1. **Problem Statement**

The application must organize vertices and edges into a graph and a minimal spanning tree and allow the user to perform modification actions to such data.

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

* Each vertex and edge identifier will only consist of a maximum of 20 character.
* The input data from the file will always be in the proper format with a correct number of characters on each line
* Only the update directives defined in the specifications will be entered
* All edge weights entered will be positive floating-point values
  1. **Specifications**
* Must read initial vertices and edges from a file and create an initial minimum spanning tree.
* All duplicate vertices and edges should be ignored
* All errors should be printed to the screen
* All operations performed on the MST should be printed to the screen
* Prim’s Algorithm should be used to compute the initial MST
  + Use the first inputted vertex as the starting vertex
* User will choose to read update directives from a file or enter the directives manually
* MST update directives include
  + Print-graph
  + print-mst
  + path [vertex 1] [vertex 2]
  + insert-vertex [vertex]
  + insert-edge [vertex 1] [vertex 2] [weight]
  + decrease-weight [vertex 1] [vertex 2] [weight]
  + delete-vertex [vertex]
  + delete-edge [vertex 1] [vertex 2]
  + increase-weight [vertex 1] [ vertex 2] [weight]
  + quit
* Verify that all update directives are valid

1. **Decomposition Diagram**

Main

Print all operations that are performed to the screen.

Print all errors to the screen.

Structure the edges and vertices into a minimum spanning tree using a multiway tree

Process all update directives

Put all edges and vertices into an undirected weighted graph

File or user input containing MST update directives.

Initial file containing vertices and edges.

Input

Output

Process

1. **Test Strategy**

* File Handling
* Valid Inputs
* Invalid Inputs

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| File Handling | 1.1 | Enter a file name for the initial vertices and edges that exists |  |  |  |  |
| File Handling | 1.2 | Enter a file name for the initial edges and vertices that does not exist |  |  |  |  |
| Valid Inputs | 2.1 | The initial file contains a list of vertices and edges that are not duplicates |  |  |  |  |
| Valid Inputs | 2.2 | The initial file contains a list of vertices and edges with some duplicate vertices |  |  |  |  |
| Valid Inputs | 2.3 | The initial file contains a list of vertices and edges with some duplicate edges |  |  |  |  |
| Valid inputs | 2.4 | The initial file contains a list of vertices and edges with duplicates of each |  |  |  |  |
| Valid Inputs | 2.5 | The weights of edges in the initial input file are all equal |  |  |  |  |
| Valid Inputs | 2.6 | The weights of edges in the initial input file are all different |  |  |  |  |
| Valid Inputs | 2.7 | The initial vertices will all be unique, and the edges will create two unconnected graphs |  |  |  |  |
| Invalid Inputs | 3.1 | The initial file will contain no vertices and only edges |  |  |  |  |
| Invalid Inputs | 3.2 | The initial file will contain no edges and only vertices |  |  |  |  |
| Invalid Inputs | 3.3 | The initial file will contain no edges and no vertices |  |  |  |  |

1. **Initial Algorithm**

* Vertex class
  + Private attributes:
    - String for the vertex identifier
  + Public:
    - Get and set methods for the identifier
* Edge class
  + Private attributes:
    - Weight: float for the weight of the edge
    - Vertex1: string for the vertex identifier
    - Vertex2: string for the vertex identifier
  + Public:
    - Get and set methods for all private attributes
* Graph class
  + Private attributes:
    - adjMatrix: 2-dimensional vector array representing a matrix
    - Vertices: vector array containing all vertices
    - Edges: vector array containing all edges
  + Constructor:
    - Parameters:
      * Vector containing all vertices
    - Set the vertices vector equal to the vector passes as a parameter
  + addVertex function
    - Parameters:
      * Vertex: vertex object to add
    - add the vertex to the vertices vector
    - increase the size of the adjMatrix first dimension (columns)
    - increase the size of the adjMatrix second dimension (columns
  + addEdge function
    - Parameters:
      * Edge: edge object to add
    - If either of the two vertices of the edge don’t exist in the vertex vector
      * Print “[ERROR] One of the vertices in the edge do not exist, cannot create edge!”
    - Add the edge to the edges vector
    - Find the index of both vertices in the vertex vector (v1 and v2)
    - Set the float value of [v1][v2] in the adjMatrix equal to the weight of the edge
    - Set the float value of [v2][v1] in the adjMatrix equal to the weight of the edge
  + printGraph function:
    - if adjMatrix is not empty
      * for each row in the matrix
        + print each character in the row separated by 2 spaces
        + move to a new line
    - if the adjMatrix is empty
      * print “[ERROR] Graph is empty, cannot print!”
  + deleteVertex function:
    - Parameters:
      * Vertex identifier string
    - if adjMatrix is not empty
      * find the value of the index identifier in the vertex vector and store its index, then remove it
      * for each edge in the edges vector
        + if either vertex identifier of the edge is equal to the inputted vertex identifier, delete the edge
      * for each column in the adjMatrix
        + for each row in the adjMatrix

remove the item from the vector located at the index of the identifier deleted from the vertex vector

