|  |  |
| --- | --- |
| Systems and Algorithms  MST Report  Donncha Cassidy-Hand   |  | | --- | | D14123580 | |

**Introduction**

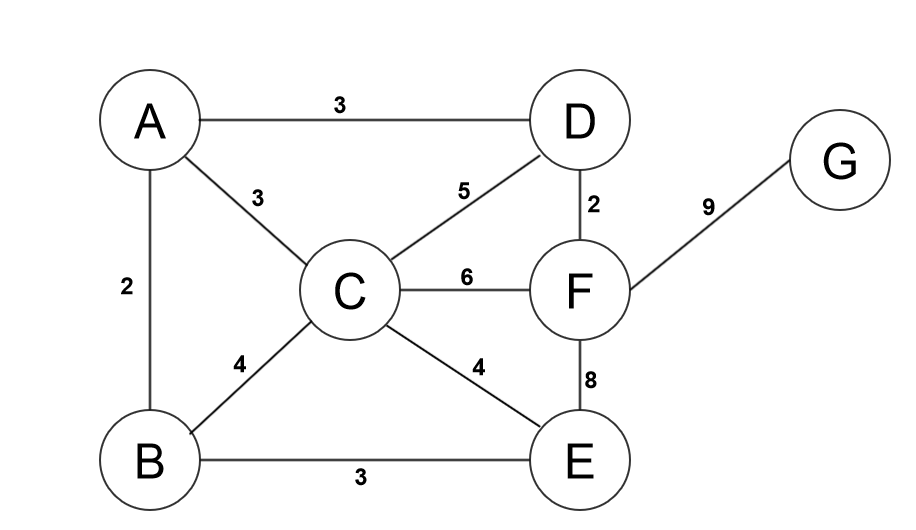
This program uses Prim’s Algorithm to return Minimum Spanning Tree weight. It uses heaps and linked lists to create and calculate the matrix through reading in a specified file. Once the file has been read it will create a linked list in an attempt to display the vertices and edges positions. After this it will then cycle through the heap to find the MST and output the calculation of the total weight of the MST.

**Prim’s Algorithm Explained:**

Is a greed algorithm that finds a minimum spanning tree (MST) for a connected weighted undirected graph, this means it will complete the graph visiting every node but completing it with the minimum weight possible. The algorithm will work at any selected node and will generate the MST from that node onward to complete the graph.

**My Diagram**

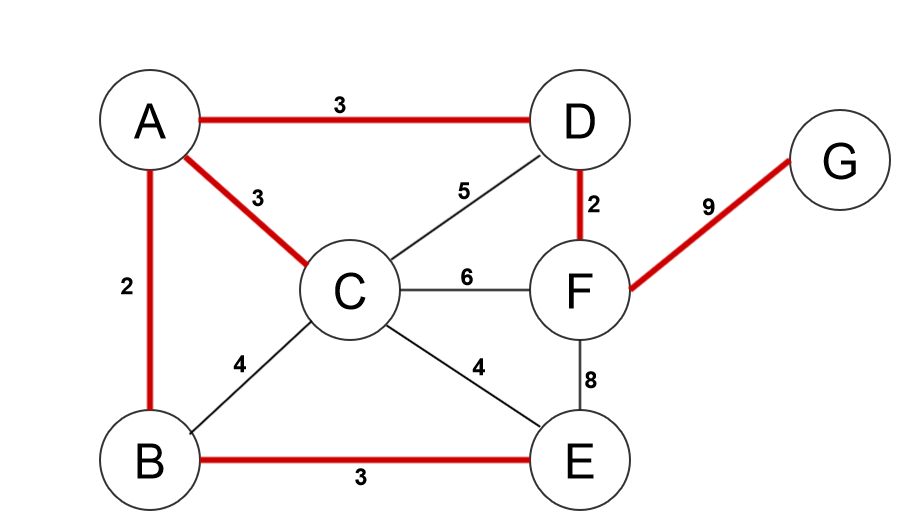
This is a diagram I have constructed for testing purposes with the program I have created.



