GRAPHS

Day 1

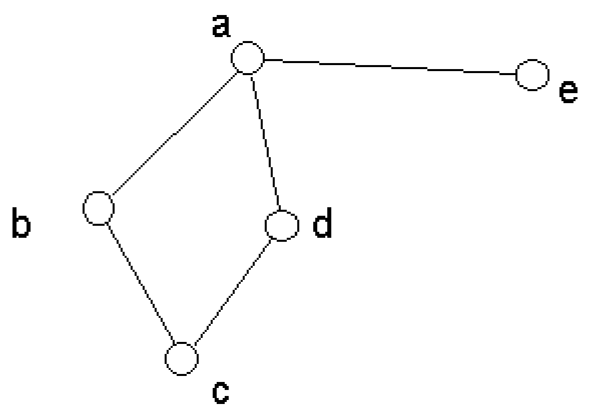
Youtube link :

https://youtu.be/JyYzPMsT8xo

Contents:

Basic Terms

What is a Graph?

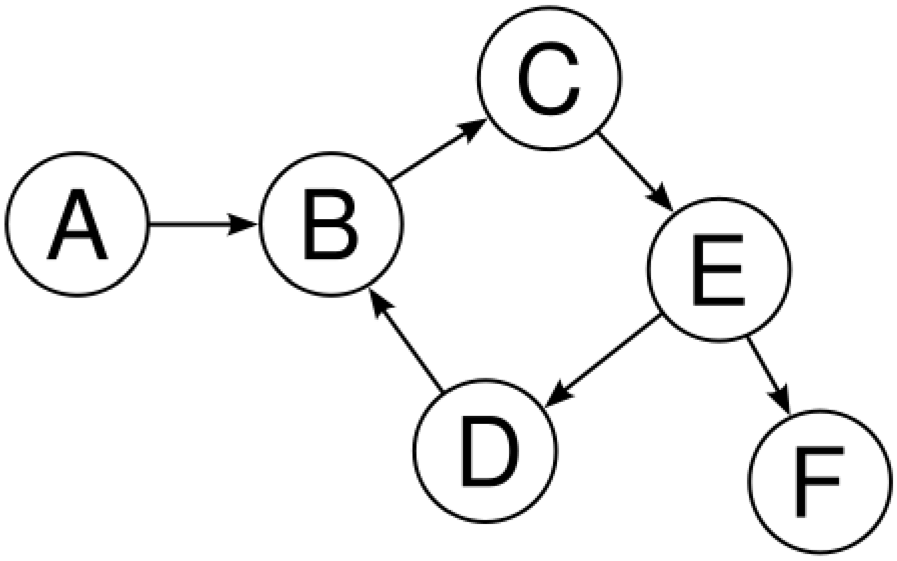


nodes/vertices:a,b,c,d,e
edges

Graph: set of nodes + set of edges

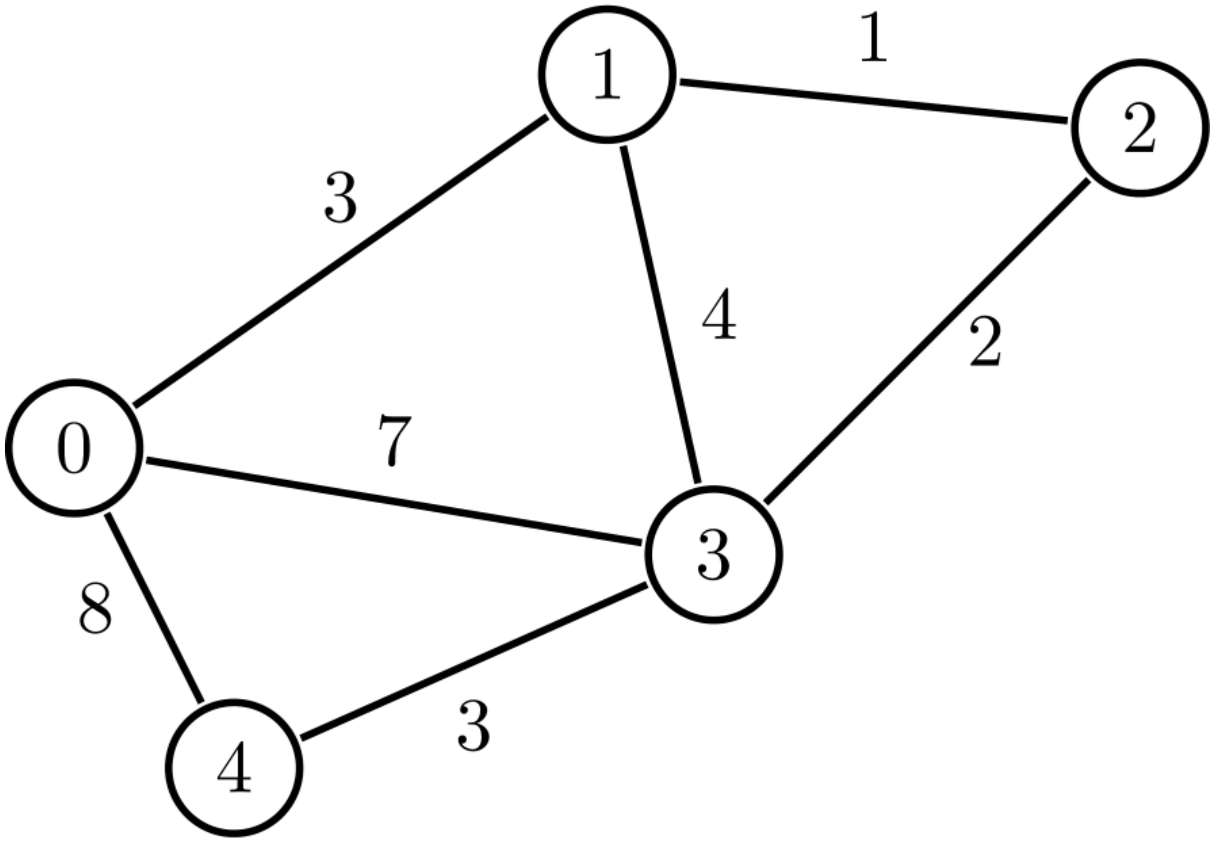
Graph Terminologies:

