    10,    11,   9,    8,    10,   9,
9,  10,    8,    9,    9,    6,    8,    9,    9,    8,    10,   9,    9,    8,
    10,    7,    9,    10,   8,    9,
9,  8,     10,   7,    7,    8,    10,   5,    9,    10,   10,   9,    9,    10,
    10,    9,    9,    8,    10,   9,
7,  10,    8,    9,    9,    10,   10,   9,    5,    8,    10,   7,    9,    10,  8,
    7,     9,    10,   10,   9,
11, 8,     8,    9,    9,    8,    8,    11,   9,    8,    8,    9,    9,    8,   6,
    9,     9,    8,    8,    9,
9,  8,     8,    11,   11,   6,    8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     9,    4,    8,    9,
11, 8,     8,    9,    9,    8,    8,    11,   7,    10,   10,   7,    7,    10,
    10,    9,    7,    10,   10,   7,
7,  10,    10,   7,    5,    10,   10,   7,    7,    10,   10,   9,    11,   8,   8,
    9,     9,    8,    8,    11,
9,  6,     8,    9,    9,    8,    8,    9,    11,   8,    8,    9,    9,    8,   8,
    9,     9,    8,    10,   9,
7,  10,    8,    9,    9,    10,   10,   7,    7,    10,   10,   7,    9,    8,   8,
    9,     9,    8,    10,   9,
7,  10,    8,    7,    7,    10,   10,   7,    7,    10,   8,    7,    9,    8,
    10,    7,    7,    8,    10,   7,
9,  8,     8,    7,    9,    6,    8,    9,    9,    8,    8,    7,    9,    8,   8,
    9,     11,   6,    10,   7,
9,  8,     10,   9,    11,   8,    8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     9,    8,    8,    7,
9,  8,     8,    9,    9,    6,    8,    9,    9,    8,    8,    7,    7,    10,  8,
    7,     9,    10,   10,   7,
9,  6,     10,   9,    9,    8,    10,   7,    9,    10,   10,   9,    9,    8,
    10,    9,    9,    8,    8,    7,
9,  10,    8,    9,    7,    10,   10,   9,    9,    10,   10,   9,    9,    10,  8,
    7,     7,    10,   8,    9,
9,  8,     8,    9,    9,    8,    8,    11,   9,    10,   8,    9,    9,    8,   8,
    9,     11,   8,    6,    9,
9,  8,     8,    11,   9,    8,    8,    7,    9,    8,    8,    9,    9,    8,   8,
    9,     9,    8,    8,    9,
9,  8,     8,    9,    9,    6,    6,    7,    9,    10,   10,   9,    9,    8,
    10,    9,    9,    10,   8,    9,
9,  8,     8,    9,    7,    10,   8,    7,    7,    8,    10,   7,    9,    8,   8,
    7,     7,    10,   8,    9,
```

```
7,  10,    10,  7,   7,   10,  10,  7,   9,   10,  10,  7,   7,   8,
    10,    5,   9,   8,   8,   9,
9,  8,     8,   9,   7,   8,   8,   9,   9,   8,   8,   9,   9,   8,   8,
    9,     9,   6,   6,   7,
9,  8,     8,   9,   9,   8,   6,   7,   7,   8,   6,   9,   11,  8,
    10,    9,   9,   8,   10,  9,
11, 8,     8,   9,   9,   8,   10,  7,   9,   10,  8,   5,   7,   10,  8,
    7,     9,   8,   10,  7,
9,  10,    12,  9,   9,   10,  10,  9,   7,   10,  8,   9,   9,   10,
    10,    7,   7,   6,   8,   7,
9,  10,    10,  9,   9,   10,  10,  9,   9,   8,   10,  9,   9,   8,   8,
    9,     11,  10,  6,   7,
9,  8,     6,   7,   9,   8,   8,   9,   9,   8,   10,  11,  9,   8,   8,
    9,     11,  6,   8,   9,
9,  8,     6,   9,   9,   4,   8,   9,   9,   8,   8,   9,   11,  8,   8,
    9,     9,   8,   8,   11,
9,  10,    10,  7,   7,   10,  8,   9,   5,   8,   8,   5,   9,   10,
    10,    7,   9,   8,   10,  9,
7,  10,    8,   9,   9,   10,  8,   9,   7,   8,   10,  9,   9,   10,  8,
    9,     7,   8,   8,   9,
7,  10,    8,   9,   9,   10,  10,  7,   9,   6,   8,   7,   11,  10,  8,
    7,     9,   8,   8,   9,
9,  6,     8,   9,   9,   8,   10,  11,  9,   8,   8,   9,   9,   8,   6,
    9,     9,   8,   8,   9,
9,  8,     8,   9,   9,   8,   8,   7,   9,   6,   8,   9,   9,   8,   8,
    7,     7,   10,  8,   7,
9,  10,    10,  7,   7,   10,  8,   7,   7,   8,   10,  7,   7,   8,
    10,    9,   7,   8,   8,   7,
9,  8,     10,  5,   9,   10,  10,  9,   7,   8,   10,  7,   7,   8,
    10,    5,   9,   8,   10,  7,
9,  10,    8,   9,   11,  8,   8,   9,   9,   6,   8,   11,  9,   8,   8,
    9,     9,   8,   6,   9,
9,  6,     8,   9,   9,   8,   6,   11,  11,  10,  8,   9,   9,   10,
    10,    9,   9,   8,   8,   9,
9,  8,     10,  9,   9,   10,  8,   9,   9,   8,   6,   9,   9,   10,
    10,    9,   9,   8,   10,  9,
9,  10,    8,   9,   9,   8,   8,   9,   7,   10,  8,   7,   7,   8,
    10,    7,   9,   8,   6,   9,
9,  8,     8,   9,   7,   6,   6,   9,   9,   8,   6,   9,   9,   8,   6,
    9,     7,   6,   8,   9,
7,  10,    10,  9,   7,   8,   8,   7,   7,   8,   8,   7,   7,   10,  8,
    7,     7,   10,  8,   7,
5,  8,     10,  7,   7,   10,  8,   7,   7,   8,   10,  7,   5,   8,   8,
    7,     7,   10,  8,   7,
5,  10,    8,   5,   7,   10,  8,   5,   9,   8,   6,   9,   9,   6,   6,
    9,     9,   6,   6,   9,
9,  6,     6,   9,   9,   8,   6,   9,   9,   6,   4,   9,   9,   10,  8,
    7,     9,   8,   10,  9,
7,  8,     8,   9,   9,   8,   8,   9,   7,   10,  10,  9,   9,   10,
    10,    7,   9,   8,   8,   9,
```

```
9,  8,     8,    9,    9,    10,   8,    9,    9,    8,    8,    9,    9,    8,
    10,    9,    9,    8,    10,   11,
7,  8,     8,    7,    7,    8,    10,   7,    5,    8,    8,    7,    7,    10,   8,
    7,     7,    10,   8,    9,
7,  10,    10,   7,    9,    6,    8,    9,    9,    8,    6,    9,    9,    8,    8,
    9,     9,    8,    8,    9,
11, 8,     8,    9,    9,    8,    10,   11,   9,    8,    6,    7,    9,    6,    8,
    9,     7,    6,    6,    9,
9,  6,     6,    9,    9,    8,    8,    9,    9,    8,    8,    9,    7,    10,   8,
    5,     7,    8,    8,    7,
7,  8,     10,   7,    7,    8,    8,    7,    7,    10,   8,    7,    9,    10,
    10,    7,    7,    10,   10,   5,
7,  8,     10,   7,    7,    8,    8,    7,    7,    10,   8,    7,    5,    10,   8,
    7,     5,    8,    8,    5,
9,  8,     8,    9,    9,    6,    8,    9,    9,    8,    6,    9,    9,    6,    6,
    9,     9,    8,    6,    7,
7,  6,     8,    9,    9,    6,    6,    9,    9,    6,    8,    9,    9,    6,    6,
    9,     9,    6,    6,    9,
9,  8,     6,    9,    9,    6,    4,    9,    7,    8,    10,   7,    7,    8,
    10,    7,    9,    8,    10,   7,
7,  8,     8,    7,    7,    10,   8,    5,    7,    8,    10,   7,    9,    10,   8,
    9,     9,    8,    10,   11,
9,  10,    8,    9,    9,    8,    8,    9,    9,    10,   10,   9,    9,    8,    6,
    9,     9,    10,   8,    7,
9,  8,     10,   9,    9,    8,    10,   9,    9,    8,    10,   9,    7,    10,   8,
    7,     9,    8,    10,   7,
11, 6,     6,    9,    9,    6,    6,    11,   9,    6,    6,    9,    9,    6,    6,
    9,     9,    6,    6,    9,
9,  6,     6,    9,    7,    10,   10,   7,    7,    8,    10,   7,    5,    8,
    10,    7,    7,    8,    8,    7,
5,  10,    8,    5,    7,    8,    8,    5,    7,    10,   8,    5,    7,    8,
    10,    7,    7,    8,    10,   7,
7,  8,     8,    7,    5,    10,   8,    7,    5,    10,   8,    5,    9,    8,    6,
    7,     9,    6,    8,    9,
9,  6,     8,    9,    9,    6,    8,    9,    9,    8,    6,    7,    9,    6,    8,
    9,     9,    10,   8,    9,
9,  10,    10,   9,    7,    8,    8,    9,    9,    10,   8,    9,    7,    10,   8,
    9,     7,    8,    10,   7,
9,  8,     8,    9,    9,    10,   8,    9,    9,    8,    8,    9,    9,    8,
    10,    9,    11,   8,    8,    9,
9,  8,     10,   9,    9,    8,    10,   9,    9,    10,   8,    9,    7,    10,
    10,    9,    9,    10,   8,    7,
9,  10,    10,   7,    5,    8,    10,   9,    9,    8,    8,    9,    9,    8,    8,
    9,     7,    8,    8,    9,
9,  8,     8,    9,    9,    8,    6,    9,    9,    8,    6,    7,    7,    8,    8,
    9,     7,    8,    8,    7,
7,  8,     6,    9,    7,    8,    10,   7,    5,    8,    10,   5,    7,    8,    6,
    7,     7,    8,    6,    9,
9,  8,     8,    7,    7,    8,    8,    9,    11,   6,    6,    9,    9,    8,    6,
    7,     9,    8,    4,    9,
```

```
9,  8,     6,    9,    7,    6,    8,    7,    9,    8,    4,    9,    9,    8,   6,
    9,     7,    8,    4,    9,
7,  6,     8,    7,    5,    8,    8,    7,    7,    10,   10,   5,    7,    10,  8,
    7,     7,    8,    8,    7,
7,  10,    8,    7,    3,    8,    10,   7,    7,    6,    10,   11,   9,    8,   8,
    9,     9,    8,    8,    9,
11, 10,    6,    7,    9,    8,    8,    9,    9,    8,    8,    9,    9,    10,  8,
    7,     7,    10,   10,   9,
7,  8,     10,   7,    7,    10,   10,   7,    9,    8,    10,   9,    9,    10,  8,
    9,     7,    8,    8,    7,
9,  6,     8,    7,    7,    8,    6,    9,    9,    6,    6,    9,    7,    8,   8,
    9,     9,    8,    8,    9,
7,  10,    8,    5,    7,    8,    8,    7,    7,    8,    8,    7,    7,    8,
    10,    7,    7,    10,   10,   7,
9,  8,     10,   7,    7,    6,    8,    9,    5,    10,   8,    5,    7,    10,  6,
    9,     7,    8,    8,    7,
7,  8,     8,    7,    7,    8,    8,    7,    7,    4,    8,    11,   7,    8,   8,
    7,     9,    8,    6,    9,
9,  8,     6,    7,    9,    8,    6,    9,    9,    6,    8,    9,    9,    6,   8,
    9,     9,    8,    6,    9,
7,  8,     6,    9,    11,   8,    6,    9,    7,    6,    6,    7,    9,    6,   6,
    7,     7,    8,    8,    9,
7,  8,     8,    7,    5,    8,    8,    9,    7,    8,    10,   7,    7,    8,
    10,    7,    7,    6,    6,    5,
7,  10,    8,    7,    7,    8,    10,   7,    7,    8,    8,    7,    7,    8,
    10,    7,    7,    10,   10,   7,
5,  8,     8,    7,    7,    8,    6,    7,    9,    8,    8,    7,    9,    6,   8,
    9,     9,    8,    8,    9,
9,  8,     6,    7,    9,    6,    8,    7,    9,    8,    10,   9,    7,    10,  8,
    9,     7,    10,   8,    9,
9,  10,    10,   9,    9,    10,   8,    7,    5,    10,   10,   9,    9,    6,
    10,    11,   7,    8,    8,    7,
9,  8,     8,    9,    11,   10,   8,    9,    9,    8,    6,    9,    9,    8,   8,
    9,     7,    10,   8,    7,
7,  10,    10,   7,    7,    10,   8,    5,    5,    8,    8,    7,    5,    8,   8,
    7,     7,    8,    10,   5,
7,  8,     8,    9,    9,    8,    8,    9,    9,    6,    6,    9,    9,    6,   4,
    7,     7,    8,    6,    9,
7,  4,     8,    7,    9,    6,    6,    11,   7,    8,    6,    9,    9,    8,   8,
    9,     9,    6,    6,    7,
7,  6,     6,    9,    9,    8,    6,    7,    5,    6,    8,    9,    7,    10,  8,
    7,     7,    10,   8,    9,
7,  8,     8,    5,    7,    8,    8,    7,    7,    6,    8,    7,    9,    8,   8,
    9,     9,    8,    8,    9,
9,  8,     8,    9,    9,    8,    6,    9,    7,    8,    8,    7,    9,    8,   6,
    9,     9,    10,   10,   7,
7,  8,     10,   9,    9,    10,   10,   7,    7,    8,    10,   7,    9,    8,
    10,    7,    9,    10,   8,    9,
9,  8,     10,   7,    9,    8,    8,    9,    9,    10,   10,   7,    9,    10,
    10,    9,    9,    10,   10,   5,
```

```
7,  8,     10,  9,   9,   8,   8,   7,   9,   8,   8,   9,   9,   8,   8,
    9,     9,   8,   8,   9,
9,  8,     6,   9,   9,   8,   6,   7,   5,   8,   8,   7,   7,   8,   6,
    7,     9,   10,  8,   5,
7,  8,     10,  9,   7,   8,   8,   7,   7,   6,   8,   7,   7,   8,   6,
    9,     9,   8,   8,   7,
7,  6,     8,   9,   9,   8,   8,   9,   9,   8,   6,   9,   7,   6,   8,
    7,     9,   6,   6,   9,
9,  6,     6,   9,   11,  8,   6,   7,   7,   8,   8,   9,   7,   6,   6,
    9,     9,   8,   6,   7,
7,  10,    8,   7,   7,   10,  10,  7,   7,   8,   8,   7,   7,   8,   8,
    7,     7,   10,  10,  5,
7,  8,     8,   7,   7,   8,   10,  9,   11,  6,   6,   9,   11,  8,
    10,    7,   9,   8,   8,   9,
9,  8,     8,   9,   9,   8,   8,   9,   9,   10,  8,   7,   7,   10,
    10,    9,   7,   8,   10,  7,
7,  10,    10,  7,   9,   8,   10,  9,   9,   10,  8,   9,   7,   8,   8,
    9,     9,   6,   4,   7,
11, 8,     8,   7,   7,   6,   8,   9,   9,   8,   6,   9,   9,   6,   8,
    9,     7,   8,   8,   7,
7,  10,    10,  7,   5,   8,   8,   7,   7,   8,   8,   7,   7,   10,
    10,    9,   7,   8,   10,  7,
7,  8,     8,   7,   7,   8,   6,   5,   9,   10,  8,   7,   5,   6,
    10,    9,   7,   8,   8,   7,
7,  8,     8,   7,   7,   8,   8,   7,   9,   6,   8,   7,   7,   8,   6,
    9,     9,   6,   6,   9,
7,  8,     8,   9,   9,   8,   8,   9,   7,   8,   8,   9,   9,   4,   8,
    9,     9,   6,   6,   7,
9,  6,     8,   9,   7,   8,   6,   7,   9,   4,   8,   7,   7,   8,   8,
    5,     7,   10,  8,   7,
9,  8,     8,   7,   7,   8,   8,   9,   7,   8,   10,  7,   7,   6,   6,
    5,     5,   10,  8,   7,
7,  8,     8,   5,   7,   10,  8,   7,   7,   8,   10,  7,   7,   10,  8,
    3,     7,   8,   10,  7,
7,  8,     6,   7,   11,  6,   8,   7,   9,   8,   6,   9,   9,   8,   8,
    9,     9,   8,   6,   9,
7,  8,     4,   9,   9,   8,   10,  7,   9,   10,  8,   9,   9,   10,
    10,    9,   7,   8,   10,  9,
9,  10,    8,   5,   7,   10,  10,  9,   9,   8,   8,   9,   9,   6,   8,
    9,     9,   8,   8,   9,
9,  8,     8,   9,   7,   8,   8,   9,   7,   8,   6,   9,   7,   10,  8,
    7,     5,   8,   10,  7,
7,  10,    8,   7,   7,   8,   10,  5,   5,   8,   8,   7,   7,   8,
    10,    5,   9,   6,   8,   7,
11, 6,     6,   9,   9,   8,   8,   9,   9,   6,   8,   9,   7,   6,   6,
    9,     7,   6,   6,   7,
9,  8,     6,   9,   9,   6,   8,   7,   9,   8,   8,   9,   7,   4,   6,
    9,     7,   8,   4,   7,
9,  6,     8,   7,   7,   8,   8,   7,   7,   10,  8,   7,   9,   8,   8,
    7,     7,   6,   8,   9,
```

```
5,  8,  10,  7,  5,  8,  6,  7,  9,  8,  10,  9,  11,  8,  6,
    7,  11,  8,  10,  9,
7,  8,  8,  9,  9,  8,  6,  9,  9,  8,  8,  9,  9,  10,
    10,  9,  7,  10,  10,  9,
7,  10,  8,  7,  7,  10,  8,  7,  9,  8,  10,  9,  7,  10,  8,
    9,  9,  6,  8,  9,
9,  8,  6,  11,  9,  8,  8,  9,  9,  8,  8,  9,  11,  8,  8,
    9,  9,  4,  8,  11,
9,  8,  10,  7,  7,  10,  8,  9,  7,  8,  10,  7,  7,  10,
    10,  7,  9,  10,  8,  7,
7,  10,  10,  5,  9,  8,  8,  7,  7,  6,  6,  9,  9,  8,  8,
    5,  9,  8,  8,  7,
9,  6,  8,  5,  7,  6,  6,  9,  7,  8,  10,  7,  7,  8,  8,
    7,  7,  6,  8,  9,
7,  10,  8,  7,  7,  8,  8,  5,  9,  8,  6,  5,  7,  8,
    10,  7,  7,  8,  8,  7,
7,  8,  10,  5,  7,  8,  8,  9,  7,  6,  6,  9,  7,  8,  8,
    5,  9,  8,  8,  9,
9,  8,  8,  9,  9,  6,  8,  9,  9,  8,  6,  7,  9,  8,  6,
    9,  7,  4,  8,  9,
5,  8,  12,  9,  9,  8,  8,  9,  9,  10,  10,  7,  9,  10,  8,
    7,  9,  10,  10,  7,
7,  10,  10,  9,  11,  8,  6,  9,  9,  8,  8,  9,  9,  8,  8,
    9,  9,  8,  8,  9,
11,  8,  8,  9,  9,  10,  8,  11,  5,  8,  10,  7,  9,  6,  6,
    7,  9,  8,  10,  5,
7,  8,  8,  7,  7,  8,  8,  7,  7,  8,  8,  7,  9,  8,  6,
    7,  9,  6,  8,  9,
7,  6,  6,  9,  9,  6,  6,  9,  9,  8,  8,  9,  9,  8,  8,
    9,  9,  6,  6,  9,
7,  8,  6,  9,  9,  6,  8,  7,  7,  8,  8,  7,  9,  6,  8,
    9,  9,  8,  8,  9,
5,  6,  10,  9,  7,  8,  8,  7,  7,  8,  8,  7,  9,  10,  6,
    5,  7,  8,  8,  7,
7,  8,  8,  7,  5,  8,  10,  9,  7,  6,  6,  7,  7,  8,  8,
    7,  9,  10,  8,  7,
7,  8,  8,  5,  7,  8,  8,  7,  9,  6,  6,  7,  9,  8,  6,
    9,  7,  8,  8,  9,
9,  8,  8,  9,  9,  8,  6,  7,  7,  8,  6,  9,  9,  8,  8,
    9,  9,  6,  8,  9,
9,  6,  8,  7,  9,  8,  8,  9,  9,  8,  6,  7,  9,  4,  8,
    9,  7,  8,  10,  7,
7,  8,  8,  7,  7,  6,  10,  7,  7,  8,  8,  9,  7,  8,  8,
    9,  5,  8,  6,  5,
11,  8,  8,  9,  9,  8,  8,  11,  9,  8,  8,  9,  9,  8,  8,
    9,  11,  8,  8,  9,
9,  6,  8,  9,  7,  10,  10,  9,  7,  8,  10,  7,  7,  8,
    10,  7,  9,  10,  8,  7,
7,  8,  8,  7,  7,  10,  8,  7,  9,  8,  6,  9,  7,  6,  8,
    9,  9,  8,  6,  9,
```

