**Collection of Vertices(Nodes) and Edges**

**Graphs**

**EDGE**

**Vertices**

Toh, graph is nothing but the non – linear data structure having nodes those are connected through edges. Apna Tree bhi ek prakar ka graph hi h.

**Edge: Finite set of ordered pair of form (u, v) called as edge.**

**(u, v) != (v, u) in directed graph**

\*\* Edge may contain weight i.e. cost, value, time. \*\*

Dekho, node kya h…. DATA and data connected kaise h ye edge batata h.

Aur edge basically 2 type ke hote h:

1. **Directed Edge**: One way connection

Mtlb N3 se N5 related h but

N5, N3 ka frnd nai h.

1. **Undirected Edge**: Two way connection.

Yaha N3 and N5 dono ek dusre ke frnd h

Aur apan n3 se n5 and vice versa

Travel kr sakte h.

Ye double arrow ya no arrow, dono se signify hota h.

Jisme saare edges directed ho usko **directed graph** and jisme saare undirected ho, unko **undirected graph** kehte h.

Vese toh graph kai jagah use hota h but tumhaare hisaab se ek mst sa example deta hu,

Toh… Facebook ka naam suna h?

Han voi wala, usme connections graph ke through hi h, jaise necche deko

Ab, dekho, ye facebook h, isme chahe rudra Rudra

Request bheje ya mai bheju, accept krne mai

Hum dono frnd keh layenge, mtlb hum double Chu

Sided edge se jude h, Hence Fb is an Shrey

Example of undirected graph.

Anuj

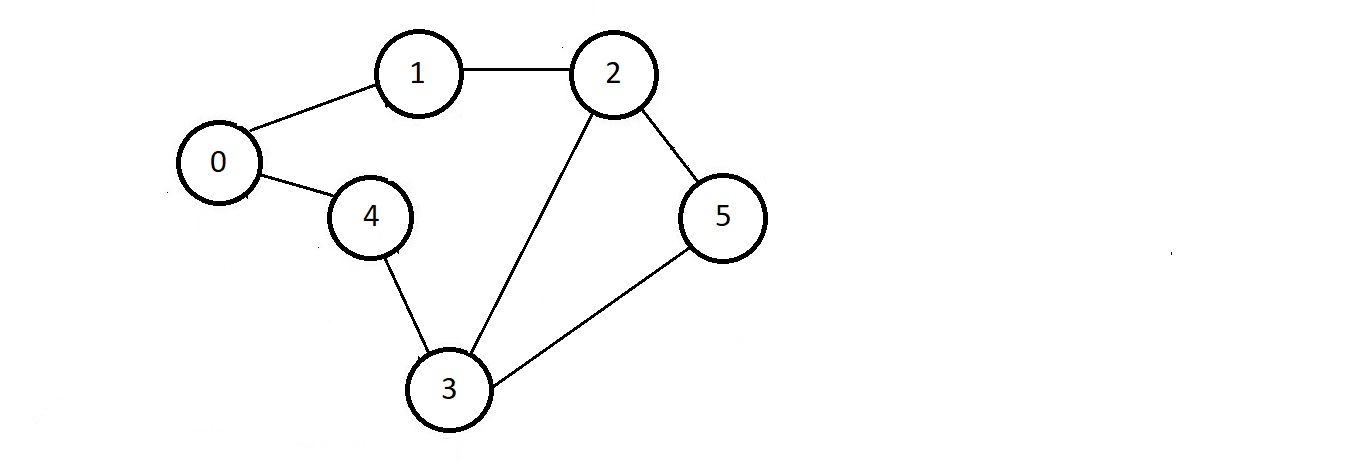
**Undirected Graph**

(Most probably meko Anuj recommended mai aaye) Dikshita

Ab Insta ka example lete h… Insta mai agar rudra ne meko follow request bheji toh chance h ki mai uski accept kr lunga but mai follow back na karu. Ye aa gaya directed connection mai. Toh basically insta ek normal graph h usme directed and undirected dono edges bani hui h.