Directed Graph/Digraph:

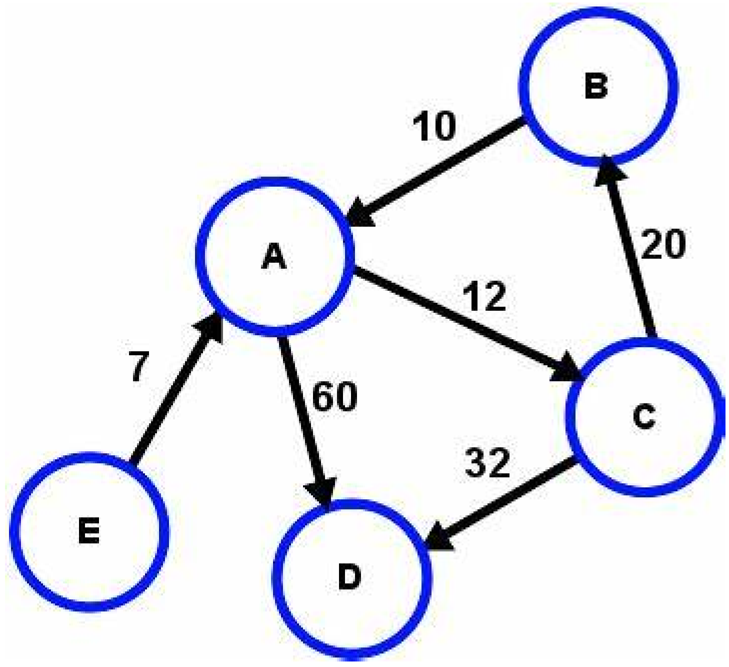


Edges have a direction associated with them
(Observer the arrows in the above graph)

Weighted Graphs:

