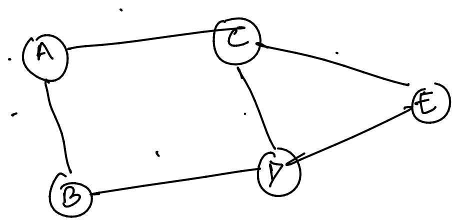
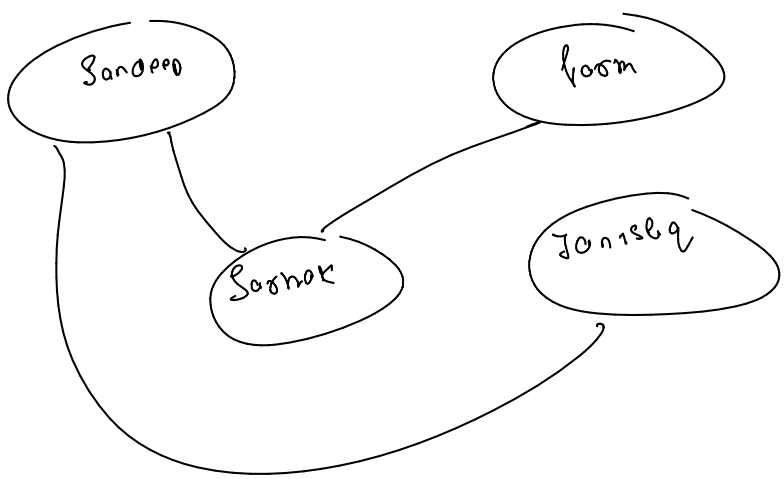
23 August 2021 19:01

Graphs

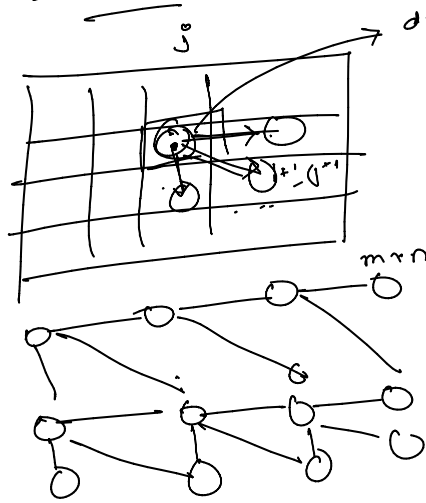
collection of nodes (vertices) and edges.

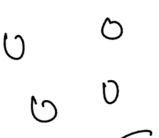




Internet

$dp(i)(j) = f(i,j)$





null graph

$\Rightarrow$ $G = (V, E)$

E → set of pairs of vertices, each e. called edge

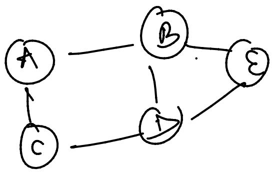
V → arbitrary non-empty ... whose

null graph

ϕ

arbitrary non-empty set, whose element is called vertex

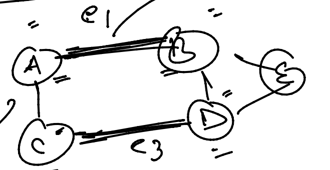
called edge



$V = \{A, B, C, D, E\}$

$E = \{(A,B), (A,C), (C,D), (B,D), (B,E)\}$

Undirected Graphs



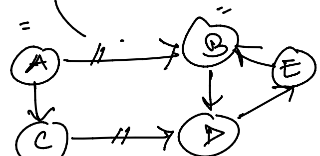
$\{A, B\}$ or $\{B, A\}$

$\{A, C\}$ or $\{C, A\}$

$\{A, B\} \leftarrow$ set

tail head

(A, B)