* + - * delete the column in the adjMatrix located at the index of the identifier deleted from the vertex vector
      * print “Deleted [identifier] from the graph”
    - if adjMatrix is empty
      * print “[ERROR] Graph is empty, there is no nodes to remove!”
  + deleteEdge function:
    - Parameters
      * Edge object
    - For each edge in the edges vector
      * If both vertices of the edge equal the vertices of the inputted edge object
        + Delete this edge from the edges vector
        + Find the index of each vertex in the vertex vector
        + Set the value in adjMatrix at [v1][v2] equal to 0
        + Set the value in adjMatrix at [v2][v1] equal to 0
        + Print “Deleted edge from [v1] [v2]”
        + Return from the function
    - Print “[ERROR] Edge does not exist”
  + increaseWeight function:
    - Parameters
      * Two string vertex identifiers
      * Weight float value to increase by
    - For each edge in the edges vector
      * If both vertices of the edge equal the inputted vertex identifiers
        + Increase the weight of that edge object by the amount inputted
        + Find the index of each vertex identifier in the vertex vector
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Print “Increase edge [v1] [v2] weight to [new edge weight]”
        + Return from the function
    - Print “[ERROR] Could not find the edge”
  + decreaseWeight function:
    - Parameters
      * Two string vertex identifiers
      * Weight float value to decrease by
    - For each edge in the edges vector
      * If both vertices of the edge equal the inputted vertex identifiers
        + Decrease the weight of that edge object by the amount inputted
        + Find the index of each vertex identifier in the vertex vector
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Print “Decreased edge [v1] [v2] weight to [new edge weight]”
        + Return from the function
    - Print “[ERROR] Could not find the edge”
* minheap class
  + Private Attributes:
    - Elements array containing the heap elements (which are edge objects)
    - maxSize integer for the size of the heap
    - size integer for the current size of the heap
  + Constructor
    - Parameters:
      * Integer for the size of the heap
    - Set maxSize equal to the input integer
    - Initialize the elements array with a size of maxSize
  + insertElement function:
    - parameters:
      * edge object to insert
    - if currSize equals the maxSize
      * print “[ERROR] Heap is at the max size”
      * return from the function
    - increase the currSize by 1
    - set the index equal to the current size -1
    - insert the edge object in the elements array at the index location
    - while index is not equal to 0 and weight of element at (2\*index)/2 is greater than weight element at the current index
      * swap the two elements in the elements array
      * set index equal to (2\*index)/2
  + removeMinElement function:
    - if the currSize is less than or equal to 0
      * return a null object
    - if the currSize is equal to 1
      * decrease currSize by 1
      * return the element at index 0 in the elements array
    - create a min edge object equal to the element at index 0 in the index array
    - set the element at index 0 equal to the last element in the array
    - decrease the currSize by 1
    - call the heapify function and pass 0 as the parameter
    - return the min edge object
  + heapify function:
    - Parameters:
      * Index integer to heapify
    - Create left index equal to (index\*2 + 1)
    - Create right index equal to (index\*2 + 2)
    - Create smallest index equal to input index
    - If smallest is less than currSize and element at left index is less than element at input index
      * Set smallest index equal to left index
    - If right index is less than cirrSize and element at right index is less than element at smallest index
      * Set smallest index equal to right index
    - If smallest does not equal input index
      * Swap elements in array at locations input index and smallest index
      * Call heapify function recursively passing smallest as the input parameter
* MST class
  + Inherits from Graph class
  + Private Attributes:
    - Tree node
      * Vertex object
      * Pointer to child node
      * Pointer to sibling node
      * Pointer to parent node
      * Weight to parent node
    - Root node for the mst
    - vector for visited vertices of type vertex object
  + gernerateMST function
    - if the adjMatrix is empty
      * print “[ERROR] Cannot create MST, the adjacency matrix is empty”
      * return from the function’
    - integer for maxVertices equal to size of vertex vector
    - integer for curIndex equal to 0
    - minheap object initialized with a size of maxVertices
    - while visited vector size is less than or equal to size of maxVertices
      * if root is null
        + add vertex at current index visited vector
        + create a new tree node with the vertex object and set root equal to this node
        + integer x for index equal to 0
        + while x is less than maxVertices

if value of adjMatrix [curIndex][x] is greater than 0

create an edge object

vertex1 equal to identifier of curIndex

vertex2 equal to identifier of x

weight equal to value of adjMatrix [curIndex][x]

insert the edge object into the minheap

increase x by 1

* + - * else if minheap is not empty
        + remove the min edge from the min heap
        + add the identifier of vertex2 of the edge object to the visited vector
        + set curIndex equal to index of vertex2 in the vertices vector
        + integer x equal to 0
        + while x is less than maxVertices

if value of adjMatrix [curIndex][x] is greater than 0

create an edge object

vertex1 equal to identifier of curIndex

vertex2 equal to identifier of x

weight equal to value of adjMatrix [curIndex][x]

insert the edge object into the minheap

increase x by 1

* + - * + create a new tree node

set the vertex object equal to the object that corresponds to the vertex 2 identifier

