HW4 Theory

Alexander Kazantsev

February 3, 2016

Problem 1

A binary tree has n nodes, assuming $n > 0$ and sufficiently large

The maximum height of the tree is $n - 1$. This is possible if each node only has one child, and since $h = 0$ if $n = 1$ then $max(h) = n - 1$ if the nodes are all linearly connected. Simply, it is more relatable to a linked list of size n.

The minimum height of the tree is $log_2(n)$. The amount of elements n in a binary tree grows with height h at a rate $n = 2^h$. Using the rule of logs it is obviously apparent that $log(_2(n) = h$.

Problem 2

A tree has n nodes, assuming $n > 1$ and sufficiently large

The maximum height of the tree is $n - 1$, the reason is the same as above.

The minimum height is 1. The reason is that since there is no limit on the amount of children a node can have, $n-1$ nodes can be the children of a parent node. The reason the minimum is 1 and not 0 is that there must be a single root for the tree.

Problem 4.6

Part a

MAKENULL: O(n)

UNION: O($nlog(n)$)

INTERSECTION:O(n)

MEMBER:O(1)

MIN:O(n)

INSERT:O(1^*)

DELETE:O(1)

Part b

MAKENULL: O(n)

UNION: O($nlog(n)$)

INTERSECTION:O(n)

MEMBER:O(1)

MIN:O(n)

INSERT:O(n)

DELETE:O(1)

Part c

MAKENULL: O(n)

UNION: O(n^2))

INTERSECTION:O(n^2)

MEMBER:O(n)

MIN:O(n)

INSERT:O(1)

DELETE:O(n)

Part d

MAKENULL: O(n)

UNION: O(n^2)

INTERSECTION:O(n^2)

MEMBER:O(n)

MIN:O(n)

INSERT:O(1)

DELETE:O(1)

Problem 4.7

Part a

Assuming the numbers were hashed in order of least to greatest, walks distance of 1

Hash table = [0:125, 1:1, 2:8, 3:64 , 4:216, 5:343, 6:27]

Part b

Hash table = [0:(343), 1:(1, 8, 64), 2:, 3:, 4:, 5:, 6:(27, 125, 216)]

Problem 5

Part a

The functions hash value is easily predictable, and will most likely result in many hash collisions which will lead to a mostly linear lookup.

Part b

This is non deterministic, only by luck will the value be obtained through a lookup in constant time.

Problem 6

The name of the data structure will be "AKSet" or "AK" for short (named after me of course!)

An AK will need to be a variant of a hash table to retrieve and delete elements in constant time. AK will be a variant of a open hash table.

AK will have precomputed thresholds to increase the number of buckets.

AK will hash a value and attempt to throw it into the hash table. Before adding it to the table it must first check whether the value is already in the set. After the value is hashed AK will do linear check over the bucket to see if the element exists. Since the amount of buckets is dynamic, that means the amount of elements per bucket is (relatively) constant. Therefore the check is not linear, but constant. If an element is added to the set and the average amount of elements per bucket is greater than the set threshold, then the hash table needs to be resiized which is a linear operation. This yields a time of $O(1^*)$ for insertion as most insertions will complete in constant time.

For deletion AK will perform some of the same tasks such as hashing the value and checking whether it exists in the set. The condition for the check is flipped for deletion, as it will continue only if the element does exist. If it does AK will delete it from the bucket which again is a constant operation. If the elements per bucket average drops below a threshold it would not make much sense to resize the hash table, so deletion yields $O(1)$ time.

Problem 7

```
increaseBuckets (currentTable, newTable,newSize)
    buckets = currentTable.buckets
    for b in buckets
        for e in b.elements
            newHash = hash(e, newSize)
            newTable.insert(newHash, e)
```

The time complexity is dependent on whether the amount of elements per bucket average is kept constant. If it is, then it yields O(n), otherwise it will be closer to O(n^2).