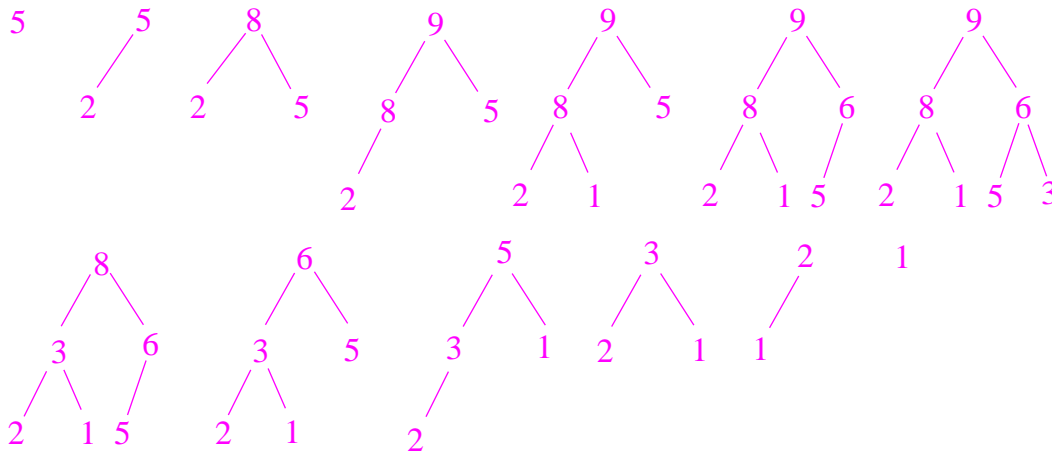


COMP3600/COMP6466 in 2009 – Tutorial Three

Question 1.

Insert the keys 5, 2, 8, 9, 1, 6, 3 into a max-heap one at a time, then remove the key in the root repeatedly until the heap is empty. What is the time complexity of sorting in this fashion?



The time is $O(n \lg n)$, which is not surprising since this is just heap-sort.

Question 2.

In the open addressing schema of Hash table, three probing techniques have been introduced, they are **linear probing**, **quadratic probing**, and **double hashing**. Point out how many different probing sequences for each of the schemes. Compare the advantages and disadvantages of each of the techniques.

Assume that there are m slots in the hash table. Then,

- there are $O(m)$ probing sequences for **linear probing**,

$$h(k, i) = (h'(k) + i) \pmod n,$$

$$0 \leq h'(k) \leq m - 1 \text{ and } 1 \leq i \leq m.$$

- there are $O(m)$ probing sequences for **quadratic probing**,

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \pmod n,$$

$$0 \leq h'(k) \leq m - 1, 1 \leq i \leq m, c_1 \text{ and } c_2 \text{ are constants.}$$

- and there are $O(m^2)$ probing sequences for double hashing.

$$h(k, i) = (h_1(k) + ih_2(k)) \pmod n,$$

$$0 \leq h_1(k), h_2(k) \leq m - 1 \text{ and } 1 \leq i \leq m.$$

- **Linear probing** is very simple, and takes less time, but it suffices the primary clustering problem.
- **Quadratic probing** avoids the primary clustering problem, its computation is easy, but it suffices the secondary clustering problem.
- **Double hashing** is a ideal hash approach. Although it prevents both primary and secondary clustering problems, it is more complicated and take much longer running time for hashing.

Question 3.

Design a hash function for the open addressing schema such that there are no primary and secondary clustering problems by using it for hashing. In addition, the number of probing sequences derived from it is no less than $O(m^3)$.

Let $h(k, i) = (h_1(k) + i * h_2(k) + i^2 * h_3(k)) \pmod m$, where $h_j()$ is a hash function for j , $1 \leq j \leq 3$. Clearly, this hash function avoids the primary and secondary clustering problems, and the total different number of probing sequences derived from it is $O(m^3)$, due to $h_1(k), h_2(k), h_3(k) \in \{1, 2, \dots, m\}$.

