LEARN DSA WITH C++

WEEK :: 13

LEARN DSA
WITH C++

COURSE INSTRUCTOR BY
ROHIT NEGI

Check Profile

MADE BY-
PRADUM SINGHA

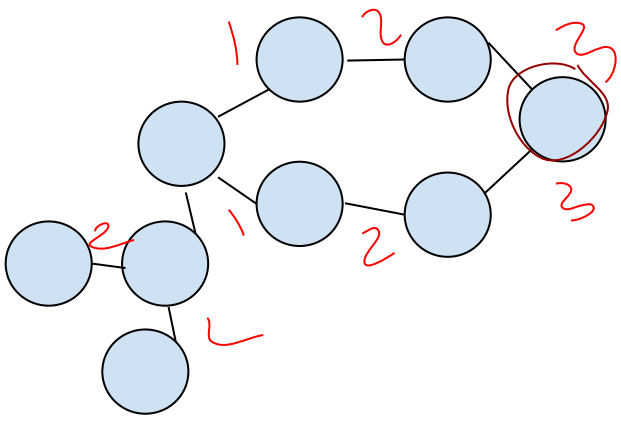
My profile

LEARN DSA WITH C++

WEEK :: 13 **DAY**: 01 **DATE**: 17-07-2023

DETECT CYCLE IN GRAPH

Detect Cycle in a undirected graph :- BFS



1. Start walking all nodes 1 step from the roots.
2. When they meet each other , it is more possible to cycle.

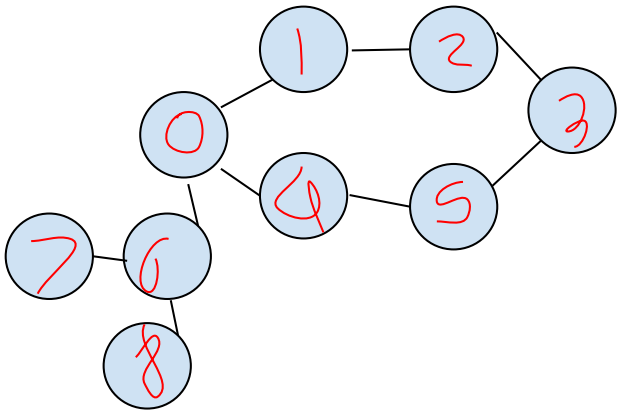
{ Dot back return to root }

Rolues :

Already visited

Ignore parent

How it Works :-



Initially all 0

0	1	2	3	4	5
0	0	0	0	0	0
1	1	0	0	1	0
1	1	1	0	1	1
1	1	1	1	1	1

3 already 1 : so it is possible to cycle

Detect cycle in an undirected graph << GeeksforGeeks >>

```
class Solution {
  public:
    // Function to detect cycle in an undirected graph.
    bool BFS(int node, vector<int> adj[], vector<bool> &visit)
    {
        queue<pair<int,int>>q;
        q.push({node, -1});
        visit[node] =1;
        while(!q.empty())
        {
            int child = q.front().first;
            int parent = q.front().second;
```

```
            q.pop();

            for(int i=0; i<adj[child].size(); i++)
            {
                // Adjacent node is not visited
                // visited node make 1
                if(visit[adj[child][i]] ==0)
                {
                    if(visit[adj[child][i]] =1);
                    q.push({adj[child][i], child});
                }
            // adjacent node already visited
            // ignore the parent node
            // Cycle is preset
            else
               {
                  if(adj[child][i] != parent)
                   return 1;
               }
            }
        }
    }
    bool isCycle(int V, vector<int> adj[]) {
        // Code here

        vector<bool>visit(V, 0);

        for(int i=0; i<V; i++)
        {
            if(!visit[i])
            {
                bool ans = BFS(i, adj, visit);
                if(ans == 1)
                return 1;
            }
        }
        return 0;
    }
};
```