**G = (V, E)**

**Graph G is the collection of vertices and edges connecting these vertices.**



V = {0, 1, 2, 3, 4, 5} E = {(0,1), (1,2), (2,3), (2,5), (3,5), (3,4), (0,4)}

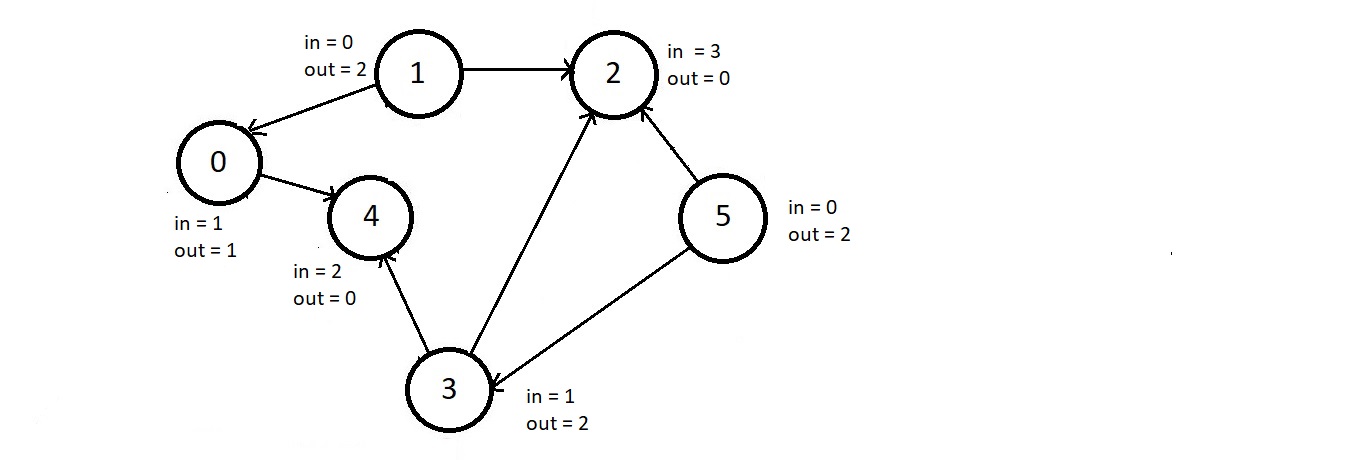
V is the set of all vertices and E is the set of all edges. Edge connecting node

N1, N2 is represented as (N1, N2). And hence a Graph is represented as:

**G = (V, E)**

**Indegree of a node:** No. of Edges coming to the node. Also known as just “in”

**Outdegree of a node:** No. of Edges leaving the node. Also known as just “out”



Toh Dekho yaar, graph har jagah h. Graph concept ke bina social network(Fb),

**Application of Graph**

Map network(GMaps), aur yaha tk ki complex websties possible bhi nai ho skti thi. Jaise,

Katni se Indore ke liye koi direct route h ya nai?,

oneway h ya 2 way?

Fb mai kon kiska friend h?

Aur INTERNET ka naam suna h na? Toh internet kaise kaam krta h?

Accha google ka home page kholte h toh YouTube dikhta h na? Pr usme click krne ke baad youtube se wapas google ka koi option nai rehta…..Ab agar ye isko kisi data type mai store krna chaahe toh graph use karenge aur directed edge se link rahega google and youtube.

Aise hi kon si site kis se link h ya kisi ek website mai kon sa page kis page se link h ye sb graph ka use krke easy store and annalyse kr skte h apna.

Different web browsers apne search results aise hi data ko annalyse krke show krte h aur improve krte h.

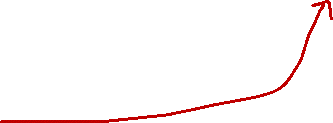
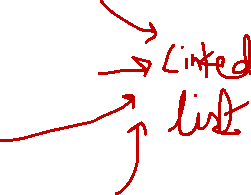
Baaki isko represent kaise krte h ye neeche dekhte h…..

