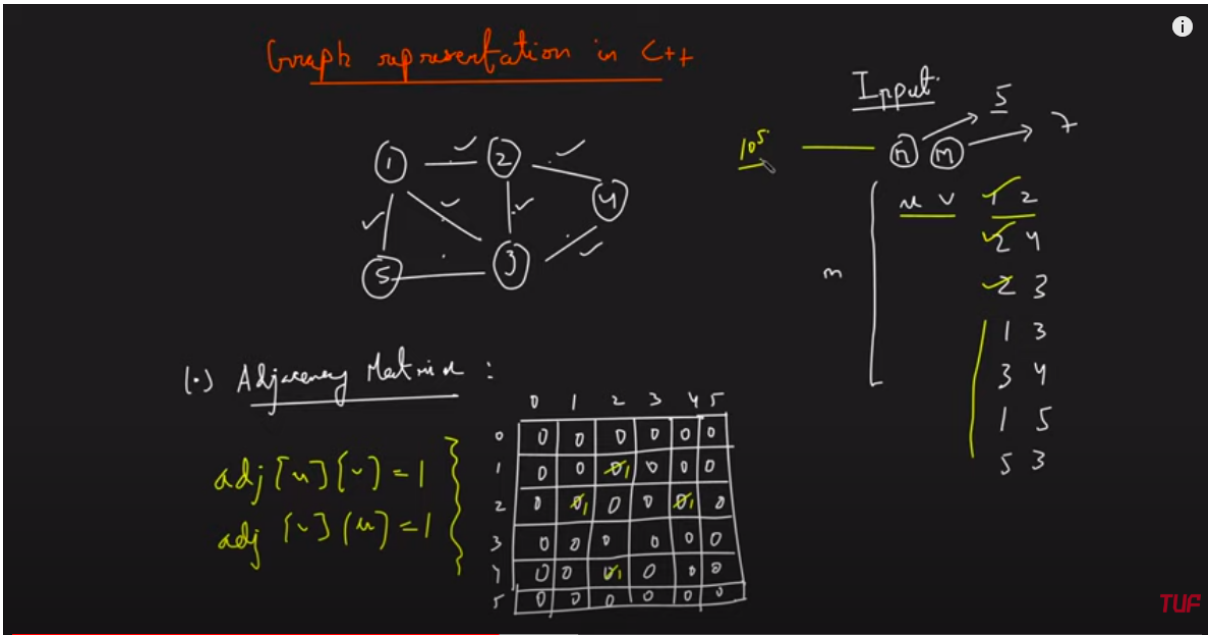
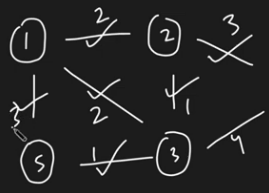



```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;

    // declare the adjacent matrix
    int adj[n+1][n+1];

    // take edges as input
    for(int i = 0;i<m;i++) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = 1;
        adj[v][u] = 1;
    }
    return 0;
}
```

Graph Representation in C++

int arr[6]
pair<int, int> arr[6]



N = 5

vector<int> adj[6]

u	v
1	2
1	5
1	3
3	5
2	3
2	4
3	4

7 edges.

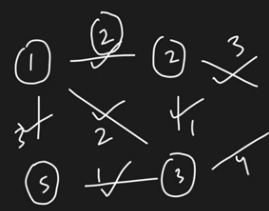
SC → O(N + 2E)

adj[u].push_back(v)
adj[v].push_back(u)

0	
1	(2, 5, 3)
2	(1, 3, 4)
3	(1, 5, 2, 4)
4	(2, 3)
5	(1, 3

Graph Representation in C++

int arr[6]
pair<int, int> arr[6]



N = 5

pair<int, int>

vector<int> adj[6]

original
wt

u	v
1	2
1	5
1	3
3	5
2	3
2	4
3	4

7 edges.

SC → O(N + 2E) + 2E << N*N

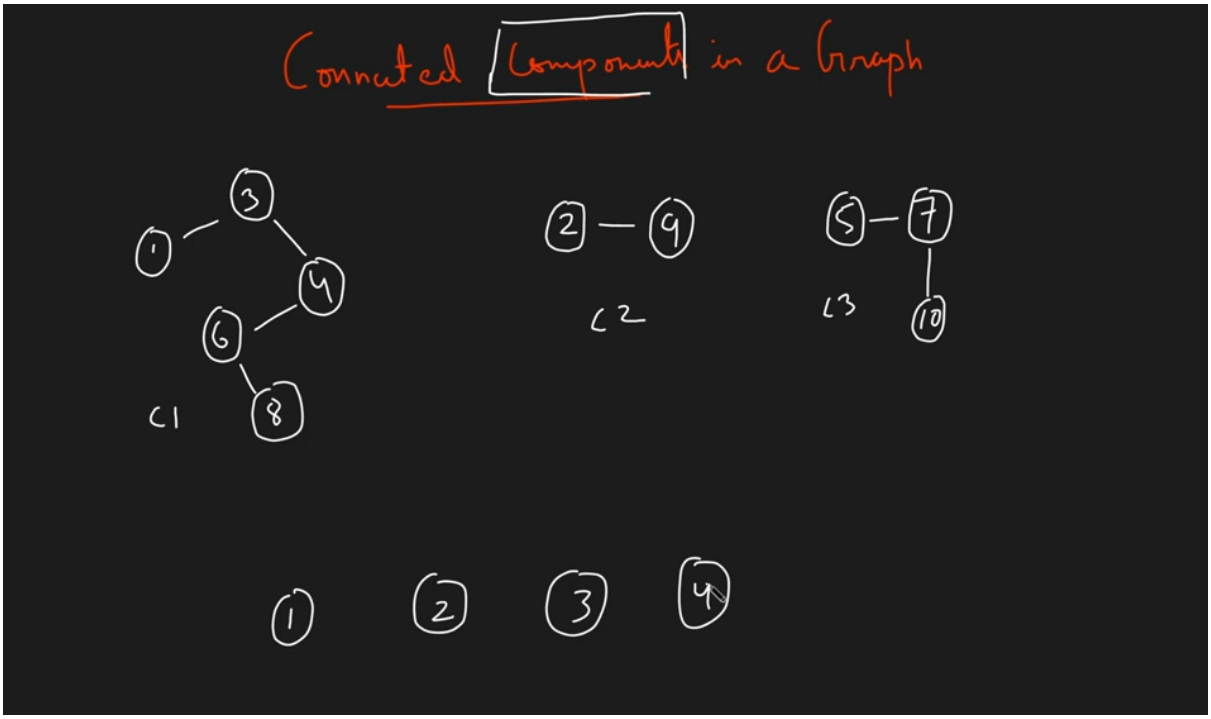
adj[u].push_back(v)
adj[v].push_back(u)

({v, w})

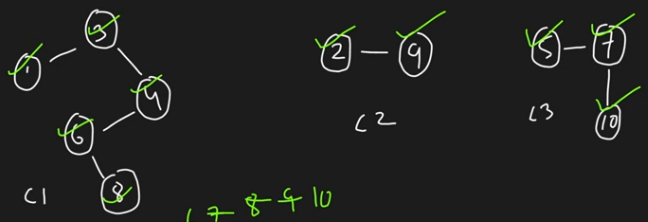
0	
1	((2, 4) 5, 3)
2	(1, 3, 4)
3	(1, 5, 2, 4)
4	(2, 3)
5	(1, 3

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;

    vector<pair<int,int>> adj[n+1];
    for(int i = 0;i<m;i++) {
        int u, v, wt;
        cin >> u >> v >> wt;

        adj[u].push_back({v, wt});
        adj[v].push_back({u, wt});
    }
    return 0;
```

TUF



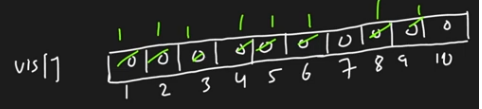
Connected Components in a Graph



1 3 4 5 6 7 8 9 10

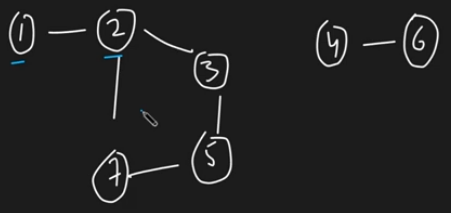
```
for( i=1; i<=10 ; i++)
{
  if (!vis[i])
  {
    dfs
    bfs
  }
}
```

DFS / BFS



dfs()

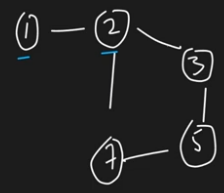
1) BFS (Breadth First Search)



1	2
2	1 3 7
3	2 5
4	6
5	3 7
6	4
7	2 5

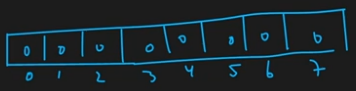
1 2 3 7 5 4 6

1) BFS (Breadth First Search)



bfs(1)

1	2
2	1 3 7
3	2 5
4	6
5	3 7
6	4
7	2 5



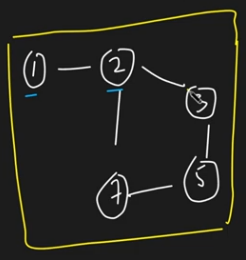
vis

i=1

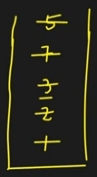
```
for (i = 1 → n)
{
    if (vis[i] == 0)
        bfs(i);
{
```

1) BFS (Breadth First Search)

1, 2, 3, 7, 5



bfs(1)

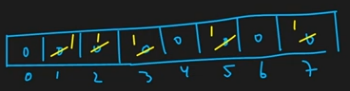


node = 1

node = 2

node = 3 7 5

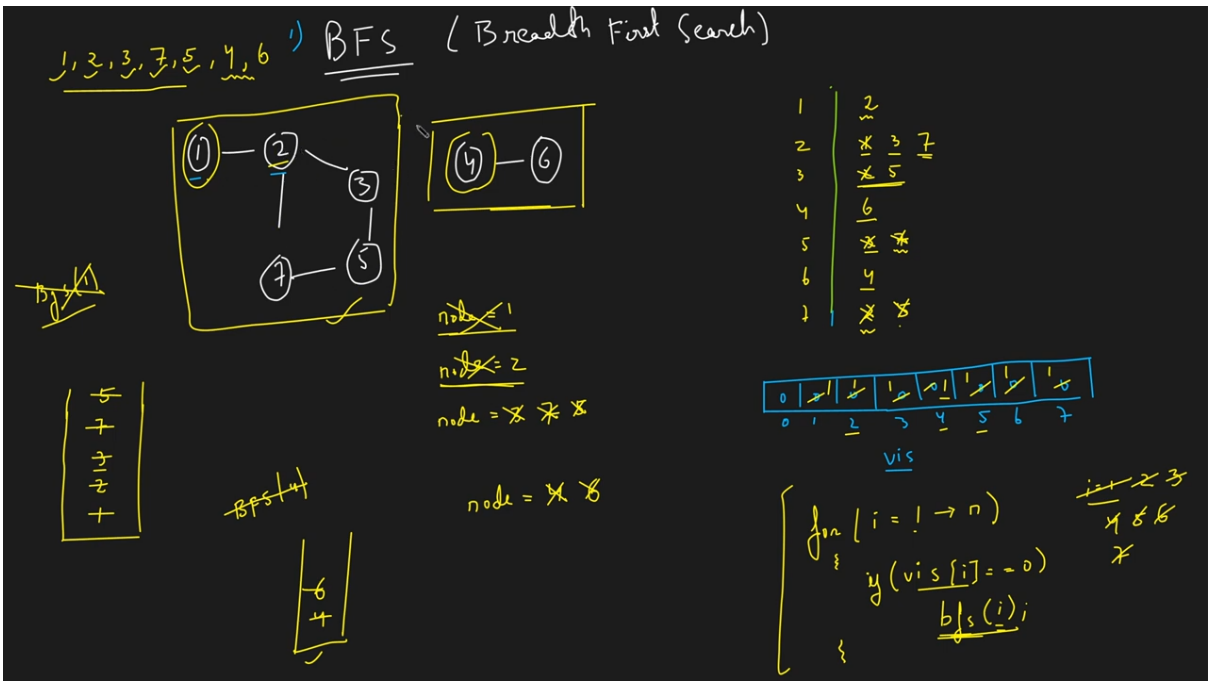
1	2
2	1 3 7
3	2 5
4	6
5	3 7
6	4
7	2 5

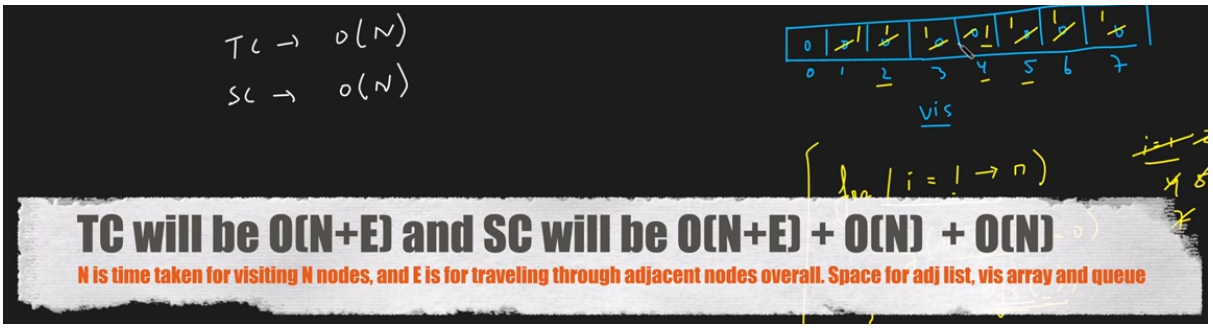


vis

i=1

```
for (i = 1 → n)
{
    if (vis[i] == 0)
        bfs(i);
{
```

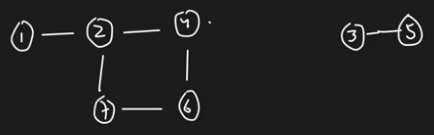





```java
class Solution
{
    public ArrayList<Integer> bfsOfGraph(int V, ArrayList<ArrayList<Integer>> adj)
    {

        ArrayList<Integer> bfs = new ArrayList<>();
        boolean vis[] = new boolean[V+1];

        for(int i = 1;i<=V;i++) {
            if(vis[i] == false) {

                Queue<Integer> q = new LinkedList<>();
                q.add(i);
                vis[i] = true;

                while (!q.isEmpty())
                {
                    Integer node = q.poll();
                    bfs.add(node);

                    for(Integer it: adj.get(node)) {
                        if(vis[it] == false) {
                            vis[it] = true;
                            q.add(it);
                        }
                    }
                }
            }
        }

        return bfs;