```
7,  6,    6,   9,   9,   8,   6,   9,   9,   8,   8,   9,   5,   6,
    10,   7,   9,   8,   8,   7,
9,  8,    10,  7,   7,   8,   6,   7,   7,   8,   8,   7,   5,   8,   8,
    7,    7,   8,   10,  7,
7,  8,    8,   7,   7,   8,   8,   9,   7,   10,  8,   5,   7,   6,   6,
    7,    9,   8,   8,   5,
9,  6,    8,   9,   9,   8,   8,   9,   7,   8,   8,   7,   7,   8,   6,
    5,    9,   6,   8,   7,
5,  6,    6,   9,   7,   10,  10,  7,   9,   8,   10,  7,   9,   8,
    10,   7,   7,   10,  8,   7,
7,  8,    8,   7,   7,   10,  8,   7,   11,  8,   8,   9,   9,   8,   8,
    11,   9,   8,   6,   9,
9,  6,    8,   9,   9,   6,   8,   9,   9,   8,   6,   11,  11,  8,   8,
    9,    9,   8,   8,   11,
9,  8,    8,   9,   9,   8,   8,   9,   11,  8,   6,   9,   9,   8,   8,
    9,    9,   8,   10,  7,
9,  10,   8,   9,   7,   8,   10,  9,   9,   10,  10,  9,   9,   10,
    10,   7,   7,   8,   8,   9,
7,  10,   8,   7,   7,   10,  10,  7,   9,   10,  8,   7,   7,   8,
    10,   9,   7,   8,   10,  7,
7,  8,    8,   7,   9,   8,   6,   9,   9,   8,   8,   9,   9,   8,   8,
    9,    9,   8,   8,   9,
9,  8,    8,   7,   9,   8,   6,   7,   9,   6,   8,   9,   9,   8,   8,
    7,    11,  10,  6,   9,
9,  8,    10,  11,  9,   8,   8,   9,   7,   8,   8,   9,   7,   10,  8,
    9,    7,   10,  10,  7,
9,  10,   8,   9,   9,   10,  10,  9,   9,   10,  6,   9,   7,   8,
    10,   9,   9,   8,   8,   9,
7,  10,   8,   9,   9,   8,   10,  7,   7,   10,  10,  7,   9,   8,
    10,   9,   9,   10,  10,  9,
9,  8,    8,   9,   9,   8,   8,   11,  9,   6,   8,   9,   9,   8,   8,
    9,    11,  8,   10,  9,
9,  8,    8,   11,  9,   8,   8,   9,   9,   8,   6,   7,   11,  10,  8,
    9,    7,   6,   10,  11,
9,  8,    8,   9,   9,   8,   8,   9,   9,   10,  8,   7,   9,   8,
    10,   9,   7,   8,   8,   9,
9,  8,    8,   9,   7,   10,  10,  9,   9,   10,  10,  7,   9,   8,
    10,   9,   7,   10,  8,   5,
9,  12,   8,   9,   7,   8,   10,  9,   9,   10,  10,  7,   7,   10,
    10,   9,   9,   6,   8,   11,
7,  10,   8,   7,   9,   10,  6,   11,  9,   8,   8,   9,   9,   8,   8,
    9,    9,   8,   8,   9,
9,  8,    8,   9,   7,   8,   8,   9,   9,   8,   8,   9,   9,   8,   8,
    9,    9,   6,   8,   9,
9,  6,    8,   7,   9,   8,   8,   7,   7,   10,  8,   9,   7,   10,
    10,   7,   7,   10,  10,  7,
9,  10,   10,  7,   7,   8,   10,  5,   7,   10,  8,   9,   9,   10,
    10,   7,   9,   10,  10,  9,
7,  8,    10,  9,   9,   10,  6,   7,   9,   8,   10,  9,   9,   8,   6,
    9,    9,   8,   8,   9,
```

```
9,  8,  8,  9,  9,  8,  8,  9,  9,  8,  8,  9,  7,  8,  6,
    7,  11, 6,  8,  9,
9,  8,  6,  9,  9,  8,  8,  9,  9,  8,  8,  9,  11, 8,  4,
    9,  9,  8,  8,  11,
9,  8,  10, 9,  5,  10, 10, 9,  7,  10, 8,  7,  7,  10,
    10, 9,  9,  8,  8,  7,
5,  10, 8,  9,  9,  10, 10, 9,  9,  8,  10, 9,  9,  8,  8,
    7,  9,  8,  8,  7,
7,  10, 10, 9,  9,  8,  10, 7,  9,  8,  8,  9,  9,  8,  8,
    9,  9,  8,  8,  9,
7,  8,  8,  9,  9,  6,  8,  9,  9,  6,  8,  7,  9,  8,  8,
    11, 9,  10, 6,  7,
9,  10, 8,  11, 9,  6,  8,  9,  9,  8,  8,  7,  9,  8,  8,
    9,  7,  10, 8,  9,
7,  10, 10, 7,  7,  10, 8,  9,  7,  8,  10, 7,  7,  8,
    10, 7,  7,  8,  8,  7,
9,  10, 10, 7,  7,  10, 8,  9,  9,  10, 10, 9,  5,  8,
    10, 7,  9,  8,  8,  5,
7,  10, 8,  9,  11, 8,  8,  9,  9,  6,  8,  11, 9,  8,  8,
    9,  9,  8,  6,  9,
9,  6,  8,  9,  9,  8,  6,  11, 9,  10, 10, 7,  7,  10,
    10, 7,  7,  10, 10, 7,
7,  10, 10, 7,  9,  10, 10, 7,  7,  10, 10, 5,  11, 8,
    10, 9,  9,  8,  8,  9,
9,  8,  8,  9,  9,  8,  8,  9,  11, 8,  8,  9,  9,  6,  8,
    11, 11, 8,  8,  9,
9,  8,  8,  11, 9,  6,  8,  9,  9,  8,  8,  9,  11, 6,  6,
    9,  9,  8,  6,  9,
9,  10, 10, 9,  7,  10, 10, 9,  9,  10, 10, 7,  7,  10,
    10, 7,  9,  8,  8,  7,
7,  8,  8,  9,  9,  10, 10, 7,  9,  10, 10, 9,  7,  8,
    10, 7,  9,  10, 10, 7,
9,  8,  10, 7,  7,  8,  8,  9,  11, 10, 8,  9,  9,  8,  8,
    11, 9,  8,  10, 9,
9,  10, 8,  9,  11, 8,  8,  9,  9,  8,  8,  7,  11, 8,  6,
    9,  9,  8,  8,  9,
9,  8,  8,  9,  9,  8,  8,  9,  11, 8,  8,  9,  9,  10, 8,
    11, 7,  10, 10, 9,
9,  10, 10, 7,  9,  10, 10, 9,  9,  10, 10, 9,  9,  10,
    10, 9,  9,  10, 10, 11,
9,  10, 8,  7,  7,  10, 10, 9,  7,  8,  10, 7,  7,  10,
    10, 7,  9,  8,  10, 9,
9,  10, 8,  9,  9,  8,  10, 9,  9,  8,  8,  9,  9,  8,  8,
    9,  9,  8,  8,  9,
9,  8,  8,  9,  9,  10, 10, 11, 9,  8,  8,  9,  9,  8,  8,
    11, 9,  6,  8,  9,
9,  8,  8,  9,  11, 8,  10, 9,  9,  8,  8,  11, 9,  10, 8,
    7,  7,  10, 10, 9,
7,  8,  10, 7,  7,  10, 10, 7,  9,  8,  10, 9,  9,  10, 8,
    9,  9,  10, 8,  7,
```

```
7,  10,    10,  9,   7,   10,  10,  9,   7,   10,  10,  9,   9,   8,  8,
    7,     7,   10,  8,   9,
9,  8,     8,   9,   9,   8,   8,   11,  9,   10,  8,   9,   9,   8,  8,
    9,     11,  8,   6,   9,
9,  8,     8,   11,  11,  8,   8,   9,   9,   8,   10,  11,  9,   8,
    10,    9,   9,   10,  8,   9,
11, 8,     8,   9,   9,   8,   8,   7,   9,   10,  10,  9,   7,   10,
    10,    9,   9,   10,  10,  7,
7,  10,    10,  7,   9,   8,   8,   7,   7,   8,   8,   9,   7,   10,
    10,    7,   7,   10,  10,  9,
7,  10,    10,  7,   7,   10,  10,  7,   9,   10,  10,  7,   7,   10,
    10,    5,   11,  8,   6,   9,
9,  8,     8,   11,  9,   8,   8,   9,   9,   8,   8,   9,   11,  6,  8,
    9,     9,   6,   6,   9,
9,  8,     10,  9,   9,   8,   8,   9,   9,   8,   8,   9,   9,   8,  8,
    9,     9,   10,  8,   9,
9,  10,    8,   11,  9,   10,  8,   7,   7,   10,  10,  9,   7,   10,
    10,    9,   7,   10,  10,  9,
9,  8,     8,   7,   7,   10,  8,   9,   9,   10,  10,  7,   9,   10,
    10,    9,   7,   8,   10,  7,
9,  10,    10,  7,   9,   8,   10,  7,   7,   8,   8,   9,   11,  8,  8,
    9,     9,   8,   8,   11,
9,  6,     8,   9,   9,   8,   8,   9,   11,  6,   6,   9,   9,   8,  6,
    9,     11,  8,   6,   9,
9,  8,     8,   11,  9,   8,   8,   9,   9,   8,   8,   9,   11,  6,  8,
    9,     9,   6,   6,   9,
7,  10,    10,  7,   7,   10,  10,  7,   7,   10,  10,  7,   7,   10,
    10,    7,   7,   10,  10,  7,
7,  10,    10,  7};
char table7[] =
{1, 2,     4,   5,   5,   4,   2,   3,   5,   6,   6,   5,   5,   6,  4,
    5,     5,   6,   6,   5,
5,  4,     6,   5,   5,   6,   4,   5,   5,   6,   6,   7,   3,   4,  4,
    5,     3,   4,   2,   3,
5,  6,     4,   5,   3,   4,   6,   7,   5,   4,   6,   5,   5,   6,  6,
    5,     7,   6,   6,   7,
7,  6,     6,   5,   7,   6,   6,   7,   5,   6,   6,   7,   5,   4,  6,
    5,     5,   6,   6,   5,
5,  6,     4,   5,   5,   6,   4,   5,   5,   6,   4,   5,   3,   4,  4,
    5,     5,   4,   6,   5,
5,  6,     6,   5,   5,   4,   6,   5,   5,   4,   4,   3,   5,   4,  6,
    5,     5,   4,   6,   5,
5,  6,     6,   5,   5,   4,   6,   7,   5,   4,   4,   3,   5,   4,  4,
    3,     7,   6,   4,   3,
3,  2,     6,   5,   5,   6,   4,   3,   3,   2,   6,   7,   5,   4,  4,
    3,     5,   4,   6,   5,
5,  6,     4,   3,   5,   4,   6,   7,   7,   6,   6,   5,   5,   6,  6,
    5,     5,   6,   4,   5,
5,  4,     6,   5,   3,   4,   6,   5,   5,   6,   4,   5,   5,   4,  4,
    5,     3,   4,   6,   5,
```

```
5,  6,     4,    5,    5,    6,    6,    5,    5,    4,    4,    5,    5,    6,   6,
    5,     7,    6,    6,    7,
5,  6,     6,    5,    5,    6,    4,    5,    7,    6,    6,    7,    5,    6,   6,
    7,     5,    6,    6,    7,
7,  6,     6,    7,    5,    6,    6,    7,    7,    6,    6,    5,    5,    6,   6,
    5,     5,    4,    6,    7,
7,  6,     6,    5,    5,    6,    6,    5,    5,    4,    4,    5,    5,    4,   4,
    3,     5,    6,    6,    5,
5,  4,     6,    5,    5,    4,    6,    5,    5,    4,    4,    3,    5,    4,   6,
    5,     5,    4,    6,    5,
5,  4,     6,    5,    5,    6,    6,    5,    5,    6,    4,    5,    7,    6,   6,
    5,     7,    6,    6,    7,
5,  6,     6,    7,    7,    6,    6,    7,    5,    6,    6,    5,    5,    6,   4,
    5,     5,    6,    2,    3,
3,  4,     6,    7,    3,    4,    4,    5,    3,    4,    4,    5,    7,    6,   4,
    5,     5,    6,    6,    5,
5,  6,     4,    5,    5,    6,    4,    5,    3,    4,    4,    5,    5,    6,   4,
    5,     5,    6,    6,    5,
5,  6,     4,    5,    7,    6,    6,    5,    5,    6,    6,    5,    5,    4,   6,
    5,     5,    6,    6,    5,
5,  4,     4,    5,    5,    6,    4,    3,    7,    6,    6,    5,    7,    6,   6,
    5,     5,    4,    6,    5,
7,  6,     6,    5,    7,    6,    6,    7,    7,    6,    6,    5,    7,    6,   6,
    5,     7,    6,    6,    5,
7,  6,     6,    7,    5,    4,    6,    5,    5,    6,    6,    5,    7,    6,   6,
    7,     5,    6,    6,    5,
5,  4,     4,    5,    5,    6,    6,    5,    5,    4,    6,    5,    5,    6,   4,
    3,     5,    4,    4,    5,
5,  6,     4,    5,    5,    6,    4,    5,    5,    6,    6,    5,    3,    4,   4,
    5,     5,    4,    4,    5,
5,  6,     6,    5,    5,    6,    6,    7,    7,    6,    4,    5,    5,    6,   6,
    5,     5,    6,    6,    7,
7,  6,     6,    5,    7,    6,    6,    7,    7,    6,    6,    7,    5,    6,   6,
    5,     7,    6,    6,    7,
5,  6,     6,    7,    7,    6,    4,    5,    5,    6,    6,    5,    7,    6,   8,
    7,     7,    8,    6,    5,
7,  6,     6,    7,    7,    6,    8,    7,    7,    8,    6,    7,    7,    8,   8,
    7};

// Use very ugly and unsafe macro to swap items instead of classic routine
with
//   pointers for the sole reason of brevity
//#define SWAP(a,b) TEMP=a;a=b;b=TEMP;
// number 65='A' is often subtracted to convert char ABC... to number
0,1,2,...
#define CHAROFFSET 65

void updateColorConfigOld()
{
    int i, j;
```

```
    for(i=0;i<6;i++)
      for(j=0;j<9;j++)
            colorConfigOld[i][j] = colorConfig[i][j];
}

void reverseColorConfigOld()
{
    int i, j;
    for(i=0;i<6;i++)
      for(j=0;j<9;j++)
            colorConfig[i][j] = colorConfigOld[i][j];
}

void updateCube()
{
    int i, j;
    for(i=0;i<6;i++)
      for(j=0;j<9;j++)
            SET_COLOR(i,j,colorConfig[i][j]);
}

void U1()
{
    colorConfig[2][0] = colorConfigOld[2][6];
    colorConfig[2][6] = colorConfigOld[2][8];
    colorConfig[2][8] = colorConfigOld[2][2];
    colorConfig[2][2] = colorConfigOld[2][0];
    colorConfig[2][1] = colorConfigOld[2][3];
    colorConfig[2][3] = colorConfigOld[2][7];
    colorConfig[2][7] = colorConfigOld[2][5];
    colorConfig[2][5] = colorConfigOld[2][1];
    colorConfig[4][0] = colorConfigOld[0][0];
    colorConfig[0][0] = colorConfigOld[1][0];
    colorConfig[1][0] = colorConfigOld[3][0];
    colorConfig[3][0] = colorConfigOld[4][0];
    colorConfig[4][1] = colorConfigOld[0][1];
    colorConfig[0][1] = colorConfigOld[1][1];
    colorConfig[1][1] = colorConfigOld[3][1];
    colorConfig[3][1] = colorConfigOld[4][1];
    colorConfig[4][2] = colorConfigOld[0][2];
    colorConfig[0][2] = colorConfigOld[1][2];
    colorConfig[1][2] = colorConfigOld[3][2];
    colorConfig[3][2] = colorConfigOld[4][2];
}

void U2()
{
    colorConfig[2][0] = colorConfigOld[2][8];
    colorConfig[2][6] = colorConfigOld[2][2];
    colorConfig[2][8] = colorConfigOld[2][0];
```

```
    colorConfig[2][2] = colorConfigOld[2][6];
    colorConfig[2][1] = colorConfigOld[2][7];
    colorConfig[2][3] = colorConfigOld[2][5];
    colorConfig[2][7] = colorConfigOld[2][1];
    colorConfig[2][5] = colorConfigOld[2][3];
    colorConfig[4][0] = colorConfigOld[1][0];
    colorConfig[0][0] = colorConfigOld[3][0];
    colorConfig[1][0] = colorConfigOld[4][0];
    colorConfig[3][0] = colorConfigOld[0][0];
    colorConfig[4][1] = colorConfigOld[1][1];
    colorConfig[0][1] = colorConfigOld[3][1];
    colorConfig[1][1] = colorConfigOld[4][1];
    colorConfig[3][1] = colorConfigOld[0][1];
    colorConfig[4][2] = colorConfigOld[1][2];
    colorConfig[0][2] = colorConfigOld[3][2];
    colorConfig[1][2] = colorConfigOld[4][2];
    colorConfig[3][2] = colorConfigOld[0][2];
}

void U3()
{
    colorConfig[2][0] = colorConfigOld[2][2];
    colorConfig[2][6] = colorConfigOld[2][0];
    colorConfig[2][8] = colorConfigOld[2][6];
    colorConfig[2][2] = colorConfigOld[2][8];
    colorConfig[2][1] = colorConfigOld[2][5];
    colorConfig[2][3] = colorConfigOld[2][1];
    colorConfig[2][7] = colorConfigOld[2][3];
    colorConfig[2][5] = colorConfigOld[2][7];
    colorConfig[4][0] = colorConfigOld[3][0];
    colorConfig[0][0] = colorConfigOld[4][0];
    colorConfig[1][0] = colorConfigOld[0][0];
    colorConfig[3][0] = colorConfigOld[1][0];
    colorConfig[4][1] = colorConfigOld[3][1];
    colorConfig[0][1] = colorConfigOld[4][1];
    colorConfig[1][1] = colorConfigOld[0][1];
    colorConfig[3][1] = colorConfigOld[1][1];
    colorConfig[4][2] = colorConfigOld[3][2];
    colorConfig[0][2] = colorConfigOld[4][2];
    colorConfig[1][2] = colorConfigOld[0][2];
    colorConfig[3][2] = colorConfigOld[1][2];
}

void D1()
{
    colorConfig[5][0] = colorConfigOld[5][6];
    colorConfig[5][6] = colorConfigOld[5][8];
    colorConfig[5][8] = colorConfigOld[5][2];
    colorConfig[5][2] = colorConfigOld[5][0];
    colorConfig[5][1] = colorConfigOld[5][3];
```

```
    colorConfig[5][3] = colorConfigOld[5][7];
    colorConfig[5][7] = colorConfigOld[5][5];
    colorConfig[5][5] = colorConfigOld[5][1];
    colorConfig[4][6] = colorConfigOld[3][6];
    colorConfig[0][6] = colorConfigOld[4][6];
    colorConfig[1][6] = colorConfigOld[0][6];
    colorConfig[3][6] = colorConfigOld[1][6];
    colorConfig[4][7] = colorConfigOld[3][7];
    colorConfig[0][7] = colorConfigOld[4][7];
    colorConfig[1][7] = colorConfigOld[0][7];
    colorConfig[3][7] = colorConfigOld[1][7];
    colorConfig[4][8] = colorConfigOld[3][8];
    colorConfig[0][8] = colorConfigOld[4][8];
    colorConfig[1][8] = colorConfigOld[0][8];
    colorConfig[3][8] = colorConfigOld[1][8];
}

void D2()
{
    colorConfig[5][0] = colorConfigOld[5][8];
    colorConfig[5][6] = colorConfigOld[5][2];
    colorConfig[5][8] = colorConfigOld[5][0];
    colorConfig[5][2] = colorConfigOld[5][6];
    colorConfig[5][1] = colorConfigOld[5][7];
    colorConfig[5][3] = colorConfigOld[5][5];
    colorConfig[5][7] = colorConfigOld[5][1];
    colorConfig[5][5] = colorConfigOld[5][3];
    colorConfig[4][6] = colorConfigOld[1][6];
    colorConfig[0][6] = colorConfigOld[3][6];
    colorConfig[1][6] = colorConfigOld[4][6];
    colorConfig[3][6] = colorConfigOld[0][6];
    colorConfig[4][7] = colorConfigOld[1][7];
    colorConfig[0][7] = colorConfigOld[3][7];
    colorConfig[1][7] = colorConfigOld[4][7];
    colorConfig[3][7] = colorConfigOld[0][7];
    colorConfig[4][8] = colorConfigOld[1][8];
    colorConfig[0][8] = colorConfigOld[3][8];
    colorConfig[1][8] = colorConfigOld[4][8];
    colorConfig[3][8] = colorConfigOld[0][8];
}

void D3()
{
    colorConfig[5][0] = colorConfigOld[5][2];
    colorConfig[5][6] = colorConfigOld[5][0];
    colorConfig[5][8] = colorConfigOld[5][6];
    colorConfig[5][2] = colorConfigOld[5][8];
    colorConfig[5][1] = colorConfigOld[5][5];
    colorConfig[5][3] = colorConfigOld[5][1];
    colorConfig[5][7] = colorConfigOld[5][3];
```

```
    colorConfig[5][5] = colorConfigOld[5][7];
    colorConfig[4][6] = colorConfigOld[0][6];
    colorConfig[0][6] = colorConfigOld[1][6];
    colorConfig[1][6] = colorConfigOld[3][6];
    colorConfig[3][6] = colorConfigOld[4][6];
    colorConfig[4][7] = colorConfigOld[0][7];
    colorConfig[0][7] = colorConfigOld[1][7];
    colorConfig[1][7] = colorConfigOld[3][7];
    colorConfig[3][7] = colorConfigOld[4][7];
    colorConfig[4][8] = colorConfigOld[0][8];
    colorConfig[0][8] = colorConfigOld[1][8];
    colorConfig[1][8] = colorConfigOld[3][8];
    colorConfig[3][8] = colorConfigOld[4][8];
}

void F1()
{
    colorConfig[0][0] = colorConfigOld[0][6];
    colorConfig[0][6] = colorConfigOld[0][8];
    colorConfig[0][8] = colorConfigOld[0][2];
    colorConfig[0][2] = colorConfigOld[0][0];
    colorConfig[0][1] = colorConfigOld[0][3];
    colorConfig[0][3] = colorConfigOld[0][7];
    colorConfig[0][7] = colorConfigOld[0][5];
    colorConfig[0][5] = colorConfigOld[0][1];
    colorConfig[2][6] = colorConfigOld[4][8];
    colorConfig[1][0] = colorConfigOld[2][6];
    colorConfig[5][0] = colorConfigOld[1][0];
    colorConfig[4][8] = colorConfigOld[5][0];
    colorConfig[2][7] = colorConfigOld[4][5];
    colorConfig[1][3] = colorConfigOld[2][7];
    colorConfig[5][3] = colorConfigOld[1][3];
    colorConfig[4][5] = colorConfigOld[5][3];
    colorConfig[2][8] = colorConfigOld[4][2];
    colorConfig[1][6] = colorConfigOld[2][8];
    colorConfig[5][6] = colorConfigOld[1][6];
    colorConfig[4][2] = colorConfigOld[5][6];
}

void F2()
{
    colorConfig[0][0] = colorConfigOld[0][8];
    colorConfig[0][6] = colorConfigOld[0][2];
    colorConfig[0][8] = colorConfigOld[0][0];
    colorConfig[0][2] = colorConfigOld[0][6];
    colorConfig[0][1] = colorConfigOld[0][7];
    colorConfig[0][3] = colorConfigOld[0][5];
    colorConfig[0][7] = colorConfigOld[0][1];
    colorConfig[0][5] = colorConfigOld[0][3];
    colorConfig[2][6] = colorConfigOld[5][0];
```