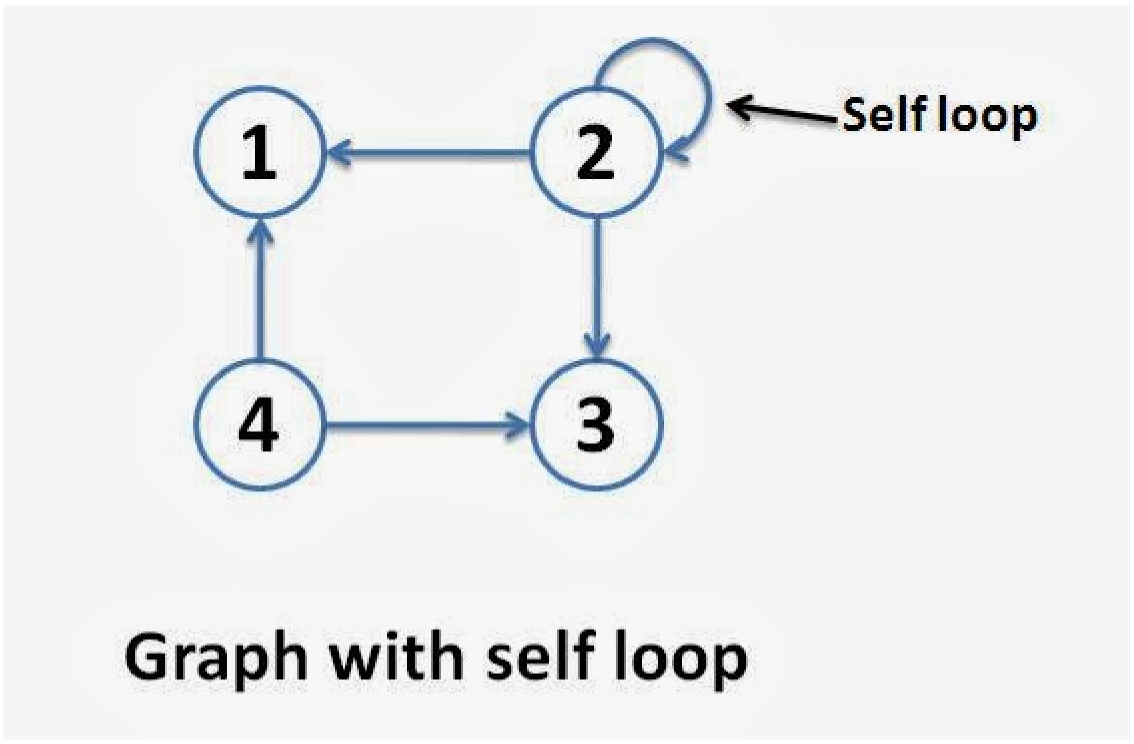


Undirected weighted graph



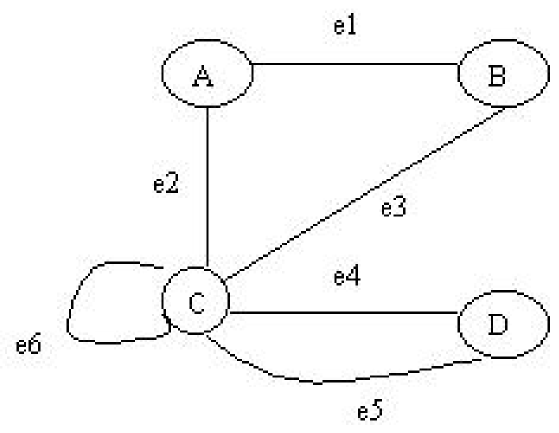
Directed Weighted Graph

Self loops and Parallel edges:



