1. **Problem Statement**

Create an MST using Prim’s Algorithm given a connected graph providing the MST and the total cost.

1. **Requirements**
   1. **Assumptions**

Input file values will be integers

File FORMAT is correct

Number of vertices and number of edges in input file

Consider correct values if 0 or greater

Negative values are invalid

Negative value for edges – no edges will appear in the input file for this graph

File may contain multiple graphs

Undirected graph

Graph input will be a connected graph even after disgarding invalid edges

* 1. **Specifications**

Display message “Welcome to the MST Test Program” to user

Display message “Enter output file name: ” to user

Read and use the user entered output file name

If output file cannot be used

Display message “file <user output file name> cannot be opened – program terminated” to user

Display message “Welcome to the MST Test Program” to output file

Display message “Testing Default Scenario” to user and output file

Create an empty graph and test functionality – No MST

Display message “Testing File Data” to user and output file

Display message “Enter file name for graph data: ” to user

Read user entered input data file name

Display message “File name for graph data: <input file name>” to ouput file

Perform file validation

If cannot open

Display message “file <user input file name> cannot be opened or does not exist – program terminated”

If file exists but is empty

Display message “file <user input file name> contains no data – program terminated”

For each graph in the input file data

Create full graph

Number of vertices and number of edges is first line of each set of graph data

if number of vertices less than zero

display message “ERROR: number of vertices: <number of vertices> is less than zero” to user and output file

display message “Empty Graph Will Be Created” to user and output file

create empty graph

otherwise

if number of vertices is equal to 0

display message “Number of vertices: <number of vertices> is equal to zero” to user and output file

display message “Empty Graph Will Be Created” to user and output file

create empty graph

otherwise vertices value is greater than 0

display message “Number of vertices: <number of vertices> is valid” to user and output file

if number of edges is less than number of vertices - 1 (zero or less - input file will have NO edge data; greater than zero but less than number of vertices – 1 cannot be connected graph)

display message “ERROR: <number of edges> edges invalid to create connected graph” to user and output file

display message “Empty Graph Will Be Created” to user and output file

create empty graph

if number of edges is less than 0

program will treat as zero edges – file will not contain edges

otherwise

display “Graph with <number of vertices> and <number of edges> will be created” to user and output file

create graph with specified number of vertices

Display “Number of input edges to process is: <number of edges>” to user and output file

attempt to add all edges from the input file to the graph

if empty graph edges cannot be added

display message “Empty Graph – Cannot Add Edge: <source>, <destination>, <weight>” to user and output file

if invalid value for vertex (non-existent vertex – negative vertex value, 6 vertices in graph and vertex value is 10)

display message “Invalid Source or Destination Vertex – Cannot Add Edge: <source>, <destination>, <weight> - Edge request ignored” to user and output file

if invalid value for weight (weight must be greater than 0)

display message “Invalid Weight – Cannot Add Edge: <source>, <destination>, <weight> - Edge request ignored” to user and output file

otherwise edge can be added to graph

undirected graph so there are two edges added to graph adjacency list

display message “Edge Added: <source>, <destination>, <weight>” to user and output file

Print the full graph adjacency list

display message “Full Graph – Adjacency List” to user and output file

For each vertex display graph adjacency list to user and output file in format

Adj[vertex] -> (destination1, cost1) (destination2, cost2)

Create the MST

Start with vertex 0

add edges to partial MST until complete (Prim’s algorithm) using a priority queue

Print the MST

display message “Minimum Spanning Tree” to user and output file

if empty graph

display message “Empty Graph – No MST” to user and output file

otherwise

list all edges and weights of the MST

display message “Edge: <nbr> - < connected vertex> weight: <edge weight>” to user and output file

display message “Total cost of MST: <total MST Weight>” to user and output file

display message “MST Graph – Adjacency List” to user and output file

for each vertex display MST adjacency list to user and output file in format

Adj[vertex] -> (destination1, cost1) (destination2, cost2)

Display message “Thank you for running the MST Test Program written by <your name>!” to user and output file