```
    colorConfig[1][0] = colorConfigOld[4][8];
    colorConfig[5][0] = colorConfigOld[2][6];
    colorConfig[4][8] = colorConfigOld[1][0];
    colorConfig[2][7] = colorConfigOld[5][3];
    colorConfig[1][3] = colorConfigOld[4][5];
    colorConfig[5][3] = colorConfigOld[2][7];
    colorConfig[4][5] = colorConfigOld[1][3];
    colorConfig[2][8] = colorConfigOld[5][6];
    colorConfig[1][6] = colorConfigOld[4][2];
    colorConfig[5][6] = colorConfigOld[2][8];
    colorConfig[4][2] = colorConfigOld[1][6];
}

void F3()
{
    colorConfig[0][0] = colorConfigOld[0][2];
    colorConfig[0][6] = colorConfigOld[0][0];
    colorConfig[0][8] = colorConfigOld[0][6];
    colorConfig[0][2] = colorConfigOld[0][8];
    colorConfig[0][1] = colorConfigOld[0][5];
    colorConfig[0][3] = colorConfigOld[0][1];
    colorConfig[0][7] = colorConfigOld[0][3];
    colorConfig[0][5] = colorConfigOld[0][7];
    colorConfig[2][6] = colorConfigOld[1][0];
    colorConfig[1][0] = colorConfigOld[5][0];
    colorConfig[5][0] = colorConfigOld[4][8];
    colorConfig[4][8] = colorConfigOld[2][6];
    colorConfig[2][7] = colorConfigOld[1][3];
    colorConfig[1][3] = colorConfigOld[5][3];
    colorConfig[5][3] = colorConfigOld[4][5];
    colorConfig[4][5] = colorConfigOld[2][7];
    colorConfig[2][8] = colorConfigOld[1][6];
    colorConfig[1][6] = colorConfigOld[5][6];
    colorConfig[5][6] = colorConfigOld[4][2];
    colorConfig[4][2] = colorConfigOld[2][8];
}

void B1()
{
    colorConfig[3][0] = colorConfigOld[3][6];
    colorConfig[3][6] = colorConfigOld[3][8];
    colorConfig[3][8] = colorConfigOld[3][2];
    colorConfig[3][2] = colorConfigOld[3][0];
    colorConfig[3][1] = colorConfigOld[3][3];
    colorConfig[3][3] = colorConfigOld[3][7];
    colorConfig[3][7] = colorConfigOld[3][5];
    colorConfig[3][5] = colorConfigOld[3][1];
    colorConfig[2][0] = colorConfigOld[1][2];
    colorConfig[1][2] = colorConfigOld[5][2];
    colorConfig[5][2] = colorConfigOld[4][6];
```

```
    colorConfig[4][6] = colorConfigOld[2][0];
    colorConfig[2][1] = colorConfigOld[1][5];
    colorConfig[1][5] = colorConfigOld[5][5];
    colorConfig[5][5] = colorConfigOld[4][3];
    colorConfig[4][3] = colorConfigOld[2][1];
    colorConfig[2][2] = colorConfigOld[1][8];
    colorConfig[1][8] = colorConfigOld[5][8];
    colorConfig[5][8] = colorConfigOld[4][0];
    colorConfig[4][0] = colorConfigOld[2][2];
}

void B2()
{
    colorConfig[3][0] = colorConfigOld[3][8];
    colorConfig[3][6] = colorConfigOld[3][2];
    colorConfig[3][8] = colorConfigOld[3][0];
    colorConfig[3][2] = colorConfigOld[3][6];
    colorConfig[3][1] = colorConfigOld[3][7];
    colorConfig[3][3] = colorConfigOld[3][5];
    colorConfig[3][7] = colorConfigOld[3][1];
    colorConfig[3][5] = colorConfigOld[3][3];
    colorConfig[2][0] = colorConfigOld[5][2];
    colorConfig[1][2] = colorConfigOld[4][6];
    colorConfig[5][2] = colorConfigOld[2][0];
    colorConfig[4][6] = colorConfigOld[1][2];
    colorConfig[2][1] = colorConfigOld[5][5];
    colorConfig[1][5] = colorConfigOld[4][3];
    colorConfig[5][5] = colorConfigOld[2][1];
    colorConfig[4][3] = colorConfigOld[1][5];
    colorConfig[2][2] = colorConfigOld[5][8];
    colorConfig[1][8] = colorConfigOld[4][0];
    colorConfig[5][8] = colorConfigOld[2][2];
    colorConfig[4][0] = colorConfigOld[1][8];
}

void B3()
{
    colorConfig[3][0] = colorConfigOld[3][2];
    colorConfig[3][6] = colorConfigOld[3][0];
    colorConfig[3][8] = colorConfigOld[3][6];
    colorConfig[3][2] = colorConfigOld[3][8];
    colorConfig[3][1] = colorConfigOld[3][5];
    colorConfig[3][3] = colorConfigOld[3][1];
    colorConfig[3][7] = colorConfigOld[3][3];
    colorConfig[3][5] = colorConfigOld[3][7];
    colorConfig[2][0] = colorConfigOld[4][6];
    colorConfig[1][2] = colorConfigOld[2][0];
    colorConfig[5][2] = colorConfigOld[1][2];
    colorConfig[4][6] = colorConfigOld[5][2];
    colorConfig[2][1] = colorConfigOld[4][3];
```

```
    colorConfig[1][5] = colorConfigOld[2][1];
    colorConfig[5][5] = colorConfigOld[1][5];
    colorConfig[4][3] = colorConfigOld[5][5];
    colorConfig[2][2] = colorConfigOld[4][0];
    colorConfig[1][8] = colorConfigOld[2][2];
    colorConfig[5][8] = colorConfigOld[1][8];
    colorConfig[4][0] = colorConfigOld[5][8];
}

void L1()
{
    colorConfig[4][0] = colorConfigOld[4][6];
    colorConfig[4][6] = colorConfigOld[4][8];
    colorConfig[4][8] = colorConfigOld[4][2];
    colorConfig[4][2] = colorConfigOld[4][0];
    colorConfig[4][1] = colorConfigOld[4][3];
    colorConfig[4][3] = colorConfigOld[4][7];
    colorConfig[4][7] = colorConfigOld[4][5];
    colorConfig[4][5] = colorConfigOld[4][1];
    colorConfig[2][0] = colorConfigOld[3][8];
    colorConfig[0][0] = colorConfigOld[2][0];
    colorConfig[5][6] = colorConfigOld[0][0];
    colorConfig[3][8] = colorConfigOld[5][6];
    colorConfig[2][3] = colorConfigOld[3][5];
    colorConfig[0][3] = colorConfigOld[2][3];
    colorConfig[5][7] = colorConfigOld[0][3];
    colorConfig[3][5] = colorConfigOld[5][7];
    colorConfig[2][6] = colorConfigOld[3][2];
    colorConfig[0][6] = colorConfigOld[2][6];
    colorConfig[5][8] = colorConfigOld[0][6];
    colorConfig[3][2] = colorConfigOld[5][8];
}

void L2()
{
    colorConfig[4][0] = colorConfigOld[4][8];
    colorConfig[4][6] = colorConfigOld[4][2];
    colorConfig[4][8] = colorConfigOld[4][0];
    colorConfig[4][2] = colorConfigOld[4][6];
    colorConfig[4][1] = colorConfigOld[4][7];
    colorConfig[4][3] = colorConfigOld[4][5];
    colorConfig[4][7] = colorConfigOld[4][1];
    colorConfig[4][5] = colorConfigOld[4][3];
    colorConfig[2][0] = colorConfigOld[5][6];
    colorConfig[0][0] = colorConfigOld[3][8];
    colorConfig[5][6] = colorConfigOld[2][0];
    colorConfig[3][8] = colorConfigOld[0][0];
    colorConfig[2][3] = colorConfigOld[5][7];
    colorConfig[0][3] = colorConfigOld[3][5];
    colorConfig[5][7] = colorConfigOld[2][3];
```

```
    colorConfig[3][5] = colorConfigOld[0][3];
    colorConfig[2][6] = colorConfigOld[5][8];
    colorConfig[0][6] = colorConfigOld[3][2];
    colorConfig[5][8] = colorConfigOld[2][6];
    colorConfig[3][2] = colorConfigOld[0][6];
}

void L3()
{
    colorConfig[4][0] = colorConfigOld[4][2];
    colorConfig[4][6] = colorConfigOld[4][0];
    colorConfig[4][8] = colorConfigOld[4][6];
    colorConfig[4][2] = colorConfigOld[4][8];
    colorConfig[4][1] = colorConfigOld[4][5];
    colorConfig[4][3] = colorConfigOld[4][1];
    colorConfig[4][7] = colorConfigOld[4][3];
    colorConfig[4][5] = colorConfigOld[4][7];
    colorConfig[2][0] = colorConfigOld[0][0];
    colorConfig[0][0] = colorConfigOld[5][6];
    colorConfig[5][6] = colorConfigOld[3][8];
    colorConfig[3][8] = colorConfigOld[2][0];
    colorConfig[2][3] = colorConfigOld[0][3];
    colorConfig[0][3] = colorConfigOld[5][7];
    colorConfig[5][7] = colorConfigOld[3][5];
    colorConfig[3][5] = colorConfigOld[2][3];
    colorConfig[2][6] = colorConfigOld[0][6];
    colorConfig[0][6] = colorConfigOld[5][8];
    colorConfig[5][8] = colorConfigOld[3][2];
    colorConfig[3][2] = colorConfigOld[2][6];
}

void R1()
{
    colorConfig[1][0] = colorConfigOld[1][6];
    colorConfig[1][6] = colorConfigOld[1][8];
    colorConfig[1][8] = colorConfigOld[1][2];
    colorConfig[1][2] = colorConfigOld[1][0];
    colorConfig[1][1] = colorConfigOld[1][3];
    colorConfig[1][3] = colorConfigOld[1][7];
    colorConfig[1][7] = colorConfigOld[1][5];
    colorConfig[1][5] = colorConfigOld[1][1];
    colorConfig[2][2] = colorConfigOld[0][2];
    colorConfig[0][2] = colorConfigOld[5][0];
    colorConfig[5][0] = colorConfigOld[3][6];
    colorConfig[3][6] = colorConfigOld[2][2];
    colorConfig[2][5] = colorConfigOld[0][5];
    colorConfig[0][5] = colorConfigOld[5][1];
    colorConfig[5][1] = colorConfigOld[3][3];
    colorConfig[3][3] = colorConfigOld[2][5];
    colorConfig[2][8] = colorConfigOld[0][8];
```

```
    colorConfig[0][8] = colorConfigOld[5][2];
    colorConfig[5][2] = colorConfigOld[3][0];
    colorConfig[3][0] = colorConfigOld[2][8];
}

void R2()
{
    colorConfig[1][0] = colorConfigOld[1][8];
    colorConfig[1][6] = colorConfigOld[1][2];
    colorConfig[1][8] = colorConfigOld[1][0];
    colorConfig[1][2] = colorConfigOld[1][6];
    colorConfig[1][1] = colorConfigOld[1][7];
    colorConfig[1][3] = colorConfigOld[1][5];
    colorConfig[1][7] = colorConfigOld[1][1];
    colorConfig[1][5] = colorConfigOld[1][3];
    colorConfig[2][2] = colorConfigOld[5][0];
    colorConfig[0][2] = colorConfigOld[3][6];
    colorConfig[5][0] = colorConfigOld[2][2];
    colorConfig[3][6] = colorConfigOld[0][2];
    colorConfig[2][5] = colorConfigOld[5][1];
    colorConfig[0][5] = colorConfigOld[3][3];
    colorConfig[5][1] = colorConfigOld[2][5];
    colorConfig[3][3] = colorConfigOld[0][5];
    colorConfig[2][8] = colorConfigOld[5][2];
    colorConfig[0][8] = colorConfigOld[3][0];
    colorConfig[5][2] = colorConfigOld[2][8];
    colorConfig[3][0] = colorConfigOld[0][8];
}

void R3()
{
    colorConfig[1][0] = colorConfigOld[1][2];
    colorConfig[1][6] = colorConfigOld[1][0];
    colorConfig[1][8] = colorConfigOld[1][6];
    colorConfig[1][2] = colorConfigOld[1][8];
    colorConfig[1][1] = colorConfigOld[1][5];
    colorConfig[1][3] = colorConfigOld[1][1];
    colorConfig[1][7] = colorConfigOld[1][3];
    colorConfig[1][5] = colorConfigOld[1][7];
    colorConfig[2][2] = colorConfigOld[3][6];
    colorConfig[0][2] = colorConfigOld[2][2];
    colorConfig[5][0] = colorConfigOld[0][2];
    colorConfig[3][6] = colorConfigOld[5][0];
    colorConfig[2][5] = colorConfigOld[3][3];
    colorConfig[0][5] = colorConfigOld[2][5];
    colorConfig[5][1] = colorConfigOld[0][5];
    colorConfig[3][3] = colorConfigOld[5][1];
    colorConfig[2][8] = colorConfigOld[3][0];
    colorConfig[0][8] = colorConfigOld[2][8];
    colorConfig[5][2] = colorConfigOld[0][8];
```

```
    colorConfig[3][0] = colorConfigOld[5][2];
}

void display_step(int i, char *str)
{
    if(!strcmp(solution[i], "FF")) {
     SET_TEXT(2, "CONGRATULATIONS!\n\nTHE CUBE IS SOLVED!\n\n");
    } else if(!strcmp(solution[i], "U1")) {
     SET_TEXT(2, "NEXT STEP:\n\nTOP FACE\nCLOCKWISE\n");
    } else if(!strcmp(solution[i], "U2")) {
     SET_TEXT(2, "NEXT STEP:\n\nTOP FACE\n180 DEGREES\n");
    } else if(!strcmp(solution[i], "U3")) {
     SET_TEXT(2, "NEXT STEP:\n\nTOP FACE\nCOUNTER-CLOCKWISE\n");
    } else if(!strcmp(solution[i], "D1")) {
     SET_TEXT(2, "NEXT STEP:\n\nBOTTOM FACE\nCLOCKWISE\n");
    } else if(!strcmp(solution[i], "D2")) {
     SET_TEXT(2, "NEXT STEP:\n\nBOTTOM FACE\n180 DEGREES\n");
    } else if(!strcmp(solution[i], "D3")) {
     SET_TEXT(2, "NEXT STEP:\n\nBOTTOM FACE\nCOUNTER-CLOCKWISE\n");
    } else if(!strcmp(solution[i], "F1")) {
     SET_TEXT(2, "NEXT STEP:\n\nFRONT FACE\nCLOCKWISE\n");
    } else if(!strcmp(solution[i], "F2")) {
     SET_TEXT(2, "NEXT STEP:\n\nFRONT FACE\n180 DEGREES\n");
    } else if(!strcmp(solution[i], "F3")) {
     SET_TEXT(2, "NEXT STEP:\n\nFRONT FACE\nCOUNTER-CLOCKWISE\n");
    } else if(!strcmp(solution[i], "B1")) {
     SET_TEXT(2, "NEXT STEP:\n\nBACK FACE\nCLOCKWISE\n");
    } else if(!strcmp(solution[i], "B2")) {
     SET_TEXT(2, "NEXT STEP:\n\nBACK FACE\n180 DEGREES\n");
    } else if(!strcmp(solution[i], "B3")) {
     SET_TEXT(2, "NEXT STEP:\n\nBACK FACE\nCOUNTER-CLOCKWISE\n");
    } else if(!strcmp(solution[i], "L1")) {
     SET_TEXT(2, "NEXT STEP:\n\nLEFT FACE\nCLOCKWISE\n");
    } else if(!strcmp(solution[i], "L2")) {
     SET_TEXT(2, "NEXT STEP:\n\nLEFT FACE\n180 DEGREES\n");
    } else if(!strcmp(solution[i], "L3")) {
     SET_TEXT(2, "NEXT STEP:\n\nLEFT FACE\nCOUNTER-CLOCKWISE\n");
    } else if(!strcmp(solution[i], "R1")) {
     SET_TEXT(2, "NEXT STEP:\n\nRIGHT FACE\nCLOCKWISE\n");
    } else if(!strcmp(solution[i], "R2")) {
     SET_TEXT(2, "NEXT STEP:\n\nRIGHT FACE\n180 DEGREES\n");
    } else if(!strcmp(solution[i], "R3")) {
     SET_TEXT(2, "NEXT STEP:\n\nRIGHT FACE\nCOUNTER-CLOCKWISE\n");
    }
}
```

---

main.c

```
/*
  Jaap Scherphuis,  24/01/2004,  jaapsch_at_yahoo_do_com

  Thistlethwaite's algorithm.
*/

#include "main.h"

// get index of cube position from table t
int getposition(int t){
    int i=-1,n=0;
    int corn[8],j,k,l,corn2[4];
    switch(t){
    // case 0 does nothing so returns 0
    case 1://edgeflip
     // 12 bits, set bit if edge is flipped
     for(;++i<12;) n+= GET_ORI(i)<<i;
     break;
    case 2://cornertwist
     // get base 3 number of 8 digits - each digit is corner twist
     for(i=20;--i>11;) n=n*3+GET_ORI(i);
     break;
    case 3://middle edge choice
     // 12 bits, set bit if edge belongs in Um middle slice
     for(;++i<12;) n+= (GET_POS(i)&8)?(1<<i):0;
     break;
    case 4://ud slice choice
     // 8 bits, set bit if UD edge belongs in Fm middle slice
     for(;++i<8;) n+= (GET_POS(i)&4)?(1<<i):0;
     break;
    case 5://tetrad choice, twist and parity
     // 8 bits, set bit if corner belongs in second tetrad.
     // also separate pieces for twist/parity determination
     k=j=0;
     for(;++i<8;)
          if((l=GET_POS(i+12)-12)&4){
               corn[l]=k++;
               n+=1<<i;
          }else corn[j++]=l;
     //Find permutation of second tetrad after solving first
     for(i=0;i<4;i++) corn2[i]=corn[4+corn[i]];
     //Solve one piece of second tetrad
     for(;--i;) corn2[i]^=corn2[0];

     // encode parity/tetrad twist
     n=n*6+corn2[1]*2-2;
     if(corn2[3]<corn2[2])n++;
     break;
    case 6://two edge and one corner orbit, permutation
     numOfPermtonum+=3;
```

```
      n=PERMTONUM0()*576+PERMTONUM1()*24+PERMTONUM3();
      //n=PERMTONUM0();
      break;
    case 7://one edge and one corner orbit, permutation
      numOfPermtonum+=2;
      n=PERMTONUM2()*24+PERMTONUM4();
      //n=PERMTONUM1();
      break;
    }
    return n;
}

void filltable(){
    tables[0] = table0;
    tables[1] = table1;
    tables[2] = table2;
    tables[3] = table3;
    tables[4] = table4;
    tables[5] = table5;
    tables[6] = table6;
    tables[7] = table7;
}

// Pruned tree search. recursive.
int searchphase(int movesleft, int movesdone,int lastmove){
    int i,j;

    // prune - position must still be solvable in the remaining moves
available
    if( tables[phase  ][getposition(phase  )]-1 > movesleft ||
        tables[phase+1][getposition(phase+1)]-1 > movesleft ) return 0;

    // If no moves left to do, we have solved this phase
    if(!movesleft) return 1;

    // not solved. try each face move
    for(i=6;i--;){
      // do not repeat same face, nor do opposite after DLB.
      if( i-lastmove && (i-lastmove+1 || (i|1) ) ){
            move[movesdone]=i;
            // try 1,2,3 quarter turns of that face
            for(j=0;++j<4;){
                  //do move and remember it
                  DOMOVE(i);
                  moveamount[movesdone]=j;
                  //Check if phase only allows half moves of this face
                  if( (j==2 || i>=phase ) &&
                        //search on
                        searchphase(movesleft-1,movesdone+1,i) ) return 1;
            }
```

```
            // put face back to original position.
            DOMOVE(i);
        }
    }
    // no solution found
    return 0;
}

void inputFormatTransform(int colorConfig[6][9])
{
    int i;
    char centerColorPosition[6];

    for(i=0; i<6; i++)
        centerColorPosition[colorConfig[i][4] - 1] = CENTER_COLOR_POSITION[i];

    for(i=0; i<12; i++)
    {
        transformedInputFormat[i][0] =
centerColorPosition[colorConfig[SIDE_INDEX_1[i]][SIDE_INDEX_2[i]]-1];
        transformedInputFormat[i][1] =
centerColorPosition[colorConfig[SIDE_INDEX_3[i]][SIDE_INDEX_4[i]]-1];
        transformedInputFormat[i][2] = '\0';
    }

    for(i=0; i<8; i++)
    {
        transformedInputFormat[i+12][0] =
centerColorPosition[colorConfig[CORNER_INDEX_1[i]][CORNER_INDEX_2[i]]-1];
        transformedInputFormat[i+12][1] =
centerColorPosition[colorConfig[CORNER_INDEX_3[i]][CORNER_INDEX_4[i]]-1];
        transformedInputFormat[i+12][2] =
centerColorPosition[colorConfig[CORNER_INDEX_5[i]][CORNER_INDEX_6[i]]-1];
    }
}

int inputColor()
{/*
    int colorNum[6] = {0,0,0,0,0,0};
    int centerColorTaken[6] = {0,0,0,0,0,0};
    int aa, ii, jj;
    char *line = (char *)malloc(sizeof(char) * 11);

    printf("-----------------------------------------------------------------
\n");
    printf("Please input the color configuration.\n");
    printf("white = 1, red = 2, blue = 3, orange = 4, green = 5, yellow =
6\n\n");

    printf("Sequence: up -> down -> front -> right -> back -> left\n");
```

```
    printf("Please input from left to right, top to bottom, e.g.
544462653\n");
    for(ii=1; ii<=6; ii++)
    {

        printf("Please input for side %d:", ii);
        SET_BORDER(ii-1);
        fgets(line, 11, stdin);
        if((int)strlen(line) != 10)
        {
            printf("Invalid input.\n");
            ii--;
        }
        else
        {
            for(jj=0; jj<9; jj++)
            {
                if(line[jj] < '1' || line[jj] > '6')
                {
                    printf("Invalid input.\n");
                    ii--;
                    break;
                }
                else
                {
                    colorConfig[ii-1][jj] = (int)(line[jj] - '0');
                }
            }
        }
        for(aa=0;aa<9;aa++)
        {
           SET_COLOR(ii-1,aa,colorConfig[ii-1][aa]);
        }
    }
    */
    int colorNum[6];
    int centerColorTaken[6];
    int face = 0, i, j;
    SET_BORDER(face);

CubeInput:

    for(i=0; i<6; i++) {
      colorNum[i] = 0;
      centerColorTaken[i] = 0;
    }

    for (;;) {
        status = read_make_code(&decode_mode, &key);
        if (status == PS2_SUCCESS) {
```