* + - * + create a parent node object equal to the return of findNode

pass the root node and the vertex1 identifier as parameters

* + - * + if parent has no child

set child of parent equal to the new node

set the parent of the new node equal to parent

set the parent weight of the new node equal to the weight of the edge

* + - * + else

create new node for current equal to parent child

while current has sibling

set current equal to current’s sibling

set current sibling equal to new node

set new node parent equal to parent

set the parent weight of the new node equal to the weight of the edge

* + - * else
        + print “[ERROR] The graph is a disjointed graph, cannot create a complete minimum spanning tree using all vertices. Only the vertices with connections to the first vertex are used in the MST”
        + return from the function
  + printMST function
    - Parameters:
      * Root node
      * Level integer
    - Print “.” Times the size of level
    - Print the root node identifier
    - If root has a child
      * Increase level by 1
      * Call printMST
        + Pass the root’s child as a parameter and level
    - If root has a sibling
      * Call printMST
        + Pass the root’s sibling as a parameter and the level
  + findNode function
    - parameters:
      * root tree node
      * string identifier of the vertex to search for
    - node for found
    - if root node identifier equals searched identifier
      * return root
    - if root has child
      * set found equal to return of findNode
        + pass the child node, the identifier to search for as parameters
      * if found is not null
        + return found
    - if root has sibling
      * set found equal to return of findNode
        + pass the child node, the identifier to search for as parameters
      * if the found is not null
        + return found
    - return null
  + findPath function
    - Parameters:
      * Vertex1 identifier
      * Vertex2 identifier
    - V1 equal to the index of vertex1 in the vertices vector
    - V2 equal to the index of vertex2 in the vertices vector
    - If v1 and v2 are equal
      * Print “Both vertices are equal to each other
      * Return 0
    - Float Vector for distances
    - Boolean Vector for included in the shortest path
    - Initialize distance vector to infinity
    - Initialize bool vector to false
    - Set the distance at index v1 equal to 0
    - For each vertex in vertices vector
      * Set int x equal to return of minDist function
        + Pass distance vector and bool vector as parameters
      * Set bool vector of index x equal tot rue
      * Integer I equal to 0
      * While I is less than vertex vector size
        + If the bool at I index is false and the weight at adjMatrix [x][i] is not 0 and distance at x is not infinity and distance at index x + weight at adjMatrix[x][i] is less than distance at i

Set distance at index I equal to distance at x + weight at adjMatrix [x][i]

* + - If distance at index v2 is greater than 0
      * Return distance at index v2
    - Else
      * Print “There is no possible path between the two vertices, they may not exist in the same tree”
      * Return 0
  + insertVertex function (override)
    - accepts the same parameters and performs all actions of insertVertex from the graph class
    - call the generateMST function
  + insertEdge function (override)
    - accepts the same parameters and performs all actions of insertEdge from the graph class
    - call the generateMST function
  + decreaseWeight function (override)
    - accepts the same parameters and performs all actions of decreaseWeight from the graph class
    - call the generateMST function
  + increaseWeight function (override)
    - accepts the same parameters and performs all actions of increaseWeight from the graph class
    - call the generateMST function
  + deleteVertex function (override)
    - accepts the same parameters and performs all actions of deleteVertex from the graph class
    - call the generate MST function
  + deleteEdge function (override)
    - accepts the same parameters and performs all actions of deleteEdge from the graph class
    - call the generateMST function
* Main function:
  + String for the initial input vertex file directory
  + String for the initial input edge file directory
  + Ask the user to enter the location of the initial input files
  + Create an MST object
  + Integer for the number of vertices
  + Integer for the number of edges
  + Check if the files exist
    - If either file does not exist
      * Print “[ERROR] File does not exist {file name}”
      * Return from the function
  + Open the vertex file
    - Set the value of number of vertices equal to the value on the first line in the file
    - For every other line
      * Call insertVertex on the mst object
        + Pass the vertex identifier from the current line as a parameter
  + Open the edge file
    - Set the value of number of edges equal to the value on the first line in the file
    - For every other line
      * Call insertEdge on the mst object
        + Pass the two vertex identifiers and the weight from the line as a parameter
  + Ask the user to enter directives by a file or by keyboard input
  + If by file
    - Ask the user to enter the directory of the file containing the directives
    - Check if the file exists
      * If it does not exist
        + Print “[ERROR] file does not exist {file name}”
        + Return from the function
      * Open the file
        + For each line in the file

Split the strings by spaces into an input vector

If vector [0] equals “Print-Graph”

Call printGraph function on the mst object

If vector [0] equals “print-mst”

Call printMST function on the mst object

If vector [0] equals “path”

Call findPath function on the mst Object

Pass vector [1] and vector [2] as parameters

If vector [0] equals “insert-vertex”

Call insertVertex function on the mst object

Pass vector [1] as a parameter

If vector [0] equals “insert-edge”

Call insertEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

If vector [0] equals “decrease-weight”

Call decreaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

If vector [0] equals “increase-weight”

Call increaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

If vector [0] equals “delete-edge”

Call deleteEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) as parameters

If vector [0] equals “delete-vertex”

Call deleteVertex function on the mst object

Pass vector [1] as a parameter

If vector [0] equals “quit”

Return from the function (exits the program)

* + If by keyboard
    - While true
      * Ask the user to enter a directive
      * Split the entered line into an input vector separated by spaces
      * If vector [0] equals “Print-Graph”
        + Call printGraph function on the mst object
      * If vector [0] equals “print-mst”
        + Call printMST function on the mst object
      * If vector [0] equals “path”
        + Call findPath function on the mst Object