Rotten Oranges << GeeksforGeeks >>

```
class Solution {
public:
    // i, j present in grid
    bool check(int i, int j, int row, int col)
    {
        return i > -1 && i < row && j > -1 && j < col;
    }

    //Function to find minimum time required to rot all oranges.
    int orangesRotting(vector<vector<int>>& grid)
    {
        int n = grid.size();  // row
        int m = grid[0].size(); // col
        int row[4] = {-1, 1, 0, 0};
        int col[4] = {0, 0, 1, -1};
```

```
        int GoodOranges = 0;
        // 3 things, row, col, timer
        queue<pair<pair<int, int>, int>> q;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++)
            {
                // count good orange
                if (grid[i][j] == 1)
                    GoodOranges++;
                //Push Rotten Oranges in queue
                else if (grid[i][j] == 2)
                {
                    q.push({{i, j}, 0});
                }
            }
        }
        int i, j, timer = 0;
        while (!q.empty())
        {
            i = q.front().first.first;
            j = q.front().first.second;
            timer = q.front().second;
            q.pop();

            for(int k=0; k<4; k++)
            {
                if(check(i+row[k], j+col[k],n,m) && grid[i+row[k]][j+col[k]] ==1)
                {
                    grid[i+row[k]][j+col[k]]=0;
                    GoodOranges--;
                    q.push({{i+row[k], j+col[k]}, timer+1});
                }
            }
        }

        if (GoodOranges)
            return -1;
        else
            return timer;
    }
};
```

LEARN DSA WITH C++

WEEK :: 13 **DAY**: 02 **DATE**: 18-07-2023

BFS IN ADVANCE USING GRAPH

Find the number of islands << GeeksforGeeks >>

```
class Solution {
```

```
public:
    // Function to find the number of islands.
    bool check(int i, int j, int row, int col) {
        return i > -1 && i < row && j > -1 && j < col;
    }

    void BFS(vector<vector<char>>& grid, int i, int j) {
        int row[8] = {1, 1, 1, -1, -1, -1, 0, 0};
        int col[8] = {-1, 0, 1, -1, 0, 1, -1, 1};

        int n = grid.size();
        int m = grid[0].size();
        queue<pair<int, int>> q;
        q.push({i, j});

        while (!q.empty()) {
            i = q.front().first;
            j = q.front().second;
            q.pop();

            for (int k = 0; k < 8; k++) {
                if (check(i+row[k], j+col[k],n,m) && grid[i+row[k]][j+col[k]] ==
'1') {
                    grid[i + row[k]][j + col[k]] = '0';
                    q.push({i + row[k], j + col[k]});
                }
            }
        }
    }

    int numIslands(vector<vector<char>>& grid) {
        // Code here

        int n = grid.size();
        int m = grid[0].size();
        int count = 0; // Count Number of Island
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == '1') {
                    grid[i][j] = '0';
                    count++;
                    BFS(grid, i, j);
                }
            }
        }
        return count;
    }

};
```

Steps by Knight << GeeksforGeeks >>

```
class Solution
{
public:
    // Corrected check function to validate if a position is within the board
boundaries.
```

```
    bool check(int i, int j, int n)
    {
        return i > 0 && j > 0 && i <= n && j <= n;
    }

    //Function to find out minimum steps Knight needs to reach the target position.
    int minStepToReachTarget(vector<int>& KnightPos, vector<int>& TargetPos, int N)
    {
        // Handling the edge case where Knight and Target positions are the same.
        if (KnightPos[0] == TargetPos[0] && KnightPos[1] == TargetPos[1])
            return 0;

        // Code here
        int row[8] = {2, 2, -2, -2, 1, 1, -1, -1};
        int col[8] = {1, -1, 1, -1, 2, -2, 2, -2};

        // row, col, here
        queue<pair<pair<int, int>, int>> q;
        q.push({{KnightPos[0], KnightPos[1]}, 0});

        // don't repeat the same way which is already visited
        vector<vector<bool>> visit(N + 1, vector<bool>(N + 1, false));
        visit[KnightPos[0]][KnightPos[1]] = true;

        int i, j, step;
        while (!q.empty())
        {
            i = q.front().first.first;
            j = q.front().first.second;
            step = q.front().second;
            if (i == TargetPos[0] && j == TargetPos[1])
                return step;
            q.pop();

            for (int k = 0; k < 8; k++)
            {
                if (check(i + row[k], j + col[k], N) && !visit[i + row[k]][j +
col[k]])
                {
                    visit[i + row[k]][j + col[k]] = true;
                    q.push({{i + row[k], j + col[k]}, step + 1});
                }
            }
        }
        return -1;
    }
};
```