        // Code here
    }
}
```

```cpp
class Solution {
public:
    vector<int> bfsOfGraph(int V, vector<int> adj[]){
        vector<int> bfs;
        vector<int> vis(V+1, 0);

        for(int i = 1;i<=V;i++) {
            if(!vis[i]) {
                queue<int> q;
                q.push(i);
                vis[i] = 1;
                while(!q.empty()) {
                    int node = q.front();
                    q.pop();
                    bfs.push_back(node);

                    for(auto it : adj[node]) {
                        if(!vis[it]) {
                            q.push(it);
                            vis[it] = 1;
                        }
                    }
                }
            }
        }

        return bfs;
    }
};
} // } Driver Code Ends
```

DFS (Depth First Search)

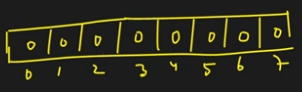


Adj List.

1 → 2
2 → 1 4 7
3 → 5
4 → 2 6
5 → 3
6 → 4 7
7 → 2 6

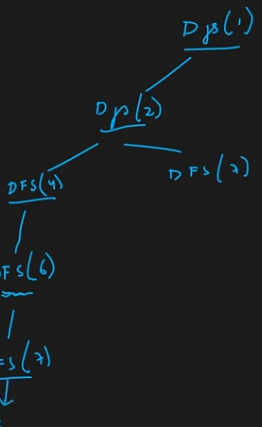
```
for( i = 1 → 7)
{
  if( !vis[i])
     dfs( i)
}
```

vis



DFS (Depth First Search)

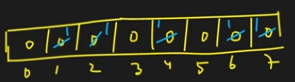
```
for( i = 1 → 7)
{
  if( !vis[i])
    dfs( i)
}
```



Adj List.

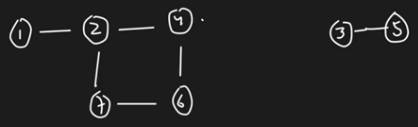
1 → 2
2 → 1 4 7
3 → 5
4 → 2 6
5 → 3
6 → 4 7
7 → 2 6

Vis

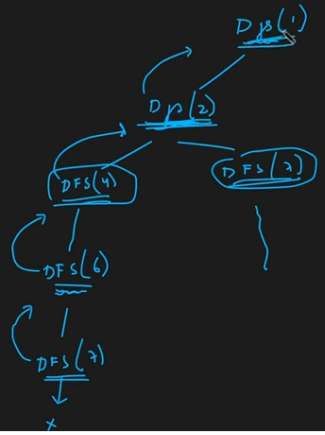


1 2 4 6 7

DFS (Depth First Search)



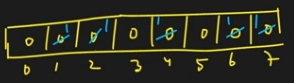
```
for( i = 1 → 7)
{
  if( !vis[i])
    dfs( i)
}
```



Adj List.

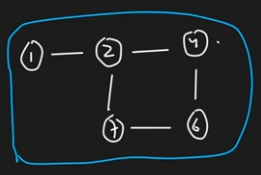
1 → 2
2 → 1 4 7
3 → 5
4 → 2 6
5 → 3
6 → 4 7
7 → 2 6

Vis



1 2 4 6 7 3 5

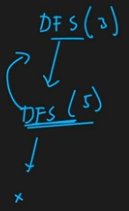
DFS (Depth First Search)



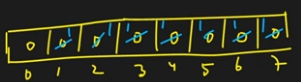
Adj list.

1 → 2
2 → ~~1~~ 4 7
3 → 5
4 → ~~2~~ 6
5 → 3
6 → ~~4~~ 7
7 → ~~2~~ ~~6~~

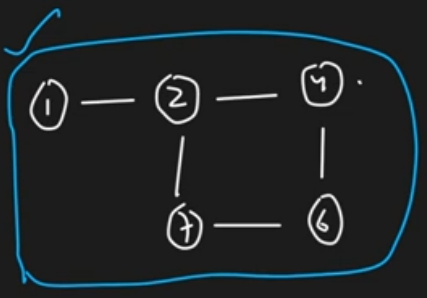
```
3
2
i=1
for( i = 1 → 7)
{
   if( !vis[i])
      dfs( i)
}
```

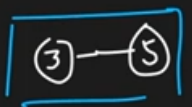


vis

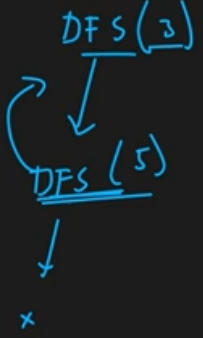


8 ~~7~~ ~~6~~ ~~5~~ ~~4~~ ~~3~~ ~~2~~





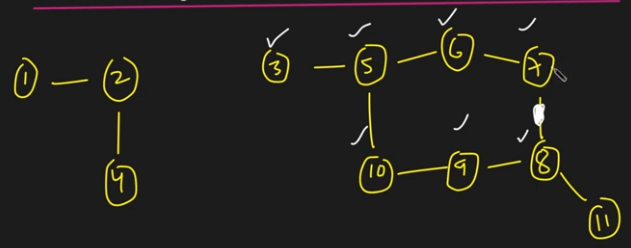

```
i=1
for( i = 1 → 7)
{
   if( !vis[i])      i=1 { 3
      dfs( i).
}
```



```java
class Solution
{
    public void dfs(int node, boolean vis[], ArrayList<ArrayList<Integer>> adj, ArrayList<Integer> storeDfs) {
        storeDfs.add(node);
        vis[node] = true;
        for(Integer it: adj.get(node)) {
            if(vis[it] == false) {
                dfs(it, vis, adj, storeDfs);
            }
        }
    }
    public ArrayList<Integer> dfsOfGraph(int V, ArrayList<ArrayList<Integer>> adj)
    {
        ArrayList<Integer> storeDfs = new ArrayList<>();
        boolean vis[] = new boolean[V+1];
        for(int i = 1;i<=V;i++) {
            if(vis[i] == 0) {
                dfs(i, vis, adj, storeDfs);
            }
        }
        return storeDfs;
    }
}
```

```cpp
class Solution {
    void dfs(int node, vector<int> &vis, vector<int> adj[], vector<int> &storeDfs) {
        storeDfs.push_back(node);
        vis[node] = 1;
        for(auto it : adj[node]) {
            if(!vis[it]) {
                dfs(it, vis, adj, storeDfs);
            }
        }
    }
public:
    vector<int>dfsOfGraph(int V, vector<int> adj[]){
        vector<int> storeDfs;
        vector<int> vis(V+1, 0);
        for(int i = 1;i<=V;i++) {
            if(!vis[i]) {
                dfs(i, vis, adj, storeDfs);
            }
        }

        return storeDfs;
    }
};
```

Detect a cycle in Undirected Graph (BFS)



adj list

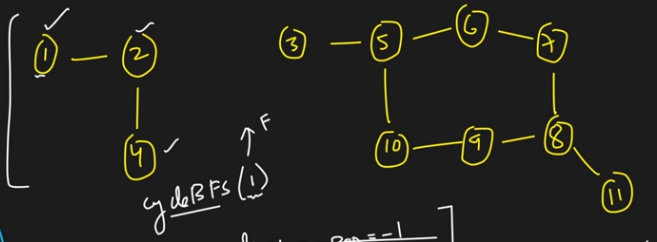
1	2
2	1 4
3	5
4	2
5	3 10 16
6	5 7
7	6 8
8	7 9 11
9	10 8
10	5 9
11	8

```
for (i = 1 - N)
{
  if (!vis[i])
    if (cycleBfs(i))
      return T;
}
```

vis

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0

Detect a cycle in Undirected Graph (BFS)



~~node = 1 par = -1~~

~~node = 2 prev = 1~~

~~node = 4 par = 2~~

~~(4, 2)~~ (2, 1) (1, -1)

Q <node, par>

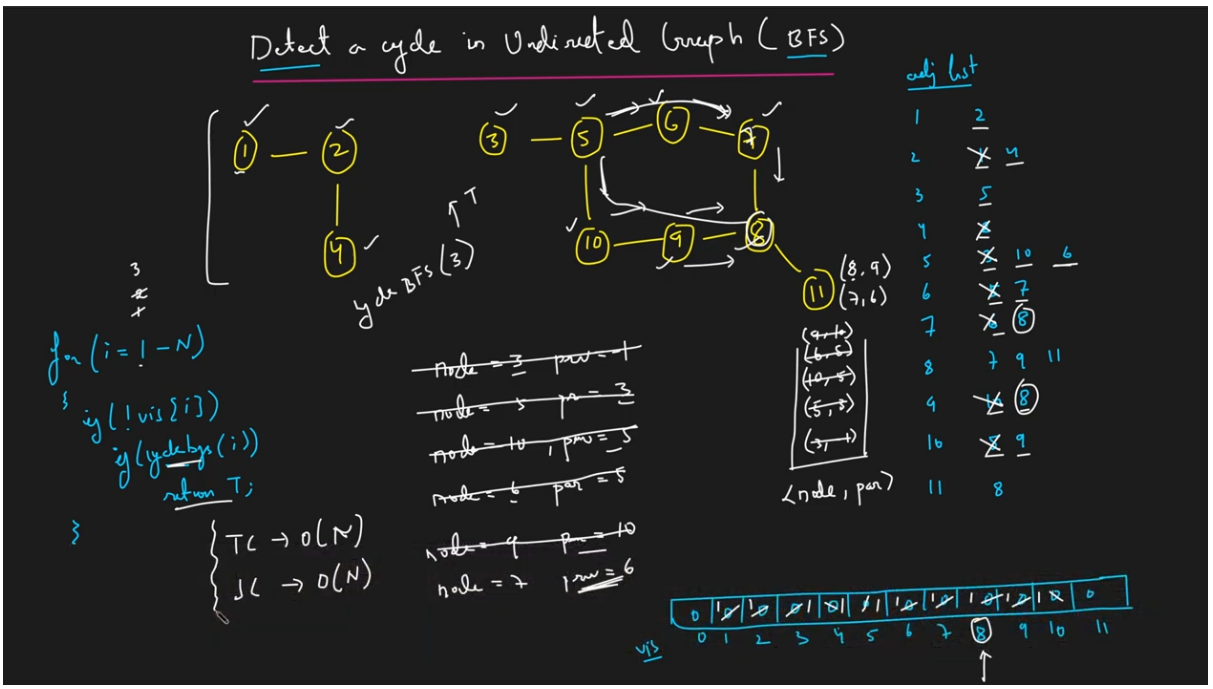
adj list

1	2
2	~~2~~ ~~4~~
3	5
4	~~2~~
5	3 10 16
6	5 7
7	6 8
8	7 9 11
9	10 8
10	5 9
11	8

```
for (i = 1 - N)
{
  if (!vis[i])
    if (cycleBfs(i))
      return T;
}
```

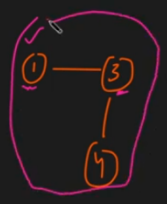
vis

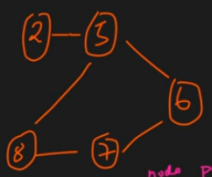
0	1	2	3	4	5	6	7	8	9	10	11
0	~~1~~	~~1~~	0	~~1~~	0	0	0	0	0	0	0




```cpp
class Solution {

public:
    bool checkForCycle(int s, int V, vector<int> adj[], vector<int>& visited)
    {
        // Create a queue for BFS
        queue<pair<int,int>> q;

        visited[s] = true;
        q.push({s, -1});

        while (!q.empty()) {

            int node = q.front().first;
            int par = q.front().second;
            q.pop();

            for (auto it : adj[node]) {
                if (!visited[it]) {
                    visited[it] = true;
                    q.push({it, node});
                }
                else if (par != it)
                    return true;
            }
        }
        return false;
    }
public:
    bool isCycle(int V, vector<int>adj[]){
        vector<int> vis(V+1, 0);
        for(int i = 1;i<=V;i++) {
            if(!vis[i]) {
                if(checkForCycle(i, V, adj, vis)) return true;
            }
        }

        return false;
    }
};
```

Cycle Detection in Undirected Graph (DFS)

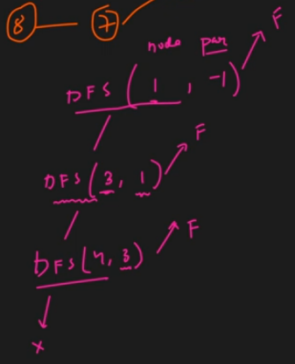


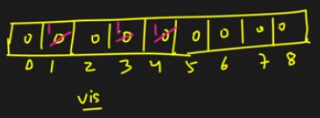


adj list

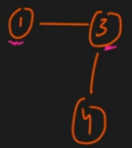
1	3
2	5
3	1 4
4	3
5	2 6 8
6	5 7
7	6 8
8	7 5

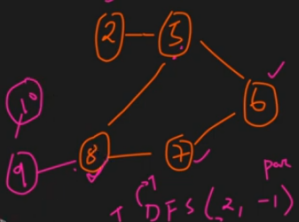
```
for (i = 1 → 8)
{
  if (vis[i] == 0)
  {
    if (cycleDFS(i))
      return T;
  }
}
```





Cycle Detection in Undirected Graph (DFS)

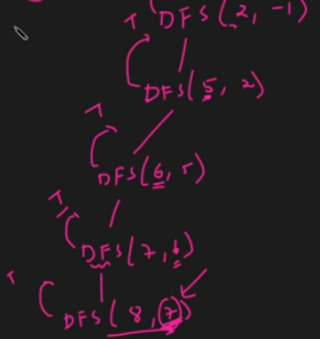


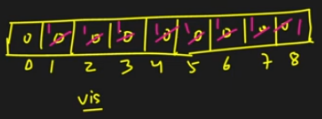


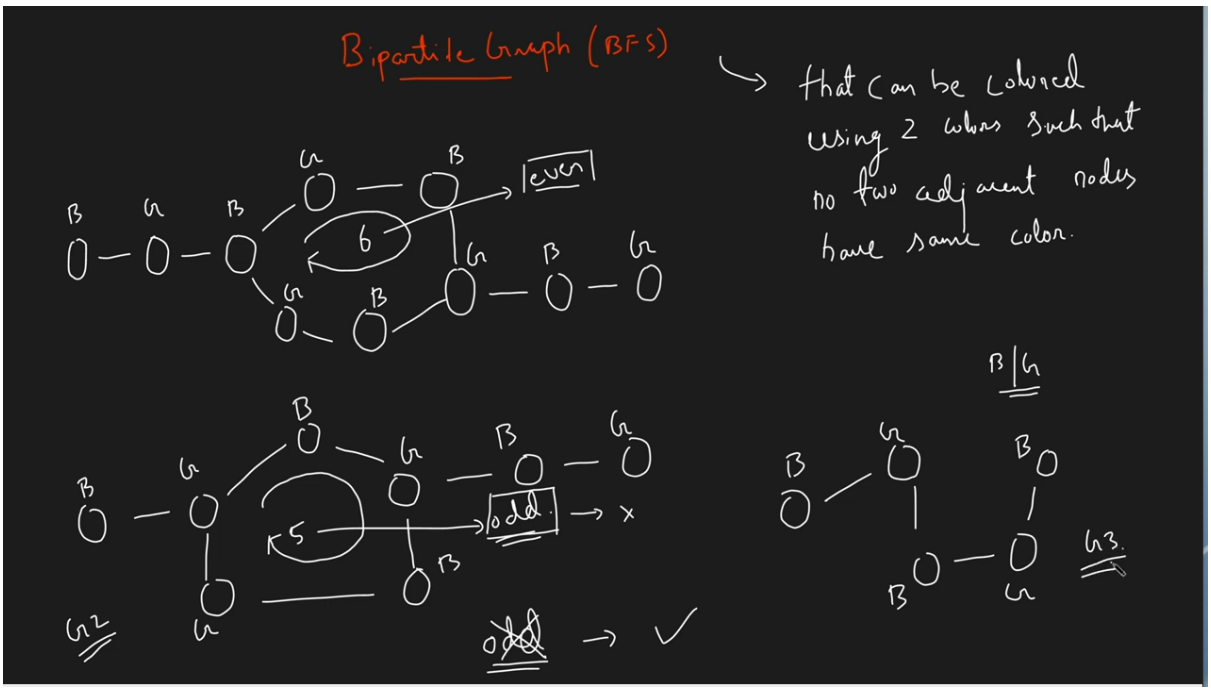
adj list

1	3
2	5
3	1 4
4	3
5	2 6 8
6	5 7
7	6 8
8	7 5