Pass vector [1] and vector [2] as parameters

* + - * If vector [0] equals “insert-vertex”
        + Call insertVertex function on the mst object

Pass vector [1] as a parameter

* + - * If vector [0] equals “insert-edge”
        + Call insertEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

* + - * If vector [0] equals “decrease-weight”
        + Call decreaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

* + - * If vector [0] equals “increase-weight”
        + Call increaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

* + - * If vector [0] equals “delete-edge”
        + Call deleteEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) as parameters

* + - * If vector [0] equals “delete-vertex”
        + Call deleteVertex function on the mst object

Pass vector [1] as a parameter

* + - * If vector [0] equals “quit”
        + Return from the function (exits the program)

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| File Handling | 1.1 | Enter a file name for the initial vertices and edges that exists | “valid.txt” | Opens file properly |  |  |
| File Handling | 1.2 | Enter a file name for the initial edges and vertices that does not exist | “Invalid.txt” | “[ERROR] File invalid.txt does not exist |  |  |
| File Handling | 1.3 | Enter a file name for the directives that does not exist | “dddirectives.txt” | “[ERROR] File dddirectives.txt does not exist |  |  |
| File Handling | 1.4 | Enter a file name for the directive that exists but is empty | “directivesE.txt” | “[ERROR] File directivesE.txt is empty” |  |  |
| File Handling | 1.5 | Enter a file name for the directives that exists and has data | “directives.txt” | Opens the directives file properly |  |  |
| Valid Inputs | 2.1 | The initial file contains a list of vertices and edges that are not duplicates | initialV\_1.txt  intialE\_1.txt | Opens and adds the vertices to the mst/graph |  |  |
| Valid Inputs | 2.2 | The initial file contains a list of vertices and edges with some duplicate vertices | initialV\_2.txt  initialE\_1.txt | Opens and adds the vertices and edges to the graph/mst and ignores the duplicate vertices |  |  |
| Valid Inputs | 2.3 | The initial file contains a list of vertices and edges with some duplicate edges | InitialV\_1.txt  InitialE\_2.txt | Opens and adds the vertices and edges to the graph/mst and ignores the duplicate edges |  |  |
| Valid inputs | 2.4 | The initial file contains a list of vertices and edges with duplicates of each | initialV\_2.txt  initialE\_2.txt | Opens and adds the vertices and edges to the graph/mst and ignores all duplicate values |  |  |
| Valid Inputs | 2.5 | The weights of edges in the initial input file are all equal | initialV\_2.txt  initialE\_3.txt | Opens and adds the vertices and edges to the graph/mst will all weights of the edges being equal |  |  |
| Valid Inputs | 2.6 | The weights of edges in the initial input file are all different | initialV\_2.txt  initialE\_2.txt | Opens and adds the vertices and edges to the graph/mst |  |  |
| Valid Inputs | 2.7 | The initial vertices will all be unique, and the edges will create two unconnected graphs | initialV\_2.txt  initialE\_4.txt | Opens and adds vertices and edges to the graph/mst causing the mst to be split into two trees |  |  |
| Valid Inputs | 2.8 | Use the print-graph directive on a valid graph | After test 2.6  “print-graph” | Prints the graph to the screen |  |  |
| Valid Inputs | 2.9 | Use the print-mst directive on an mst that is valid and not disjointed | After test 2.6  “print-mst” | Prints the mst on the screen in prefix order |  |  |
| Valid Inputs | 2.11 | Use the print-mst directive on a mst that is valid and is disjointed | After test 2.7  “print-mst” | Prints two separate mst’s to the screen in prefix order |  |  |
| Valid Inputs | 2.12 | Attempt to find the path of two edges that exist | After test 2.6  “path v14 v264” | Print the total weight between the two nodes |  |  |
| Valid Inputs | 2.13 | Use the insert-vertex directive to insert a vertex that is not a duplicate of any vertices | After test 2.6  “insert-vertex v98” | Inserts the vertex into the mst |  |  |
| Valid Inputs | 2.14 | Use insert-edge directive to insert an edge that is not a duplicate of any edges and the vertices exist | After test 2.6  “insert-edge v78 v19 4.1” | Inserts the edge into the mst and updates the mst structure |  |  |
| Valid Inputs | 2.15 | Use decrease-weight directive decrease the weight of an edge that exists to value above 0 | After test 2.14  “decrease-weight v78 v19 2.v0” | Decrease the weight of the edge and updates the mst structure |  |  |
| Valid Inputs | 2.16 | Use increase-weight directive to increase the weight of an edge that exists | After test 2.14  “decrease-weight v78 v19 9.1” | Increases the weight of the edge and updates the mst structure |  |  |
| Valid Inputs | 2.17 | Use delete-vertex directive to delete a vertex that exists and has no edges | After test 2.6 “delete-vertex v1” | Deletes the vertex and updates the mst structure |  |  |
| Valid Inputs | 2.18 | Use delete-vertex directive to delete a vertex that exists and has an edge | After test 2.6  “delete-vertex v78” | Deletes the vertex and updates the mst structure |  |  |
| Valid Inputs | 2.19 | Use delete-vertex directive to delete a vertex that exists and has multiple edges | After test 2.6  “delete-vertex v19” | Deletes the vertex and updates the mst structure |  |  |