Replace O's with X's << GeeksforGeeks >>

```
class Solution {
public:
    bool check(int i, int j, int n, int m)
    {
        return i >- 1 && j >-1 && i < n && j < m;
    }
```

```
    vector<vector<char>> fill(int n, int m, vector<vector<char>>& mat)
    {
        int row[4] = {1, -1, 0, 0};
        int col[4] = {0, 0, 1, -1};

        queue<pair<int, int>> q;
        vector<vector<char>> ans(n, vector<char>(m, 'X'));
        vector<vector<bool>> visit(n, vector<bool>(m, 0));

        for (int j = 0; j < m; j++)
        {
            if (mat[0][j] == 'O')
            {
                q.push({0, j});
                ans[0][j] = 'O'; // Replace 'O' with 'X'
                visit[0][j] = 1;
            }
        }

        for (int j = 0; j < m; j++)
        {
            if (mat[n - 1][j] == 'O')
            {
                q.push({n - 1, j});
                ans[n - 1][j] = 'O'; // Replace 'O' with 'X'
                visit[n - 1][j] = 1;
            }
        }

        for (int i = 1; i < n - 1; i++)
        {
            if (mat[i][0] == 'O')
            {
                q.push({i, 0});
                ans[i][0] = 'O'; // Replace 'O' with 'X'
                visit[i][0] = 1;
            }
        }

        for (int i = 1; i < n - 1; i++)
        {
            if (mat[i][m - 1] == 'O')
            {
                q.push({i, m - 1});
                ans[i][m - 1] = 'O'; // Replace 'O' with 'X'
                visit[i][m - 1] = 1;
            }
        }

        int u, v;
        while (!q.empty())
        {
            u = q.front().first;
            v = q.front().second;
            q.pop();

            for (int k = 0; k < 4; k++) {
                if (check(u + row[k], v + col[k], n, m) && !visit[u + row[k]][v +
col[k]])
```

```
                {
                    visit[u + row[k]][v + col[k]] = 1;
                    if (mat[u + row[k]][v + col[k]] == 'O')
                    {
                        ans[u + row[k]][v + col[k]] = 'O'; // Replace 'O' with 'X'
                        q.push({u + row[k], v + col[k]});
                    }
                }
            }
        }

        return ans;
    }
};
```

LEARN DSA WITH C++

WEEK :: 13 **DAY**: 03 **DATE**: 19-07-2023

DIRECTED GRAPH CYCLE + TOPOLOGICAL SORT

Detect cycle in a directed graph << GeeksforGeeks >>

```
class Solution {
  public:
  bool DFS(vector<int> adj[], vector<bool>&visited, vector<bool> &path, int node)
  {
      visited[node] =1;
      path[node] =1;

      for(int i=0; i<adj[node].size(); i++)
      {
          // Adjacent node is not visited
          if(visited[adj[node][i]]==0)
          {
              if(DFS(adj, visited, path, adj[node][i]))
              return -1;
          }
          // adjacent node is visited
          else
          {
              // path =1
              if(path[adj[node][i]])
              return -1;
          }
      }
      path[node]=0;
      return 0;
  }
    // Function to detect cycle in a directed graph.
    bool isCyclic(int V, vector<int> adj[]) {
        // code here
```

```
        // visited
        vector<bool>visited(V,0);
        //path
        vector<bool>path(V,0);

        for(int i=0; i<V; i++)
        {
            if(!visited[i])
            {
                if(DFS(adj, visited, path,i))
                return 1;
            }
        }
        return 0;
    }
};
```

Topological sort << GeeksforGeeks >>

```
class Solution
{
      public:
      //Function to return list containing vertices in Topological order.
      void DFS(vector<int> adj[], stack<int> &s, vector<bool> &visit, int node)
      {
          visit[node] =1;
          for(int i=0; i<adj[node].size(); i++)
          {
              if(!visit[adj[node][i]])
              DFS(adj, s, visit, adj[node][i]);
          }

          s.push(node);
          return;
      }

      vector<int> topoSort(int V, vector<int> adj[])
      {
          // code here
          stack<int>s;
          vector<bool>visit(V,0);
          for(int i=0; i<V; i++)
          {
              if(!visit[i])
              DFS(adj, s, visit,i);
          }
          vector<int>ans;
          while(!s.empty())
          {
              ans.push_back(s.top());
              s.pop();
          }
          return ans;
      }
};
```

