CSE221: Data Structures
Sample Final Exam: Suggested Answers

Antoine Vigneron

December 10, 2021

This sample midterm is for you to see what type of questions will be asked during the final exam. Solutions are given at the end.

Question 1:
(Textbook R-6.4) Draw a representation of an initially empty vector A after performing the following sequence of operations: insert(0, 4), insert(0, 3), insert(0, 2), insert(2, 1), insert(1, 5), insert(1, 6), insert(3, 7), insert(0, 8).

Solution 1:
(8,2,6,3,7,1,4)

Question 2:
(Textbook C-6.5.) Describe a function for performing a card shuffle of an array of $2n$ elements, by converting it into two lists. A card shuffle is a permutation where a list L is cut into two lists, L_1 and L_2 , where L_1 is the first half of L and L_2 is the second half of L, and then these two lists are merged into one by taking the first element in L_1, then the first element in L_2, followed by the second element in L_1, the second element in L_2, and so on.

Solution 2:
The pseudocode is given below

```
procedure SHUFFLE($A$, $n$)
    $T \leftarrow A$
    for $i \leftarrow 0, n-1$ do
        $T[2*i] \leftarrow A[i]$
        $T[2*i+1] \leftarrow A[i+n]$
    return $T$
```

Question 3:
Consider the tree T shown in Figure 1. Give the order in which the nodes appear in a (i) preorder, (ii) inorder and (iii) postorder traversal of T.

Solution 3:
(i) a,b,d,e,c,f (ii) d,b,e,a,f,c (iii) d,e,b,f,c,a

Question 4:

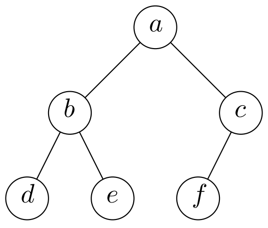

Figure 1:

Suppose that a full binary tree has p internal nodes and q leaves. Prove that $q = p + 1$.

Solution 4:
We prove it by induction. Let T be a tree with p internal nodes and q leaves. We make a proof by induction.

- Base case: If $p = 0$, then T consists of a single root node. This node is a leaf, so $q = 1 = 1 + p$.
- Inductive case: Suppose that $q \geqslant 1$. Let v be an internal node of T with largest depth. Then v has two children that are leaves. Let T' be obtained by removing these two leaves from T. Then T' has $p' = p - 1$ internal nodes and $q' = q - 1$ leaves, because we replace a subtree consisting of 1 internal node and two leaves with a subtree consisting of one leaf. Thus by induction, we must have $q' = 1 + p'$, which implies that $q = 1 + p$.

Question 5:
(Textbook C7-6) Give an $O(n)$-time algorithm for computing the depth of all the nodes of a tree T , where n is the number of nodes of T.

Solution 5:
It can be done by DFS. The pseudocode is given below. In order to compute all the depths, call ALLDEPTHS(T, T.root(), 0).

```
procedure ALLDEPTHS($T$, $v$, $d$)
    $v$.depth $\leftarrow d$
    for each child $c$ of $v$ do
        ALLDEPTHS($T$, $c$, $d + 1$)
```

Question 6:
(Textbook C-7.24) Let T be a tree with n nodes. Define the lowest common ancestor (LCA) between two nodes v and w as the lowest node in T that has both v and w as descendents (where we allow a node to be a descendent of itself). Given two nodes v and w, describe an efficient algorithm for finding the LCA of v and w. What is the running time of your method?

Solution 6:
Here is a description of an $O(h)$-time algorithm, where h is the height of T.

- First compute the path $v = v_k, v_{k-1}, \ldots, v_2, v_1 = T$.root() by following the parent pointers until the root is reached.

- Compute the path $w = w_q, w_{k-1}, \ldots, v_2, v_1 = T.\text{root}()$ in the same way.
- Find the largest i such that $w_i = v_i$. Then the LCA is this node $w_i = v_i$.

