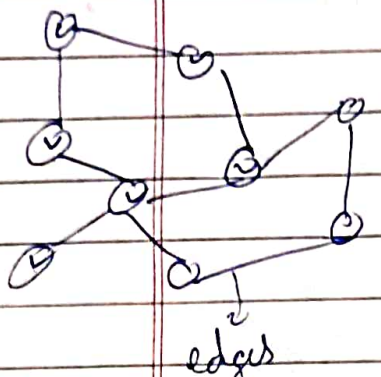
(84) Lecture 5 | Graph Introduction

CLASSTIME Pg No
Date / /

(A) Graph algorithm :- A graph is a collection of nodes where each node might or might not point to other nodes.



The nodes represent real life entities & are connected by edges representing relationship b/w the nodes/entities

(B) Mathematical defn →

$$G = (V, E)$$

Graph is defn a ordered pair of Set V and Set E (Vertices, Edges).

(C) Types ⇒
- Directed
- Undirected
- Weighted
- Unweighted

⇒ Terminologies

(1) Multigraph ⇒ ⇒ undirected | loop
⇒ multiple edges in 2 nodes

(2) Simple graphs ⇒ Undirected
⇒ multiple edges / loops (not allowed)

(3) Complete ⇒ every node is directly connected to every other node.

CLASSTIME Pg. No.
Date / /

(7) Connected ⇒ connected to every not but not directly

Path :- (graph vertices arranged in sequence)

cycle :- vertices can be arranged in cyclic sequence

DAG ⇒ (Directed Acyclic Graph)

Degree ⇒ total edges connected to a vertex

- indegree (incident)
- outdegree (going outward)

} directed

Trees :- it is a connected graph with no cycles.

⇒ Remove edge from tree and get forest

Disconnected graph → Set of Vertices made → {connected components}

* facts ⇒ Tree ⇒ $|E| = |V| - 1$

forest ⇒ $|E| = |V| - 1$ (max)

connected ⇒ $|E| = |V| - 1$ (min)

undirected complete ⇒ $|E| = {}^{|V|}C_2$ (max)

directed complete ⇒ $|E| = |V| * (|V| - 1)$

CLASSTIME | Pg. No.
Date / /

Lecture - 85 Graph Implementation

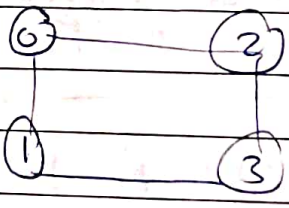
edgelist method

```
class Node {
    int data;
}
class edg {
    Node src;
    Node dest;
    int wt;
    bool dir;
}
class Graph {
    Vector< Edg > Edges;
    Vector {Node} Vertical;
}
```

Adjacency list

⇒ graph as array of linked list

0	→ (2) → (1)
1	→ (0) → (3)
2	→ (0) → (3)
3	→ (1) → (2)



{ array → Vertex
{ LL → Neighbour

⇒ Directed graph me entry direction k according karoyunge.

⇒ If there will be an weighted graph then we store a pair → 1st part having vertex
2nd " " weight

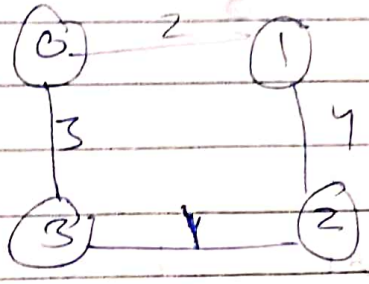
CLASSTIME | Pg. No.
Date / /

① Adjacency Map → array of hashmap / Unordered-map

0	→ {{1,2}, {3,3}}
1	→ {{0,2}, {2,4}}
2	→ {{1,4}, {3,1}}
3	

{ array → Vertex
hashmap → neighbour

⇒ efficient to check whether a node is neighbour or Not



② Adjacency Matrix :-

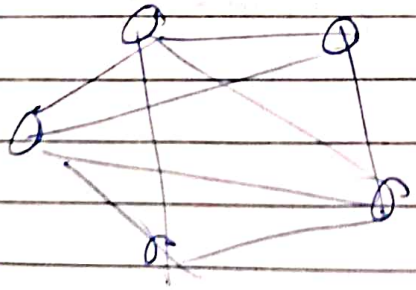
2D Matrix [V x V]
↳ vertices

	0	1	2	3
0				
1				
2				
3				

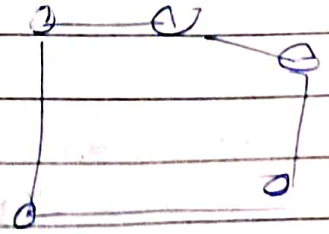
$$adj[i][j] = \begin{cases} 1 & \text{there's an edge from } i \text{ to } j \text{ vertex} \\ 0 & \text{No edge from } i \text{ to } j \end{cases}$$

⇒ Directed mei jaadi hui !
but aadi hui 0;

Dense graph



Sparse graph

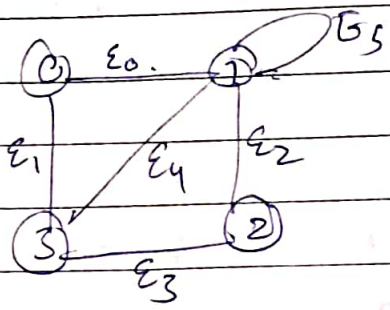


⑧ Incidence Matrix:

→ Prepare V x e matrix

vertices ↙ ↘ edges

$m[i][j] = \begin{cases} 1 & \text{if } i^{th} \text{ vertex belongs to the } j^{th} \text{ edge} \\ 0 & \end{cases}$



E

V	0	1	2	3	4	5	
0	1	1	0	0	0	0	
1	0	0	1	0	1	0	
2	0	0	0	1	0	1	2 (self loop)
3							

Column Sum = 2
Row Sum = Degree.

