CSE 102: Data Structures and Algorithms (Sections A and B); END-SEM Exam Aug 1, 2022; 11 questions, total 60 marks.

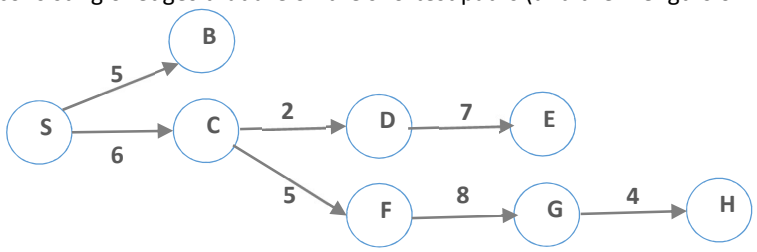
1. **(8 marks)** Suppose that we want to use AVL trees as a min-priority-queue data structure. That is a data structure S supporting the following operations.
   - **Insert(S, x):** Inserts the element **x** into the set **S**. This operation could be written as **S ← S U {x}.**
   - **Minimum(S):** Returns the element of **S** with the minimum key.
   - **Extract-Min(S):** Removes and returns the element of **S** with the smallest key.
   - **Decrease-Key(S, x, k):** Decreases the value of element **x**'s key to the new value k, which is assumed to be at least as large as **x**'s current key value.

   What is the complexity of the above operations? Justify your answer. Note: the most efficient algorithms will fetch full marks.

2. **(5 marks)** Prove that the height of an AVL tree is **O(log(n))**.

3. **(7 marks)** Perform the **Build-Max-Heap** operation (i.e., building the max heap using the bottom-up approach) on the following array elements in the given order. Draw the initial tree using the array elements and then draw the resulting tree after every intermediate step during the Build-Max-Heap operation. The root of the initial tree is the first element of the array, i.e., 1. Array elements: 1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 13, 12.

4. **(3 marks)** We ran Dijkstra's algorithm on a graph, and computed the shortest paths from vertex **S** to other vertices. The resulting tree consisting of edges that are on the shortest paths (and their lengths or weights) are shown – others are not shown.



   a. What is the length of the shortest path from **S** to each vertex, **B, C, D, E, F, G, H**?
   b. What is the order of vertices in which their shortest paths are computed when we use Dijkstra's algorithm? *What property of Dijkstra's algorithm makes it possible for you to determine the order?*

5. **(4 marks)** This is in continuation with Question 1. Each vertex (or node) **V** maintains a "routing table" that takes the form of a dictionary, RT-at-V = {(f1, n1), (f2, n2), …}, where **f1** or **f2** is a "final destination node" and **n1** or **n2** is the corresponding "next node" on the shortest path to **f1** or **f2**. As an example, <u>one</u> entry in routing table at **C**, **RT-at-C**, is **(E, D).** In other words, the next-note on the shortest path to **E** is **D**. Clearly there may be more entries in **RT-at-C**. As another example, the **RT-at-E** will have only one entry, viz**. (E, -).** What is the complete routing table at other vertices?

   RT-at-S? RT-at-B? RT-at-C? RT-at-D? RT-at-E? RT-at-F?

6. **(5 marks)** We give below the function **DFS-Visit(G, u)** to do a "depth-first search" on a <u>directed</u> graph, **G = (V, E),** starting at vertex **u**. But, before function **DFS-Visit(G, u)** is called the following initialization is done:

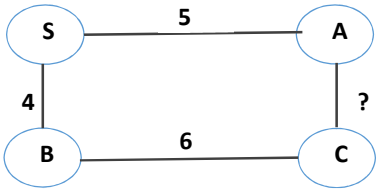
```
for each vertex u ε G.V
      u.color = 'white'

//Recursion-based algorithm to do "depth-first search"
DFS-Visit(G, u)
      u.color = 'grey'
      for each vertex v ε G.Adj[u]
            if v.color == 'white'
                  DFS-Visit(G, v)
      u.color = 'black'
```

   (a) Indicate as to what is the event during traversal that prompts one to say that one has encountered a directed cycle.
   (b) THEN **modify & re-write the above algorithm** to determine whether the graph is a directed acyclic graph (DAG) or not.

7. **(4 marks)** A simple undirected graph G is given below, together with the weights of only **(S, A), (S, B), and (B, C).** For some reason we would like edge **(B, C)** to be on the shortest path from **S** to **C**. BUT we don't want **(B, C)** to be an edge in the

minimum spanning tree of graph G. What is the range of values of the weight of edge **(A, C)** so that edge **(B, C)** is on the shortest path from **S** to **C**, and is NOT in the minimum spanning tree of G. Give me the range such as **X < weight(A,C) < Y**. *Other than giving the range of values of weight of edge **(A, B)**, argue as to why that is indeed the case*. Please do not use the operator ≥ or ≤ since that will leave the decision on how the adjacency lists are created.

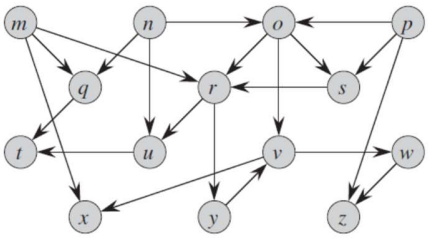


8. **(4 marks)** We will use Hash-tables of size 11 to store and retrieve information as **(key, value)** pairs. Further, **h(k) = k mod 11.** We will use **quadratic probing** to resolve collisions. What are the contents of the Hash-table**, T**, once we insert the following (**key, value**) pairs (in the order left to right)?

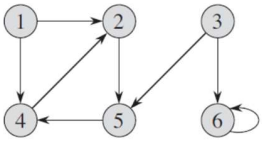
(11, a), (121, b), (33, c), (75, d), (4, e), (93, f), (100, g), (1, h).

Table	0	1	2	3	4	5	6	7	8	9	10
T											

9. **(6 marks)** We are given directed graph with courses, **m** through **z** as vertices that a student must complete to graduate. An edge **<a, b>** in the graph implies that course **a** is a pre-requisite for course **b**. Compute/obtain a linear sequence in which the courses could be taken. Give the DFS tree generated during traversal, as also the start- and finish times for each vertex. NOTE: the graph is represented as collection of adjacency lists, & courses are listed in adjacency lists in alphabetical order.



10. **(8 marks)** A directed graph may be represented using adjacency lists, one for each vertex **X**. The list for vertex **X** links list nodes that point to vertices **To** which there is an edge from **X**. This makes it easy to identify and operate on vertices to which there is an edge. But, identifying and operating on vertices **From** which there is an edge **To** vertex **X** is not so easy. Can you propose a data structure to represent a graph so that it is equally easy to identify and operate on vertices **To** which there is an edge or on vertices **From** which there is an edge. Make sure that the storage required is only marginally more that the storage required for adjacency lists discussed in class. INSTEAD OF WRITING A LONG STORY SIMPLY SHOW THE LISTS AND THEIR CONTENT FOR THE DIRECTED GRAPH GIVEN BELOW.



11. **(6 marks)** Consider using Prim's algorithm to compute a minimum spanning tree, **MST**, for graph below. The first step starts by including the node **A** in the MST. That is **MST = {A}.** At the end of first 3 steps **MST = {A, B, C},** as shown below. What is not shown is the min binary heap that the algorithm uses to store the nodes outside the MST so that the node with the smallest weight to any vertex in MST can be easily found. What is the min binary heap at end of step 1, step 2 step 3.