KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

DEPARTMENT OF APPLIED INFORMATICS

DISCRETE STRUCTRURES (P170B008)

HOMEWORK

Problem Nr. A02

student : Abdelrhman Ahmed

Presented to: prof. Nečiūnas Audrius

KAUNAS

2022

1. Problem (nr. A02)

Find the minimal spanning tree using the Kruskal’s algorithm.

2. Analysis of the problem

- The graph is connected and has one connected component.
- The graph is unidirectional.
- The graph has V number of vertex so we should be getting spanning graph with number of edges equal to V -1 .

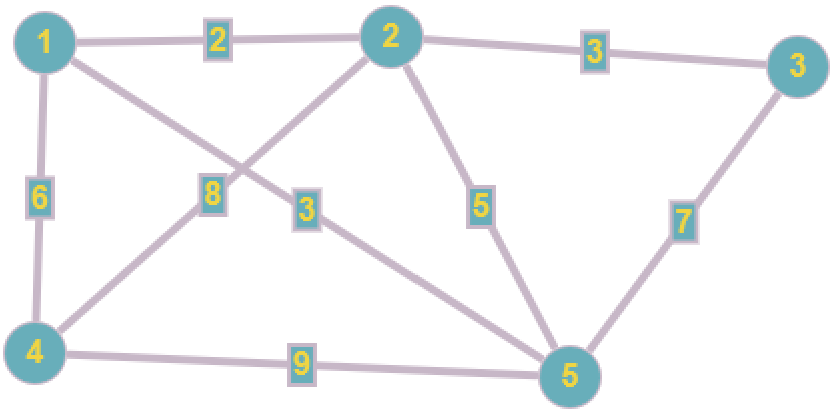
1. We will perform **Find – Union** algorithm.

 Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
 Union: Join two subsets into a single subset. Here first we have to check if the two subsets belong to the same set. If not, then we cannot perform union.

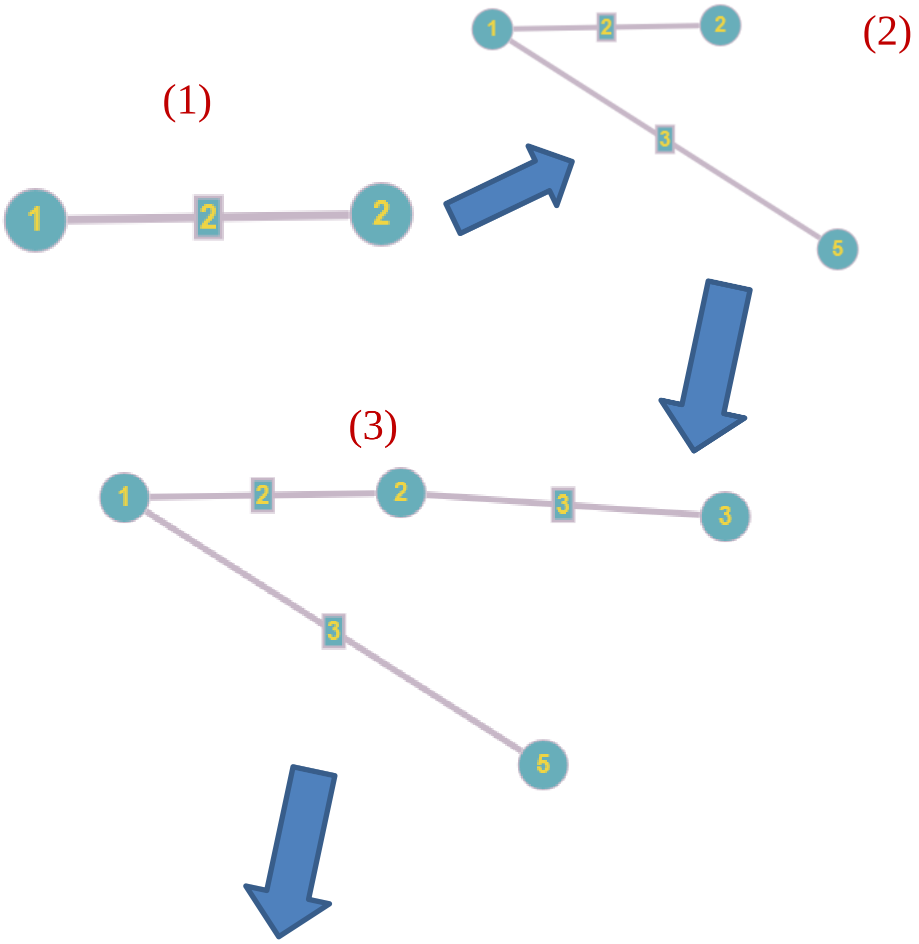
2. Meanwhile we will be running a loop to traverse through the adjacency matrix.
3. Sort all the edges in non-decreasing order of their weight.
4. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
5. Repeat step#4 until there are (V-1) edges in the spanning tree.

