18/02/2024

Page No.
Date

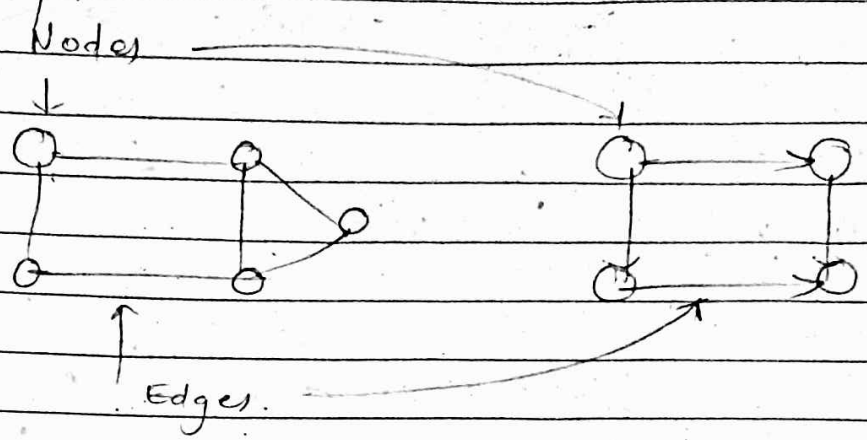
Day 16 OF DSA.

Topic -

Graphs

check Box	
☑	Graph Representation
☑	DFS & BFS Traversal
☑	Graph Implementation
☑	DFS & BFS Implementation

Graph :-



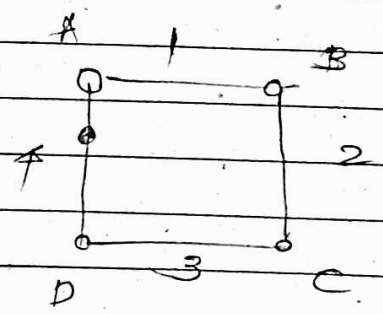
Undirected Graph Directed graph

(1) Nodes :- Entity to store data

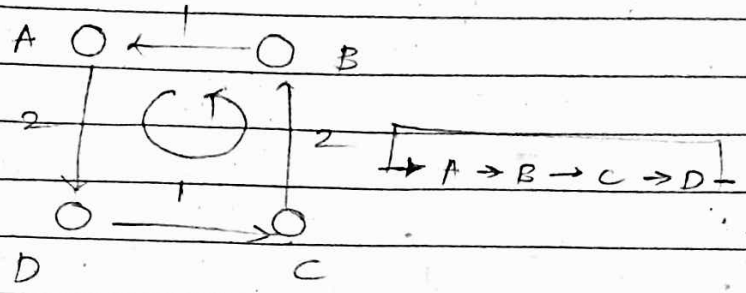
(2) edges = connection bet^ Nodes

(3) degree = In degree
Out degree

* Weighted graph :-

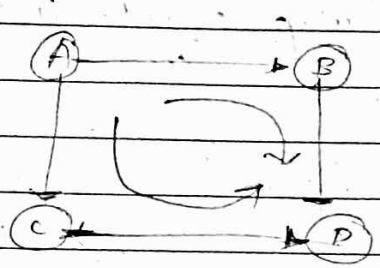


* Cyclic Graph:-



Weighted cyclic directed graph

* Acyclic Graph:-



* Graph :-
  - Adjacency Matrix
  - Adjacency list

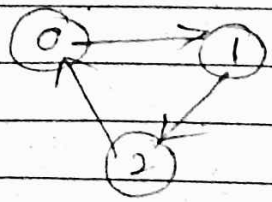
* Adjacency Matrix:

Input → No. of Nodes
No. of edges

n = 3 , m = 3

	0	1	2
0	0	1	0
1	0	0	1
2	1	0	0

0 → 1
1 → 2
2 → 0

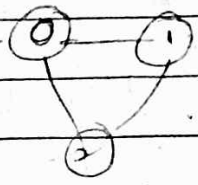


Space complexity: O(n²)

* Adjacency List :-

n = 3
m = 3

0 → 1, 2
1 → 2, 0
2 → 0, 1



0 — 1
1 — 2
2 — 0

* ~~Implementation~~ :-

Adj List :- map < int, list<int> > ;
char

```
class graph
{
  public:
    unordered_map<int, list<int>> adj;

Insert Edge
    void addEdge ( int u, int v, bool direction)
    {
        adj[u].push_back(v);
        if (direction == 0)
        {
            adj[v].push_back(u);
        }
    }
```

Page No.
Date

Printing Adj List :-

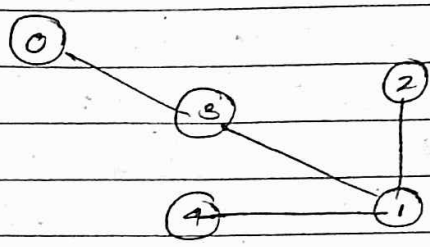
```
void printAdjList()
{
    For(auto i : adj)
    {
        Cout << i.First << "-> ";
        For(auto j : i.second)
        {
            Cout << j.<<" , ";
        }
        Cout << endl;
    }
}
```

