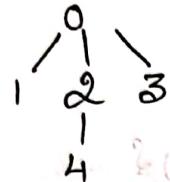


Graphs

① BFS Traversal

I/P :-



O/P :- 0 1 2 3 4

Code

```
vector<int> bfs (int v, vector<int> adj [ ]) {
```

```
    vector<int> res;
```

```
    vector<int> visited (v, 0);
```

```
    queue<int> q;
```

```
    q.push(0);
```

```
    visited[0] = 1;
```

```
    while (!q.empty ()) {
```

```
        int temp = q.front();
```

```
        res.push_back (temp);
```

```
        for (auto x: adj [temp]) {
```

```
            if (!visited [x]) {
```

```
                visited [x] = 1;
```

```
                q.push (x);
```

```
} }
```

```
    q.pop();
```

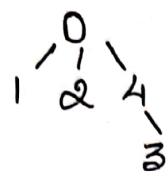
```
}
```

```
return res;
```

```
}
```

#② DFS traversal

I/P:-



O/P:- 0, 1, 2, 4, 3

Code

```
vector<int> dfs(int v, vector<int> adj[]) {  
    vector<int> visited(v, 0);  
    vector<int> res;  
    for(int i = 0; i < v; i++) { // Use for loop to check  
        if(!visited[i])  
            dfsUtil(i, adj, visited, res);  
    }  
    return res;  
}
```

```
void dfsUtil(int src, vector<int> adj[], vector<int> &vis,  
             vector<int> &res) {  
    visited[src] = 1;  
    res.push_back(src);  
    for(auto x : adj[src]) {  
        if(!visited[x])  
            dfsUtil(x, adj, vis, res);  
    }  
}
```

#3 Rat in a Maze Problem (Search in a Maze)

SIP:-

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

OIP:- RRRARRR A RRRRR

① Cell with 0 are blocked

② Movement in 4 directions.

Code : vector<string> res;

vector<string> findpath (vector<vector<int>> &m, int n) {
res. clear(); // always perform when declared
globally

vector<vector<int>> visited (n, vector<int>(n, 0));

if (m[0][0] == 0 || m[n-1][n-1] == 0)

return res;

string s = " ";

dfsUtil(0, 0, s, m, n, visited);

cout << res.begin() << endl;

return res;

}

void dfsUtil (int i, int j, string s, &m, int n, &visited) {

if (i < 0 || j < 0 || i >= n || j >= n) // Search within
bounds

return;

if (m[i][j] == 0 || visited[i][j] == 1) // Prevent going
in a loop.

// Destination reached

if (i == n-1 && j == n-1)

res.push_back(s);

return;

}

visited[i][j] = 1;

dfsUtil(i-1, j, s + "U", m, n, visited);

dfsUtil(i+1, j, s + "D", m, n, visited);

dfsUtil(i, j-1, s + "L", m, n, visited);

dfsUtil(i, j+1, s + "R", m, n, visited);

visited[i][j] = 0; // Backtrack.

#④ Flood Fill algorithm

$$\text{I/P: } \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\text{O/P: } \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Code

```
vector<vector<int>> floodFill(vector<vector<int>>&image, int sr,
                                int sc, int newColor) {
    int x = image[0].size();
    int y = image[0][0].size();
    vector<vector<int>> visited(x, vector<int>(y, 0));
    int oldColor = image[sr][sc];
    dfsUtil(sr, sc, x, y, image, visited, oldColor, newColor);
    return image;
}
```

```
void dfsUtil(int i, int j, int x, int y, &image, &visited,
             int oldColor, int newColor) {
    if (i < 0 || j < 0 || i >= x || j >= y)
        return;
    if (image[i][j] != oldColor || visited[i][j] == 1)
        return;
    visited[i][j] = 1;
    image[i][j] = newColor;
    dfsUtil(i - 1, j, x, y, image, visited, oldColor, newColor);
    dfsUtil(i + 1, j, x, y, image, visited, oldColor, newColor);
    dfsUtil(i, j - 1, x, y, image, visited, oldColor, newColor);
    dfsUtil(i, j + 1, x, y, image, visited, oldColor, newColor);
}
```

#⑤ Make wired connections

- ① So connect each node - find out number of connected components
- ② Connections = components - 1.

