3137 Data Structures and Algorithms in C++

Lecture 8

July 31 2006

Shlomo Hershkop

1

Announcements

- Will grade midterms by tomorrow
- Will review all questions today
- Wednesday, can we get started a little earlier ?
  - I will need to end class about ½ hour early…
- Next HW posted
- Last two weeks…don't fall off the wagon as we pick up the pace!

2

Outline

- Disjointed DS
- Graph theory
- Midterm Review

- Reading:
  - Chapter 8-8.3, 9-9.3

3

Review

- We’ve covered some DS to keep track of a set of elements
  - lists
  - trees
  - hashtables
  - queues

4

- Given a set of elements its sometimes useful to be able to break them into a number of separate sets and be able to manipulate individual items or sets of related items
- Practical Example: New Orleans flood alert system

5

Equivalence Class ADT

- Union(a,b)
  - merges 2 equivalence classes
- Find(a)
  - retrieves equivalence class containing a

6

Analyzing Equivalence classes

- Amortized analysis
  - some kind of analysis requires a high level approach, since we won't always get worst case running time
    - Example: Wc, Bc, Bc, Bc, Wc, Bc, Bc, Bc, Bc, Bc…
- When doing the analysis here we will be interested in series of M operations
  - that way we can factor in all instances
  - different than dealing with average case
- lets go to implementation:

7

Arrays

- use an array and store name of class at each position
- example of union
- what is the running time
  - find?
  - union?

8

running times

- so find we can do in O(1)

- merge, worst could be O(N)
  - for M merges on N items
    - O(NM)
    - M can't exceed N
    - so $O(N^2)$

9

linked lists

- each member will be in its own list

- merge ?
- find ?

10

- so there must be a better way

- something to notice:

1. the name of the elements are static
2. the name of the set is arbitrary

11

solution

- use tree type relationship
  - here it is graphically
- forest:
  - collection of trees

12

- element which sits at the top is the root
- root will be the name of the class

- find spits out the root
- union(a,b) makes b a child of a

13

implementation

- forest can be an array with each item containing the index of the parent
- -1 could be no parent, hence root
- constructor puts -1 in each element's spot

- lets do some code

14

find

```
int find(int N) {
   if(set[N] < 0)
      return N
   else
      return find(set[N])
}
```

15

union

```
void union(int item1, int item2) {
  int root1, root2;
  root1 = find(item1);
  root2 = find(item2);
  if(root1 != root2)
     set[root2] = root1;
}
```

16

- runtimes ??

17

- find(n)
- union(1)
- haven’t really improved it much

18

tricks

- so need a trick to speed things up by not having deep trees

- Union by size
- Union by height

19

Union by size

- Rule: always make the SMALLER tree a subtree of larger one

- Will need to keep track of how many nodes in set
  - Why is that??
  - ideas ??

20

Union by height

- Rule: always make shallower tree a subtree of the deeper one
- how will the height change ??
- can you code this ?

21

Theory

- Idea: Can we prove that is unions are done by size, the depth of any node i will not be more than log(N) ??

22

Proof

- lets take an arbitrary node…call it bob!
- say bob is minding his own business at level 0
- at each union the depth can increase for bob by 0 or 1
- when bob’s depth increase it is placed in a tree twice as large as before
- so it can only increase its depth at most log(N) times

23

runtimes

- worst M log N
- find: Log n
- union Log n

24

More tricks

- one thing to notice, is that for nodes at end of long path, find is costly

25

Path Compression

- Like AVL trees, we want to make later operations faster by putting in some work now
- need to use amortized analysis
- Trick:
  - When doing a find make every node along the path to the root point to the root
  - here is an example
  - what is the overhead?

26

Quick Question

- now that we are somewhat proficient in DS
- how would you program an algorithm to generate mazes ??
  - how would you represent a maze ?
  - algorithm ?

27

summary

- so these are good data structures for manipulating groups of items
- many times, we are also interested in connections between groups of items (not just membership)
- Example: all people who called norway today ☺

28

- Graph Theory!!
  - very old, branch of math
  - many open problems
  - recent discoveries on interesting applications

- Trees are linear DS since we have a root and leaves, so can start at specific point and end at a specific point
- A more general DS would be to have 2 way links so can go anywhere
- not clear where beginning is, so need to be careful about algorithms

29

Definition

- A graph G=(V,E) consists of a set of vertices (nodes) V and edges E where each edge is a pair (v,w) such that v,w in V

30

- Directed Graph – A graph where vertices have direction
- Undirected Graph
- Weights – in addition can associate a weight with each edge

31

Example

- think of a map
- locations and mileage
- think of the internet
  - what are the nodes ? edges ?

32

Definition

- Path = a sequence of vertices v_1, v_2, v_3.. such that the vertices are adjacent
- Length = number of edges
- Loop = edges that starts and end at the same vertex

33

More definitions

- Simple path = a path with distinct vertices
  - that doesn't cross itself
- Cycle = path with the first and last vertex being the same

34

- DAG ?