```
            switch (decode_mode) {
            case KB_ASCII_MAKE_CODE :
                  switch (key) {
                  case 97://KP 1
                        colorConfig[face][6] =
colorConfig[face][6]==6?1:colorConfig[face][6]+1;
                        SET_COLOR(face,6,colorConfig[face][6]);
                        break;
                  case 98://KP 2
                        colorConfig[face][7] =
colorConfig[face][7]==6?1:colorConfig[face][7]+1;
                        SET_COLOR(face,7,colorConfig[face][7]);
                        break;
                  case 99://KP 3
                        colorConfig[face][8] =
colorConfig[face][8]==6?1:colorConfig[face][8]+1;
                        SET_COLOR(face,8,colorConfig[face][8]);
                        break;
                  case 100://KP 4
                        colorConfig[face][3] =
colorConfig[face][3]==6?1:colorConfig[face][3]+1;
                        SET_COLOR(face,3,colorConfig[face][3]);
                        break;
                  case 101://KP 5
                        colorConfig[face][4] =
colorConfig[face][4]==6?1:colorConfig[face][4]+1;
                        SET_COLOR(face,4,colorConfig[face][4]);
                        break;
                  case 102://KP 6
                        colorConfig[face][5] =
colorConfig[face][5]==6?1:colorConfig[face][5]+1;
                        SET_COLOR(face,5,colorConfig[face][5]);
                        break;
                  case 103://KP 7
                        colorConfig[face][0] =
colorConfig[face][0]==6?1:colorConfig[face][0]+1;
                        SET_COLOR(face,0,colorConfig[face][0]);
                        break;
                  case 104://KP 8
                        colorConfig[face][1] =
colorConfig[face][1]==6?1:colorConfig[face][1]+1;
                        SET_COLOR(face,1,colorConfig[face][1]);
                        break;
                  case 105://KP 9
                        colorConfig[face][2] =
colorConfig[face][2]==6?1:colorConfig[face][2]+1;
                        SET_COLOR(face,2,colorConfig[face][2]);
                        break;
                  }
                  break ;
```

```
            case KB_LONG_BINARY_MAKE_CODE :
            case KB_BINARY_MAKE_CODE :
                  switch (key) {
                  case 0x5a: //enter key
                        goto CubeCheck;
                        break;
                  case 0x0D: //tab key
                        face = (face+1)%6;
                        SET_BORDER(face);
                        break;
                  default:
                        break;
                  }
                  break;
            case KB_BREAK_CODE :
                  // do nothing
                  default:
                  break ;
            }
        }
        else {
          printf(" Keyboard error ....\n");
        }
    }

CubeCheck:

    for(i=0; i<6; i++)
    {
        for(j=0; j<9; j++)
        {
            if(++colorNum[colorConfig[i][j]-1] > 9)
            {
                SET_TEXT(2, "NOT A VALID CUBE\n\n\n\n");
                goto CubeInput;
            }
        }
    }

    for(i=0; i<6; i++)
    {
        int tmpColor = colorConfig[i][4] - 1;
        if(!centerColorTaken[tmpColor])
        {
            centerColorTaken[tmpColor] = 1;
        }
        else
        {
           SET_TEXT(2, "NOT A VALID CUBE\n\n\n\n");
            goto CubeInput;
```

```
        }
    }

    inputFormatTransform(colorConfig);

//    free(line);
    return 0;
}

void solveCube()
{
    int f,i=0,j=0,k=0,pc,mor;

    // initialise tables
    for(; k<20; k++) val[k]=k<12?2:3;

    //for(; j<8; j++) filltable(j);
    filltable();

    // read input, 20 pieces worth
    for(; i<20; i++){
      f=pc=k=mor=0;
      for(;f<val[i];f++){
            j=strchr(faces,transformedInputFormat[i][f])-faces;
            // keep track of principal facelet for orientation
            if(j>k) {k=j;mor=f;}
            //construct bit hash code
            pc+= 1<<j;
      }
      // find which cubelet it belongs, i.e. the label for this piece
      for(f=0; f<20; f++)
            if(pc==bithash[f]-64) break;
      // store piece
      //pos[order[i]-CHAROFFSET]=f;
      SET_POS(order[i]-CHAROFFSET, f);
      //ori[order[i]-CHAROFFSET]=mor%val[i];
      SET_ORI(order[i]-CHAROFFSET, mor%val[i]);
    }

    SET_PERMTONUM0(14);
    SET_PERMTONUM1(14);
    SET_PERMTONUM2(0);
    SET_PERMTONUM3(6);
    SET_PERMTONUM4(1);

    //solve the cube
    // four phases
    k = 0;
    printf("Solution: ");
    for(phase=0 ; phase<8; phase+=2){
```

```
      // try each depth till solved
      for(j=0; !searchphase(j,0,9); j++);
      //output result of this phase
      for(i=0; i<j; i++){
            printf("%c%d ", "FBRLUD"[move[i]], moveamount[i]);
            solution[k][0] = "FBRLUD"[move[i]];
            solution[k][1] = '0' + moveamount[i];
            solution[k++][2] = '\0';
      }
    }
    solution[k][0] = 'F';
    solution[k][1] = 'F';
    solution[k][2] = '\0';
    printf("\n");
}

void checkSolution()
{
    int i, j, flag;
    for(i=0;i<79;i++) {
      if(!strcmp(solution[i], "FF") || !strcmp(solution[i+1], "FF"))
            return;
      if(solution[i][0] == solution[i+1][0]) {
            flag = 1;
            if(solution[i][1] == '1' && solution[i+1][1] == '2')
                  solution[i][1] = '3';
            else if(solution[i][1] == '3' && solution[i+1][1] == '2')
                  solution[i][1] = '1';
            else if(solution[i][1] == '2' && solution[i+1][1] == '1')
                  solution[i][1] = '3';
            else if(solution[i][1] == '2' && solution[i+1][1] == '3')
                  solution[i][1] = '1';
            else
                  flag = 0;
            if(flag) {
                  j = i + 1;
                  for(;j<79;j++)
                        strcpy(solution[j], solution[j+1]);
            }
      }
    }
}

void demo()
{
    int i;
    for(i=0;i<79;i++) {
      display_step(i, solution[i]);
      for (;;) {
            status = read_make_code(&decode_mode, &key);
```

```
if (status == PS2_SUCCESS) {
    switch (decode_mode) {
    case KB_ASCII_MAKE_CODE :
         break ;
    case KB_LONG_BINARY_MAKE_CODE :
    case KB_BINARY_MAKE_CODE :
         switch (key) {
         case 0x76: //esc key
              return;
              break;
         case 0x6B: //left arrow key
              if(i) {
                   updateColorConfigOld();
                   if(!strcmp(solution[i-1], "U1")) {
                        U3();
                   } else if(!strcmp(solution[i-1], "U2")) {
                        U2();
                   } else if(!strcmp(solution[i-1], "U3")) {
                        U1();
                   } else if(!strcmp(solution[i-1], "D1")) {
                        D3();
                   } else if(!strcmp(solution[i-1], "D2")) {
                        D2();
                   } else if(!strcmp(solution[i-1], "D3")) {
                        D1();
                   } else if(!strcmp(solution[i-1], "F1")) {
                        F3();
                   } else if(!strcmp(solution[i-1], "F2")) {
                        F2();
                   } else if(!strcmp(solution[i-1], "F3")) {
                        F1();
                   } else if(!strcmp(solution[i-1], "B1")) {
                        B3();
                   } else if(!strcmp(solution[i-1], "B2")) {
                        B2();
                   } else if(!strcmp(solution[i-1], "B3")) {
                        B1();
                   } else if(!strcmp(solution[i-1], "L1")) {
                        L3();
                   } else if(!strcmp(solution[i-1], "L2")) {
                        L2();
                   } else if(!strcmp(solution[i-1], "L3")) {
                        L1();
                   } else if(!strcmp(solution[i-1], "R1")) {
                        R3();
                   } else if(!strcmp(solution[i-1], "R2")) {
                        R2();
                   } else if(!strcmp(solution[i-1], "R3")) {
                        R1();
                   }
```

```
            updateCube();
                  i -= 2;
            } else {
                  i -= 1;
            }
            goto NextIteration;
            break;
      case 0x74: //right arrow key
            updateColorConfigOld();
            if(!strcmp(solution[i], "FF")) {
                  i--;
                  goto NextIteration;
            } else if(!strcmp(solution[i], "U1")) {
                  U1();
            } else if(!strcmp(solution[i], "U2")) {
                  U2();
            } else if(!strcmp(solution[i], "U3")) {
                  U3();
            } else if(!strcmp(solution[i], "D1")) {
                  D1();
            } else if(!strcmp(solution[i], "D2")) {
                  D2();
            } else if(!strcmp(solution[i], "D3")) {
                  D3();
            } else if(!strcmp(solution[i], "F1")) {
                  F1();
            } else if(!strcmp(solution[i], "F2")) {
                  F2();
            } else if(!strcmp(solution[i], "F3")) {
                  F3();
            } else if(!strcmp(solution[i], "B1")) {
                  B1();
            } else if(!strcmp(solution[i], "B2")) {
                  B2();
            } else if(!strcmp(solution[i], "B3")) {
                  B3();
            } else if(!strcmp(solution[i], "L1")) {
                  L1();
            } else if(!strcmp(solution[i], "L2")) {
                  L2();
            } else if(!strcmp(solution[i], "L3")) {
                  L3();
            } else if(!strcmp(solution[i], "R1")) {
                  R1();
            } else if(!strcmp(solution[i], "R2")) {
                  R2();
            } else if(!strcmp(solution[i], "R3")) {
                  R3();
            }
            updateCube();
```

```
                              goto NextIteration;
                              break;
                         default:
                              break;
                         }
                         break;
                    case KB_BREAK_CODE :
                         // do nothing
                         default:
                         break ;
                    }
               }
               else {
                 printf(" Keyboard error ....\n");
               }
          }

NextIteration:
      continue;
    }
}

int main(){

    //unsigned int timerLow, timerHigh;
    //unsigned long timerValue, timerValueAdd;
    //double duration;

    //int i,j,k;
    int i;
    SET_TEXT(1, "WELCOME TO RUBIK'S CUBE SOLVER");
    SET_TEXT(2, "INITIALIZING...\n\n\n\n");

    // Initialize the keyboard
    clear_FIFO();
    switch (get_mode()) {
    case PS2_KEYBOARD:
      break;
    case PS2_MOUSE:
      printf("Error: Mouse detected on PS/2 port\n");
      goto ErrorExit;
    default:
      printf("Error: Unrecognized or no device on PS/2 port\n");
      goto ErrorExit;
    }

Restart:

    numOfDomove = 0;
    numOfPermtonum = 0;
```

```
    SET_TEXT(2, "\n\n\n\n\n");
    SET_TEXT(3, "USE THE NUMPAD AND\nTAB KEY TO INPUT\n\nPRESS ENTER TO
START");

    for(i=0;i<9;i++) {
      colorConfig[0][i] = 3;
      colorConfig[1][i] = 6;
      colorConfig[2][i] = 4;
      colorConfig[3][i] = 5;
      colorConfig[4][i] = 1;
      colorConfig[5][i] = 2;
    }
    updateCube();

    if(inputColor())
        return -1;

   // RESET_TIMER();
   // ENABLE_TIMER();
    CLEAR_BORDER();
    SET_TEXT(2, "SOLVING...\n\n\n\n\n");
    SET_TEXT(3, "\n\n\n\n\n");
    solveCube();

   // DISABLE_TIMER();
   // timerLow = GET_TIMER_LOW();
  //  timerHigh = GET_TIMER_HIGH();
  //  timerValue = ((unsigned long)timerHigh) << 16;
  //  timerValueAdd = timerValue + (unsigned long)timerLow;
  //  duration = ((double)timerValueAdd)*40/1000000000;

//    printf("Running time: %lf\n", duration);
    printf("Number of Domove: %lu\n", numOfDomove);
    printf("Number of Permtonum: %lu\n", numOfPermtonum);
/*
    k=0;
    while(1) {
      printf("%s ", solution[k]);
      k++;
      if(!strcmp(solution[k], "FF"))
           break;
    }
    printf("\n");*/
    checkSolution();
    SET_TEXT(3, "USE LEFT AND RIGHT\nARROWS TO NAVIGATE\n\nPRESS ESC TO
RESTART");
    demo();
    goto Restart;
ErrorExit:
```

```
    printf("Program terminated with an error condition\n");
    return 1;
}

------------------------------------------------------------------
alt_up_ps2_port.h
------------------------------------------------------------------

#ifndef __PS2_INTERFACE_H__
#define __PS2_INTERFACE_H__

#include <io.h>
#include <alt_types.h>
#include "system.h"
#include "alt_up_ps2_port_regs.h"

#define ALT_UP_PS2_BASE ALT_UP_PS2_0_BASE

/**
 * @brief The Enum type for PS/2 device type
 **/
typedef enum {
  // @brief Indicate that the device is a PS/2 Mouse
  PS2_MOUSE = 0,
  // @brief Indicate that the device is a PS/2 Keyboard
  PS2_KEYBOARD = 1,
  // @brief The program cannot determine what type the device is
  PS2_UNKNOWN = 2
} PS2_DEVICE;

#define DEFAULT_PS2_TIMEOUT_VAL  700000

#define PS2_SUCCESS (0)
#define PS2_TIMEOUT (-1)
#define PS2_ERROR   (-2)

#define PS2_ACK (0xFA)

////////////////////////////////////////////////////////////
// Control Register Operations

/**
 * @brief Read the contents of the Control register for the PS/2 port
 *
 * @return Register contents (32 bits, bits 10, 8 and 0 are used for
 * CE, RI and RE respectively. Other bits are reserved)
 **/
alt_u32 read_ctrl_reg();

/**
```

```
 * @brief Set the contents of the Control register
 *
 * @param ctrl_data -- contents to be written into the Control register
 *
 **/
void write_ctrl_reg(alt_u32 ctrl_data);

/**
 * @brief Extract the RI (Read Interrupt) bit from the Control register
 *
 * @param ctrl_reg -- the Control register
 *
 * @return 8-bit number, where bit 0 is the value of the RI bit
 **/
alt_u8 read_RI_bit(alt_u32 ctrl_reg);

/**
 * @brief Extract the RE (Read Interrupt Enable) bit from the Control
register
 *
 * @param ctrl_reg -- the Control register
 *
 * @return 8-bit number, where bit 0 is the value of the RE bit
 **/
alt_u8 read_RE_bit(alt_u32 ctrl_reg);

/**
 * @brief Extract the CE (Command Error) bit from the Control register
 *
 * @param ctrl_reg -- the Control register
 *
 * @return 8-bit number, where bit 0 is the value of the CE bit
 **/
alt_u8 read_CE_bit(alt_u32 ctrl_reg);

////////////////////////////////////////////////////////
// Data Register Operations

/**
 * @brief Read the contents of the Data register
 *
 * @return 32 bits of the Data register. Bits 31-16 indicate the number
 * of available bytes in the FIFO (RAVAIL), bits 7-0 are the data received
 * from the PS/2 device
 *
 **/
alt_u32 read_data_reg();

/**
 * @brief Read the DATA byte from the Data register
```

```
 *
 * @param data_reg  -- Data register
 *
 * @return Bits 7-0 of the Data register
 **/
alt_u8 read_data_byte(alt_u32 data_reg);

/**
 * @brief Find the number of bytes available to read in the FIFO buffer
 * of the PS/2 port
 *
 * @param data_reg  -- the Data register
 *
 * @return The number represented by bits 31-16 of the Data register
 **/
alt_u16 read_num_bytes_available(alt_u32 data_reg);

////////////////////////////////////////////////////////
// Actions

/**
 * @brief Check the PS/2 peripheral's mode (whether it is a keyboard or
 *        a mouse)
 *
 * @return PS2_MOUSE for mouse, or PS2_KEYBOARD for keyboard
 *
 * @note This operation will reset the PS/2 peripheral. Usually,
 * it should be used only at the beginning of a program.
 **/
PS2_DEVICE get_mode();

/**
 * @brief Clear the FIFO's contents
 **/
void clear_FIFO();

/**
 * @brief Wait for the acknowledge byte (0xFA) from the PS/2 peripheral
 *
 * @param timeout -- the number of cycles of timeout
 *
 * @return \c PS2_SUCCESS on receving ACK signal, or \c PS2_TIMEOUT on
timeout.
 **/
int wait_for_ack(unsigned timeout);

/**
 * @brief Send a one-byte command to the PS/2 peripheral
 *
 * @param byte -- the one-byte command to be sent
```

```
 *
 * @return \c PS2_ERROR if the CE bit of the Control register is
 * set to 1, otherwise \c PS2_SUCCESS
 **/
int write_data_byte(alt_u8 byte);

/**
 * @brief Send a one-byte command to the PS/2 peripheral and
 *        wait for the ACK signal
 *
 * @param byte -- the one-byte command to be sent.
 *        See <tt> alt_up_ps2_port_regs.h </tt> in the sdk directory or
 *        any reference for the PS/2 protocol for details.
 *
 * @return PS2_ERROR if the CE bit of the Control register is set to 1, or
 *         PS2_TIMEOUT on timeout, or
 *         PS2_SUCCESS if the ACK signal is received before timeout
 **/
int write_data_byte_with_ack(alt_u8 byte, unsigned timeout);

/**
 * @brief Read the DATA byte from the PS/2 FIFO,
 *        using a user-defined timeout value
 *
 * @param byte  -- the byte read from the FIFO for the PS/2 Core
 * @param time_out  -- the user-defined timeout value. Setting
 *        time_out to 0 will disable the time-out mechanism
 *
 * @return \c PS2_SUCCESS on reading data, or \c PS2_TIMEOUT on timeout
 **/
int read_data_byte_with_timeout(alt_u8 *byte, alt_u32 time_out);

#endif
```

---

alt_up_ps2_port.c

```
#include <nios2.h>
#include "alt_up_ps2_port.h"

PS2_DEVICE get_mode()
{
  alt_u8 byte;
  //send the reset request, wait for ACK
  int status = write_data_byte_with_ack(0xff, DEFAULT_PS2_TIMEOUT_VAL);
  if (status == PS2_SUCCESS) {
    // reset succeed, now try to get the BAT result, AA means passed
    status = read_data_byte_with_timeout(&byte, DEFAULT_PS2_TIMEOUT_VAL);
    if (status == PS2_SUCCESS && byte == 0xAA) {
```

```
      //get the 2nd byte
      status = read_data_byte_with_timeout(&byte, DEFAULT_PS2_TIMEOUT_VAL);
      if (status == PS2_TIMEOUT) {
    //for keyboard, only 2 bytes are sent(ACK, PASS/FAIL), so timeout
    return PS2_KEYBOARD;
      } else if (status == PS2_SUCCESS && byte == 0x00) {
    //for mouse, it will sent out 0x00 after sending out ACK and PASS/FAIL.
    return PS2_MOUSE;
      }
    }
  }
  // when writing data to the PS/2 device, error occurs...
  return PS2_UNKNOWN;
}

void clear_FIFO()
{
  // The DATA byte of the data register will be automatically cleared after
  // a read, so we simply keep reading it until there are no available bytes
  alt_u16 num = 0;
  alt_u32 data_reg = 0;
  do {
    // read the data register (the DATA byte is cleared)
    data_reg = read_data_reg();
    // get the number of available bytes from the RAVAIL part of data
register
    num = read_num_bytes_available(data_reg);
  } while (num > 0);
}

////////////////////////////////////////////////////////////
// Control Register Operations
void write_ctrl_reg(alt_u32 ctrl_data)
{
  IOWR_ALT_UP_PS2_PORT_CONTROL(ALT_UP_PS2_BASE, ctrl_data);
}

alt_u32 read_ctrl_reg()
{
  alt_u32 ctrl_reg = IORD_ALT_UP_PS2_PORT_CONTROL(ALT_UP_PS2_BASE);
  return ctrl_reg;
}

alt_u8 read_RI_bit(alt_u32 ctrl_reg)
{
  alt_u8 ri = (alt_u8) ((ctrl_reg & ALT_UP_PS2_PORT_CONTROL_RI_MSK)
           >> ALT_UP_PS2_PORT_CONTROL_RI_OFST);
  return ri;
}
```

```
alt_u8 read_RE_bit(alt_u32 ctrl_reg)
{
  alt_u8 re = (alt_u8) ((ctrl_reg & ALT_UP_PS2_PORT_CONTROL_RE_MSK)
           >> ALT_UP_PS2_PORT_CONTROL_RE_OFST);
  return re;
}

alt_u8 read_CE_bit(alt_u32 ctrl_reg)
{
  alt_u8 re = (alt_u8) ((ctrl_reg & ALT_UP_PS2_PORT_CONTROL_CE_MSK)
           >> ALT_UP_PS2_PORT_CONTROL_CE_OFST);
  return re;
}

////////////////////////////////////////////////////////
// Data Register Operations

alt_u32 read_data_reg()
{
  alt_u32 data_reg = IORD_ALT_UP_PS2_PORT_DATA(ALT_UP_PS2_BASE);
  return data_reg;
}

alt_u16 read_num_bytes_available(alt_u32 data_reg)
{
  alt_u16 ravail = (alt_u16)((data_reg & ALT_UP_PS2_PORT_DATA_REG_RAVAIL_MSK )
                >> ALT_UP_PS2_PORT_DATA_REG_RAVAIL_OFST);
  return ravail;
}

alt_u8 read_data_byte(alt_u32 data_reg)
{
  alt_u8 data = (alt_u8) ( (data_reg & ALT_UP_PS2_PORT_DATA_REG_DATA_MSK)
              >> ALT_UP_PS2_PORT_DATA_REG_DATA_OFST) ;
  return data;
}

int write_data_byte(alt_u8 byte)
{
  //note: data are only located at the lower 8 bits
  //note: the software send command to the PS2 peripheral through the data
  //       register rather than the control register
  IOWR_ALT_UP_PS2_PORT_DATA(ALT_UP_PS2_BASE, byte);
  alt_u32 ctrl_reg = IORD_ALT_UP_PS2_PORT_DATA(ALT_UP_PS2_BASE);
  if ( read_CE_bit(ctrl_reg) ) {
    //CE bit is set --> error occurs on sending commands
    return PS2_ERROR;
  }
  return PS2_SUCCESS;
}
```

```
int write_data_byte_with_ack(alt_u8 byte, unsigned timeout)
{
  //note: data are only located at the lower 8 bits
  //note: the software send command to the PS2 peripheral through the data
  //       register rather than the control register
  int send_status = write_data_byte(byte);
  if ( send_status != PS2_SUCCESS ) {
    // return on sending error
    return send_status;
  }

  int ack_status = wait_for_ack(timeout);
  return ack_status;
}

int read_data_byte_with_timeout(alt_u8 *byte, alt_u32 time_out)
{
  alt_u32 data_reg = 0;
  alt_u16 num = 0;
  alt_u32 count = 0;
  for (;;) {
    count++;
    data_reg = read_data_reg();
    num = read_num_bytes_available(data_reg);
    if (num > 0) {
      *byte = read_data_byte(data_reg);
      return PS2_SUCCESS;
    }
    //timeout = 0 means to disable the timeout
    if ( time_out!= 0 && count > time_out) {
      return PS2_TIMEOUT;
    }
  }
}

int wait_for_ack(unsigned timeout)
{
  alt_u8 ack = 0;
  alt_u8 data = 0;
  alt_u8 status = PS2_SUCCESS;
  for (;;) {
    status = read_data_byte_with_timeout(&data, timeout);
    if ( status == PS2_SUCCESS ) {
      if (data == PS2_ACK)
    return PS2_SUCCESS;
    } else {
      return status;
    }
  }
```