Topological sort << GeeksforGeeks >> another way

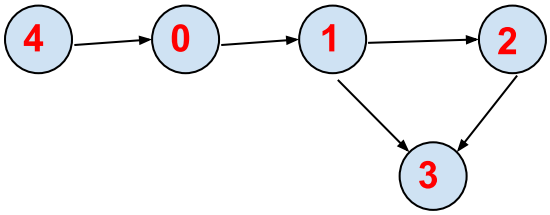
```
class Solution
{
public:
    // Function to return list containing vertices in Topological order.
    vector<int> topoSort(int V, vector<int> adj[])
    {
        // code here
        vector<int> indeg(V, 0);
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < adj[i].size(); j++) // Corrected the loop index here
            {
                indeg[adj[i][j]]++;
            }
        }

        queue<int> q;
        for (int i = 0; i < V; i++)
        {
            if (!indeg[i])
                q.push(i);
        }

        vector<int> ans;
        int node;

        while (!q.empty())
        {
            node = q.front();
            q.pop();
            ans.push_back(node);
            for (int i = 0; i < adj[node].size(); i++)
            {
                indeg[adj[node][i]]--;
                if (indeg[adj[node][i]] == 0)
                    q.push(adj[node][i]);
            }
        }

        return ans;
    }
};
```

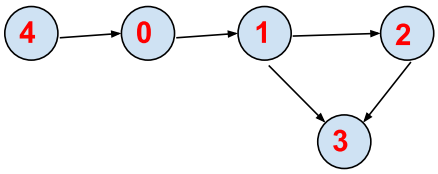
LEARN DSA WITH C++

WEEK :: 13 **DAY**: 04 **DATE**: 20-07-2023

SHORTEST DISTANCE + SPANNING TREE

TOPOLOGICAL Sort **Kahn's Algorithm**

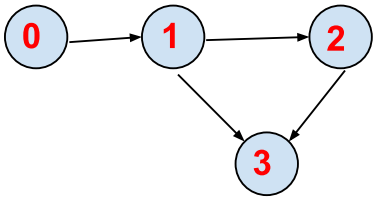




indeg :

0	1	2	3	4
0	1	1	2	0

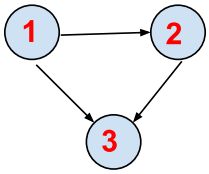
Queue :
q= 4
4



indeg :

0	1	2	3	4
0	0	1	2	0

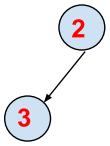
Queue :
q= 0
4 0



indeg :

0	1	2	3	4
0	0	0	1	0

Queue :
q= 1
4 0 1



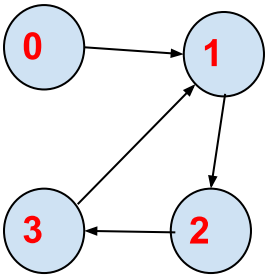
indeg :

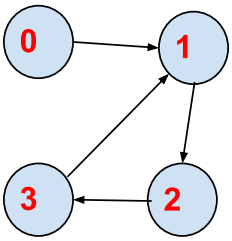
0	1	2	3	4
0	0	0	0	0

Queue :
q= 2
4 0 1 2

Queue :
q= 2
4 0 1 2 3

TOPOLOGICAL Sort Cycle **Kahn's Algorithm**



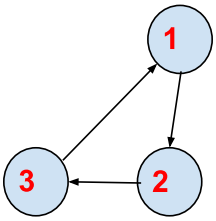


indeg :

0	1	2	3
0	1	1	1

Queue:
q = 0

0



indeg :

0	1	2	3
0	1	1	1

Queue:
q = 0
0
No one enter in Queue because there are no ZERO

Find Minimum Step

1	0	0	1
1	1	1	0
0	0	1	1
0	1	1	1
1	1	1	1

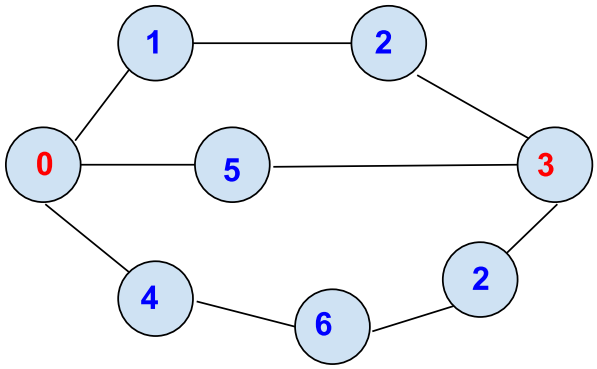
First 1 = start; second 1 = Target
0 = No entry
At a time go 1 step

