**Program 3: Dynamic Minimum Spanning Trees**

Due Dates: Part 1 &2 July 16, 2021

**Overview:**

The problem to solve is using a priority queue to compute a minimum spanning tree.

Given a fully connected undirected graph where each edge has a weight, find the set of edges with the least total sum of weights using Prim’s algorithm.

You would ask, why would I use a priority queue in Prim’s algorithm? The reason why people use it is because it significantly speeds up the runtime of the algorithm. It turns from O(V2 + E) to O(E\*log(V)).

Basically, at each step in Prim’s algorithm, you're looking for the minimum edge with one vertex in the partial minimum spanning tree, and one vertex not in the tree, and you're going to add that edge to the tree. How do you do that efficiently? If you have a way to efficiently order all the edges connected to a vertex in your partial spanning tree, you can simply iterate through them until you find an edge with an acceptable vertex.

Without such an ordered data structure, you'd have to iterate through all candidate edges each time to find the minimum, rather than being able to efficiently grab the minimum directly.

The input data describing the graph will be the nodes and associated list of edges. The program will need to covert that input into to an adjacency list: for every node in the, there will be list of the nodes it’s connected to and the weight.

adj[0] → (1, 1) (3, 3) vertex 0 is connected to vertex 1 (cost 1) and vertex 3 (cost 3)

adj[1] → (0, 1) (2, 6) (3, 5) (4, 1) vertex 1 is connected to vertex 0, 2, 3, and 4

. . .

First, the adjacency list could be represented as a list of edge nodes.

For the priority queue you are to write your own routine to handle this – you CANNOT use built-in library routines to implement the priority queue

Turn-in Template

* test plan version 2 will have additional test cases you need to add
* algorithms are logic as close as possible without coding and testing so do expect to have to change some aspects (hence update algorithm in part 2) as you code and test

Input Format

Input and output file names are to be entered by the user.

Line 1: Two space-separated integers: X , the number of nodes in the graph, and Y , the number of edges.

Lines 2 . . . Y+1 :

Line i contains three space-separated numbers describing an edge:

si and ti , the IDs of the two nodes involved, and wi , the weight of the edge.

Sample input (**input file name format is MST#.dat where this is file MST1.dat**)

6 9

0 1 1

1 3 5

3 0 3

3 4 1

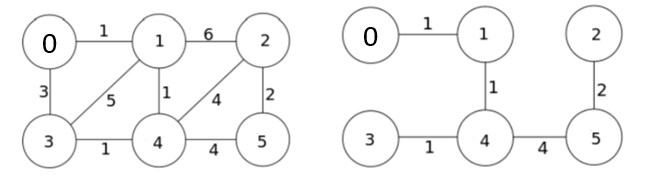
1 4 1

1 2 6

5 2 2

2 4 4

5 4 4



Graph MST

Line 1: 6 nodes (0 – 5) and 9 edges to follow line 2 - 10

Line 2: 0 1 1 describes edge between 0 and 1 of weight 1

Line 3: 1 3 5 describes edge between 1 and 3 of weight 5

…

The MST sum is 1+1+1+4+2 = 9

Output to Screen and File (output file name format: **MST#.out**)

Echo print the input (format appropriately per display messages in program)

Print the adjacency list for the full graph

Print the edge list and adjacency list for the MST – see example below

- if number of nodes < 10 print each iteration of building the MST

Some Assumptions

Input format is correct

Input line 1 numbers are correct if not less than 0

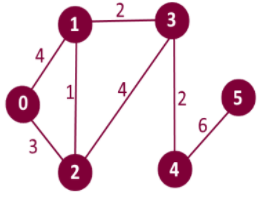
Example Exception Handling

File handling (file does not exist,empty) stop program

Error if node on edges do not exist error message to screen and file

- error in file – continue next input

Example output for first graph MST2.dat



6 vertices (numbered 0 – 5) and 7 edges

6 7

0 1 4

2 3 4

1 3 2

0 2 3

4 5 6

1 2 1

3 4 2

Full Graph – Adjacency List (note: order does not matter)

Adj[0] -> (1,4) (2, 3)

Adj[1] -> (0,4) (2,1) (3,2)

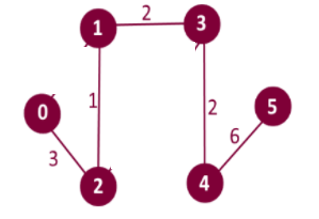
Adj[2] -> (0,3) (1,1) (3,4)

Adj[3] -> (1,2) (2,4) (4,2)

Adj[4] -> (3,2) (5,6)

Adj[5] -> (4,6)

MST



Minimum Spanning Tree

Edge: 1 - 2 weight: 1 resultSet[1] parent 2 weight 1 1 – 2 weight 1

Edge: 2 - 0 weight: 3 resultSet[2] parent 0 weight 3 2 – 0 weight 3

Edge: 3 - 1 weight: 2

Edge: 4 - 3 weight: 2 for (int i = 1; I < numberOfVertices; i++)

Edge: 5 - 4 weight: 6 {cout << “Edge: “ << I << “ – “ << resultSet[i].parent

Total cost of MST: 14 << “ weight: “ << resultSet[i].weight << endl;}

MST Graph – Adjacency List

Adj[0] -> (2,3)

Adj[1] -> (1,1) (3,2)

Adj[2] -> (0,3) (1,1)

Adj[3] -> (1,2) (4,2)

Adj[4] -> (3,2) (5,6)

Adj[5] -> (4,6)

REQUIRED FILES/TESTS:

**Input file does not exist or is empty – error message to screen and output file - stop program**

**You MUST submit three additional test files to the 4 files below WITH entries in the test plan for tests you created**

MST1.dat

6 9

0 1 1

1 3 5

3 0 3

3 4 1

1 4 1

1 2 6

5 2 2

2 4 4

5 4 4

MST2.dat (2 graphs in same file)

6 7

0 1 4

2 3 4

1 3 2

0 2 3

4 5 6

1 2 1

3 4 2

5 5

0 1 1

1 3 5

3 0 3

2 4 1

3 4 6

MST3.dat (listed error edges)

6 16

0 1 1

1 3 5

-3 1 4 invalid source vertex

3 0 3

3 4 1

6 1 6 invalid source vertex

1 4 1

1-3 4 invalid destination vertex

2 4 0 invalid weight

1 2 6

5 2 2

2 4 -3 invalid weight

2 4 4

5 14 5 invalid destination vertex

5 4 4

15 3 7  invalid source vertex

MST4.dat (spacing to show what would be considered separate graphs)

0 0

-1 3

0 1 1

1 2 1

2 3 4

6 -4

5 0

0 5

0 1 3

1 2 6

2 3 5

3 4 6

4 5 3

5 3

0 3 5

0 4 6

1 2 4