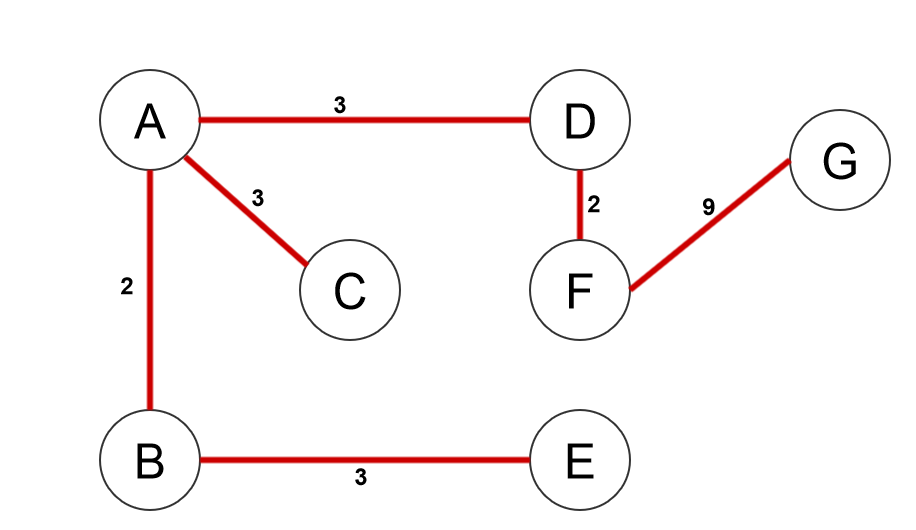
Below is a numbered interpretation representing the diagram above, this is so that the program may read this numbered version of the diagram in. After the numbers have been read in it will create a logical linked list to represent it as a diagram and then preform the calculations and output it in a manner which is represented in the diagram using Letters as the vertices rather than numbers as it reads it in.

|  |
| --- |
| 7 11  1 2 2  1 3 3  1 4 3  2 3 4  2 5 3  3 4 5  3 5 1  3 6 6  4 6 7  5 6 8  6 7 9 |

This is the graph showing the MST circuit or route highlighted in red after Prim’s algorithm has been provided to show the MST of the graph, assuming we start on Node ‘A’.



As the MST circuit or route is been displayed the graph will look like this for the MST.



**Output:**

Adding the weight of all the edges that the traverse route taken for the minimal spanning tree we can obtain the weight of the graph.

In this case it is…

2 + 3 + 3 + 3 + 2 + 9 = 22

So the weight of the minimal spanning tree starting Node A is 22.

The traverse route is represented as…

@ -> A

A -> B

A -> C

A -> D

B -> E

D -> F

F -> G

**Step by Step construction of MST:**

Heap h : is a heap to find the next vertex nearest the MST so that it can be added to the MST.

Dist[] : is an int array that records the current distance of a vertex from the MST

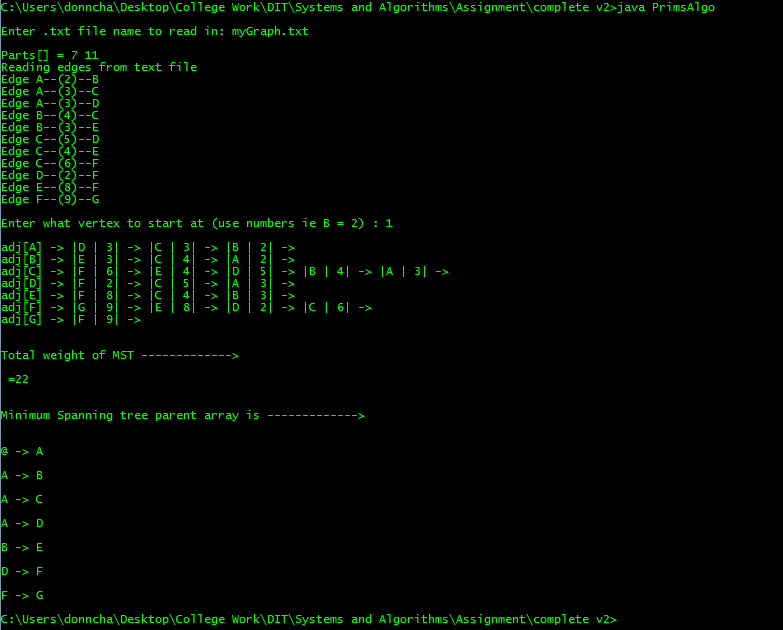
Parent[] : is an int array that stores the MST

hPos[] : is an int array that records the position of any vertex with the heap array a[]. Vertex v is in position hPos[v] in a[].

**Conclusion:**

I have produced a program which can take in a graph presented in a text format, allowing the user to provide an input on which vertex to start at and generate the MST of the graph from there, using Prim’s Algorithm.