```
  return PS2_TIMEOUT;
}
```

---

alt_up_ps2_port_regs.h

```
#ifndef __ALT_UP_PS2_PORT_REGS_H__
#define __ALT_UP_PS2_PORT_REGS_H__

/*
 * Data Register
 */
#define ALT_UP_PS2_PORT_DATA_REG                    0
#define IOADDR_ALT_UP_PS2_PORT_DATA(base)           \
        __IO_CALC_ADDRESS_NATIVE(base, ALT_UP_PS2_PORT_DATA_REG)
#define IORD_ALT_UP_PS2_PORT_DATA(base)             \
        IORD(base, ALT_UP_PS2_PORT_DATA_REG)
#define IOWR_ALT_UP_PS2_PORT_DATA(base, data)       \
        IOWR(base, ALT_UP_PS2_PORT_DATA_REG, data)

#define ALT_UP_PS2_PORT_DATA_REG_DATA_MSK    (0x000000FF)
#define ALT_UP_PS2_PORT_DATA_REG_DATA_OFST   (0)
#define ALT_UP_PS2_PORT_DATA_REG_RVALID_MSK  (0x00008000)
#define ALT_UP_PS2_PORT_DATA_REG_RVALID_OFST       (15)
#define ALT_UP_PS2_PORT_DATA_REG_RAVAIL_MSK  (0xFFFF0000)
#define ALT_UP_PS2_PORT_DATA_REG_RAVAIL_OFST       (16)

/*
 * Control Register
 */
#define ALT_UP_PS2_PORT_CONTROL_REG                 1
#define IOADDR_ALT_UP_PS2_PORT_CONTROL(base)        \
        __IO_CALC_ADDRESS_NATIVE(base, ALT_UP_PS2_PORT_CONTROL_REG)
#define IORD_ALT_UP_PS2_PORT_CONTROL(base)          \
        IORD(base, ALT_UP_PS2_PORT_CONTROL_REG)
#define IOWR_ALT_UP_PS2_PORT_CONTROL(base, data)    \
        IOWR(base, ALT_UP_PS2_PORT_CONTROL_REG, data)

#define ALT_UP_PS2_PORT_CONTROL_RE_MSK              (0x00000001)
#define ALT_UP_PS2_PORT_CONTROL_RE_OFST             (0)
#define ALT_UP_PS2_PORT_CONTROL_RI_MSK              (0x00000100)
#define ALT_UP_PS2_PORT_CONTROL_RI_OFST             (8)
#define ALT_UP_PS2_PORT_CONTROL_CE_MSK               (0x00000400)
#define ALT_UP_PS2_PORT_CONTROL_CE_OFST       (10)

#endif
```

---

ps2_keyboard.h

```
--------------------------------------------------------------

#ifndef __PS2_KEYBOARD_H__
#define __PS2_KEYBOARD_H__
#include "alt_up_ps2_port.h"

#define KB_RESET 0xFF
#define KB_SET_DEFAULT 0xF6
#define KB_DISABLE 0xF5
#define KB_ENABLE 0xF4
#define KB_SET_TYPE_RATE_DELAY 0xF3

/**
 * @brief The Enum type for the type of keyboard code received
 **/
typedef enum
  {
    /** @brief --- Make Code that corresponds to an ASCII character.
    For example, the ASCII Make Code for letter <tt>A</tt> is 1C
     */
    KB_ASCII_MAKE_CODE = 1,
    /** @brief --- Make Code that corresponds to a non-ASCII character.
    For example, the Binary (Non-ASCII) Make Code for
    <tt>Left Alt</tt> is 11
     */
    KB_BINARY_MAKE_CODE = 2,
    /** @brief --- Make Code that has two bytes (the first byte is E0).
    For example, the Long Binary Make Code for <tt>Right Alt</tt>
    is "E0 11"
     */
    KB_LONG_BINARY_MAKE_CODE = 3,
    /** @brief --- Normal Break Code that has two bytes (the first byte is
F0).
    For example, the Break Code for letter <tt>A</tt> is "F0 1C"
     */
    KB_BREAK_CODE = 4,
    /** @brief --- Long Break Code that has three bytes (the first two bytes
    are E0, F0). For example, the Long Break Code for <tt>Right Alt</tt>
    is "E0 F0 11"
     */
    KB_LONG_BREAK_CODE = 5,
    /** @brief --- Codes that the decode FSM cannot decode
     */
    KB_INVALID_CODE = 6
  } KB_CODE_TYPE;

/**
 * @brief Get the make code of the key when a key is pressed
 *
 * @param decode_mode -- indicates which type of code
```

```
 * (Make Code, Break Code, etc.) is received from the keyboard when the
 * key is pressed
 *
 * @param buf -- points to the location that stores the make code of
 *               the key pressed
 * @note For KB_LONG_BINARY_MAKE_CODE and KB_BREAK_CODE, only the
 *       second byte is retured. For KB_LONG_BREAK_CODE, only the
 *       third byte is returned
 *
 * @return \c PS2_TIMEOUT on timeout, or \c PS2_ERROR on error,
 *         otherwise \c PS2_SUCCESS
 **/
extern int read_make_code(KB_CODE_TYPE *decode_mode, alt_u8 *buf);

/**
 * @brief Set the repeat/delay rate of the keyboard
 *
 * @param rate -- an 8-bit number that represents the repeat/delay rate
 *                of the keyboard
 *
 * @return PS2_SUCCESS on success, otherwise PS2_ERROR
 **/
extern alt_u32 set_keyboard_rate(alt_u8 rate);

/**
 * @brief Send the reset command to the keyboard
 *
 * @return \c PS2_SUCCESS on passing the BAT (Basic Assurance Test),
 *         otherwise \c PS2_ERROR
 **/
extern alt_u32 reset_keyboard();

#endif
```

---

ps2_keyboard.c

```
#include "ps2_keyboard.h"

#define NUM_SCAN_CODES  102

/////////////////////////////////////////////////////////////////
// Table of scan code, make code and their corresponding values
// These data are useful for developing more features for the keyboard
//
alt_u8 *key_table[NUM_SCAN_CODES] = {
  "A", "B", "C", "D", "E", "F", "G", "H",
  "I", "J", "K", "L", "M", "N", "O", "P",
  "Q", "R", "S", "T", "U", "V", "W", "X",
```

```
  "Y", "Z", "0", "1", "2", "3", "4", "5",
  "6", "7", "8", "9", "`", "-", "=", "\\",
  "BKSP", "SPACE", "TAB", "CAPS", "L SHFT", "L CTRL", "L GUI", "L ALT",
  "R SHFT", "R CTRL", "R GUI", "R ALT", "APPS", "ENTER", "ESC", "F1",
  "F2", "F3", "F4", "F5", "F6", "F7", "F8", "F9",
  "F10", "F11", "F12", "SCROLL", "[", "INSERT", "HOME", "PG UP",
  "DELETE", "END", "PG DN", "U ARROW", "L ARROW", "D ARROW", "R ARROW", "NUM",
  "KP /", "KP *", "KP -", "KP +", "KP ENTER", "KP .", "KP 0", "KP 1",
  "KP 2", "KP 3", "KP 4", "KP 5", "KP 6", "KP 7", "KP 8", "KP 9",
  "]", ";", "'", ",", ".", "/"
};

alt_u8 ascii_codes[NUM_SCAN_CODES] = {
  'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
  'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
  'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
  'Y', 'Z', '0', '1', '2', '3', '4', '5',
  '6', '7', '8', '9', '`', '-', '=', 0,
  0x08, 0, 0x09, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0x0A, 0x1B,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, '[', 0, 0,
  0, 0x7F, 0, 0, 0, 0, 0, 0,
  0, '/', '*', '-', '+', 0x0A, '.', '0', 'a',//Note: 'a'-'i' represent KP
'1'-'9'
  'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
  ']', ';', '\'', ',', '.', '/'
};

alt_u8 single_byte_make_code[NUM_SCAN_CODES] = {
  0x1C, 0x32, 0x21, 0x23, 0x24, 0x2B, 0x34, 0x33,
  0x43, 0x3B, 0x42, 0x4B, 0x3A, 0x31, 0x44, 0x4D,
  0x15, 0x2D, 0x1B, 0x2C, 0x3C, 0x2A, 0x1D, 0x22,
  0x35, 0x1A, 0x45, 0x16, 0x1E, 0x26, 0x25, 0x2E,
  0x36, 0x3D, 0x3E, 0x46, 0x0E, 0x4E, 0x55, 0x5D,
  0x66, 0x29, 0x0D, 0x58, 0x12, 0x14,    0, 0x11,
  0x59,    0,    0,    0,    0, 0x5A, 0x76, 0x05,
  0x06, 0x04, 0x0C, 0x03, 0x0B, 0x83, 0x0A, 0x01,
  0x09, 0x78, 0x07, 0x7E, 0x54,    0,    0,    0,
  0,    0,    0,    0,    0,    0,    0, 0x77,
  0, 0x7C, 0x7B, 0x79,    0, 0x71, 0x70, 0x69,
  0x72, 0x7A, 0x6B, 0x73, 0x74, 0x6C, 0x75, 0x7D,
  0x5B, 0x4C, 0x52, 0x41, 0x49, 0x4A };

alt_u8 multi_byte_make_code[NUM_SCAN_CODES] = {
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
```

```
  0, 0, 0, 0, 0, 0, 0x1F, 0,
  0, 0x14, 0x27, 0x11, 0x2F, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0x70, 0x6C, 0x7D,
  0x71, 0x69, 0x7A, 0x75, 0x6B, 0x72, 0x74, 0,
  0x4A, 0, 0, 0, 0x5A, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0 };
////////////////////////////////////////////////////////////////////

// States for the Keyboard Decode FSM
typedef enum
  {
    STATE_INIT,
    STATE_LONG_BINARY_MAKE_CODE,
    STATE_BREAK_CODE ,
    STATE_DONE
  } DECODE_STATE;

//helper function for get_next_state
alt_u8 get_multi_byte_make_code_index(alt_u8 code)
{
  alt_u8 i;
  for (i = 0; i < NUM_SCAN_CODES; i++ ) {
    if ( multi_byte_make_code[i] == code )
      return i;
  }
  return NUM_SCAN_CODES;
}

//helper function for get_next_state
alt_u8 get_single_byte_make_code_index(alt_u8 code)
{
  alt_u8 i;
  for (i = 0; i < NUM_SCAN_CODES; i++ ) {
    if ( single_byte_make_code[i] == code )
      return i;
  }
  return NUM_SCAN_CODES;
}

//helper function for read_make_code
/* FSM Diagram (Main transitions)
 * Normal bytes: bytes that are not 0xF0 or 0xE0

 ______
 |     |
 |     |
 |    INIT ------ 0xF0 ----> BREAK CODE
 |     |                      /  |
 |     |                     /   |
```

```
 |    0xE0                 /    |
Normal |                   /    Normal
 |     |      ----0xF0--->      |
 |     V     /                  |
 |    LONG  /                   V
 | MAKE/BREAK --- Normal ----> DONE
 |    CODE                      ^
 X------------------------------|

*/

DECODE_STATE get_next_state(DECODE_STATE state,
               alt_u8 byte,
               KB_CODE_TYPE *decode_mode,
               alt_u8 *buf)
{
  DECODE_STATE next_state = STATE_INIT;
  alt_u16 idx = NUM_SCAN_CODES;
  switch (state) {
  case STATE_INIT:
    if ( byte == 0xE0 ) {
      next_state = STATE_LONG_BINARY_MAKE_CODE;
    } else if (byte == 0xF0) {
      next_state = STATE_BREAK_CODE;
    } else {
      idx = get_single_byte_make_code_index(byte);
      if ( (idx < 40 || idx == 68 || idx > 79) && ( idx != NUM_SCAN_CODES ) )
{
    *decode_mode = KB_ASCII_MAKE_CODE;
    *buf= ascii_codes[idx];
      } else {
    *decode_mode = KB_BINARY_MAKE_CODE;
    *buf = byte;
      }
      next_state = STATE_DONE;
    }
    break;
  case STATE_LONG_BINARY_MAKE_CODE:
    if ( byte != 0xF0 && byte!= 0xE0) {
      *decode_mode = KB_LONG_BINARY_MAKE_CODE;
      *buf = byte;
      next_state = STATE_DONE;
    } else {
      next_state = STATE_BREAK_CODE;
    }
    break;
  case STATE_BREAK_CODE:
    if ( byte != 0xF0 && byte != 0xE0) {
      *decode_mode = KB_BREAK_CODE;
      *buf = byte;
```

```
      next_state = STATE_DONE;
    } else {
      next_state = STATE_BREAK_CODE;
    }
    break;
  default:
    *decode_mode = KB_INVALID_CODE;
    next_state = STATE_INIT;
    break;
  }
  return next_state;
}

int read_make_code(KB_CODE_TYPE *decode_mode, alt_u8 *buf)
{
  alt_u8 byte = 0;
  int status_read =0;
  *decode_mode = KB_INVALID_CODE;
  DECODE_STATE state = STATE_INIT;
  do {
    status_read = read_data_byte_with_timeout(&byte, 0);
    //FIXME: When the user press the keyboard extremely fast, data may get
    //occasionally get lost

    if (status_read == PS2_ERROR)
      return PS2_ERROR;

    state = get_next_state(state, byte, decode_mode, buf);
  } while (state != STATE_DONE);

  return PS2_SUCCESS;
}

alt_u32 set_keyboard_rate(alt_u8 rate)
{
  alt_u8 byte;
  // send the set keyboard rate command
  int status_send = write_data_byte_with_ack(0xF3, DEFAULT_PS2_TIMEOUT_VAL);
  if ( status_send == PS2_SUCCESS ) {
    // we received ACK, so send out the desired rate now
    status_send = write_data_byte_with_ack(rate & 0x1F,
                          DEFAULT_PS2_TIMEOUT_VAL);
  }
  return status_send;
}

alt_u32 reset_keyboard()
{
  alt_u8 byte;
  // send out the reset command
```

```
  int status = write_data_byte_with_ack(0xff, DEFAULT_PS2_TIMEOUT_VAL);
  if ( status == PS2_SUCCESS) {
    // received the ACK for reset, now check the BAT result
    status = read_data_byte_with_timeout(&byte, DEFAULT_PS2_TIMEOUT_VAL);
    if (status == PS2_SUCCESS && byte == 0xAA) {
      // BAT succeed
    } else {
      // BAT failed
      status == PS2_ERROR;
    }
  }
  return status;
}
```

Source Code – Hardware

```
-----------------------------------------------------------------
project.vhd
-----------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity project is
  port (
    -- Clocks

    CLOCK_50 : in std_logic;                        -- 50 MHz

     -- SRAM

    SRAM_DQ : inout std_logic_vector(15 downto 0); -- Data bus 16 Bits
    SRAM_ADDR : out std_logic_vector(17 downto 0); -- Address bus 18 Bits
    SRAM_UB_N,                                      -- High-byte Data Mask
    SRAM_LB_N,                                      -- Low-byte Data Mask
    SRAM_WE_N,                                      -- Write Enable
    SRAM_CE_N,                                      -- Chip Enable
    SRAM_OE_N : out std_logic;                      -- Output Enable

     -- VGA output

    VGA_CLK,                                            -- Clock
    VGA_HS,                                             -- H_SYNC
    VGA_VS,                                             -- V_SYNC
    VGA_BLANK,                                          -- BLANK
    VGA_SYNC : out std_logic;                           -- SYNC
    VGA_R,                                              -- Red[9:0]
    VGA_G,                                              -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0);           -- Blue[9:0]
```

```
     -- PS/2 port

    PS2_DAT,                    -- Data
    PS2_CLK : inout std_logic     -- Clock

    );
end project;

architecture rtl of project is

signal counter : unsigned(15 downto 0);
signal reset_n : std_logic;

begin

process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      if counter = x"ffff" then
        reset_n <= '1';
      else
        reset_n <= '0';
        counter <= counter + 1;
      end if;
    end if;
  end process;

  nios : entity work.nios_system port map (
    clk_0                          => CLOCK_50,
    reset_n                        => reset_n,
    SRAM_ADDR_from_the_sram        => SRAM_ADDR,
    SRAM_CE_N_from_the_sram        => SRAM_CE_N,
    SRAM_DQ_to_and_from_the_sram   => SRAM_DQ,
    SRAM_LB_N_from_the_sram        => SRAM_LB_N,
    SRAM_OE_N_from_the_sram        => SRAM_OE_N,
    SRAM_UB_N_from_the_sram        => SRAM_UB_N,
    SRAM_WE_N_from_the_sram        => SRAM_WE_N,

     VGA_CLK_from_the_vga_raster   => VGA_CLK,
    VGA_HS_from_the_vga_raster     => VGA_HS,
    VGA_VS_from_the_vga_raster     => VGA_VS,
    VGA_BLANK_from_the_vga_raster => VGA_BLANK,
    VGA_SYNC_from_the_vga_raster   => VGA_SYNC,
    VGA_R_from_the_vga_raster      => VGA_R,
    VGA_G_from_the_vga_raster      => VGA_G,
    VGA_B_from_the_vga_raster      => VGA_B,

     PS2_CLK_to_and_from_the_ALT_UP_PS2_0   => PS2_CLK,
     PS2_DAT_to_and_from_the_ALT_UP_PS2_0   => PS2_DAT
```

```
    );

end rtl;

--------------------------------------------------------------------------
acceleration_block.vhd
--------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity acceleration_block is
    port(
           clk        : in  std_logic;
           reset_n          : in  std_logic;
           read       : in  std_logic;
           write      : in  std_logic;
           chipselect : in  std_logic;
           address          : in  std_logic_vector(4 downto 0);
           readdata   : out std_logic_vector(7 downto 0);
           writedata  : in  std_logic_vector(7 downto 0)
    );
end acceleration_block;

architecture implementation of acceleration_block is

  signal cpu_addr_m     : std_logic;
  signal cpu_ram_wren   : std_logic;
  signal cpu_ram_rden   : std_logic;
  signal reg_m_en       : std_logic;
  signal reg_m_reset    : std_logic;
  signal reg_m_start    : std_logic;
  signal reg_m          : std_logic_vector(2 downto 0);

  signal ram_wren_a      : std_logic;
  signal ram_addr_a      : std_logic_vector(4 downto 0);
  signal ram_from_data_a : std_logic_vector(6 downto 0);
  signal ram_to_data_a   : std_logic_vector(6 downto 0);
  signal ram_wren_b      : std_logic;
  signal ram_addr_b      : std_logic_vector(4 downto 0);
  signal ram_from_data_b : std_logic_vector(6 downto 0);
  signal ram_to_data_b   : std_logic_vector(6 downto 0);

  signal domove_addr_gen_a : std_logic_vector(4 downto 0);
  signal domove_addr_gen_b : std_logic_vector(4 downto 0);

  signal perm_addr_gen     : std_logic_vector(4 downto 0);

  signal fsm_mux_sel     : std_logic;
```

```
  signal fsm_reg_en      : std_logic;
  signal fsm_ram_wren_a  : std_logic;
  signal fsm_ram_wren_b  : std_logic;
  signal fsm_cpu_en      : std_logic;
  signal fsm_cnt_en      : std_logic;
  signal   fsm_cnt_reset   : std_logic;
  signal   fsm_perm_sel    : std_logic;
  signal   fsm_perm_wren   : std_logic;
  signal   fsm_perm_addr   : std_logic_vector(1 downto 0);
  signal fsm_step        : std_logic_vector(2 downto 0);

  signal twist_data_a : std_logic_vector(6 downto 0);
  signal twist_data_b : std_logic_vector(6 downto 0);

  signal mux_data_a : std_logic_vector(6 downto 0);
  signal mux_data_b : std_logic_vector(6 downto 0);

  signal reg_data_a : std_logic_vector(6 downto 0);
  signal reg_data_b : std_logic_vector(6 downto 0);

  signal binary_cnt_int : integer range 0 to 31;
  signal binary_cnt     : std_logic_vector(4 downto 0);

  signal perm_addr      : std_logic_vector(4 downto 0);
  signal permtonum_out  : std_logic_vector(4 downto 0);

begin

  cpu_addr_m <= address(4) and not address(3) and address(2) and not
address(1) and not address(0);
  cpu_ram_wren <= chipselect and write and not read and reset_n;
  cpu_ram_rden <= chipselect and not write and read and reset_n;

  with fsm_cpu_en select reg_m_en <=
    reg_m_reset                  when '0',
    cpu_addr_m and cpu_ram_wren   when others;

  reg_m_reset <= fsm_step(2) and fsm_step(1) and fsm_step(0);
  reg_m_start <= reg_m(2) and reg_m(1) and reg_m(0);

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        reg_m <= "111";
      elsif reg_m_en = '1' then
        reg_m <= writedata(2 downto 0);
        if reg_m_reset = '1' then
          reg_m <= "111";
        end if;
```

```
    end if;
  end if;
end process;

ram_wren_a <= fsm_ram_wren_a;

with fsm_perm_wren select perm_addr <=
  binary_cnt        when '0',
  perm_addr_gen     when others;

with fsm_perm_sel select ram_addr_a <=
  domove_addr_gen_a when '0',
  perm_addr         when others;

with fsm_perm_sel select ram_to_data_a <=
  mux_data_a             when '0',
  "00" & permtonum_out   when others;

with fsm_cpu_en select ram_wren_b <=
  fsm_ram_wren_b when '0',
  cpu_ram_wren   when others;

with fsm_cpu_en and reg_m_start select ram_addr_b <=
  domove_addr_gen_b when '0',
  address           when others;

with fsm_cpu_en select ram_to_data_b <=
  mux_data_b            when '0',
  writedata(6 downto 0) when others;

with cpu_ram_rden select readdata <=
  '0' & ram_from_data_b when '1',
  "11111111"            when others;

process (clk)
begin
  if rising_edge(clk) then
    if fsm_reg_en = '1' then
      reg_data_a <= twist_data_a;
      reg_data_b <= twist_data_b;
    end if;
  end if;
end process;

with fsm_mux_sel select mux_data_a <=
  twist_data_a when '0',
  reg_data_a   when others;

with fsm_mux_sel select mux_data_b <=
  twist_data_b when '0',
```

```
  reg_data_b   when others;

DUAL_PORT_RAM: entity work.true_dual_port_ram_single_clock port map(

    clock => clk,

    address_a => ram_addr_a,
    data_a    => ram_to_data_a,
    wren_a    => ram_wren_a,
    q_a       => ram_from_data_a,

    address_b => ram_addr_b,
    data_b    => ram_to_data_b,
    wren_b    => ram_wren_b,
    q_b       => ram_from_data_b
);