Indegree of 2: 1
Outdegree of 2: 3
Total degree of 2 = 3 + 1 = 4

adj[2][1]=1
adj[2][2]=1
adj[2][3]=1
adj[2][4]=0



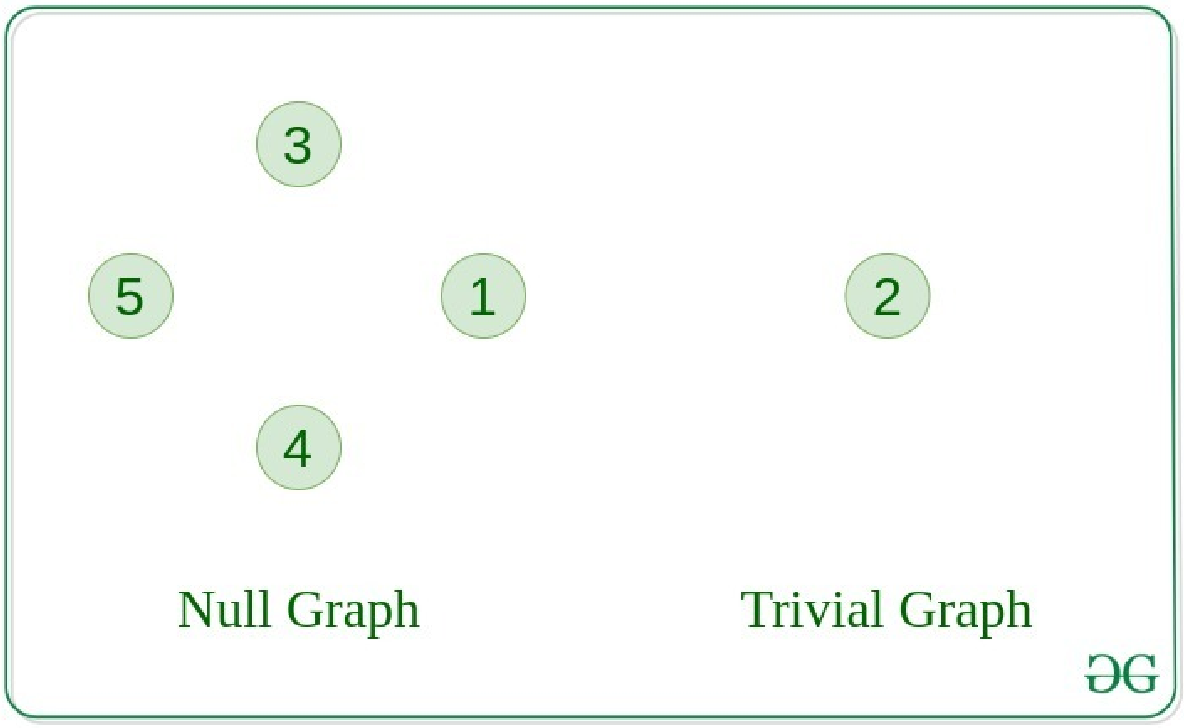
e6-->self loop
degree of C=6

e4,e5-->parallel edges

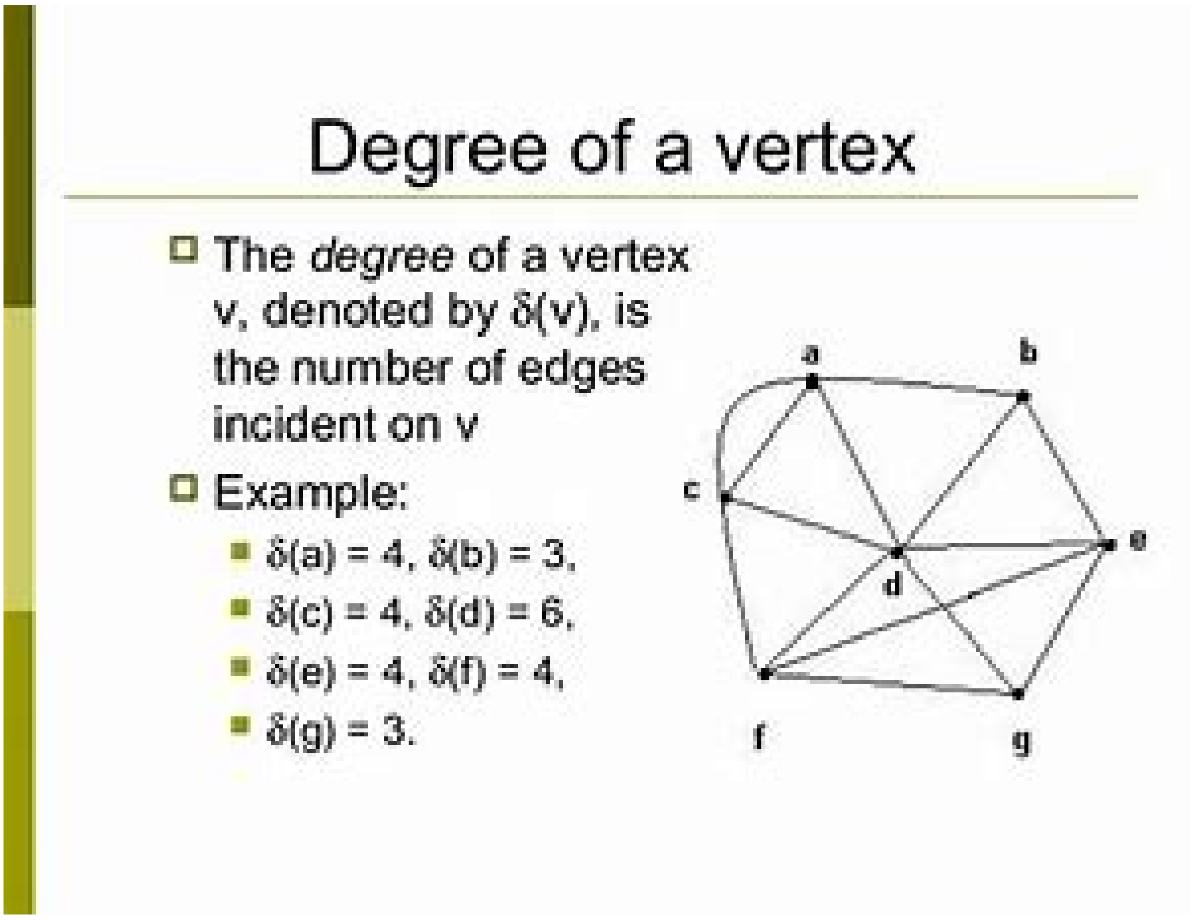
Simple Graphs

A graph having **no self loop** and **no parallel edges**.
(**Important:** In 90% of questions, given graph would be simple. Still, in any question if the graph given is not simple, we will first convert it into a simple graph by removing self loops and parallel edges and then solve the question)

Null and Trivial Graphs


3
5
1
2
4
Null Graph
Trivial Graph

Degree of a vertex in an undirected graph



Degree of a vertex

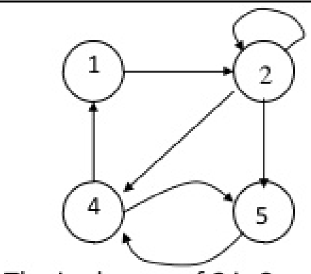
- The *degree* of a vertex v, denoted by δ(v), is the number of edges incident on v
- Example:
  - δ(a) = 4, δ(b) = 3,
  - δ(c) = 4, δ(d) = 6,
  - δ(e) = 4, δ(f) = 4,
  - δ(g) = 3.