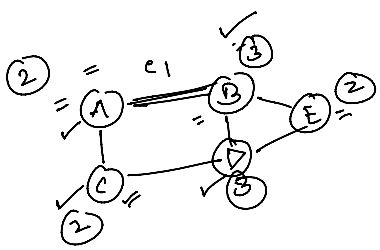


(C, D)

or

$C \to D$

Directed Graphs

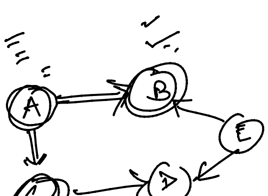


Adjacent nodes

Neighbour

Degree of nodes $\Rightarrow$ # neighbours

in $= 0$
out $= 2$



in $= 0$
out $= 2$

in $= 1$
out $= 1$

$A \to B$

A is a prec. of B

or

B is a succ. of A

In-degree

Out-degree

$|E|_{min} = 0$

Undirected Graph

$|V| \Rightarrow$ total vertices

$$= |E|_{max} = \frac{|V|\,(|V|-1)}{2}$$

A — B

B — A

A → B

B → A

$$\Rightarrow {}^{|V|}C_2$$

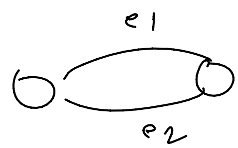
Directed Graphs

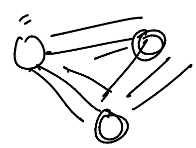
$$= |E|_{max} = |V|\,(|V|-1)$$

complete Graph

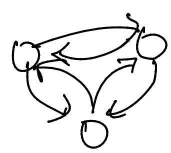
$|V|-1$

Parallel Edges





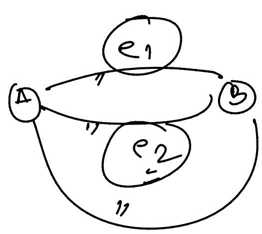
K_3



Parallel Edges

undirected

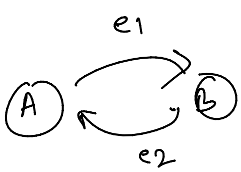
{A, B}

