PRISM ALGORITHM

package com.sorting;

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class MST {

private static final int V = 5;

int minKey(int key[], Boolean mstSet[])

{

// Initialize min value

int min = Integer.MAX\_VALUE, min\_index = -1;

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

if (mstSet[v] == false && key[v] < min) {

min = key[v];

min\_index = v;

}

return min\_index;

}

// A utility function to print the constructed MST stored in

// parent[]

void printMST(int parent[], int graph[][])

{

System.out.println("Edge \tWeight");

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

System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);

}

// Function to construct and print MST for a graph represented

// using adjacency matrix representation

void primMST(int graph[][])

{

// Array to store constructed MST

int parent[] = new int[V];

// Key values used to pick minimum weight edge in cut

int key[] = new int[V];

// To represent set of vertices not yet included in MST

Boolean mstSet[] = new Boolean[V];

// Initialize all keys as INFINITE

for (int i = 0; i < V; i++) {

key[i] = Integer.MAX\_VALUE;

mstSet[i] = false;

}

// Always include first 1st vertex in MST.

key[0] = 0; // Make key 0 so that this vertex is

// picked as first vertex

parent[0] = -1; // First node is always root of MST

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

// Pick thd minimum key vertex from the set of vertices

// not yet included in MST

int u = minKey(key, mstSet);

// Add the picked vertex to the MST Set

mstSet[u] = true;

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

if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

printMST(parent, graph);

}

public static void main(String[] args)

{

MST t = new MST();

int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

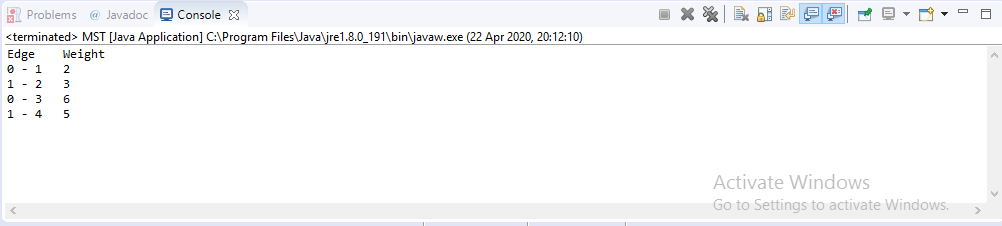
{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

{ 0, 5, 7, 9, 0 } };

t.primMST(graph);

}



KRISKAL ALGORITHM

package com.sorting;

import java.util.\*;

import java.lang.\*;

import java.io.\*;

class Graph

{

// A class to represent a graph edge

class Edge implements Comparable<Edge>

{

int src, dest, weight;

// Comparator function used for sorting edges

// based on their weight

public int compareTo(Edge compareEdge)

{

return this.weight-compareEdge.weight;

}

};

// A class to represent a subset for union-find

class subset

{

int parent, rank;

};

int V, E; // V-> no. of vertices & E->no.of edges

Edge edge[]; // collection of all edges

// Creates a graph with V vertices and E edges

Graph(int v, int e)

{

V = v;

E = e;

edge = new Edge[E];

for (int i=0; i<e; ++i)

edge[i] = new Edge();

}

// A utility function to find set of an element i

// (uses path compression technique)

int find(subset subsets[], int i)

{

// find root and make root as parent of i (path compression)

if (subsets[i].parent != i)

subsets[i].parent = find(subsets, subsets[i].parent);

return subsets[i].parent;

}

// A function that does union of two sets of x and y

// (uses union by rank)

void Union(subset subsets[], int x, int y)

{

int xroot = find(subsets, x);

int yroot = find(subsets, y);

// Attach smaller rank tree under root of high rank tree

// (Union by Rank)

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

else if (subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

// If ranks are same, then make one as root and increment

// its rank by one

else

{

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

// The main function to construct MST using Kruskal's algorithm

void KruskalMST()

{

Edge result[] = new Edge[V]; // Tnis will store the resultant MST

int e = 0; // An index variable, used for result[]

int i = 0; // An index variable, used for sorted edges

for (i=0; i<V; ++i)

result[i] = new Edge();

// Step 1: Sort all the edges in non-decreasing order of their

// weight. If we are not allowed to change the given graph, we

// can create a copy of array of edges

Arrays.sort(edge);

// Allocate memory for creating V ssubsets

subset subsets[] = new subset[V];

for(i=0; i<V; ++i)

subsets[i]=new subset();

// Create V subsets with single elements

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

{

subsets[v].parent = v;

subsets[v].rank = 0;

}

i = 0; // Index used to pick next edge

// Number of edges to be taken is equal to V-1

while (e < V - 1)

{

// Step 2: Pick the smallest edge. And increment

// the index for next iteration

Edge next\_edge = new Edge();

next\_edge = edge[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

// If including this edge does't cause cycle,

// include it in result and increment the index

// of result for next edge

if (x != y)

{

result[e++] = next\_edge;

Union(subsets, x, y);

}

// Else discard the next\_edge

}

// print the contents of result[] to display

// the built MST

System.out.println("Following are the edges in " +

"the constructed MST");

for (i = 0; i < e; ++i)

System.out.println(result[i].src+" -- " +

result[i].dest+" == " + result[i].weight);

}

// Driver Program

public static void main (String[] args)

{

int V = 4;

int E = 5;

Graph graph = new Graph(V, E);

// add edge 0-1

graph.edge[0].src = 0;

graph.edge[0].dest = 1;

graph.edge[0].weight = 10;

// add edge 0-2

graph.edge[1].src = 0;

graph.edge[1].dest = 2;

graph.edge[1].weight = 6;

// add edge 0-3

graph.edge[2].src = 0;

graph.edge[2].dest = 3;

graph.edge[2].weight = 5;

// add edge 1-3

graph.edge[3].src = 1;

graph.edge[3].dest = 3;

graph.edge[3].weight = 15;

// add edge 2-3

graph.edge[4].src = 2;

graph.edge[4].dest = 3;

graph.edge[4].weight = 4;

graph.KruskalMST();

}

}

