DAA Assignment 5

**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)

{

already

// before adding that edge in ed array check whether it has

// been added in MST array mst\_flag = 0;

for (int k = 0; k < mst\_ctr; k++)

{

mst[k].v2 == i))

if ((mst[k].v1 == i && mst[k].v2 == j) || (mst[k].v1 == j &&

{

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();

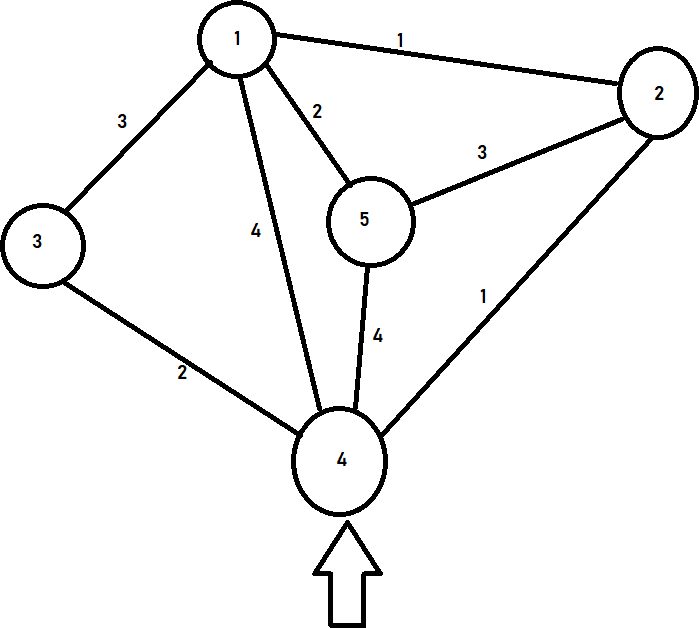
auto duration = duration\_cast<nanoseconds>(stop - start); cout << "\nTime taken by function: "<< duration.count() << "

nanoseconds";

return 0;

}

Graph :



**Output:**