Code

```
int makeConnected(int n, vector<vector<int>> &connections){  
    vector<int> adj[n];  
    int m = connections.size();  
    if (m < n - 1) // Connections available < (n-1)  
        return -1; // Not possible  
    for (auto i : connections) { // Make adj list.  
        adj[i[0]].push_back(i[1]);  
        adj[i[1]].push_back(i[0]);  
    }  
    vector<int> visited(n, 0);  
    int countConnected = 0;  
    for (int i = 0; i < n; i++) {  
        if (!visited[i]) {  
            countConnected++;  
            dfsUtil(i, adj, visited);  
        }  
    }  
    return countConnected;  
}  
void dfsUtil(int src, vector<int> adj[], &vector<int> &visited){  
    visited[src] = 1;  
    for (auto x : adj[src]) {  
        if (!visited[x])  
            dfsUtil(x, adj, visited);  
    }  
}
```

#6 Dijkstra algorithm

- ① Shortest path from given src to all vertices
- ② No -ve wgt in graph.

use set; set stores data based on acc. sorting of data.

③ if smaller dist found

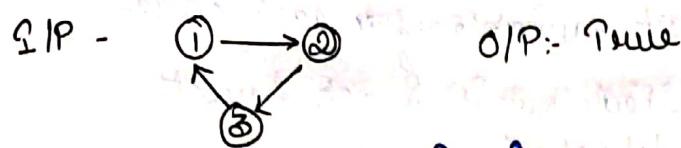
 if vertex not in set; insert directly
 else
 remove and insert.

Code

```
vector<int> dijkstra(int V, vector<vector<int>> adj[], int S){  
    vector<int> distance(V, INT_MAX);  
    distance[S] = 0;  
    set<pair<int, int>> s; // Set of pairs has values  
    s.insert({0, S}); // sorted acc. to first value  
    while (!s.empty()) {  
        auto it = s.begin(); // first value has shortest  
        pair<int, int> num = *it; dist = num.first;  
        int v = num.second;  
        s.erase(it);  
        for (auto x : adj[v]) {  
            if (dist + adj[x][1] < distance[x[0]]) {  
                auto ii = s.find({distance[x[0]], x[0]});  
                if (ii == s.end()) { // Not found → Insert.  
                    s.insert({dist + x[1], x[0]});  
                    distance[x[0]] = dist + x[1];  
                } else { // Not found → Remove & insert  
                    s.erase(ii);  
                    s.insert({dist + x[1], x[0]});  
                    distance[x[0]] = dist + x[1];  
                }  
            }  
        }  
    }  
}
```

between distance;

#7 Detect Cycle in Directed Graph (DFS)



O/P:- True

① Maintain an order array along with vis.

② If node already vis and present in order array it means it was already in path. Cycle found

Code

```
bool isCyclic(int V, vector<int> adj[]){
```

```
vector<int> order(V, 0);
```

```
vector<int> visited(V, 0);
```

```
for(int i=0; i < V; i++) {
```

```
if (!visited[i]) {
```

```
bool check = dfsUtil(i, adj, order, visited);
```

```
if (check == true)
```

```
return true;
```

```
}
```

```
return false;
```

```
bool dfsUtil(int src, vector<int> adj[], &visited, &order) {
```

```
visited[src] = 1;
```

```
order[src] = 1; // White - auto load
```

```
for (auto x : adj[src]) {
```

```
if (!visited[x]) {
```

```
bool check = dfsUtil(x, adj, &visited, &order);
```

```
if (check == true)
```

```
return true; // White, & set
```

```
}
```

```
if (visited[x] && order[x] == 1) // Visited and already present
```

```
return true; // Cycle detected in order so
```

```
cycle
```

```
order[src] = 0; // Backtrack.
```

```
return false;
```

```
}
```

#8 Detect Cycle in Undirected Graph (DFS)

- ① Maintain a parent variable for each node.
- ② If a node is visited but it is not parent; then cycle detected.

