SOLUTION 1:

```
/*
    This program finds the minimum number of cables required to connect all workstations in a network.
    It uses the disjoint set data structure to store the connections between workstations.
    A union-find algorithm is used to find the root of each workstation and to determine if two workstations are connected.
    If two workstations are not connected, then a cable is added to connect them.
    The number of cables added is counted and returned as the minimum number of cables needed.
*/

#include <stdio.h>
#include <stdlib.h>

// Function to find the root of a given node
int find(int node, int parent[node])
{
    if (parent[node] == node)
        return node;
    return find(parent[node], parent);
}

// Function to connect two nodes
void union_op(int node, int a, int b, int parent[node])
{
    int a_set = find(a, parent);
    int b_set = find(b, parent);
    parent[a_set] = b_set;
}

// Function to count the number of cables required
int count_cables(int n, int connections[][2], int k)
{
    int parent[n];

    // Initialize parent array
    for (int i=0; i<n; i++)
        parent[i] = i;

    int cable_count = 0;

    // Iterate over all connections
    for (int i=0; i<k; i++)
    {
        int a = connections[i][0];
        int b = connections[i][1];

        // Check if the two nodes are already connected
        if (find(a, parent) != find(b, parent))
        {
            // If not, connect them and increment cable count
            union_op(n, a, b, parent);
            cable_count++;
        }
    }
    return cable_count;
}

// Driver function
int main()
{
    int n = 4, k = 3;
    int connections[][2] = {{0, 1}, {0, 2}, {1, 2}};
    printf("Minimum number of cables required to connect all workstations: %d",
        count_cables(n, connections, k));
    return 0;
}
```

SOLUTION 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int min_cables(int n, int** connections, int connectionsSize, int* connectionsColSize);

int main(int argc, char** argv)
{
    int n = 4;
    int connectionsSize = 3;
    int connectionsColSize = 2;
    int** connections =(int*)malloc(connectionsSize * sizeof(int));
    for (int i = 0; i < connectionsSize; i++) {
        connections[i] = (int*)malloc(connectionsColSize * sizeof(int));
    }
    connections[0][0] = 0; connections[0][1] = 1;
    connections[1][0] = 0; connections[1][1] = 2;
    connections[2][0] = 1; connections[2][1] = 2;

    printf("%d\n", min_cables(n, connections, connectionsSize, connectionsColSize));

    for (int i = 0; i < connectionsSize; i++) {
        free(connections[i]);
    }
    free(connections);
    return 0;
}

int min_cables(int n, int** connections, int connectionsSize, int* connectionsColSize) {
    if (n < 2) {
        return 0;
    }
    if (connectionsSize == 0) {
        return -1;
    }
    int* arr = (int*)calloc(n, sizeof(int));
    int count = 0;
    for (int i = 0; i < connectionsSize; i++) {
        int a = connections[i][0];
        int b = connections[i][1];
        if (arr[a] == 0 && arr[b] == 0) {
            arr[a] = ++count;
            arr[b] = count;
        }
        else if (arr[a] != 0 && arr[b] == 0) {
            arr[b] = arr[a];
        }
        else if (arr[b] != 0 && arr[a] == 0) {
            arr[a] = arr[b];
        }
        else if (arr[a] != arr[b]) {
            int t = arr[b];
            for (int j = 0; j < n; j++) {
                if (arr[j] == t) {
                    arr[j] = arr[a];
                }
            }
        }
    }
    int ret = 0;
    for (int i = 0; i < n; i++) {
        if (arr[i] == 0) {
            ret++;
        }
    }
    free(arr);
    return ret - 1;
}
```