| Valid Inputs | 2.20 | Use delete-edge directive to delete an edge that exists and will not create a disconnected graph | After test 2.6  “delete-edge v19 v2” | Deletes the edge and updates the mst structure |  |  |
| Valid Inputs | 2.21 | Use delete-edge directive to delete an edge that exists and will cause a disconnected graph | After test 2.6  “delete-edge v19 v10” | Deletes the edge and updates the mst structure |  |  |
| ~~Invalid Inputs~~ | ~~3.1~~ | ~~The initial file will contain no vertices and only edges~~ |  |  |  |  |
| ~~Invalid Inputs~~ | ~~3.2~~ | ~~The initial file will contain no edges and only vertices~~ |  |  |  |  |
| ~~Invalid Inputs~~ | ~~3.3~~ | ~~The initial file will contain no edges and no vertices~~ |  |  |  |  |
| Invalid Inputs | 3.4 | Use the path directive of 2 vertices where one does not exist | After test 2.6  “path v19 v1” | “[ERROR] vertex v1 does not exist” |  |  |
| Invalid Inputs | 3.5 | Use the path directive of 2 vertices where neither exist | After test 2.6  “path v2  v1” | “[ERROR] vertex v1 does not exist”  “[ERROR] vertex v2 does not exist” |  |  |
| Invalid Inputs | 3.6 | Use the insert-vertex directive and insert a vertex that is a duplicate | After test 2.6  “insert-vertex v19” | “[ERROR] vertex is a duplicate” |  |  |
| Invalid Inputs | 3.7 | Use the insert-edge directive to insert an edge where only one vertex exists | After test 2.6 “insert-edge v19 v1 10” | [ERROR] vertex v1 does not exist |  |  |
| Invalid Inputs | 3.8 | Use the insert-edge directive to insert an edge where neither vertex exists | After test 2.6 “insert-edge v2 v1 10” | [ERROR] vertex v1 does not exist  [ERROR] vertex v2 does not exist |  |  |
| Invalid Inputs | 3.9 | Use the insert-edge directive to insert an edge with a negative weight | After test 2.6  “insert-edge v78 v19  -4.1” | “[ERROR] Weight must be greater than 0” |  |  |
| Invalid Inputs | 3.10 | Use the insert-edge directive to insert an edge with a weight equal to 0 | After test 2.6  “insert-edge v78 v19 0” | “[ERROR] Weight must be greater than 0” |  |  |
| Invalid Inputs | 3.11 | Use the decrease-weight directive to decrease the weight of an edge to a value less than or equal to 0 | After test 2.14  “decrease-weight v78 v19 20” | “[ERROR] Weight must be greater than 0” |  |  |
| Invalid Inputs | 3.12 | Use the increase weight directive to increase the weight by a negative value so that the weight is less than or equal to 0 | After test 2.14  “decrease-weight v78 v19 4.1” | “[ERROR] Weight must be greater than 0” |  |  |
| Invalid Inputs | 3.13 | Use the delete-vertex directive to delete a vertex that does not exist | After test 2.6 “delete-vertex v1” | “[ERROR] vertex does not exist” |  |  |
| Invalid Inputs | 3.14 | Use the delete-edge directive to delete an edge where only one vertex exists | After test 2.14  “delete-edge v78 v1” | “[ERROR] vertex v1 does not exist” |  |  |
| Invalid Inputs | 3.15 | Use the delete-edge directive to delete an edge where neither vertex exists | After test 2.14  “delete-edge v2 v1” | “[ERROR] vertex v2 does not exist”  “[ERROR] vertex v1 does not exist” |  |  |
| Invalid Inputs | 3.16 | Use the delete-edge directive to delete an edge between two valid vertices but no edge exists between them | After test 2.6  “delete-edge v3 v5” | “[ERROR] No edge exists between the two vertices” |  |  |
| Invalid Inputs | 3.17 | Enter a directive that is not a valid operation | “print x y” | “[ERROR] unrecognized directive” |  |  |
| Invalid Inputs | 3.18 | Using the path directive, enter the two vertices in an invalid format | “path v12v133” | “[ERROR] invalid format for directive path” |  |  |
| Invalid Inputs | 3.19 | Using the Insert-Vertex directive, enter an invalid an extra value after the vertex identifier | “path v12 v13 v18 | “[ERROR] invalid format for directive path” |  |  |
| Invalid Inputs | 3.20 | Using the insert-vertex directive, enter no vertex identifier | “insert-vertex” | “[ERROR] invalid format for directive insert-vertex” |  |  |
| Invalid Inputs | 3.21 | Using decrease-weight directive, enter a string for the weight value | “decrease-weight v13 v14 sda” | “[ERROR] Invalid format for directive decrease-weight” |  |  |
| Invalid Inputs | 3.22 | Using decrease-weight directive, enter no value for the weight | “decrease-weight v13 v14” | “[ERROR] Invalid format for directive decrease-weight” |  |  |
| Invalid Inputs | 3.23 | Using the increase-weight directive, enter no value for the weight | “increase-weight v13 v14” | “[ERROR] Invalid format for directive increase-weight” |  |  |
| Invalid  Inputs | 3.24 | Using the increase-weight directive,  Enter a string value for the weight | “increase-weight v13 v14 sda” | “[ERROR] Invalid format for directive increase-weight” |  |  |
| Invalid Inputs | 3.25 | Using the delete-vertex directive enter no value for the vertex | “delete-vertex” | “[ERROR] invalid format for directive delete-vertex” |  |  |
| Invalid Inputs | 3.26 | Using the delete-vertex directive enter an extra vertex | “delete-vertex v12 v23” | “[ERROR] invalid format for directive delete-vertex” |  |  |
| Invalid Inputs | 3.27 | Using the delete-edge directive enter no edge value | “delete-edge” | “[ERROR] invalid format for directive delete-edge” |  |  |
| Invalid Inputs | 3.28 | Using the delete-edge directive enter an extra edge value | “delete-edge v12 v14 v298” | “[ERROR] invalid format for directive delete-edge” |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Test files:

