CS3 - Data Structures Semester Final Examination

Indian Statistical Institute, Bangalore

November 2016. Max marks 50; Max time 3 hours

Q1. ($6 \times 2 = 12$) Short answers.

(a) A method to test if a number p is prime is to test if there is some number between 2 and $\sqrt{p}$ which divides p. Clearly this algorithm does $\Theta(\sqrt{p})$ divides, thus it has a better complexity than an algorithm to add p numbers which is $\Theta(p)$. True or False? Why? For this argument assume a divide or an add is a constant time operation.

(b) If T is an m-way search tree (not necessarily balanced) of height h on n elements, what are the lower and upper bounds on n in terms of h and m?

(c) Lets assume that one can find the middle value of n array elements in $\Theta(n)$ time. If we use that to split the array into equal parts, rest of quicksort being the same what is the worst case running time of quicksort?

(d) In a min-heap on an array of n elements, what is the time to restore the heap property if an arbitrary element's value is changed.

(e) An array of n elements happens to be sorted in increasing order. The usual $\Theta(n)$ algorithm to create a min-heap on it is applied. How many moves are performed to create the heap? How many compares?

(f) We need to perform the following operation on a directed graph with n vertices and m edges: `For each vertex, update the weight of each edge out of that vertex by one.` What is the complexity of this operation if we use (i) adjacency list representation (ii) adjacency matrix representation.

Q2. (2+2+1+1=6) You have n student *records* in an array, each of which is a large structure with about fifty items pertaining to all of the student's information over his time in the college. You don't want to move those records around and you don't want to copy them. You are asked to do what it takes to support quickly searching for students given a combination of hostel name and room number, note that rooms are shared and you need to find all the students in that room. Searching fast is of prime importance, the other things like how much time it takes to build the data structure is unimportant.

(a) Give three options you may consider for a data structure for this? What is your choice among these and why?

(b) Briefly describe how the insert and the search work for your choice.

(c) What is the time to build your data structures of choice?

(d) What is the time to search in your data structure in each case?

Q3. (1+4+1=6) You are given a maze which is an $n \times n$ square grid with some transitions between adjacent squares allowed and some not allowed. You are also given a *start* square and a *end* square and the problem is to find a path from the start square to the end square.

(a) How will you model a maze as a graph $G(V, E)$?

(b) Given this model, write out the algorithm to find a *shortest* path and print all the edges along the path.

(c) What is the time complexity of your algorithm?

Q4. (6+2+2=10) Prim's minimum weight spanning tree algorithm for undirected graphs finds a minimum spanning tree (*MST*) of $G(V, E)$ starting from an arbitrary vertex of V, maintaining the vertices of $V - T$ in a heap ordered by their $d[v]$ values, where T is the partly constructed MST at any stage. Finally T is an MST.

(a) A step in Prim's algorithm is to possibly change the d values of vertices adjacent to a vertex v extracted from the top of the heap. Written loosely it is as below:

```
forall u in Adj(v):
    if u is not already in T
         if w(v,u) < d[u]
             d[u]=w(u,v)  /* w(u,v) is the weight of the edge (u,v) */
             p(u)=v   /* (u,p[u]) is in the MST when u is included in T */
```

This requires one to (i) find where u is in the heap and then (ii)adjust the heap because $d[u]$ is now decremented. How may one do each of these steps efficiently?

(b) Show that if weights of all edges in E are increased by the same value, then T is a MST of G iff T is an MST in the new graph.

(c) Does Prim's algorithm work even if G has negative edge weights? Give an argument or a counter-example.

Q5. ($4 \times 1+4=8$) DFS of a DAG $D(V, E)$ produces a collection of trees $F = t_1, t_2, ..., t_k$. Assume we also record the start and finish times of each vertex v during the search $(s[v], f[v])$. Also a topological sort is an ordering of vertices of $V : v_1, v_2, ..., v_n$ such that if $\overrightarrow{(v_i, v_j)} \in E$ then $i < j$. For (a) to (d) below prove or disprove the statement. For (e) write pseudo code.

(a) If $\overrightarrow{(u, v)}$ is a cross edge then the interval $[s(v), f(v)]$ is a sub interval of $[s(u), f(u)]$.

(b) If u has non-zero indegree and non-zero out-degree then the tree t_i containing u has at least two vertices.

(c) If every topological sort of D has the same vertex u as its first vertex then that vertex is the unique degree zero vertex in D.

(d) If u is a leaf of t_i for some i then outdegree(u) $= 0$.

(e) An algorithm for topological sort is to repeatedly eliminate an in-degree zero vertex from D. Show how to do this in $\Theta(|V| + |E|)$ time.

Q6. (3+2+3=8) For a BST the height $h(v)$ of a node v is defined as the maximum number of vertices on a path from v to a leaf. Height of an empty tree is defined as zero.

(a) For a given BST, show how to use post order traversal to determine the height of every node and in the process compute the balance factor of each node defined as $BF(v) = h(left(v)) - h(right(v))$.

(b) Show all the intermediate and final trees if inserts happen in order of the given keys $(0, 5, 10, 15, 1, 2)$. Write out the balance factors of all the nodes after each insert.

(c) What would be the corresponding trees if this was an AVL tree. Indicate the balancing rotations that occur.