**Representation of Graph**

There are many types of representation of graph but we will be looking at some most used types:

1. **Adjacency List:** Its well pretty simple way to represent a graph in which we simply are chod yaar, dekh ek ek kr ke node ko pakdna chaalu kr, aur as us node se jo jo connected h usko ek linked list mai store kr le aur fir bs us list ke head ko node se point kara de, simple.

Han han pata h nai smjhe neeche dekho:



R

D

S

R

S

A

J

S

A

D

A

R

It’s a simple array which stores the data and pointer to head of corresponding linked list.



1. **Adjacency Matrix:** It is even simpler representation then that of above one as there is only one 2D array used. Chalo bahut English ho gai, suno sb, tic tac toe khele ho na? bs voi krna h, Maan lo matrix be

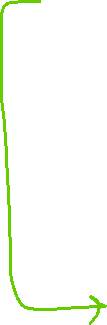
Aij – i row h aur j column ab agar i, j se connected ho toh true bole toh 1 likh dena nai toh zero likh dena….Simple!

Angrezi mai bole toh:

Aij = 1, if there is an edge between i and j, 0 otherwise.

If i = j, then if there is a self loop then 1 or else 0.

Pata h ye bhi palle nai pada, toh ye dekho neeche(upper wale graph ka)



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | S | R | A | D | J |
| S | 0 | 1 | 1 | 1 | 0 |
| R | 1 | 0 | 1 | 0 | 1 |
| A | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 0 | 1 | 0 | 0 |
| J | 0 | 1 | 0 | 0 | 0 |

\*\* S khud se bhi connected ho skta h, us condition ko self loop kehte h and A22 Yahi pooch ra h, since S donot have self loop isiliye zero likha h. \*\*



# Apna row se padhte h mtlb, A23 ko read karenge, is S connected to R? aise hi baaki read karenge,

A22 ko, Is A self looped?

\*\*\*\*\*Ek baat notice karna, apan ka jo uper waala graph h vo undirected graph h isiliye dono side se connection h, isiliye agar A23 (S connected to R) true h toh A32(R connected to S) bhi true rahega. \*\*\*\*

Ab easily apan matrix ko transverse kr ke apna graph bana skte h. Just by seeing 0 and 1.

1. **Cost Adjacency Matrix:** Ab Maan lo agar nodes kisi city to represent kr raye h, jaise

S: Sagar, R: Rishikesh, D: Delhi, A: Amritsar and J: Jammu



Then ek jagah se dusri jagah jaane ke liye kharcha lagega ya nai, Kharcha kisi bhi tarah ka ho skta h jaise time ka kharcha, paisa, energy, etc. Ye sb ek jagah se dusri jaane ka cost kehlaaya.



Let cost be as shown here,

Now on ticket there were

a offer of free ticket from

A to R hence cost is Zero.

Ab bs adjacency matrix waali matrix mai hi bs 1 ki jagah

Cost likh do and 0 ki jagah bhi chalta but size 0 can be the cost isliye 0 ki jagah -1 likhenge.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | S | R | A | D | J |
| S | -1 | 50 | -1 | 150 | -1 |
| R | 50 | -1 | 0 | -1 | 500 |
| A | -1 | 0 | -1 | 300 | -1 |
| D | 150 | -1 | 300 | -1 | -1 |
| J | -1 | 500 | -1 | -1 | -1 |

**\*\* Ab uper waala concept tum saaf saaf dekh skte ho, since graph undirected h isiliye har ek cost 2 jagah likhi h \*\***

**Agar 0 nai bhi hota tb bhi is type ke graph mai false ke liye -1 use krna.**

Dekho yaar aur bahut prakar ke representations hote h, but vo jyada use nai hote, mtlb use krna pad bhi skta h agar DSA ke kisi question mai aaya toh, isiliye fatak se dekh lo:

1. Edge Set: Are normal re jaisa starting mai set banaya tha na apan ne, vessa. Aree vo agar node n1 and n4 connected h toh (n1, n4) set mai daal do, vesa waala.

