DAA Assignment 4

Name: Vishal Sule

Roll no.: 234

PRN: 0120190064

Problem statement: Design & Implement Prims Algorithm using Greedy Approach. Calculate the time complexity of the algorithm

Code:

```
#include<iostream>
#include <bits/stdc++.h>
#include <algorithm>
#include <chrono>

using namespace std;
using namespace std::chrono;

class disjointset
{
        public:
          int djset[20];
          disjointset(int v)
          {
            for (int i = 0; i <= v; i++)
```

```
    {
        djset[i] = i;
    }
}
int find_root(int v)
{
    while (v != djset[v])
    {
        v = djset[v];
    }
    return v;
}
void take_union(int v1, int v2)
{
    int r1 = find_root(v1);
    int r2 = find_root(v2);
    if (v1 == r1 && v2 == r2)
    {
        djset[v1] = v2;
    }
    else if (v1 != r1 && v2 == r2)
    {
        djset[v2] = v1;
    }
    else if (v1 == r1 && v2 != r2)
    {
```

```
                djset[v1] = v2;
            }
            else if (v1 != r1 && v2 != r2)
            {
                djset[r1] = r2;
            }
        }
};

class edge
{
    public:
        int  v1;
        int  v2;
        int wt;
};

class prims_graph
{
    public:
        int v_p;
        int data[20][20];
        prims_graph(int vt)
        {
            v_p = vt;
        }
```

```
        void prims_algorithm();
};

void prims_graph::prims_algorithm()
{
    int sv;
    int visited[v_p];
    int i, j, k;
    edge ed[20];
    edge mst[20];
    edge discarded_edge[20];
    int mst_ctr = 0;
    int edge_ctr = 0;
    int mst_flag = 0;
    int discarded_ctr = 0;
    int discarded_flag = 0;
    disjointset d(v_p);
    for (i = 0; i < v_p; i++)
    {
        visited[i] = 0;
    }
    cout << "\n Enter the start vertex : ";
    cin >> sv;
    sv = sv - 1;
    visited[sv] = 1;
    while (1)
```

```
    {
        int visited_flag = 0;
        for (i = 0; i < v_p; i++)
        {
            if (visited[i] == 1)
                visited_flag++;
        }
        if (visited_flag == v_p)
            break;
        edge_ctr = 0;
        for (i = 0; i < v_p; i++)
        {
            if (visited[i] == 1)
            {
                for (j = 0; j < v_p; j++)
                {
                    if (data[i][j] != 999)
                    {
                        // before adding that edge in ed array check whether it has
already
                        // been added in MST array
                        mst_flag = 0;
                        for (int k = 0; k < mst_ctr; k++)
                        {
                            if ((mst[k].v1 == i && mst[k].v2 == j) || (mst[k].v1 == j &&
mst[k].v2 == i))
                            {
```

```
                    mst_flag = 1;
                    break;
                }
            }
            discarded_flag = 0;
            for (int k = 0; k < discarded_ctr; k++)
            {
                if ((discarded_edge[k].v1 == i && discarded_edge[k].v2 == j) ||
(discarded_edge[k].v1 == j && discarded_edge[k].v2 == i))
                {
                    discarded_flag = 1;
                    break;
                }
            }
            // edge is not in MST array and is not present in discarded array
            if (mst_flag == 0 && discarded_flag == 0)
            {
                ed[edge_ctr].v1 = i;
                ed[edge_ctr].v2 = j;
                ed[edge_ctr].wt = data[i][j];
                edge_ctr++;
            }
          }
        }
      }
    }
    edge min_edge;
```