```
for (i = 2 → 8)
{
  if (vis[i] == 0)
  {
    if (cycleDFS(i))
      return T;
  }
}
```



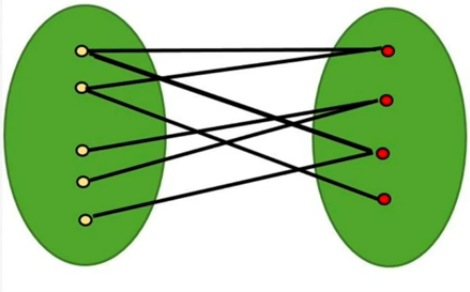


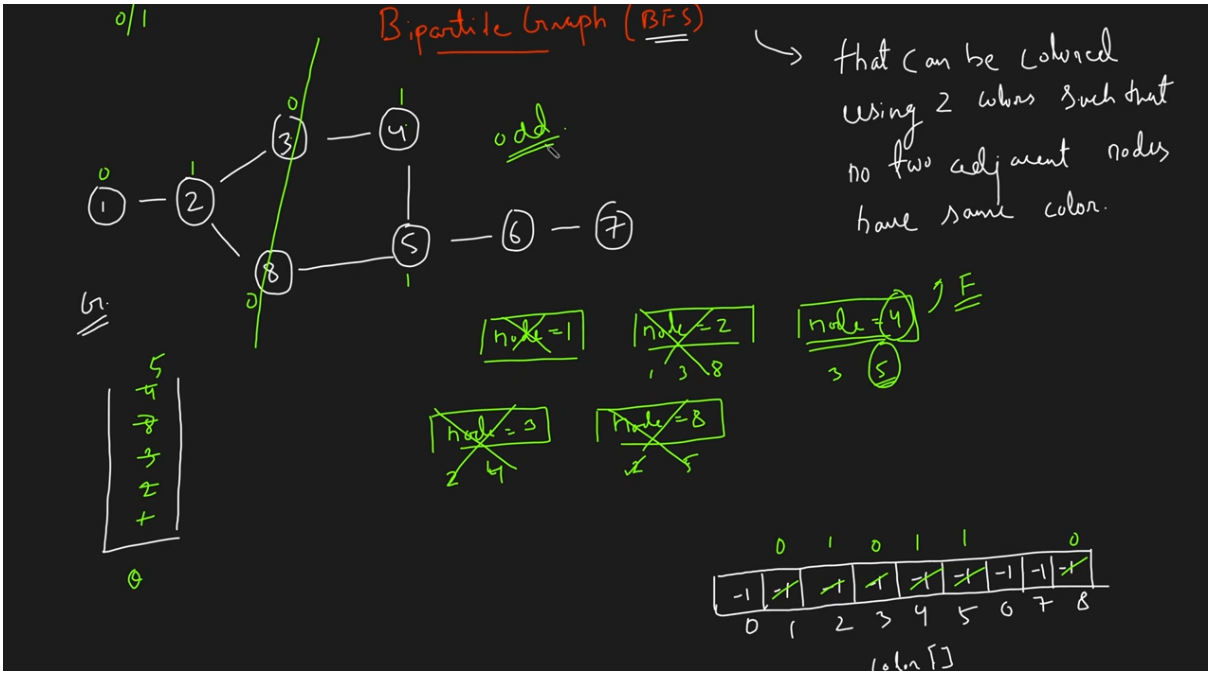


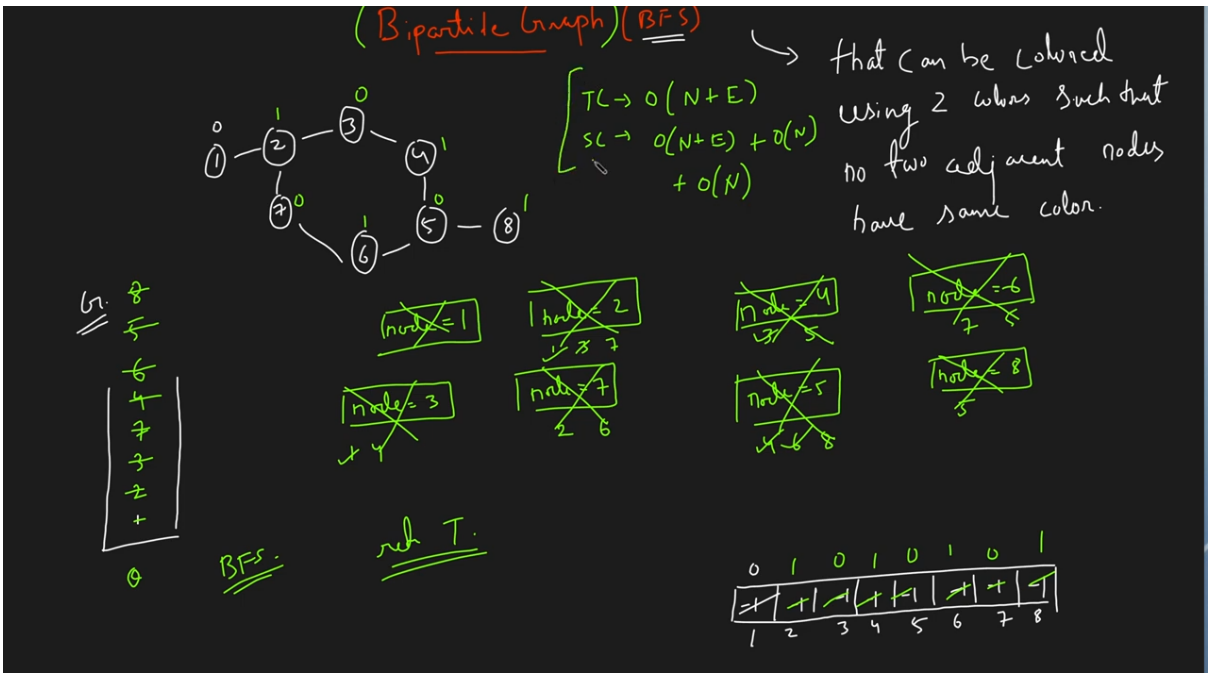
Check whether a given graph is Bipartite or not

Difficulty Level : Medium • Last Updated : 19 Apr, 2021

A Bipartite Graph is a graph whose vertices can be divided into two independent sets, U and V such that every edge (u, v) either connects a vertex from U to V or a vertex from V to U. In other words, for every edge (u, v), either u belongs to U and v to V, or u belongs to V and v to U. We can also say that there is no edge that connects vertices of same set.

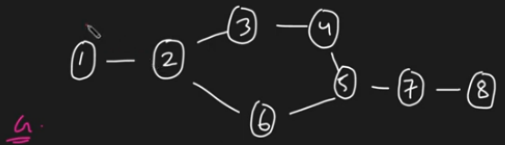



Bipartite Graph (BFS)
that can be colored using 2 colors such that no two adjacent nodes have same color.
odd
node = 1
node = 2
node = 4
node = 3
node = 8
F
-1 -1 -1 -1 -1 -1 -1 -1 -1
0 1 2 3 4 5 6 7 8
color[]


(Bipartite Graph) (BFS)
that can be colored using 2 colors such that no two adjacent nodes have same color.
TC → O(N+E)
SC → O(N+E) + O(N) + O(N)
node = 1
node = 2
node = 4
node = 6
node = 3
node = 7
node = 5
node = 8
BFS.
nah T.
-1 -1 -1 -1 -1 -1 -1 -1 -1
1 2 3 4 5 6 7 8

Bipartite Graph (DFS)

→ that can be colored using 2 colors such that no two adjacent nodes have same color.



G.

DFS(1)

-1	1	-1	-1	-1	-1	-1	-1	-1
0	1	2	3	4	5	6	7	8

color[]

G.

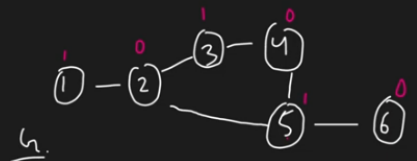
DFS(1)
DFS(2)
DFS(3)
DFS(4)
DFS(5)
DFS(6)
DFS(7)
DFS(8)

1	0	1	0	1	0	1	0	1
0	1	2	3	4	5	6	7	8

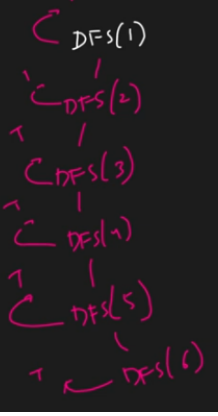
color[]

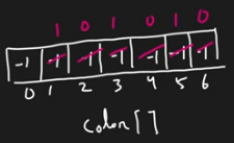
Bipartite Graph (DFS)

↳ that can be colored using 2 colors such that no two adjacent nodes have same color.



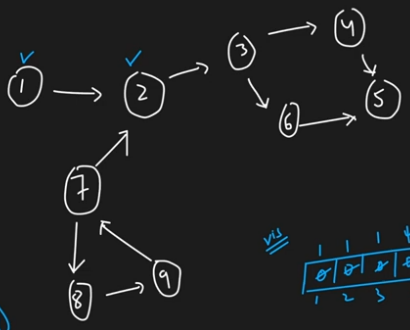
TC → O(N+E)
SC → O(N+E) + O(N) + O(N)





color[]

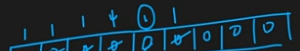
Detect a cycle in Directed Graph (DFS)

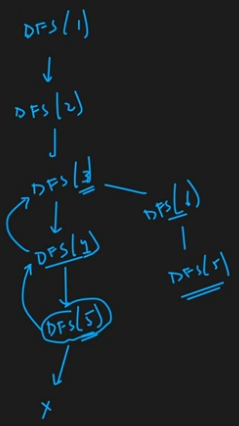


adjacency List

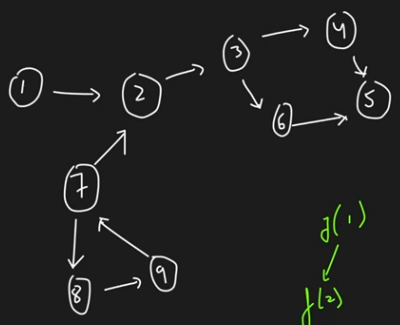
1 → 2
2 → 3
3 → 4, 6
4 → 5
5 → -
6 → 5
7 → 2, 8
8 → 9
9 → 7

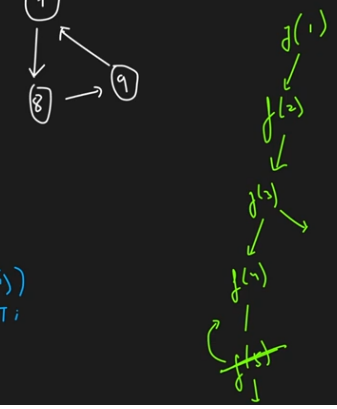
vis





Detect a cycle in Directed Graph (DFS)





adjacency List

1 → 2

2 → 3

3 → 4, 6

4 → 5

5 → =

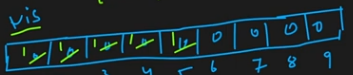
6 → 5

7 → 2, 8

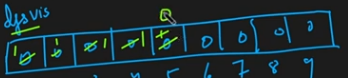
8 → 9

9 → 7

vis

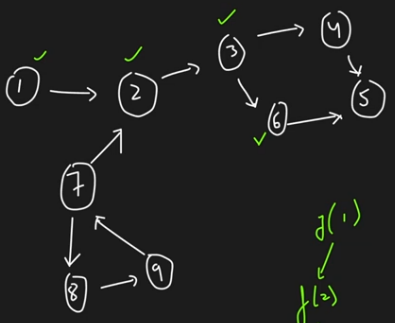


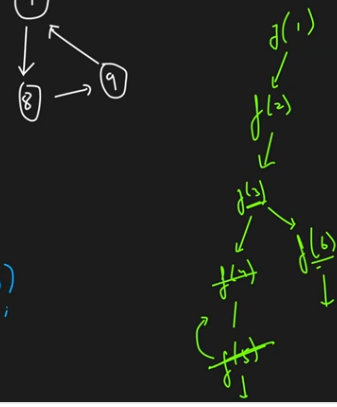
dfsvis




```
for(i = 1 → 9)
{
  if(!vis[i])
  {
    if(cycleCheck(i))
      return T;
  }
}
```


Detect a cycle in Directed Graph (DFS)





adjacency List

1 → 2

2 → 3

3 → 4, 6

4 → 5

5 → =

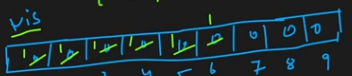
6 → 5

7 → 2, 8

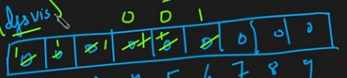
8 → 9

9 → 7

vis

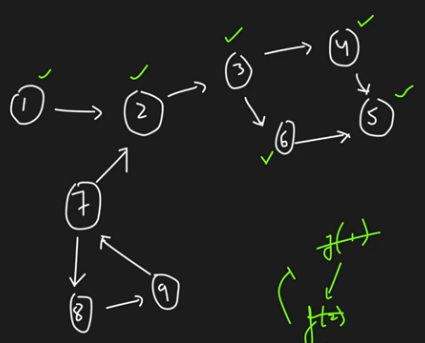


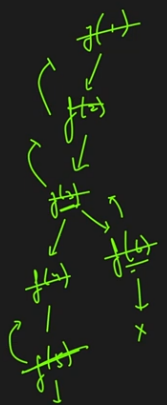
dfsvis