E = {(S,R), (S,D), (R, A), (R, J), (A, D)} (jaruri nai ye sahi ho, search kr lena)

Well ye apan ko dekh ke smjhne toh mst h but computer babu nai smjh paata aasaani se isiliiye use nai hota ye jyada.

**Cost Adjacency List:**Similar to the adjacency list, but instead of just storing the node value, we’ll also store the cost of the edge too in the linked list. Arree jo linked list banate h na usme bs ek data value aur add kr dete h, Cost add krne.

**Compact List Representation:**here, the entire graph is compressed and stored in just one single 1D array. Isse dur hi raho re baba…..

**Adjacency Matrix(AM):** V x V Matrix, where v = vertex

**Pro’s and Con’s of Above rep.**

**If Adj[i][j] = 1,**

Indicate there is a edge between i and j.

**# AM is always symmetric for undirected graph.**

**PROS:**

1. Easier to implement.
2. Removing an edge take O(1) time complexity.
3. Checking edge between 2 vertices also takes O(1).

**CONS:**

1. Takes more space i.e. O(V2))

(even our graph is a linear i.e. like a linked list, AM takes V2 space).

1. Adding a vertex is O(V2) time.

**Adjacency List:** size of array = No. of vertices.

**PROS:**

1. Saves space, only takes O(|V| + |E|) space.

(Where v= No. of vertex and e = No. of edges.)

In Worst case(every node is connect to each other G(V, 2), it consumes O(V2) space.)

1. Adding a vertex is easy.

**Cons:**

Checking edge between 2 vertex is not efficient and take O(V) time.

Visiting every node of graph is known as transversal of graph.

**Transversal of Graph**

There are 2 main and important ways to transverse a graph:

1. BFS – Breadth First Search
2. DFS – Depth First Search
3. **BFS:**  Bahut simple algorithm h re graph transverse krne ka, toh chalo fir fatak se krte h, BFS ke liye apan ko 2 cheezo ki zarurat rahegi, ek visited list(array) and ek ExploreQueue(queue).

Dekho, pehle toh graph ka koi bhi ek node pakad lo, aur usko queue mai and visited mai daal do.

AB bs krna aisa h ki queue se top element pop karo and jo element nikla h usse connected vertex AGAR VISITED LIST MAI NA HO toh visited mai and queue mai, dono mai daal do. Aur bs aise hi krte raho jb tk queue empty na ho jaaye.

Aur jb queue empty hoga toh, TADAA… visited list mai saare elements aa chuke h.

Ab dekho yaaar, abhi tk maine jaha dekha h waha graph mai data store krne ke bajaye, index ko vertex kaha gaya h and vo kiskis se connected h ye important h, isiliye algo mai

Jb check krna padta h ki ye visited mai h ya nai, ye na krna pade isiliye array ke index ko vertex maana gaya h and jb visit kr lete h toh uss visited array ke uss index ko 1 kr dete h(initially saare 0 kr dete h).

Acche se smjhne Graphs\_BFS.cpp and isi ka matrix waala h, vo dekh lo.

1. **DFS:** Graph\_DFS\_Transversal.cpp mai bahut acche se smjhaaya h, dekh lo.

**Connected Graph:** Graph in which there is a path to go from any node to any other node. Seedhi bhaasa mai, vo graph jiska koi bhi vertex disconnected na, aisa na ho ki vo kisi vertex se connected nai h.

**Some Important Terms**



1. **Complete Graph:** Vo graph jiski har vertex baaki saari vertices se connected ho. Bole toh, A graph in which there is a edge between any 2 vertices.

**Every node connected to**



**each other**

1. **Subgraph:** A graph whose vertices and edges are subset of graph G = (V,E). Bole toh, kisi graph ka koi ek part utha ke ek dusra graph bana do (let S) jiski saari vertices and edges Bade graph G mai ho.