* Creating & printing :-

```
vector<vector<int>> printAdjacency(int n, int m,
    vector<vector<int>> &edges)
{
    vector<int> ans[n];

    For(int i=0; i<m; i++)
    {
        int u = edges[i][0];
        int v = edges[i][1];

        ans[u].push_back(v);
        ans[v].push_back(u);
    }

    vector<vector<int>> adj(n);
    For(int i=0; i<n; i++){

        adj[i].push_back;
        For(int j=0; j<ans[i].size(); j++)
        {
            adj[i].push_back(ans[i][j]);
        }
    }
    return adj;
}
```

Page No.
Date

BFS – Breadth First Search :



Adj List	Visited Node
(5)	(3)
✓ 0 → 3	0 → ~~F~~ T
✓ 1 → 2, 4, 3	1 → ~~F~~ T
✓ 2 → 1	2 → ~~F~~ T
✓ 3 → 0, 1	3 → ~~F~~ T
✓ 4 → 1	4 → ~~F~~ T

(II)

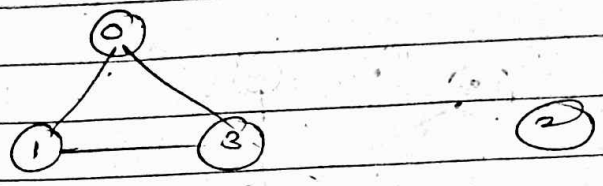
~~X~~
~~X~~
~~X~~
~~X~~
~~X~~

(I) Front Node = 0, X
Front Node = 3 X
Front Node = 1 X
Front Node = 2 X
Front Node = 4 X

(4)

ans → 0, 3, 1, 2, 4

* Disconnected Graph :-



All Nodes are not connected

BFS in Graph :-

① void prepareAdjList (unordered_map <int, set<int>> &adjList, vector<pair<int, int>> &edges) {

1st 2nd For(int i=0; i<edges.size(); i++)

u v {

0 int u = edges[i].first;

1 int v = edges[i].second;

2

adjList[u].insert(v);

adjList[v].insert(u);

}

}

② void printAdj (unordered_map <int, set<int>> &adjList) {

For(auto i : adjList) {

cout << i.first << " -> ";

For(auto j : i.second) {

cout << j << ", ";

} cout << endl; }

(2) void bFS (unordered_map <int, set<int>> &adjList,
unordered_map <int, bool> &visited,
vector <int> &ans,
int node) {

queue <int> q;
q.push (node);
(3) visited [node] = 1;

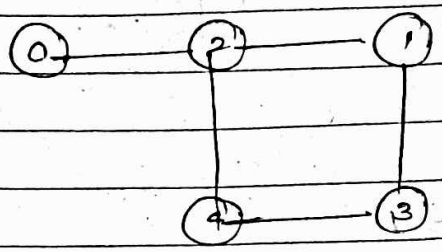
while (!q.empty()) {
(1) int frontNode = q.front();
(2) q.pop;
(4) ans.push_back [frontNode];
(5) for (auto i: adjList [frontNode]) {
if (!visited [i]) {
q.push (i);
visited [i] = 1;
}
}
}
}

(4) vector <int> BFS (int vertex, vector <pair <int,
int >> edges) {

unordered_map <int, set <int> adjList;
vector <int> ans;
unordered_map <int, bool > visited;
prepareAdjList (adjList, edges);
for (int i=0; i<vertex; i++) {
if (!visited [i]) {
bFS (adjList, visited, ans, i);
}
}
return ans;

DFS Traversal

Depth First Search



adj List (3)

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

visited. (2)

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

(1)

dfs(0)
dfs(2)
dfs(1)
dfs(3)
dfs(4)

- Ans = 0, 2, 1, 3, 4.

* Implementation:-

```
vector<vector<int>> depthFirstSearch(
        int V, int E, vector<vector<int>>
                & edges)
{
    unordered_map<int, list<int>> adj;
    For ( int i=0; i<edges.size(); i++)
    {
        int u = edges[i][0];
        int v = edges[i][1];
```

Adj List

undirected graph ⟶

```
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
```

result ⟶ `vector<vector<int>> ans;`

visited map ⟶ `unordered_map<int, bool> visited;`

```
    For (int i=0; i<V; i++) {
        if( !visited[i]){
            vector<int> component;
            dfs (i, visited, adj, component);
            ans.push_back(component);
        }
    }
    return ans;
}
```

main dfs function ⟶

Page No.
Date

```
void dfs(int node, unordered_map<int, bool>
        &visited, unordered_map<int,
        list<int>> &adj,
        vector<int> component){

    component.push_back(node);
    visited[node] = true;

    for(auto i: adj[node]){
        if(!visited[i]){

            dfs(i, visited, adj, component);
        }
    }
}
```