```
for(i = 1 → 9)
{
  if(!vis[i])
  {
    if(cycleCheck(i))
      return T;
  }
}
```

Detect a cycle in Directed Graph (DFS)





adjacency List

1 → 2

2 → 3

3 → 4, 6

4 → 5

5 → =

6 → 5

7 → 2, 8

8 → 9

9 → 7

vis

1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	0	0	0

dfsvis

1	2	3	4	5	6	7	8	9
0	0	0	0	0				

for(i = 1 → 9)

{

if(! vis(i))

{

if(cycleCheck (i) → T

return T;

}

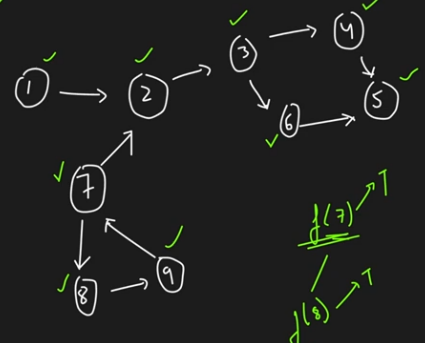
}

Detect a cycle in Directed Graph (DFS)

TC → O(N + E)

SC → O(2N)

ASC → O(N)



f(7) → T

f(8) → T

f(9) → T

vis[node] = 1

dfsvis[node] = 1

adjacency List

1 → 2

2 → 3

3 → 4, 6

4 → 5

5 → =

6 → 5

7 → 2, 8

8 → 9

9 → 7

vis

1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	1	1	1

dfsvis

1	2	3	4	5	6	7	8	9
0	0	0	0	0		1		

for(i = 1 → 9)

{

if(! vis(i))

{

if(cycleCheck (i) → T

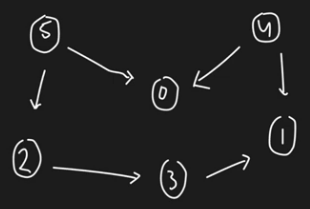
return T;

}

}

Topological Sorting

→ linear ordering of vertices such that if there is an edge $u \to v$, u appears before v in that ordering.



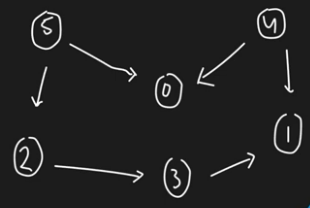
5 4 2 3 1 0

u → v
4 → 0
2 → 3
5 → 0
4 → 1

0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2

Topological Sorting (DFS)

→ linear ordering of vertices such that if there is an edge $u \to v$, u appears before v in that ordering.



dfs(0)
dfs(1)
dfs(4)
dfs(3)

adj

0 — =
1 — =
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2

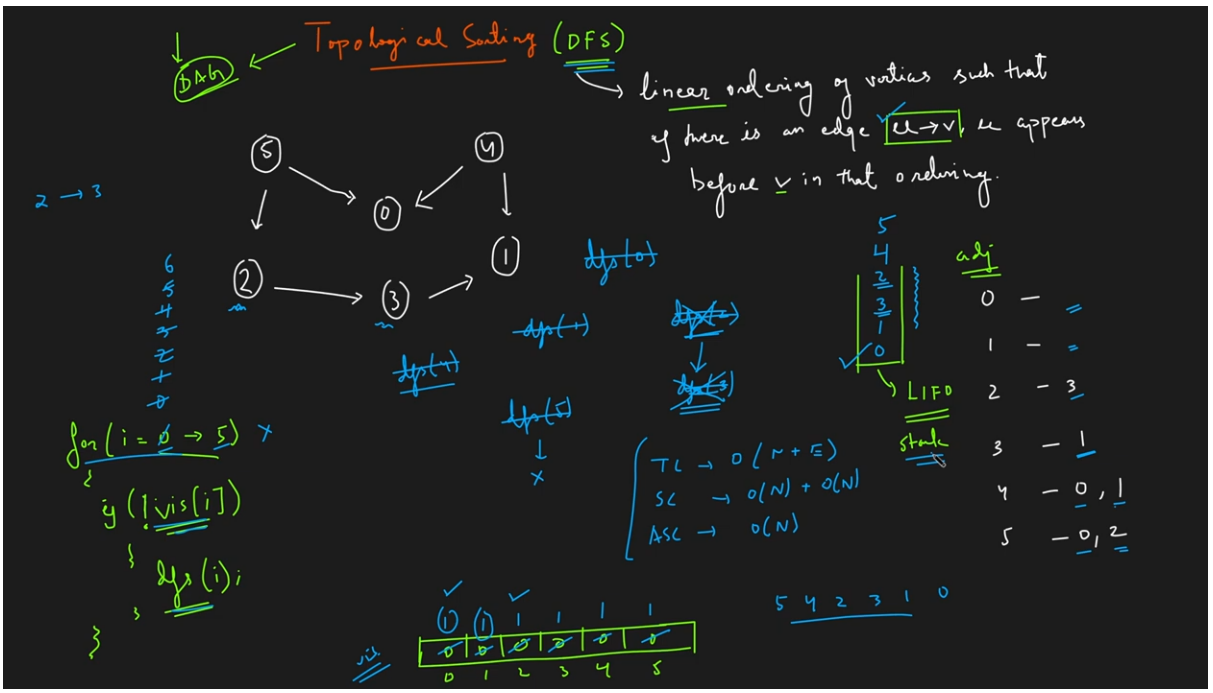
3
1
0

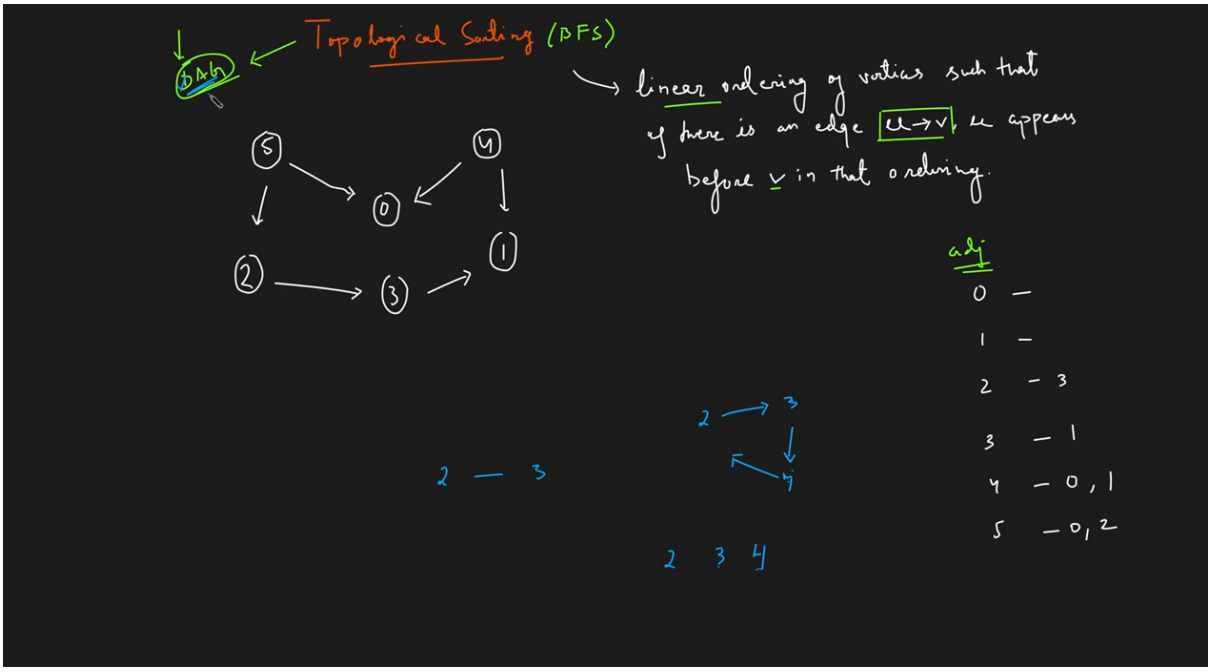
→ LIFO
stack

```
for (i = 0 → 5)
    if (!vis[i])
        dfs(i);
```

vis

0	1	2	3	4	5
0	0	0	0	0	0
1	(1)	1			

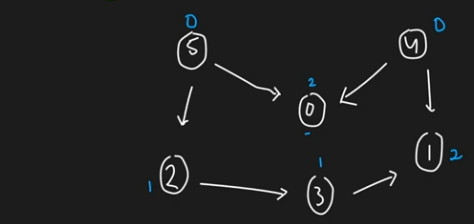

Topological Sorting (DFS)
DAG
linear ordering of vertices such that if there is an edge u → v, u appears before v in that ordering.
2 → 3
for (i = 0 → 5)
if (!vis[i])
dfs(i);
LIFO
stack
adj
0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2
TC → O(N + E)
SC → O(N) + O(N)
ASC → O(N)
5 4 2 3 1 0


Topological Sorting (DFS)
DAG
linear ordering of vertices such that if there is an edge u → v, u appears before v in that ordering.
adj
0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2
2 — 3
2 3 4

Topological Sorting (BFS)

DAG

→ linear ordering of vertices such that if there is an edge $u \to v$, u appears before v in that ordering.



4

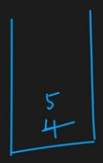
2	2	1	1	0	0
0	1	2	3	4	5

Indegree

adj

0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2

node = 4



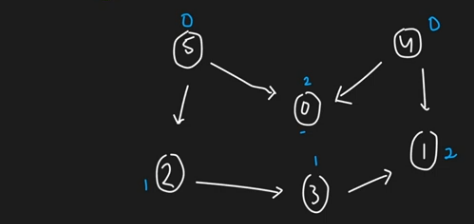
0
↳ FIFO

Topological Sorting (BFS)

DAG

→ linear ordering of vertices such that if there is an edge $u \to v$, u appears before v in that ordering.

4 5



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

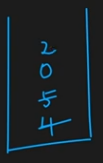
Indegree

adj

0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2

node = 5

→ 0, 2

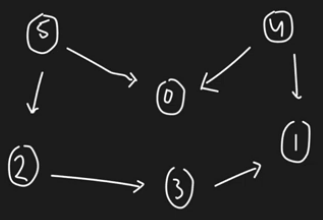


0
↳ FIFO

Topological Sorting (BFS)

DAG

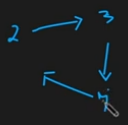
→ linear ordering of vertices such that if there is an edge $u \to v$, u appears before v in that ordering.



adj

0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2

2 — 3

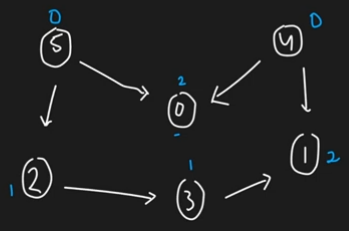


2 3 4

Topological Sorting (BFS)

DAG

→ linear ordering of vertices such that if there is an edge $u \to v$, u appears before v in that ordering.



2 2 1 1 0 0

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

Indegree

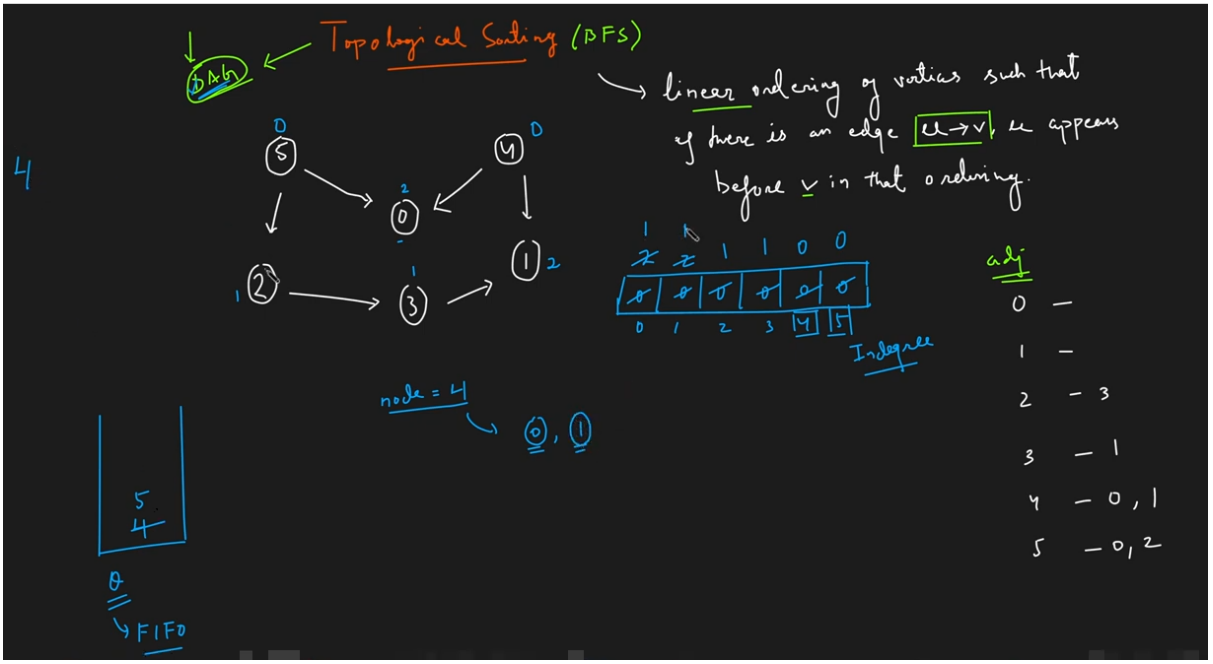
node = 4

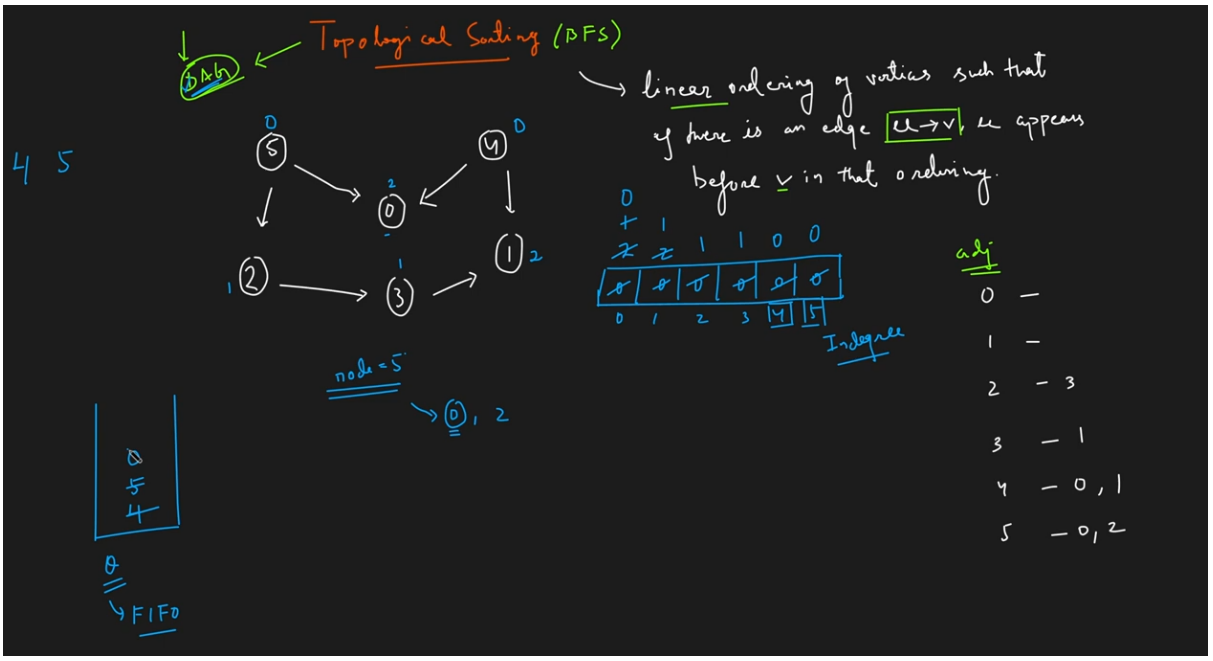
0

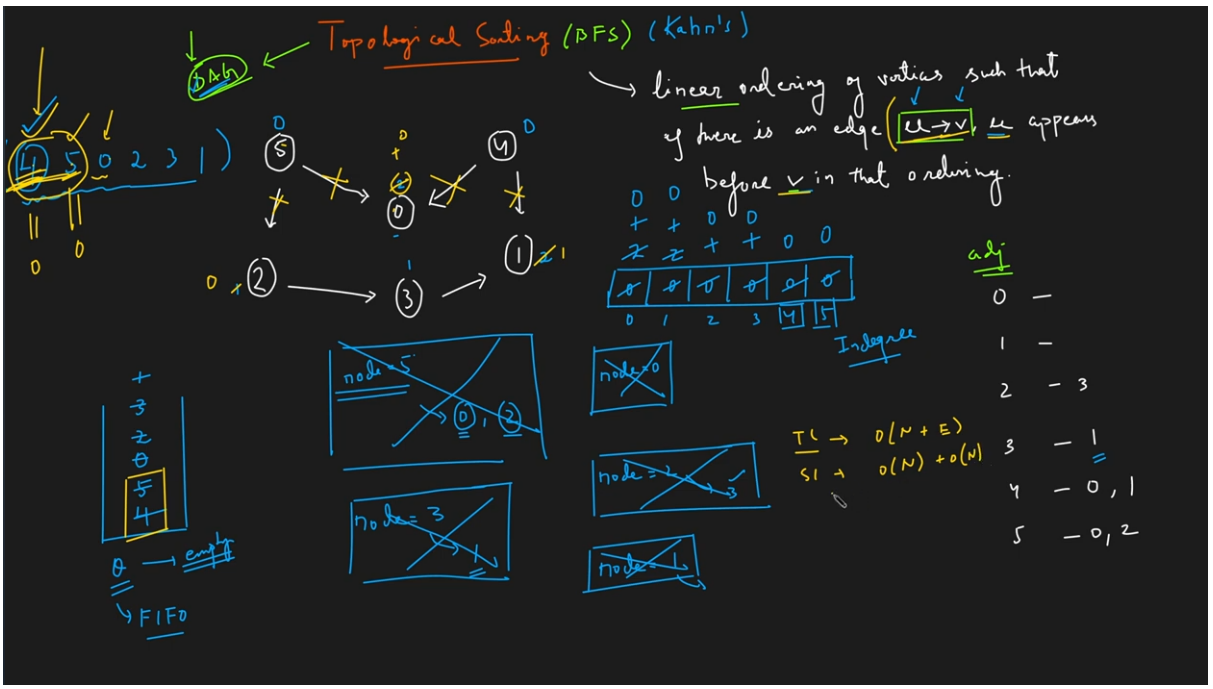
↳ FIFO

adj

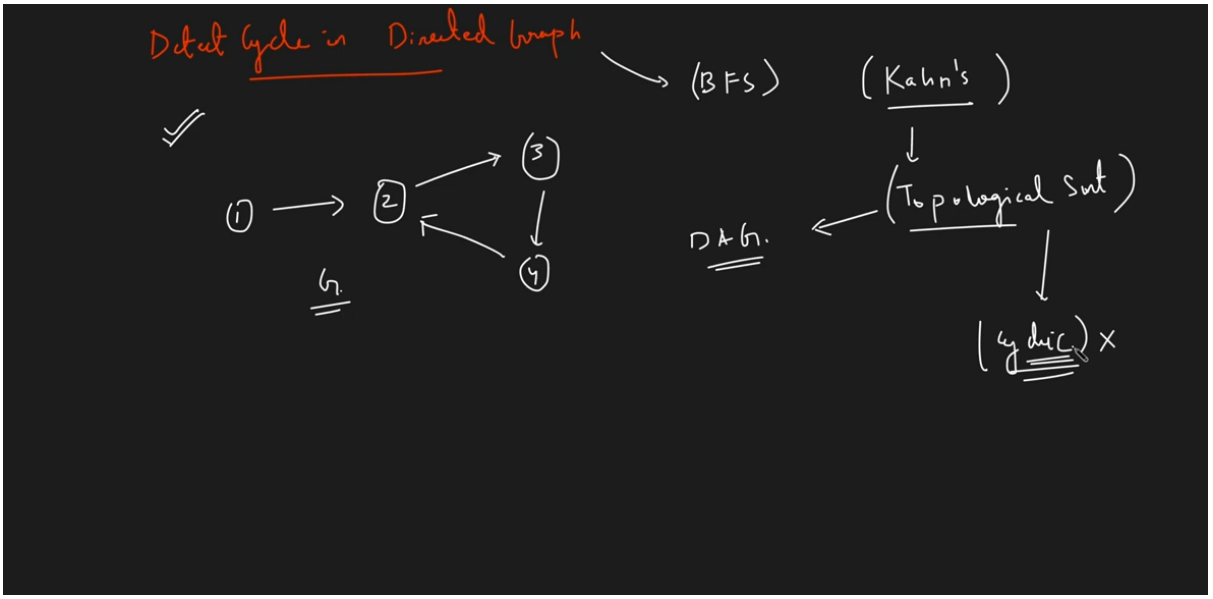
0 —
1 —
2 — 3
3 — 1
4 — 0, 1
5 — 0, 2


Topological Sorting (DFS)
DAG
linear ordering of vertices such that if there is an edge u→v, u appears before v in that ordering.
4
node = 4
0, 1
5
4
0
FIFO
adj
0 –
1 –
2 – 3
3 – 1
4 – 0, 1
5 – 0, 2
Indegree


Topological Sorting (DFS)
DAG
linear ordering of vertices such that if there is an edge u→v, u appears before v in that ordering.
4 5
node = 5
0, 2
0
5
4
0
FIFO
adj
0 –
1 –
2 – 3
3 – 1
4 – 0, 1
5 – 0, 2
Indegree

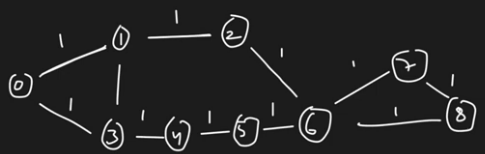