**S(V, E) C G(V, E)**



**A subgraph (let it be s) is said to be a spanning tree of graph G(v, e) if and only if it satisfies following points:**



**Spanning Tree**

1. **Subgraph must be a connected Graph.**
2. **Every vertex of G must be present in S(subgraph).**
3. **No. of edges in S must be exactly equal to 1 less than the number of vertices(Nodes) in graph G.**

**Total number of edges = |V| - 1**

**Where V is no. of vertices in graph G**

**No. of Spanning trees a complete graph can have = n(n-2)**

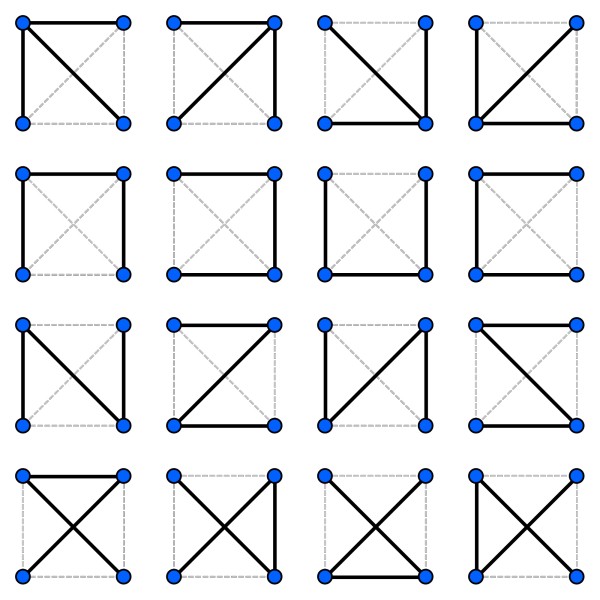


One of the spanning tree of tis graph, since the subgraph looks like tree, hence known as spanning tree.

**Spanning tree of this connected Graph:**

No. of spanning trees it can have =

**n(n-2) = 4(4-2) = 16**



**Spanning Tree Cost = Sum of weights of all edges.**

**Cost of Spanning Tree**

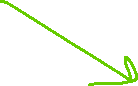
Consider below graph, let Alphabets represent the citys and numbers written on edges represent the travel cost between those cities.



Spanning tree



Cost = 10+2+5+0 = 17



Cost = 5 +20+0+2 = 25 Cost = 5 +20+10+0 = 35



**The spanning tree having minimum cost is known as Minimum Spanning Tree.**

Also known as Greedy Approach

**Prim’s Algorithm**

Now, as you have seen above that any graph can have many spanning tree, hence to find the minimum spanning tree this algo is used.

In this Algo, we need to maintain 2 sets,

V: set of vertices left to visit, A: set of vetices visited

Initially V have all the vertices except any one(from which you want to start) and A only have that one starting node.

now, consider below graph G, and let’s start with node S,

V = {A, F, J, D}, A = {S}



Dekho, poore algo kaa saar bs ye h ki A set mai jo nodes h, unsse connected saari edges ko dekho, aur jo sbse km mile usse apne subgraph(spanning tree) mai add krlo(aur A set mai daaal do, and V se hata do).

Below is the spanning tree of above graph using Prim’s algo:



Our spanning tree

V = {A, F, J, D}, A = {S}

Check cost of every edge connected to S. F is least cost edge.

V = {A, J, D}

A = {S, F}

Now check cost of every node connected to S and F, least is F-J.

V = {A, D}

A = {S, F, J}

Again, check least cost edge of S, F, J. S-A and S-D are of very high cost while F-J is already present, left F-A,

V = {D}

A = {S, F, J , A}

Finally, Check every edge connected to S, F, J, A. an as we can see,

A-D is free of cost hence,

V = {}

A = {S, F, J, A, D}

Consider One more:



Starts with 1

|  |  |
| --- | --- |
| **V** | **A** |
| {0, 2, 3, 4, 5 ,6} | {1} |
| {0,2,3,4,5} | {1, 6} |
| {0,2,3,4} | {1,6,5} |
| {0,3,4} | {1,6,5,2} |
| {0,4} | {1,6,5,2,3} |
| {4} | {1,6,5,2,3,0} |
| {} | {1,6,5,2,3,0,4} |

* Above list sequence number(jis order mai edge add ki)



* Cost

COST : 6+11+5+10+17+2 = 51