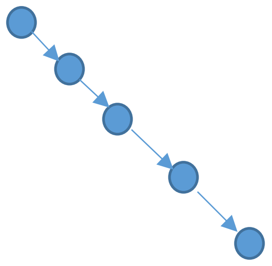
Homework 4

D Yoan L Mekontchou Yomba

CS 260

1) N nodes

To maximize the height of the tree with n nodes, each node must have 1 sibling only and they must propagate in the same direction. i.e



This would generate a tree(binary with a maximal height of h = n-1.

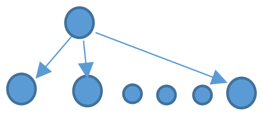
The minimum height of a binary tree with n nodes entails forming a complete tree meaning

Completely packing the upper layers before continuing to lower levels. Now, given n vertices, $1 + 2 + 3 + \ldots + 2^{n-1} + 2^n = 2^0 + 2^1 + \cdots + 2^n \to n = \sum_{i=0}^{n} 2^i = \frac{2^{n+1}-1}{2-1} = 2 * 2^n - 1$

$$n + 1 = 2 * 2^n \to \frac{n+1}{2} = 2^h \to h = \log\left(\frac{n+1}{2}\right) \to \log(n+1) - 1$$

$$for\ the\ case\ we\ have\ a\ complete\ tree\ and\ apprximately \log(n)\ if\ the\ tree\ isn't\ complete$$

2) A tree which is not a binary tree has a maximal height of n-1 as the tree can be arranged linearly and doesn't have the same constraints as the binary tree of having a maximal degree of 2 at each node. A tree which isn't a binary tree has a minimal height of 1 because the tree lacks the constraint of at most 2 connections for any node, the root can have a degree of n-1 which would create a minimal tree with height 1.

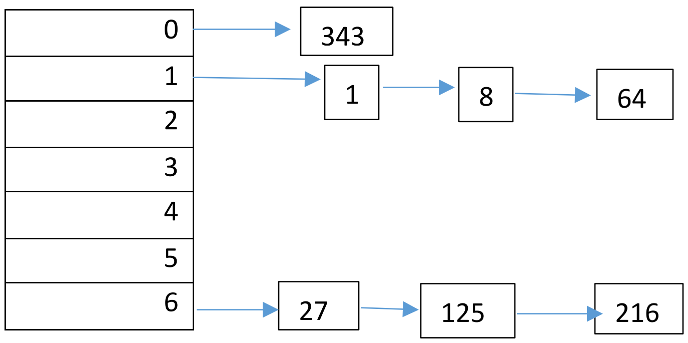


3) 4.6] Analyzing question 4.4
   a. Makenull → $O(n)$
   b. Union → $O(n^2)$
   c. Intersection → $O(n^2)$
   d. Member → $O(n)$
   e. Min → $O(1)$
   f. Insert → $O(n)$
   g. Delete → $O(1)$

4.6] Analyzing question 4.5

	Open Hash	Closed Hash	Unsorted List	Fixed Length Array & Pointer To the last position
Make null	$O(n)$	$O(1)$	$O(1)$	$O(1)$
Union	$O(logn)$	$O(logn)$	$O(n^2)$	$O(nlogn)$
Intersection	$O(1)$	$O(n)$	$O(n^2)$	$O(nlogn)$
Member	$O(1)$	$O(1)$	$O(n)$	$O(n)$
Min	$O(n)$	$O(n)$	$O(n)$	$O(1)$
Insert	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Delete	$O(logn)$	$O(1)$	$O(n)$	$O(n)$

4) 4.7] Open Hashing



Closed Hashing

125	1	8	64	216	343	27

5) 1. Hash keys are character strings. H(x) computes the length of the string. This is not a good hash function because the number it generates lacks randomness since it computes the length of a string. This would additionally cause an immense amount of collisions in the closed hashing case assuming a linear rehashing strategy is used and in the case of open hashing, because multiple strings would hash to the same value, the size of the set would be extremely long making operations operate in the worst case.

2. This random hash function could be problematic as 1 sample key could hash to a multitude of values which would make the lookup operation particularly tough

6) I aim to use a closed hash function with quadratic probing to limit the clustering effect. My hash function would be → $hash\ function = (H(x) + i^2) mod\ value$

To Delete

```
$val = (H(value) + 0) mod\ X$
$if\ closed\ hash[val]\ has\ value$
    $set\ closed\ hash[val]\ to\ empty$
$else$
$PROBE\ again\ by\ incrementing\ i\ by\ 1\ so\ at\ next\ iteration\ new\ value\ will\ be$
    $val_1 = (H(value) + 1^2) mod\ X$
```

To Insert

```
$val = (H(value) + 0) mod\ X$
$if\ closed\ hash[val]\ is\ empty$
    $closed\ hash[val] = value$
$else$
$PROBE\ again\ by\ incrementing\ i\ by\ 1\ so\ at\ next\ iteration\ new\ value\ will\ be$
    $val_1 = (H(value) + 1^2) mod\ X$
```

All operations will be performed in constant time

7)

```
initial_table_size = old_table.getSize()

new_table_size = 2*initial_table_size

new_hash = hash_closed(new_table_size)

for item in old_table

        if(!empty(new_hash[hash(item)]))

                new_hash.setvalue(hash[item], item)

        else

                ret = probe(hash[item], ‘quadratic’)

                if ret != ‘Null’

                        new_hash.setvalue(ret, item)

                else

                        continue
```