⇒ Most efficient implementation → Adjacency list

Code {Unweighted}

```
#include<vector>
#include<list>
using namespace std;

vector<list<int>> graph;
int v;
void addEdge(int src, int dest, bool bidir = true){
    graph[src].push_back(dest);
    if(bidir){
        graph[dest].push_back(src);
    }
}
void display
```

```
Void display() {
    for(int i=0; i< graph.size(); i++) {
        cout<< i <<"->";
        for(auto el: graph[i]) {
            cout << el <<",";
        }
        cout<<"\n";
    }
}
int main() {
    cin>>v;
    graph.resize(v, list<int>());
    int e;
    while(e--){
        int s,d;
        cin>>s>>d;
        addedge(s,d);
    }
    display();
    return 0;
}
```

#Code for weighted

```
Vector<list<pair<int,int>>> graph;
int v;
Void addedge(int src, int Dest, int Wt, bool bidir=true){
    graph[src].push_back({Dest,Wt});
    if(bidir){
        graph[Dest].push_back({Src,Wt});
    }
}
```

```
Void display() {
    for(int i=0; i<graph.size(); i++) {
        cout<<i<<"->";
        for(auto el: graph[i]) {
            cout<<"("<<el.first<<" "<<el.second<<");
        }
        cout<<"\n";
    }
}
int main() {
    cin>>v;
    graph.resize(v, list<pair<int,int>>());
    int e
    cin>>e;
    while(e--) {
        int s,d,wt
        cin>>s>>d>>wt;
        add_edge(s,d,wt,false); // false mean graph is directional
    }
    display();
    return 0;
}
```

// Now give s,d in input and represent a graph using adjacency matrix.

Q3 Topological Sort (Leetcode)

all path from source to target

⇒ Unordered map can also be used in place of list/ Adjacency map

```
Vector<Unordered_map<int, int>> graph;
int v;
Void addEdge(int ---- (Same)
    graph[src][Dest] = Wt;
    if (bidir) {
        graph[Dest][src] = Wt;
    }
}
Void display() {
    for(int i=0; i<graph.size(); i++){
        cout<<i<<"->";
        for(auto el: graph[i]) {
            cout<<"("<<el.first<<" "<<el.second<<"),";
        }
        cout<<"\n";
    }
}
```

Same

list change → Unordered map

CLASSTIME Pg. No.
Date / /

Lecture-86 (BFS/DFS)

Graph Traversals

(A) DFS (Recursion)

(1) Given a path calculate all paths b/w two vertices.

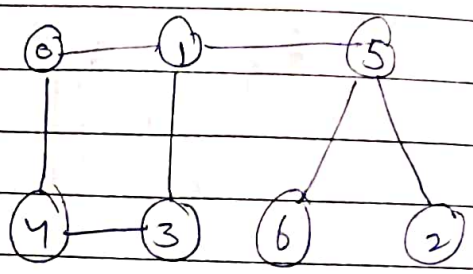
(or)

(2) Given a graph check whether there is a path b/w any two vertices or not ??

Hint (Recursively)

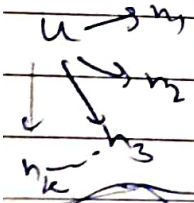
→ neighbour se path hai ya nahi to node se bhi hoga.

→ Jisko travel karenge Usko visited mark karenge



$$f(u,v) = \begin{cases} f(n_1, v) \\ f(n_2, v) \\ f(n_3, v) \\ \vdots \\ f(n_k, v) \end{cases}$$

$f(u,v)$ → whether there is a path from u to v or not.



n_1, n_2, n_3 and all these neighbours (neighbours) are unvisited.