```cpp
class Solution{
    public:
    vector<int> topoSort(int N, vector<int> adj[]) {
        queue<int> q;
        vector<int> indegree(N, 0);
        for(int i = 0;i<N;i++) {
            for(auto it: adj[i]) {
                indegree[it]++;
            }
        }

        for(int i = 0;i<N;i++) {
            if(indegree[i] == 0) {
                q.push(i);
            }
        }
        vector<int> topo;
        while(!q.empty()) {
            int node = q.front();
            q.pop();
            topo.push_back(node);
            for(auto it : adj[node]) {
                indegree[it]--;
                if(indegree[it] == 0) {
                    q.push(it);
                }
            }
        }
        return topo;
    }
};
```




```cpp
class Solution {
public:
    bool isCyclic(int N, vector<int> adj[]) {
        queue<int> q;
        vector<int> indegree(N, 0);
        for(int i = 0;i<N;i++) {
            for(auto it: adj[i]) {
                indegree[it]++;
            }
        }

        for(int i = 0;i<N;i++) {
            if(indegree[i] == 0) {
                q.push(i);
            }
        }
        int cnt = 0;
        while(!q.empty()) {
            int node = q.front();
            q.pop();
            cnt++;
            for(auto it : adj[node]) {
                indegree[it]--;
                if(indegree[it] == 0) {
                    q.push(it);
                }
            }
        }
        if(cnt == N) return false;
        return true;
    }
};
// } Driver Code Ends
```

Shortest Distance in Undirected Graph



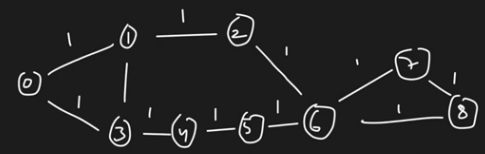
src = 0

src des
0 — 6 → 3

adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

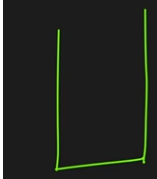
Shortest Distance in Undirected Graph



src = 0

adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

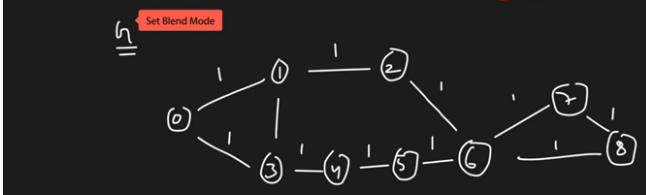


Q → node

dist

∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	2	3	4	5	6	7	8

Shortest Distance in Undirected Graph



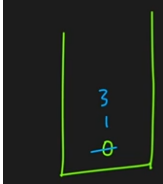
src = 0

adj

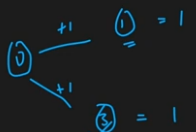
0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

node = 0
dis = 0

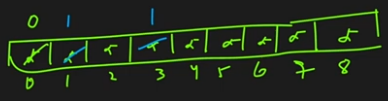
1 , 3



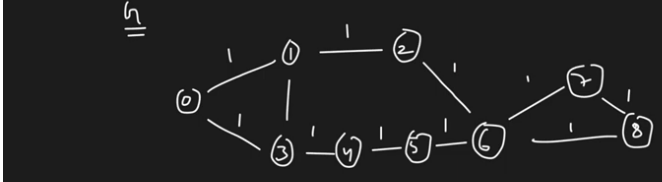
Q → node.
↳ FIFO.



dist



Shortest Distance in Undirected Graph

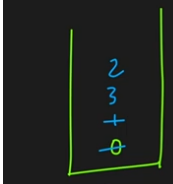


src = 0

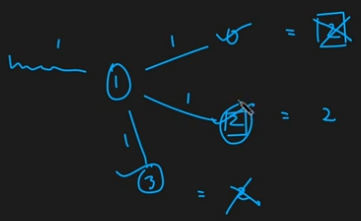
adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

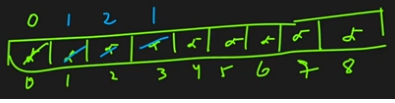
node = 1
dis = 1



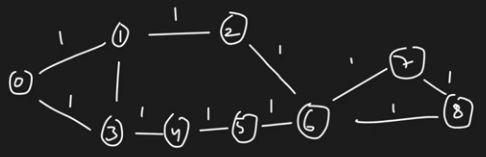
Q → node.
↳ FIFO.



dist



Shortest Distance in Undirected Graph



src = 0

adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

node = 3

Q → node

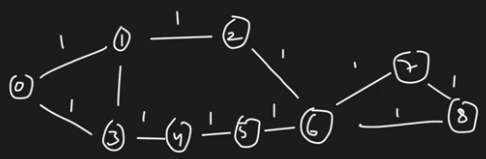
↳ FIFO

dist

0 1 2 1 2 ∞ ∞ ∞ ∞

0 1 2 3 4 5 6 7 8

Shortest Distance in Undirected Graph



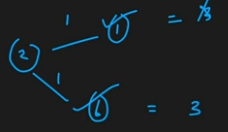
src = 0

adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

node = 2

src 2



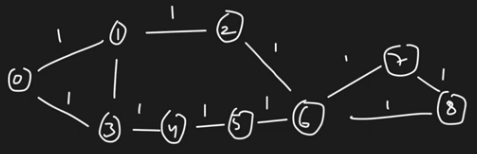
Q → node

↳ FIFO

dist

0 1 2 1 2 3 ∞ ∞ ∞

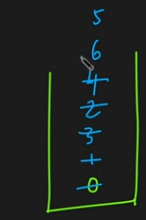
Shortest Distance in Undirected Graph



src = 0

adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7

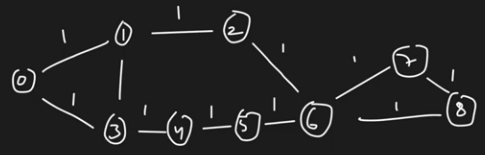


Q → node
↳ FIFO

dest

0	1	2	1	2	3	3	
0	1	2	3	4	5	6	7	8

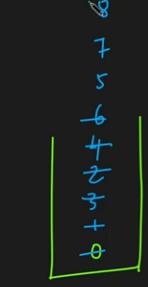
Shortest Distance in Undirected Graph



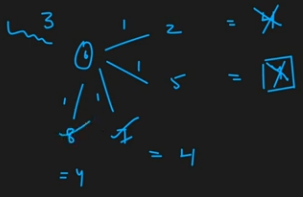
src = 0

adj

0 → 1, 3
1 → 0, 2, 3
2 → 1, 6
3 → 0, 4
4 → 3, 5
5 → 4, 6
6 → 2 5 7 8
7 → 6, 8
8 → 6, 7



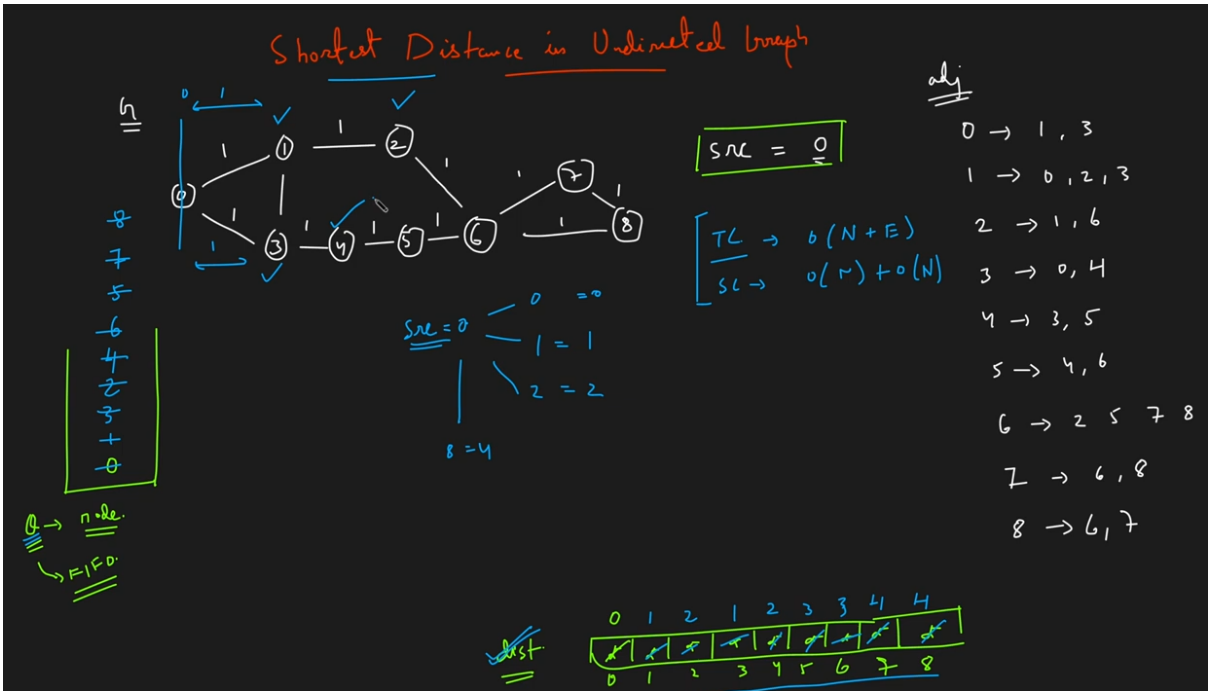
node = 6



Q → node
↳ FIFO

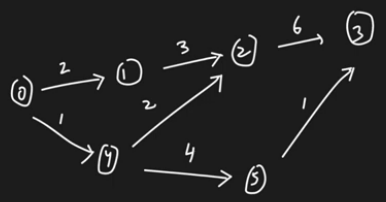
dest

0	1	2	1	2	3	3	4	4
0	1	2	3	4	5	6	7	8




```cpp
void BFS(vector<int> adj[], int N, int src)
{
    int dist[N];
    for(int i = 0;i<N;i++) dist[i] = INT_MAX;
    queue<int>  q;

    dist[src] = 0;
    q.push(src);

    while(q.empty()==false)
    {
        int node = q.front();
        q.pop();

        for(auto it:adj[node]){
            if(dist[node] + 1 < dist[it]){
                dist[it]=dist[node]+1;
                q.push(it);
            }
        }
    }
    for(int i = 0;i<N;i++) cout << dist[i] << " ";