Seen in use below:



**What I have learned:**

* Have gained extensive knowledge of graph traversing algorithms and its application to a computer program.
* Have done multiple graph calculations using Prim’s Algorithm, fully understanding how the algorithm works and why.
* How to construct graphs that are applicable to Prim’s Algorithm and how to manually convert the graph into a text file so it’s readable by the program.
* Real world applications of Prim’s Algorithm.
* Gained further understanding of heaps and how they work.

**Dist and Parent array printout:**

Below is the printout of the Dist and Parent arrays after each traverse.

|  |  |
| --- | --- |
| **Dist[]** | **Parent[]** |
| **Traverse 1** | |
| A -> @  B -> A  C -> A  D -> A  E -> @  F -> @  G -> @ | 1 -> -1  2 -> 2  3 -> 3  4 -> 3  5 -> ∞  6 -> ∞  7 -> ∞ |
| **Traverse 2** | |
| A -> @  B -> A  C -> A  D -> A  E -> @  F -> D  G -> @ | 1 -> -1  2 -> 1  3 -> 3  4 -> -3  5 -> ∞  6 -> 7  7 -> ∞ |
| **Traverse 3** | |
| A -> @  B -> A  C -> A  D -> A  E -> B  F -> D  G -> F | 1 -> -1  2 -> -1  3 -> 3  4 -> -3  5 -> 1  6 -> -1  7 -> 9 |
| **Traverse 4** | |
| A -> @  B -> A  C -> E  D -> A  E -> B  F -> D  G -> F | 1 -> -1  2 -> -1  3 -> 1  4 -> -3  5 -> -1  6 -> -1  7 -> 1 |
| **Traverse 5** | |
| A -> @  B -> A  C -> E  D -> A  E -> B  F -> D  G -> F | 1 -> -1  2 -> -1  3 -> 1  4 -> -3  5 -> -1  6 -> -1  7 -> -1 |
| **Traverse 6** | |
| A -> @  B -> A  C -> E  D -> A  E -> B  F -> D  G -> F | 1 -> -1  2 -> -1  3 -> -1  4 -> -3  5 -> -1  6 -> -1  7 -> -1 |
| **Traverse 7** | |
| A -> @  B -> A  C -> E  D -> A  E -> B  F -> D  G -> F | 1 -> -1  2 -> -1  3 -> -1  4 -> -3  5 -> -1  6 -> -1  7 -> -1 |

**Algorithms Code:**

Below is a table containing the code from the entire program I created for calculating Prim’s Algorithm:

|  |
| --- |
| //Prim conversion// Simple weighted graph representation  // Uses an Adjacency Linked Lists, suitable for sparse graphs  import java.io.\*;  //import static java.lang.Integer.MAX\_VALUE;  //import static java.lang.Integer.MIN\_VALUE;  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;  }  public void insert(int x) {  h[++N] = x;  siftUp(N);  }  public boolean isEmpty() {  return (N == 0);  }  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;  }  h[k] = v;  }  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++;  }  if (dist[v] <= dist[h[j]]) {  break;  }  h[k] = h[j];  hPos[h[k]] = k;  k = j;  }  h[k] = v;  hPos[v] = k;  }  public int remove() {  int v = h[1];  h[1] = h[N--];  siftDown(1);  return v;  }  }  class Graph {  class Node {  public int vert; // vertex variable  public int wgt; // weight variable  public Node next;  Node(int vert, int wgt, Node n) {  this.vert = vert;  this.wgt = wgt;  next = n;  }  Node() {}  }  private int V, E;  private Node[] adj;  private Node z;  private int[] mst;  private int count = 0;  private int last = Integer.MIN\_VALUE;  private int[] visited;  private int id;  public int getCount() {  return (count);  }  public int getLast() {  return (last);  }  public Graph(String graphFile) throws IOException {  int u, v;  int e, wgt;  Node t;  FileReader fr = new FileReader(graphFile);  BufferedReader reader = new BufferedReader(fr);  String splits = " +";  String line = reader.readLine();  String[] parts = line.split(splits);  System.out.println("Parts[] = " + parts[0] + " " + parts[1]);  V = Integer.parseInt(parts[0]);  E = Integer.parseInt(parts[1]);  z = new Node();  z.next = z;  adj = new Node[V + 1];  for (v = 1; v <= V; ++v)  adj[v] = z;  System.out.println("Reading edges from text file");  for (e = 1; e <= E; ++e) {  line = reader.readLine();  parts = line.split(splits);  u = Integer.parseInt(parts[0]);  v = Integer.parseInt(parts[1]);  wgt = Integer.parseInt(parts[2]);  System.out.println("Edge " + toChar(u) + "--(" + wgt + ")--" + toChar(v));  if (u > last) last = u;  if (v > last) last = v;  adj[v] = new Node(u, wgt, adj[v]);  adj[u] = new Node(v, wgt, adj[u]);  count++;  }  }  private char toChar(int u) {  return (char)(u + 64);  }  public void display() {  int v;  Node n;  for (v = 1; v <= V; ++v) {  System.out.print("\nadj[" + toChar(v) + "] ->");  for (n = adj[v]; n != z; n = n.next)  System.out.print(" |" + toChar(n.vert) + " | " + n.wgt + "| ->");  }  System.out.println("");  }  public void MST\_Prim(int s, int count) {  int v, u;  int wgt, wgt\_sum = 0;  int[] dist = new int[count];  int[] parent = new int[count];  int[] hPos = new int[count];  Node t;  for (v = 0; v <= V; v++) {  dist[v] = Integer.MAX\_VALUE;  parent[v] = 0;  hPos[v] = 0;  }  Heap h = new Heap(V, hPos, dist);  h.insert(s);  dist[s] = 0;  Heap pq = new Heap(V, dist, hPos);  pq.insert(s);  while (!h.isEmpty()) {  v = h.remove();  dist[v] = -dist[v];  Node n;  int w;  for (n = adj[v]; n != z; n = n.next) {  u = n.vert;  w = n.wgt;  if (w < dist[u]) {  if (dist[u] != Integer.MAX\_VALUE) {  wgt\_sum -= dist[u];  }  dist[u] = w;  parent[u] = v;  wgt\_sum += w;  if (hPos[u] == 0) {  h.insert(u);  } else {  h.siftUp(hPos[u]);  }  }  }  // used to print the parent[] and dist[] arrays to see how it traverses through the graph  /\*  System.out.println("\n");  for(int i = 1; i <= V; ++i) {  System.out.println("");  System.out.println( toChar(i) + " -> " + toChar(parent[i]));  }  System.out.println("\n");  for(int b = 1; b <= V; ++b) {  System.out.println("");  System.out.println(b + " -> " + dist[b]);  } \*/  }  System.out.print("\n\nTotal weight of MST ------------->\n\n =" + wgt\_sum + "\n");  mst = parent;  }  public void showMST() {  System.out.println("\n");  System.out.println("Minimum Spanning tree parent array is ------------->\n");  for (int v = 1; v <= V; ++v) {  System.out.println("");  System.out.println(toChar(mst[v]) + " -> " + toChar(v)); // reversed from the skeleton code  }  }  }  public class PrimsAlgo {  public static void main(String[] args) throws IOException {  // error handling  System.out.println(" ");  String fname = new String(getUserInput("Enter .txt file name to read in: "));  System.out.println(" ");  boolean fileError = true;  Graph g = null;  while (fileError) {  String newfName = new String();  try {  g = new Graph(fname);  fileError = !fileError; // wont run if an fileError is caught  } catch (IOException e) { // to catch if the file is incorrect named  try {  newfName = getUserInput("File not found, enter a valid file name: ");  } catch (IOException f) {  System.out.println("Invalid input");  }  fname = newfName; //reassign the string fname to this one.  }  }  if (g == null) {  g = new Graph("wgraph2.txt"); //this will never happen anyway so.  }  fileError = true;  int getNum = g.getLast();  while (fileError) {  getNum = Integer.MIN\_VALUE;  try {  System.out.println(" ");  getNum = Math.abs(Integer.parseInt(getUserInput("Enter what vertex to start at (use numbers ie B = 2) : ")));  fileError = false;  } catch (IOException f) {  System.out.println("Invalid input, must use numeric values");  }  if (fileError == false && getNum > g.getLast()) {  System.out.println("Number to high, must be below " + (g.getLast() + 1));  fileError = true;  }  }  // updaters  g.display();  g.MST\_Prim(getNum, g.getCount());  g.showMST();  }  // get user input  public static String getUserInput(String q) throws IOException {  Console console = System.console();  String input = console.readLine(q);  return input;  }  } |