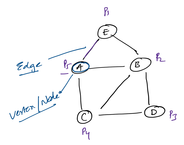
Graphs 1
Monday, 18 July 2021 7:47 PM

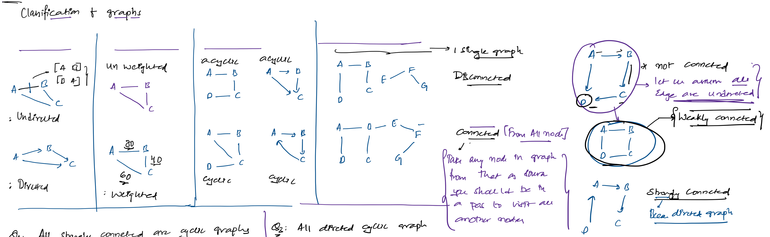
Graph:
It is collection of Nodes & Edges

Nodes:



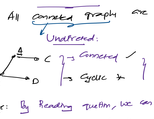
Edges: Want to connect 2 vertices

Classification of graphs



Q1: All Strongly connected are cycle graphs
↳ Directed graph:

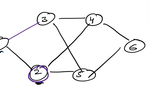
Q2: All connected graph are cycle: (False)
↳ Undirected:



Q3: All directed cycle graph are strongly connected

Note: By Reading question, we can always get weather it is directed/undirected

How to store a graph



Data: (N: no: of vertices)
N, M (M: No: of Edges)
(Edge: Son, Cn)

Ex: 6 8
1 3 | 3 4
1 2 | 2 5
5 6 | 3 5
4 6 | 4 2
: Input

N nodes: {1, 2, 3, 4, 5, N}

Adj Matrix | N Nodes
mat[7][7] | Mat[N+1][N+1]

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

→ How to fill/undirected graph
mat[u][v] = 1 mat[v][u] = 1
Directed graph
mat[u][v] = 1
SC: O(N²)

Adj List

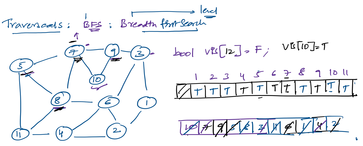
1	3, 2
2	1, 4, 5
3	1, 4, 5
4	3, 2, 6
5	3, 2, 6
6	4, 5

→ Array of vector[N+1] (N, M)
→ vector<int> g[N+1]
If There is Edge u-v
For a single Edge?
undirected { g[u].push_back(v) ; g[v].push_back(u) } 2 value
Directed { g[u].push_back(v) } 1 value
SC: O(N+M) / O(M)

→ Given (u & v), Check if there is a direct edge from (u to v)?
Edge Mat: g[u][v] = 1 : O(1)
Edge List: We have to Traverse on for g[u] O(N)
↳ Note: We can actually run array & binsearch O(L)

Weighted graph / Direction
↳ In question
u v w
Edge from u to v with weight w
Edge List: ✓
g[u].push_back({v, w})
→ Indicate node
→ weight
Edge Mat: ✓
g[u][v] = w

Traversals: BFS : Breadth first Search
↳ level



bool vis[12] = F; vis[10] = T

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

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

Given 5 & 0, Check if There is a path
vis[0] = T : 5-0 path is there
vis[0] = F : 5-0 → no path

7	5 8 10
7	7 10 5
5	7 8 11

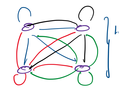
8	5 7 6 11
3	1 7 6
11	5 4

6	3 2 4 8
1	2 3
4	6 2 11
2	1 4 6

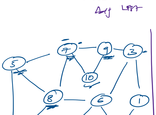
TC: → In every level
: We pop a node : 1
: We traverse on vis : n → Adj List → N + M
: Adj List → Sum of all a value = M

```
bool bfs( vector<int> g[], int N, int s, int d){
    bool vis[N+1] = {F}, int dist[N+1] = N+1;
    queue<int> q; q.enqueue(s);
    vis[s] = True;
    dist[s] = 0;
    par[s] = s;
    while( q.size() > 0){
        int u = q.front();
        q.deque();
        for(int i = 0; i < g[u].size(); i++){
            int v = g[u][i];
            if( vis[v] == false){
                q.enqueue(v);
                dist[v] = dist[u]+1
                vis[v] = true;
                par[v] = u
            }
        }
    }
    return vis[d];
}
```

TC: O(N+M)
SC: O(N)



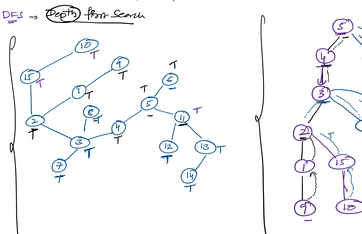
↳ Edge
↳ Nodes : ↳ Edge
N Node : N² Edges
BFS: O(N+M) → M ≈ N² ⇒ O(N²) → Dense graphs
→ M ≈ N ⇒ O(N) → Sparse graphs



// BFS, will give shortest path from given (S - D)
→ Also give shortest path from S → D

// Easy we have parent array
Given d:
while(d != par[d]) { ans(d); d = par[d]; }
// path:

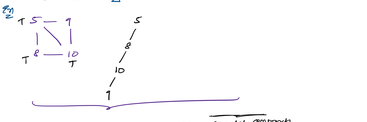
DFS → Depth First Search



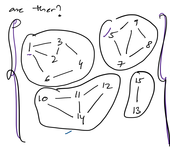

```
void dfs( vector<int> g[], int s, bool vis[]){
    if( vis[s] == True) return;
    vis[s] = True;
    // Call dfs for all nodes attached to s
    for(int i = 0; i < g[s].size(); i++){
        int v = g[s][i];
        dfs( g, v, vis)
    }
}
```

TC: O(N+M) SC: O(N)

// DFS wont give shortest path



Given a graph, Calculate how many connected components are there?
{ if from a node you can visit all other nodes }



vis:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T

: 4 times

Q: How many time we apply DFS to make all nodes are visited
TC: O(N+M)

Tree with N Nodes?
: Edge are there : N-1

→ No: of components : 1 ✓

→ Given a undirected graph, Cycle Detection

Given a graph with N Nodes
N = 7

Component	Edges
1	= N-1 → No cycle > N-1 → Cycle
2	C1: N1, N1-1 C2: N2, N2-1 N1 + N2 - 2 N-2
3	= N-3 → No cycle > N-3 → Cycle
i	= N-i → No cycle > N-i → cycle there

→ With same concept Apply for directed graph?
N = 5

Component wont work for directed Edges