LEARN DSA WITH C++

WEEK :: 14

LEARN DSA
WITH C++

COURSE INSTRUCTOR BY
ROHIT NEGI

Check Profile

MADE BY-
PRADUM SINGHA

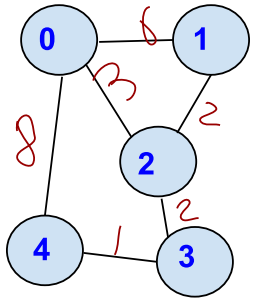
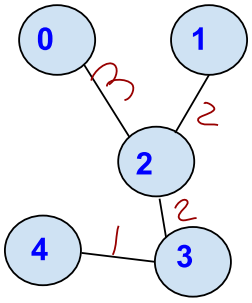
My profile

LEARN DSA WITH C++

WEEK :: 14 **DAY**: 01 **DATE**: 25-07-2023

PRIMS AND KRUSKAL ALGORITHM

Prims Algorithm :- **MST**

Main problem	MST	Logic
		Connect small distance Remove high distance Reach all node in minimum distance for required low cost No Cycle available Ans = 8

Queue

{weight,Node}

q = {0, 0}

= {6,1}, {3,2}, {8,4} take {3,2}

= {6,1}, {2,3}, {8,4}, {2,1} take {2,3}

= {6,1}, {1,4}, {8,4}, {2,1} take {1,4}

= {6,1}, {8,4}, {2,1} take {2,1} [we can stop because all node visited]

Weight

0	1	2	3	4
max	max	max	max	max
0	6	3	max	8
0	2	3	2	8
0	2	3	2	1

Parent:

0	1	2	3	4
-1	-1	-1	-1	-1
-1	0	0	-1	0
-1	2	0	2	0
-1	2	0	2	3

Visited:

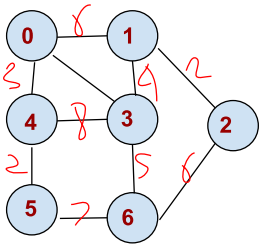
0	1	2	3	4
0	0	0	0	0
1	0	0	0	0
1	0	1	0	0
1	0	1	1	0
1	0	1	1	1
1	1	1	1	1

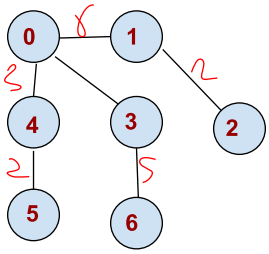
Minimum Spanning Tree << GeeksforGeeks >>

```
class Solution
{
public:
    int spanningTree(int V, vector<vector<int>> adj[])
    {
```

```
        //weight
        vector<int> weight(V, INT_MAX);
        //Parent
        vector<int> parent(V, -1);
        //visited
        vector<bool> visited(V, 0);
        int count=0, cost = 0;
        //count of vertex
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int,
int>>> q;
        q.push({0, 0});  // path , Node
        int node, path, adjNode, adjPath;

        while (!q.empty())
        {
            path = q.top().first;
            node = q.top().second;
            q.pop();

            //If node is already visited, skep the step
            if (visited[node]==1)
                continue;

            //node is not visited yet
            visited[node] = 1;

            // look at Adjacent Node
            for (int i = 0; i < adj[node].size(); i++)
            {
                int adjNode = adj[node][i][0];
                int adjPath = adj[node][i][1];
                if (!visited[adjNode] && adjPath < weight[adjNode])
                {
                    q.push({adjPath, adjNode});
                    parent[adjNode] = node;
                    weight[adjNode] = adjPath;
                }
            }
        }

        for (int i = 1; i < V; i++)
            cost += weight[i];

        return cost;
    }
};
```

KRUSKAL ALGORITHM





1. Connect all small edges
2. If there is no small edge between two nodes, then it is connected.
3. No Cycle create

Create edge – Disconnected edge

Rank

Edge :- (0,1); (1,2); (3,4); (6,5); (2,3); (1,5);

Find Parent :

Node 1 = Pv; Node2 = Pu

rank[Pv] = rank[Pu]
rank[Pv]++;
parent[Pv]=Pu;

rank[Pv] > rank[Pu]
parent[Pu] = Pv;

rank[Pv] < rank[Pu]
parent[Pv]=Pu;

