CSCI 705 Algorithms
Homework 5
Due 3/22/2024

Saad Mneimneh
Computer Science
Hunter College of CUNY

Readings
Based on Hashing, Binary Search Trees, and Red-Black Trees.

Problem 1
An array of size n has a majority element x iff x appears more than $n/2$ times in A. Of course, not every array has a majority element.

(a) Describe how you can use hashing to find the majority element in expected $O(n)$ time.

Note: If elements are shown one at a time, you can easily modify your algorithm to provide the majority element (or declare that it does not exist) in expected $O(1)$ time upon seeing every new element.

(b) Describe an $O(n)$ time algorithm to do the same.

Problem 2

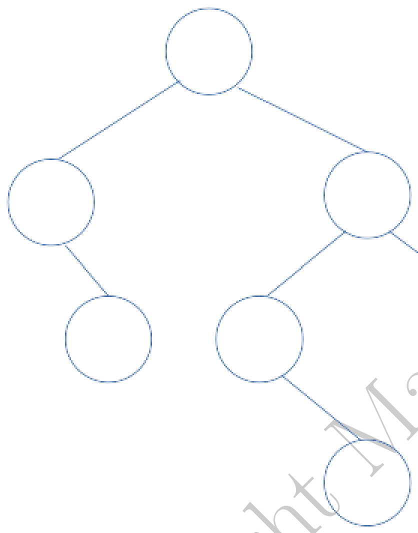
(a) Design an algorithm that converts a binary search tree on n elements into a min-heap containing the same elements in $\Theta(n)$ time.

(b) (Optional) Can you do the same **inplace**, i.e. convert the tree into a min-heap in $\Theta(n)$ without using any extra storage. Here adopt the standard representation of a tree for the heap with the three $left$, $right$, and $parent$ pointers. For this, you need to have a precise pseudocode to show how you will utilize existing pointers in the tree.

(c) Can you reverse the process, i.e. convert a min-heap containing n elements into a binary search tree on the same elements in $\Theta(n)$ time? How, or why not?

Problem 3

(a) Place the keys $\{1, 2, 3, 4, 5, 6, 7\}$ in the following binary search tree.



(b) Color the nodes red and black to make the tree a valid red-black tree.

(c) Perform a Right-Rotate at the node containing key 6. Draw the binary search tree that results and label the tree with the same keys. Can you color the nodes to make the tree a valid red-black tree? Explain.

Problem 4 (Optional)
This came up in class discussion. Consider a hash function for strings where

$$h(s) = \Big(\sum_{i=0}^{n-1} s[i](m+1)^i\Big) \bmod m$$

where s is a string of length n, and $s[i]$ is its i^{th} character interpreted as an integer in $[0, m]$. Prove that swapping two characters in s produces the same hash value. *Hint*: Use the binomial theorem.

Problem 5 (Understand a little more the math of universal hashing)
In the analysis of hash tables, we often make the assumption that the hash function has the simple uniform hashing property: This means that given a key k, $P[h(k) = i] = 1/m$ for every k and $i = 0 \ldots m-1$, **independently** of where any other element hashes to.

(a) Show that a universal hash family does not imply simple uniform hashing. *Hint*: You can show this by an example of a universal hash family that always maps a given key to the same value.

We say that a family $\mathbb{H}$ of hash functions is 2-universal if for every pair of different keys (x, y), and for every h chosen at random from $\mathbb{H}$, $(h(x), h(y))$ is equally likely to be any of the m^2 possible values $(0, 0), \ldots, (m-1, m-1)$.

(b) Show that a 2-universal family is universal and has the simple uniform hashing property. (Therefore, a universal family is not necessarily 2-universal.)

Hint: Show $P(h(x) = h(y)) = 1/m$ and that $P(h(k) = k) = 1/m$. Finally, find $P(h(x) = k|h(y) = \ell)$.

(c) Let m be prime. Suppose we modify the universal hash family we saw in class in the following way (where each key is divided into r chunks, each of which is in $[0, m-1]$):

$$h(x_0, \ldots, x_{r-1}) = \left(b + \sum_{i=0}^{r-1} a_i x_i\right) \bmod m$$

and $(a_0, \ldots, a_{r-1}, b)$ are chosen at random in $[0, m-1]$. Show that this family is 2-universal. *Hint*: Assume $h(x) = k$ and $h(y) = \ell$ and subtract them to perform an analysis similar to the one we did in class.

(d) Suppose $\mathbb{H}$ is universal, and we choose a hash function randomly from $\mathbb{H}$ and keep it secret. Suppose that an adversary who knows $\mathbb{H}$ learned the value of $h(x)$. Show that the adversary can now find a $y \neq x$ such that $h(x) = h(y)$ with probability greater than $1/m$. Show this cannot be the case for a 2-universal hash family.

Hint: The way I approached this problem for universal hashing is by considering three keys x, y, and z that hash into $\{0, 1\}$ (so $m = 2$), and constructing a universal hash family with 2 functions h_1 and h_2 such that the family is universal. Then explored what choices the adversary can make. For 2-universal hashing, one should be able to argue that the adversary's chance is at most $1/m$ by looking for $P(h(x) = h(y)|h(x) = k)$.