**Algorithms and Data Structures**

*­­MST Report*

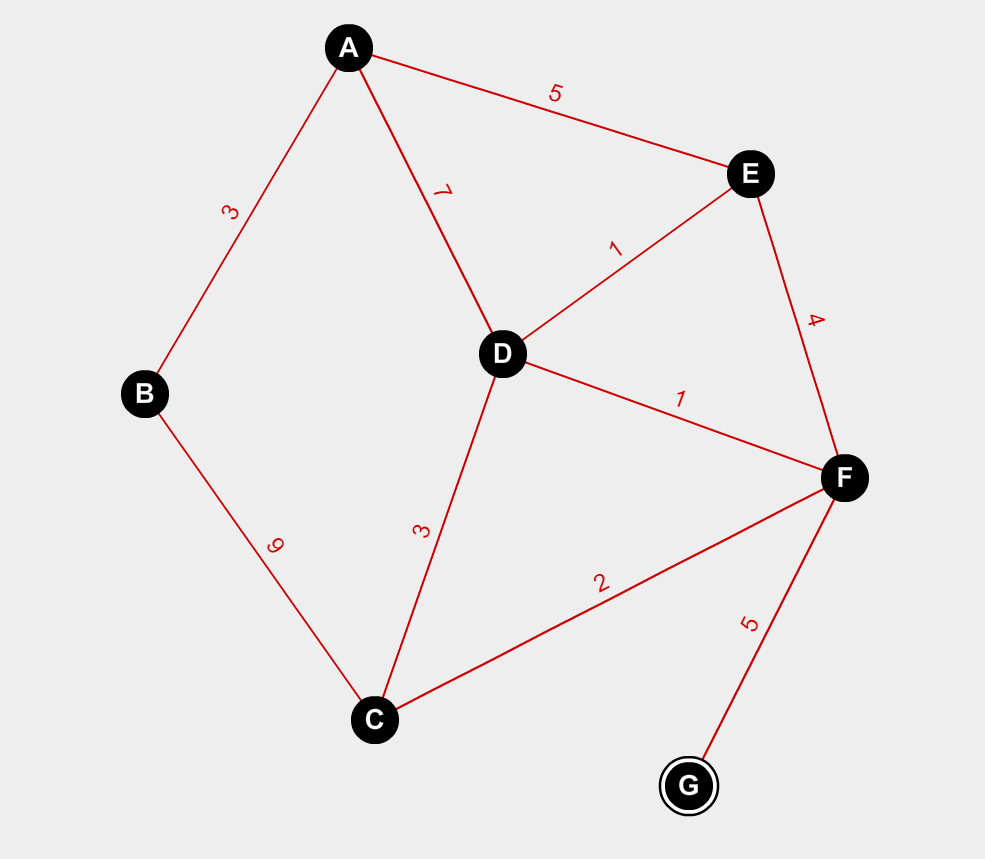
*Daniel Tilley – C14337041*

**Introduction**

This java program uses Prims Algorithm to figure out the MST (Minimum Spanning Tree) of an undirected weighted graph. Prims Algorithm is a greed algorithm that uses heaps and arrays along with linked lists to return the desired result. It starts at a node that is entered by the user, goes through all other nodes on the graph and calculates an MST accordingly. The graph is read in from a text file (explained below) and once the file has been read it will display the vertices and edge positions using heaps, arrays and linked lists. The program will then begin to execute the algorithm and return the MST along with the MST weight.

**The Graph**

Below is the graph I used to test my java program. It consists of 7 vertices and 9 edges. On the left we have the graph written down in number form. The first line indicates the number of vertices and the number of edges. From then on, each line is an indication of which vertex is connected to which vertex and how much weight their shared edge has. For example, the second line tells us that point 1 (vertex A) is connected to point 2 (vertex B) and has a weight of 3. This holds true for all lines that follow also.