initialV\_1.txt Vertices that are not duplicates

initialV\_2.txt Vertices with some duplicates

initialE\_1.txt Edges with no duplicates

initialE\_2.txt Edges with some duplicates

initialE\_3.txt Edges with all the same weights

initialE\_4.txt Edges that will create a disconnected graph/mst

1. **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**

* Vertex class
  + Private attributes:
    - String for the vertex identifier
  + Public:
    - Get and set methods for the identifier
* Edge class
  + Private attributes:
    - Weight: float for the weight of the edge
    - Vertex1: string for the vertex identifier
    - Vertex2: string for the vertex identifier
  + Public:
    - Get and set methods for all private attributes
* Graph class
  + Private attributes:
    - adjMatrix: 2-dimensional vector array representing a matrix
    - Vertices: vector array containing all vertices
    - Edges: vector array containing all edges
  + Constructor:
    - Parameters:
      * Vector containing all vertices
    - Set the vertices vector equal to the vector passes as a parameter
  + addVertex function
    - Parameters:
      * Vertex: vertex object to add
    - add the vertex to the vertices vector
    - increase the size of the adjMatrix first dimension (columns)
    - increase the size of the adjMatrix second dimension (rows)
  + addEdge function
    - Parameters:
      * Edge: edge object to add
    - If either of the two vertices of the edge don’t exist in the vertex vector
      * Print “[ERROR] One of the vertices in the edge do not exist, cannot create edge!”
    - Add the edge to the edges vector
    - Find the index of both vertices in the vertex vector (v1 and v2)
    - Set the float value of [v1][v2] in the adjMatrix equal to the weight of the edge
    - Set the float value of [v2][v1] in the adjMatrix equal to the weight of the edge
  + printGraph function:
    - if adjMatrix is not empty
      * for each row in the matrix
        + print each character in the row separated by 2 spaces
        + move to a new line
    - if the adjMatrix is empty
      * print “[ERROR] Graph is empty, cannot print!”
  + deleteVertex function:
    - Parameters:
      * Vertex identifier string
    - if adjMatrix is not empty
      * find the value of the index identifier in the vertex vector and store its index, then remove it
      * for each edge in the edges vector
        + if either vertex identifier of the edge is equal to the inputted vertex identifier, delete the edge
      * for each column in the adjMatrix
        + for each row in the adjMatrix

remove the item from the vector located at the index of the identifier deleted from the vertex vector

