**Program No:** 01

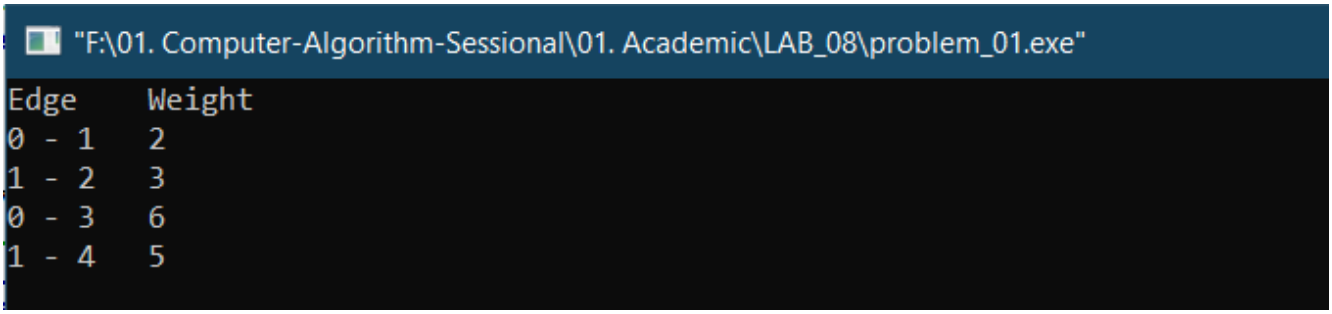
**Program statement:** Prim's Algorithm

**Program Code:**

```
#include <iostream>
#include <limits.h>
using namespace std;

#define V 5

int minKey(int key[], bool mstSet[]) {
    int min = INT_MAX, min_index;
    for (int v = 0; v < V; v++)
        if (!mstSet[v] && key[v] < min)
            min = key[v], min_index = v;
    return min_index;
}

void printMST(int parent[], int graph[V][V]) {
    cout << "Edge\tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] << "\n";
}

void primMST(int graph[V][V]) {
    int parent[V], key[V];
    bool mstSet[V];

    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, 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] && !mstSet[v] && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    printMST(parent, graph);
}

int main() {
```

```
    int graph[V][V] = {
        {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}
    };

    primMST(graph);
    return 0;
}
```

**Program Output:**




**Program No:** 02

**Program statement:** Kruskal's Algorithm

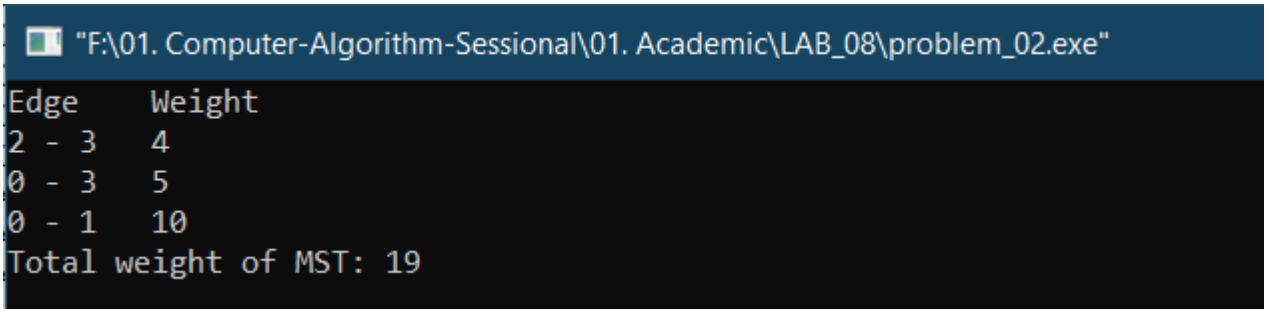
**Program Code:**

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};
```

```
int find(int v, vector<int>& parent) {
    if (parent[v] == v)
        return v;
    return parent[v] = find(parent[v], parent);
}

void union_sets(int a, int b, vector<int>& parent, vector<int>& rank) {
    a = find(a, parent);
    b = find(b, parent);
    if (a != b) {
        if (rank[a] < rank[b])
            swap(a, b);
        parent[b] = a;
        if (rank[a] == rank[b])
            rank[a]++;
    }
}

int main() {
    int V = 4;
    vector<Edge> edges = {
        {0, 1, 10},
        {0, 2, 6},
        {0, 3, 5},
        {1, 3, 15},
        {2, 3, 4}
    };

    sort(edges.begin(), edges.end());

    vector<int> parent(V), rank(V, 0);
    for (int i = 0; i < V; i++)
        parent[i] = i;

    vector<Edge> result;
    for (Edge e : edges) {
        if (find(e.u, parent) != find(e.v, parent)) {
            result.push_back(e);
            union_sets(e.u, e.v, parent, rank);
        }
    }

    cout << "Edge \tWeight\n";
    int total = 0;
    for (Edge e : result) {
        cout << e.u << " - " << e.v << " \t" << e.weight << "\n";
        total += e.weight;
    }
```

```
    cout << "Total weight of MST: " << total << endl;
    return 0;
}
```

**Program Output:**




**Program No:** 03

**Program statement:** Dijkstra's Algorithm

**Program Code:**

```
#include <iostream>
#include <vector>
#include <queue>
#include <climits>
using namespace std;

typedef pair<int, int> pii;

void dijkstra(int V, vector<pii> adj[], int src) {
    vector<int> dist(V, INT_MAX);
    dist[src] = 0;
    priority_queue<pii, vector<pii>, greater<pii>> pq;
    pq.push({0, src});
    while (!pq.empty()) {
        int u = pq.top().second;
        pq.pop();
        for (auto& edge : adj[u]) {
            int v = edge.first;
            int weight = edge.second;
            if (dist[v] > dist[u] + weight) {
```

```
                dist[v] = dist[u] + weight;
                pq.push({dist[v], v});
            }
        }
    }
    cout << "Vertex\tDistance from Source\n";
    for (int i = 0; i < V; ++i)
        cout << i << "\t" << dist[i] << "\n";
}

int main() {
    int V = 5;
    vector<pii> adj[V];
    adj[0].push_back({1, 10});
    adj[0].push_back({4, 5});
    adj[1].push_back({2, 1});
    adj[1].push_back({4, 2});
    adj[2].push_back({3, 4});
    adj[3].push_back({2, 6});
    adj[3].push_back({0, 7});
    adj[4].push_back({1, 3});
    adj[4].push_back({2, 9});
    adj[4].push_back({3, 2});
    dijkstra(V, adj, 0);
    return 0;
}
```

**Program Output:**