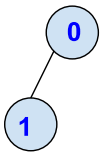
Rank:-

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

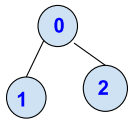
Parent:-

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

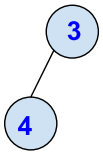
Pu=0; Pv=1;
rank(0,0)



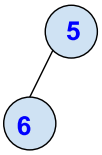
Pu=0;Pv=0
rank(0,0)



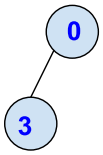
Pu=3;Pv=4
rank(0,0)



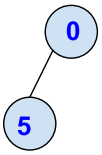
Pu=5;Pv=6
rank(0,0)

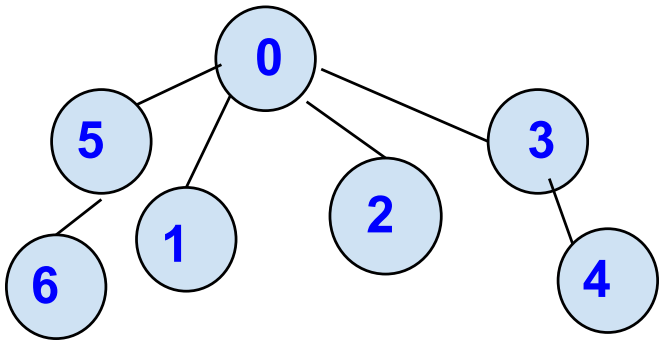


(2,3)-parent different



(0,5)-parent different





Minimum Spanning Tree << GeeksforGeeks >>

```
class Solution {
public:
    int findPar(int u, vector<int> &parent) {
        if (u == parent[u])
            return u;

        return parent[u] = findPar(parent[u], parent);
    }

    bool unionSet(int u, int v, vector<int> &rank, vector<int> &parent) {
        int pu = findPar(u, parent);
        int pv = findPar(v, parent);
        if (pu == pv)
            return true; // already connected

        if (rank[pu] == rank[pv]) {
            rank[pu]++;
            parent[pv] = pu;
        } else if (rank[pu] < rank[pv]) {
            parent[pu] = pv;
        } else {
            parent[pv] = pu;
        }

        return false;
    }

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

        // min heap, insert every edge
        priority_queue<pair<pair<int, int>, int>, vector<pair<pair<int, int>, int>>,
greater<pair<pair<int, int>,int>>> q;
        int u, v, w;

        for (int i = 0; i < V; i++)
            for (int j = 0; j < adj[i].size(); j++) {
                u = i;
                v = adj[i][j][0];
                w = adj[i][j][1];
                q.push({ {w, u}, v });
            }
        int count_edges = 0, cost = 0;
        while (!q.empty()) {
            if (count_edges == V - 1)
                break;
            w = q.top().first.first;
            u = q.top().first.second;
            v = q.top().second;
            q.pop();
```

```
            if (unionSet(u, v, rank, parent))
                continue;

            count_edges++;
            cost += w;
        }
        return cost;
    }
};
```

LEARN DSA WITH C++

WEEK :: 14 **DAY**: 02 **DATE**: 26-07-2023

STRONGLY CONNECTED COMPONENT + BELLMAN FORD

Bridge edge in a graph << GeeksforGeeks >>

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

        visited[node] = 1;
        for (int i = 0; i < adj[node].size(); i++)
        {
            if (!visited[adj[node][i]])
            {
                if (DFS(adj[node][i], target, adj, visited))
                    return 1;
            }
        }
        return 0;
    }

    //Function to find if the given edge is a bridge in graph.
    int isBridge(int V, vector<int> adj[], int c, int d)
    {
        // Update the adjacency list to remove the edge (c, d)
        for (int i = 0; i < adj[c].size(); i++)
        {
            if (adj[c][i] == d)
            {
                adj[c][i]=c;
                break;
            }
        }

        for (int i = 0; i < adj[d].size(); i++)
```

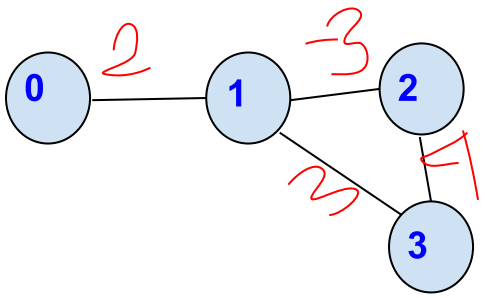
```
        {
            if (adj[d][i] == c)
            {
                adj[d][i]=d;
                break;
            }
        }

        vector<bool> visited(V, 0);
        // 0: Target node not found, the edge is a bridge
        // 1: Target node found, the edge is not a bridge
        return !DFS(c, d, adj, visited);
    }
};
```

Strongly Connected Components (Kosaraju's Algo) << GeeksforGeeks >>