* + - * delete the column in the adjMatrix located at the index of the identifier deleted from the vertex vector
      * print “Deleted [identifier] from the graph”
    - if adjMatrix is empty
      * print “[ERROR] Graph is empty, there is no nodes to remove!”
  + deleteEdge function:
    - Parameters
      * Edge object
    - For each edge in the edges vector
      * If both vertices of the edge equal the vertices of the inputted edge object
        + Delete this edge from the edges vector
        + Find the index of each vertex in the vertex vector
        + Set the value in adjMatrix at [v1][v2] equal to 0
        + Set the value in adjMatrix at [v2][v1] equal to 0
        + Print “Deleted edge from [v1] [v2]”
        + Return from the function
    - Print “[ERROR] Edge does not exist”
  + increaseWeight function:
    - Parameters
      * Two string vertex identifiers
      * Weight float value to increase by
    - For each edge in the edges vector
      * If both vertices of the edge equal the inputted vertex identifiers
        + Increase the weight of that edge object by the amount inputted
        + Find the index of each vertex identifier in the vertex vector
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Print “Increase edge [v1] [v2] weight to [new edge weight]”
        + Return from the function
    - Print “[ERROR] Could not find the edge”
  + decreaseWeight function:
    - Parameters
      * Two string vertex identifiers
      * Weight float value to decrease by
    - For each edge in the edges vector
      * If both vertices of the edge equal the inputted vertex identifiers
        + Decrease the weight of that edge object by the amount inputted
        + Find the index of each vertex identifier in the vertex vector
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Set the value in adjMatrix at [v1][v2] equal to the modified weight of the edge
        + Print “Decreased edge [v1] [v2] weight to [new edge weight]”
        + Return from the function
    - Print “[ERROR] Could not find the edge”
* minheap class
  + Private Attributes:
    - Elements array containing the heap elements (which are edge objects)
    - maxSize integer for the size of the heap
    - size integer for the current size of the heap
  + Constructor
    - Parameters:
      * Integer for the size of the heap
    - Set maxSize equal to the input integer
    - Initialize the elements array with a size of maxSize
  + insertElement function:
    - parameters:
      * edge object to insert
    - if currSize equals the maxSize
      * print “[ERROR] Heap is at the max size”
      * return from the function
    - increase the currSize by 1
    - set the index equal to the current size -1
    - insert the edge object in the elements array at the index location
    - while index is not equal to 0 and weight of element at (2\*index)/2 is greater than weight element at the current index
      * swap the two elements in the elements array
      * set index equal to (2\*index)/2
  + removeMinElement function:
    - if the currSize is less than or equal to 0
      * return a null object
    - if the currSize is equal to 1
      * decrease currSize by 1
      * return the element at index 0 in the elements array
    - create a min edge object equal to the element at index 0 in the index array
    - set the element at index 0 equal to the last element in the array
    - decrease the currSize by 1
    - call the heapify function and pass 0 as the parameter
    - return the min edge object
  + heapify function:
    - Parameters:
      * Index integer to heapify
    - Create left index equal to (index\*2 + 1)
    - Create right index equal to (index\*2 + 2)
    - Create smallest index equal to input index
    - If smallest is less than currSize and element at left index is less than element at input index
      * Set smallest index equal to left index
    - If right index is less than cirrSize and element at right index is less than element at smallest index
      * Set smallest index equal to right index
    - If smallest does not equal input index
      * Swap elements in array at locations input index and smallest index
      * Call heapify function recursively passing smallest as the input parameter
* MST class
  + Inherits from Graph class
  + Private Attributes:
    - Tree node
      * Vertex object
      * Pointer to child node
      * Pointer to sibling node
      * Pointer to parent node
      * Weight to parent node
    - Root node for the mst
    - vector for visited vertices of type vertex object
  + gernerateMST function
    - if the adjMatrix is empty
      * print “[ERROR] Cannot create MST, the adjacency matrix is empty”
      * return from the function’
    - integer for maxVertices equal to size of vertex vector
    - integer for curIndex equal to 0
    - minheap object initialized with a size of maxVertices
    - while visited vector size is less than or equal to size of maxVertices
      * if root is null
        + add vertex at current index to the visited vector
        + create a new tree node with the vertex object and set root equal to this node
        + integer x for index equal to 0
        + while x is less than maxVertices

if value of adjMatrix [curIndex][x] is greater than 0

create an edge object

vertex1 equal to identifier of curIndex

vertex2 equal to identifier of x

weight equal to value of adjMatrix [curIndex][x]

insert the edge object into the minheap

increase x by 1

* + - * else if minheap is not empty
        + remove the min edge from the min heap
        + add the identifier of vertex2 of the edge object to the visited vector
        + set curIndex equal to index of vertex2 in the vertices vector
        + integer x equal to 0
        + while x is less than maxVertices

if value of adjMatrix [curIndex][x] is greater than 0

create an edge object

vertex1 equal to identifier of curIndex

vertex2 equal to identifier of x

weight equal to value of adjMatrix [curIndex][x]

insert the edge object into the minheap

increase x by 1

* + - * + create a new tree node

set the vertex object equal to the object that corresponds to the vertex 2 identifier

* + - * + create a parent node object equal to the return of findNode

pass the root node and the vertex1 identifier as parameters

* + - * + if parent has no child

set child of parent equal to the new node

set the parent of the new node equal to parent

set the parent weight of the new node equal to the weight of the edge

* + - * + else

create new node for current equal to parent child

while current has sibling

set current equal to current’s sibling

set current sibling equal to new node

set new node parent equal to parent

set the parent weight of the new node equal to the weight of the edge

* + - * else
        + print “[ERROR] The graph is a disjointed graph, cannot create a complete minimum spanning tree using all vertices. Only the vertices with connections to the first vertex are used in the MST”
        + return from the function
  + printMST function
    - Parameters:
      * Root node
      * Level integer
    - Print “.” Times the size of level
    - Print the root node identifier
    - If root has a child
      * Increase level by 1
      * Call printMST
        + Pass the root’s child as a parameter and level
    - If root has a sibling
      * Call printMST
        + Pass the root’s sibling as a parameter and the level
  + findNode function
    - parameters:
      * root tree node
      * string identifier of the vertex to search for
    - node for found
    - if root node identifier equals searched identifier
      * return root
    - if root has child
      * set found equal to return of findNode
        + pass the child node, the identifier to search for as parameters
      * if found is not null
        + return found
    - if root has sibling
      * set found equal to return of findNode
        + pass the sibling node, the identifier to search for as parameters
      * if the found is not null
        + return found
    - return null
  + minDist function
    - Paramters:
      * Float vector for distances
      * Bool vector for included
    - Int min initialized to infinity
    - Int minindex initialized to infinity
    - For each value in the vertices vector
      * If the value is not included and distances of that index is less than min
        + Set min equal to the distance at the index
        + Set minindex equal to current index
    - If min is equal to infinity
      * Return infinity
    - Else return minindex
  + findPath function
    - Parameters:
      * Vertex1 identifier
      * Vertex2 identifier
    - V1 equal to the index of vertex1 in the vertices vector
    - V2 equal to the index of vertex2 in the vertices vector
    - If v1 and v2 are equal
      * Print “Both vertices are equal to each other
      * Return 0
    - Float Vector for distances
    - Boolean Vector for included in the shortest path
    - Initialize distance vector to infinity
    - Initialize bool vector to false
    - Set the distance at index v1 equal to 0
    - For each vertex in vertices vector
      * Set int x equal to return of minDist function
        + Pass distance vector and bool vector as parameters
      * Set bool vector of index x equal to true
      * Integer I equal to 0
      * While I is less than vertex vector size
        + If the bool at I index is false and the weight at adjMatrix [x][i] is not 0 and distance at x is not infinity and distance at index x + weight at adjMatrix[x][i] is less than distance at i