Degree of self loop is 2

Degree of a node in directed graph

Degree (Directed Graphs)

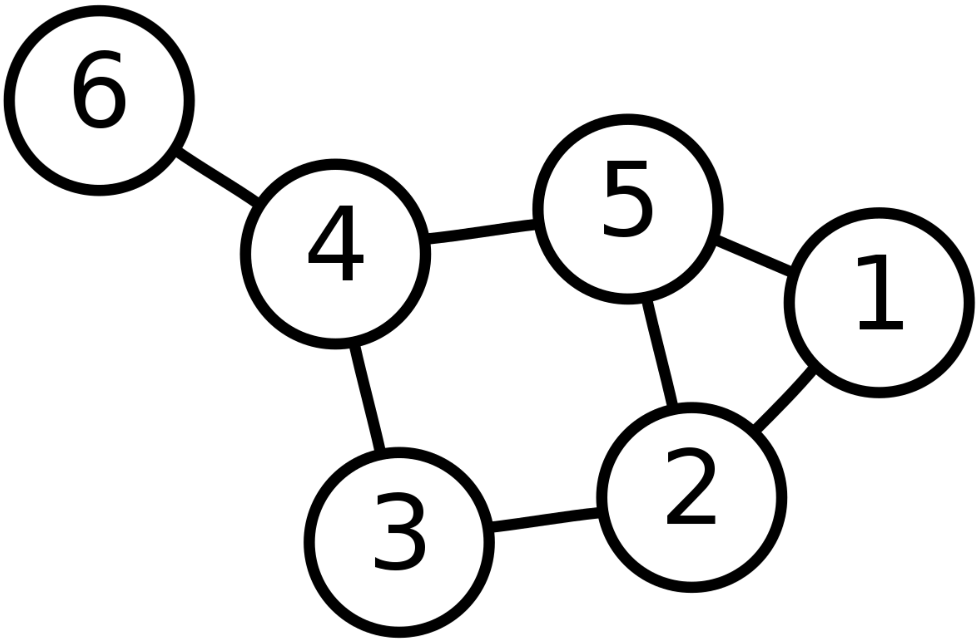
- In degree: Number of edges entering a node
- Out degree: Number of edges leaving a node
- Degree = Indegree + Outdegree



The in degree of 2 is 2 and the out degree of 2 is 3.

Graph Representation

1. Adjacency Matrix:



n -> total number of nodes

n=6

adj[i][j] = 0 if there is no edge between i and j

adj[i][j] = 1 if there is an edge between i and j

adj[n][n] → n X n

adj[1][1]=0

adj[1][2]=1

adj[1][3]=0

adj[1][4]=0

adj[1][5]=1

adj[1][6]=0

adj[3][2]=1

adj[3][4]=1

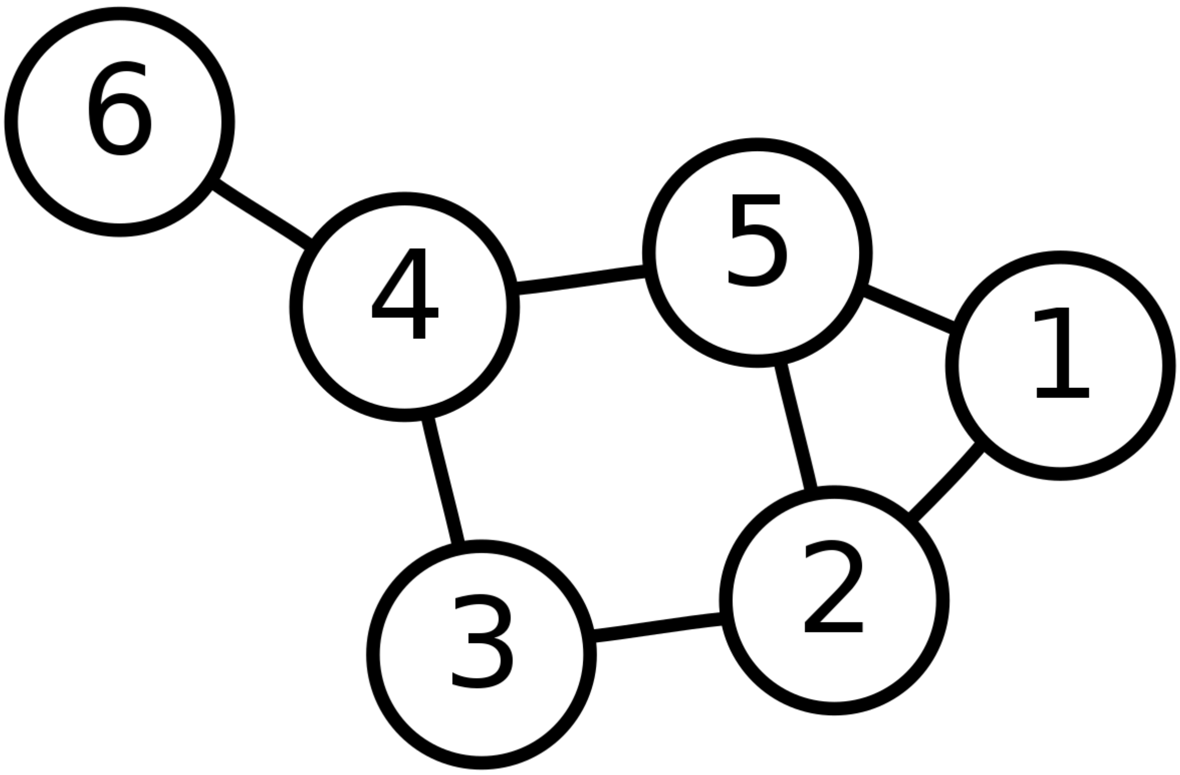
rest all → adj[3][x]=0

Memory complexity: O(n^2)