1. **Decomposition Diagram** (Used to break program down into components visually. Can have as many components as needed. Defines functionality that will solve the problem – does NOT define a flow )

Main

* Input
  + User file name
    - File validation
  + File Data
    - File data edits
      * Format:
        + number of vertices, number of edges
        + source vertex, destination vertex, weight
* Process
  + Create graph
  + Create MST
* Output
  + Welcome message
  + Input error messages
  + Print full graph – adjacency list
  + Print MST – edge sequence and adjacency list
  + End message

1. **Test Strategy**

File Testing (exist, empty)

Valid data

Invalid data

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| File Testing | 1 | File does not exist |  |  |  |  |
| File Testing | 2 | File exists but empty |  |  |  |  |
| Valid data | 1 | Valid connected graph vertices and edges |  |  |  |  |
| Valid data | 2 | Empty graph |  |  |  |  |
| Invalid data | 1 | Invalid number of vertices |  |  |  |  |
| Invalid data | 2 | Invalid number of edges |  |  |  |  |
| Invalid data | 3 | Invalid edge source vertex |  |  |  |  |
| Invalid data | 4 | Invalid edge destination vertex |  |  |  |  |
| Invalid data | 5 | Invalid edge weight |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Initial Algorithm**

Data: Object Definitions

Struct pqData

Data:

integers: keyWeight, keyDestinationVertex, keySourceVertex

Class edge

Data:

integers: sourceVertex, destinationVertex, edgeWeight

link: nextEdge

Actions:

default constructor: initalize all data to -1

3 paramater constructor: integers source, destination, weight

Assign to appropriate class variables

Class resultSetClass

Data:

integers: parent, weight

Actions:

Default constructor: set all variables to value of -1

Class graph

Data:

integer: numberOfVertices

array of linked lists (must be able to hold all vertices): adjacencyList Graph, adjacencyListMST

Actions:

default constructor:

Set numberOfVertices to zero

display messsage “Default - Empty Graph Created”

1 parameter constructor: integer vertices

Set numberOfVertices to vertices

initialize adjacencyListGraph for each vertex as empty list; points to edge object

addEdge: 3 integer parameters source, destination, weight

if numberOfVertices equals zero

display message “Empty Graph – Cannot Add Edge: <source>,

<destination>, <weight>” to user and output file

otherwise

if either source or destination is less than zero or greater than numberOfVertices

display message “Invalid Source or Destination Vertex – Cannot Add Edge: <source>, <destination>, <weight> - Edge request ignored” to user and output file

if either weight is zero or less

display message “Invalid Weight – Cannot Add Edge: <source>, <destination>, <weight> - Edge request ignored” to user and output file

otherwise edge can be added to graph

create edge object – source, destination, weight

add to source vertex in adjacencyListGraph

display message “Edge Added: <source>, <destination>, <weight>” to user and output file

create edge object – destination, source, weight since undirected graph

add to destination vertex in adjacencyListGraph

display message “Edge Added: <destination>, <source>, <weight>” to user and output file

printGraph – no parameters

display message “Full Graph – Adjacency List” to user and output file

Loop through adjacencyListGraph

For each vertex display to user and output file adjacency list in format:

Adj[vertex] -> (destination1, cost1) (destination2, cost2)

primMST – no parameters

Create pqData extractedPQData

Create pqData intoPQData

Create boolean array mst of size numberOfVertices

Initialize mst values to false

Create resultSetClass array resultSet of size numberOfVertices ) default constructor

Initialize resultSet to point to resultSetClass instances

Create integer array weights of size numberOfVertices

Initialize weights values to maximum integer value (e.g. C++ INT\_MAX)

Vertex 0 is starting vertex – create non-edge priority queue entry to start MST

Set weights[0] to zero

Set pqData keyWeight to weights[0]

and keyDestinationVertex to 0

and keySourceVertex to 0

Add pqData to min-heap priority queue (you are to code your own priority queue – you cannot use library methods)

Set resultSet[0].parent to -1 (vertex 0 has no parent)

Loop while priority queue is not empty

Dequeue root from priority queue into extractedPQData – dequeueing minimum edge where keyDestinationVertex is vertex that will be added to the MST

