COMP 3270 Assignment 4 (100 points)

**Due by 11:59PM on Friday, July 28th, 2023**

Instructions:

1. Late submissions **will not** be accepted unless prior permission has been granted or there is a valid and verifiable excuse.
2. Think carefully; formulate your answers, and then write them out concisely using English, logic, mathematics and pseudocode (no programming language syntax).
3. Type your final answers in this Word document.
4. Don’t turn in handwritten answers with scribbling, cross-outs, erasures, etc. If an answer is unreadable, it will earn zero points. **Neatly and cleanly handwritten submissions are acceptable**.

**1. (10 points)** Show d and π values that result from running Breadth First Search on the directed graph below using vertex 3 as the start node.















**2. (10 points)** Show how Depth First Search works on the graph below by marking on the graph the discovery and finishing times (d and f) for each vertex and the classification of each edge. Assume that the for loops in DFS and DFS-VISIT consider vertices alphabetically.

Q= d= 1, f = 16

S= d= 2, f =7

V= d= 3, f = 6

W= d= 4, f= 5

T= d= 8, f= 15

X= d= 9, f = 12

Z= d= 10, f =11

Y= d= 13, f =14

R= d= 17, f= 20

U= d=18, f =19



















**3. (15 points)** List the vertices of the graph below in Topological Order, as produced by the Topological Sort algorithm. Assume that the for loops in DFS and DFS-VISIT consider vertices alphabetically.

















p,n,o,s,m,r,y,v,w,z,u,x,q,t

**4. (15 points)** Do Problem 24.1-1 (p. 654) (you do not have to do the last part, i.e., running the algorithm again after changing an edge weight).

**Run the Bellman-Ford algorithm on the directed graph of Figure 24.4, using vertex z as the source. In each pass, relax edges in the same order as in the figure, and show the d and values after each pass.**

If we change our source to z and use the same ordering of edges to decide what to relax, the d values after successive iterations of relaxation are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | t | x | y | z |
|  |  |  |  | 0 |
| 2 |  | 7 |  | 0 |
| 2 | 5 | 7 | 9 | 0 |
| 2 | 5 | 6 | 9 | 0 |
| 2 | 4 | 6 | 9 | 0 |

The values are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | t | x | y | z |
| NIL | NIL | NIL | NIL | NIL |
| z | NIL | z | NIL | NIL |
| z | x | z | s | NIL |
| z | x | y | s | NIL |
| z | x | y | s | NIL |

**5. (15 points)** Do Problem 24.2-1 (p. 657 of the text). Show the results similar to Fig. 24.5.

**Run DAG-SHORTEST-PATHS on the directed graph of Figure 24.5, using vertex r as the source.**

If we run the procedure on the DAG given in figure 24.5, but start at vertex r, we have that the d values after successive iterations of relaxation are:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| r | s | t | x | y | z |
| 0 |  |  |  |  |  |
| 0 | 5 | 3 |  |  |  |
| 0 | 5 | 3 | 11 |  |  |
| 0 | 5 | 3 | 10 | 7 | 5 |
| 0 | 5 | 3 | 10 | 7 | 5 |
| 0 | 5 | 3 | 10 | 7 | 5 |

The values:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| r | s | t | x | y | z |
| NIL | NIL | NIL | NIL | NIL | NIL |
| NIL | r | r | NIL | NIL | NIL |
| NIL | r | r | s | NIL | NIL |
| NIL | r | r | t | t | t |
| NIL | r | r | t | t | t |
| NIL | r | r | t | t | t |

**6. (15 points)** Do Problem 24.3-1 (p. 662 of the text).

**Run Dijkstra’s algorithm on the directed graph of Figure 24.2, first using vertex s as the source and then using vertex ´ as the source. In the style of Figure 24.6, show the d and values and the vertices in set S after each iteration of the while loop**

We first have s as the source, in this case, the sequence of extractions from the priority queue are: s, t, y,x,z. The d values after each iteration are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | t | x | y | z |
| 0 | 3 |  | 5 |  |
| 0 | 3 | 9 | 5 |  |
| 0 | 3 | 9 | 5 | 11 |
| 0 | 3 | 9 | 5 | 11 |
| 0 | 3 | 9 | 5 | 11 |

**values**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | t | x | y | z |
| NIL | s | NIL | NIL | NIL |
| NIL | s | x | s | NIL |
| NIL | s | x | s | y |
| NIL | s | x | s | y |
| NIL | s | x | s | y |

**(7) (10 points)** Supposethat a graph G has a Minimum Spanning Tree (MST) computed. How quickly can we update the MST if we add a new vertex and incident edges to G. Propose and outline a strategy and present an algorithm (you can reuse graph algorithms covered in class as building blocks as part of your solution) and evaluate its asymptotic complexity.

The minimum spanning tree of a Graph is the union of minimum spanning trees of its connected components.

If we add a new vertex and incident edges to G, then complexity should be O(|E'|) where E' is the new set of edges to connect the incoming component.

Based on the type of MST, algorithm and complexity can however vary.

**For a greedy MST, we try to calculate the path with minimum weight when adding a new path**

Algorithm:

1. The greedy method works on the basis of this selection policy: choose the minimum-weight remaining edge. If that edge does not create a cycle in the evolving tree, add it to the tree.

2. For finding and deleting the min-weight edge, use a minheap where its nodes are the labels+weights of the graph edges.

3. For cycle detection, note that

* T is a forest at any given time,
* adding an edge eliminates two trees from the forest and replaces them by a new tree containg the union of the nodes of the two old trees, and
* and edge e=(x,y) creates a cycle if both x and y belong to the same tree in the forest.

Complexity:

* O(|E|) to build the heap
* up to |E| calls to U and F, taking O(|E|log n) time

therefore, the total time is O(|E|log |E|).

**(8) (10 points)** What is the running time of BFS if we represent its input graph by an adjacency

matrix and modify the algorithm to handle this form of input? Explain and justify your answer.

The time of iterating ll edges becomes O() from O(E). Therefore the running time is O(V+) =

V is the number of vertices or nodes in the graph.

**The reason for the running time is:**

For initializing the Adjacency Matrix: creating an adjacency matrix for a graph with V node requires O(V2) space and time. This is on the grounds that for every node, we want to store its association status with each and every other node.

The BFS algorithm investigates all the nodes and edges in the graph. All for every node, it visits its neighbors. In the nearness matrix representation, checking whether there is an edge between two nodes should be possible in constant time

O(1). Be that as it may, for every vertex, we want to analyze all V nodes to find its neighbors, which brings about O(V) time complexity for every node visit.

For visiting all nodes the worst case scenario in which the BFS will visit all V nodes in the diagram. For every node visit, it takes

O(V) time to really look at its neighbors in the nearness adjacency matrix representation. So the overall time complexity for visiting all nodes becomes

O(V2).