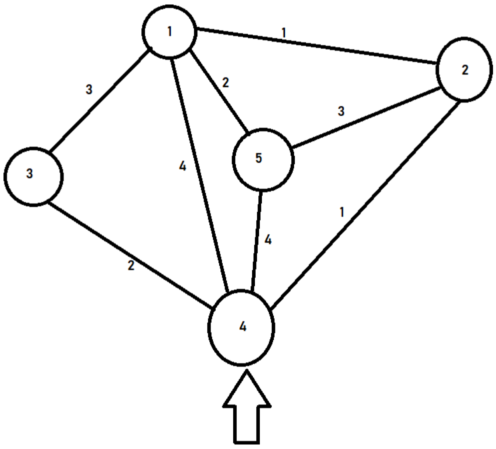
```
min_edge.v1 = 0;
min_edge.v2 = 0;
min_edge.wt = 999;
for (k = 0; k < edge_ctr; k++)
{
   if (ed[k].wt < min_edge.wt)
   {
      min_edge.v1 = ed[k].v1;
      min_edge.v2 = ed[k].v2;
      min_edge.wt = ed[k].wt;
   }
}
// we will get min wt edge in min_edge variable
int r1 = d.find_root(min_edge.v1);
int r2 = d.find_root(min_edge.v2);
if (r1 != r2)
{
   mst[mst_ctr].v1 = min_edge.v1;
   mst[mst_ctr].v2 = min_edge.v2;
   mst[mst_ctr].wt = min_edge.wt;
   mst_ctr++;
   d.take_union(min_edge.v1, min_edge.v2);
   visited[min_edge.v1] = 1;
   visited[min_edge.v2] = 1;
}
else // including the edge in MST will create a cycle so discard it
```

```
        {
            discarded_edge[discarded_ctr].v1 = min_edge.v1;
            discarded_edge[discarded_ctr].v2 = min_edge.v2;
            discarded_edge[discarded_ctr].wt = min_edge.wt;
            discarded_ctr++;
        }
    }
    int sum = 0;
    cout << "\n MST is: ";
    for (i = 0; i < mst_ctr; i++)
    {
        cout << endl
             << " " << mst[i].v1 + 1 << " to " << mst[i].v2 + 1 << " ==> " << mst[i].wt;
        sum = sum + mst[i].wt;
    }
    cout << "\nTotal is " << sum;
}

class kruskal_graph
{
        public:
            int v_k;
            int e;
            edge ed[20];
            kruskal_graph(int vertices, int edges)
            {
```

```
            v_k = vertices;
            e = edges;
        }
        void accept_graph();
        void display_graph();
        void kruskal_mst();
        void sort_edges();
};


int main()
{
    int ch = 1;
    int v_k, e;
    int v_p;

    cout << "------>>> Prims Algorithm <<<----- " << endl;
    cout << "\n Enter the number of vertices in the graph: ";
    cin >> v_p;
    prims_graph g1(v_p);
    for (int i = 0; i < v_p; i++)
        g1.data[i][i] = 999;
    for (int i = 0; i < v_p; i++)
    {
        for (int j = i + 1; j < v_p; j++)
        {
```

```
            cout << "\n Enter the cost of edge between " << i + 1 << " to " << j + 1 <<
" : ";
            cin >> g1.data[i][j];
            g1.data[j][i] = g1.data[i][j];
        }
    }
    auto start = high_resolution_clock::now();

           g1.prims_algorithm();

           auto stop = high_resolution_clock::now();
           auto duration = duration_cast<nanoseconds>(stop - start);
           cout << "\nTime taken by function: "<< duration.count() << "
nanoseconds";



    return 0;
}
```

Graph :



Output:

```
------>>> Prims Algorithm <<<------

 Enter the number of vertices in the graph: 5

 Enter the cost of edge between 1 to 2 : 1

 Enter the cost of edge between 1 to 3 : 3

 Enter the cost of edge between 1 to 4 : 4

 Enter the cost of edge between 1 to 5 : 2

 Enter the cost of edge between 2 to 3 : 999

 Enter the cost of edge between 2 to 4 : 1

 Enter the cost of edge between 2 to 5 : 3

 Enter the cost of edge between 3 to 4 : 2

 Enter the cost of edge between 3 to 5 : 999

 Enter the cost of edge between 4 to 5 : 4

 Enter the start vertex : 4

 MST is:
 4 to 2 ==> 1
 2 to 1 ==> 1
 1 to 5 ==> 2
 4 to 3 ==> 2
Total is 6
Time taken by function: 2300200114 nanoseconds

...Program finished with exit code 0
Press ENTER to exit console.
```