Code

```
bool isCycle(int v, vector<int> adj[]) {  
    vector<int> visited(v, 0);  
    for (int i = 0; i < v; i++) {  
        if (!visited[i]) {  
            parent of initial
```

```
            bool check = dfsUtil(i, -1, adj, visited);
```

```
            if (check == true)
```

```
                return true;
```

```
            } else if (check == false)
```

```
                return false;
```

```
} // isCycle
```

```
bool dfsUtil(int src, int par, vector<int> adj[], &visited)  
visited[src] = 1;
```

```
for (auto x : adj[src]) {
```

```
    if (!visited[x]) {
```

```
        bool check = dfsUtil(x, src, adj, visited);
```

```
        if (check == true)
```

```
            return true;
```

```
} else if (check == false)
```

```
// Visited and x != parent.
```

```
else if (visited[x] && par != x)
```

```
    parent of x
```

```
    return true;
```

```
} else if (check == false)
```

```
    parent of x
```

```
    return false;
```

```
} // dfsUtil
```

#① Topological sort (Kahn's Algo)

① Sorting based on the number of indegrees.

Code

```
vector<int> topoSort (int V, vector<int> adj [ ]) {
```

```
    vector<int> res;
```

```
    vector<int> inDeg (V, 0);
```

```
    queue<int> q;
```

```
    for (int i = 0; i < V; i++) {
```

```
        for (auto x : adj[i]) {
```

```
            inDeg[x]++;
```

```
} // for loop - II
```

```
// Push nodes whose indegree == 0
```

```
for (int i = 0; i < V; i++) {
```

```
    if (inDeg[i] == 0) {
```

```
        q.push(i);
```

```
}
```

```
while (!q.empty()) {
```

```
    int temp = q.front();
```

```
(questioned res.push_back(temp);
```

```
    q.pop();
```

```
    for (int auto x : adj[temp]) {
```

```
        inDeg[x]--;
```

```
        if (inDeg[x] == 0) {
```

```
            q.push(x);
```

```
        } // if loop - I
```

```
    } // for loop - III
```

```
} // while loop - IV
```

```
}
```

#10 Course Schedule - I/II

Check whether all courses can be taken.

Print the courses to be taken in order (II)

Code

```
bool canFinish(int numCourses, vector<vector<int>>
                prerequisites)
{
    vector<int> adj[numCourses];
    vector<int> in[numCourses, 0];
    for (auto x: prerequisites) {
        adj[x[1]].push_back(x[0]);
        in[x[0]]++;
    }
    queue<int> q; // II - vector<int> res;
    int coursesProcessed = 0;
    for (int i = 0; i < numCourses; i++) {
        if (in[i] == 0) {
            coursesProcessed++;
            q.push(i);
        }
    }
    while (!q.empty()) {
        int temp = q.front(); // II - res.push_back(temp)
        q.pop();
        for (auto x: adj[temp]) {
            in[x]--;
            if (in[x] == 0) {
                coursesProcessed++;
                q.push(x); // II - if (numCourses != coursesProcessed)
                            // return res;
            }
        }
    }
    return (coursesProcessed == numProcessed);
}
```

Prim's Algorithm Minimum Spanning Tree

⑪

- ① Select the min cost edge connected to existing edges.

Code

```
int spanningTree(int V, vector<vector<int>> adj[],  
    vector<bool> inMST(V, false));  
  
int ans = 0;  
  
priority queue<pair> minHeap;  
minHeap.push({0, 0}); // distance followed by vertex.  
  
while(!minHeap.empty())  
    auto best = minHeap.top();  
    minHeap.pop();  
  
    int dist = best.first;  
    int toVertex = best.second;  
  
    if(inMST[toVertex]) // already present skip  
        continue;  
    else  
        inMST[toVertex] = true;  
        ans += dist;  
        for(auto v: adj[toVertex])  
            if(!inMST[v[0]])  
                minHeap.push({v[1], v[0]});
```

return ans;

Similarly Same Problem -

Commutable Islands

(Interview Bit)

#12 Bellman Ford algorithm

① Works well when there are negative weight present.