}
```

Shortest Path in a weighted DAG



src = 0

vector<int> adj[ ]

pair<int, int>

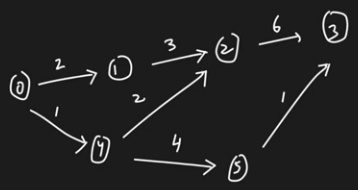
src dis

0 - 3 = 6 (0 →1 4 →4 5 →1 3)

class Pair

{

Shortest Path in a weighted DAG



src = 0

adj List.

0 → (1,2) , (4,1)

1 → (2,3)

2 → (3,6)

3 →

4 → (2,2) , (5,4)

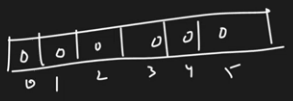
5 → (3,1)

for(i = 0 → 5)

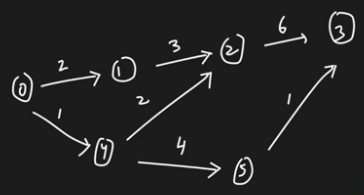
{

if (! vis[i])

dfs(i)



Shortest Path in a weighted DAG



src = 0

Adj List.

0 → (1, 2) , (4, 1)

1 → (2, 3)

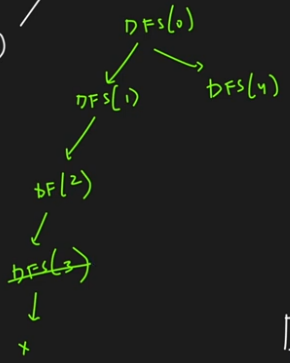
2 → (3, 6)

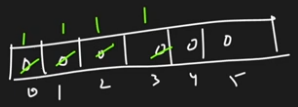
3 →

4 → (2, 2) , (5, 4)

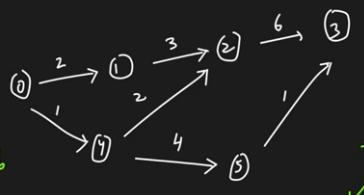
5 → (3, 1)


```
for( i = 0 → 5)
{
    if (!vis[i])
        dfs(i)
}
```





Shortest Path in a weighted DAG



src = 0

Adj List.

0 → (1, 2) , (4, 1)

1 → (2, 3)

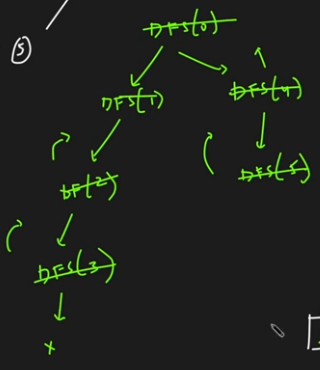
2 → (3, 6)

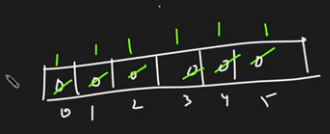
3 →

4 → (2, 2) , (5, 4)

5 → (3, 1)

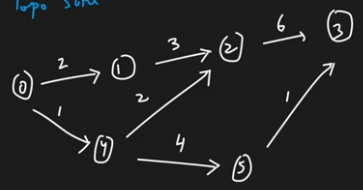

```
for( i = 0 → 5)
{
    if (!vis[i])
        dfs(i)
}
```





Shortest Path in a weighted DAG

step1 : find Topo Sort



src = 0

adj List.

0 → (1, 2) , (4, 1)

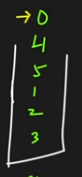
1 → (2, 3)

2 → (3, 6)

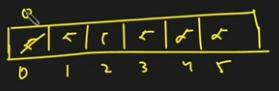
3 →

4 → (2, 2) , (5, 4)

5 → (3, 1)

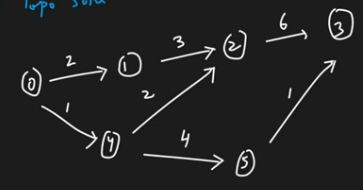


LIFO



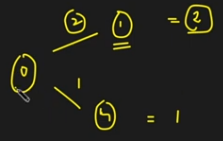
Shortest Path in a weighted DAG

step1 : find Topo Sort



src = 0

node = 0



if(dis[node] != -1)

adj List.

0 → (1, 2) , (4, 1)

1 → (2, 3)

2 → (3, 6)

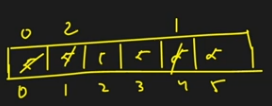
3 →

4 → (2, 2) , (5, 4)

5 → (3, 1)

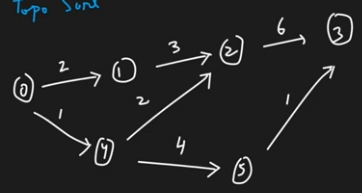
LIFO

dis



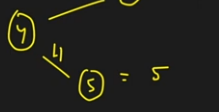
Shortest Path in a weighted DAG

step1 : find Topo Sort



src = 0

node = 4



if (dis [node] != ∞)

adj List.

0 → (1, 2) , (4, 1)

1 → (2, 3)

2 → (3, 6)

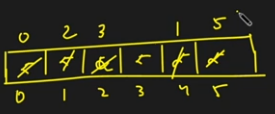
3 →

4 → (2, 2) , (5, 4)

5 → (3, 1)

St

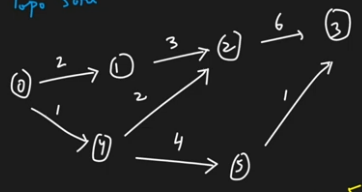
Topo



dis

Shortest Path in a weighted DAG

step1 : find Topo Sort



src = 0

if (dis [node] != ∞)

adj List.

0 → (1, 2) , (4, 1)

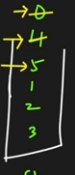
1 → (2, 3)

2 → (3, 6)

3 →

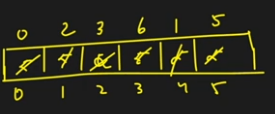
4 → (2, 2) , (5, 4)

5 → (3, 1)



St

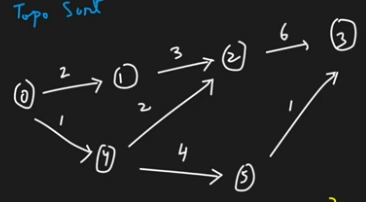
Topo



dis

Shortest Path in a weighted DAG

step1 : find Topo Sort



src = 0

adj List.

0 → (1, 2) , (4, 1)

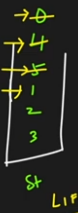
1 → (2, 3)

2 → (3, 6)

3 →

4 → (2, 2) , (5, 4)

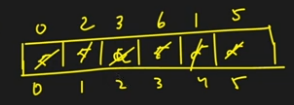
5 → (3, 1)



LIFO

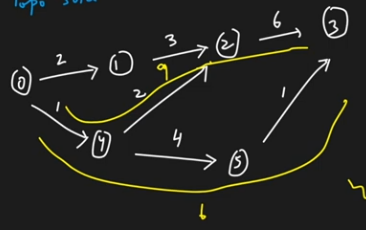
if(dis[node] != inf)

dis



Shortest Path in a weighted DAG

step1 : find Topo Sort



src = 0

adj List.

0 → (1, 2) , (4, 1)

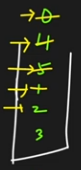
1 → (2, 3)

2 → (3, 6)

3 →

4 → (2, 2) , (5, 4)

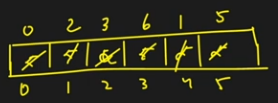
5 → (3, 1)

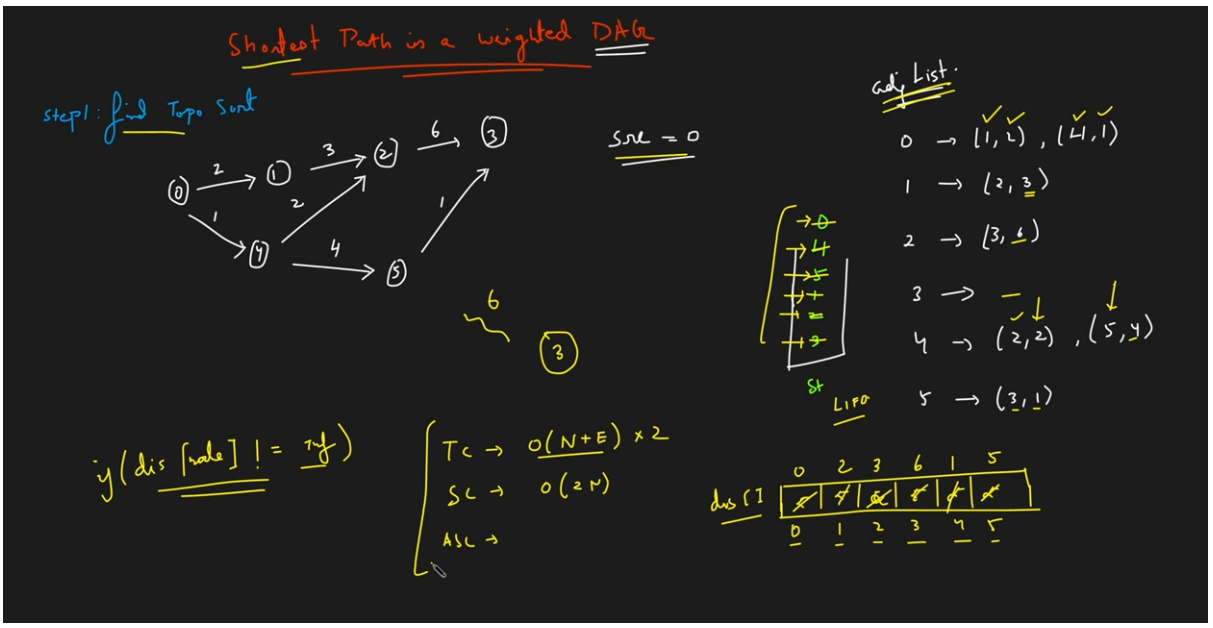


LIFO

if(dis[node] != inf)

dis



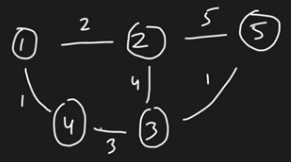



```cpp
int main()
{
    int n, m;
    cin >> n >> m;
    vector<pair<int,int>> adj[n];
    for(int i = 0;i<m;i++) {
        int u, v, wt;
        cin >> u >> v >> wt;
        adj[u].push_back({v, wt});
    }

    shortestPath(0, n, adj);

    return 0;
}
```

```cpp
void shortestPath(int src, int N, vector<pair<int,int>> adj[])
{
    int vis[N] = {0};
    stack<int> st;
    for (int i = 0; i < N; i++)
        if (!vis[i])
            findTopoSort(i, vis, st, adj);

    int dist[N];
    for (int i = 0; i < N; i++)
        dist[i] = 1e9;
    dist[src] = 0;

    while(!st.empty())
    {
        int node = st.top();
        st.pop();

        // if the node has been reached previously
        if (dist[node] != INF) {
            for(auto it : adj[node]) {
                if(dist[node] + it.second < dist[it.first]) {
                    dist[it.first] = dist[node] + it.second;
                }
            }
        }
    }