Code

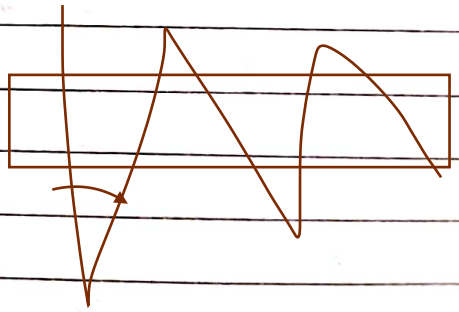
```
vector<list<int>> graph;
int v;
void add_edge(int src, int dest, bool bi_dir = true){
    graph[src].push_back(dest);
    if (bi_dir){
        graph[dest].push_back(src);
    }
}
```

Unordered_set<int> visited;

```
bool dfs(int curr, int end){
    visited.insert(curr);
    if(curr==end) return true;
    visited.insert(curr);
        for(auto neighbour: graph[i]){
            if(!visited.count(neighbour)){
                bool result = dfs(neighbour, end);
                if(result) return true;
            }
        }
    return false;
}

void
bool anypath(int src, int dest){
    return(src, dest)
}
int main(){
    same
    cout<<anypath<<endl;
}
```

all path

```
#include<unordered_set>
 "  <list>
 "  <vector>
vector<vector<int>> result;
unordered_set<int> path;
vector<list<int>> graph;
void add edge(int src, int dest, bool bi_dir=true){
    graph[src].push_back(dest);
    graph[dest].push_back(src);
    if(bi_dir){
        graph[dest].push_back(src);
    }
}
```



```
void dfs(int curr, int end, vector<int> path) {

    if (curr == end) {
        path.push_back(curr);
        result.push_back(path);
        path.pop_back();
    }
    visited.insert(curr);
    path.push_back(curr);
    for(auto neighbour : graph[curr]) {

        if (not visited.count(neighbour)) {
            bool result = dfs(neighbour, end);
            if(result) return true;
            cout << neighbour << "\n";
            dfs(neighbour, end, path)
        }
    }
    path.pop_back();
    visited.erase(curr);
    return;
}
```

Breadth First Search

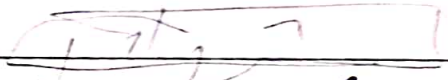
CLASSTIME Pg. No.
Date / /

→ Traverse immediate neighbours first together

→ Use to Find shortest path in Unweighted graph.

Same Prev. Code

```
void bfs(int Src, vector<int>& dist){
    queue<int> qu;
    ~~visited.insert(Src);~~ visited.clear()
    dist.resize(V, INT_MAX);
    dist[Src] = 0;
    visited.insert(Src);
    qu.push(Src);
    ~~for(auto~~
    while(not qu.empty()){
        int curr = qu.front();
        qu.pop();
        cout<<curr<<" ";
        for(auto neighbour : graph[curr]){
            if(not visited.count(neighbour)){
                qu.push(neighbour);
                visited.insert(neighbour);
                dist[neighbour] = dist[curr]+1;
            }
        }
    }
    cout<<"\n";
}
int main(){
    resize graph acc. to (V)
    add edges
    Enter x and bfs(x,dist)
    Print dist[];
```



Lecture 87 — Problems on DFS/BFS

CLASSTIME Pg. No.
Date / /

int main() {

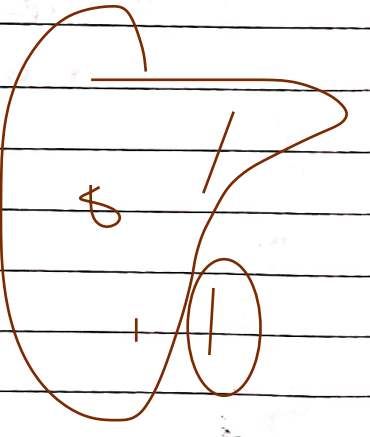
(*) Connected components

↳ Jitni baar BFS/DFS call ho utne connected components hote hai

Code

// same upto add edges ---

```
void dfs(int node, unordered_set<int>& visited) {
    visited.insert(node);
    for(auto neighbour : graph[node]) {
        if(!visited.count(neighbour)) {
            dfs(neighbour, visited);
        }
    }
}

int connectedComponent() {
    int result = 0;
    unordered_set<int> visited;
    for(int i=0; i<v; i++) {
        // go to every vertex
        // if from vertex we can initialise a dfs,
        //   we get one more cc.
        if(visited.count(i) == 0) {
            result++;
            dfs(i, visited);
        }
    }
    return result;
}

int main() {
    Same
```



Q1 LeetCode 200 Number of Islands;

Q2 LeetCode 417 Pacific Atlantic Water flow;
Multiple Source BFS.

Q3 LeetCode 994 Rotting Oranges
Multilevel Multisource

Q4 LeetCode 542 01 Matrix [Important] (Khud Karo) HW

Done Again apply a reverse approach as Q417

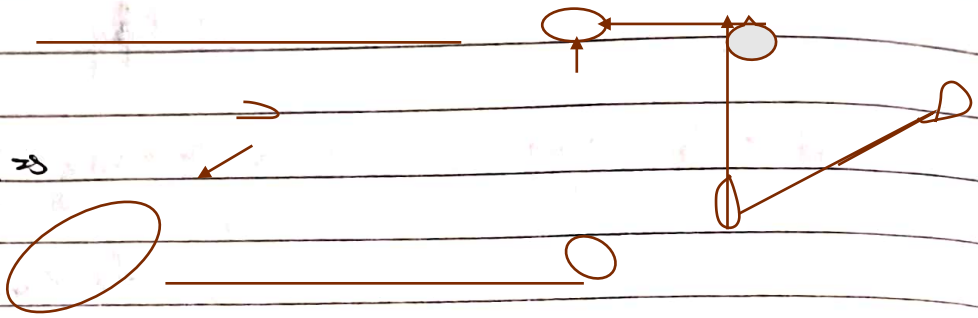
CLASSTIME | Pg. No.
Date / /

Lecture-88

(Graphs : Disjoint Set Union) DSU

⊙ Linear DS ⇒

⊙ Non-linear DS ⇒



Clustering/Grouping :- You will be having some element & you need to add them/segregate them in different groups/clusters. and sometimes we need to identify the group any element belongs to.