② However if -ve wgt cycle is present will not

Code

```
vector<int> dist-BF(int V, vector<vector<int>> adj, int s);
```

```
vector<int> dis(V, 1000000000);
```

```
dis[s] = 0; // Source Vertex.
```

// Relax for (n-1) times.

```
for (int i=0; i < V-1; i++) {
```

```
    for (int j=0; j < adj.size(); j++) {
```

```
        int u = adj[j][0]; // Initial Vertex
```

```
        int v = adj[j][1]; // Final Vertex
```

```
        int w = adj[j][2]; // Wgt.
```

```
        if (dis[u] != 1e8 && dis[u] + w < dis[v])
```

```
            dis[v] = dis[u] + w;
```

```
} // end of for loop
```

```
}
```

return dis; // [v] for v in range

```
}
```

(Total Time Complexity)

Finding -ve wgt cycle;

After (n-1) relaxations; relax for 1 more time

If dis of any vertex changes; then there is a -ve wgt cycle present

Example statement
(dis and adj).

#⑬ Floyd-Warshall Algorithm

The algorithm is used to find shortest dist b/w every pair of vertices in a given edge weighted directed graph.

Mathematical relationship:-

$$A^k[i, j] = \min \{ A^{k-1}[i, j]; A^{k-1}[i, k] + A^{k-1}[k, j] \}$$

$k = 0 \text{ to } (n) \quad [\text{no. of vertices}]$

Generally if a path does not exist b/w i and j ; it is shown by ∞ .

Code

```
void floydWarshall(vector<vector<int>> &mat) {
    int n = mat[0].size();
    for(int k=0; k<n; k++) {
        for(int i=0; i<n; i++) {
            for(int j=0; j<n; j++) {
                // Note some int. paths are not infinity
                if(mat[i][k] != INT_MAX && mat[k][j] != INT_MAX)
                    mat[i][j] = min(mat[i][j],
                                     mat[i][k] + mat[k][j]);
            }
        }
    }
}
```

If inst. of ∞ ; paths that are not connected by are presented by -1

Transform all '-1' to INT_MAX;
perform calculation. Replace INT_MAX by -1 in final answer.

#14 Travelling Salesman Problem

Given a set of cities and dist. b/w every pair of cities, problem is to find the shortest path that visits every city exactly once and returns to the starting point. (Find min. wgt path in Hamiltonian)

Code ($O(n!)$)

```
int total_cost(vector<vector<int>> cost) {
```

```
    int V = cost.size();
```

```
    int start = 0;
```

```
    vector<int> vertex;
```

```
    for (int i = 0; i < V; i++) {
```

```
        if (i != start)
```

```
            vertex.push_back(i);
```

```
    int minPath = INT_MAX;
```

```
    do {
```

```
        int currPath = 0;
```

Calculate

```
    int k = start;
```

cost for

```
// for (int i = 0; i < vertex.size(); i++) {
```

for this

```
    currPath += cost[k][vertex[i]]
```

path from

```
    k = vertex[i];
```

Start

```
// Include last node to src (so return to starting pt)
```

```
currPath += cost[k][start];
```

```
minPath = min(minPath, currPath);
```

```
while (next_permutation(vertex.begin(), vertex.end())) {
```

|| Do for all possible
permutation of paths

```
    minPath = min(minPath, currPath);
```

#15 Count Number of Islands

I/P:- $\begin{bmatrix} "1" & "1" & "1" & "1" & "0" \\ "1" & "1" & "0" & "1" & "0" \\ "1" & "1" & "0" & "0" & "0" \\ "0" & "0" & "0" & "0" & "0" \end{bmatrix}$

O/P:-

1
1
1
1

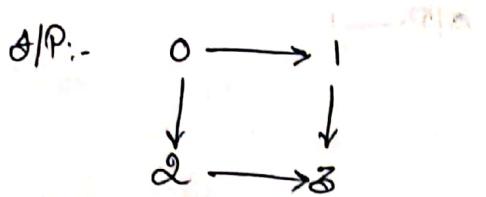
① Find no. of connected components in matrix using DFS.

