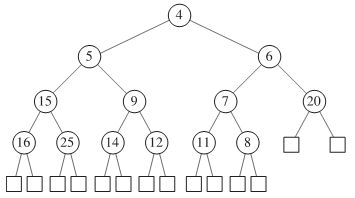
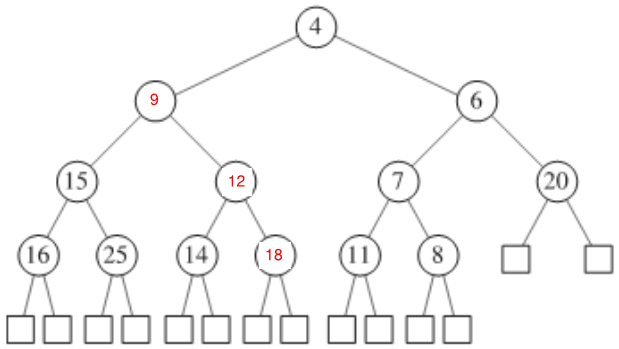
Anthony Rusignuolo

CS 600WS – Advanced Algorithms

Reza Peyrovian

Homework 3

I pledge my honor that I have abided by the Stevens Honor System.

1. R-5.6 Give an example of a worst-case list with n elements for insertion-sort, and show that insertion-sort runs in Ω(n2) time on such a list.
   1. list WC = [1,2,3,4,5] or any list whose values are in increasing order.  
      Every time an item is taken from WC (from the front to the back) and inserted into the priority queue P it will try to also put it at the front of P, but it will obviously be larger than all the values in the list and therefore have to compare to all of them resulting in comparisons. That is therefore going to run in Ω(n2) time.
2. R-5.14 Show the steps for replacing 5 with 18 in the heap of Figure 5.6.
   1.   
      If 18 replaced 5, it would be swapped with 9, then swapped with 12 to reveal the following tree  
      
3. C-5.9 Let T be a heap storing n keys. Give an efﬁcient algorithm for reporting all the keys in T that are smaller than or equal to a given query key x (which is not necessarily in T). For example, given the heap of Figure 5.6 and query key x =7, the algorithm should report 4, 5, 6, 7. Note that the keys do not need to be reported in sorted order. Ideally, your algorithm should run in O(k) time, where k is the number of keys reported.
   1. Algorithm keyFinder(T, x)  
       root = t.root()  
       Algorithm keyFinderHelper(node, x)  
       if x < node.key()  
       return []  
       if x == node.key()  
       return [node.key()]  
       if x > node.key()  
       return [node.key()] + keyFinderHelper(node.left, x) + keyFinderHelper(node.right, x)  
       return keyFinderHelper(root, x)
4. A-5.3 Suppose you work for a major airline and are given the job of writing the algorithm for processing upgrades into ﬁrst class on various ﬂights. Any frequent ﬂyer can request an upgrade for his or her up-coming ﬂight using this online system. Frequent ﬂyers have different priorities, which are determined ﬁrst by frequent ﬂyer status (which can be, in order, silver, gold, platinum, and super) and then, if there are ties, by length of time in the waiting list. In addition, at any time prior to the ﬂight, a frequent ﬂyer can cancel his or her upgrade request (for instance, if he or she wants to take a different ﬂight), using a conﬁrmation code they got when he or she made his or her upgrade request. When it is time to determine upgrades for a ﬂight that is about to depart, the gate agents inform the system of the number, k, of seats available in ﬁrst class, and it needs to match those seats with the k highest-priority passengers on the waiting list. Describe a system that can process upgrade requests and cancellations in O(log n) time and can determine the k highest-priority ﬂyers on the waiting list in O(k log n) time, where n is the number of frequent ﬂyers on the waiting list.
   1. Using a priority queue implemented with a heap and the keyFinder algorithm written above for question C-5.9 will allow for the desired functionality in O(klogn) time. For this to work, the flyer’s key will be their status and their time in line would be their value in the queue.
5. C-6.6 A ***multimap*** is data structure that allows for multiple values to be associated with the same key. It has a put(k, v) method, which inserts an item with key k and value v even if there is already an item with key k (but not the same key-value pair), and a FindAll(k) method, which returns all the values that have the key k. Describe a scheme that implements a multimap so that the put(k, v) method runs in O(1) expected time and the FindAll(k) method runs in O(1 + s) time, where s is the number of values with key k.
   1. A hashmap whose values are all linked lists. Therefore, anytime a value is inserted, the key is looked up in O91) time and appended to the linked list associated in O(1) time as well. Retuning all values would simply entail looking up a key in O(1) and returning all values in the list O(s).
6. A-6.4 Sports announcers are expected to keep talking during a broadcast of a sporting event even when there is nothing actually happening, such as during half-time. One common way to ﬁll empty time is with sports trivia. Suppose, then, that you are going to be a sports announcer for the big game between the Bears and the Anteaters. To ﬁll the empty time during half-time, you would like to say that this is the nth time that a game between the Bears and Anteaters has had a score of i-versus-j at half-time. The problem is that you don’t know the values of i and j yet, of course, because the game hasn’t happened yet, and, once half-time arrives you won’t have time to look through the entire list of Bear-Anteater half-time scores to count the number of times the pair (i, j) appears. Describe an efﬁcient scheme for processing the list of Bear-Anteater half-time scores before the game so that you can quickly say, right at the start of half-time, how many times the pair (i, j) has occurred at similar moments in the past. Ideally, you would like the processing task to take time proportional to the number of previous games and the querying task to take constant time.
   1. Simply create a hashtable using cuckoo hashing to reduce collisions and maintain the speed of searching. The keys would be pairs of scores (i, j) and the values would be the number of games with the same scores at half time. The method for populating the previous scores would run in O(n) time, n being the number of games, because it would simply have to create an instance in the hash table with a value of 0 if the key didn’t already exist, or increment the existing value if the key did exist. Querying the data would run in O(1) time because of the cuckoo implementation.
7. A-6.5 Imagine that you are building an online plagiarism checker, which allows teachers in the land of Edutopia to submit papers written by their students and check if any of those students have copied whole sections from a set, D, of documents written in the Edutopian language that you have collected from the Internet. You have at your disposal a parser, P, that can take any document, d, and separate it into a sequence of its n words in their given order (with duplicates included) in O(n) time. You also have a perfect hash function, h, that maps any Edutopian word to an integer in the range from 1 to 1,000,000, with no collisions, in constant time. It is considered an act of plagiarism if any student uses a sequence of m words (in their given order) from a document in D, where m is a parameter set by parliament. Describe a system whereby you can read in an Edutopian document, d, of n words, and test if it contains an act of plagiarism. Your system should process the set of documents in D in expected time proportional to their total length, which is done just once. Then, your system should be able to process any given document, d, of n words, in expected O(n + m) time (not O(nm) time!) to detect a possible act of plagiarism.
   1. To process the set of documents, D, lists of m words would be stored in a hash table using a polynomial hash function to reduce the possibility of collisions. Then, a similar process would occur for the given document, d. The hash table for document d would then be traversed and checked against the hash table for D resulting in an O(n + m) time.