FSM: entity work.moore_state_machine port map(

   clk          => clk,
   reset_n      => reset_n,
   m            => reg_m,
    cnt          => binary_cnt,
    cnt_en       => fsm_cnt_en,
    cnt_reset    => fsm_cnt_reset,
    perm_sel     => fsm_perm_sel,
    perm_wren    => fsm_perm_wren,
    perm_addr    => fsm_perm_addr,
   mux_sel      => fsm_mux_sel,
   reg_en       => fsm_reg_en,
   ram_wren_a   => fsm_ram_wren_a,
   ram_wren_b   => fsm_ram_wren_b,
   cpu_en       => fsm_cpu_en,
   step         => fsm_step

);

TWIST: entity work.twist port map(

   m          => reg_m,
   step       => fsm_step,
   data_a_in  => ram_from_data_a,
   data_b_in  => ram_from_data_b,
   data_a_out => twist_data_a,
   data_b_out => twist_data_b

);

DOMOVE_ADDRESS_GENERATOR: entity work.domove_addr_gen port map(
```

```
      m          => reg_m,
      step       => fsm_step,
      addr_a     => domove_addr_gen_a,
      addr_b     => domove_addr_gen_b

  );

  PERM_ADDRESS_GENERATOR: entity work.perm_addr_gen port map(

      addr_sel   => binary_cnt(4 downto 2),
       addr_out   => perm_addr_gen

  );

  binary_cnt <= std_logic_vector(to_unsigned(binary_cnt_int, 5));

  BINARY_COUNTER: entity work.binary_counter port map(

       clk        => clk,
      reset      => fsm_cnt_reset,
      enable     => fsm_cnt_en,
      q          => binary_cnt_int

  );

  PERMTONUM: entity work.permtonum port map(

      clk        => clk,
      data       => ram_from_data_a(6 downto 2),
      addr       => fsm_perm_addr,
      q          => permtonum_out
  );

end implementation;
```

---

binary_counter.vhd

```
-- Quartus II VHDL Template
-- Binary Counter

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity binary_counter is

      generic
```

```
	(
		MIN_COUNT : natural := 0;
		MAX_COUNT : natural := 31
	);

	port
	(
		clk	: in std_logic;
		reset	: in std_logic;
		enable	: in std_logic;
		q		: out integer range MIN_COUNT to MAX_COUNT
	);

end entity;

architecture rtl of binary_counter is
begin

	process (clk)
		variable	cnt		: integer range MIN_COUNT to MAX_COUNT;
	begin
		if (rising_edge(clk)) then

			if reset = '1' then
				-- Reset the counter to 0
				cnt := 0;

			elsif enable = '1' then
				-- Increment the counter if counting is enabled
				cnt := cnt + 1;

			end if;
		end if;

		-- Output the current count
		q <= cnt;
	end process;

end rtl;
```

```
--------------------------------------------------------------------
domove_addr_gen.vhd
--------------------------------------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity domove_addr_gen is
	port(
			m	: in std_logic_vector(2 downto 0);
			step	: in std_logic_vector(2 downto 0);
			addr_a: out std_logic_vector(4 downto 0);
			addr_b: out std_logic_vector(4 downto 0)
	);
end domove_addr_gen;

architecture implementation of domove_addr_gen is

 type ram_type is array(31 downto 0) of
  std_logic_vector(4 downto 0);
 signal RAM1 : ram_type := (
  "00000", "00000", "00000", "00000", "00000", "00000", "00000", "00000",
  "00001", "00101", "00011", "00111", "00000", "00110", "00010", "00100",
  "00110", "01001", "00111", "01011", "00100", "01010", "00101", "01000",
  "00010", "01011", "00011", "01010", "00000", "01000", "00001", "01001");
 signal RAM2 : ram_type := (
  "00000", "00000", "00000", "00000", "00000", "00000", "00000", "00000",
  "01110", "10001", "01111", "10000", "01100", "10011", "01101", "10010",
  "01101", "10011", "01110", "10000", "01100", "10010", "01111", "10001",
  "10010", "01101", "10000", "01111", "10011", "01100", "10001", "01110");
 signal addr : std_logic_vector(4 downto 0);

begin

 addr(4 downto 2) <= m;
 addr(1) <= (step(2) and step(1)) or
	(step(1) and step(0)) or
	    (step(2) and step(0));
 addr(0) <= (step(2) and not step(1)) or
	(step(2) and step(0)) or
	    (not step(1) and step(0));

 addr_a <= RAM1(to_integer(unsigned(addr)));
 addr_b <= RAM2(to_integer(unsigned(addr)));

end implementation;
```

---

moore_state_machine.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity moore_state_machine is
	port(
		clk		: in  std_logic;
		reset_n	: in  std_logic;
		m		: in  std_logic_vector(2 downto 0);
		cnt		: in  std_logic_vector(4 downto 0);
		cnt_en	: out std_logic;
		cnt_reset	: out std_logic;
		perm_sel	: out std_logic;
		perm_wren	: out std_logic;
		perm_addr	: out std_logic_vector(1 downto 0);
		mux_sel	: out std_logic;
		reg_en	: out std_logic;
		ram_wren_a	: out std_logic;
		ram_wren_b	: out std_logic;
		cpu_en	: out std_logic;
		step	: out std_logic_vector(2 downto 0)
	);
end moore_state_machine;

architecture rtl of moore_state_machine is

 type state_type is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13);
 signal state : state_type;

begin

	process (clk, reset_n)
	begin
		if reset_n = '0' then
			state <= s0;
		elsif rising_edge(clk) then
			case state is
				when s0=>
					if m = "111" then
						state <= s0;
					else
						state <= s1;
					end if;
				when s1=>
					state <= s2;
				when s2=>
					state <= s3;
```

```
when s3 =>
        state <= s4;
when s4 =>
        state <= s5;
when s5 =>
        state <= s6;
when s6 =>
        state <= s7;
when s7 =>
        state <= s8;
when s8=>
        if m = "111" then
                state <= s9;
        else
                state <= s1;
        end if;
when s9=>
        if m = "111" then
                state <= s10;
        else
                state <= s1;
        end if;
when s10=>
        if m = "111" then
                state <= s11;
        else
                state <= s1;
        end if;
when s11=>
        if m = "111" then
                state <= s12;
        else
                state <= s1;
        end if;
when s12=>
        if m = "111" then
                state <= s13;
        else
                state <= s1;
        end if;
when s13=>
  if cnt = "10011" then
          state <= s0;
        elsif m = "111" then
                state <= s8;
        else
```

```
                                        state <= s1;
                                    end if;
                        end case;
                end if;
    end process;

    process (state)
    begin
                case state is
                        when s0 =>
                          cnt_en <= '0';
                    cnt_reset <= '0';
                    perm_sel <= '0';
                            perm_wren <= '0';
                            perm_addr <= "00";
                            mux_sel <= '0';
            reg_en <= '0';
                            ram_wren_a <= '0';
                            ram_wren_b <= '0';
                            cpu_en <= '1';
step <= "000";
                        when s1 =>
                          cnt_en <= '0';
                    cnt_reset <= '0';
                    perm_sel <= '0';
                            perm_wren <= '0';
                            perm_addr <= "00";
                            mux_sel <= '0';
            reg_en <= '1';
                            ram_wren_a <= '0';
                            ram_wren_b <= '0';
                            cpu_en <= '0';
step <= "001";
                        when s2 =>
                          cnt_en <= '0';
                    cnt_reset <= '0';
                    perm_sel <= '0';
                            perm_wren <= '0';
                            perm_addr <= "00";
                            mux_sel <= '0';
            reg_en <= '0';
                            ram_wren_a <= '1';
                            ram_wren_b <= '1';
                            cpu_en <= '0';
step <= "010";
                        when s3 =>
```

```
                cnt_en <= '0';
            cnt_reset <= '0';
            perm_sel <= '0';
                perm_wren <= '0';
                perm_addr <= "00";
                mux_sel <= '0';
        reg_en <= '0';
                ram_wren_a <= '0';
                ram_wren_b <= '0';
                cpu_en <= '0';
step <= "011";
              when s4 =>
                cnt_en <= '0';
            cnt_reset <= '0';
            perm_sel <= '0';
                perm_wren <= '0';
                perm_addr <= "00";
                mux_sel <= '0';
        reg_en <= '0';
                ram_wren_a <= '1';
                ram_wren_b <= '1';
                cpu_en <= '0';
step <= "100";
              when s5 =>
                cnt_en <= '0';
            cnt_reset <= '0';
            perm_sel <= '0';
                perm_wren <= '0';
                perm_addr <= "00";
                mux_sel <= '0';
        reg_en <= '0';
                ram_wren_a <= '0';
                ram_wren_b <= '0';
                cpu_en <= '0';
step <= "101";
              when s6 =>
                cnt_en <= '0';
            cnt_reset <= '0';
            perm_sel <= '0';
                perm_wren <= '0';
                perm_addr <= "00";
                mux_sel <= '0';
        reg_en <= '0';
                ram_wren_a <= '1';
                ram_wren_b <= '1';
                cpu_en <= '0';
```

```
step <= "110";
                    when s7 =>
                     cnt_en <= '0';
              cnt_reset <= '1';
              perm_sel <= '0';
                     perm_wren <= '0';
                     perm_addr <= "00";
                     mux_sel <= '1';
       reg_en <= '0';
                     ram_wren_a <= '1';
                     ram_wren_b <= '1';
                     cpu_en <= '0';
step <= "111";
                    when s8 =>
                     cnt_en <= '1';
              cnt_reset <= '0';
              perm_sel <= '1';
                     perm_wren <= '0';
                     perm_addr <= "00";
                     mux_sel <= '0';
       reg_en <= '0';
                     ram_wren_a <= '0';
                     ram_wren_b <= '0';
                     cpu_en <= '1';
step <= "000";
                    when s9 =>
                     cnt_en <= '1';
              cnt_reset <= '0';
              perm_sel <= '1';
                     perm_wren <= '0';
                     perm_addr <= "00";
                     mux_sel <= '0';
       reg_en <= '0';
                     ram_wren_a <= '0';
                     ram_wren_b <= '0';
                     cpu_en <= '1';
step <= "000";
                    when s10 =>
                     cnt_en <= '1';
              cnt_reset <= '0';
              perm_sel <= '1';
                     perm_wren <= '0';
                     perm_addr <= "01";
                     mux_sel <= '0';
       reg_en <= '0';
                     ram_wren_a <= '0';
```

```
                        ram_wren_b <= '0';
                        cpu_en <= '1';
        step <= "000";
                       when s11 =>
                        cnt_en <= '0';
                    cnt_reset <= '0';
                    perm_sel <= '1';
                        perm_wren <= '0';
                        perm_addr <= "10";
                        mux_sel <= '0';
            reg_en <= '0';
                        ram_wren_a <= '0';
                        ram_wren_b <= '0';
                        cpu_en <= '1';
        step <= "000";
                       when s12 =>
                        cnt_en <= '0';
                    cnt_reset <= '0';
                    perm_sel <= '1';
                        perm_wren <= '0';
                        perm_addr <= "11";
                        mux_sel <= '0';
            reg_en <= '0';
                        ram_wren_a <= '0';
                        ram_wren_b <= '0';
                        cpu_en <= '1';
        step <= "000";
                       when s13 =>
                        cnt_en <= '1';
                    cnt_reset <= '0';
                    perm_sel <= '1';
                        perm_wren <= '1';
                        perm_addr <= "00";
                        mux_sel <= '0';
            reg_en <= '0';
                        ram_wren_a <= '1';
                        ram_wren_b <= '0';
                        cpu_en <= '1';
        step <= "000";
                end case;
      end process;

end rtl;
```

---

perm_addr_gen.vhd

```
--------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity perm_addr_gen is
        port(
                        addr_sel : in std_logic_vector(2 downto 0);
                        addr_out        : out std_logic_vector(4 downto 0)
        );
end perm_addr_gen;

architecture implementation of perm_addr_gen is

 type ram_type is array(7 downto 0) of
  std_logic_vector(4 downto 0);
 signal RAM : ram_type := (
  "11100", "11011", "11010", "11001", "11000", "10111", "10110", "10101"
         );

begin

 addr_out <= RAM(to_integer(unsigned(addr_sel)));

end implementation;

--------------------------------------------------------------------
permtonum.vhd
--------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity permtonum is
        port(
                        clk             : in std_logic;
                        data    : in std_logic_vector(4 downto 0);
                        addr    : in std_logic_vector(1 downto 0);
                        q       : out std_logic_vector(4 downto 0)
        );
end permtonum;

architecture implementation of permtonum is

 signal latch0    : std_logic_vector(4 downto 0);
```

```
  signal latch1    : std_logic_vector(4 downto 0);
  signal latch2    : std_logic_vector(4 downto 0);
  signal latch3    : std_logic_vector(4 downto 0);
  signal latch0_en  : std_logic;
  signal latch1_en  : std_logic;
  signal latch2_en  : std_logic;
  signal latch3_en  : std_logic;
  signal comp      : std_logic_vector(5 downto 0);

begin

  latch0_en <= clk and not addr(1) and not addr(0);
  latch1_en <= clk and not addr(1) and     addr(0);
  latch2_en <= clk and     addr(1) and not addr(0);
  latch3_en <= clk and     addr(1) and     addr(0);

  process (latch0_en, data)
  begin
   if latch0_en = '1' then
     latch0 <= data;
   end if;
  end process;

  process (latch1_en, data)
  begin
   if latch1_en = '1' then
     latch1 <= data;
           end if;
  end process;

  process (latch2_en, data)
  begin
   if latch2_en = '1' then
     latch2 <= data;
           end if;
  end process;

  process (latch3_en, data)
  begin
   if latch3_en = '1' then
     latch3 <= data;
           end if;
  end process;

  process (latch1, latch0)
  begin
```

```
  if latch1 < latch0 then
    comp(5) <= '1';
          else
            comp(5) <= '0';
  end if;
end process;

process (latch2, latch0)
begin
  if latch2 < latch0 then
    comp(4) <= '1';
          else
            comp(4) <= '0';
  end if;
end process;

process (latch3, latch0)
begin
  if latch3 < latch0 then
    comp(3) <= '1';
          else
            comp(3) <= '0';
  end if;
end process;

process (latch2, latch1)
begin
  if latch2 < latch1 then
    comp(2) <= '1';
          else
            comp(2) <= '0';
  end if;
end process;

process (latch3, latch1)
begin
  if latch3 < latch1 then
    comp(1) <= '1';
          else
            comp(1) <= '0';
  end if;
end process;

process (latch3, latch2)
begin
  if latch3 < latch2 then
```

```
    comp(0) <= '1';
          else
            comp(0) <= '0';
   end if;
 end process;

 ROM: entity work.single_port_rom port map(

   clk  => clk,
   addr => to_integer(unsigned(comp)),
   q    => q

 );

end implementation;

--------------------------------------------------------------------
true_dual_port_ram_single_clock.vhd
--------------------------------------------------------------------

-- megafunction wizard: %RAM: 2-PORT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram

-- ============================================================
-- File Name: true_dual_port_ram_single_clock.vhd
-- Megafunction Name(s):
--                         altsyncram
--
-- Simulation Library Files(s):
--                         altera_mf
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 12.1 Build 177 11/07/2012 SJ Full Version
-- ************************************************************


--Copyright (C) 1991-2012 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
```

```
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY true_dual_port_ram_single_clock IS
	PORT
	(
		address_a	: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		address_b	: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		data_a		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		data_b		: IN STD_LOGIC_VECTOR (6 DOWNTO 0);
		wren_a			: IN STD_LOGIC  := '0';
		wren_b			: IN STD_LOGIC  := '0';
		q_a		: OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
		q_b		: OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
	);
END true_dual_port_ram_single_clock;


ARCHITECTURE SYN OF true_dual_port_ram_single_clock IS

	SIGNAL sub_wire0	: STD_LOGIC_VECTOR (6 DOWNTO 0);
	SIGNAL sub_wire1	: STD_LOGIC_VECTOR (6 DOWNTO 0);



	COMPONENT altsyncram
	GENERIC (
		address_reg_b			: STRING;
		clock_enable_input_a		: STRING;
		clock_enable_input_b		: STRING;
		clock_enable_output_a			: STRING;
		clock_enable_output_b			: STRING;
		indata_reg_b		: STRING;
```

```
		intended_device_family		: STRING;
		lpm_type		: STRING;
		numwords_a		: NATURAL;
		numwords_b		: NATURAL;
		operation_mode		: STRING;
		outdata_aclr_a		: STRING;
		outdata_aclr_b		: STRING;
		outdata_reg_a		: STRING;
		outdata_reg_b		: STRING;
		power_up_uninitialized		: STRING;
		read_during_write_mode_mixed_ports		: STRING;
		widthad_a		: NATURAL;
		widthad_b		: NATURAL;
		width_a		: NATURAL;
		width_b		: NATURAL;
		width_byteena_a		: NATURAL;
		width_byteena_b		: NATURAL;
		wrcontrol_wraddress_reg_b		: STRING
	);
	PORT (
			clock0 : IN STD_LOGIC ;
			wren_a		: IN STD_LOGIC ;
			address_b		: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
			data_b : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
			q_a		: OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
			wren_b		: IN STD_LOGIC ;
			address_a		: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
			data_a : IN STD_LOGIC_VECTOR (6 DOWNTO 0);
			q_b		: OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
	);
	END COMPONENT;

BEGIN
	q_a	<= sub_wire0(6 DOWNTO 0);
	q_b	<= sub_wire1(6 DOWNTO 0);

	altsyncram_component : altsyncram
	GENERIC MAP (
		address_reg_b => "CLOCK0",
		clock_enable_input_a => "BYPASS",
		clock_enable_input_b => "BYPASS",
		clock_enable_output_a => "BYPASS",
		clock_enable_output_b => "BYPASS",
		indata_reg_b => "CLOCK0",
		intended_device_family => "Cyclone II",
		lpm_type => "altsyncram",
```

```
		numwords_a => 32,
		numwords_b => 32,
		operation_mode => "BIDIR_DUAL_PORT",
		outdata_aclr_a => "NONE",
		outdata_aclr_b => "NONE",
		outdata_reg_a => "UNREGISTERED",
		outdata_reg_b => "UNREGISTERED",
		power_up_uninitialized => "FALSE",
		read_during_write_mode_mixed_ports => "DONT_CARE",
		widthad_a => 5,
		widthad_b => 5,
		width_a => 7,
		width_b => 7,
		width_byteena_a => 1,
		width_byteena_b => 1,
		wrcontrol_wraddress_reg_b => "CLOCK0"
	)
	PORT MAP (
		clock0 => clock,
		wren_a => wren_a,
		address_b => address_b,
		data_b => data_b,
		wren_b => wren_b,
		address_a => address_a,
		data_a => data_a,
		q_a => sub_wire0,
		q_b => sub_wire1
	);



END SYN;

-- ============================================================
-- CNX file retrieval info
-- ============================================================
-- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
-- Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
-- Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
-- Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "1"
-- Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
```

-- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
-- Retrieval info: PRIVATE: CLRdata NUMERIC "0"
-- Retrieval info: PRIVATE: CLRq NUMERIC "0"
-- Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
-- Retrieval info: PRIVATE: CLRrren NUMERIC "0"
-- Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
-- Retrieval info: PRIVATE: CLRwren NUMERIC "0"
-- Retrieval info: PRIVATE: Clock NUMERIC "0"
-- Retrieval info: PRIVATE: Clock_A NUMERIC "0"
-- Retrieval info: PRIVATE: Clock_B NUMERIC "0"
-- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
-- Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
-- Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "1"
-- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
-- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
-- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
-- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
-- Retrieval info: PRIVATE: MEMSIZE NUMERIC "224"
-- Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
-- Retrieval info: PRIVATE: MIFfilename STRING ""
-- Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "3"
-- Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
-- Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
-- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
-- Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
-- Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
-- Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
-- Retrieval info: PRIVATE: REGdata NUMERIC "1"
-- Retrieval info: PRIVATE: REGq NUMERIC "0"
-- Retrieval info: PRIVATE: REGrdaddress NUMERIC "0"
-- Retrieval info: PRIVATE: REGrren NUMEsignal from_data_b : std_logic_vector(6 downto 0);RIC "0"
-- Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
-- Retrieval info: PRIVATE: REGwren NUMERIC "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
-- Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
-- Retrieval info: PRIVATE: VarWidth NUMERIC "0"
-- Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "7"
-- Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "7"
-- Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "7"
-- Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "7"
-- Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
-- Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "1"

-- Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
-- Retrieval info: PRIVATE: enable NUMERIC "0"
-- Retrieval info: PRIVATE: rden NUMERIC "0"
-- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
-- Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
-- Retrieval info: CONSTANT: INDATA_REG_B STRING "CLOCK0"
-- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
-- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "32"
-- Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "32"
-- Retrieval info: CONSTANT: OPERATION_MODE STRING "BIDIR_DUAL_PORT"
-- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
-- Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
-- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "UNREGISTERED"
-- Retrieval info: CONSTANT: OUTDATA_REG_B STRING "UNREGISTERED"
-- Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
-- Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
-- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "5"
-- Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "5"
-- Retrieval info: CONSTANT: WIDTH_A NUMERIC "7"
-- Retrieval info: CONSTANT: WIDTH_B NUMERIC "7"
-- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
-- Retrieval info: CONSTANT: WIDTH_BYTEENA_B NUMERIC "1"
-- Retrieval info: CONSTANT: WRCONTROL_WRADDRESS_REG_B STRING "CLOCK0"
-- Retrieval info: USED_PORT: address_a 0 0 5 0 INPUT NODEFVAL "address_a[4..0]"
-- Retrieval info: USED_PORT: address_b 0 0 5 0 INPUT NODEFVAL "address_b[4..0]"
-- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
-- Retrieval info: USED_PORT: data_a 0 0 7 0 INPUT NODEFVAL "data_a[6..0]"
-- Retrieval info: USED_PORT: data_b 0 0 7 0 INPUT NODEFVAL "data_b[6..0]"
-- Retrieval info: USED_PORT: q_a 0 0 7 0 OUTPUT NODEFVAL "q_a[6..0]"
-- Retrieval info: USED_PORT: q_b 0 0 7 0 OUTPUT NODEFVAL "q_b[6..0]"
-- Retrieval info: USED_PORT: wren_a 0 0 0 0 INPUT GND "wren_a"
-- Retrieval info: USED_PORT: wren_b 0 0 0 0 INPUT GND "wren_b"
-- Retrieval info: CONNECT: @address_a 0 0 5 0 address_a 0 0 5 0
-- Retrieval info: CONNECT: @address_b 0 0 5 0 address_b 0 0 5 0
-- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
-- Retrieval info: CONNECT: @data_a 0 0 7 0 data_a 0 0 7 0
-- Retrieval info: CONNECT: @data_b 0 0 7 0 data_b 0 0 7 0
-- Retrieval info: CONNECT: @wren_a 0 0 0 0 wren_a 0 0 0 0
-- Retrieval info: CONNECT: @wren_b 0 0 0 0 wren_b 0 0 0 0
-- Retrieval info: CONNECT: q_a 0 0 7 0 @q_a 0 0 7 0
-- Retrieval info: CONNECT: q_b 0 0 7 0 @q_b 0 0 7 0

```
-- Retrieval info: GEN_FILE: TYPE_NORMAL true_dual_port_ram_single_clock.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL true_dual_port_ram_single_clock.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL true_dual_port_ram_single_clock.cmp TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL true_dual_port_ram_single_clock.bsf FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL true_dual_port_ram_single_clock_inst.vhd FALSE
-- Retrieval info: LIB_FILE: altera_mf
```

```
-----------------------------------------------------------------
twist.vhd
-----------------------------------------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;

entity twist is
	port(
			m		: in std_logic_vector(2 downto 0);
			step		: in std_logic_vector(2 downto 0);
			data_a_in	: in std_logic_vector(6 downto 0);
			data_b_in	: in std_logic_vector(6 downto 0);
			data_a_out	: out std_logic_vector(6 downto 0);
			data_b_out	: out std_logic_vector(6 downto 0)
	);
end twist;

architecture implementation of twist is

signal ori_a : std_logic_vector(1 downto 0);
signal ori_b : std_logic_vector(1 downto 0);

begin

 data_a_out(6 downto 2) <= data_a_in(6 downto 2);
 data_b_out(6 downto 2) <= data_b_in(6 downto 2);

 with step select ori_a <=
  '0' & not data_a_in(0) when "001",
  '0' & not data_a_in(0) when "010",
  '0' & not data_a_in(0) when "100",
  '0' & not data_a_in(0) when "110",
  "00" when others;

 with m select data_a_out(1 downto 0) <=
  ori_a		when "000",
  ori_a		when "001",
  data_a_in(1 downto 0) when others;
```

```
 with step select ori_b <=
  data_b_in(0) & (not data_b_in(1) and not data_b_in(0)) when "001",
  (not data_b_in(1) and not data_b_in(0)) & data_b_in(1) when "010",
  data_b_in(0) & (not data_b_in(1) and not data_b_in(0)) when "100",
  (not data_b_in(1) and not data_b_in(0)) & data_b_in(1) when "110",
  "00" when others;

 with m select data_b_out(1 downto 0) <=
  ori_b            when "000",
  ori_b            when "001",
  ori_b            when "010",
  ori_b            when "011",
  data_b_in(1 downto 0) when others;

end implementation;

--------------------------------------------------------------------
de2_vga_raster.vhd
--------------------------------------------------------------------

library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;


entity de2_vga_raster is


 port (

  clk        : in  std_logic;

  reset_n    : in  std_logic;

  write      : in  std_logic;

  chipselect : in  std_logic;

  address    : in  std_logic_vector(7 downto 0);

  writedata  : in  std_logic_vector(7 downto 0);
```

```
    VGA_CLK,                    -- Clock
    VGA_HS,                     -- H_SYNC
    VGA_VS,                     -- V_SYNC
    VGA_BLANK,                  -- BLANK
    VGA_SYNC : out std_logic;   -- SYNC
    VGA_R,                      -- Red[9:0]
    VGA_G,                      -- Green[9:0]
    VGA_B : out std_logic_vector(9 downto 0) -- Blue[9:0]
    );
end de2_vga_raster;

architecture rtl of de2_vga_raster is

  -- Video parameters

  constant HTOTAL      : integer := 800;
  constant HSYNC       : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE     : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL      : integer := 525;
```

```
constant VSYNC       : integer := 2;

constant VBACK_PORCH  : integer := 33;

constant VACTIVE     : integer := 480;

constant VFRONT_PORCH : integer := 10;



constant CUBIE_HEIGHT   : integer := 36;

constant V1RegionOffset : integer := 60;
constant V2RegionOffset : integer := 259;
constant V3RegionOffset : integer := 176;
constant V4RegionOffset : integer := 144;
constant V5RegionOffset : integer := 160;
constant V6RegionOffset : integer := 479;



-- Signals for the video controller

signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)

signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)

signal MyHcount : unsigned(9 downto 0);

signal MyVcount : unsigned(9 downto 0);

signal EndOfLine, EndOfField : std_logic;

signal vga_hblank, vga_hsync, vga_vblank, vga_vsync : std_logic;  -- Sync. signals

signal clk25 : std_logic := '0';



signal HRegion          : std_logic_vector(11 downto 0);
signal V1Region, V2Region : std_logic_vector(6  downto 0);
signal V3Region, V4Region : std_logic_vector(3  downto 0);
signal V5Region, V6Region : std_logic_vector(3  downto 0);
signal face             : std_logic_vector(2  downto 0);
signal position         : std_logic_vector(3  downto 0);
```

```
  signal OneHot          : std_logic_vector(63 downto 0);


  signal SBorderHRegion              : std_logic_vector(7 downto 0);
  signal SBorderV1Region, SBorderV2Region : std_logic_vector(3 downto 0);
  signal SBorderV3Region, SBorderV4Region : std_logic_vector(1 downto 0);
  signal SBorderV5Region, SBorderV6Region : std_logic_vector(1 downto 0);
  signal SBorder                     : std_logic_vector(5 downto 0);

  signal ram_address    : unsigned(5 downto 0);

  signal raddr            : integer range 0 to 63;
  signal raddr_reg1     : integer range 0 to 63;
  signal raddr_reg2     : integer range 0 to 63;
  signal lines            : std_logic;
  signal color            : std_logic_vector(2 downto 0);
  signal border_reg     : std_logic_vector(5 downto 0);
  signal Face4line, Face4hline1, Face4hline2, Face4hline4          : std_logic;
  signal         Face4vline1, Face4vline2, Face4vline3, Face4vline4 : std_logic;
  signal Face5line, Face5hline1, Face5hline2, Face5hline4          : std_logic;
  signal         Face5vline1, Face5vline2, Face5vline3, Face5vline4 : std_logic;
  signal Face6line, Face6hline1, Face6hline2, Face6hline3          : std_logic;
  signal         Face6vline1, Face6vline2                    : std_logic;


  signal raddr_text     : unsigned(7 downto 0);
  signal text_addr      : std_logic_vector(5 downto 0);
  signal text_data      : std_logic_vector(7 downto 0);
  signal ram_selection, text_display, text_region1      : std_logic;
  signal text_region23_v, text_region2_h, text_region3_h : std_logic;
  signal subtraction_result     : std_logic_vector(6 downto 0);
  signal text_region          : std_logic;


begin


  ram_address <= unsigned(address(5 downto 0));

  ram_selection <= address(7) or address(6);

  RAM_COLOR: entity work.simple_dual_port_ram_single_clock_color port map(

    clk        => clk,
```

```
  raddr       => raddr,
  waddr       => to_integer(ram_address),
  data        => writedata(2 downto 0),
  we          => chipselect and write and not ram_selection,
  q           => color
);
RAM_TEXT: entity work.simple_dual_port_ram_single_clock_text port map(
  clk         => clk,
  raddr       => to_integer(raddr_text),
  waddr       => to_integer(unsigned(address(7 downto 0))) - 64,
  data        => writedata(5 downto 0),
  we          => chipselect and write and ram_selection,
  q           => text_addr
);
CHAR_ROM: entity work.char_rom port map (
 clock        => clk,
 address      => text_addr(5 downto 0) & std_logic_vector(MyVcount(2 downto 0)),
 q            => text_data
);


--25M clk


process (clk)
begin
```

```
  if rising_edge(clk) then
    clk25 <= not clk25;
  end if;
end process;

-- Horizontal and vertical counters

HCounter : process (clk25)
begin
  if rising_edge(clk25) then
    if reset_n = '0' then
      Hcount <= (others => '0');
    elsif EndOfLine = '1' then
      Hcount <= (others => '0');
    else
      Hcount <= Hcount + 1;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk25)
```

```
begin
  if rising_edge(clk25) then
    if reset_n = '0' then
      Vcount <= (others => '0');
    elsif EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk25)
begin
  if rising_edge(clk25) then
    if reset_n = '0' or EndOfLine = '1' then
      vga_hsync <= '1';
```

```
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk25)
begin
  if rising_edge(clk25) then
    if reset_n = '0' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk25)
begin
  if rising_edge(clk25) then
    if reset_n = '0' then
```

```
      vga_vsync <= '1';
    elsif EndOfLine ='1' then
      if EndOfField = '1' then
        vga_vsync <= '1';
      elsif Vcount = VSYNC - 1 then
        vga_vsync <= '0';
      end if;
    end if;
  end if;
end process VSyncGen;

VBlankGen : process (clk25)
begin
  if rising_edge(clk25) then
    if reset_n = '0' then
      vga_vblank <= '1';
    elsif EndOfLine = '1' then
      if Vcount = VSYNC + VBACK_PORCH - 1 then
        vga_vblank <= '0';
      elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
        vga_vblank <= '1';
      end if;
    end if;
```

```
  end if;

end process VBlankGen;


-- Cube generator


MyHcount <= Hcount - HSYNC - HBACK_PORCH;

MyVcount <= Vcount - VSYNC - VBACK_PORCH;



HRegion(11) <= '1' when MyHcount(9 downto 5) = 7  else '0';
HRegion(10) <= '1' when MyHcount(9 downto 5) = 8  else '0';

HRegion(9)  <= '1' when MyHcount(9 downto 5) = 9  else '0';
HRegion(8)  <= '1' when MyHcount(9 downto 5) = 10 else '0';

HRegion(7)  <= '1' when MyHcount(9 downto 5) = 11 else '0';
HRegion(6)  <= '1' when MyHcount(9 downto 5) = 12 else '0';

HRegion(5)  <= '1' when MyHcount(9 downto 5) = 3  else '0';
HRegion(4)  <= '1' when MyHcount(9 downto 5) = 4  else '0';

HRegion(3)  <= '1' when MyHcount(9 downto 5) = 5  else '0';
HRegion(2)  <= '1' when MyHcount(9 downto 5) = 14 else '0';

HRegion(1)  <= '1' when MyHcount(9 downto 5) = 15 else '0';
HRegion(0)  <= '1' when MyHcount(9 downto 5) = 16 else '0';

V1Region(0) <= '1' when MyVcount < MyHcount/2               - V1RegionOffset + 12 else '0';
V1Region(1) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT   - V1RegionOffset + 8  else '0';
V1Region(2) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*2 - V1RegionOffset + 4  else
'0';
V1Region(3) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 - V1RegionOffset     else '0';
V1Region(4) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*4 - V1RegionOffset     else '0';
V1Region(5) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*5 - V1RegionOffset     else '0';
V1Region(6) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*6 - V1RegionOffset     else '0';

V2Region(0) <= '1' when MyVcount < V2RegionOffset - MyHcount/2               + 12 else '0';
V2Region(1) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT   + 8  else '0';
V2Region(2) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*2 + 4  else
```

```
'0';
 V2Region(3) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3     else '0';
 V2Region(4) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*4     else '0';
 V2Region(5) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*5     else '0';
 V2Region(6) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*6     else '0';

 V3Region(0) <= '1' when MyVcount < MyHcount/2               - V3RegionOffset     else '0';
 V3Region(1) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT   - V3RegionOffset     else '0';
 V3Region(2) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*2 - V3RegionOffset     else '0';
 V3Region(3) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 - V3RegionOffset     else '0';

 V4Region(0) <= '1' when MyVcount < V4RegionOffset - MyHcount/2                  else '0';
 V4Region(1) <= '1' when MyVcount < V4RegionOffset - MyHcount/2 + CUBIE_HEIGHT       else '0';
 V4Region(2) <= '1' when MyVcount < V4RegionOffset - MyHcount/2 + CUBIE_HEIGHT*2     else '0';
 V4Region(3) <= '1' when MyVcount < V4RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3     else '0';

 V5Region(0) <= '1' when MyVcount < MyHcount/2               + V5RegionOffset + 12 else '0';
 V5Region(1) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT   + V5RegionOffset + 8  else '0';
 V5Region(2) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*2 + V5RegionOffset + 4  else
'0';
 V5Region(3) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 + V5RegionOffset     else '0';

 V6Region(0) <= '1' when MyVcount < V6RegionOffset - MyHcount/2               + 12 else '0';
 V6Region(1) <= '1' when MyVcount < V6RegionOffset - MyHcount/2 + CUBIE_HEIGHT   + 8  else '0';
 V6Region(2) <= '1' when MyVcount < V6RegionOffset - MyHcount/2 + CUBIE_HEIGHT*2 + 4  else
'0';
 V6Region(3) <= '1' when MyVcount < V6RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3     else '0';

 FaceDetector : process (clk25)

 begin
  if rising_edge(clk25) then
   if((HRegion(11) or HRegion(10) or HRegion(9)) and (V1Region(6) and not V1Region(3))) = '1'
then
     face <= "000";
   elsif((HRegion(8) or HRegion(7) or HRegion(6)) and (V2Region(6) and not V2Region(3))) = '1'
then
     face <= "001";
   elsif((V1Region(3) and not V1Region(0)) and (V2Region(3) and not V2Region(0))) = '1' then
     face <= "010";
   elsif((HRegion(0) or HRegion(1) or HRegion(2)) and (V3Region(3) and not V3Region(0))) = '1'
then
     face <= "011";
   elsif((HRegion(3) or HRegion(4) or HRegion(5)) and (V4Region(3) and not V4Region(0))) = '1'
then
     face <= "100";
```

```
    elsif((V5Region(3) and not V5Region(0)) and (V6Region(3) and not V6Region(0))) = '1' then
      face <= "101";
    else
      face <= "111";
    end if;
   end if;
 end process FaceDetector;

 PositionDetector : process (clk25)

 begin
   if rising_edge(clk25) then
    if(OneHot(0) or OneHot(9) or OneHot(18) or OneHot(27) or OneHot(36) or OneHot(45)) = '1'
then
      position <= "0000";
    elsif(OneHot(1) or OneHot(10) or OneHot(19) or OneHot(28) or OneHot(37) or OneHot(46)) =
'1' then
      position <= "0001";
    elsif(OneHot(2) or OneHot(11) or OneHot(20) or OneHot(29) or OneHot(38) or OneHot(47)) =
'1' then
      position <= "0010";
    elsif(OneHot(3) or OneHot(12) or OneHot(21) or OneHot(30) or OneHot(39) or OneHot(48)) =
'1' then
      position <= "0011";
    elsif(OneHot(4) or OneHot(13) or OneHot(22) or OneHot(31) or OneHot(40) or OneHot(49)) =
'1' then
      position <= "0100";
    elsif(OneHot(5) or OneHot(14) or OneHot(23) or OneHot(32) or OneHot(41) or OneHot(50)) =
'1' then
      position <= "0101";
    elsif(OneHot(6) or OneHot(15) or OneHot(24) or OneHot(33) or OneHot(42) or OneHot(51)) =
'1' then
      position <= "0110";
    elsif(OneHot(7) or OneHot(16) or OneHot(25) or OneHot(34) or OneHot(43) or OneHot(52)) =
'1' then
      position <= "0111";
    elsif(OneHot(8) or OneHot(17) or OneHot(26) or OneHot(35) or OneHot(44) or OneHot(53)) =
'1' then
      position <= "1000";
    else
      position <= "0000";
    end if;
   end if;
 end process PositionDetector;

 raddr <= to_integer(unsigned(face(2 downto 0) & "000") + unsigned("000" & face(2 downto 0))
```

```
        + unsigned("00" & position(3 downto 0)));

OneHot(0)  <= HRegion(11) and V1Region(4) and not V1Region(3); --face1
OneHot(1)  <= HRegion(10) and V1Region(4) and not V1Region(3);
OneHot(2)  <= HRegion(9)  and V1Region(4) and not V1Region(3);
OneHot(3)  <= HRegion(11) and V1Region(5) and not V1Region(4);
OneHot(4)  <= HRegion(10) and V1Region(5) and not V1Region(4);
OneHot(5)  <= HRegion(9)  and V1Region(5) and not V1Region(4);
OneHot(6)  <= HRegion(11) and V1Region(6) and not V1Region(5);
OneHot(7)  <= HRegion(10) and V1Region(6) and not V1Region(5);
OneHot(8)  <= HRegion(9)  and V1Region(6) and not V1Region(5);

OneHot(9)  <= HRegion(8)  and V2Region(4) and not V2Region(3); --face2
OneHot(10) <= HRegion(7)  and V2Region(4) and not V2Region(3);
OneHot(11) <= HRegion(6)  and V2Region(4) and not V2Region(3);
OneHot(12) <= HRegion(8)  and V2Region(5) and not V2Region(4);
OneHot(13) <= HRegion(7)  and V2Region(5) and not V2Region(4);
OneHot(14) <= HRegion(6)  and V2Region(5) and not V2Region(4);
OneHot(15) <= HRegion(8)  and V2Region(6) and not V2Region(5);
OneHot(16) <= HRegion(7)  and V2Region(6) and not V2Region(5);
OneHot(17) <= HRegion(6)  and V2Region(6) and not V2Region(5);

OneHot(18) <= V1Region(1) and not V1Region(0) and V2Region(1) and not V2Region(0); --face3
OneHot(19) <= V1Region(1) and not V1Region(0) and V2Region(2) and not V2Region(1);
OneHot(20) <= V1Region(1) and not V1Region(0) and V2Region(3) and not V2Region(2);
OneHot(21) <= V1Region(2) and not V1Region(1) and V2Region(1) and not V2Region(0);
OneHot(22) <= V1Region(2) and not V1Region(1) and V2Region(2) and not V2Region(1);
OneHot(23) <= V1Region(2) and not V1Region(1) and V2Region(3) and not V2Region(2);
OneHot(24) <= V1Region(3) and not V1Region(2) and V2Region(1) and not V2Region(0);
OneHot(25) <= V1Region(3) and not V1Region(2) and V2Region(2) and not V2Region(1);
OneHot(26) <= V1Region(3) and not V1Region(2) and V2Region(3) and not V2Region(2);

OneHot(27) <= HRegion(0)  and V3Region(1) and not V3Region(0); --face4
OneHot(28) <= HRegion(1)  and V3Region(1) and not V3Region(0);
OneHot(29) <= HRegion(2)  and V3Region(1) and not V3Region(0);
OneHot(30) <= HRegion(0)  and V3Region(2) and not V3Region(1);
OneHot(31) <= HRegion(1)  and V3Region(2) and not V3Region(1);
OneHot(32) <= HRegion(2)  and V3Region(2) and not V3Region(1);
OneHot(33) <= HRegion(0)  and V3Region(3) and not V3Region(2);
OneHot(34) <= HRegion(1)  and V3Region(3) and not V3Region(2);
OneHot(35) <= HRegion(2)  and V3Region(3) and not V3Region(2);

OneHot(36) <= HRegion(3)  and V4Region(1) and not V4Region(0); --face5
OneHot(37) <= HRegion(4)  and V4Region(1) and not V4Region(0);
OneHot(38) <= HRegion(5)  and V4Region(1) and not V4Region(0);
OneHot(39) <= HRegion(3)  and V4Region(2) and not V4Region(1);
```

```
OneHot(40) <= HRegion(4)  and V4Region(2) and not V4Region(1);
OneHot(41) <= HRegion(5)  and V4Region(2) and not V4Region(1);
OneHot(42) <= HRegion(3)  and V4Region(3) and not V4Region(2);
OneHot(43) <= HRegion(4)  and V4Region(3) and not V4Region(2);
OneHot(44) <= HRegion(5)  and V4Region(3) and not V4Region(2);

OneHot(45) <= V5Region(3) and not V5Region(2) and V6Region(3) and not V6Region(2); --face6
OneHot(46) <= V5Region(2) and not V5Region(1) and V6Region(3) and not V6Region(2);
OneHot(47) <= V5Region(1) and not V5Region(0) and V6Region(3) and not V6Region(2);
OneHot(48) <= V5Region(3) and not V5Region(2) and V6Region(2) and not V6Region(1);
OneHot(49) <= V5Region(2) and not V5Region(1) and V6Region(2) and not V6Region(1);
OneHot(50) <= V5Region(1) and not V5Region(0) and V6Region(2) and not V6Region(1);
OneHot(51) <= V5Region(3) and not V5Region(2) and V6Region(1) and not V6Region(0);
OneHot(52) <= V5Region(2) and not V5Region(1) and V6Region(1) and not V6Region(0);
OneHot(53) <= V5Region(1) and not V5Region(0) and V6Region(1) and not V6Region(0);

FacehlineGen : process (clk25)

begin
 if rising_edge(clk25) then

   if MyHcount(9 downto 5) = 10 then
    Face4hline1 <= '1';
   elsif MyHcount(9 downto 5) = 14 then
    Face4hline1 <= '0';
   end if;

   if MyHcount(9 downto 5) = 13 then
    Face4hline2 <= '1';
   elsif MyHcount(9 downto 5) = 17 then
    Face4hline2 <= '0';
   end if;

   if MyHcount(9 downto 5) = 13 then
    Face4hline4 <= '1';
   else
    Face4hline4 <= '0';
   end if;

   if MyHcount(9 downto 5) = 6 then
    Face5hline1 <= '1';
   elsif MyHcount(9 downto 5) = 10 then
    Face5hline1 <= '0';
   end if;

   if MyHcount(9 downto 5) = 3 then
```

```
      Face5hline2 <= '1';
    elsif MyHcount(9 downto 5) = 7 then
      Face5hline2 <= '0';
    end if;

    if MyHcount(9 downto 5) = 6 then
      Face5hline4 <= '1';
    else
      Face5hline4 <= '0';
    end if;

    if MyHcount = 224 or MyHcount = 225 then
      Face6hline1 <= '1';
    else
      Face6hline1 <= '0';
    end if;

    if MyHcount = 320 or MyHcount = 321 then
      Face6hline2 <= '1';
    else
      Face6hline2 <= '0';
    end if;

    if MyHcount = 416 or MyHcount = 417 then
      Face6hline3 <= '1';
    else
      Face6hline3 <= '0';
    end if;

   end if;
  end process FacehlineGen;

  Face4vline1 <= '1' when MyVcount = V2RegionOffset - MyHcount/2              + 12 else '0';
  Face4vline2 <= '1' when MyVcount = V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3     else '0';
  Face4vline3 <= '1' when MyVcount = V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*6     else '0';
  Face4vline4 <= '1' when MyVcount = V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3 + 12 else
'0';
  Face4line    <= (Face4hline1 and Face4vline1) or (Face4hline2 and Face4vline2) or (Face4hline2
and Face4vline3) or (Face4hline4 and Face4vline4);

  Face5vline1 <= '1' when MyVcount = MyHcount/2              - V1RegionOffset + 12 else '0';
  Face5vline2 <= '1' when MyVcount = MyHcount/2 + CUBIE_HEIGHT*3 - V1RegionOffset     else '0';
  Face5vline3 <= '1' when MyVcount = MyHcount/2 + CUBIE_HEIGHT*6 - V1RegionOffset     else '0';
  Face5vline4 <= '1' when MyVcount = MyHcount/2 + CUBIE_HEIGHT*3 - V1RegionOffset + 12 else
'0';
  Face5line    <= (Face5hline1 and Face5vline1) or (Face5hline2 and Face5vline2) or (Face5hline2
```

```
and Face5vline3) or (Face5hline4 and Face5vline4);

  Face6vline1 <= '1' when MyVcount > 266 and MyVcount < 380 else '0';
  Face6vline2 <= '1' when MyVcount > 314 and MyVcount < 428 else '0';
  Face6line   <= (Face6hline1 and Face6vline1) or (Face6hline2 and Face6vline2) or (Face6hline3
and Face6vline1);

  RaddrReg : process (clk25)

  begin

    if rising_edge(clk25) then

      raddr_reg1 <= raddr;

      raddr_reg2 <= raddr_reg1;

    end if;

  end process RaddrReg;

  lines <= '0' when raddr_reg1 = raddr and raddr_reg1 = raddr_reg2 and (Face4line or Face5line or
Face6line) = '0' else '1';

  --Selection Border Register

  process (clk)
  begin
    if rising_edge(clk) then
      if reset_n = '0' then
        border_reg <= "000001";
      elsif (chipselect and write) = '1' and address = "00110110" then
        border_reg <= writedata(5 downto 0);
      end if;
    end if;
  end process;


  -- Selection border generator

  SBorderHRegion(7)  <= '1' when MyHcount(9 downto 3) > 10 else '0';
  SBorderHRegion(6)  <= '1' when MyHcount(9 downto 3) > 24 else '0';

  SBorderHRegion(5)  <= '1' when MyHcount(9 downto 3) > 54 else '0';
  SBorderHRegion(4)  <= '1' when MyHcount(9 downto 3) > 68 else '0';
  SBorderHRegion(3)  <= '1' when MyHcount(9 downto 3) > 26 else '0';
```

```
  SBorderHRegion(2)  <= '1' when MyHcount(9 downto 3) > 40 else '0';

  SBorderHRegion(1)  <= '1' when MyHcount(9 downto 3) > 38 else '0';
  SBorderHRegion(0)  <= '1' when MyHcount(9 downto 3) > 52 else '0';

  SBorderV1Region(0) <= '1' when MyVcount < MyHcount/2              - V1RegionOffset + 4 else '0';
  SBorderV1Region(1) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 - V1RegionOffset -
8 else '0';
  SBorderV1Region(2) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 - V1RegionOffset +
8 else '0';
  SBorderV1Region(3) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*6 - V1RegionOffset +
8 else '0';

  SBorderV2Region(0) <= '1' when MyVcount < V2RegionOffset - MyHcount/2              + 4 else '0';
  SBorderV2Region(1) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3 -
8 else '0';
  SBorderV2Region(2) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3 +
8 else '0';
  SBorderV2Region(3) <= '1' when MyVcount < V2RegionOffset - MyHcount/2 + CUBIE_HEIGHT*6 +
8 else '0';

  SBorderV3Region(0) <= '1' when MyVcount < MyHcount/2              - V3RegionOffset - 8 else '0';
  SBorderV3Region(1) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 - V3RegionOffset +
8 else '0';

  SBorderV4Region(0) <= '1' when MyVcount < V4RegionOffset - MyHcount/2              - 8 else '0';
  SBorderV4Region(1) <= '1' when MyVcount < V4RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3 +
8 else '0';

  SBorderV5Region(0) <= '1' when MyVcount < MyHcount/2              + V5RegionOffset + 4 else '0';
  SBorderV5Region(1) <= '1' when MyVcount < MyHcount/2 + CUBIE_HEIGHT*3 + V5RegionOffset +
8 else '0';

  SBorderV6Region(0) <= '1' when MyVcount < V6RegionOffset - MyHcount/2              + 4 else '0';
  SBorderV6Region(1) <= '1' when MyVcount < V6RegionOffset - MyHcount/2 + CUBIE_HEIGHT*3 +
8 else '0';

  SBorderGenerator : process (clk25)

  begin
    if rising_edge(clk25) then
     SBorder(0) <= SBorderHRegion(3) and not SBorderHRegion(2) and SBorderV1Region(3) and
not SBorderV1Region(1) and
            not ((HRegion(11) or HRegion(10) or HRegion(9)) and V1Region(6) and not V1Region(3));
     SBorder(1) <= SBorderHRegion(1) and not SBorderHRegion(0) and SBorderV2Region(3) and
not SBorderV2Region(1) and
```

```
            not ((HRegion(8)  or HRegion(7)  or HRegion(6)) and V2Region(6) and not V2Region(3));
    SBorder(2) <= SBorderV1Region(2) and not SBorderV1Region(0) and SBorderV2Region(2) and
not SBorderV2Region(0) and
            not (V1Region(3) and not V1Region(0) and V2Region(3) and not V2Region(0));
    SBorder(3) <= SBorderHRegion(5) and not SBorderHRegion(4) and SBorderV3Region(1) and
not SBorderV3Region(0) and
            not ((HRegion(2)  or HRegion(1)  or HRegion(0)) and V3Region(3) and not V3Region(0));
    SBorder(4) <= SBorderHRegion(7) and not SBorderHRegion(6) and SBorderV4Region(1) and
not SBorderV4Region(0) and
            not ((HRegion(5)  or HRegion(4)  or HRegion(3)) and V4Region(3) and not V4Region(0));
    SBorder(5) <= SBorderV5Region(1) and not SBorderV5Region(0) and SBorderV6Region(1) and
not SBorderV6Region(0) and
            not (V5Region(3) and not V5Region(0) and V6Region(3) and not V6Region(0));
   end if;
 end process SBorderGenerator;

 -- Text Display

 Mux_8_to_1: entity work.mux_8_to_1 port map(
   data_in     => text_data(7 downto 0),
   sel         => std_logic_vector(MyHcount(2 downto 0)),
   data_out    => text_display

 );

 TextRegion: process (clk25)

 begin
  if rising_edge(clk25) then
    if MyVcount(9 downto 3) = 3 then
     if MyHcount(9 downto 3) = 25 then
       text_region1 <= '1';
     elsif MyHcount(9 downto 3) = 55 then
       text_region1 <= '0';
     end if;
    else
     text_region1 <= '0';
    end if;

    if MyVcount(9 downto 3) = 39 then
     text_region23_v <= '1';
    elsif MyVcount(9 downto 3) = 43 then
     text_region23_v <= '0';
    end if;

    if MyHcount(9 downto 3) = 4 then
```

```
      text_region2_h <= '1';
    elsif MyHcount(9 downto 3) = 24 then
      text_region2_h <= '0';
    end if;

    if MyHcount(9 downto 3) = 56 then
      text_region3_h <= '1';
    elsif MyHcount(9 downto 3) = 76 then
      text_region3_h <= '0';
    end if;

    text_region <= text_region1 or ((text_region2_h or text_region3_h) and text_region23_v);
   end if;
 end process TextRegion;

 subtraction_result <= std_logic_vector(MyVcount(9 downto 3)-39);

 raddr_text  <=
   unsigned('0' & std_logic_vector(MyHcount(9 downto 3)))-25 when text_region1 = '1' else
   26 + unsigned(subtraction_result(3 downto 0) & "0000") + unsigned(subtraction_result(5 downto
0) & "00")
      + unsigned('0' & std_logic_vector(MyHcount(9 downto 3)))
     when (text_region23_v and text_region2_h) = '1' else
   54 + unsigned(subtraction_result(3 downto 0) & "0000") + unsigned(subtraction_result(5 downto
0) & "00")
      + unsigned('0' & std_logic_vector(MyHcount(9 downto 3)))
     when (text_region23_v and text_region3_h) = '1';


 -- Registered video signals going to the video DAC


 VideoOut: process (clk25, reset_n)

 begin

   if reset_n = '0' then

     VGA_R <= "0000000000";

     VGA_G <= "0000000000";

     VGA_B <= "0000000000";

   elsif rising_edge(clk25) then
```

```
if lines = '1' or (text_region and text_display) = '1' then

      VGA_R <= "0000000000";

  VGA_G <= "0000000000";

  VGA_B <= "0000000000";
elsif ((border_reg(0) and SBorder(0)) or
      (border_reg(1) and SBorder(1)) or
      (border_reg(2) and SBorder(2)) or
      (border_reg(3) and SBorder(3)) or
      (border_reg(4) and SBorder(4)) or
      (border_reg(5) and SBorder(5))) = '1' then
  VGA_R <= "1010110100";

      VGA_G <= "1110000100";

      VGA_B <= "1110000100";
elsif color = "110" then --yellow

      VGA_R <= "1111111111";

  VGA_G <= "1111111111";

  VGA_B <= "0000000000";

elsif color = "010" then --red

      VGA_R <= "1111111111";

  VGA_G <= "0000000000";

  VGA_B <= "0000000000";

elsif color = "011" then --blue

      VGA_R <= "0000000000";

  VGA_G <= "0000000000";

  VGA_B <= "1111111111";

elsif color = "101" then --green

      VGA_R <= "0000000000";
```

```
    VGA_G <= "1111111111";
    VGA_B <= "0000000000";
  elsif color = "001" then --white
      VGA_R <= "1111111111";
    VGA_G <= "1111111111";
    VGA_B <= "1111111111";
  elsif color = "100" then --orange
      VGA_R <= "1111111111";
    VGA_G <= "0110001000";
    VGA_B <= "0001000000";
  elsif vga_hblank = '0' and vga_vblank = '0' then
    VGA_R <= "1111111111";
    VGA_G <= "1111111111";
    VGA_B <= "1111111111";
  else
    VGA_R <= "0000000000";
    VGA_G <= "0000000000";
    VGA_B <= "0000000000";
  end if;
 end if;
end process VideoOut;

VGA_CLK <= clk25;
```

```
 VGA_HS <= not vga_hsync;

 VGA_VS <= not vga_vsync;

 VGA_SYNC <= '0';

 VGA_BLANK <= not (vga_hsync or vga_vsync);



end rtl;

--------------------------------------------------------------------
char_rom.vhd
--------------------------------------------------------------------

-- megafunction wizard: %ROM: 1-PORT%
-- GENERATION: STANDARD
-- VERSION: WM1.0
-- MODULE: altsyncram

-- ============================================================
-- File Name: char_rom.vhd
-- Megafunction Name(s):
--                         altsyncram
--
-- Simulation Library Files(s):
--                         altera_mf
-- ============================================================
-- ************************************************************
-- THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
--
-- 12.1 Build 177 11/07/2012 SJ Full Version
-- ************************************************************


--Copyright (C) 1991-2012 Altera Corporation
--Your use of Altera Corporation's design tools, logic functions
--and other software and tools, and its AMPP partner logic
--functions, and any output files from any of the foregoing
--(including device programming or simulation files), and any
--associated documentation or information are expressly subject
--to the terms and conditions of the Altera Program License
--Subscription Agreement, Altera MegaCore Function License
--Agreement, or other applicable license agreement, including,
--without limitation, that your use is for the sole purpose of
```

```
--programming logic devices manufactured by Altera and sold by
--Altera or its authorized distributors.  Please refer to the
--applicable agreement for further details.


LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY altera_mf;
USE altera_mf.all;

ENTITY char_rom IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		q		: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
	);
END char_rom;


ARCHITECTURE SYN OF char_rom IS

	SIGNAL sub_wire0	: STD_LOGIC_VECTOR (7 DOWNTO 0);

	COMPONENT altsyncram
	GENERIC (
		clock_enable_input_a		: STRING;
		clock_enable_output_a		: STRING;
		init_file		: STRING;
		intended_device_family		: STRING;
		lpm_hint		: STRING;
		lpm_type		: STRING;
		numwords_a		: NATURAL;
		operation_mode		: STRING;
		outdata_aclr_a		: STRING;
		outdata_reg_a		: STRING;
		widthad_a		: NATURAL;
		width_a		: NATURAL;
		width_byteena_a		: NATURAL
	);
	PORT (
			address_a	: IN STD_LOGIC_VECTOR (8 DOWNTO 0);
			clock0	: IN STD_LOGIC ;
			q_a	: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
	);
```

```
	END COMPONENT;

BEGIN
	q    <= sub_wire0(7 DOWNTO 0);

	altsyncram_component : altsyncram
	GENERIC MAP (
		clock_enable_input_a => "BYPASS",
		clock_enable_output_a => "BYPASS",
		init_file => "../chars.mif",
		intended_device_family => "Cyclone II",
		lpm_hint => "ENABLE_RUNTIME_MOD=NO",
		lpm_type => "altsyncram",
		numwords_a => 384,
		operation_mode => "ROM",
		outdata_aclr_a => "NONE",
		outdata_reg_a => "CLOCK0",
		widthad_a => 9,
		width_a => 8,
		width_byteena_a => 1
	)
	PORT MAP (
		address_a => address,
		clock0 => clock,
		q_a => sub_wire0
	);



END SYN;

-- ============================================================
-- CNX file retrieval info
-- ============================================================
-- Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
-- Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
-- Retrieval info: PRIVATE: AclrByte NUMERIC "0"
-- Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
-- Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
-- Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
-- Retrieval info: PRIVATE: Clken NUMERIC "0"
-- Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
-- Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
```

```
-- Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
-- Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
-- Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
-- Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
-- Retrieval info: PRIVATE: MIFfilename STRING "../chars.mif"
-- Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "384"
-- Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "0"
-- Retrieval info: PRIVATE: RegAddr NUMERIC "1"
-- Retrieval info: PRIVATE: RegOutput NUMERIC "1"
-- Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
-- Retrieval info: PRIVATE: SingleClock NUMERIC "1"
-- Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
-- Retrieval info: PRIVATE: WidthAddr NUMERIC "9"
-- Retrieval info: PRIVATE: WidthData NUMERIC "8"
-- Retrieval info: PRIVATE: rden NUMERIC "0"
-- Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
-- Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
-- Retrieval info: CONSTANT: INIT_FILE STRING "../chars.mif"
-- Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone II"
-- Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
-- Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
-- Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "384"
-- Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
-- Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
-- Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
-- Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "9"
-- Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
-- Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
-- Retrieval info: USED_PORT: address 0 0 9 0 INPUT NODEFVAL "address[8..0]"
-- Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
-- Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
-- Retrieval info: CONNECT: @address_a 0 0 9 0 address 0 0 9 0
-- Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
-- Retrieval info: CONNECT: q 0 0 8 0 @q_a 0 0 8 0
-- Retrieval info: GEN_FILE: TYPE_NORMAL char_rom.vhd TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL char_rom.inc FALSE
-- Retrieval info: GEN_FILE: TYPE_NORMAL char_rom.cmp TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL char_rom.bsf TRUE
-- Retrieval info: GEN_FILE: TYPE_NORMAL char_rom_inst.vhd TRUE
-- Retrieval info: LIB_FILE: altera_mf
```