7 9

1 2 3

1 4 7

1 5 5

2 3 9

3 4 3

3 6 2

4 5 1

5 6 4

6 7 5

**MST Graph and Construction**

In order to implement prims algorithm in Java, you first need to create a few arrays and classes in order for it to work. The steps taken by the algorithm are listed below:

1. Initialise a tree with one vertex chosen by user.
2. Add one edge to the tree that connects the previous vertex to vertices that are not in the tree yet.
3. Repeat step 2 until all edges have been added

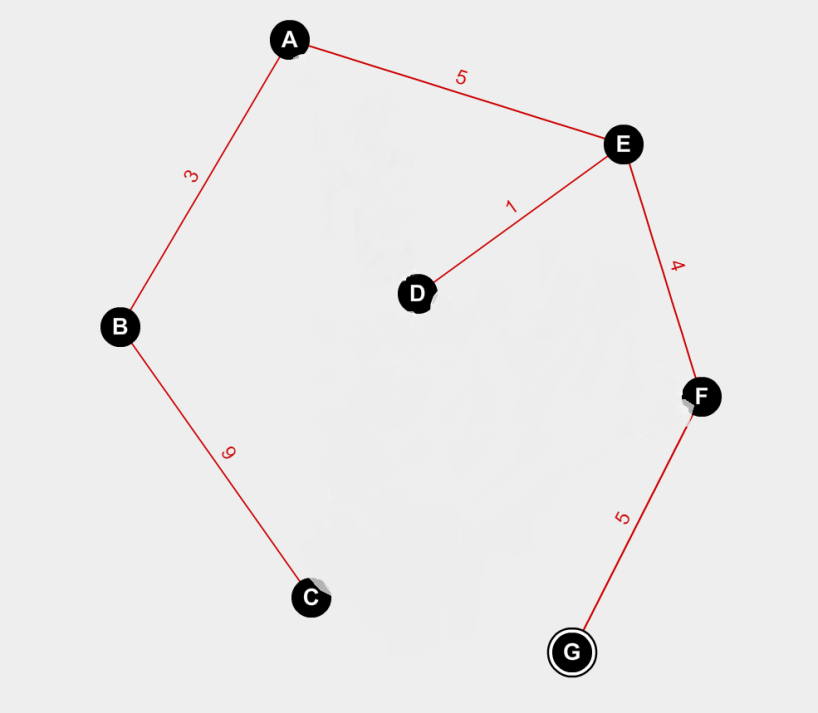
Heap h: This is a class that is used as a representation of the heap. It is used to find the next nearest vertex to the current vertex so that it can be added to the MST.

hPos []: This is an integer array that is used to store the position of the vertices within the heap. This array is used in conjunction with the heap array a[] and it can be said that a point on the heap is stored at hPos[v] in the array a[].

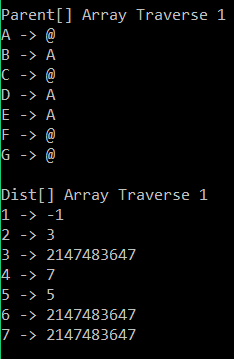
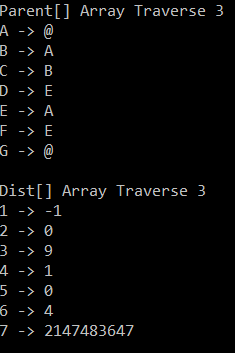
Dist []: This is an integer array that records the current distance of a vertex that is used in the MST

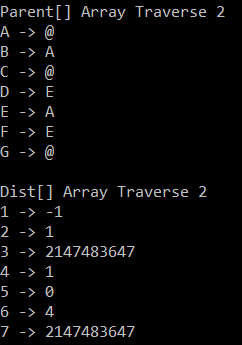
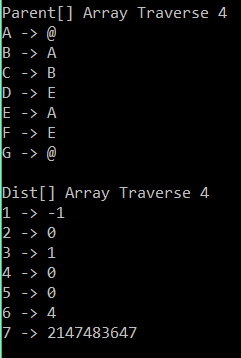
Parent []: This is an integer array that is the used as a final storage place to hold the MST.

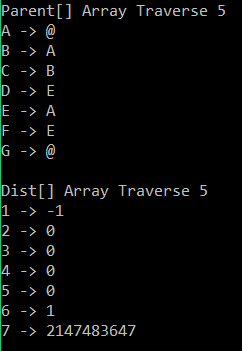
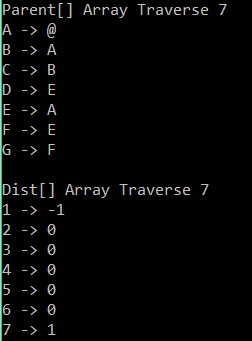
Below is a graphical representation of a MST given that you start from point A. The total weight is 27.