Code

```
int numIslands (vector<vector<char>> &grid) {
    int n = grid.size();
    int m = grid[0].size();
    vector<vector<bool>> vis (n, vector<bool> (m, false));
    int countIslands = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if ((vis[i][j] == false) && (grid[i][j] == '1')) {
                countIslands++;
                dfs(i, j, n, m, vis, grid);
            }
        }
    }
    return countIslands;
}

void dfs (int i, int j, int n, int m, &vis, &grid) {
    if (i < 0 || i >= n || j < 0 || j >= m) return;
    if (vis[i][j] || grid[i][j] != '1') return;
    vis[i][j] = true;
    int area = 1;
    area += dfs(i+1, j, n, m, vis, grid);
    area += dfs(i-1, j, n, m, vis, grid);
    area += dfs(i, j+1, n, m, vis, grid);
    area += dfs(i, j-1, n, m, vis, grid);
}
```

#16 All Paths from Source to Target in DAG



O/P:- $\{[0, 1, 3], [0, 2, 3]\}$

```
Code vector<vector<int>> paths;  
vector<vector<int>> allPaths(vector<vector<int>> &graph);  
vector<int> p;  
dfs(0, graph, p);  
return paths;  
};  
void dfs(int v, &graph, vector<int> p){  
    p.push_back(v);  
    if (v == graph.size() - 1){  
        paths.push_back(p);  
        p.clear();  
        return;  
    }  
    for (int i = 0; i < graph[v].size(); i++)  
        dfs(graph[v][i], graph, p);  
};
```

17) alien dictionary (N words and K starting alphabets find order in alien dictionary)

I/P: - { "caa", "aaa", "aab" }

O/P: - "c", "a", "b"

① idea is to use topo sort.

Code

```
string findOrder(string dict[], int N, int K){  
    vector<int> adj[K], indeg(K, 0);  
    for(int i = 0; i < N-1; i++) {  
        string w1 = dict[i];  
        string w2 = dict[i+1];  
        for(int j = 0; j < min(w1.length(), w2.length()); j++) {  
            if(w1[j] != w2[j]) {  
                adj[w1[j] - 'a'].push_back(w2[j] - 'a');  
                indeg[w2[j] - 'a']++; // Create  
                break; // connected  
            } // Put as integer graph.  
        } //  
    } //  
    queue<int> q;  
    for(int i = 0; i < K; i++) {  
        if(indegree[i] == 0) q.push(i);  
    } //  
    string res = "";  
    while(!q.empty()) {  
        int temp = q.front();  
        char chr = temp + 'a';  
        q.pop();  
        res = res + chr;  
        for(auto x: adj[temp]) {  
            indegree[x]--;  
            if(indegree[x] == 0)  
                q.push(x);  
        } //  
    } //  
    return res;  
}
```

#18) Flower Planting with No Adjacent Colouring problem (leet code 1042)

I/P:- $n = 3$

path = $[1, 2], [2, 3], [3, 1]$

O/P:- $[1, 2, 3]$.

Code

```
vector<int> gardenNoAdj(int n, vector<vector<int>>
&path)
```

```
vector<int> adj[n+1];
```

```
vector<int> color(n);
```

```
adj[i[0]].push_back(i[1]);
```

```
adj[i[1]].push_back(i[0]);
```

```
for(int i=1; i<=n; i++) {
```

// Initially all colors are available

```
color[i] = 5, 0;
```

```
for(auto a: adj[i]) {
```

// Ans vector stores color of node which is pair

// Visit adj vertices and mark color as "1" if
not available

```
color[ans[i-1]] = 1;
```

```
}
```

```
for(int j=1; j<=4; j++) {
```