35

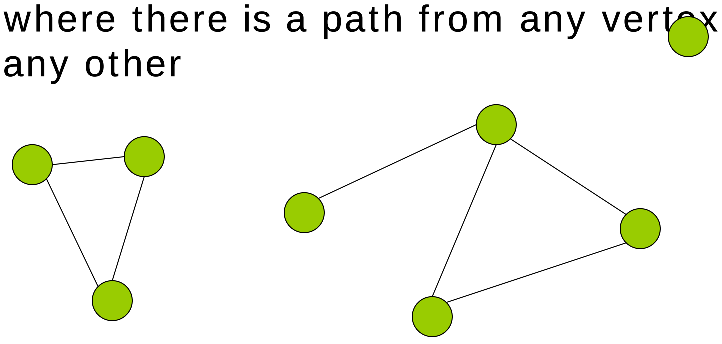
DAG

- Directed Acyclic Path

36

definitions

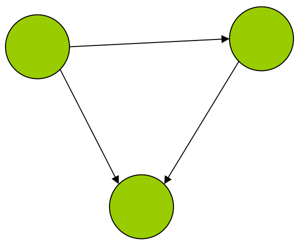
- connected graph = undirected graph where there is a path from any vertex to any other



37

Strongly Connected

- directed graph where path from any vertex to any other



?????

38

more definitions

- weakly connected – directed graph which would have been connected if it was undirected
- Complete graph = every vertex connected to every other

39

Point

- Very general DS
- Air transit
  - airports
  - flights
  - weight = cost
- Can answer:
  - what are the cheapest flights ?
  - shorted flights ?
  - where we should add flights ?
  - can we reach every city ?

40

one last definition

- Degree of a node:
  - undirected
    - number of connections of the node
  - Directed
    - in degree
    - out degree

programming

- so with 2 weeks left to DS
- how would you code graphs ?

- some ideas:
  - array
  - linked list

43

Arrays

- 2 dimensional n by n array
- lets do a simple example with 4 nodes
  - A, B, C, D
  - Directed..
  - How would an undirected graph be represented?

44

question

- how would we use weights here ?
- how would the code look from a high level view ?

45

Linked List

- you can also use a list representation called an adjacency list
- list of nodes
- linked list of edges

46

Resources

- how much space (in terms of V and E) would it take to represent a graph on either implementation ?
- what does that tell you?

47

Searching

- graph can be used for many different things
- one operation we would like to support is searching the graph
- we want to visit nodes without wasting time or missing any

48

Example

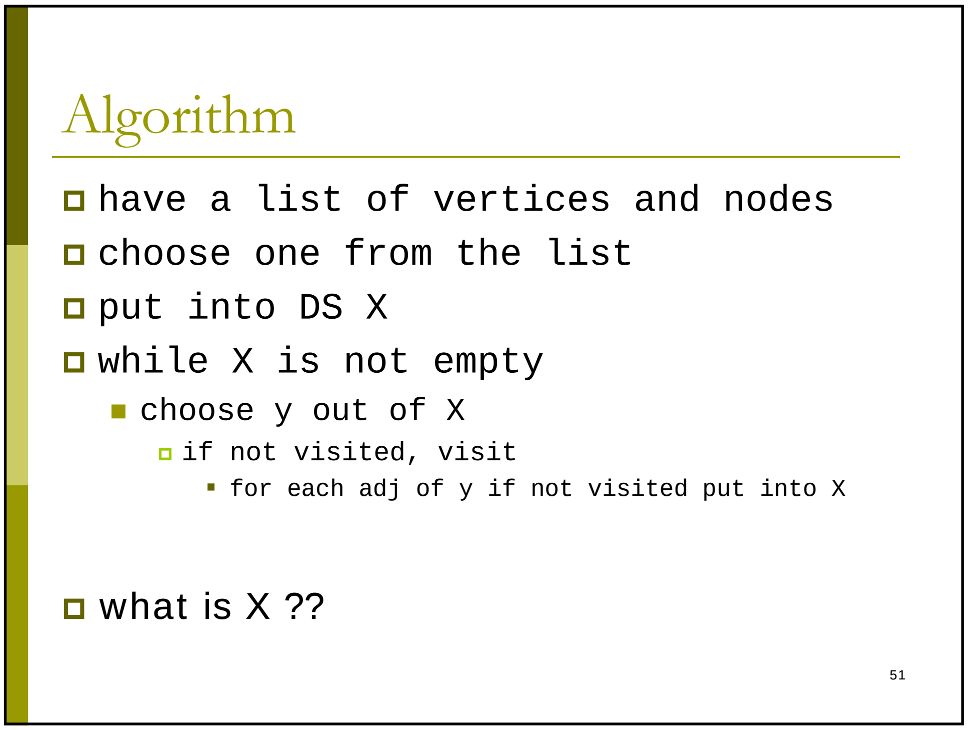
- say you represent a maze as a graph
- what strategy would you use to explore the maze ??

