|  |
| --- |
| CIS 350 – Data Structures |
| Program 3 – Turn in 2 |
| Fall 2014 |

|  |
| --- |
| Daniel Frey  11-24-2014 |

Table of Contents

[1. Problem Summary 2](#_Toc404093558)

[2. Requirements Document 2](#_Toc404093559)

[2.1. Program Requirements 2](#_Toc404093560)

[2.2. Implementation Requirements 3](#_Toc404093561)

[2.3. Assumptions 3](#_Toc404093562)

[3. Decomposition Diagram 4](#_Toc404093563)

[4. Order 5](#_Toc404093564)

[5. Testing Strategy 5](#_Toc404093565)

[6. Test Plan – Version 1 5](#_Toc404093566)

[7. Initial Algorithm 6](#_Toc404093567)

[8. Test Plan – Version 2 7](#_Toc404093568)

[9. Final Algorithm 8](#_Toc404093569)

[10. Program Source Code 8](#_Toc404093570)

[11. Test Plan – Version 3 8](#_Toc404093571)

[12. Screenshots and Data 8](#_Toc404093572)

[13. Error Log 8](#_Toc404093573)

[14. Final Program Status 8](#_Toc404093574)

# Problem Summary

Input an undirected graph with weighted edges and create a minimum spanning tree, which will update with changes to the graph.

# Requirements Document

## Program Requirements

* For graph input
  + Take an input file of vertices
    - First line contains number of vertices
    - Following lines contain vertices
  + Take an input file of edges
    - First line contains number of edges
    - Following contain two vertices and weight for the edge
  + Store in a data structure
    - Maximum of 100 vertices
* Compute minimum spanning tree using Prim’s Algorithm
  + First input vertex is starting point
  + Output sequence
* Update directives
  + May come from file and/or keyboard input
  + Directives that alter graph will cause graph to update
    - **Print the graph:** Print the entire graph.
    - **Print the MST:** Print the contents of the current MST(s). Print the MST according to a preorder traversal of the graph given the current root(s).
    - **Path:** Given two vertex identifiers, compute and print the weight and path between them in the MST using Dijkstra’s algorithm. If the vertices are not in the same tree, then print a message indicating this.
    - **Insert vertex:** Insert a vertex (with no edges) in the graph given its identifier.
    - **Insert edge:** Insert an edge between two existing vertices into the graph
    - **Decrease weight:** Decrease the weight on an existing edge in the graph by the given amount.
    - **Delete vertex:** Delete the given vertex from the graph and all its incident edges.
    - **Delete edge:** Delete the given edge from the graph.
    - **Increase weight:** Increase the weight on an existing edge by the given amount.
  + After each update, a message with what was done will be given

## Implementation Requirements

* Three separate data structures
  + Undirected weighted graph
    - Must use adjacency list or matrix
    - Edges must have cross links between
  + Multiway tree
    - Used to store minimum spanning tree
      * Use firstChild nextSibling representation
      * Parent pointer
  + Binary heap
    - Used in Prim’s Algorithm
* Min spanning tree will be rebuilt from scratch after each update operation
* Cannot use sophisticated data structures
  + Priority queues, trees, dictionaries, etc.

## Assumptions

* Input files are correct/valid
  + Vertex file will contain at least 2 vertices
  + Edge file will contain at least 1 edge
* Directive input will be valid

# Decomposition Diagram

# Order

1. Graph
   1. Print graph
2. Minimum spanning tree
   1. Binary heap
   2. Prim’s algorithm
   3. Print MST
3. Update directives
   1. Dijkstra’s algorithm

# Testing Strategy

* Vertex file
* Edge file
* Minimum spanning tree
* Update Directives

# Test Plan – Version 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Strategy Category | Test Number | Description | Input | Expected Result | Actual Result | Pass/Fail |
| Vertex file | 1.1 | Contains correct number of vertices |  |  |  |  |
| Vertex file | 1.2 | Duplicates ignores |  |  |  |  |
| Edge file | 2.1 | Contains correct number of edges |  |  |  |  |
| Edge file | 2.2 | Edges contain existing vertices |  |  |  |  |
| Edge file | 2.3 | Duplicates ignored |  |  |  |  |
| MST | 3.1 | Edges added to MST |  |  |  |  |
| MST | 3.2 | Sequence output |  |  |  |  |
| Update directives | 4.1 | Print graph |  |  |  |  |
| Update directives | 4.2 | Print MST |  |  |  |  |
| Update directives | 4.3 | Find path using Dijkstra’s algorithm |  |  |  |  |
| Update directives | 4.4 | Insert vertex |  |  |  |  |
| Update directives | 4.5 | Insert edge |  |  |  |  |
| Update directives | 4.6 | Decrease weight |  |  |  |  |
| Update directives | 4.7 | Delete vertex |  |  |  |  |
| Update directives | 4.8 | Delete edge |  |  |  |  |
| Update directives | 4.9 | Increase weight |  |  |  |  |