// If any color is available

ans[i-1] = j; color of node with

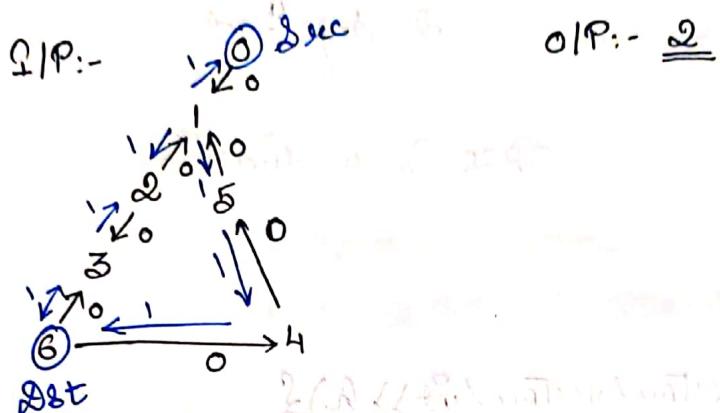
```
break;
```

```
}
```

return ans;

```
}
```

#19) Minimum Edges to Reverse to make path from
src to destination



- ① Make a reverse edge corresponding to each edge.
- ② Give a wgt of 0 to original edge
Give a wgt of 1 to reverse edge

(as shown in i/p figure)
- ③ Now applying Dijkstra shortest path algo,
we find no. of reversals req.

[If we move towards our edge 0 cost]
[Moving in reverse edge incres cost of 1]
(which signify reversals)

#20) Covid Spread (Rotten Oranges)

I/P: 2 1 0 2 1
1 0 1 2 1
1 0 0 2 1

- 0: Empty Ward
- 1: Uninfected Patient
- 2: Infected

O/P:- 2 time units

BFS in 4 directions.

Code

```
int covidSpread (vector<vector<int>> h){
```

```
    int n = h.size();
```

```
    int m = h[0].size();
```

```
    vector<vector<int>> infectionTime (n, vector<int>(m,
```

```
                                            it));
```

```
    queue<pair<pair<int, int>, int>> q;
```

```
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < m; j++) {
```

```
            if (hospital[i][j] == 2) {
```

```
                q.push({{i, j}, 0});
```

```
                infectionTime[i][j] = 0;
```

```
            }
```

```
        }
```

```
    }
```

```
    // Traverse in 4 directions.
```

```
    vector<pair<int, int>> direction
```

```
        = {{1, 0}, {0, -1}, {-1, 0}, {0, 1}};
```

```
    while (!q.empty()) {
```

```
        auto cur = q.front();
```

```
        q.pop();
```

for(auto & it : direction) {
 int i = curr.first.first + it.first;
 int j = curr.first.second + it.second;

// Search within bounds

if(i < 0 || j < 0 || i >= n || j >= m || A[i][j] != -1)

// infectionTime[i][j] = -1
 continue;

g.push({i, j}, curr.second + 1);

infectionTime[i][j] = curr.second + 1;

}

}

int maxTime = 0, flag = 0;

for(int i = 0; i < n; i++) {

for(int j = 0; j < m; j++) {

if(hospital[i][j] != 1)

continue;

if(hospital[i][j] == 1 && infectionTime == -1)

between -1; // all patients not
 injected

maxTime = max(maxTime, infectionTime[i][j]);

}

between maxTime;

}

#Q) Check if Graph is Bipartite

Two Teams?

Graph is 2 colorable.

Interview Bit

int flag = 0;

bool isBipartite (vector<int> &adj) {

vector<int> color (n, -1);

flag = 0;

for (int i = 0; i < n; i++) {

if (color[i] == -1) {

dfs(i, 0, adj, color);

}

}

return (flag) ? 0 : 1;

}

void dfs (int sec, int par, int cle, adj, &color) {

color[sec] = cle;

for (auto it : adj[sec]) {

if (color[it] == -1) {

dfs(it, sec, 3 - cle, adj, color);

else if (it != par && cle == color[it]) {

flag = 1;

break;

}

}

}

#28) Journey to the Moon (Hackerrank)

Each pair of nodes given is from same country.
Determine total number of pairs from diff. countries
you can choose from.