49

Expansion

- DFS
  - depth first search
- BFS
  - breadth first search

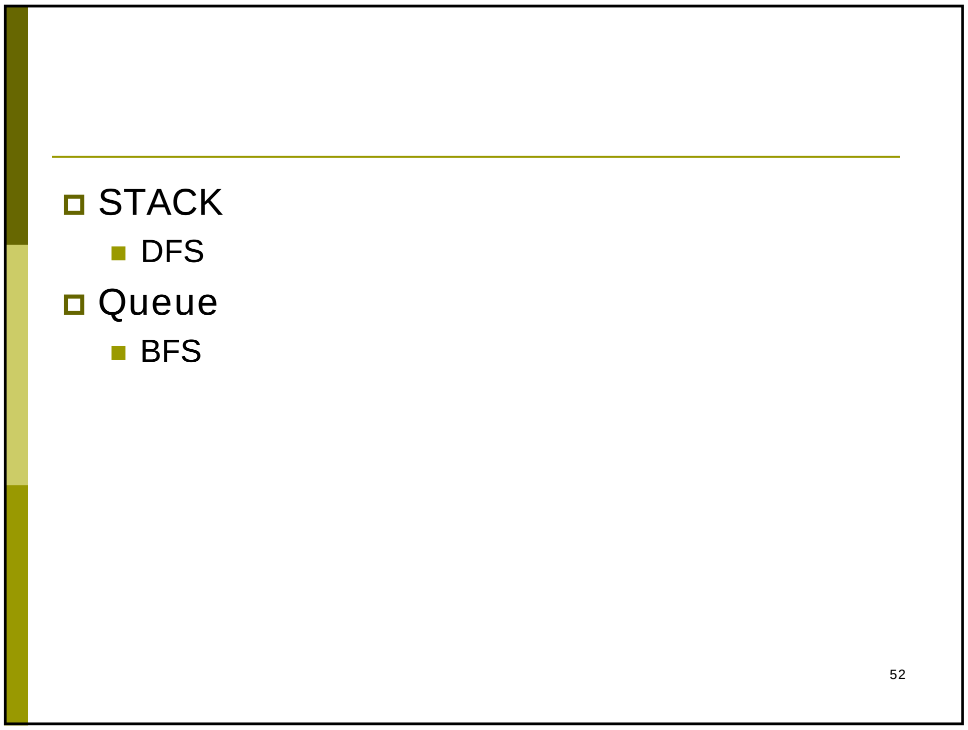
50



Algorithm

- have a list of vertices and nodes
- choose one from the list
- put into DS X
- while X is not empty
  - choose y out of X
    - if not visited, visit
      - for each adj of y if not visited put into X
- what is X ??

51



- STACK
  - DFS
- Queue
  - BFS

52

- We showed how to expand a graph to visit all nodes, any ideas on how to sort a graph ??
- what does it mean to sort a graph anyway ??

53

- Midterm Review

54

- Proof by induction
- Big O
- Theta bound
- ADT
- Extendible hashing

55

runtime?

```
Sum =0;
for (i=1; i<n n; i++ )
     for (j=1; j < i * i ; j++)
              if( j % i == 0)
                  For (k =0 ; k < j ; k++ )
                        Sum++;
```

56

- Is it possible to have two different huffman trees for a set of symbols with given frequencies ? Either provide an example where two different trees can be created or prove that there is only a single tree

- If you were to create a tic tac toe player and the computer would use a tee structure to represent all possible moves, what would the branching factor of the tree be and what would be the height of the tree assuming you represented every possible move ?

algorithm and runtime?

```
int foohoo(array, n) {
   if (n ==1)
        return array[0];
   else{
   int temp = foohoo(array,n-1);
   if(temp < array[n-1];
        return temp;
   else
        return array[n-1];
   }
}
```

59

sorting

- Given an array as input [E,X,A,M,P,L,E] show what happens (each stage) when you use selection sort and then the same input for bubble sort

60

hashing

- Why is it a bad idea for a hash function to depend on a single character (anywhere) in the input ?

61

- Explain how hashing can be applied to check whether all elements of a list are distinct ? what is the running time ??

62

- Which collision strategy is good for lazy deletion and which is not?

- Suppose you knew ahead of time which keys you will be using in your hash table, is it possible to invent a hash function so that there are zero collisions?? (more than yes/no) please.

- You fall asleep in your data structures course and find yourself in another class
- They are trying to stress test various kinds of glass jars to determine the height for which they can be dropped and still not break.
- They have a setup where they have a ladder leaning against the wall with n steps. you want to find the highest step which the glass jar will survive a fall (call it the HSR = highest safe rung).
- Clearly they have no idea where to start…lucky you know data structures and algorithms

a) describe an algorithm which takes n jars to find the correct HSR
b) now describe an algorithm which takes log n jars
c) what is the best algorithm you can think of ?

Next

- do reading
- start homework
- reminder we will be meeting Wednesday at 5pm-7:40pm