Terminologies

① Leader/Parent of the groups

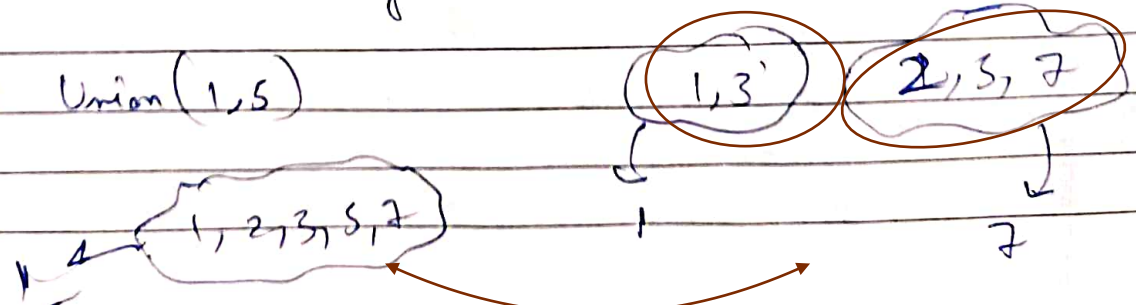
→ To uniquely identify a group we will pick any element from the group & make it the representative/leader/parent of the graph.

⊛ DSU

Q: What funct. does need to support??

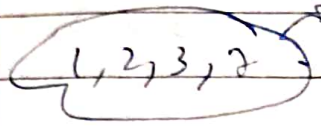
① Union(a, b) ⇒ adds the group where element b belongs to the group where element a belongs or vice-a-versa

eg: Union(1,5)



CLASSTIME | Pg. No.
Date / /

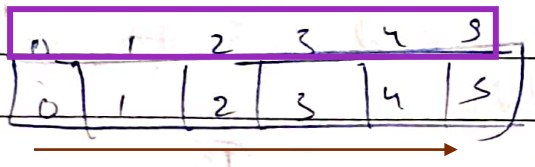
② find(x)/get(x) : this will be used to find which group x belongs to. will will return the parent of the ~~grp~~ group that x belong to.



get(2) = 3
get(3) = 3.

⊙ Approach 1 : Represent every group as a set
More Time complexity
So, Not Used :)

⊙ Approach 2 : Can we use array ??



⇒ index element ko represent karenge aur jo index pe ⇒ value hai vo uske parent ko represent karega

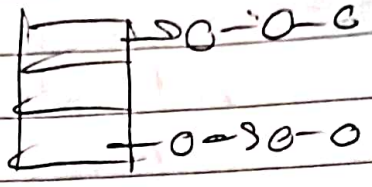
⇒ Jaise jaise merge karte jayenge parent change hojayegaa.

⇒ Loop chala denge aur jisme bhi merge karna hoga parent ko ussi group ke parent me change kardenge

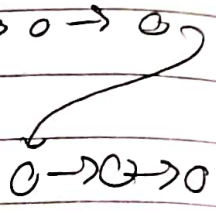
```
int find(n) {
    return par[n];
}
void Union(a, b) {
    a = find(a);
    b = find(b);
    for(i=0; i<n; i++)
        if(par[i] == b) {
            par[i] = a;
        }
}
```

CLASSTIME Pg No
Date / /

Approach 3 : # LL



Union O(1)
find O(N)



Union by Size :-

→ smaller grp ko larger graf me merge karo

O(log N)

Qs If for an element we need to find parent, what is one of the good D.S to represent child & parent relationship.

→ Tree

```
void Union(a,b)
    a = find(a);
    b = find(b);
    if (S2[b] < S2[a]) {
        S2[a] += S2[b]
        par[b] = a
    } else {
        S2[b] += S2[a]
        par[a] = b
    }
```

```
int find(x) {
    if (par[x] == x) {
        return x
    }
    return find(par[x])
}
```

(Union by Rank)

CLASSTIME | Pg. No.
Date / /

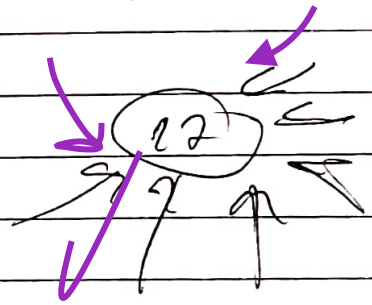
Rank ⇒ Jo bhi max no. of link bante ha parent tak Pahonchne k usse kehte ha rank.

```
Void Union (a,b) {
    a = find(a)
    b = find (b)
    if (rank[a] ≤ rank[b])
        par[a] = b
        rank[b] ++
    } else {
        par[b] = a
        rank[a] ++;
    }
```

O(log n)

⇒ Optimize the approach by path compression.