Set distance at index I equal to distance at x + weight at adjMatrix [x][i]

* + - If distance at index v2 is greater than 0
      * Return distance at index v2
    - Else
      * Print “There is no possible path between the two vertices, they may not exist in the same tree”
      * Return 0
  + addVertex function (override)
    - accepts the same parameters and performs all actions of insertVertex from the graph class
    - call the generateMST function
  + addEdge function (override)
    - accepts the same parameters and performs all actions of insertEdge from the graph class
    - call the generateMST function
  + decreaseWeight function (override)
    - accepts the same parameters and performs all actions of decreaseWeight from the graph class
    - call the generateMST function
  + increaseWeight function (override)
    - accepts the same parameters and performs all actions of increaseWeight from the graph class
    - call the generateMST function
  + deleteVertex function (override)
    - accepts the same parameters and performs all actions of deleteVertex from the graph class
    - call the generate MST function
  + deleteEdge function (override)
    - accepts the same parameters and performs all actions of deleteEdge from the graph class
    - call the generateMST function
* Main function:
  + String for the initial input vertex file directory
  + String for the initial input edge file directory
  + Ask the user to enter the location of the initial input files
  + Create an MST object
  + Integer for the number of vertices
  + Integer for the number of edges
  + Check if the files exist
    - If either file does not exist
      * Print “[ERROR] File does not exist {file name}”
      * Return from the function
  + Open the vertex file
    - Set the value of number of vertices equal to the value on the first line in the file
    - For every ~~other~~ line
      * Call insertVertex on the mst object
        + Pass the vertex identifier from the current line as a parameter
  + Open the edge file
    - Set the value of number of edges equal to the value on the first line in the file
    - For every other line
      * Call insertEdge on the mst object
        + Pass the two vertex identifiers and the weight from the line as a parameter
  + Ask the user to enter directives by a file or by keyboard input
  + If by file
    - Ask the user to enter the directory of the file containing the directives
    - Check if the file exists
      * If it does not exist
        + Print “[ERROR] file does not exist {file name}”
        + Return from the function
      * Open the file
        + For each line in the file

Split the strings by spaces into an input vector

If vector [0] equals “Print-Graph”

Call printGraph function on the mst object

If vector [0] equals “print-mst”

Call printMST function on the mst object

If vector [0] equals “path”

Call findPath function on the mst Object

Pass vector [1] and vector [2] as parameters

If vector [0] equals “insert-vertex”

Call insertVertex function on the mst object

Pass vector [1] as a parameter

If vector [0] equals “insert-edge”

Call insertEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

If vector [0] equals “decrease-weight”

Call decreaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

If vector [0] equals “increase-weight”

Call increaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

If vector [0] equals “delete-edge”

Call deleteEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) as parameters

If vector [0] equals “delete-vertex”

Call deleteVertex function on the mst object

Pass vector [1] as a parameter

If vector [0] equals “quit”

Return from the function (exits the program)

* + If by keyboard
    - While true
      * Ask the user to enter a directive
      * Split the entered line into an input vector separated by spaces
      * If vector [0] equals “Print-Graph”
        + Call printGraph function on the mst object
      * If vector [0] equals “print-mst”
        + Call printMST function on the mst object
      * If vector [0] equals “path”
        + Call findPath function on the mst Object

Pass vector [1] and vector [2] as parameters

* + - * If vector [0] equals “insert-vertex”
        + Call insertVertex function on the mst object

Pass vector [1] as a parameter

* + - * If vector [0] equals “insert-edge”
        + Call insertEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

* + - * If vector [0] equals “decrease-weight”
        + Call decreaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

* + - * If vector [0] equals “increase-weight”
        + Call increaseWeight function on the mst object

Pass vector [1], vector [2] (vertex identifiers) and vector [3] (weight) as parameters

* + - * If vector [0] equals “delete-edge”
        + Call deleteEdge function on the mst object

Pass vector [1], vector [2] (vertex identifiers) as parameters

* + - * If vector [0] equals “delete-vertex”
        + Call deleteVertex function on the mst object

Pass vector [1] as a parameter

* + - * If vector [0] equals “quit”
        + Return from the function (exits the program)

1. **Test Plan Version 3**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Screenshots**
2. **Error Log**

|  |  |  |
| --- | --- | --- |
| Error Type | Cause of Error | Solution to Error |
|  |  |  |
|  |  |  |

1. **Status**