Set mst[extractedPQData.keyDestinationVertex] to true

If extractedPQData.keyDestinationVertex and extractedPQData.keySourceVertes are both zero

skip over – vertex 0 start priority queue entry that is not an edge

otherwise

Add edges to adjacencyListMST for source and destination vertices (since undirected graph) for extractedPQData keySourceVertex and keyDestinationVertex and keyWeight values

Iterate through all the adjacent vertices to newly added vertex and update the weights as needed

For each edge in extractedPQData.keyDestinationVertex adjacency list

If mst[edge.destinationVertex] is equal to false (the destination vertex not in MST)

If weights[edge.destinationVertex] is greater than edge.edgeWeight

Assign edge.edgeWeight, edge.destinationVertex, and edge.sourceVertex to intoPQData keyWeight, keyDestinationVertex, and keySourceVertex

Add intoPQData to priority queue

Update resultSetClass

Set resultSet[edge.destinationVertex]. parent to extractedPQData.keyDestinationVertex

Set resultSet[edge.destinationVertex].weight to edge.edgeWeight

Set weights[edge.destinationVertex] to edge.edgeWeight

printMST – no parameters

Create integer totalMSTWeight, initialize to zero

Display message “Minimum Spanning Tree”

If numberOfVertices equals zero

Display message “Empty Graph – No MST”

Return from method

Loop through resultSet (nbr from 1 to number of vertices - 1)

Display message “Edge: <nbr> - <resultSet[nbr].parent> weight: <resultSet[nbr].weight”

Add resultSet[nbr].weight to totalMSTWeight

Display message “Total cost of MST: <totalMSTWeight>”

Display message “MST Graph – Adjacency List”

Loop through adjacencyListMST

For each vertex display adjacency list in format

Adj[vertex] -> (destination1, cost1) (destination2, cost2)

destructor: deallocate objects in adjacency list

Program: main

Main:

Display message “Welcome to the MST Test Program” to user

Display message “Enter output file name: ” to user

Read user entered output file name

Open output file

If output file cannot be opened

Display message “file <user output file name> cannot be opened – program terminated” to user

Terminate program

Display message “Output file: <user output file name>” to output file

Display message “Testing Default Scenario” to user and output file

Create default graph constructor instance – empty graph

Call method mstPrim – no MST created

Call method printMST – no MST created message

Display message “Testing File Data” to user and output file

Display message “Enter file name for graph data” to user

Read user entered input file name

Display message “File name for graph data: <input file name>” to ouput file

Open file

If file cannot be opened

Display message “File <user file name> cannot be opened or does not exist – program terminated” to user and output file

Terminate program

otherwise

If file opens but has no data in it

Display message “File <user file name> contains no data – program terminated” to user and output file

Terminate program

otherwise file has data to process

Loop until end of file – each loop instance is one graph

Number of vertices and number of edges is first line of each set of graph data

if number of vertices is less than zero

display message “ERROR: number of vertices: <vertices> is less than zero”

display message “Empty Graph Will Be Created”

create graph object with parameter of 0 vertices – empty graph

otherwise

if number of vertices is equal to zero

display message “Number of vertices: <vertices> is equal to zero”

display message “Empty Graph Will Be Created”

create graph object with parameter of 0 vertices – empty graph

otherwise vertices value is greater than zero

display message “Number of vertices: <vertices> is valid”

if number of edges is less than number of vertices - 1 (zero or less - input file will have NO edge data; greater than zero but less than number of vertices – 1 cannot be connected graph)

display message “ERROR: <number of edges> edges invalid to create connected graph” to user and output file

display message “Empty Graph Will Be Created” to user and output file

create graph object with parameter of 0 vertices – empty graph

if number of edges is less than 0

program will treat as zero edges – file will not contain edges

otherwise

display “Graph with <number of vertices> and <number of edges> will be created” to user and output file

create graph object with parameter of number of vertices

Display “Number of input edges to process is: <number of edges>” to user and output file

Loop for second data in the file (number of edges)