1)
```
int find (x) {
    if (par[x] == x) return x;
    return par[x] = find (par[x])
}
```



Sbka parent same set taught by niveditha mam

(*) inverse ackerman func.ⁿ

↳ more optimization upto $\log^* n$

$\log^* n$ ⇒ it represents that if you have a value of n, and you repeadly apply $\log_2 n$ on this value then how many ops you can reduce it to ≤ 1

operations

$n = \log_2 n$

CLASSTIME | Pg. No.
Date / /

Code

```
#include <iostream>
#include <vector>
using namespace std;

int find(Vector<int> &parent, int x) {
    //This method returns which
        group/cluster x belongs to
    return parent[x] = (parent == x) x ?
                        find(parent, parent[x]);
}

Void Union(Vector<int> &parent, Vector<int> &rank, int a,
                                                    int b){
    a = find(parent, a)
    b = find(parent, b)

    Void Union
    if (rank[a] ≥ rank[b]) {
        rank[a]++;
        parent[b] = a;
    }else{
        rank[b]++;
        parent[a] = b;
    }
}
int main() {
    int n, m;
    Cin>>n>>m;
        //n → elements, m → no. of queries
    Vector<int> parent(n+1);
    Vector<int> rank(n+1, 0)
```

```
int
for(i=0; i<n; i++) {
    parent[i] = i;
}

while (m--) {
    string str;
    cin >> str;
    if (str == Union) {
        int x, y;
        cin >> x >> y;
        Union(parent, rank, x, y);
    } else {
        int x;
        cin >> x
        cout << find(parent, x) << "\n";
    }
```

Lecture - 89

(Topological Sorting & Cycle Detection)

Algo → can be solved for DAG or Directed Acyclic Graph

(*) Dependency resolution

Tensor flow → library of machine learning
Jisme koi DS do or vo isse train krke model dega.

Pytorch
↓ numpy
↓ pandas

(*) Kahns Algo :- Jis bhi node ki ~~degree~~ indegree zero hogi usse installation start hogi.
Or jaise jaise indegree 0 hoti jayegi node ko queue me dalte jayenge or install karte jayenge.

Code →

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

vector<list<int>> graph;
int V;
void addedge(int a, int b, bool bidir = true){
    graph[a].push_back(b);
    if(bidir){
        graph[b].push_back(a);
    }
}
```

```
Void topoBFS() {

    Vector<indegree>
    Vector<int> indegree(V, 0);
    for(int i=0; i<v; i++){
        for(auto neighbour : graph[i]) {

            indegree[neighbour]++

        }
    }

    queue<int> qu;
    Unordered_set<int> vis;
    for(int i=0; i<v, i++) {
        if(indegree[i]==0){
            qu.push(i);
            vis.insert(i);
        }
    } Cout<<
    while(nat qu.empty) {
        int curr = qu.front(); Cout<<curr<<" ";
        qu.pop().
        for(auto neighbour : graph[curr]) {
if(not vis.count(neighbour)) { indegree[neighbour]--
            if(indegree[neighbour]==0) {
                qu.push(neighbour);
                vis.insert(neighbour);
            }
        }
    }
}
```

```
int main() {
    enter no V, E, creating graph with V;

    enter edges of graphs

    topoBFS();
    return 0;
}
```

Qy Leet Code 207, 210 Course Schedule ;

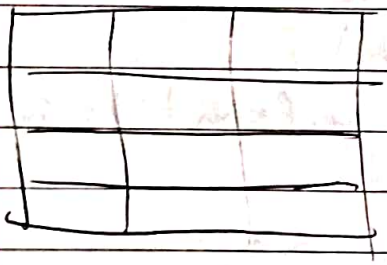
H.W // // //

Q for 2nd Technical round

Qy Given a Grid of n×m size. Every cell of the grid are marked as L, R, U, D. Character on a cell denotes if you are standing at backside what direction you can move to. Check if we start from (0,0), can we reach (n-1, m-1) ??

⇒ No more space than given grid
⇒ Grid can't be modify

Cycle Detection



Approach 1 ; n×m (Worst case) iss liye traversing bata hai ke mein cycle present h

Approach 2 ; Slow/Fast Pointer

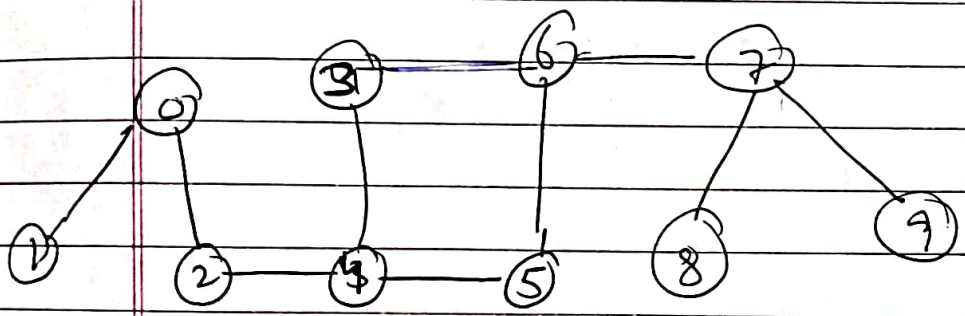
CLASSTIME | Pg. No.
Date / /

Q) Given a directed graph, detect if it is having cycles or not ??

↳ Kahn's Algo

Q) How to detect cycles in an Undirected graph

BFS DFS DSU (Better option)



(1) We will read each edge & on the vertices of the edge apply Union operation

(2) Agr koi bhi 2 node alag alag component mei hai to & unka union kelo or agar & same component me hai (i.e. same parent hai) to cycle bann rahi hai (har edge ko traverse karte hue)

→ eg 3,6 already has a diff. path

→ Copy the same DSU code and modify it by changing Union function as return type & bool.

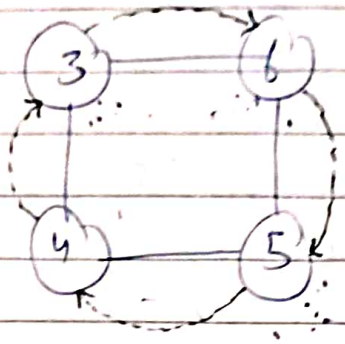
⇒ and make a base case : if (a == b) return true;

