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.

d=

d=3

π =nil

π =4

d=0

π =nil

d=1

π =3

π =5

π =3

d=2

d=1

**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.



**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, x, w, z, u, 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).

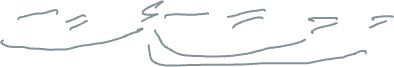
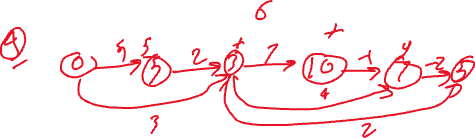
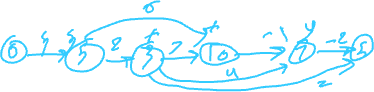
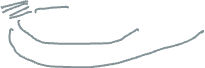
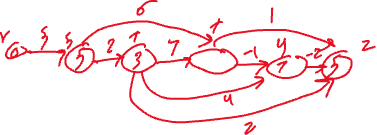
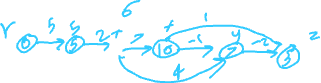
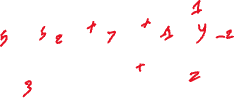
d Values:

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

π Values:

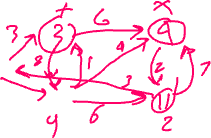
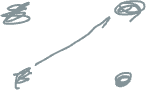
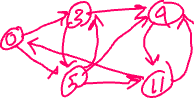
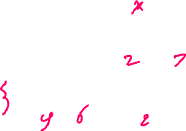
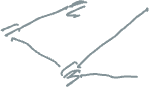
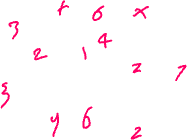
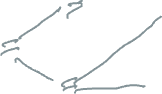
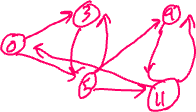
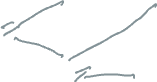
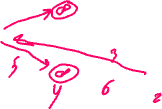
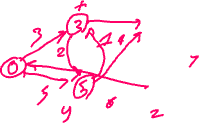
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s: | t: | x: | y: | z: |
| null | null | null | null | null |
| z | null | z | null | null |
| z | x | z | s | null |
| z | x | y | s | null |
| z | x | y | s | null |

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

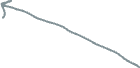
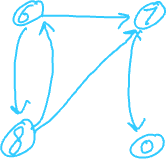
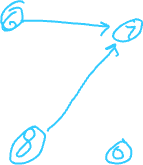
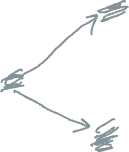
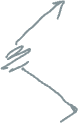
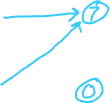
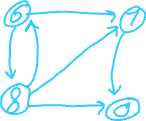
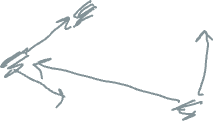
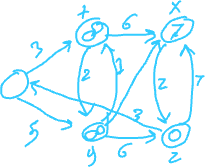
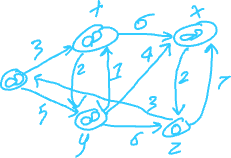


1. **(15 points)** Do Problem 24.3-1 (p. 662 of the text).

S as a source



**Z as a source**



**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.

Let v be a new vertex that will be and let T be the original MST with root s. If we add the lightest indicate edge to T then we getting a new spanning tree T1, for each new edge left say (u , v), we add it to the current tree, Which must result in a cycle, and then we remove the edge with the max weight on the cycle, we will keep doing this you until all the new added edge with are processed.  to find the max-weighted edge in a cycle that must first find then part p, from v to s and part P1 from u to s. From the root s down, we will compare the vertices appearing on p & p1., the first common vertex x on both paths is the least common ancestor of U and V, which is the path from x to u and the path from x to v and the age (u, v) from the cycle. The max-weighted edge on this cycle is the max of edges on the path p & p1, and edge (u , v), this would run in O(ev) time, where e is the number of new edges.

**(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 running time of BFS with an adjacency matrix representation is O(V^2), where V is the number of vertices in the graph. Because in the adjacency matrix representation, finding the neighbors of a vertex requires scanning the entire row of the matrix, which takes O(V) time. As BFS performs this operation for each of the V vertices, the overall time complexity becomes O(V^2). The space complexity remains O(V) as we still need to maintain a queue to store the vertices to be processed