# Initial Algorithm

* While the vertex file is not empty
  + Read in number of vertices
    - If > 100
      * Fail
  + Read in vertices
    - Store vertex info in vertex node
      * Contains identifier
      * Contains pointer to adj. matrix
  + If numbers do not match
    - Give error and stop program
* While the edge file is not empty
  + Read in number of edges
  + Read in edges
    - Store edge info in edge node
      * Contains neighboring edge
      * Contains edge weight
      * Contains cross link
    - Put edge weight in adjacency matrix
  + If numbers do not match
    - Give error and stop program
* Binary heap
  + Pointer to array of data
    - Dynamic array
  + Left child
    - 2i+1
  + Right child
    - 2i+2
  + Parent
    - (i-1)/2
  + Get min
    - Return array[0]
  + Is empty
    - True if heap size is 0
  + Bubble up
    - If index is not equal to 0
      * If parent greater than current
        + Swap
        + Repeat bubble up
  + Insert
    - Increment heap size by 1
    - Insert element in array of heap size – 1
    - Bubble up from heap size – 1
  + Bubble down
    - If no children
      * Done
    - If one child
      * Swap if greater than
      * Bubble down from child
    - If two children
      * Find smallest child and swap
      * Bubble down from child
  + Delete
    - Copy last element to root
    - Decrement heap size by 1
    - Bubble down from root
  + Prim’s algorithm
    - From the remaining edges select the minimum cost edge
      * Add to the already selected edges
  + Dijkstra’s algorithm
    - From all vertices that have not been visited
      * select the one that results in the least additional total path length

# Test Plan – Version 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Strategy Category | Test Number | Description | Input | Expected Result | Actual Result | Pass/Fail |
| Vertex file | 1.1 | Contains correct number of vertices | Vertex file | Num of vertices matches num given |  |  |
| Vertex file | 1.2 | Duplicates ignores | Duplicate vertex | Duplicate ignored |  |  |
| Edge file | 2.1 | Contains correct number of edges | Edge file | Num of edges matches num given |  |  |
| Edge file | 2.2 | Edges contain existing vertices | Edge | Edge added if existing vertices |  |  |
| Edge file | 2.3 | Duplicates ignored | Duplicate edge | Duplicate ignored |  |  |
| MST | 3.1 | Edges added to MST | Adjacency matrix | Min cost edge added |  |  |
| MST | 3.2 | Order of sequences output | Adjacency matrix | Steps of building MST |  |  |
| Update directives | 4.1 | Print graph | User directive | Graph printed |  |  |
| Update directives | 4.2 | Print MST | User directive | MST printed |  |  |
| Update directives | 4.3 | Find path using Dijkstra’s algorithm | User directive, MST | Path found in MST |  |  |
| Update directives | 4.4 | Insert vertex | User directive | Vertex inserted |  |  |
| Update directives | 4.5 | Insert edge | User directive | Edge inserted |  |  |
| Update directives | 4.6 | Decrease weight | User directive | Weight decreased |  |  |
| Update directives | 4.7 | Delete vertex | User directive | Vertex deleted |  |  |
| Update directives | 4.8 | Delete edge | User directive | Edge deleted |  |  |
| Update directives | 4.9 | Increase weight | User directive | Weight increased |  |  |

# Final Algorithm

* While the vertex file is not empty
  + Read in number of vertices
    - If > 100
      * Fail
  + Read in vertices
    - * Store in mapping with specific identifier for matrix
      * Since multiples may exist, remap with new identifiers
  + If numbers do not match
    - Give error and stop program
* While the edge file is not empty
  + Read in number of edges
  + Read in edges
    - Put edge weight in adjacency matrix
  + If numbers do not match
    - Give error and stop program
* Binary heap
  + Pointer to array of data
    - Dynamic array
  + Left child
    - 2i+1
  + Right child
    - 2i+2
  + Parent
    - (i-1)/2
  + Get min
    - Return array[0]
  + Is empty
    - True if heap size is 0
  + Bubble up
    - If index is not equal to 0
      * If parent greater than current
        + Swap
        + Repeat bubble up
  + Insert
    - Increment heap size by 1
    - Insert element in array of heap size – 1
    - Bubble up from heap size – 1
  + Bubble down
    - If no children
      * Done
    - If one child
      * Swap if greater than
      * Bubble down from child
    - If two children
      * Find smallest child and swap
      * Bubble down from child
* Prim’s algorithm
  + From the remaining edges select the minimum cost edge
  + Add to the already selected edges
* Graph
  + Build
    - Initialize array to input size
    - Cycle through all locations and set equal to 0
  + Add edge
    - Check if valid vertices
    - If edge doesn’t already exist, set value to weight
  + Insert vertex
    - If vertex does not exist
    - Make mapping of vertex with unique identifier
    - Create new adjacency matrix of +1 size
    - Copy old adj. matrix elements to new
    - Set row/column of added vertex to 0
    - Delete old
    - Old pointer = new adj. matrix
  + Delete vertex
    - If vertex does exist
    - Create new array of size of one less than current
    - Look up location of vertex in array
    - Swap columns/rows from location and onward
    - Copy from old to new array minus the deleted vertex row/column (outer most)
    - Redo mapping
    - Update old=new
  + Delete edge
    - Look up vertices to see if they exist
    - Go to location and set weight to 0
  + Increase weight
    - Look up vertices to see if they exist
    - Go to location and increment by weight
  + Decrease weight
    - Look up vertices to see if they exist
    - Go to location and decrement by weight