Question 7:
What is the most appropriate choice for implementing a priority queue:

(a) A singly-linked list

(b) A hash table with separate chaining

(c) A heap

(d) A hash table with open addressing

Solution 7:
(c)

Question 8:
(Textbook R-8.5) Suppose you label each node v of a binary tree T with a key equal to the preorder rank of v. Under what circumstances is T a heap?

Solution 8:
If it is a *complete* binary tree.

Question 9:
(Textbook C-8.4) Show how to implement the stack ADT using only a priority queue and one additional member variable.

Solution 9:
Let S be our stack. We add one member variable which is an integer r. To perform a push(e) operation, we decrement r, and insert the pair (e, r) into the priority queue Q. We say that $(e, r) < (e, r')$ whenever $r < r'$. Then S.top() is implemented by computing $(e, r) = Q$.removeMin(), and returning e. The S.pop() operation is just Q.removeMin().

Question 10:
(Textbook R-9.7) Draw the 11-entry hash table that results from using the hash function, $h(i) = (3i + 5) \bmod 11$, to hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, and 5, assuming collisions are handled by chaining.

Solution 10:
Here is the result:

```
0:  13
1:  94, 39
2:
3:
4:
5:  44, 88, 11
6:
7:
8:  12, 23
9:  16, 5
10: 20
```

Question 11:
(Textbook C-9.11) Suppose that each row of an $n \times n$ array A consists of 1's and 0's such that, in any row of A, all the 1's come before any 0's in that row. Assuming A is already in memory, describe a method running in $O(n \log n)$ time (not $O(n^2)$ time!) for counting the number of 1's in A.

Solution 11:
For each row, we can find the index of the last '1' by binary search in $O(\log n)$ time. Then we can just sum up these indices, which yields the desired result. Overall, this algorithm takes $O(n \log n)$ time.

Question 12:
Draw all the binary search trees that store the keys {1, 2, 3}.

Solution 12:
There are 5:

```
  2      1        1           3        3
 / \      \        \         /        /
1   3      2        3       2        1
            \      /       /          \
             3    2       1            2
```

Question 13:
(Textbook R-13.8) Would you use the adjacency list structure or the adjacency matrix structure in each of the following cases? Justify your choice.

a. The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.

b. The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.

c. You need to answer the query isAdjacentTo as fast as possible, no matter how much space you use.

Solution 13:

a. I would use an adjacency list, because an adjacency matrix would have $(20,000)^2 = 400,000,000$ cells, so it would be much larger that an adjacency list of 20,000 edges.

b. I would still use an adjacency list, because the adjacency matrix has 20 times more cells than there are nodes in the adjacency list, so even though the adjacency list records extra pointers, it should still be better.

c. The adjacency matrix is better, because it answers these queries in $O(1)$ time, while an adjacency list may need $\Omega(n)$ in the worst case.

Question 14:
(Textbook C-13.18) Design an efficient algorithm for finding a longest directed path from a vertex s to a vertex t of an acyclic weighted digraph G. Specify the graph representation used and any auxiliary data structures used. Also, analyze the time complexity of your algorithm.

Solution 14:
We use an adjacency list representation.

- Compute a topological ordering of the graph. Discard the nodes that come before s or after t, obtaining a sequence $s = v_0 < v_1 < \cdots < v_k = t$.
- Let $D[0 \dots k]$ be an array storing the length of the longest path to $v_0, \dots, v_k$
- Traverse the array D from $i = 0$ to $i = k$. At each step, set $D[i]$ to me the maximum of $D[c] + w(c, i)$ over all incoming edges (c, i).
- Now $D[k]$ records the *length* of the longest path to t.
- To return the path itself, go backwards from t, finding for each v_i in the path the ancestor c that maximized $D[c] + w(c, i)$

This algorithm runs in $O(n + m)$ time, because we process each vertex and each edge at most once in the forward and the backward phase.