Homework 3

Data Structures and Algorithms in JAVA
Shlomo Hershkop
Department of Computer Science
Columbia University
Summer 2002

Due Thursday, Aug 1 (at beginning of class)

Theory (50 points)

1. (5 points) Prove by induction that the minimum number of nodes in an AVL tree of height \( h \) is \( N_h = F_{h+2} - 1 \) for \( h \geq 0 \)

2. (4 points) Show (Draw) the results of inserting 2, 1, 4, 5, 9, 13, 3, 17 into an initially empty AVL tree.

3. (4 points) Draw an example of an AVL tree such that a single remove operation could require \( \Theta(\log n) \) rotations from a leaf to the root in order to restore the height-balance property. (Use triangles to represent subtrees that are not affected by this operation.)

4. (5 points) A \( B^* \)-tree of order \( M \) is a B-tree in which each interior node has between \( 2M/3 \) and \( M \) children. Describe a method to perform insertion into a \( B^* \)-tree.

5. (5 points) Draw the \( B \)-tree with \( M=5, L=7 \) resulting from inserting the following keys (in this order) into an initially empty tree: 4, 40, 23, 50, 11, 34, 62, 78, 66, 22, 90, 59, 25, 72, 64, 77, 10, 12.

6. (6 points) Describe a modified version of the \( B \)-tree insertion algorithm so that each time we create an overflow because of a split node \( v \), we redistribute keys among \( v \)'s siblings such that each sibling holds roughly the same number of keys (possibly cascading the split up to the parent of \( v \)). What is the minimum fraction of each block that will always be filled using this scheme?

7. (8 points) Given input \{3471, 3123, 1673, 1499, 3444, 6979, 8989\} and a hash function \( h(x) = x \mod 10 \), show the resulting:
   - Separate chaining hash table.
   - Open addressing hash table using linear probing.
   - Open addressing hash table using quadratic probing.
   - Open addressing hash table with second hash function \( h_2(x) = 7 - (x \mod 7) \).

8. (4 points) Show the result of rehashing the last two hash tables in the previous question.

9. (6 points)
   - Show the result of inserting 23, 25, 14, 27, 19, 18, 21, 28, 24, 22, 20, 17, 24, 26, 15, one at a time, into an initially empty binary heap.
   - Show the result of using the linear-time algorithm to build a binary heap using the same input.
1. (4 points) Show the result of performing four deleteMin operations in the heap of the previous heap.

10. (3 points) Prove that for binary heaps, buildHeap does at most 2N-2 comparisons between elements.

---

**Programming (55 points)**

This project involves the use of a hash table to identify a program whether it is malicious or not. **You don't need to understand the Security background in order to do the assignment.**

**Security background**

Most security systems in use today are based on signatures. This means that some expert has sat down (for lots of money) and encoded the signature of bad programs into a database. Using this data, programs such as Norton Anti-virus and Mcafee, are able to identify virus programs. We will call the anti-virus programs checker programs.

The advantage of this approach is the low false positive rates, that is, if the check program thinks something is bad, it probably is true. This is because the signatures contain enough information, that the computer has effectively “memorized” what the virus looks like. The main drawback of the signature-based approach, is the inability of checkers to identifying new viruses. In addition, this approach also has to deal with the vulnerability window. This is the time it takes for a new signature to be created, tested, and updated to the actual computers that use them. In the real world this time frame can be quite large in comparison with the spread of viruses. Especially since viruses are starting to spread via emails.

Using Data mining and Machine Learning approaches, it is possible to create a model of what a virus looks like, and use this model to identify programs which might be viruses.

One of the ways is to use byte sequences of a virus program to create a dictionary of what a byte sequence of a potential virus would look like. We will be experimenting on what window size is needed for accurate detection.

You will be given a few known virus programs (v) and a few normal programs (denoted b= benign). You will build a general dictionary program which can take both virus and benign programs and create a dictionary to score unknown programs fed to it.

All the programs for your checker have been converted to 4 byte hexadecimal strings. Each 4 bytes will be a word (when windows size =1) (example first word is 7369, second word 7020).
Please note that when you have a window of size 2, the first WORD would be (7369 7020) and second word would be (7020 6f72).

The dictionary has to be implemented as a hash table.

The data will be from files provided on the homework page. They are in tar zipped format. To unpack them just type:
tar xzf filename.tar.gz

BEFORE YOU START: Choose a collision method ahead of time and stick to it (see step 4).

1) **Reading in the files**

In this step you will have to download the 3 data files. Create a method to take a directory name and run through each file in that directory.

Modify the hash table from the book to also keep a count for each key of how many times it has been accessed (for insert). Also a counter for how many collisions (ie inserts with existing keys). The counter can either be part of the Hash table or external.

Using a 1 word window, and 5-word window.

Build a hash table A for all the files in the normal directory (hwb).
Build hash table B for all the files in the virus directory (hwv).

For each hash table built, output the following:
1) The amount of items in the Table.
2) The amount of collisions occurred during reading
3) The size of the resulting hash table
4) The load factor

2) **Pruning down the hash table**

the following has to be done for words of size 1,3,5.
In this step you will create a new Hash table called VChecker. Read in all the files from the virus directory. Then go through the files in the normal directory. For each word seen in the normal file, check if it is in Vchecker, if it exists remove it.

Do the same the other way, creating a Bchecker, build it from the normal files, and prune it from the virus files.

for each hash table built, output the following:
   1) The amount of items in the Table.
   2) The size of the resulting hash table
   3) The load factor

3) **Virus Detection.**

Now you are ready to use the dictionaries from the last step to score the unknown programs. For window size (1,3 and 5), run through each unknown program and give it a score. The score will be how many of its words appear in each dictionary. Example if you read a size 1 word as 7070, and it appears in the Vchecker dictionary, you give the unknown program 1 point for being a virus (VScore). If it appears in the Bchecker dictionary give it 1 point for being normal (Nscore). The max score will be the prediction of what type of program it is.

so for each file in the unknown directory, for each window size:
   1) Print its name
   2) Print its Vscore for each window size
   3) Prints its Nscore for each window size
   4) Print the max prediction from all the window sizes (example if window size 1 says it is a virus, but window size 3 and 5 says its not, predict it is normal).

   Do not worry about being right. This is a test of an algorithm.

4) **Collision Method:**

Discuss how the collision method you chose, affected the answers in a certain way. Use common sense and knowledge gained from class / book.

5) **EXTRA CREDIT:**

Run through all files in each of the normal and virus directories, for each directory; test how good our system is. Output a single number saying how many of the normal program were predicted normal, and another number saying how many of the virus programs were predicted virus (correctly).