---

mux_8_to_1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity mux_8_to_1 is
port (
    data_in  : in std_logic_vector(7 downto 0);
    sel      : in std_logic_vector(2 downto 0);
    data_out : out std_logic
);
end mux_8_to_1;

architecture imp of mux_8_to_1 is
begin

   data_out <=
    data_in(0) when sel = "000" else
    data_in(7) when sel = "001" else
    data_in(6) when sel = "010" else
    data_in(5) when sel = "011" else
    data_in(4) when sel = "100" else
    data_in(3) when sel = "101" else
    data_in(2) when sel = "110" else
    data_in(1) when sel = "111" else
    'X';

end imp;
```

---

simple_dual_port_ram_single_clock_color.vhd

```
-- Quartus II VHDL Template
-- Simple Dual-Port RAM with different read/write addresses but
-- single read/write clock

library ieee;
use ieee.std_logic_1164.all;

entity simple_dual_port_ram_single_clock_color is

	generic
	(
		DATA_WIDTH : natural := 3;
		ADDR_WIDTH : natural := 6
```

```
	);

	port
	(
		clk	: in std_logic;
		raddr	: in natural range 0 to 2**ADDR_WIDTH - 1;
		waddr : in natural range 0 to 2**ADDR_WIDTH - 1;
		data	: in std_logic_vector((DATA_WIDTH-1) downto 0);
		we		: in std_logic := '1';
		q		: out std_logic_vector((DATA_WIDTH -1) downto 0)
	);

end simple_dual_port_ram_single_clock_color;

architecture rtl of simple_dual_port_ram_single_clock_color is

	-- Build a 2-D array type for the RAM
	subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
	type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

	function init_ram
		return memory_t is
		variable tmp : memory_t := (others => (others => '0'));
	begin
		tmp(0)  := "011"; tmp(16) := "110"; tmp(32) := "101"; tmp(48) := "010";
		tmp(1)  := "011"; tmp(17) := "110"; tmp(33) := "101"; tmp(49) := "010";
		tmp(2)  := "011"; tmp(18) := "100"; tmp(34) := "101"; tmp(50) := "010";
		tmp(3)  := "011"; tmp(19) := "100"; tmp(35) := "101"; tmp(51) := "010";
		tmp(4)  := "011"; tmp(20) := "100"; tmp(36) := "001"; tmp(52) := "010";
		tmp(5)  := "011"; tmp(21) := "100"; tmp(37) := "001"; tmp(53) := "010";
		tmp(6)  := "011"; tmp(22) := "100"; tmp(38) := "001"; tmp(54) := "000";
		tmp(7)  := "011"; tmp(23) := "100"; tmp(39) := "001"; tmp(55) := "000";
		tmp(8)  := "011"; tmp(24) := "100"; tmp(40) := "001"; tmp(56) := "000";
		tmp(9)  := "110"; tmp(25) := "100"; tmp(41) := "001"; tmp(57) := "000";
		tmp(10) := "110"; tmp(26) := "100"; tmp(42) := "001"; tmp(58) := "000";
		tmp(11) := "110"; tmp(27) := "101"; tmp(43) := "001"; tmp(59) := "000";
		tmp(12) := "110"; tmp(28) := "101"; tmp(44) := "001"; tmp(60) := "000";
		tmp(13) := "110"; tmp(29) := "101"; tmp(45) := "010"; tmp(61) := "000";
		tmp(14) := "110"; tmp(30) := "101"; tmp(46) := "010"; tmp(62) := "000";
		tmp(15) := "110"; tmp(31) := "101"; tmp(47) := "010"; tmp(63) := "111";
		return tmp;
	end init_ram;

	-- Declare the RAM signal and specify a default value.	Quartus II
	-- will create a memory initialization file (.mif) based on the
	-- default value.
```

```
	signal ram : memory_t := init_ram;

	-- Register to hold the address
	signal addr_reg : natural range 0 to 2**ADDR_WIDTH-1;


begin

	process(clk)
	begin
	if(rising_edge(clk)) then
		if(we = '1') then
			ram(waddr) <= data;
		end if;

		-- On a read during a write to the same address, the read will
		-- return the OLD data at the address
		q <= ram(raddr);
	end if;
	end process;

end rtl;
```

```
--------------------------------------------------------------------
simple_dual_port_ram_single_clock_text.vhd
--------------------------------------------------------------------
```

```
-- Quartus II VHDL Template
-- Simple Dual-Port RAM with different read/write addresses but
-- single read/write clock

library ieee;
use ieee.std_logic_1164.all;

entity simple_dual_port_ram_single_clock_text is

	generic
	(
		DATA_WIDTH : natural := 6;
		ADDR_WIDTH : natural := 8
	);

	port
	(
		clk	: in std_logic;
		raddr	: in natural range 0 to 2**ADDR_WIDTH - 1;
```

```
		waddr : in natural range 0 to 2**ADDR_WIDTH - 1;
		data	: in std_logic_vector((DATA_WIDTH-1) downto 0);
		we	: in std_logic := '1';
		q	: out std_logic_vector((DATA_WIDTH -1) downto 0)
	);

end simple_dual_port_ram_single_clock_text;

architecture rtl of simple_dual_port_ram_single_clock_text is

	-- Build a 2-D array type for the RAM
	subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
	type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

	function init_ram
		return memory_t is
		variable tmp : memory_t := (others => (others => '0'));
	begin
		for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
			-- Initialize each address with the address itself
			tmp(addr_pos) := "100100"; -- space
		end loop;
		return tmp;
	end init_ram;

	signal ram : memory_t := init_ram;

	-- Register to hold the address
	signal addr_reg : natural range 0 to 2**ADDR_WIDTH-1;


begin

	process(clk)
	begin
	if(rising_edge(clk)) then
		if(we = '1') then
			ram(waddr) <= data;
		end if;
	end if;
	end process;
	q <= ram(raddr);

end rtl;
```

---

single_port_rom.vhd

```
-- Quartus II VHDL Template
-- Single-Port ROM

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity single_port_rom is

	generic
	(
		DATA_WIDTH : natural := 5;
		ADDR_WIDTH : natural := 6
	);

	port
	(
		clk	: in std_logic;
		addr	: in natural range 0 to 2**ADDR_WIDTH - 1;
		q		: out std_logic_vector((DATA_WIDTH -1) downto 0)
	);

end entity;

architecture rtl of single_port_rom is

	-- Build a 2-D array type for the RoM
	subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
	type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

	function init_rom
		return memory_t is
		variable tmp : memory_t := (others => (others => '0'));
	begin
		tmp(0)  := "00000"; tmp(16) := "00110"; tmp(32) := "00110"; tmp(48) := "01100";
		tmp(1)  := "00001"; tmp(17) := "00111"; tmp(33) := "00111"; tmp(49) := "01101";
		tmp(2)  := "00010"; tmp(18) := "01000"; tmp(34) := "01000"; tmp(50) := "01110";
		tmp(3)  := "00011"; tmp(19) := "01001"; tmp(35) := "01001"; tmp(51) := "01111";
		tmp(4)  := "00010"; tmp(20) := "01000"; tmp(36) := "01000"; tmp(52) := "01110";
		tmp(5)  := "00011"; tmp(21) := "01001"; tmp(37) := "01001"; tmp(53) := "01111";
		tmp(6)  := "00100"; tmp(22) := "01010"; tmp(38) := "01010"; tmp(54) := "10000";
		tmp(7)  := "00101"; tmp(23) := "01011"; tmp(39) := "01011"; tmp(55) := "10001";
		tmp(8)  := "00110"; tmp(24) := "01100"; tmp(40) := "01100"; tmp(56) := "10010";
```

```
        tmp(9)  := "00111"; tmp(25) := "01101"; tmp(41) := "01101"; tmp(57) := "10011";
        tmp(10) := "01000"; tmp(26) := "01110"; tmp(42) := "01110"; tmp(58) := "10100";
        tmp(11) := "01001"; tmp(27) := "01111"; tmp(43) := "01111"; tmp(59) := "10101";
        tmp(12) := "01000"; tmp(28) := "01110"; tmp(44) := "01110"; tmp(60) := "10100";
        tmp(13) := "01001"; tmp(29) := "01111"; tmp(45) := "01111"; tmp(61) := "10101";
        tmp(14) := "01010"; tmp(30) := "10000"; tmp(46) := "10000"; tmp(62) := "10110";
        tmp(15) := "01011"; tmp(31) := "10001"; tmp(47) := "10001"; tmp(63) := "10111";
        return tmp;
end init_rom;

-- Declare the ROM signal and specify a default value.      Quartus II
-- will create a memory initialization file (.mif) based on the
-- default value.
signal rom : memory_t := init_rom;
```