    - We will be using array **parent []** that will initially has the value of the vertex with same index
      if **Find (i) == Find(j)** this means that there is a cycle , we will avoid it

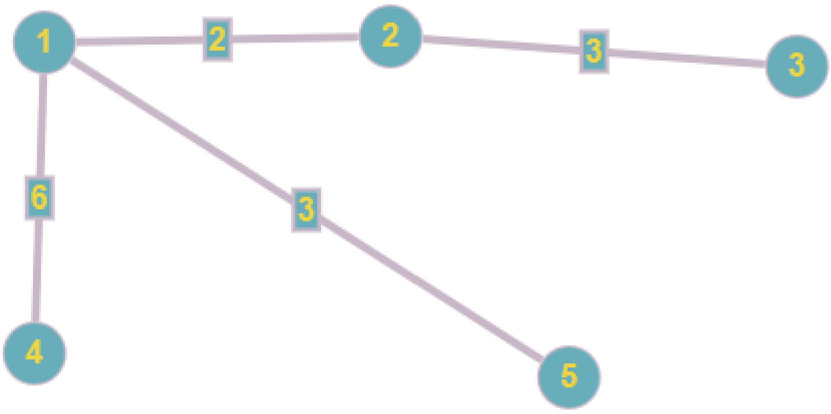
The Graph :



Here the steps procedure after applying Kruskal algorithm:

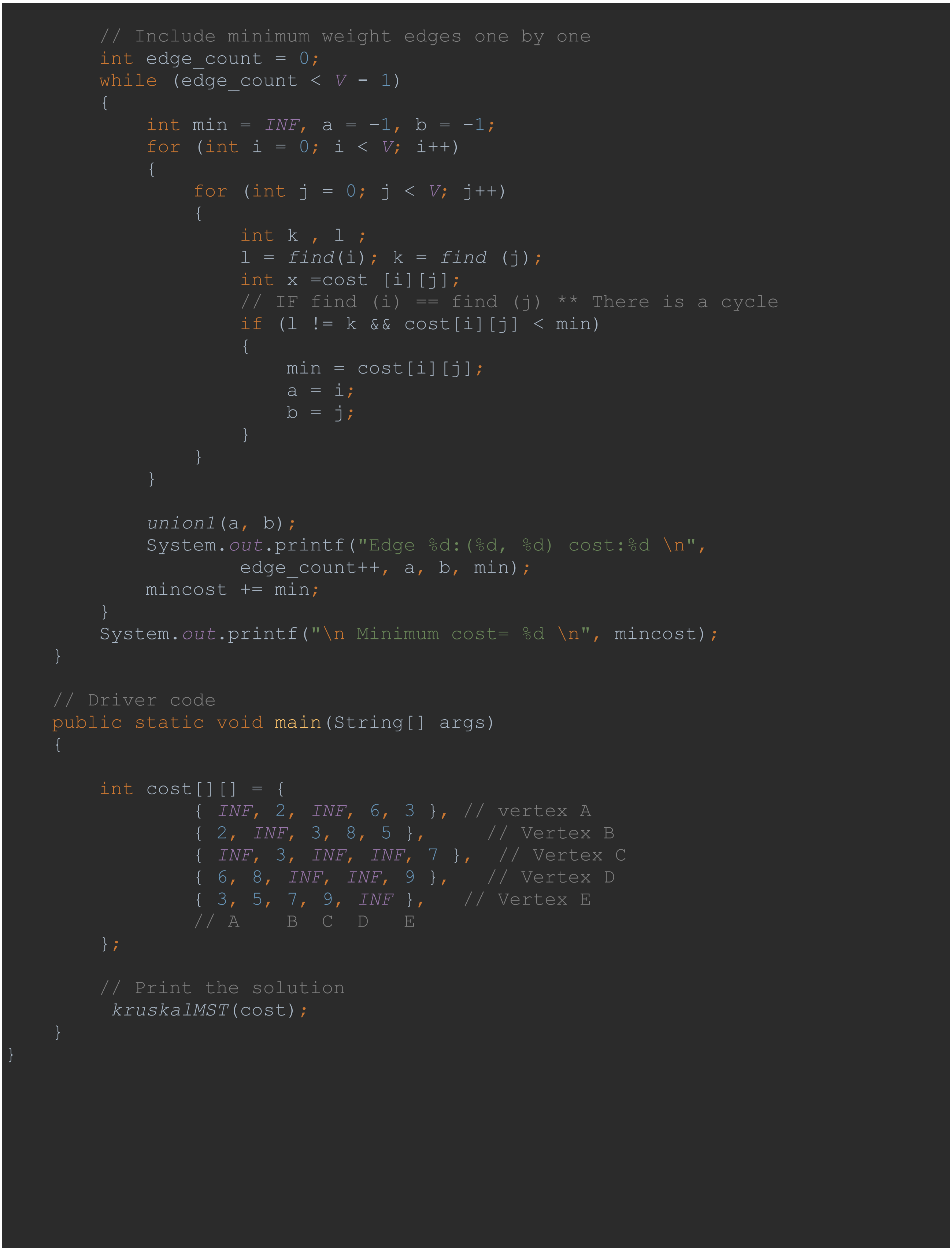


(4)



3. Source code

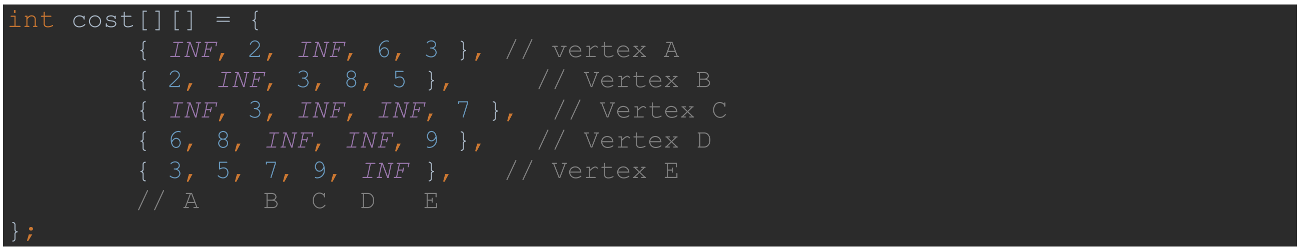
```
import java.util.*;




// Simple Java implementation for Kruskal's
// algorithm
import java.util.*;

class GFG
{

    static int V = 5;       // number of vertex
    static int[] parent = new int[V];   // array
    static int INF = Integer.MAX_VALUE;

    // Find set of vertex n
    // Determine which subset a particular element is in.
    // This can be used for determining if two elements are in the same subset.
    static int find(int n)
    {
        while(parent[n] != n)
        {
            n = parent[n];
        }
        return n;
    }

    // Join two subsets into a single subset.
    // Here first we have to check if the two subsets belong to same set.
    // If no, then we cannot perform union.
    static void union1(int i, int j)
    {
        int a = find(i) ;
        int b = find(j);
        parent[a] = b;
    }

    // Finds MST using Kruskal's algorithm
    static void kruskalMST(int cost[][])
    {
        int mincost = 0; // Cost of min MST.

        // Initialize sets of disjoint sets.
        for (int i = 0; i < V; i++)
            parent[i] = i;
        System.out.println(parent.length);
        int q =0;
        for (int z : parent)
        {
            System.out.println("Index number : " + q + " = "+ z );
            q++;
        }
```



```
        // Include minimum weight edges one by one
        int edge_count = 0;
        while (edge_count < V - 1)
        {
            int min = INF, a = -1, b = -1;
            for (int i = 0; i < V; i++)
            {
                for (int j = 0; j < V; j++)
                {
                    int k , l ;
                    l = find(i); k = find (j);
                    int x =cost [i][j];
                    // IF find (i) == find (j) ** There is a cycle
                    if (l != k && cost[i][j] < min)
                    {
                        min = cost[i][j];
                        a = i;
                        b = j;
                    }
                }
            }

            union1(a, b);
            System.out.printf("Edge %d:(%d, %d) cost:%d \n",
                    edge_count++, a, b, min);
            mincost += min;
        }
        System.out.printf("\n Minimum cost= %d \n", mincost);
    }

    // Driver code
    public static void main(String[] args)
    {

        int cost[][] = {
                { INF, 2, INF, 6, 3 }, // vertex A
                { 2, INF, 3, 8, 5 },     // Vertex B
                { INF, 3, INF, INF, 7 },  // Vertex C
                { 6, 8, INF, INF, 9 },   // Vertex D
                { 3, 5, 7, 9, INF },   // Vertex E
                // A    B  C  D   E
        };

        // Print the solution
         kruskalMST(cost);
    }
}
```