I/P:- [1,2] [2,3] [0].

O/P:- 3

(0,1); (0,2); (0,3).

$$\textcircled{1} \text{ Total no. of pairs possible} = {}^n C_2$$

$$\textcircled{2} \text{ pairs from 1 country} = {}^n C_2$$

$$\therefore {}^n C_2 - {}^n C_2 - {}^n C_2$$

Code

long long JourneyToMoon(int n, vector<int> astronaut)
if ($n = 0$)
 return 0;

vector<int> q[n];

```
for (auto i: astronaut){  
    q[i[0]].push_back(q[i[1]]);  
    q[i[1]].push_back(i[0]);
```

vector<long long> numberNodes; // no. of nodes in
vector<bool> visited(n, false); // each connected
comp.

```
for (int i=0; i<n; i++){  
    if (!visited[i]){  
        long long countNodes = 0;  
        dfs(i, q, visited, countNodes);  
        numberNodes.push_back(countNodes);  
    }  
}
```

// nC_2 total pairs

long long res = n;

res *= n - 1;

res /= 2;

// Temp. to follow given steps
// to avoid overflow

// Remove pairs from individual counts

```
for (int i = 0; i < numberNodes.size(); i++) {
```

long long x = numberNodes[i];

x *= (x - 1);

x /= 2;

res -= x;

}

2. Characteristic (if $\text{LHS} > \text{RHS}$, so LHS small at present) pair

long return res;

{

: DAT Pop (Ans) value

// DFS method to count no. of nodes in each connected component.

```
void dfs(int sec, vector<int> q[], &visited,
```

long long &countNodes){

visited[sec] = 1;

countNodes++;

```
for (auto i : q[sec]) {
```

if (!visited[i])

dfs(i, q, visited, countNodes);

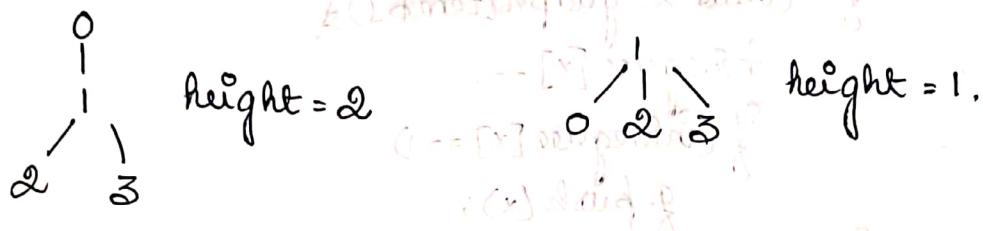
{

: (bottom - after op. 3) 6/6

#25) Find minimum height of a tree

- ① If you find out height from ~~inner nodes~~ leaf nodes it will be higher. (Leaf node : Indegree = 1)
- ② Always ^{try and} pick nodes that are non leaf nodes.

IP:- $[1, 0], [1, 2], [1, 3]$



Code

```
vector<int> findMinHgt(int n, vector<vector<int>> &edges){  
    if(n == 1)  
        return {0};  
    vector<int> graph[n];  
    vector<int> inDegree(n, 0);  
    for(auto i: edges){  
        graph[i[0]].push_back(i[1]);  
        graph[i[1]].push_back(i[0]);  
        inDegree[i[0]]++;  
        inDegree[i[1]]++;  
    }  
    queue<int> q;  
    for(int i=0; i<n; i++)  
        if(inDegree[i] == 1)  
            q.push(i);  
    return q;
```

```
while(n > 2){
```

```
    int g_size = g.size();
```

```
    n = n - g_size;
```

```
    for(int i=0; i<g_size; i++) {
```

```
        int temp = g.front();
```

```
        g.pop();
```

```
        for(auto x: graph[temp]) {
```

```
            indegree[x]--;
```

```
            if(indegree[x] == 1)
```

```
                g.push(x);
```

```
}
```

```
}
```

```
vector<int> res;
```

```
while(!g.empty()) {
```

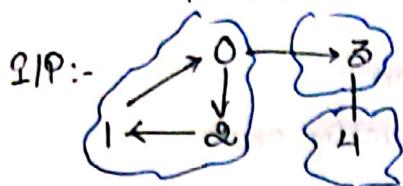
```
    res.push_back(g.front());
```

```
    g.pop();
```

```
    between res;
```

```
}
```

#24) Kosaraju algorithm to find strongly connected Components



OP:- 3

Algo

- ① Input graph
- ② Make copy of our graph
- ③ Maintain a order vector for DFS on graph
- ④ Run DFS on reverse graph from end of order vector.

