```
/*
	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;
}
```