```
bool Union(vector<int> &parent, vector<int> &rank, int a, int b){
    a = find(parent, a)
    b = find(parent, b)
    if (a == b) return true; // Cycle detected

    if (rank[a] >= rank[b]) {
        rank[a]++;
        parent[b] = a;
    } else {
        rank[b]++;
        parent[a] = b;
    }
    return false;
}

int main() {
    int n, m;
    cin >> n >> m;
    // n -> elements, m -> no. of queries
    vector<int> parent(n+1);
    vector<int> rank(n+1, 0);

    for (int i=0; i<v; i++) {
        parent[i] = i;
    }

    while (m--) {
        int x, y;
        cin >> x >> y;

        bool b = Union(parent, rank, x, y);
        if (b == true) cout << "Cycle detected" << endl;
    }
    return 0;
}
```

<bits/stdc++.h>

CLASSTIME Pg. No.
Date / /

Cycle detection using DFS



check if the visited node is not your parent then there is a cycle

Code:

```
using namespace std;
Vector<list<int>> graph;
int V;
Void add_edge(){}
Void display(){}

bool dfs(int src, int parent, Unordered_set<int>& Vis){

    Vis.insert(src);
    for(auto neighbour : graph[src]){

        if( Vis.count(neighbour) and neighbour != parent){

            return true;
        }
        if(! Vis.count(neighbour)){
            bool res = dfs(neighbour, src, Vis)
            if(res == true) return true;
        }
    }
    return false;
}
bool hasCycle(){
    unordered_set<int> vis;
    for(i=0; i<v; i++){
        if(! Vis.count(i)){
            bool res = dfs(i, -1, Vis);
            if(res = true) return true;
```

```
    return false;
}
int main() {
    // Enter v, e
    graph.resize(v, {});
    // Enter edges
    while (e--) {
        s, d
        ---
        add(s, d, false)
    }
    display();
    bool b = has_cycle();
    cout << b << "\n";
    return true
}
```

---

(2) Cycle detection by BFS (Almost Same as DFS)

if a node is already visited & it is not your parent then you have a cycle.

Difference ⇒ DFS mei parent ke liye recursive call use hota hai.
But BFS mei hum parent ke liye ek of (parent) vector banayenge.

Code

CLASSTIME Pg. No.
Date / /

```
bool bfs (int src) {

    unordered_set < int > vis;
    queue < int > qu;
    Vector < int > par(v, -1);
    qu. push( src);
    Vis. insert (src);
    while (not qu. empty()) {
        int curr = qu. front();
        qu. pop();
        for (auto neighbour : graph[i]) {
          if (Vis. count (neighbour) and par[curr] != neighbour)
          { return true }
          if (! Vis. count ( neighbour) {
              Vis. insert ( neighbour);
              par [neighbour] = curr;
              qu. push (neighbour);
          }
        }
    }
}

bool has_cycle() {
    Unordered_set <int> vis;
    for ( int i = 0; i < v; i + +) {
        if (! vis. count (i)) {
            bool result = bfs(i);
            if (result = = true) return true;
        }
    }
}
```

(207) (210) H.W

CLASSTIME	Pg. No.
Date / /	

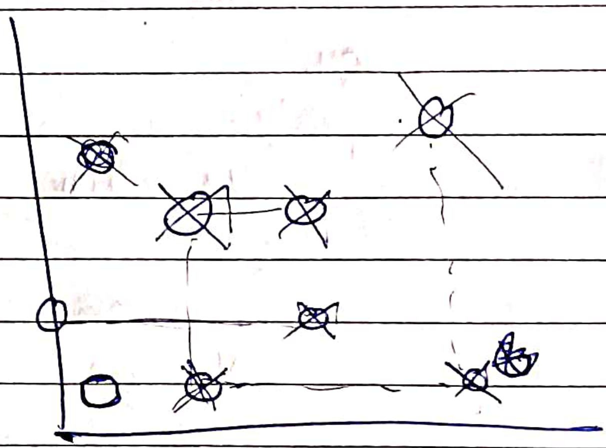
Q1 Leetcode [947] — Most stones removed with same row or column.

Leetcode [2316] } H.W

Leetcode [1559]

(*) Other lectures are in Java Notebook [C++ (Part 4)]

Q1 947



Vector (removed)

for i...

C++ (fwskills)

Page No.
Date :

Lecture -89 {Kruskal's and prim's Algo}

→ Methods to make minimum spanning tree.

⇒ Interview mei kabhi bhi spanning tree pe question aata hai toh try karna ki tum kruskals apply karo. Because Prim's thoda tricky hota hai.

→ All Nodes
⇒ Some edges
→ Avoid Cycles

(#) Kruskal's → Choose minimum weight edges 1st.
↳ edge List → Sort by weight → DSU (Cycle detection)

Qs Hacker Rank → Kruskal (MST)