```
class Solution
{
public:
    void DFS(int node, vector<vector<int>> &adj, stack<int> &s, vector<bool>
&visited)
    {
        visited[node] = 1;
        for (int i = 0; i < adj[node].size(); i++)
        {
            if (!visited[adj[node][i]])
                DFS(adj[node][i], adj, s, visited);
        }
        s.push(node);
    }

    void SCC(int node, vector<bool> &visited, vector<vector<int>> &adj)
    {
        visited[node] = true;
        for (int i = 0; i < adj[node].size(); i++)
        {
            if (!visited[adj[node][i]])
                SCC(adj[node][i], visited, adj);
        }
    }

    //Function to find the number of strongly connected components in the graph.
    int kosaraju(int V, vector<vector<int>>& adj)
    {
        // Modified Topological sort
        stack<int> s;
        vector<bool> visited(V, false);
        for (int i = 0; i < V; i++)
        {
            if (!visited[i])
                DFS(i, adj, s, visited);
        }

        // Transpose the graph, reverse the edge
        vector<vector<int>> trans(V);
        for (int i = 0; i < V; i++)
```

```
        {
            for (int j = 0; j < adj[i].size(); j++)
            {
                trans[adj[i][j]].push_back(i);
            }
        }

        int count = 0;
        for (int i = 0; i < V; i++)
            visited[i] = 0;

        while (!s.empty())
        {
            // stack top element is not visited, count++
            if (!visited[s.top()])
            {
                count++;
                SCC(s.top(), visited, trans);
            }
            s.pop();
        }
        return count;
    }
};
```

BELLMAN FORD ALGORITHM



dist [V] = min(dist[v], dist[v]+w)

u		v	w
2	->	3	4
1	->	3	3
0	->	1	2
1	->	2	-3

Edges = 4
operation = (4-1)=3

Distance from 0 to node

0	1	2	3
0	max	max	max
	2	-1	
			3
0	2	-1	3

1st:-
2 ->3 =4; 0 -> 2 -> 3 = max, max +4;
1 ->3 =3; 0 -> 1 -> 3 = max, max +3;
0 ->1 =2; 0 -> 0 -> 1 = 0, 2;
1 ->2 =-3; 0 -> 1 -> 2 = 2, -1;
2nd:-
2 ->3 =4; 0 -> 2 -> 3 = -1, 3;
1 ->3 =3; 0 -> 1 -> 3 = 2, 5;
0 ->1 =2; 0 -> 0 -> 1 = 0, 2;

	1 ->2 =-3; 0 -> 1 -> 2 = 2, -1; 3rd - 2 ->3 =4; 0 -> 2 -> 3 = -1, 3; 1 ->3 =3; 0 -> 1 -> 3 = 2, 5; 0 ->1 =2; 0 -> 0 -> 1 = 0, 2; 1 ->2 =-3; 0 -> 1 -> 2 = 2, -1;

Limitation of BELLMAN FORD

1. Not work in negative cycle
2. Not work in negative edge of undirected graph

LEARN DSA WITH C++

WEEK :: 14 **DAY**: 03 **DATE**: 27-07-2023

BELLMAN FORD + FLOYD WARSHAL

Distance from the Source (Bellman-Ford Algorithm) << GeeksforGeeks >>

```
class Solution {
  public:
    /*  Function to implement Bellman Ford
    *   edges: vector of vectors which represents the graph
    *   S: source vertex to start traversing graph with
    *   V: number of vertices
    */
    void Bellmon(vector<vector<int>> &edges, vector<int> &dist)
    {
        for(int i=0; i<edges.size(); i++)
        {
            dist[edges[i][1]] = min(dist[edges[i][1]], dist[edges[i][0]] +
edges[i][2]);
        }
    }
    vector<int> bellman_ford(int V, vector<vector<int>>& edges, int S) {
        // Code here
        vector<int>dist(V,1e8);
        dist[S]=0;

        for(int i=0; i<V-1; i++)
        {
            Bellmon(edges, dist);
        }
        // Nagetive Cycle
```

