Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Assignment 3 - 5% (40 points)

CSI2110/CSI2510 (Fall 2023)

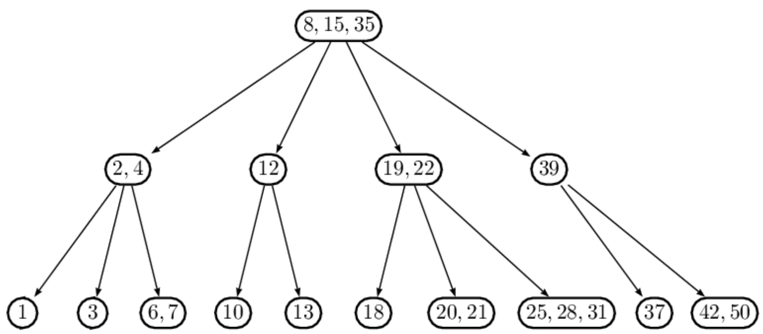
Due: Thursday, October 30, 11:59 PM.

Late assignment policy: ***1min-24hs late are accepted with 30% off; no assignments accepted after 24hs late.***

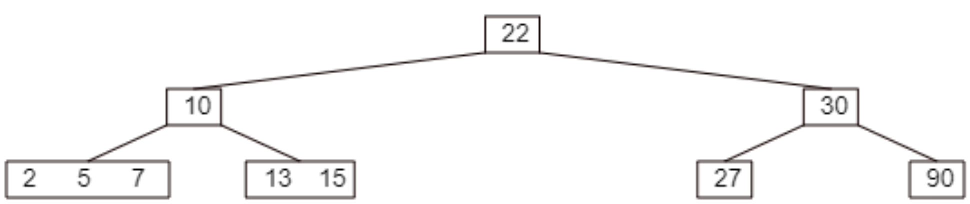
Question 1. **(2,4)-trees (10 points= 5+2.5+2.5)**

a) Insert the keys in the following order: 23, 24, 26 into the (2,4)-tree below. Use the convention used in the class notes: when splitting a 5-node with keys (k1, k2, k3, k4), the new nodes will have keys (k1,k2) and (k4) and k3 is inserted into the parent.

Only show the tree right before and right after each split operation.



b) Delete the key 30 from the following (2,4)-tree and show the resulting tree. Show any essential intermediate step.



c) Delete the key 90 from the following (2,4)-tree and show the resulting tree. Show any essential intermediate step.

Question 2. **Graph traversals (10 points= 1+1+4+4)**

Consider an undirected graph given by the following adjacency list representation

1: (1,2), (1,3), (1,4)
2: (2,1), (2,3) (2,4)
3: (3,1), (3,2), (3,4)
4: (4,1), (4,2), (4,3), (4,6)
5: (5,6), (5,7), (5,8)
6: (6,4), (6,5), (6,7)
7: (7,5), (7,6), (7,8)
8: (8,5), (8,7)

a) Draw the graph by displaying the edges on the diagram above.

b) Change the representation of the graph from adjacency lists to **adjacency matrix** and show the matrix.

c) Using the DFS algorithm in the Appendix, perform a depth-first search traversal on the given graph starting from node **1** and using the **adjacency lists** representation of the graph. The adjancency lists will influence the order in which the vertices are considered; for example, G.incidentEdges(1) will return the list : (1,2), (1,3), (1,4) so that the edges will be considered in this order.
List the vertices in the order they are visited, and list the edges in the order they are labelled by the algorithm, displaying their labels.

Vertices in order of visit:

Edges and labels in order of visit:

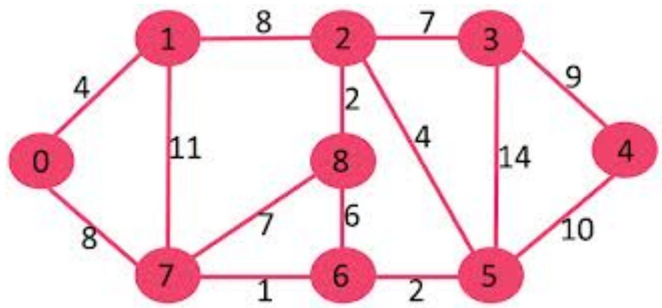
Please give the edges in the order they are labelled, display each edge in the direction of visit, and use the first letter of the label; for example - if a discovery edge was found coming from vertex b to a, the entry for this edge would be displayed "(b,a) D, "

d) Using the BFS algorithm in the Appendix, perform a breadth-first search traversal of the graph starting from node **1** and using the **adjacency lists** representation of the graph.
List the vertices in the order they are visited circling the groups of vertices that belong to each list L_0, L_1, L_2, etc. List the edges in the order they are labelled by the algorithm, displaying their labels.
Please, use a similar format as suggested in question 2c.