Code

vector<int> order;

```
int kosaraju(int V, vector<int> adj[V]) {
    order. clear();
```

vector<int> revAdj[V];

```
for (auto &t : int i = 0; i < V; i++) // Reverse graph.
```

for (auto x : adj[i])

revAdj[x]. pb(i);

{ }

vector<bool> vis(V, false);

```
for (int i = 0; i < V; i++)
```

if (!vis[i])

dfs(i, vis, adj);

{ }

vector<bool> vis(V, false);

int count = 0;

```
for (int i = order.size() - 1; i >= 0; i--)
```

if (!vis[order[i]])

dfs(order[i], vis, revAdj); // Pass to

count++;

{ }

return count;

void dfs(int src, &vis, adj[])

vis[src] = true;

for (auto x : adj[src])

if (!vis[x])

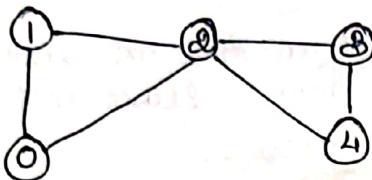
dfs(x, vis, adj);

{ }

order.push_back(src);

#25) Euler's Path & Circuit

A trail that starts & ends at the same vertex covering every edge ~~at most~~ exactly once is a Euler's circuit



Euler circuit

$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$

(Vertex can be repeated)

- ① There should be only 1 component
- ② Start & end vertex same
- ③ Each edge visited exactly once

**

All vertices must have a even-degree to be a euler graph (a graph with euler circuit)

Even degree because ($\frac{1}{2}$ node coming in vertex)
 $\frac{1}{2}$ node going out " to form cycle.

Semi-Eulerian Path

- i) Every edge should be visited once
- ii) Starting vertex \neq End vertex.

Exactly 2 vertices must have ~~one~~ odd degree (start & end vertex)

All other vertices must have even degree.

#26) Snakes and Ladders

① BFS use

② Find the coordinates

$$row = n - ((curr - 1)/n) - 1;$$

$$col = (curr - 1) \% n \rightarrow \begin{cases} i & (\text{even}) \rightarrow n - 1 - \text{col} \\ \text{else} & \rightarrow \text{col.} \end{cases}$$

Code

```
int snakesAndLadders(vector<vector<int>> &board){  
    int n = board.size();  
    int steps = 0;  
    queue<int> q;  
    vector<vector<bool>> visited(n, vector<bool>(n, false));  
    q.push(1);  
    visited[n-1][0] = true;  
    while(!q.empty()) {  
        int size = q.size();  
        for(int i=0; i<size; i++) {  
            int x = q.front();  
            q.pop();  
            if(x == n*n) steps++;  
            for(int k=1; k<=6; k++) { // 6 faces of dice  
                if(k+x > n*n) break;  
                vector<int> pos = findCoordinates(k+x, n);  
                int r = pos[0]; int c = pos[1];  
                if(visited[r][c] == true) continue;  
                visited[r][c] = true;  
                if(board[r][c] == -1) q.push(k+x);  
                else q.push(board[r][c]);  
            }  
        }  
        steps++;  
    }  
}
```

vector<int> findCoordinates (int curi, int n) {

int a = n - (curi - 1) / n - 1;

int c = (curi - 1) % n;

if ($a \% 2 == n \% 2$) {

between {a, n-1, c};

else {

between {a, c};

}

return v;

{()} public void

rights reserved (c) 2018 - 33

rights reserved (c) 2018 - 33

rights reserved (c) 2018 - 33