**Dijkstra’s algorithm**

**package** javaprogram;

**import** java.util.\*;

**import** java.lang.\*;

**import** java.io.\*;

**public** **class** Dijkstra {

**public** **static** **void** main(String[] args) {

**class** Graph\_Shortest\_Path {

**static** **final** **int** ***num\_Vertices*** = 6;

**int** minDistance(**int** path\_array[], Boolean sptSet[]) {

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

**for** (**int** v = 0; v < ***num\_Vertices***; v++)

**if** (sptSet[v] == **false** && path\_array[v] <= min) {

min = path\_array[v];

min\_index = v;

}

**return** min\_index;

}

**void** printMinpath(**int** path\_array[]) {

System.***out***.println("Vertex# \t Minimum Distance from Source");

**for** (**int** i = 0; i < ***num\_Vertices***; i++)

System.***out***.println(i + " \t\t\t " + path\_array[i]);

}

**void** algo\_dijkstra(**int** graph[][], **int** src\_node) {

**int** path\_array[] = **new** **int**[***num\_Vertices***];

Boolean sptSet[] = **new** Boolean[***num\_Vertices***];

**for** (**int** i = 0; i < ***num\_Vertices***; i++) {

path\_array[i] = Integer.***MAX\_VALUE***;

sptSet[i] = **false**;

}

path\_array[src\_node] = 0;

**for** (**int** count = 0; count < ***num\_Vertices*** - 1; count++) {

**int** u = minDistance(path\_array, sptSet);

sptSet[u] = **true**;

**for** (**int** v = 0; v < ***num\_Vertices***; v++)

**if** (!sptSet[v] && graph[u][v] != 0 && path\_array[u] !=

Integer.***MAX\_VALUE*** && path\_array[u]

+ graph[u][v] < path\_array[v])

path\_array[v] = path\_array[u] + graph[u][v];

}

printMinpath(path\_array);

}

{

**int** graph[][] = **new** **int**[][] { { 0, 2, 1, 0, 0, 0},

{ 2, 0, 7, 0, 8, 4},

{ 1, 7, 0, 7, 0, 3},

{ 0, 0, 7, 0, 8, 4},

{ 0, 8, 0, 8, 0, 5},

{ 0, 4, 3, 4, 5, 0} };

Graph\_Shortest\_Path g = **new** Graph\_Shortest\_Path();

g.algo\_dijkstra(graph, 0);

}

}

}

}

**Prim’s algorithm**

**package** javaprogram;

**import** java.util.\*;

**import** java.lang.\*;

**import** java.io.\*;

**class** MST {

**private** **static** **final** **int** ***V*** = 5;

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

{

**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;

}

**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]]);

}

**void** primMST(**int** graph[][])

{

**int** parent[] = **new** **int**[***V***];

**int** key[] = **new** **int**[***V***];

Boolean mstSet[] = **new** Boolean[***V***];

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

key[i] = Integer.***MAX\_VALUE***;

mstSet[i] = **false**;

}

key[0] = 0;

parent[0] = -1;

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

**int** u = minKey(key, mstSet);

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);

}

}

**Krunskal’s algorithm**

**package** javaprogram;

**import** java.util.\*;

**import** java.io.\*;

**public** **class** Krunskal {

**class** Graph {

**class** Edge **implements** Comparable<Edge>

{

**int** src, dest, weight;

**public** **int** compareTo(Edge compareEdge)

{

**return** **this**.weight - compareEdge.weight;

}

};

**class** subset

{

**int** parent, rank;

};

**int** V, E;

Edge edge[];

Graph(**int** v, **int** e)

{

V = v;

E = e;

edge = **new** Edge[E];

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

edge[i] = **new** Edge();

}

**int** find(subset subsets[], **int** i)

{

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

subsets[i].parent

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

**return** subsets[i].parent;

}

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

{

**int** xroot = find(subsets, x);

**int** yroot = find(subsets, y);

**if** (subsets[xroot].rank

< subsets[yroot].rank)

subsets[xroot].parent = yroot;

**else** **if** (subsets[xroot].rank

> subsets[yroot].rank)

subsets[yroot].parent = xroot;

**else** {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

**void** KruskalMST()

{

Edge result[] = **new** Edge[V];

**int** e = 0;

**int** i = 0;

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

result[i] = **new** Edge();

Arrays.*sort*(edge);

subset subsets[] = **new** subset[V];

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

subsets[i] = **new** subset();

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

{

subsets[v].parent = v;

subsets[v].rank = 0;

}

i = 0;

**while** (e < V - 1)

{

Edge next\_edge = edge[i++];

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

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

**if** (x != y) {

result[e++] = next\_edge;

Union(subsets, x, y);

}

}

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

+ "the constructed MST");

**int** minimumCost = 0;

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

{

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

+ result[i].dest

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

minimumCost += result[i].weight;

}

System.***out***.println("Minimum Cost Spanning Tree "

+ minimumCost);

}

{

**int** V = 4;

**int** E = 5;

Graph graph = **new** Graph(V, E);

graph.edge[0].src = 0;

graph.edge[0].dest = 1;

graph.edge[0].weight = 10;

graph.edge[1].src = 0;

graph.edge[1].dest = 2;

graph.edge[1].weight = 6;

graph.edge[2].src = 0;

graph.edge[2].dest = 3;

graph.edge[2].weight = 5;

graph.edge[3].src = 1;

graph.edge[3].dest = 3;

graph.edge[3].weight = 15;

graph.edge[4].src = 2;

graph.edge[4].dest = 3;

graph.edge[4].weight = 4;

graph.KruskalMST();

}

}

}