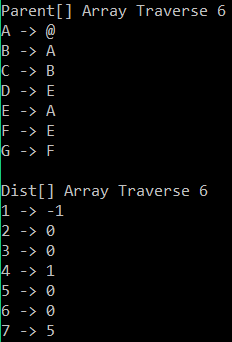
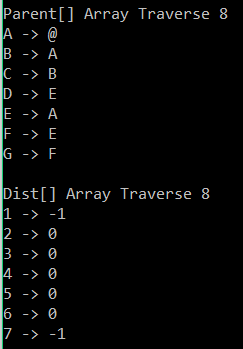
****

Below are the contents of both the parent[] and dist[] arrays after each transversal of the graph. I used simple for loops to print out the contents in Java. The max int value has been used when initialising the dist[] array.



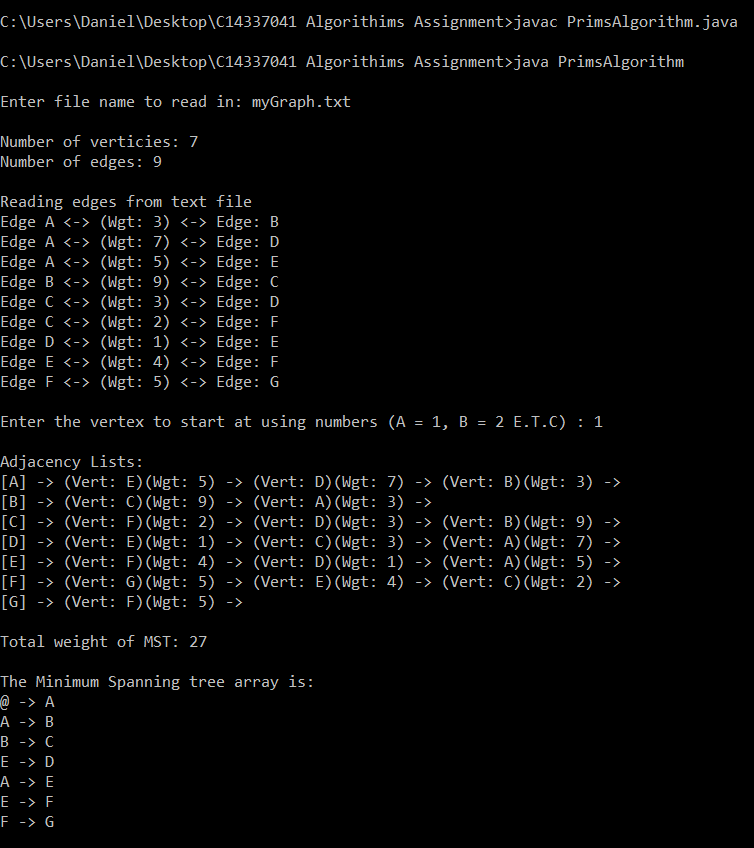






**Execution of code and Adjacency list diagram**

Below you can see what the code output looks like when run from the command line.



**Conclusion**

This assignment has been very useful for a number of reasons. The first thing I found helpful about the assignment was re-enforcing my knowledge of how to use the command line with Java. It also made me think about what methods I had to use whilst writing the code in Java as Eclipse or most other Java IDE’s will give you hints when typing out code.

Secondly, this assignment has vastly improved my knowledge on heaps and of course prims algorithm. The heap is a very powerful data structure coding and executing the program has helped me to realise this as well as some different ways of using the heap.

Finally I have learned how to take a graph and change it to a format that is readable through text. This can be very helpful and powerful when creating programs in the future.

**Code**

//Simple weighted graph representation

//Uses an Adjacency Linked Lists, suitable for sparse graphs

//Prims Algorithim

//Daniel Tilley

//C14337041

import java.io.\*;

import java.util.Scanner;

class Heap {

public int[] h; // heap array

public int[] hPos; // hPos[h[k]] == k

public int[] dist; // dist[v] = priority of v

private int N; // heap size

// The heap constructor gets passed from the Graph:

// 1. maximum heap size

// 2. reference to the dist[] array

// 3. reference to the hPos[] array

public Heap(int maxSize, int[] dist, int[] hPos) {

N = 0;

h = new int[maxSize + 1];

this.dist = dist;

this.hPos = hPos;