```
        vector<int>ans(V);
        for(int i =0; i<V; i++)
        ans[i]=dist[i];

        Bellmon(edges,dist);

        for(int i=0; i<V; i++)
        {
            if(ans[i] != dist[i])
            {
                vector<int>temp;
                temp.push_back(-1);
                return temp;
            }
        }
        return dist;
    }
};
```

```
class Solution {
public:
    /*  Function to implement Bellman Ford
    *   edges: vector of vectors which represents the graph
    *   dist: vector to store the shortest distances from the source vertex
    */
    bool Bellman(vector<vector<int>>& edges, vector<int>& dist)
    {
        bool count = false;
        for (int i = 0; i < edges.size(); i++)
        {
            if (dist[edges[i][1]] > dist[edges[i][0]] + edges[i][2])
            {
                dist[edges[i][1]] = dist[edges[i][0]] + edges[i][2];
                count = true; // Set 'count' to true when a relaxation is made
            }
        }
        return count;
    }
    vector<int> bellman_ford(int V, vector<vector<int>>& edges, int S)
    {
        vector<int> dist(V, 1e8);
        dist[S] = 0;

        // Relax edges (V-1) times to find the shortest paths
        for (int i = 0; i < V - 1; i++)
        {
            if (!Bellman(edges, dist))
                return dist; // No relaxation in this iteration, return early
        }
        // Check for negative cycles after (V-1) iterations
        if (Bellman(edges, dist))
        {
            vector<int> temp = {-1};
            return temp; // Negative cycle detected
        }

        return dist;
    }
};
```

LEARN DSA WITH C++

WEEK :: 14 **DAY**: 04 **DATE**: 28-07-2023

GRAPH ADVANCE EULER

Floyd Warshall << GeeksforGeeks >>

```
class Solution {
  public:
      void shortest_distance(vector<vector<int>>&matrix){
          // Code here
          int n= matrix.size();
          for(int i=0; i<n; i++)
          for(int j=0; j<n; j++)
          if(matrix[i][j] ==-1)
          matrix[i][j] = 1e9;

          for(int k=0; k<n; k++)
          for(int i=0; i<n; i++)
          for(int j=0;j<n; j++)
          {
              matrix[i][j] = min(matrix[i][j], matrix[i][k] + matrix[k][j]);
          }

          for(int i=0; i<n; i++)
          for(int j=0; j<n; j++)
          if(matrix[i][j] ==1e9)
          matrix[i][j] =-1;
      }
};
```

Euler Circuit in an Undirected Graph << GeeksforGeeks >>

```
class Solution {
public:

void DFS(int node, vector<int>adj[], vector<bool> &visited)
{
   visited[node] = 1;
        for (int i = 0; i < adj[node].size(); i++)
        {
            if (!visited[adj[node][i]])
                DFS(adj[node][i], adj, visited);
        }
}
      bool isEularCircuitExist(int V, vector<int>adj[])
      {
          // Code here
          vector<int> degree(V, 0);
        for (int i = 0; i < V; i++)
            degree[i] = adj[i].size();

        int node = -1;
```

```
        for (int i = 0; i < V; i++) {
            if (degree[i]) {
                node = i;
                break;
            }
        }

        if (node == -1)
            return 1;

        // Check if degree is even or odd
        for (int i = 0; i < V; i++) {
            if (degree[i] % 2)
                return 0;
        }

        // Traverse the whole circuit
        vector<bool> visited(V, 0);
        DFS(node, adj, visited);

        // Node is not visited and degree of node is exist
        for (int i = 0; i < V; i++) {
            if (!visited[i] && degree[i])
                return 0;
        }

        return 1;
    }
};
```

Circle of strings

<< Geeksforgeeks >>

```
class Solution
{
    public:
    void DFS(int node, vector<int>adj[], vector<bool> &visited)
    {
        visited[node]=1;
        for(int i=0; i<adj[node].size(); i++)
        {
            if(!visited[adj[node][i]])
            DFS(adj[node][i], adj, visited);
        }
    }
    int isCircle(int N, vector<string> A)
    {
        // code here
        vector<int>adj[26];
        vector<int>in(26,0);
        vector<int>out(26, 0);
        string s;
        for(int i=0; i<N; i++)
        {
            s = A[i];
            adj[s[0]-'a'].push_back(s[s.size()-1] -'a');
            out[s[s.size()-1] -'a']++;
            in[s[0]-'a']++;
```