Lookup time: Given a pair(i,j) , the time to check whether there is any edge between i and j or not, is called <u>loopkup time.</u>

if arr[i][j]==1//exist
else not exist..
So, **Lookup Time in adjacency matrix:** O(1)

2.Adjacency List:



For each i, Store a vector of all neighbours of i
1->2,5
2->1,3,5

3->2,4
4->3,5,6
5->1,2,4
6->4

e edges..
2 4
2 3
v[2]-->sort-->nlogn

2*no of edges..
```
vector<int> v(n)
v[1]-->int

vector< vector< int> > adj(n);
adj[1]-->vector<int>
adj[1].push_back(2)
adj[1].push_back(5)

adj[2].push_back(1)
adj[2].push_back(3)
adj[2].push_back(5)
```

Total no of edges = e
No of nodes: n

Lookup time: O(n)
space complexity: O(e)

0-1,2,3,4

Check if 0-4 forms an edge or not?

```
for(int i=0;i<v[0].size();i++)
 {
if(v[0][i]==4)
  return true
}
```

n-1 nodes..

n-1 times loop

O(n)

space:O(2*e)==O(e)

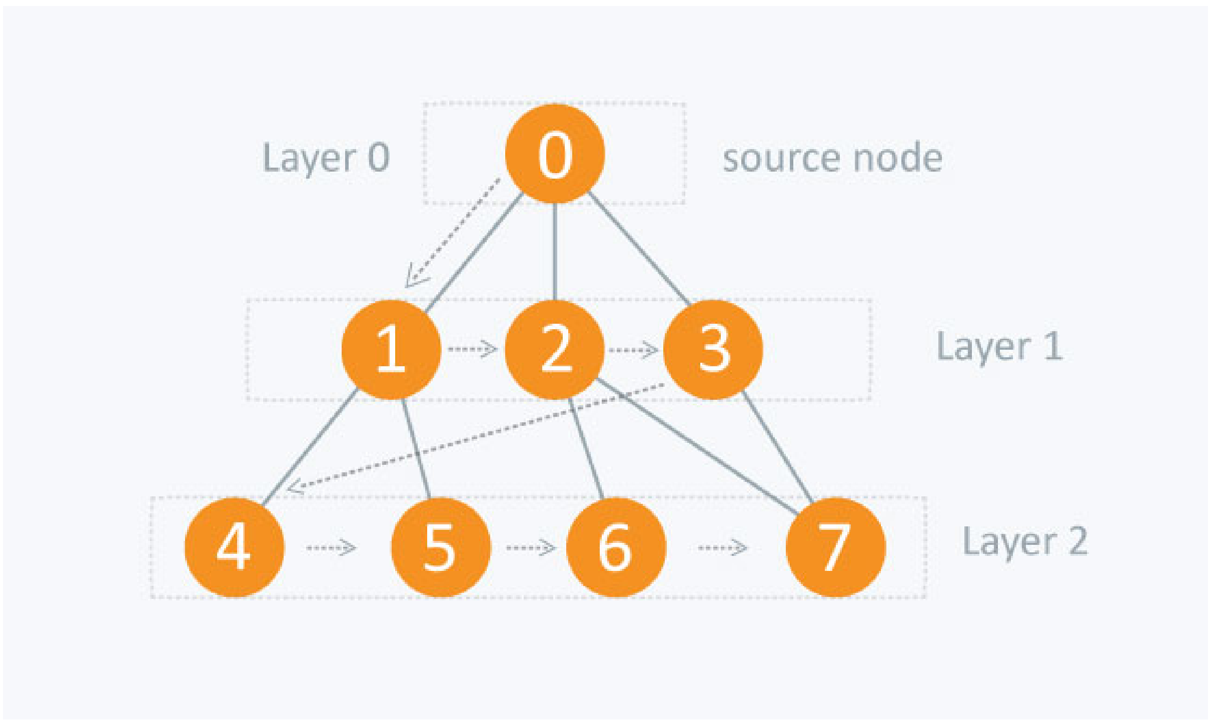
Graph Traversal

2 types : BFS (Breadth First Seach) and DFS (Depth First Seach)

Breadth First Search(BFS):

The algorithm can be understood as a fire spreading on the graph: at the zeroth step only the source s is on fire. At each step, the fire burning at each vertex spreads to all of its neighbors. In one iteration of the algorithm, the "ring of fire" is expanded in width by one unit (hence the name of the algorithm).