```
#include<bits/stdc++.h>
# define ll long long int;
# using namespace std;

int find(Vector<int> & parent; int a) {
    return (parent[a] = (parent[a] == a)? a : find(parent, parent[a]));
}

Void Union(Vector<int>& par, Vector<int>& rank, int a, int b) {

    a = find(parent, a)
    b = find(parent, a)
    if (a == b) return;
    if(rank[a] > rank[b]) {
        rank[a]++;
        par[b] = a;
    }
}
```

```
else {
    rank[b] ++;
    par[a] = b
  }
}

struct Edge {
    src;
    dest;
    wt;
}

bool cmp( Edge e1, Edge e2) {
    return e1.wt < e2.wt;
}

ll kruskals(Vector<Edges> & input; int n, int e) {

    sort (input.begin(), input.end(), cmp);
    Vector<int> par (n+1);
    Vector<int> rank (n+1, 0);
    for(int i=0; i<n; i++) {
        int parent[i] = i;
    }

    int EdgeCount = 0; //n-1
    int i = 0;
    ll ans = 0;
    while (edgeCount < n-1 and i < input.size()) {
        Edge curr = input[i] // becoze input Sorted
        int srcpar = curr.src;                    So, get min. wt edge
        int destpar = curr.dest;
        if (srcpar != destpar) {
            Union( par, rank, srcpar, destpar);
            ans += curr.wt; edgeCount++;
        }
```

//Include edge as this will not make a cycle

```
        i++;
    }
    return ans; // doesn't matter u picked the last edge or not we still need to go to next edge.
}

int main() {
    int n, e;
    cin>>n>>e;
    vector<Edge> v(e);

    for(int i=0; i<e; i++) {
        cin>> v[i].src >> v[i].dest >> v[i].wt;
    }
    cout << kruskals(v, n, e);
    return 0;
}
```

[10 min]

Qs Leet Code [1135] Connecting cities with minimum cost (Paid Question)

Approach: Normal Kruskal apply ho skta hai but iske ander 1 se jyada connected components bhi ho skte hai.
So, hum DSU ka use karenge to check how many connected components are there.
Agar 1 se jyada connected components honge to -1 return hoga.
Agar 1 hi connected component hoga to Kruskal apply kardenge

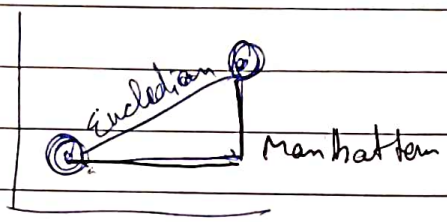
H-W

Page No
Date

Qy Leet Code 1584

* Euclidian Distance $\Rightarrow \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

* Manhattan Distance $\Rightarrow |x_2-x_1|+|y_2-y_1|$



Qy Prims (MST): Special Subtree (Hacker Rank;)

Imp

bfs + Greedy

↳ priority queue

↳ Used in Dijkstra.

⇒ In prims we start with a src & try to discover others

⇒ at the start of the algo, discover wt. of every node except src is ∞, because we have not discover anything.

⇒ Steps to write Code

* - DS - Visited set, priority_queue<pair>, Unordered_set
- Insert the pair of <-1, src> in the pq.
- One by one remove the root element of the pq.
- if the root element is already visited we will just continue.
- We mark the root visited
- We store the wt of the pair in the ans.

- update the mapping
- go to every neighbour of the curr element, and only add those which are non-visited and have a better wt proposition

---

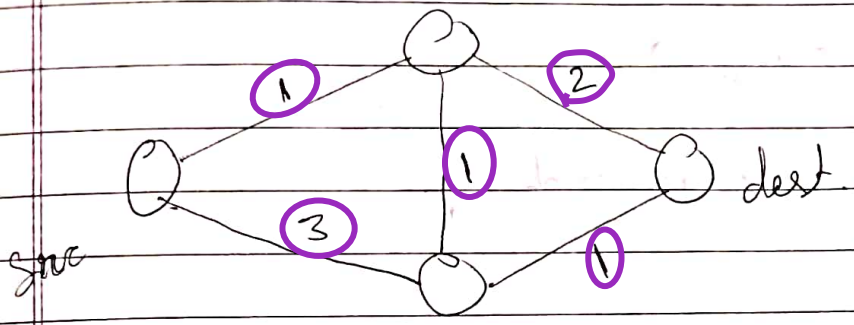
⇒ pq me pehle wt. phr node
⇒ graph mei pehle node phr wt.

Used for finding Shortest path

Page No
Date :

Lecture-90 (Dijkstra Algo)

Qs Given an undirected weighted graph, along with a src. and dest. Find the shortest path b/w src. and dest. in terms of sum of edge wts.



We already know how to solve Shortest path problem for unweighted graph ↳ BFS

Node:- → Prims me 1 node se dest. tk ka Shortest path nikala (src) jata hai. (MST) (Jha Tree ka weight km ho)
→ Dijkstra Me 1 Node se Baki Sari nodes ka Chota path nikala jata hai (in single go)

Prims Concept + pq + map + relaxation algo

↳ In concepts ka Use karke code likh skte ho

node	dist	via
0	0	-1
1	1	0
2	3	4
3	2	1
4	3	1
5	5	6
6	4	3

Concept ⇒ Sabse pehle k src leke usko through uske neighbours ka dist. nikalenge. Jis neighbour ka distance update hoga usi ko next traverse karenge. The jo jo traverse hoti jayegi usse pq me push krvate jayenge.