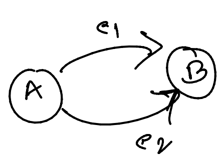


{A, B}

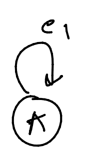
$e_1 = A \rightarrow B$

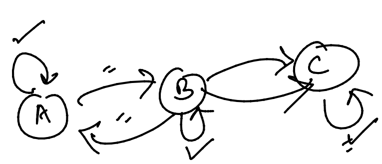
$e_2 = B \rightarrow A$





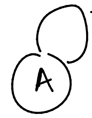
Loops





multiset

undirected edge $\Rightarrow$ $\boxed{\text{set}}$ of size 2



$\Rightarrow \{A, A\}$ ×

directed graph

Simple Graphs

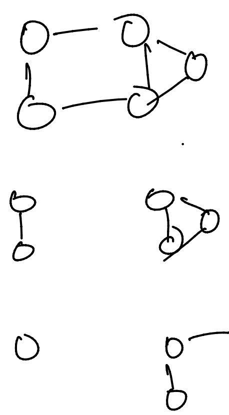
Multigraph Graphs

Subgraph

$G = (V, E)$

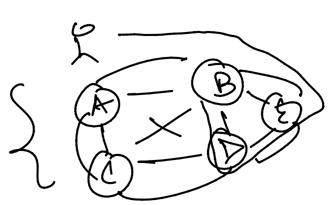
$G' = (V', E')$

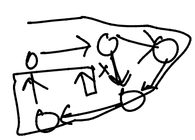
$V' \subseteq V$ and $E' \subseteq E$

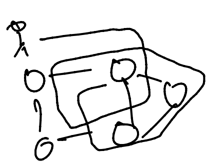


Proper Subgraph

Walk







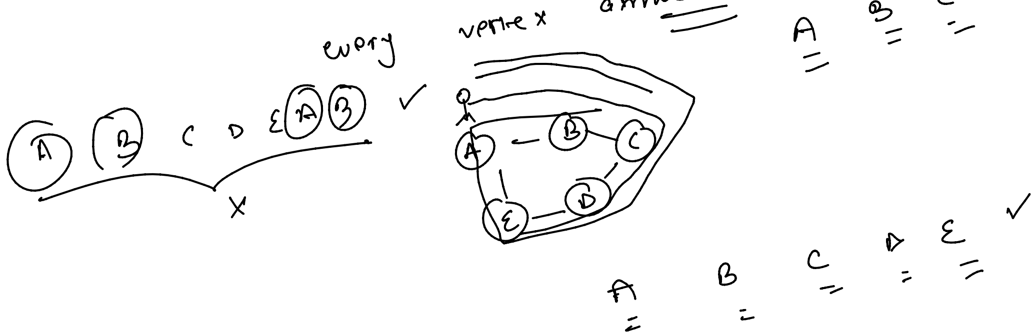
A B C D E E × walk

✓ A B E D C A

it visits

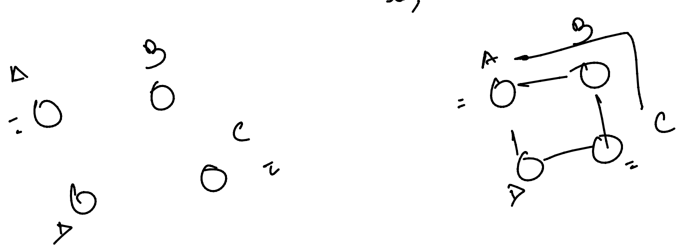
Path

A walk is a path if it visits every vertex atmost once