We use a **queue** for implementation.



q(0)
0->1,2,3
1->4,5
2->6,7
3->7

queue q
adjacency list: adj
no of nodes : n

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

4,5,6,7
5,6,7

6,7
7

```
queue<int> q;
int vis[n];
for(int i=0;i<n;i++)
 vis[i]=0;
q.push(0); // Assuming source_node=0
vis[0]=1; // visit the source node
while(!q.empty())
{
 int s=q.front();
 q.pop(); // remove front element of q, say s
for(int i=0;i<adj[s].size();i++)
{
// visit all neighbours of s
int ch=adj[s][i]//new child
if(vis[ch]==0)
{ // if child is not visited, push to queue
q.push(ch);
vis[ch]=1;
// lev[ch]=lev[s]+1;
}
}
}
```

Q. How can you find the distance of each node from the source node ? (Let’s call this distance as level of the node)

Solution:
lev[n]-->store level of each node
0->source node

lev[0]=0;
level of child = level of parent + 1
(See the comment in above code)

Q. What is a tree ?
An undirected and connected graph without any cycle.
(We will cover more about trees in next class)

Problems on BFS

Multisource BFS

If there are multiple sources, we will **put the all source vertices to the queue at first** rather than a single vertex which was in case of standard BFS.This way Multisource BFS will first visit all the source vertices. After that it will visit the vertices which are at a distance of 1 from all source vertices, then at a distance of 2 from all source vertices and so on and so forth.

Problem Link: https://www.spoj.com/problems/BITMAP/

```
3 4
2 1 2 3
1 S 1 2
2 1 2 3
```

```
3 4
0 0 0 1
0 0 1 1

0 1 1 0
```

Simple BFS Method: (TLE)

```
queue<int>q
q.push({1,1});



n*m  a=matrix
int ans[n*m];
for(i=0; i<n; i++) for(j=; j<m; j++)
if(a[i][j]==1) v.push_back({i,j})
 (n*m)/2 ,ans[i][j]=0;

for(i=0; i<n; i++){
    for(j=0; j<m; j++){
        if(a[i][j]==0){
          // curr_cordinate= {i,j}
          int min_dist=INT_MAX;
          for(auto k : v){
                int x=k.first;
                int y=k.second;
                int dist=abs(x-i)+abs(y-j);
                if(min_dist>dist)
min_dist=dist;
            }
            ans[i][j]=min_dist;
        }
    }
}
```

Time complexity=O(n*m)^2
It will give TLE

In general,
bfs-> shortest path ya distance from any source nodes
dfs-> to iterate everything (to be covered in next class)

Here, we need to use **Multi-Source BFS**.
(Just push multiple sources in the beginning)
// Breadth First Search
// shortest distance calculator (unweighted or same weighted_graph

// Multi -source BFS

```cpp
#include<cstdio>
#include<iostream>
#include<string>
#include<cstdlib>
#include<queue>
#include<stack>
#include<utility>
#include<string>
#include<cstring>
#include<set>
#include<vector>
#include<fstream>
#include<map>
#include<list>
#include<algorithm>
#define MAX 10000000
#define in_range(x, y, r, c)  (x >= 0 && x < r && y >= 0 && y < c)

using namespace std;

//queue<int> myq;
queue<pair<int,int>>mqq;
void bfs(char mat[182][182], int dis[182][182], int x, int y, int
r, int c){

    int i, j, a, b;

    dis[x][y] = 0;
    myq.push({x,y});

    while(!myq.empty()){

        i = myq.front(),first;
        j = myq.front().second;
        myq.pop();
        //i,j
        a = i-1;
        b = j;
        // dist[a][b] initially INT_MAX hai
        if(in_range(a, b, r, c) && dis[a][b] > dis[i][j] + 1 &&
mat[a][b] == '0'){
```

```
                myq.push({a,b});
                dis[a][b] = dis[i][j] + 1;
            }

            a = i+1;
            b = j;

            if(in_range(a, b, r, c) && dis[a][b] > dis[i][j] + 1 &&
mat[a][b] == '0'){
                myq.push({a,b});
                dis[a][b] = dis[i][j] + 1;
            }

            a = i;
            b = j-1;

            if(in_range(a, b, r, c) && dis[a][b] > dis[i][j] + 1 &&
mat[a][b] == '0'){
                myq.push({a,b});
                dis[a][b] = dis[i][j] + 1;
            }

            a = i;
            b = j+1;

            if(in_range(a, b, r, c) && dis[a][b] > dis[i][j] + 1 &&
mat[a][b] == '0'){
                myq.push({a,b});
                dis[a][b] = dis[i][j] + 1;
            }

        }
}

int main(){

    char mat[182][182], ch;
    int t, r, c, dis[182][182];
    scanf("%d", &t);
    while(t--){
        scanf("%d%d%c", &r, &c, &ch);
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++){
                scanf("%c", &mat[i][j]);
                dis[i][j] = MAX;
            }
            scanf("%c", &ch);
        }

        for(int i=0;i<r;i++)
            for(int j=0;j<c;j++){
                if(mat[i][j] == '1')
                    bfs(mat, dis, i, j, r, c);
            }

        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++)
                printf("%d ", dis[i][j]);
            printf("\n");
        }
    }

    system("pause");
    return 0;
}
```