Vertices in order of visit:

Edges and labels in order of visit:

Question 3. **Shortest paths (8 points = 2+6)** Use Dijkstra's algorithm to obtain a tree of shortest paths for the graph below starting from vertex 0. Visit adjacent vertices of a given vertex in increasing order of label; for example, the adjacent vertices of vertex 3 will come in the following order: 2, 4, 5.



a) **[1 point]** Draw the shorted path edges using thick solid lines on the given graph.

b) **[3 points]** Fill the table below:

I. Vertices in the order they enter the cloud (tree of shortest paths),

II. Edges in the order they enter the tree of shortest paths (only tree edges. Solid lines),

III. Final array with distances **dist**, where **dist[v]** shows the distance between the origin (vertex 0) and vertex v.

Vertices (in order)

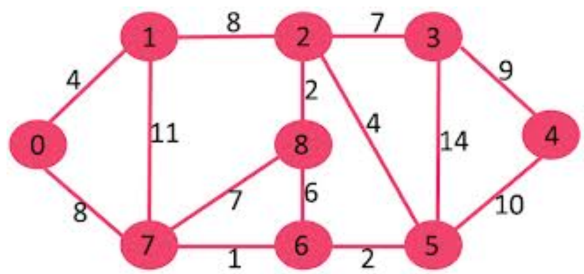
Edges (in order)

	Dist from A dist
0	0
1	
2	
3	
4	
5	
6	
7	
8	

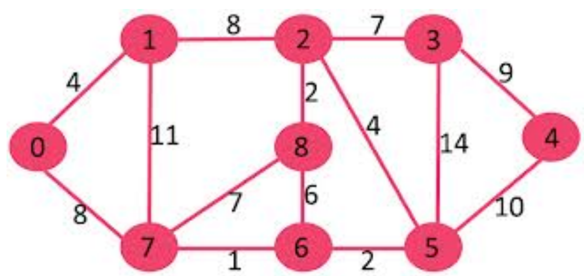
Question 4. **Minimum Spanning Tree (12 points = 1+1+8+1+1)**

Draw the Minimum Spanning Tree (MST) for the graph below using two algorithms. Indicate MST edges using thick solid lines on the given graphs. Select an adjacent vertex in alphabetical order.

a) [Prim-Jarnik's Algorithm starting from vertex 0:



b) Kruskal Algorithm



c) Fill the following table with the chosen edges in order of being chosen. Indicate each edge displaying the weight beside it, as in 01 (4), representing edge {0,1} with weight 4.

Chosen edges by the Prim-Jarnik's algorithm starting at vertex 0	Chosen edges by Kruskal's algorithm

d) What is the total weight of a MST in this graph?

e) How many minimum spanning trees are there in this graph?

Appendix

Depth-first Search algorithm (DFS)

```
Algorithm DFS(G)
      Input graph G
      Output labeling of the edges
of G as discovery edges and back
edges

for all  u ∈ G.vertices()
  setLabel(u, UNEXPLORED)
for all  e ∈ G.edges()
  setLabel(e, UNEXPLORED)
for all  v ∈ G.vertices()
  if  getLabel(v) = UNEXPLORED
      DFS(G, v)
```

```
Algorithm DFS(G, v) Input graph G and a
start vertex v of G
      Output labeling of the edges of G
in the connected component of v as
discovery edges and back edges
      setLabel(v, VISITED)
for all  e ∈ G.incidentEdges(v)
   if  getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
          setLabel(e, DISCOVERY)
          DFS(G, w)
      else
          setLabel(e, BACK)
```

Breadth-first Search algorithm (BFS)

```
Algorithm BFS(G)
      Input graph
      Output labeling of the edges
and partition of the vertices  of G
for all  u ∈ G.vertices()
  setLabel(u, UNEXPLORED)
for all  e ∈ G.edges()
  setLabel(e, UNEXPLORED)
for all  v ∈ G.vertices()
  if  getLabel(v) = UNEXPLORED
      BFS(G, v)
```

```
Algorithm BFS(G, s)
      L_0 ← new empty sequence
L_0.insertLast(s)
setLabel(s, VISITED)
i ← 0
while  ! L_i.isEmpty()
   L_i+1 ← new empty sequence
   for all  v ∈ L_i.elements()
      for all  e ∈ G.incidentEdges(v)
       if  getLabel(e) = UNEXPLORED
        w ← opposite(v,e)
        if  getLabel(w) = UNEXPLORED
          setLabel(e, DISCOVERY)
          setLabel(w, VISITED)
          L_i+1.insertLast(w)
        else
          setLabel(e, CROSS)
   i ← i +1
```