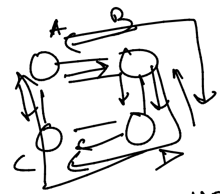


Reachability

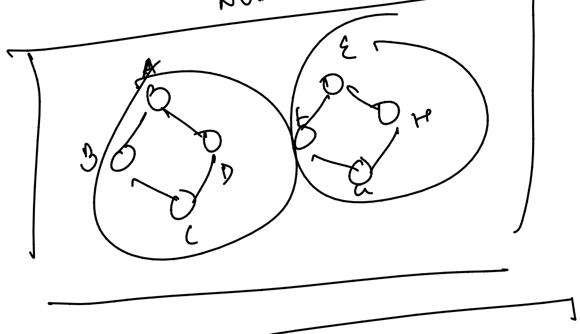
u, v



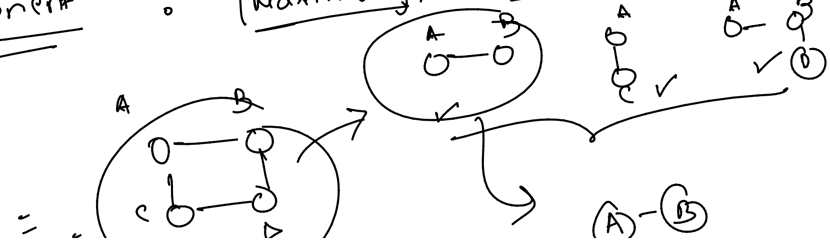
Connected Graph

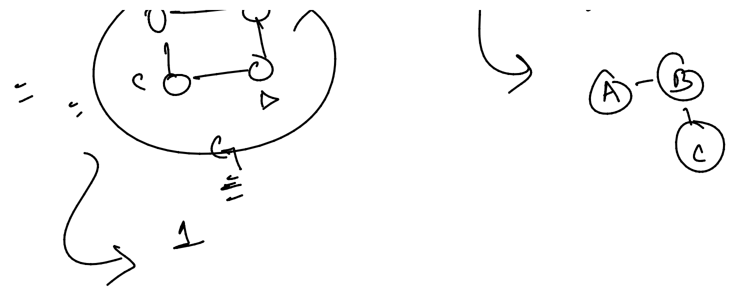


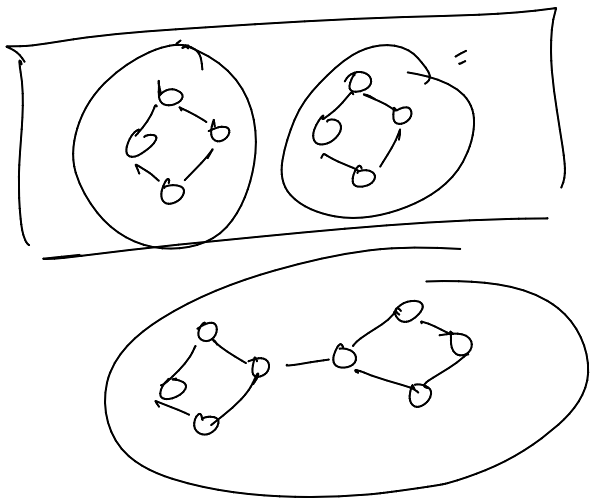
Not connected - Disconnected



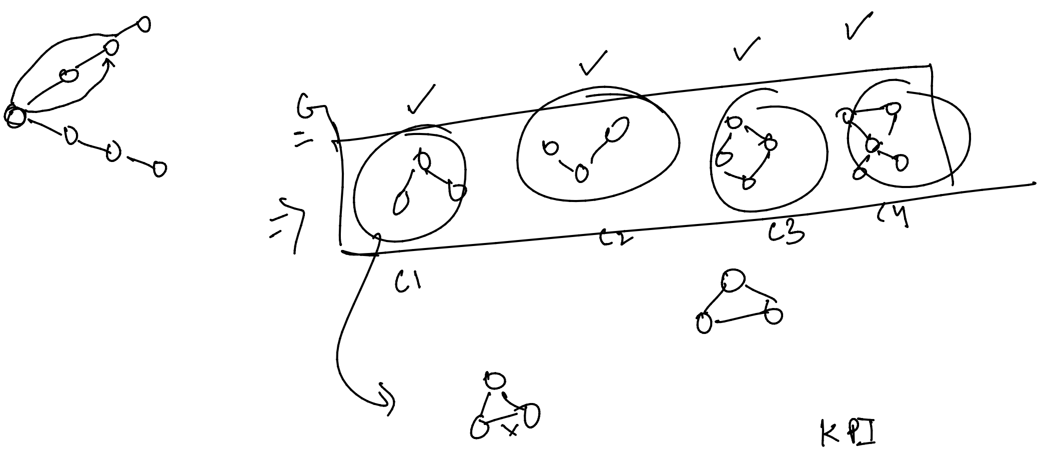
Component : $\boxed{\text{maximally}}$ connected subgraph





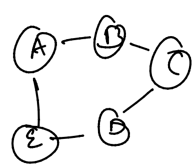


Every G has >= 1 component



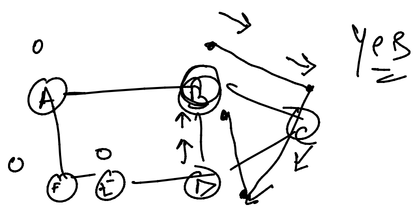
KPI

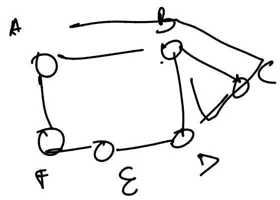
Closed walk



$\underbrace{A \quad B \quad C \quad D \quad E}$ A

Cycle : a closed walk that enters and leaves every vertex almost once.





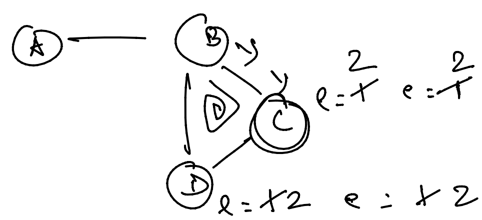
A $\boxed{B \quad C \quad D \quad B}$

walk ✓

closed ✗

$\underbrace{B \quad C \quad D \quad B}_{\text{closed walk ✓}}$ walk

cycle ?