}//end constructor

public boolean isEmpty() {

return (N == 0);

}//end id empty

//Method used from previous lab test

public void siftUp(int k) {

int v = h[k];

h[0] = 0;

dist[0] = 0;

while (dist[v] < dist[h[k / 2]]) {

h[k] = h[k / 2];

hPos[h[k]] = k;

k = k / 2;

}//end while

h[k] = v;

}//end sift up

//Method used from previous lab test

public void siftDown(int k) {

int v;

v = h[k];

h[0] = Integer.MAX\_VALUE;

while (k <= N / 2) {

int j = 2 \* k;

if (j < N && dist[h[j]] > dist[h[j + 1]]) {

j++;

}//end if

if (dist[v] <= dist[h[j]]) {

break;

}//end if

h[k] = h[j];

hPos[h[k]] = k;

k = j;

}//end while

h[k] = v;

hPos[v] = k;

}//end sift down

public void insert(int x) {

h[++N] = x;

siftUp(N);

}//end insert

public int remove() {

int v = h[1];

hPos[v] = 0; // v is no longer in heap

h[N+1] = 0; // put null node into empty spot

h[1] = h[N--];

siftDown(1);

return v;

}//end remove

}//end class heap

class Graph {

class Node {

public int vertex; // vertex variable

public int weight; // weight variable

public Node next; //next node in array

//node constructor

Node(int vertex, int weight, Node n) {

this.vertex = vertex;

this.weight = weight;

next = n;

}//end node constructor

//default constructor

Node(){

}//end default constructor

}//end class node

// V = number of vertices

// E = number of edges

// adj[] is the adjacency lists array

private int V, E;

private Node[] adj;

private Node z;

//Array to hold MST

private int[] MST;

//Used for calculating size of array

private int count = 0;

private int last = Integer.MIN\_VALUE;

// used for moving through graph

private int[] visited;

private int id;

//size of graph array

public int getCount() {

return (count);

}//end get count

//return last vertex in graph

public int getLast() {

return (last);

}//end get last

public Graph(String graphFile) throws IOException {

int u, v;

int e, weight;

Node t;

//for reading in data from file

FileReader fr = new FileReader(graphFile);

BufferedReader reader = new BufferedReader(fr);

//multiple whitespace as delimiter

String splits = " +";

String line = reader.readLine();

String[] parts = line.split(splits);

System.out.println("Number of verticies: " + parts[0] + "\nNumber of edges: " + parts[1] + "\n");

V = Integer.parseInt(parts[0]);

E = Integer.parseInt(parts[1]);

//create sentinel node

z = new Node();

z.next = z;

//create adjacency lists, initialised to sentinel node z

adj = new Node[V + 1];

for (v = 1; v <= V; ++v)

adj[v] = z;

// read the edges

System.out.println("Reading edges from text file");

//loops through all elements in array

for (e = 1; e <= E; ++e) {

line = reader.readLine();

parts = line.split(splits);

u = Integer.parseInt(parts[0]);//first vertex

v = Integer.parseInt(parts[1]);//second vertex

weight = Integer.parseInt(parts[2]);//weight

System.out.println("Edge " + toChar(u) + " <-> (Wgt: " + weight + ") <-> Edge: " + toChar(v));

//figure out which node is the biggest

if (u > last){

last = u;

}//end if

if (v > last){

last = v;

}//end if

//update adjacency array

adj[v] = new Node(u, weight, adj[v]);

adj[u] = new Node(v, weight, adj[u]);

//increment number of elements in array

count++;

}//end for

}//end Graph Class

// convert vertex into char for pretty printing

private char toChar(int u) {

return (char)(u + 64);

}//end toChar

// method to display the graph representation

public void display() {

int v;

Node n;

//print out adjacency lists

System.out.print("\nAdjacency Lists:");

for (v = 1; v <= V; ++v) {

System.out.print("\n[" + toChar(v) + "] ->");

for (n = adj[v]; n != z; n = n.next){

System.out.print(" (Vert: " + toChar(n.vertex) + ")(Wgt: " + n.weight + ") ->");

}//end for

}//end for

}//end display

public void MST\_Prim(int s, int count) {

int v, u;

int weight, wgt\_sum = 0;

//create new arrays for storing graph data

int[] dist = new int[count];

int[] parent = new int[count];

int[] hPos = new int[count];

