|  |
| --- |
| CIS 350 – Data Structures |
| Program 3 – Turn in 1 |
| Fall 2014 |

|  |
| --- |
| Daniel Frey  11-17-2014 |

Table of Contents

[1. Problem Summary 2](#_Toc403852311)

[2. Requirements Document 2](#_Toc403852312)

[2.1. Program Requirements 2](#_Toc403852313)

[2.2. Implementation Requirements 3](#_Toc403852314)

[2.3. Assumptions 3](#_Toc403852315)

[3. Decomposition Diagram 4](#_Toc403852316)

[4. Order 5](#_Toc403852317)

[5. Testing Strategy 5](#_Toc403852318)

[6. Test Plan – Version 1 5](#_Toc403852319)

[7. Initial Algorithm 6](#_Toc403852320)

[8. Test Plan – Version 2 7](#_Toc403852321)

# 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 |  |  |