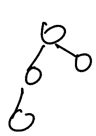
$\underbrace{B \quad C \quad D \quad B \quad C \quad B}$

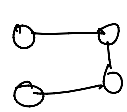
walk ✓

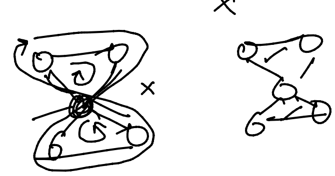
closed walk ✓

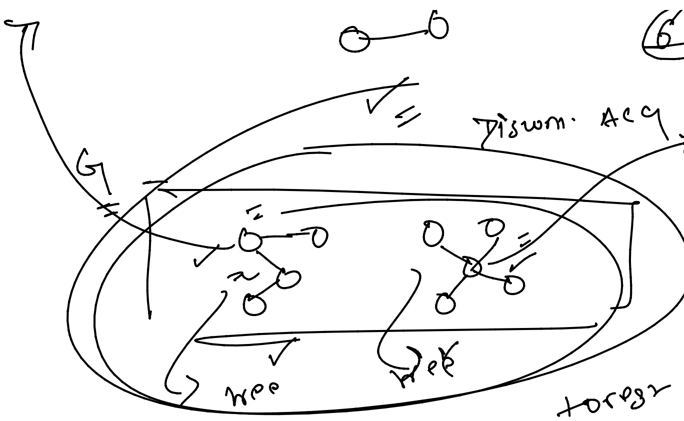
cycle ✗

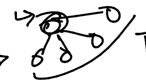
Acyclic Graph => it is a graph with no cycles i.e none of its subgraph is a cycle