Node t;

int countVar = 1; //variable used to calculate traverse

//initialise arrays

for (v = 0; v <= V; v++) {

dist[v] = Integer.MAX\_VALUE;

parent[v] = 0;

hPos[v] = 0;

}//end for

//create a new heap and insert last element (s)

Heap h = new Heap(V, hPos, dist);

h.insert(s);

dist[s] = 0;

Heap pq = new Heap(V, dist, hPos);

pq.insert(s);

//Most of alogrithim here

//run while heap is empty

while (!h.isEmpty()) {

v = h.remove();

dist[v] = -dist[v];

Node n;

int w;

//run loop while node is not equal to the sentinal node

for (n = adj[v]; n != z; n = n.next) {

u = n.vertex;

w = n.weight;

//check if current weight is less that distance stored in array

if (w < dist[u]) {

if (dist[u] != Integer.MAX\_VALUE) {

wgt\_sum -= dist[u];

}//end if

dist[u] = w;

parent[u] = v;

wgt\_sum += w;

//if node is at position 0 in array, insert into array

if (hPos[u] == 0) {

h.insert(u);

}//end if

//otherwise sift the element up the graph until it reaches position 0

else {

h.siftUp(hPos[u]);

}//end else

}//end if

}//end for

//print out the traverse of each array

////////////////////////////////////////////////////////////////////////////////////

System.out.println("");

System.out.println("Parent[] Array Traverse " + countVar);

for(int i = 1; i <= V; i ++){

System.out.println( toChar(i) + " -> " + toChar(parent[i]));

}//end for

System.out.println("");

System.out.println("Dist[] Array Traverse " + countVar);

for(int j = 1; j <= V; j ++) {

System.out.println(j + " -> " + dist[j]);

}//end for

//update count var

countVar++;

////////////////////////////////////////////////////////////////////////////////////

}//end while

//set count var back to 1 for error checking

countVar = 1;

//end of algorithim and print out results

System.out.print("\n\nTotal weight of MST: " + wgt\_sum);

MST = parent;

}

//display min spanning tree

public void displayMST() {

System.out.println("\n\nThe Minimum Spanning tree array is:");

for (int v = 1; v <= V; ++v) {

System.out.println(toChar(MST[v]) + " -> " + toChar(v)); // copied and changed from skeleton code

}//end for

}//end display MST

}//End Class Graph

public class PrimsAlgorithm {

// get user input

public static String getInput(String userFile) throws IOException {

Console console = System.console();

String input = console.readLine(userFile);

return input;

}//end getInput

//main method

public static void main(String[] args) throws IOException {

// error handling

System.out.println(" ");

String fname = new String(getInput("Enter file name to read in: "));

System.out.println(" ");

//variables

boolean checkFile = true; //used to check if file has been read or not

Graph graph = null;

//while loop that checks for any user errors when entering the file name

while (checkFile) {

//file name to read in graph

String newfName = new String();

try {

graph = new Graph(fname);

//update checkFile so that loop wont run again if there is an error when reading the file

checkFile = !checkFile;

}//end try

// checks if the file name is incorrect

catch (IOException e) {

try {

newfName = getInput("File not found, enter a file name: ");

}//end try

catch (IOException f) {

System.out.println("Invalid input");

}//end inner catch

//change fname to new file name

fname = newfName;

}//end outter catch

}//end while

//copied from skeleton code to stop errors in code

if (graph == null) {

graph = new Graph("wgraph3.txt");

}//end if

checkFile = true;

int getNum = graph.getLast();

//loop to calculate MST

while (checkFile) {

getNum = 0;

try {

System.out.println(" ");

getNum = Math.abs(Integer.parseInt(getInput("Enter the vertex to start at using numbers (A = 1, B = 2 E.T.C) : ")));

checkFile = false;

}//end try

//checks user has not enetered a string or character

catch (IOException f) {

System.out.println("Invalid input, must be of type integer");

}//end catch

//checks the number is not higher than elements in array

if (checkFile == false && getNum > graph.getLast()) {

System.out.println("Number is too high, please enter a number under " + (graph.getLast() + 1));

checkFile = true;

}//end if

}//end while

// updaters

graph.display();

graph.MST\_Prim(getNum, graph.getCount());

graph.displayMST();

}//end main

}//end class Prims Algorithim