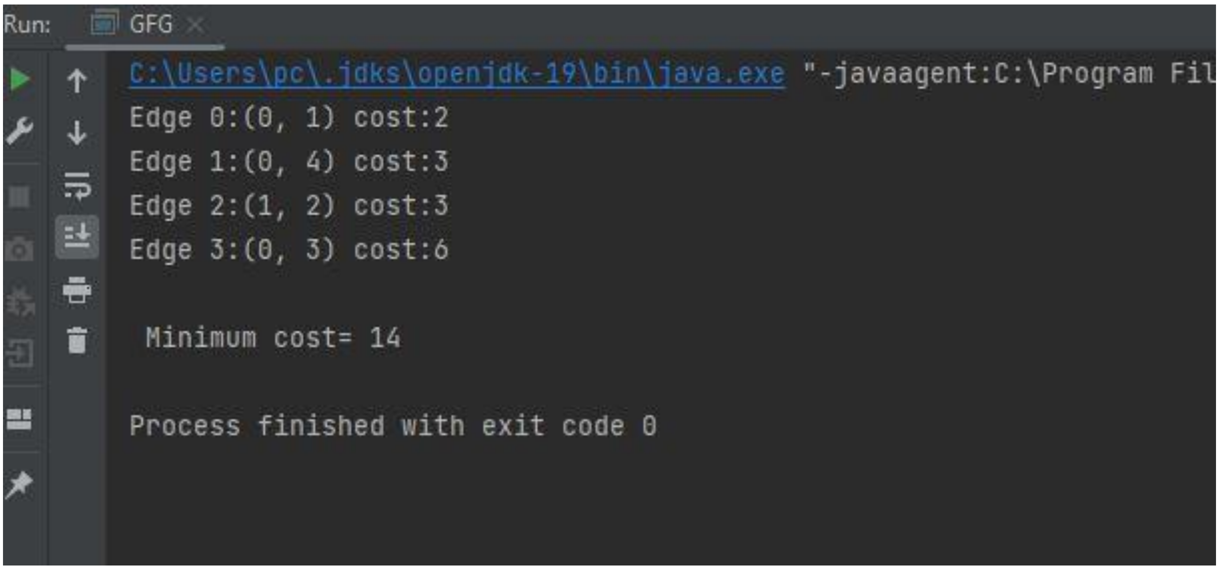
4. Test examples

Lets test the program with an adjacency matrix represented in an 2D – Array the intersection between each column .and row if it has a value this means that the two vertices are connected and the weight of their connecting edges equal the value of intersection



```
int cost[][] = {
        { INF, 2, INF, 6, 3 }, // vertex A
        { 2, INF, 3, 8, 5 },      // Vertex B
        { INF, 3, INF, INF, 7 },   // Vertex C
        { 6, 8, INF, INF, 9 },    // Vertex D
        { 3, 5, 7, 9, INF },    // Vertex E
        // A    B  C  D   E
};
```

The answer we are getting :



```
C:\Users\pc\.jdks\openjdk-19\bin\java.exe "-javaagent:C:\Program Fil
Edge 0:(0, 1) cost:2
Edge 1:(0, 4) cost:3
Edge 2:(1, 2) cost:3
Edge 3:(0, 3) cost:6

 Minimum cost= 14

Process finished with exit code 0
```

5. Conclusions

The answer we are getting after running the code match the theoretical answer with the same sequence of visiting the edges, method find used to detect the subset of the vertex given so that it can avoid making cycle method union run after finding the targeted least edge we add both vertex of this edge to a set of vertex to avoid repeating them and avoid cycles. The first edge to be found is the one connecting between vertex 0 and 1 and it has value of 2 after the third edge that is connecting between vertex 1 and 2 we didn’t choose the edge connecting vertex 1 and 4 although it has value of 5 which is less than 6 because it would make a cycle we skipped it.

6. References

1. Find – union algorithm https://aquarchitect.github.io/swift-algorithm-club/Union-Find/
2. Minimal spanning tree https://moodle.ktu.edu/course/view.php?id=42
3. Kruskal algorithm https://www.programiz.com/dsa/kruskal-algorithm