Homework 4 Theory

Joseph Mulray

Febuary 7, 2017

Problem 1:

The maximum height that a binary tree can have with n nodes, is $n - 1$. If $h(0)$ and $n = 1$ then the maximum height is $n - 1$.

The minimum height of a binary tree is:

$$log_2(n) = height$$

Each element n in a binary tree will have a height of at most 2^h for each node and by definition of logs, will result in $log_2(n) = height$.

Problem 2:

The maximum height for a tree that has n nodes is also $n - 1$ due to the fact that a tree can have any amount of children and the maxiumum height will always be $n - 1$ for each node you have.

The minimum height of a tree is 1 because the tree must have a single root because a tree with height 0 is not a tree at all.

Problem 4.6

	a	b	c	d
MAKENULL	$O(1)$	$O(n)$	$O(n)$	$O(n)$
UNION	$O(nlog(n))$	$O(nlog(n))$	$O(n)$	$O(n^2)$
INTERSECTION	$O(n)$	$O(n)$	$O(n)$	$O(n)$
MEMBER	$O(1)$	$O(1)$	$O(n)$	$O(n)$
MIN	$O(n)$	$O(n)$	$O(n)$	$O(n)$
INSERT	$O(1)$	$O(n)$	$O(1)$	$O(1)$
DELETE	$O(1)$	$O(n)$	$O(n)$	$O(n)$

Problem 4.7:

Suppose we are hashing integers with a 7-bucket hash table using the hash function h(i) = i mod 7.

a.

Open Hash

0: 343

1: 1, 8, 64

6: 27, 125, 216

Closed Hash Table

0: 125

1: 1

2: 8

3: 64

4:216

5: 343

6: 27

b.
Not consistantly possible to obtain constant time using a closed hash table and with linear resolution of collisions.

Problem 5:

Hash keys are character strings. The hash function h1(x) computes the length of the string:

This hash function that computes the length of the string that can lead to many problems, as words with hash table n buckets might have a string larger than n causing several problems when searching for values within that hash table

The hash function h2(x) computes a random number r with 1 <= r <= B:

The problem with this hash function is potentially would have a different hash each time making it unable to retrieve the values you stored in the table.

Problem 6:

1. Select an integer i from the set and delete it.
2. Add an integer i to the set

Would use a structure type array to represent a subset S in case S already contains I. We can then insert and delete in constant time $O(1)$ and where the size of the structure would be bound by $O(n)$.

Problem 7:

```
increase(size, table, next)
    begin
        buckets = table.buckets

        for x in buckets:
            index = hash(x, size)
            next.insert(index, x)
```

The running time for increasing the number of buckets in a closed hash table is constant and is $O(n)$.