# Program Source Code

Please see attached file “Source Code”

# Test Plan – Version 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Strategy Category | Test Number | Description | Input | Expected Result | Actual Result | Pass/Fail |
| Vertex file | 1.1 | Contains correct number of vertices | Vertex file | Num of vertices matches num given | Continue | Pass |
| Vertex file | 1.2 | Duplicates ignores | Duplicate vertex | Duplicate ignored | Ignored duplicate | Pass |
| Vertex file | 1.3 | Vertex numbers don’t match | Vertex file | Stop program | Program stopped | Pass |
| Vertex file | 1.4 | Too many vertices | File with >100 vertices | Stop program | Program stopped | Pass |
| Edge file | 2.1 | Contains correct number of edges | Edge file | Num of edges matches num given | Continue | Pass |
| Edge file | 2.2 | Edges contain existing vertices | Edge | Edge added if existing vertices | Edge added | Pass |
| Edge file | 2.3 | Duplicates ignored for (i,j) and (j,i) | Edge file, Duplicate edge | Duplicate ignored | Duplicated not added | Pass |
| Edge file | 2.4 | Edge numbers don’t match | Edge file | Stop program | Program stopped | Pass |
| Edge file | 2.5 | Order of edges added | Edge file | Edge (i,j) added | Edge added | Pass |
| MST | 3.1 | Edges added to MST | Adjacency matrix | Min cost edge added | Min edge added | Pass |
| MST | 3.2 | Order of sequences output | Adjacency matrix | Steps of building MST | Steps of adding to MST | Pass |
| MST | 3.3 | Disjoint graph MST | Adjacency matrix | Multiple MSTs | None | Fail |
| Update directives | 4.1 | Print graph | print-graph | Graph printed | Matrix graph printed | Pass |
| Update directives | 4.2 | Print MST | print-mst | MST printed | MST printed | Pass |
| Update directives | 4.3 | Find path using Dijkstra’s algorithm | path u v | Path found in MST | None | Fail |
| Update directives | 4.4 | Insert vertex | insert-vertex u | Vertex inserted | Inserted vertex | Pass |
| Update directives | 4.5 | Insert edge if edge does (not) exist | insert-edge u v w | Edge inserted | Edge (not) added | Pass |
| Update directives | 4.6 | Decrease weight of (non) existing edge | decrease-weight u v w | Weight decreased | Weight (not) decreased | Pass |
| Update directives | 4.7 | Delete vertex if vertex does (not) exists | delete-vertex u | Vertex deleted | Deleted Vertex | Pass |
| Update directives | 4.8 | Delete edge if edge does (not) exists | delete-edge u v w | Edge deleted | Edge (not) removed | Pass |
| Update directives | 4.9 | Increase weight of (non) existing edge | increase-weight u v w | Weight increased | Weight (not) increased | Pass |
| Update directives | 4.10 | Insert vertex if 100 already exist | Insert-vertex u | Vertex not inserted | Vertex not inserted | Pass |
| Update directives | 4.11 | Edges of deleted vertex removed | delete-vertex u | Edges removed | Removed edges | Pass |
| Update directives | 4.12 | Updates from user input | User input | Prompt user for directive | User to input directive | Pass |
| Update directives | 4.13 | Updates from file | Directive file | Updates executed from file | Updated from file | Pass |
| Update directives | 4.14 | Updates from user and file | User input, directive file | Updates executed from both | Updated from both | Pass |

# Screenshots and Data

Please see attached files/folders.

# Error Log

|  |  |  |  |
| --- | --- | --- | --- |
| Error Type | Error | How error was found | Solution |
| Logic | Reading in newline character after reading in number variable | Manual code review, breakpoints | Get the remaining parts of file line after reading first number |
| Logic | Edge weights displaying as -1 in print MST | Output review | Changed initialized value of (i,j) to 0 instead of -1 |
| Logic | isEdge checks for value not in adjacency matrix, -1 | Manual code review | Changed to check for 0 instead of -1 |
| Logic | delete-edge only deleted edge from part of graph | Output review | Delete (j,i) as well |
| Logic | No check for invalid vertex | Output review | Added check |
| Logic | Increase/decrease weight not changing value in reverse location | Output review | Change in (j,i) as well |
| Log | Prim MST not updating after inserting vertex | Output review | Update number |

# Final Program Status

Not working: disjoint graphs/single vertex MST, path.

Remaining works.