Question 4.

Almost all analysis of binary search trees assumes that the keys are distinct. Suppose the definition of the binary search tree ordering is changed to:

For each node, if the key is K , then:

- * the key in the left child (if any) is $\leq K$
- * the key in the right child (if any) is $> K$

Redesign the insertion algorithm so that all insertions create a new node, even if the key is already present. Explain how to search in such a tree.

If the existing algorithm is followed, the nodes with a given key can be separated in the tree by other nodes, which is inconvenient. Try inserting keys 4, 3, 4, 3, 4 in that order. It is better to keep all the nodes with the same key in a single path. To do this: while inserting key K , if an existing node P with key K is encountered, insert the new node between P and its left child. For searching: once one node with the requested key is found, scan down to the left to collect all the others.

Question 5.

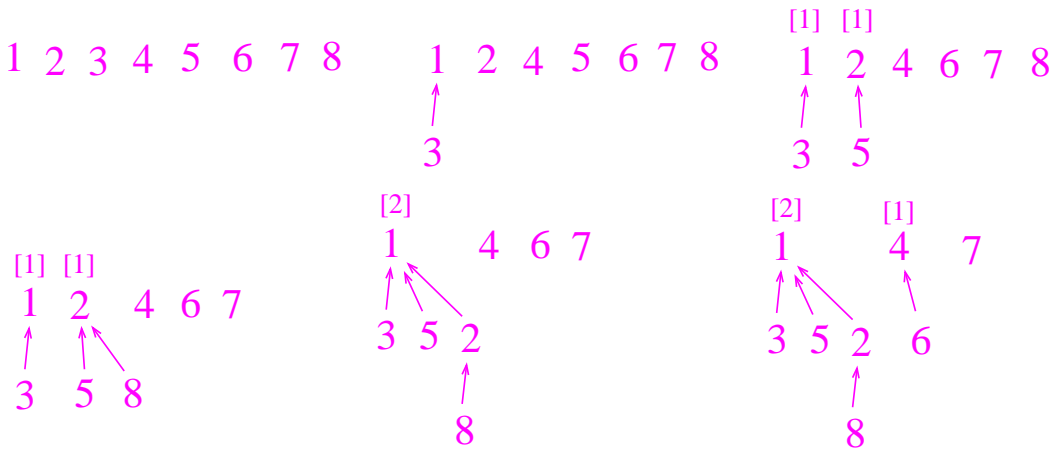
- (i) What is the difference between the binary search tree and the red-black tree?
- (ii) What is the left/right rotation? what's its purpose when applied in the red-black trees.

(i) The difference between the binary search tree and the red-black tree is that the height of binary search tree is unbounded, which means its height may be $O(n)$, while the height of a red-black tree is always bounded by $O(\log n)$, which means each of the operations on it takes $O(\log n)$ time.

(ii) The left/right rotation at a node is usually used to reduce the height difference between its left and right subtrees. However, such a transformation maintains the binary search tree property. The left/right rotation applied to the red-black tree is to maintain the properties of red-black tree properties, specially they are used when a node inserts to and deletes from a red-black tree.

Question 6.

Apply the directed tree implementation of the disjoint sets data structure, using both heuristics, to find the connected components of the graph with 8 vertices and edges provided in this order: 1-3, 2-5, 2-8, 3-5, 4-6



The numbers in square brackets are ranks. There is a choice when merging two trees of the same rank, so other solutions are possible.

Question 7.

Given an undirected graph with n vertices and m edges, show how DFS can be used to find a cycle (or prove there isn't one) in $O(n)$ time. Recall: DFS in general takes more time than that.

Run DFS until either a back edge is found (which gives a cycle) or the DFS finishes. Since there are at most $n - 1$ tree edges, one of the first n edges examined must be a back edge unless there are no back edges at all.

