CS 260 – Homework #4

Chris Kasper

February 2, 2018

Problem 1

A binary tree has n nodes. What is the maximum height of the tree? What is the minimum height? Explain your answers.

Solution

Maximum height of the tree: n-1
This is because the tree wouldn't have to be distributed evenly.

Minimum height of the tree: log2(n)
This is because the tree with the most even distribution would have the lowest height of n nodes.

Problem 2

A tree (not necessarily a binary tree) has n nodes. What are the minimum and maximum possible heights of the tree? Explain your answer.

Solution

Maximum height of the tree: n-1
This is because the tree wouldn't have to be distributed evenly.

Minimum height of the tree: 0 or 1
If a tree has 0 nodes, then it would have a minimum height of 0. However, if the tree has 1 or greater nodes, the minimum would be 1. This is because there's no restriction on the amount of children a node could have.

Problem 3 (4.6 from text)

For each of the operations and each of the implementations in Exercises 4.4 and 4.5, give the order of magnitude for the running time of the operations on sets of size n.

Solution

	Open Hash Table	**Closed Hash Table**	**Unsorted List**	**Fixed array length ***
MakeNull	O(n)	O(n)	O(n)	O(n)
Union	**?**	**?**	$O(n^2)$	$O(n^2)$
Intersection	**?**	**?**	$O(n^2)$	$O(n^2)$
Member	O(1)	O(1)	O(n)	O(n)
Min	O(n)	O(n)	O(n)	O(n)
Insert	O(1*)	O(1)	O(n)	O(1)
Delete	O(1) or O(1*)	O(1)	O(1)	O(1)

Problem 4 (4.7 from text)

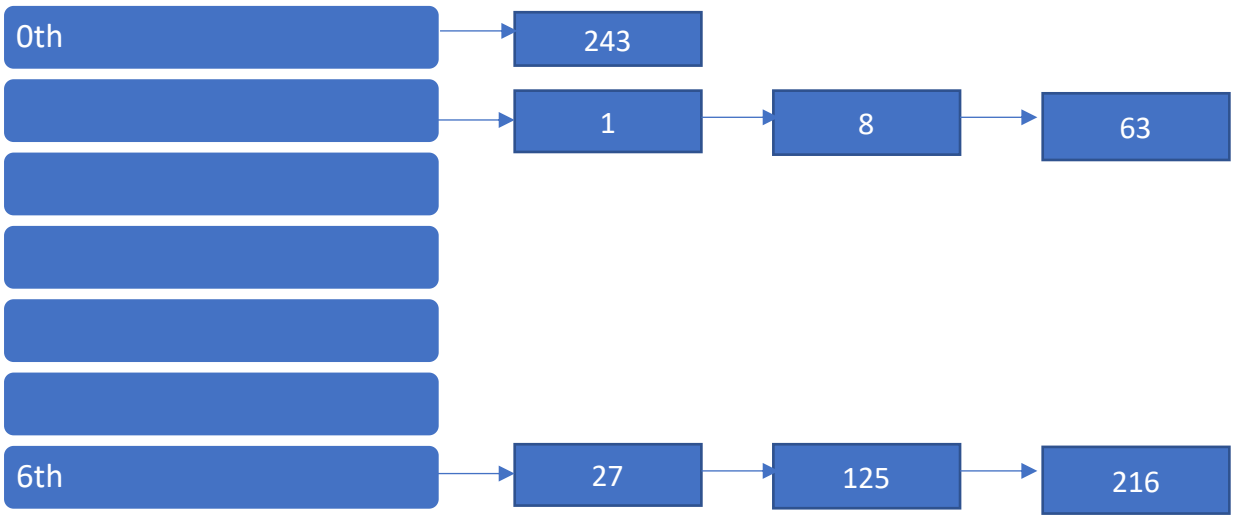
Suppose we are hashing integers with a 7-bucket hash table using the hash function $h(i) = i \textbf{ mod } 7$.

a. Show the resulting open hash table if the perfect cubes 1, 8, 27, 64, 125, 216, 343 are inserted.

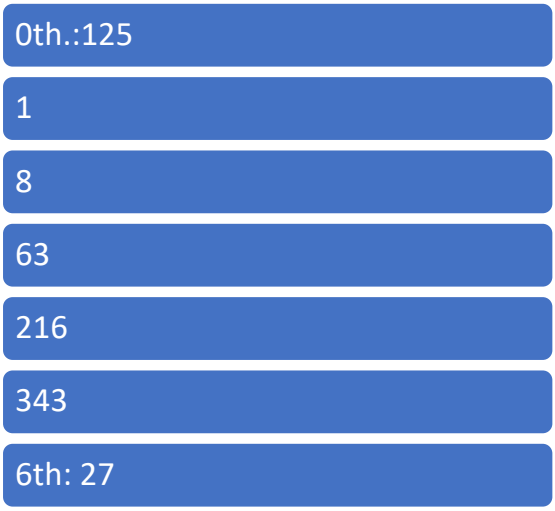
b. Repeat part (a) using a closed hash table with linear resolution of collisions.

Solution

a. Open hash table



b. Closed hash table with linear resolution for collisions



Problem 5

Following are several hash functions. None of them are very good as hash functions. Explain why they are not good hash functions.

1. Hash keys are character strings. The hash function h1(x) computes the length of the string.
2. The function h2(x) computes a random number r with $1 \leq r \leq$ B, where B is the number of buckets. It returns r.

Solution

1. This is not a good hash function because while the hash keys are unique strings, they can have the same length as other strings. This leads to cluttering.
2. Having a random hash computation leads to a random distribution. This is not ideal because you want the most even distribution as possible.

Problem 6

Design an efficient data structure for representing a subset S of the integers from 1 to n. The operations we wish to perform are these:

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

Solution

Closed Hash Table of size(n)
Each bucket has an attribute: occupied (1 means the space is occupied)
H(i)=i-1

To delete integer i from the set, first compute its hash, and then check to see if the space is occupied. If it is, then set S[Hash] to nil, and set the occupied attribute to 0. Do nothing otherwise.

To insert integer i, first compute the hash, and then check to see if the space isn't occupied. If it isn't, set S[Hash] to I, and set the occupied attribute to 1. Do nothing otherwise.

O(1) Time complexity
O(n) Space complexity

Problem 7

Write a procedure (in pseudocode!) to increase the number of buckets in a (closed) hash table. Analyze its time and space complexity.

Solution

Assuming the closed hash table is a list of lists

```
Add_Buckets(list CHT, var n){  CHT is the hash table you wish to add to
        var i = 0;                      n is the number of buckets to add
        while i < n {
                CHT.append([ ])
                i+=1
        }
}
```

O(n) Time complexity
O(n) Space complexity