Queue :

q = {(n , m) step}

q = {(0 , 0) 0}
q = {(1 , 0) 1}
q = {(1 , 1) 2}
q = {(1 , 2) 3}
q = {(2 , 2) 4}

q = {(3 , 2) 5} ; {(2 , 3) 5}
q = {(3 , 1) 6} ; {(3 , 3) 6} ; {(4 , 2) 6}
q = {(4 , 1) 7} ; {(4 , 3) 7} ;
q = {(4 , 0) 8} ; → Target Time Complexity = n*m



Sorted distance

0 -> start; 3 -> Target;

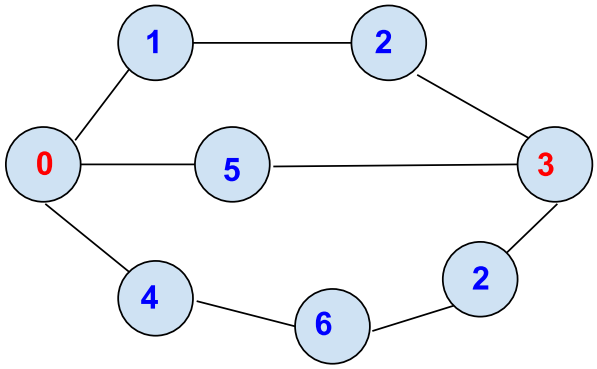
DFS

Path 1 : 0 - 1 - 2 - 3 → 3 step
Path 2 : 0 - 5 - 3 → 2 step
Path 3 : 0 - 4 - 6 -2 - 3 → 4 step
Ans :- path 2 TC = (E + V)

BFS

1 distance ;- 1 , 5, 4;
2 distance ;- 2 , 3, 6;

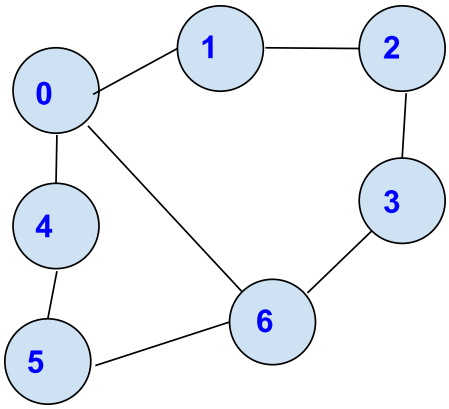
Ans :- 2 distance TC = (E + V)



Another Approach

0	1	2	3	4	5	6	7
0	inf	inf	inf	inf	inf	inf	inf
0	1	inf	inf	1	1	inf	inf
0	1	2	2	1	1	2	3

0 to 0 distance = 0
Other don't know



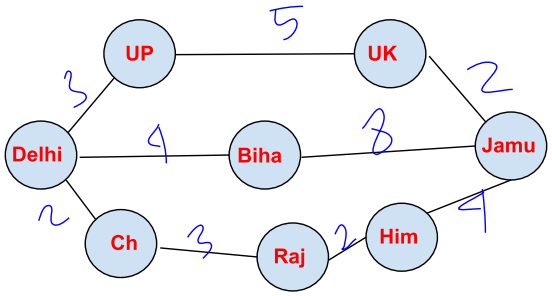
Find path 0 to 3;

0	1	2	3	4	5	6
0	0	1	6	0	6	0

Sortes path 3 - 6 - 0
Reverse : 0 - 6 - 3

Parent of 1, 4 & 6 is 0;
Parent of 2 is 1;
Parent of 3 is 6;
Parent of 5 is 6;

Find shortest Path



Queue :-

q= Delhi , 0

UP , 3;
q= Biha , 4;
Ch , 2;

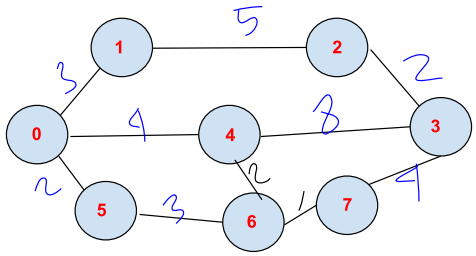
UK , 8;
q= Jamu , 12;
Raj , 5;

Jamu , 10;
q= Jamu , 12;
Him , 7;