    for (int i = 0; i < N; i++)
        (dist[i] == 1e9)? cout << "INF ": cout << dist[i] << " ";
}
```

Dijkstra's Algorithm

src = 1



(1 - 2) 2

(1 - 3) 4

(1 - 4) 1

(1 - 5) 5

adj list

1 → (2,2), (4,1)

2 → (1,2), (5,5), (3,4)

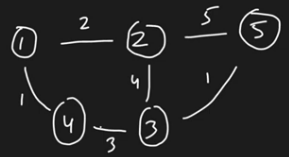
3 → (2,4), (4,3), (5,1)

4 → (1,1), (3,3)

5 → (2,5), (3,1)

Dijkstra's Algorithm

src = 1



adj list

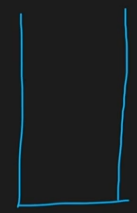
1 → (2,2), (4,1)

2 → (1,2), (5,5), (3,4)

3 → (2,4), (4,3), (5,1)

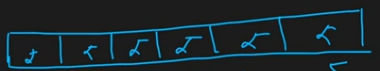
4 → (1,1), (3,3)

5 → (2,5), (3,1)



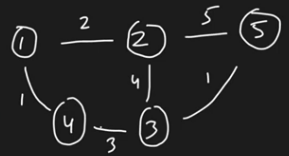
(dis, node)

PQ → Min



Dijkstra's Algorithm

src = 1



Gr.

adj list

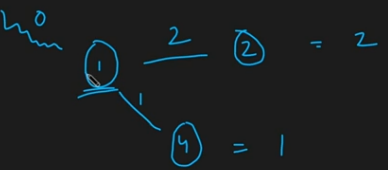
1 → (2,2), (4,1)

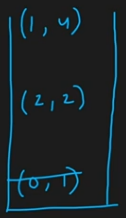
2 → (1,2), (5,5), (3,4)

3 → (2,4), (4,3), (5,1)

4 → (1,1), (3,3)

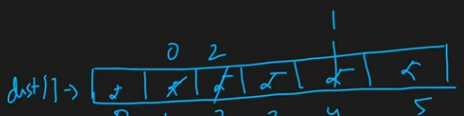
5 → (2,5), (3,1)





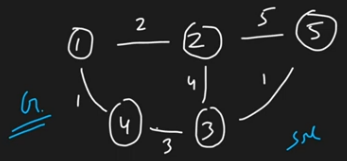
(dis, node)

PQ → Min



Dijkstra's Algorithm

src = 1



G:

src

adj list

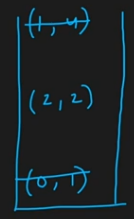
1 → (2,2), (4,1)

2 → (1,2), (5,5), (3,4)

3 → (2,4), (4,3), (5,1)

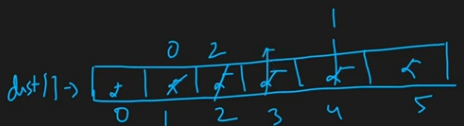
4 → (1,1), (3,3)

5 → (2,5), (3,1)



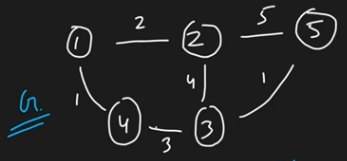
(dis, node)

PQ → Min



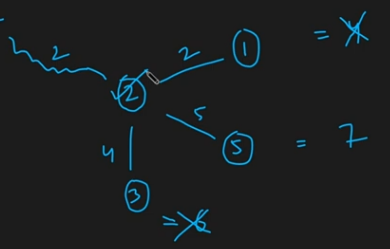
Dijkstra's Algorithm

src = 1



G:

src



adj list

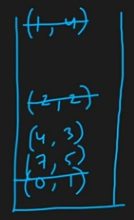
1 → (2,2), (4,1)

2 → (1,2), (5,5), (3,4)

3 → (2,4), (4,3), (5,1)

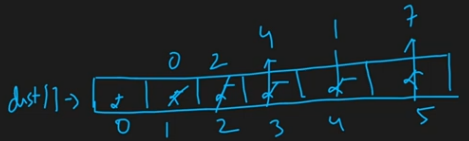
4 → (1,1), (3,3)

5 → (2,5), (3,1)



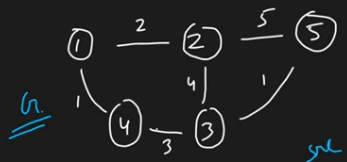
(dis, node)

PQ → Min



Dijkstra's Algorithm

src = 1



G

adj list

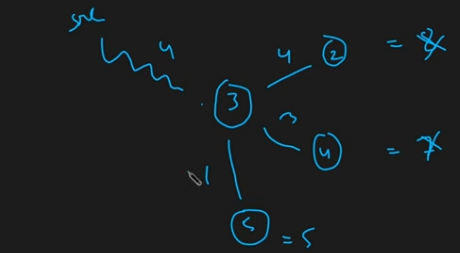
1 → (2,2), (4,1)

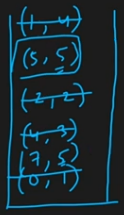
2 → (1,2), (5,5), (3,4)

3 → (2,4), (4,3), (5,1)

4 → (1,1), (3,3)

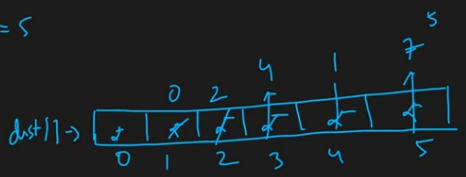
5 → (2,5), (3,1)





(dis, node)

PQ → Min



Dijkstra's Algorithm

src = 1

adj list

1 → (2,2), (4,1)

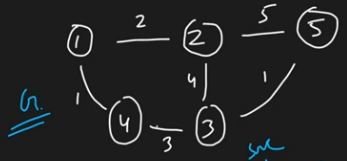
2 → (1,2), (5,5), (3,4)

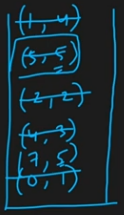
3 → (2,4), (4,3), (5,1)

4 → (1,1), (3,3)

5 → (2,5), (3,1)

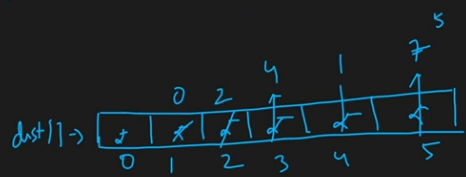
G

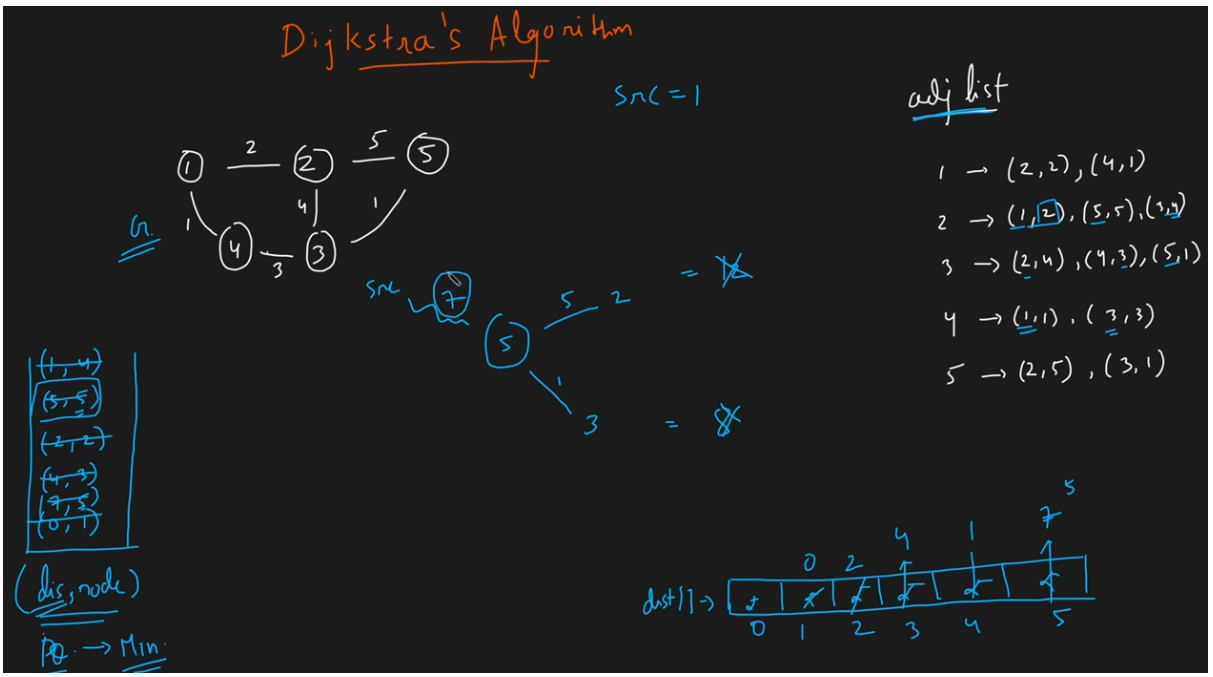


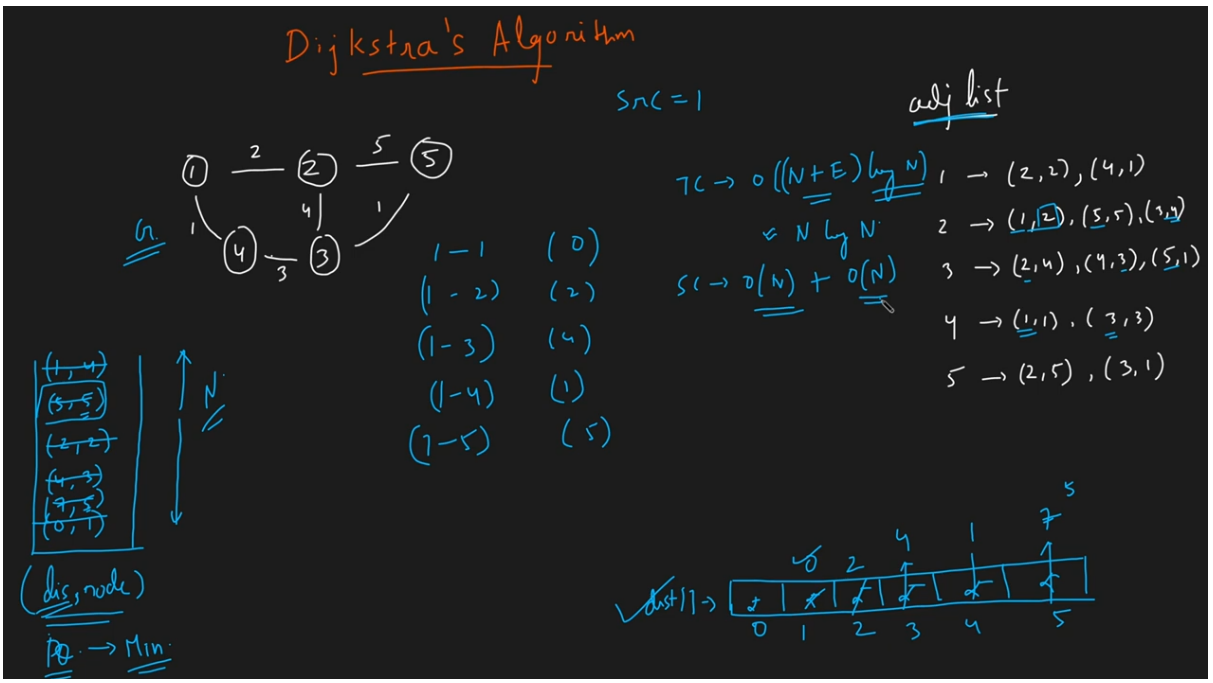


(dis, node)

PQ → Min

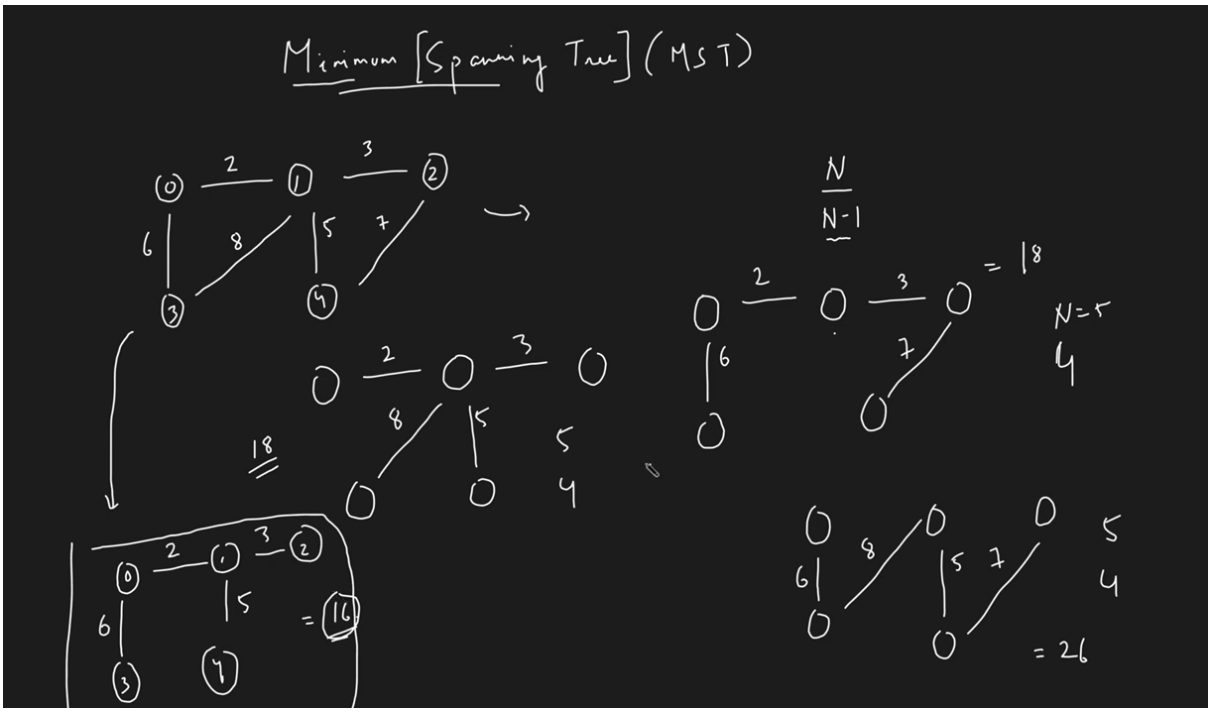




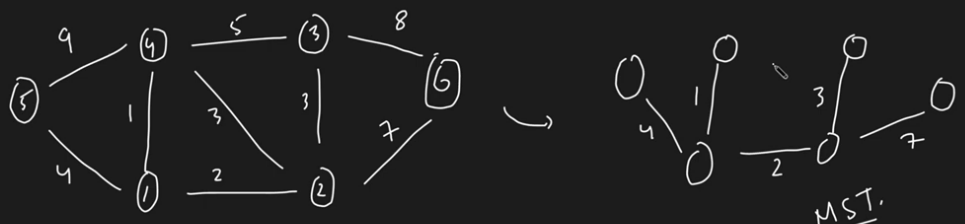



```cpp
#include<bits/stdc++.h>
using namespace std;

int main(){
    int n,m,source;
    cin >> n >> m;
    vector<pair<int,int> > g[n+1];  // 1-indexed adjacency list for of graph

    int a,b,wt;
    for(int i = 0; i<m ; i++){
        cin >> a >> b >> wt;
        g[a].push_back(make_pair(b,wt));
        g[b].push_back(make_pair(a,wt));
    }

    cin >> source;
```

```cpp
// Dijkstra's algorithm begins from here
priority_queue<pair<int,int>,vector<pair<int,int> >,greater<pair<int,int> > > pq;// min-heap ; In pair => (dist
vector<int> distTo(n+1,INT_MAX);    // 1-indexed array for calculating shortest paths;

distTo[source] = 0;
pq.push(make_pair(0,source));   // (dist,from)

while( !pq.empty() ){
    int dist = pq.top().first;
    int prev = pq.top().second;
    pq.pop();

    vector<pair<int,int> >::iterator it;
    for( auto it: g[prev]){
        int next = it.first;
        int nextDist = it.second;
        if( distTo[next] > dist + nextDist){
            distTo[next] = distTo[prev] + nextDist;
            pq.push(make_pair(distTo[next], next));
        }
    }

}

cout << "The distances from source, " << source << ", are : \n";
for(int i = 1 ; i<=n ; i++) cout << distTo[i] << " ";
cout << "\n";

