import java.util.\*;

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 vertices, edges;

Edge[] edge;

Graph(int v, int e) {

vertices = v;

edges = e;

edge = new Edge[edges];

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 kruskalAlgo() {

Edge result[] = new Edge[vertices];

int minCost = 0;

int e = 0;

int i = 0;

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

result[i] = new Edge();

Arrays.sort(edge);

Subset subsets[] = new Subset[vertices];

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

subsets[i] = new Subset();

for (int v = 0; v < vertices; ++v) {

subsets[v].parent = v;

subsets[v].rank = 0;

}

i = 0;

while (e < vertices - 1) {

Edge nextEdge = edge[i++];

int x = find(subsets, nextEdge.src);

int y = find(subsets, nextEdge.dest);

if (x != y) {

result[e++] = nextEdge;

union(subsets, x, y);

}

}

System.out.println("Edges in the Minimum Spanning Tree (Kruskal's Algorithm):");

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

System.out.println(result[i].src + " - " + result[i].dest + ": " + result[i].weight);

minCost += result[i].weight;

}

System.out.println("Total cost of MST (Kruskal's Algorithm): " + minCost);

}

void primAlgo() {

int[] parent = new int[vertices];

int[] key = new int[vertices];

boolean[] mstSet = new boolean[vertices];

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

key[i] = Integer.MAX\_VALUE;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < vertices; v++) {

if (edgeExists(u, v) && !mstSet[v] && edgeWeight(u, v) < key[v]) {

parent[v] = u;

key[v] = edgeWeight(u, v);

}

}

}

System.out.println("\nEdges in the Minimum Spanning Tree (Prim's Algorithm):");

int minCost = 0;

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

System.out.println(parent[i] + " - " + i + ": " + edgeWeight(parent[i], i));

minCost += edgeWeight(parent[i], i);

}

System.out.println("Total cost of MST (Prim's Algorithm): " + minCost);

}

boolean edgeExists(int u, int v) {

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

if ((edge[i].src == u && edge[i].dest == v) || (edge[i].src == v && edge[i].dest == u)) {

return true;

}

}

return false;

}

int edgeWeight(int u, int v) {

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

if ((edge[i].src == u && edge[i].dest == v) || (edge[i].src == v && edge[i].dest == u)) {

return edge[i].weight;

}

}

return Integer.MAX\_VALUE;

}

int minKey(int[] key, boolean[] mstSet) {

int min = Integer.MAX\_VALUE;

int minIndex = -1;

for (int v = 0; v < vertices; v++) {

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

min = key[v];

minIndex = v;

}

}

return minIndex;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of vertices: ");

int vertices = scanner.nextInt();

System.out.print("Enter the number of edges: ");

int edges = scanner.nextInt();

Graph graph = new Graph(vertices, edges);

System.out.println("Enter edges with their weights:");

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

System.out.print("Enter source vertex of edge " + (i + 1) + ": ");

graph.edge[i].src = scanner.nextInt();

System.out.print("Enter destination vertex of edge " + (i + 1) + ": ");

graph.edge[i].dest = scanner.nextInt();

System.out.print("Enter weight of edge " + (i + 1) + ": ");

graph.edge[i].weight = scanner.nextInt();

}

graph.kruskalAlgo();

graph.primAlgo();

scanner.close();

}

}