```
if(d2 + w < d1){
    mp[y] = d2 + w;
    via[y] = x;
}
```

3

Page No
Date

⇒ Jb traversal k baad pq se (pop) hoga tb use visited mark karenge

Code

↳ Almost Same code as prims.
↳ par (vector) ——change——→ via
↳ total_count, result ——→ no use
↳ return type ⇒ Unordered_map.

~~define~~ pp pair<int, int>,

⇒ Unordered_map<int, int> diskstra(int src, int n) {

priority_queue<pp, Vector<pp>, greater<pp>> pq
Vector<int> Via(n+1)
Unordered_set<int> Vis;
Unordered_map<int, int> mp;

// prims me 1 to n+1

pq. for(int i=0; i<n; i++) { // O(V)
mp[i] = INT_MAX;
}
pq.push(0, src);
mp[src] = 0;

while(!pq.empty) { // O((V+E) log V)
pp curr = pq.top();
if(Vis.count(curr.second)) {
pq.pop();
continue;
}
Vis.insert(curr.Second);
pq.pop()

Page No.
Date :

```
for (auto neighbours : gr[curr.second]) {
    if (mp[neighbours.first] > mp[curr.second] + neighbour.second
        && !vis.count(neighbour.first)) {
    {
        mp[neighbour.first] = mp[curr.second] + neighbour.second;
        pq.push({ mp[curr.second] + neighbour.second, neighbour.first});
        via[neighbour.first] = curr.second
    }
}
return mp;
}
int main() {
    int n, m;
    cin>>n>>m;
    gr.resize(n+1, list<pp>());
    while(m--) {
        int u, v, wt;
        cin>>u>>v>>wt;
        add_edge(u, v, wt);
    }
    int src;
    cin>>src;
    Unordered_map<int, int> sp = dijkstra(src, n);

    int dest;
    cin>>dest;
    cout<<sp[dest]<<"\n";
    return 0;
}
```

Page No

Date

Kruskals Prims

no. of edge < no. of Vertex → sparse graph

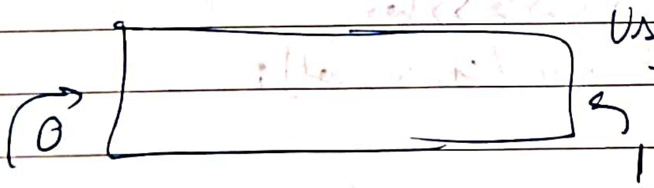
no. of edge > no. of Vertex → dense graph

⇒ Acc. to Sir kruskal use karo
⇒ But Prims karne se Tumhara Dijkstra bhi cover hojayega

Trick → ① Agar Sari edges pe weight same hai to Dijkstra (no need).
Normal BFS laga or jo path aaye use weight se multiply kardo.

② Binary Weight (0,1)

(0-1 BFS) (Google Question)



Use dequeue instead of que

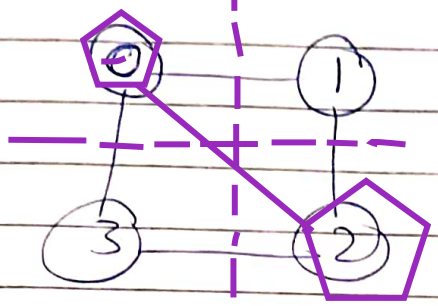
TM ⇒ O(1)

0 weight wali node ka front pe add karo or 1 weight wali k end pe

Q → Leet Code 1368 ⊙ Approach 1 → Dijkstra
⊙ Approach 2 → (0-1 BFS) (0,1)

Page No.
Date :

Qus LeetCode 785 {Is graph Bipartite}



{0, 2}, {1, 3}
Set A Set B

{2 color Problem

⇒ Hare connected Component Bipartite hai to graph bhi bipartite hai.
Nhi to Nhi hai.