## **Write a code for Dijkstra or A\* path finding algorithms (any one). What are the pros and cons of each? (C++ preferred)**

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

Code –

#include <bits/stdc++.h>

using namespace std;

const int inf = 1e7;

int main()

{

    int n, m;

    cout << "Enter number of Vertex: ";

    cin >> n;

    cout << "\nEnter number of Edges: ";

    cin >> m;

    vector<int> distance(n + 1, inf);

    vector<vector<pair<int, int>>> graph(n + 1);

    cout << "\nEnter all the nodes and weight between two nodes: format(u, v, w)" << endl;

    for (int i = 0; i < m; i++)

    {

        int u, v, w;

        cin >> u >> v >> w;

        graph[u].push\_back({v, w});

        graph[v].push\_back({u, w});

    }

int source;

    cout << "\nEnter the source node: ";

    cin >> source;

    distance[source] = 0;

    set<pair<int, int>> s;

s.insert({0, source});

    while (!s.empty())

    {

        auto x = \*(s.begin());

        s.erase(x);

        for (auto it : graph[x.second])

        {

            if (distance[it.first] > distance[x.second] + it.second)

            {

                s.erase({distance[it.first], it.first});

                distance[it.first] = distance[x.second] + it.second;

                s.insert({distance[it.first], it.first});

            }

        }

    }

    for (int i = 1; i <= n; i++)

    {

        if (distance[i] < inf)

        {

            cout << "Shortest Distance from node " << source << " to destination node " << i << " : " << distance[i] << endl;

        }

        else

        {

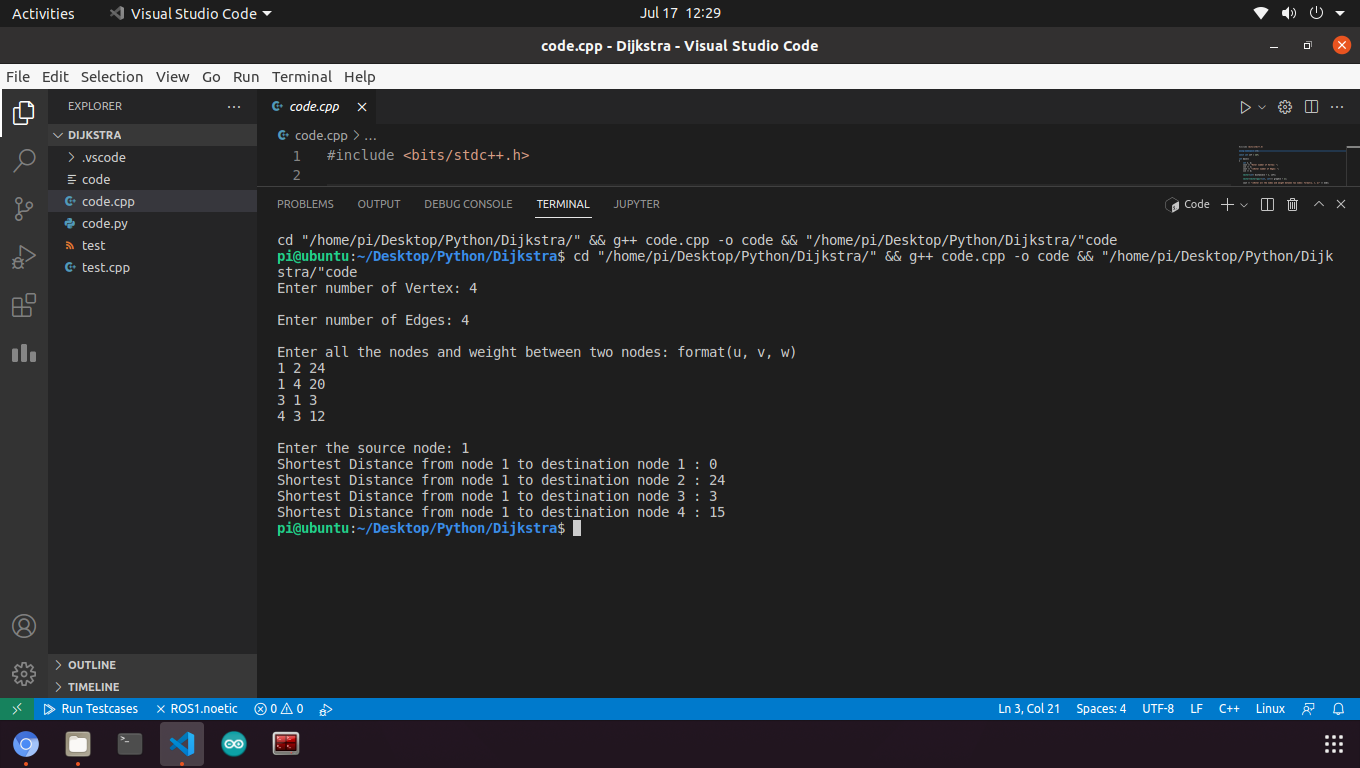
            cout << -1 << " ";