Time complexity=O(n*n*m)

// multi-source bfs
// 0-1 bfs

// you can apply bfs only if all edges have same weight

queue<int>q
        0 0 0 0 0 0 0- - - - - - - - - - - - - - - -

// Single source bfs

        vector<vector<int>>adj; {i,j} i <-> j undirected
        weight=1
        all edges have weight 1

        source vertex=1
        find shortest distance for all vertex from 2 to n from
        1

```
int dist[n] -> dist[i] shortest distance of i from vertex
1
// single -source bfs
queue<int>q;
int dist[n]={inf}
dist[1]=0;
q.push(0);
while(!q.empty()){
        int x=q.front();
        q.pop();
      for(auto i : adj[x]){
              // i node connected to x directly
              int d=d[x]+1;
```

```
            if(dist[i]>d){
             dist[i]=d;
             q.push(i);
          }
      }
}

for(i=2; i<n; i++) cout << dist[i] << ‘ ‘;
```

// dijkstra algorithm O(E*V) ALL edge may have different weight
//
// 0-1 BFS ->O(E)

Q. Find shortest distance of each node from source node, if the weights of edges can be either 0 or 1.

a->b weight 0 or 1
In queue
queue<int>q;
q.push(1); // back me push karta
q.pop(); // front se pop karta

dequeue<int>q; // important topic
push_back(), push_front()
pop_back(), pop_front()

If weight of edge is 0 -> push that in front

If weight of edge is 1-> push than in back
And the rest of the BFS remains the same.
Using this technique, we can find shortest distance.
This is called **0-1 BFS.**

Try this Problem:
https://www.spoj.com/problems/KATHTHI/

0-1 bfs

```
#include<cstdio>
#include<iostream>
#include<string>
#include<cstdlib>
#include<queue>
#include<stack>
#include<utility>
#include<string>
#include<cstring>
#include<set>
#include<cmath>
#include<vector>
#include<fstream>
#include<map>
#include<list>
#include<algorithm>
#define INF 100000000
#define in_range(x, y, r, c) (x >= 0 && x < r && y
>= 0 && y < c)

typedef long long int LLD;
typedef unsigned long long int LLU;

using namespace std;

char mat[1000][1000];
int dis[1000][1000];


void init(int r, int c){
    for(int i=0;i<r;i++)
        for(int j=0;j<c;j++)
            dis[i][j] = INF;
}

void bfs(int startX, int startY, int r, int c){
    dis[startX][startY] = 0;
    deque< pair<int, int> > q;
    pair<int, int> p;
    q.push_front(make_pair(startX, startY));
    while(!q.empty()){
        p = q.front();
        q.pop_front();
        int x = p.first;
        int y = p.second;
        int a[] = {0, -1, 0, 1};
        int b[] = {-1, 0, 1, 0};
        for(int i=0;i<4;i++){
            int tmpX = x + a[i];
            int tmpY = y + b[i];
            if(in_range(tmpX, tmpY, r, c)){
                if(mat[tmpX][tmpY] == mat[x][y] &&
dis[tmpX][tmpY] > dis[x][y]){
                    q.push_front(make_pair(tmpX,
tmpY));
                    dis[tmpX][tmpY] = dis[x][y];
                }
                else if(mat[tmpX][tmpY] !=
mat[x][y]){
                    if(dis[tmpX][tmpY] > dis[x][y] +
1){
                        q.push_back(make_pair(tmpX,
tmpY));
                        dis[tmpX][tmpY] = dis[x][y]
+ 1;
                    }
                }
            }
        }
    }
}

void display(int r, int c){
    for(int i=0;i<r;i++){
        for(int j=0;j<c;j++){
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
    for(int i=0;i<r;i++){
        for(int j=0;j<c;j++){
            cout << dis[i][j] << " ";
        }
        cout << endl;
    }
}

int main(){

    int t, r, c;
    char ch;
    scanf("%d", &t);
    while(t--){
        scanf("%d%d", &r, &c);
        scanf("%c", &ch);
        for(int i=0;i<r;i++){
            for(int j=0;j<c;j++)
                scanf("%c", &mat[i][j]);
            scanf("%c", &ch);
        }
        init(r, c);
        bfs(0, 0, r, c);
        cout << dis[r-1][c-1] << endl;
    }
    return 0;
}
```