```
        }
        int node =-1;
        for(int i=0; i<26; i++)
        {
            if(in[i] != out[i])
            return 0;
            if(in[i])
            node =i;
        }

        vector<bool>visited(26,0);
        DFS(node, adj, visited);

        for(int i=0; i<26; i++)
        {
            if(in[i] && !visited[i])
            return 0;
        };

        return 1;
    }
};
```

LEARN DSA WITH C++

WEEK :: 14 **DAY**: 05 **DATE**: 31-07-2023

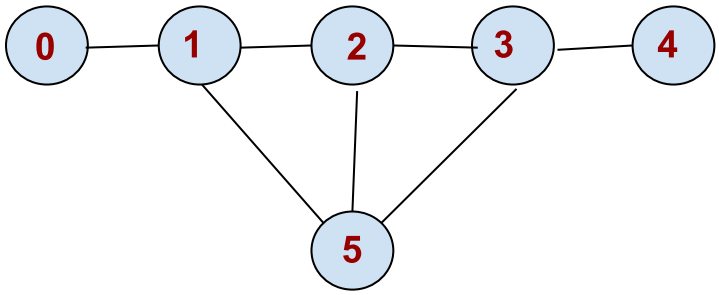
ARTICULATION POINT

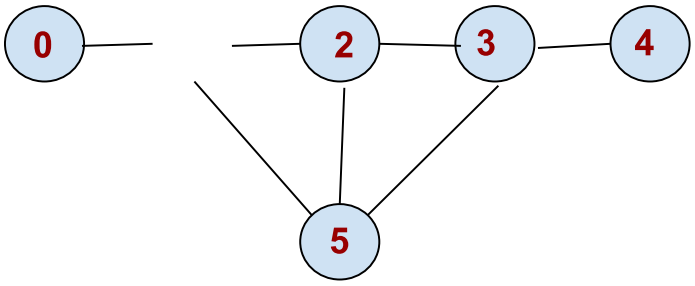
Critical Connections << GeeksforGeeks >>

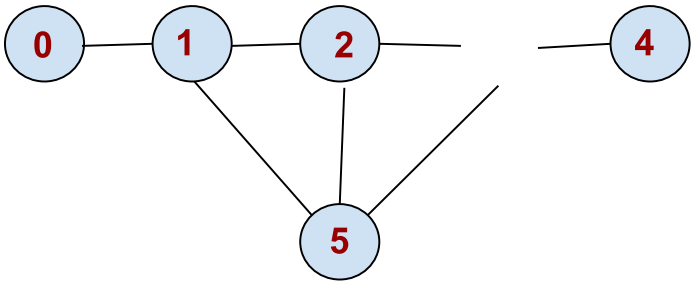
```
class Solution {
public:
int DFS(int node, vector<int> adj[], vector<int> &disc, vector<int> &low,
vector<vector<int>> &bridge, int timer, int parent)
{
    if(disc[node] != -1)
    return disc[node];
    timer++;
    disc[node] = low[node] = timer;

    for(int i=0; i<adj[node].size(); i++)
    {
        int neighbour = adj[node][i];
        if(neighbour == parent)
        continue;

        low[node] = min(low[node], DFS(neighbour, adj, disc, low, bridge, timer,
node));
```

```
        // Bridge condition
        if(low[neighbour] > disc[node])
        {
            if(node > neighbour)
            bridge.push_back({neighbour, node});
            else
            bridge.push_back({node, neighbour});
        }
    }
    return low[node];
};
      vector<vector<int>>criticalConnections(int V, vector<int> adj[]){
          // Code here
          vector<vector<int>>bridge;
          vector<int>disc(V, -1);
          vector<int>low(V, -1);
          int timer =-1, parent =-1;
          DFS(0, adj, disc, low, bridge, timer, parent);
          sort(bridge.begin(), bridge.end());

          return bridge;

      }
};
```

Articulation Point







Node 1 & 3 is an Articulation Point because if we remove them, the tree is disconnected.