        }

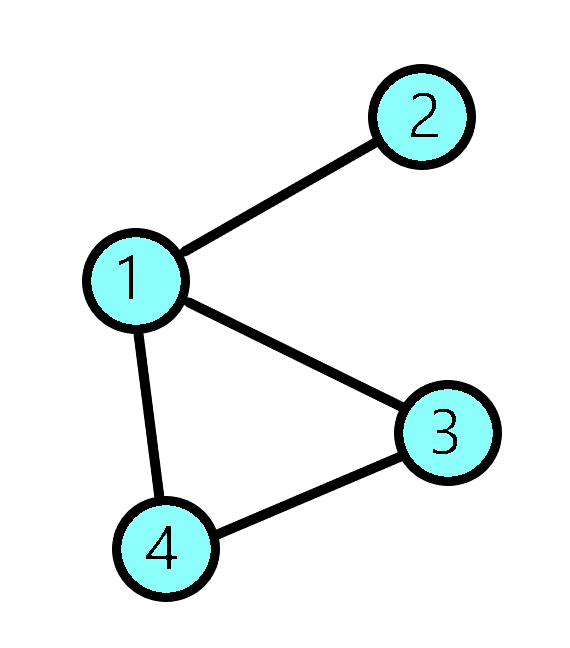
    }

}

Input/Output of code –



Explanation of working of the code –



infinity

Source

infinity

infinity

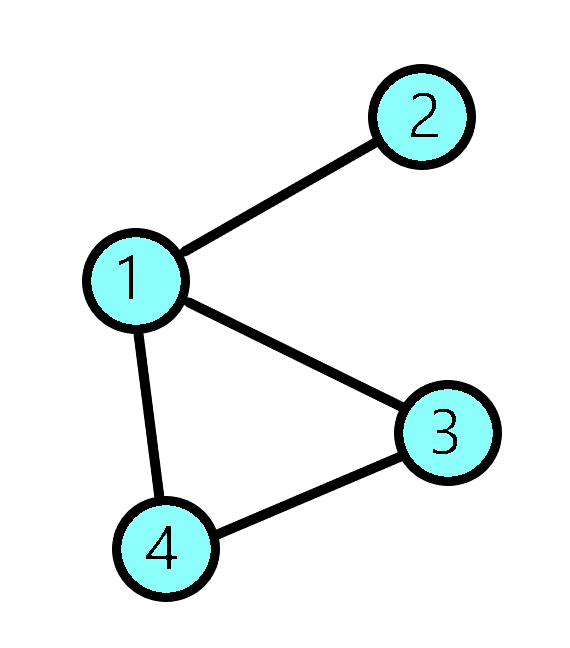
0

Distance from node 1 to 2: 24

Distance or cost from node 1 to 4: 20

Distance or cost from node 3 to 1: 3

Distance or cost from node 4 to 1: 12



12

3

20

24

infinity

Source

infinity

infinity

0

Edge relaxation process

Suppose u = 1, v = 2, d(u, v) = 24

d= distance

d(u) = 0; distance from source to source is 0.

Initial d(v) = inf;

So, if d(v) > d(u, v) + d(u) : then

d(v) = d(u, v) + d(u)

example –

if d(2) > d(1, 2) + d(1) then

d(2) = d(1, 2) + d(1) 🡺 d(2) = 24 + 0 = 24

Using this relaxation process we find all the shortest distances:

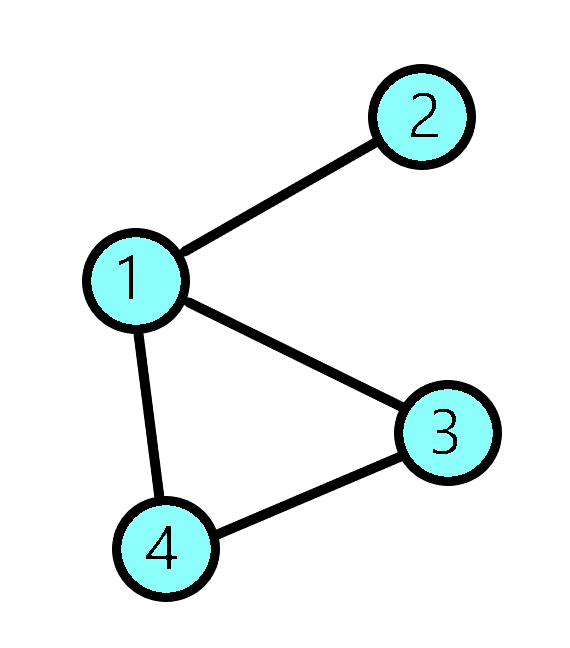
So, our final result will be this 🡺

Shortest distance from node 1 to 1: 0

Shortest distance from node 1 to 2: 24

Shortest distance from node 1 to 3: 3

Shortest distance from node 1 to 4: 15



12

3

20

24

15

Source

3

24

0

Pros and Cons on the next page:

Pros and Cons of Dijkstra Algorithm:

Pros

1. Uninformed algorithm

Dijkstra is an uninformed algorithm. This means that it does not need to know the target node beforehand. For this reason, it's optimal in cases where you don't have any prior knowledge of the graph when you cannot estimate the distance between each node and the target.

1. Good when you have multiple target nodes

Since Dijkstra picks edges with the smallest cost at each step it usually covers a large area of the graph. This is especially useful when you have multiple target nodes but you don't know which one is the closest.

CON

1. Fails for negative edge weights

If we take for example 3 Nodes (A, B and C) where they form an undirected graph with edges: AB = 3, AC = 4, BC=-2, the optimal path from A to C costs 1 and the optimal path from A to B costs 2. If we apply Dijkstra's algorithm: starting from A it will first examine B because it is the closest node. and will assign a cost of 3 to it and therefore mark it closed which means that its cost will never be re-evaluated. This means that Dijkstra's cannot evaluate negative edge weights.

Pros and Cons of A\* Algorithm:

Pros

1. It is optimal search algorithm in terms of heuristics.
2. It is one of the best heuristic search techniques.
3. It is used to solve complex search problems.
4. There is no other optimal algorithm guaranteed to expand fewer nodes than A\*.

Cons

1. This algorithm is complete if the branching factor is finite and every action has fixed cost.
2. The performance of A\* search is dependent on accuracy of heuristic algorithm used to compute the function h(n).