Read fileSource, fileDestination, fileWeight

Call addEdge in graph instance

Call printGraph

Call mstPrim

Call printMST

Deallocate graph object

Read from file to see if more graphs

End graph loop

Display message “Thank you for running the MST Test Program written by <your name>!” to user and output file

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| File Testing | 1 | File does not exist | File name that does not exist | “File <user file name> cannot be opened or does not exist – program terminated” |  |  |
| File Testing | 2 | File exists but empty | File name that exists but has no data | “File <user file name> contains no data – program terminated” |  |  |
| Valid data | 1 | Valid connected graph vertices and edges | File mst1.dat | MST with cost of 9 |  |  |
| Valid data | 2 | Empty graph – default constructor | Coded in program | “Empty Graph – No MST” |  |  |
| Valid data | 3 | Display messages to user | Coded in program | All messages verified on screen and in output file |  |  |
| Valid data | 4 | Print full graph | File mst2.dat | 2 graph adjacency lists verified |  |  |
| Valid data | 5 | Print MST | File mst2.dat | 2 MST edge lists and adjacency lists and total cost of MSTs verified |  |  |
| Invalid data | 1 | Invalid number of vertices | File mst4.dat | “Empty Graph – No MST” |  |  |
| Invalid data | 2 | Invalid number of edges | File mst4.dat | “Empty Graph – No MST” |  |  |
| Invalid data | 3 | Invalid edge source vertex | File mst3.dat | 3 error edges |  |  |
| Invalid data | 4 | Invalid edge destination vertex | File mst3.dat | 2 error edges |  |  |
| Invalid data | 5 | Invalid edge weight | File mst3.dat | 2 error edges |  |  |
| Invalid data | 6 | Try to add edges to empty graph | File mst4.dat  Graph 0 5 | Graph 0 5 edges cannot add edge error message |  |  |
| Invalid data | 7 | Not enough edges for connected graph | File mst.4  Graph 5 3 | “ERROR: 3 edges invalid to create connected graph” |  |  |

Part 1 ends here!!!!!!

1. **Code**

Copy and paste your code here. MAKE SURE TO COMMENT YOUR CODE!

A baseline for commenting is before any function add this:

//Description: What does the function do

//Pre-condition: What do input do you need for the function to work

//Post-condition: What is the end result of the function or what do you get out of the function

Also the beginning of your program should have these comments:

//Program Name:

//Programmer Name:

//Description:

//Date Created:

1. **Updated Algorithm**

Copy and paste Initial Algorithm and make any updates to reflect the changes you made in your code. HIGHLIGHT THE CHANGES YOU MAKE! Strike out deleted statements. Any statements that just have a wording change – make change and highlight (i.e. no need to strike out individual word changes). This is the FINAL documentation of your program and needs to match what code you created.

1. **Test Plan Version 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Copy and Paste from Version 2 | Un-highlight any test cases are from version 2 that were highlighted |  |  |  | What was the actual output from your code | If your actual output matches with expected output, write PASS otherwise write FAIL |
|  | Add any final test cases you could think of and HIGHLIGHT THEM |  |  |  |  | Any test cases that fail you must change your code to make the cases pass |

1. **Screenshots**

Screenshots of your testing goes here. YOU MUST HAVE A SCREENSHOT FOR EVERY TEST CASE (unless the output goes to a file in which case the screenshots are a sample of output and specifically any output that does not get sent to the output file – e.g. exceptions). A screenshot may picture multiple test cases. For each screen shot caption it with a list of the test cases are depicted in it.

1. **Error Log**

Any issues you had while testing your code are recorded in the error log as you perform testing of the “completed” code – that is, when you run through all of the test cases in the test plan.

|  |  |  |
| --- | --- | --- |
| Error Type | Cause of Error | Solution to Error |
| Log 2 types of errors:  Logic  Runtime | What specifically caused the error to occur | What did you do/change to fix the error |
|  |  |  |

Do not list any syntax errors or errors detected in unit testing as you build your program.

1. **Status**

What is the final status of your program? Does it fully work? Are there any test cases that fail and if so which ones? What needs to be done to correct the defects?