Jamu , 10;
q= Jamu , 12;
Jamu , 11;

shortest path => Delhi- UP- UK- Jamu =10 km

Dijkstra Algorithm



Find shortest path 0 to all node

Distance from 0:- **inf = INT_MAX**

0	1	2	3	4	5	6	7
0	inf	inf	inf	inf	inf	inf	inf
0	3	inf	inf	4	2	inf	inf
0	3	8	12	4	2	5	inf
0	3	8	10	4	2	5	6

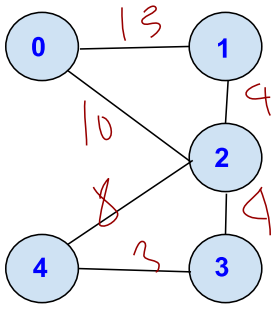
Queue :-

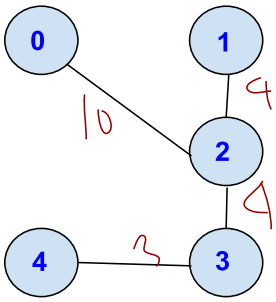
q = (1, 3);
q = (4, 4);
q = (5, 2); **take low distance**

q = (6, 5);
q = (2, 8);
q = (3, 12);

q = (7, 6);
q = (3, 10);

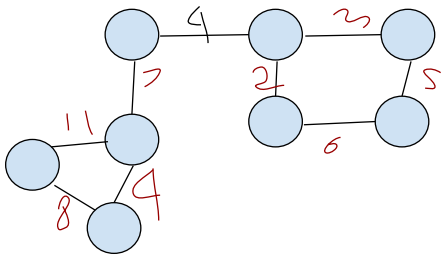
Minimum Spanning Tree :-

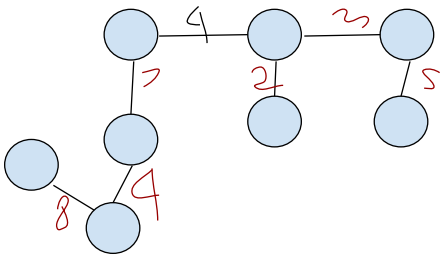




Connect all minimum distances .

Kruskal Algorithm :- Minimum edge select, if cycle is not created, the edge final.





LEARN DSA WITH C++

WEEK :: 13 **DAY**: 05 **DATE**: 24-07-2023

GRAPHS ADVANCED LEVEL

Shortest path in Undirected Graph having unit distance << GeeksforGeeks >>

```
class Solution {
  public:
    vector<int> shortestPath(vector<vector<int>>& edges, int N,int M, int src){
        // code here

        vector<int>adj[N];
        //Adjacency list create
        for(int i=0; i<M; i++)
        {
            adj[edges[i][0]].push_back(edges[i][1]);
            adj[edges[i][1]].push_back(edges[i][0]);
        };
        vector<int>dist(N);
        //return vector of distance
        for(int i=0; i<N; i++)
        dist[i]=-1;

        dist[src] =0;

        // Node + dist
        queue<pair<int,int>>q;
        q.push({src,0});

        int i, step;
        while(!q.empty())
        {
            i = q.front().first;
            step = q.front().second;
            q.pop();

            for(int k=0; k<adj[i].size(); k++)
            {
                //if adjacent node is not visited yet
                if(dist[adj[i][k]]== -1)
                {
                    q.push({adj[i][k], step+1});
                    dist[adj[i][k]] = step +1;
                }
            }
        }
        return dist;
    }
};
```

Implementing Dijkstra Algorithm << GeeksforGeeks >>

```
class Solution
{
      public:
       //Function to find the shortest distance of all the vertices
    //from the source vertex S.
    vector <int> dijkstra(int V, vector<vector<int>> adj[], int src)
    {
        // Code here
        vector<int>dist(V);
        for(int i=0; i<V; i++)
        dist[i] =-1;

         priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> q; // Fixed the priority_queue declaration
        // first = weight / dist second = Node
        q.push({0,src});

        int Node, i, j, step;
        while(!q.empty())
        {
            step = q.top().first;
            Node = q.top().second;
            q.pop();
            // Node step
            //Dist value ==-1
            // Not equal -1
            if(dist[Node] != -1)
            continue;

            dist[Node] = step;

            for(int j=0; j<adj[Node].size(); j++)
            {
                if(dist[adj[Node][j][0]] ==-1)
                {
                    q.push({step + adj[Node][j][1], adj[Node][j][0]});
                }
            }
        }
        return dist;
    }
};
```

Prims Algorithm	Kruskal Algorithm
It is connected to a small distance .	It is connected to small edges first.