return 0;
```



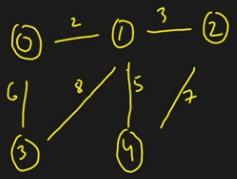
Minimum [Spanning Tree] (MST)

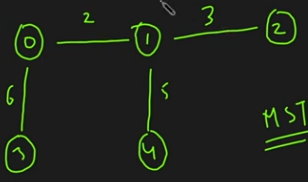


(17)

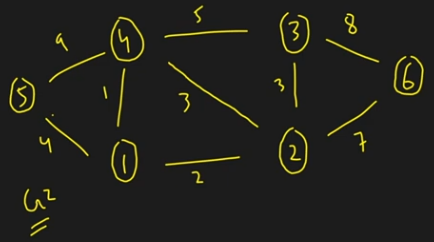
:) Prim
(:) Kruskal

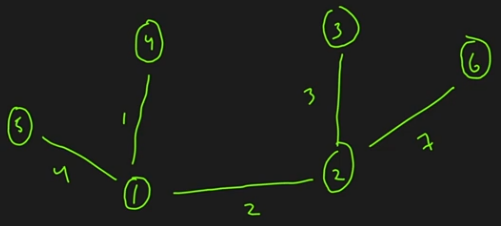
Prim's Algorithm (MST)



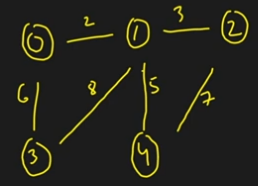


N=5





Prim's Algorithm (MST)



key

0	∞	∞	∞	∞
0	1	2	3	4

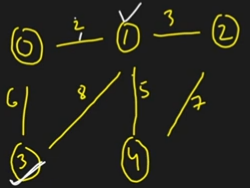
mst

F	F	F	F	F
0	1	2	3	4

parent

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

Prim's Algorithm (MST)



node = 0

1

3

key

0	~~∞~~ 2	∞	~~∞~~ 6	∞
0	1	2	3	4

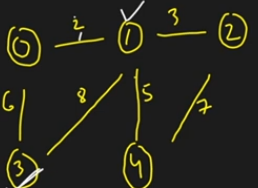
mst T

~~F~~ T	F	F	F	F
0	1	2	3	4

parent 0 0

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

Prim's Algorithm (MST)



node = 1

~~1~~

~~3~~ → 8

4 → 5

2

key →

0	~~2~~	~~3~~ 6	~~6~~ 5	∞
0	1	2	3	4

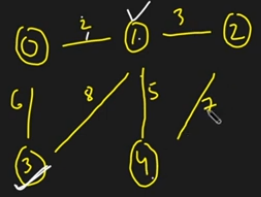
mst T T

~~F~~ T	~~F~~ T	F	F	F
0	1	2	3	4

parent 0 1 0 1

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

Prim's Algorithm (MST)



node = 2

4 → 7

key

0	1	2	3	4
0	~~X~~	X	~~7~~	~~5~~

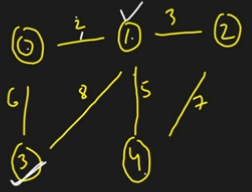
mst

0	1	2	3	4
T	T	T	F	F

parent

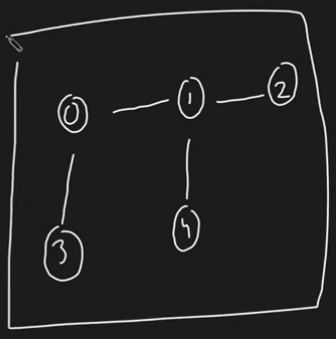
0	1	2	3	4
-1	~~0~~	~~1~~	~~0~~	~~1~~

Prim's Algorithm (MST)



node = 4

node = 3



key

0	1	2	3	4
0	~~X~~	X	~~7~~	~~5~~

mst

0	1	2	3	4
T	T	T	T	T

parent

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

0 (1 2 3 4)

Disjoint Set

n = 7

→ findPar()
→ Union()

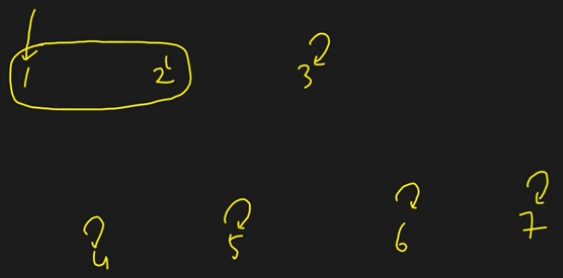
Union(1, 2)
Union(2, 3)
Union(4, 5)
Union(6, 7)
Union(5, 6)
Union(3, 7)

Disjoint Set

n = 7

→ findPar(2) = 1
→ Union()

Union(1, 2)
Union(2, 3)
Union(4, 5)
Union(6, 7)
Union(5, 6)
Union(3, 7)

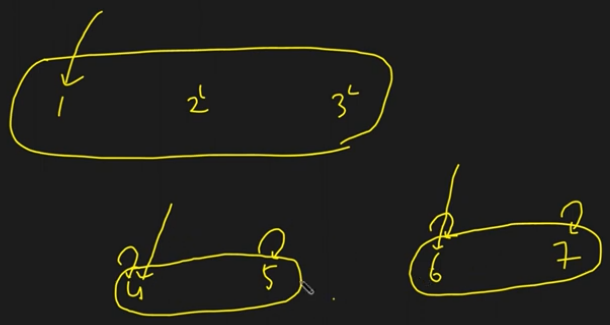


Disjoint Set

n = 7

→ findPar()
→ Union()

Union(1, 2)
Union(2, 3)
Union(4, 5)
Union(6, 7)
Union(5, 6)
Union(3, 7)

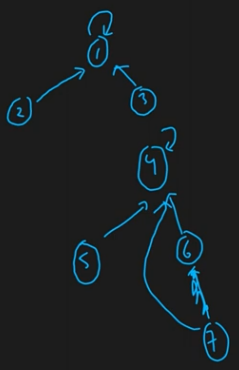


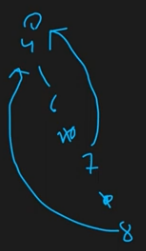
Disjoint Set → Union By Rank & (Path Compression)

→ findPar()
→ Union()

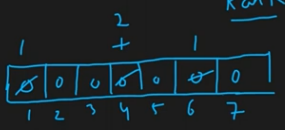
Union(1, 2)
Union(2, 3)
Union(4, 5)
Union(6, 7)
Union(5, 6)
Union(3, 7)

Union(-, 7)





Rank



3 → 1

7 → 6 → 4

7 → 6 → 4

8 → 7 → 6 → 4

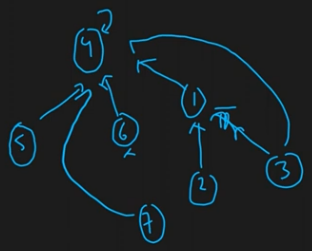
Disjoint Set → Union By Rank & (Path Compression)

O(4α) ≈ O(4)

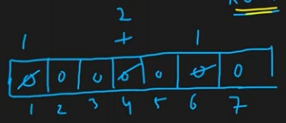
→ findPar()
→ Union()

Union(1, 2)
Union(2, 3)
Union(4, 5)
Union(6, 7)
Union(5, 6)
Union(3, 7)

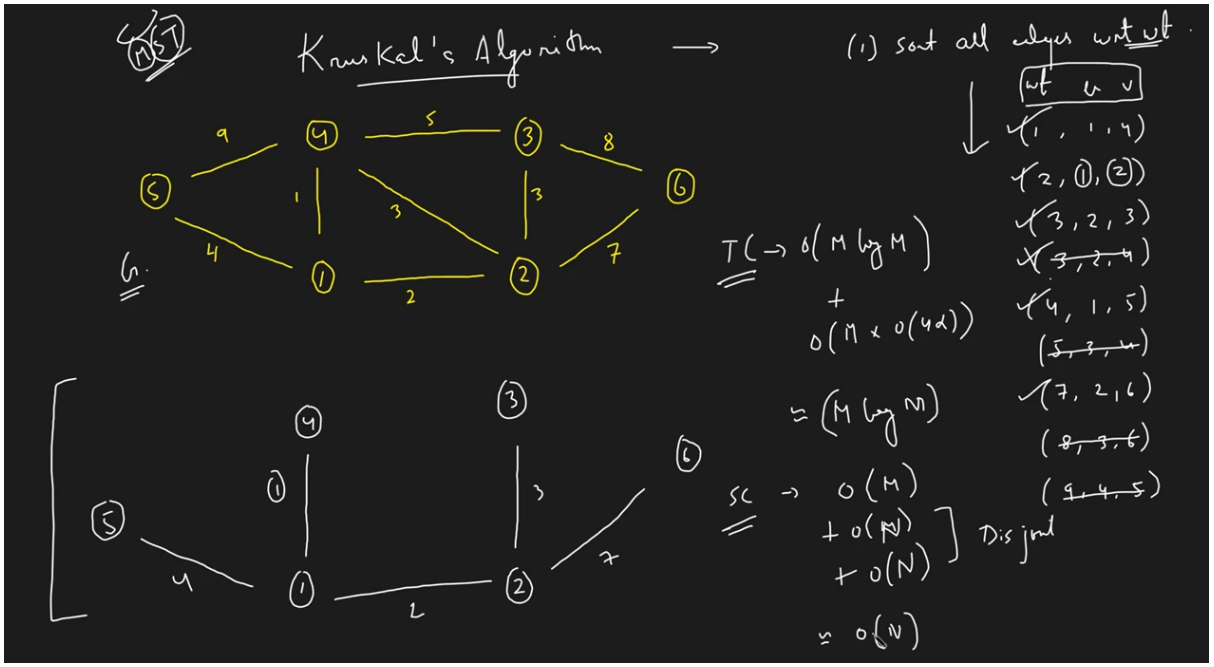
m



Rank

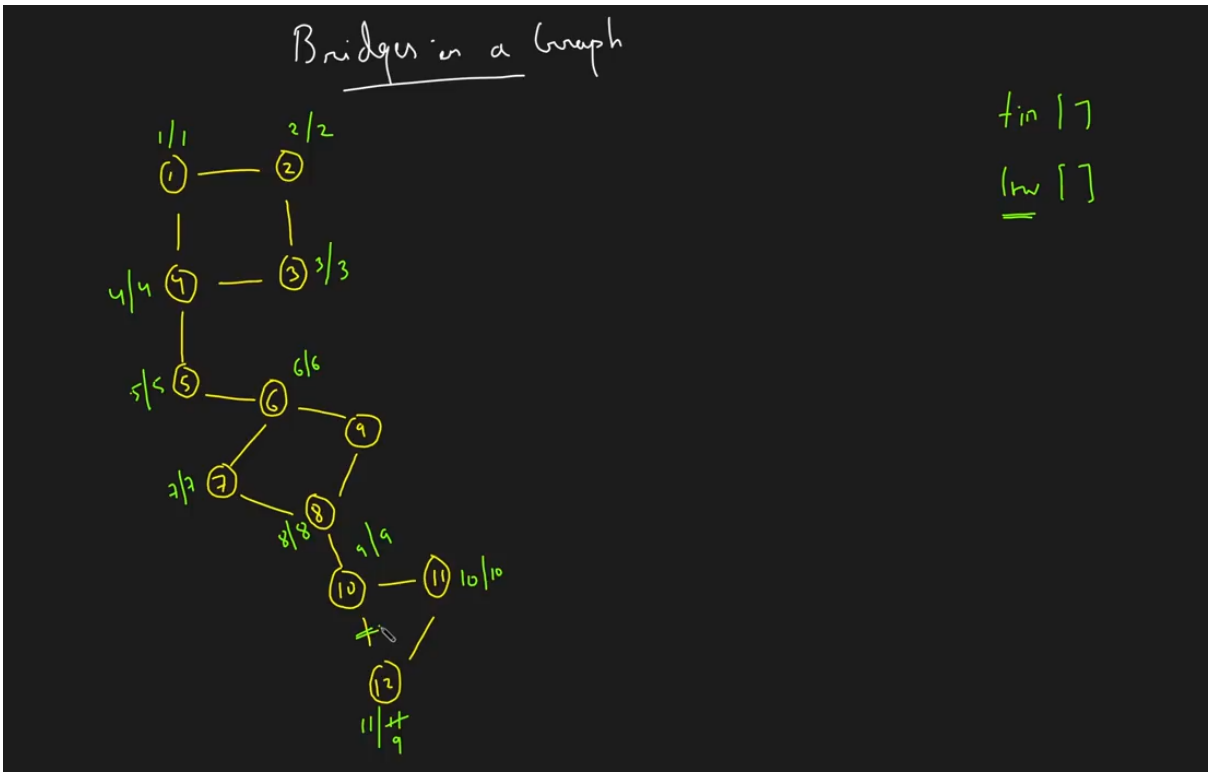


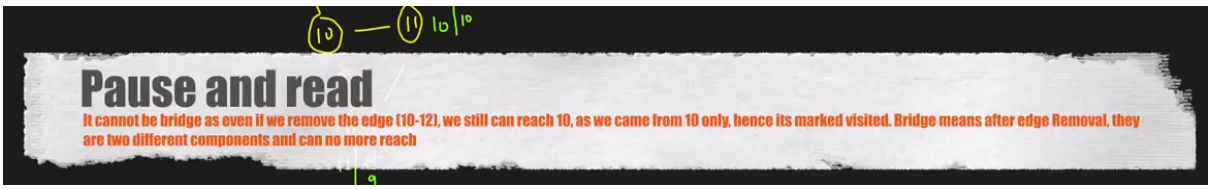
3 → 1 → 4

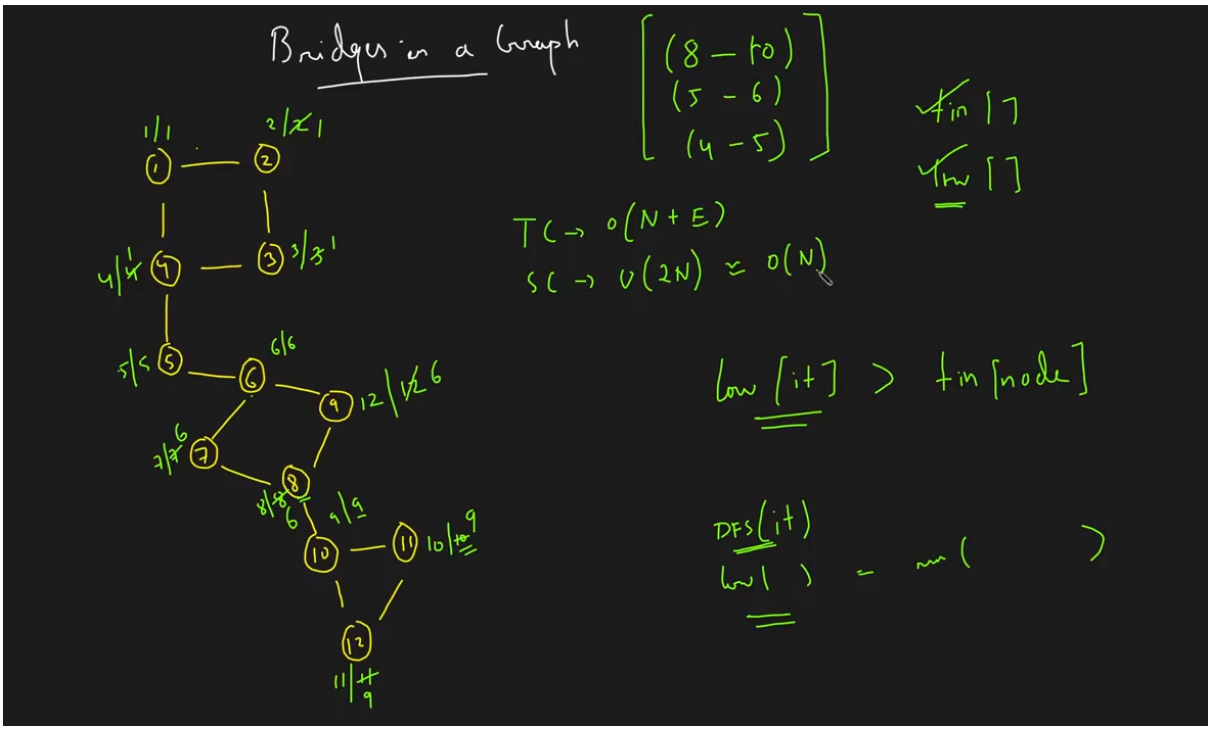



```cpp
int main(){
    int N,m;
    cin >> N >> m;
    vector<node> edges;
    for(int i = 0;i<m;i++) {
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back(node(u, v, wt));
    }
    sort(edges.begin(), edges.end(), comp);

    vector<int> parent(N);
    for(int i = 0;i<N;i++)
        parent[i] = i;
    vector<int> rank(N, 0);

    int cost = 0;
    vector<pair<int,int>> mst;
    for(auto it : edges) {
        if(findPar(it.v, parent) != findPar(it.u, parent)) {
            cost += it.wt;
            mst.push_back({it.u, it.v});
            unionn(it.u, it.v, parent, rank);
        }
    }
    cout << cost << endl;
    for(auto it : mst) cout << it.first << " - " << it.second << endl;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
struct node {
    int u;
    int v;
    int wt;
    node(int first, int second, int weight) {
        u = first;
        v = second;
        wt = weight;
    }
};

bool comp(node a, node b) {
    return a.wt < b.wt;
}

int findPar(int u, vector<int> &parent) {
    if(u == parent[u]) return u;
    return findPar(parent[u], parent);
}

void unionn(int u, int v, vector<int> &parent, vector<int> &rank) {
    u = findPar(u, parent);
    v = findPar(v, parent);
    if(rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if(rank[v] < rank[u]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}
```








```cpp
int main() {
    int n, m;
    cin >> n >> m;
    vector<int> adj[n];
    for(int i = 0;i<m;i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    vector<int> tin(n, -1);
    vector<int> low(n, -1);
    vector<int> vis(n, 0);
    int timer = 0;
    for(int i = 0;i<n;i++) {
        if(!vis[i]) {
            dfs(i, -1, vis, tin, low, timer, adj);
        }
    }
```

```cpp
include <bits/stdc++.h>
using namespace std;
void dfs(int node, int parent, vector<int> &vis, vector<int> &tin, vector<int> &low,
        int &timer, vector<int> adj[]) {

    vis[node] = 1;
    tin[node] = low[node] = timer++;
    for(auto it: adj[node]) {
        if(it == parent) continue;

        if(!vis[it]) {
            dfs(it, node, vis, tin, low, timer, adj);
            low[node] = min(low[node], low[it]);
            if(low[it] > tin[node]) {
                cout << node << " " << it << endl;
            }
        } else {
            low[node] = min(low[node], tin[it]);
        }
    }

}
```