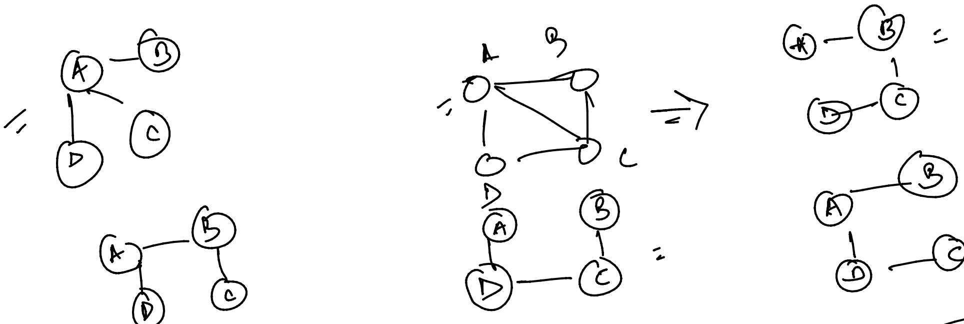
Tree

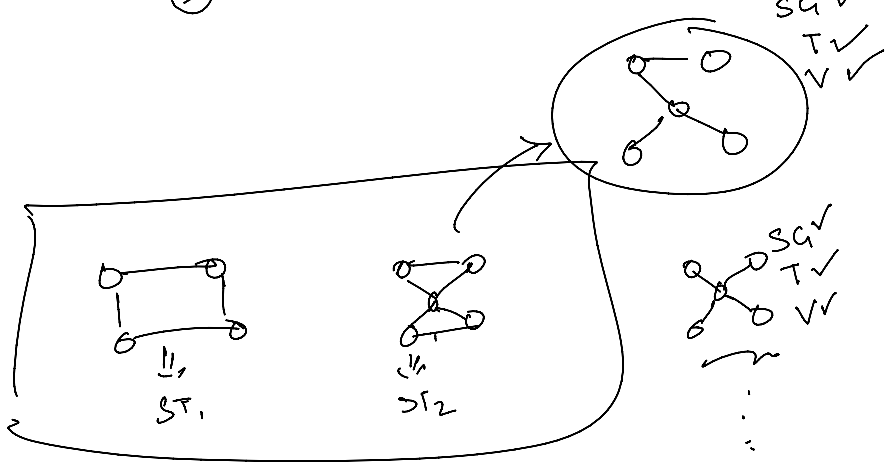
↳ acyclic connected graph

collection of tree is forest

connected acyclic graph

Spanning tree is a subgraph → tree → contains all v of G





Spanning forest

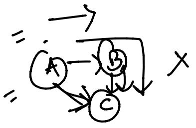
Directed Graph

walk ⇒ directed walk

path ⇒ directed path

cycle ⇒ directed cycle

Component

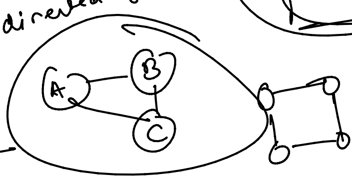


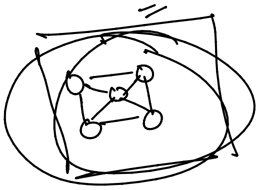
A → B ✓

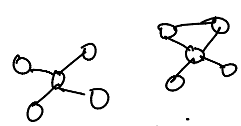
A → C ✓

C → A

Connected

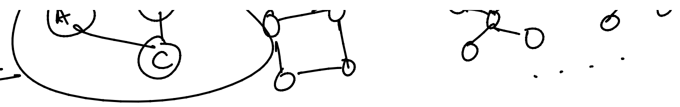




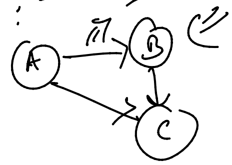


A → B ✓ C → A
A → C ✓
B → C ✓ C
B → A ✗
Strong Conn.

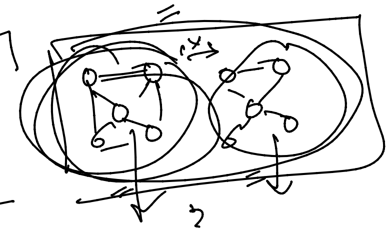
Connected
Component



Acyclic Graphs



G

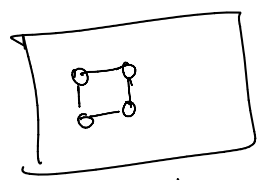


2

Weakly Conn.

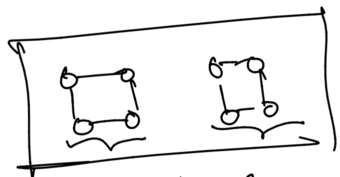
✓ A → B C → A ✗
✓ A → C C → B ✗
✗ B → A
✓ B → C

G



= 1

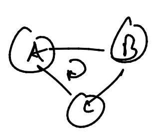
G



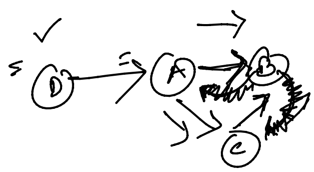
: 2

DAGs ⇒ directed acyclic graphs

undir

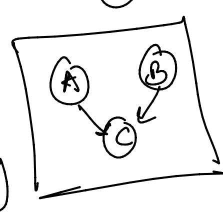


✗



DAG : ?

DAG
⇓
rooted tree



Acyclic Conn. Graph
⇓
tree

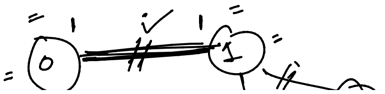
Most popular 99.990% 100%

1. Adjacency List
2. Adjacency Matrix

array that are indexed by vertices
⇓
integers

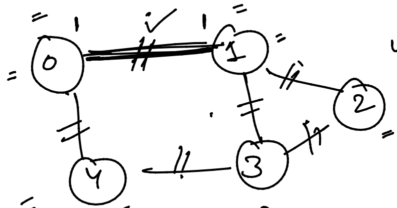
Adjacency List ⇒ array of lists st. at each idx of array, we've a list of neighbors of

(0, 1) (3, 4)



Undirected graph

(0, 1) (3, 4)
(0, 4)
(1, 2)
(1, 3)
(2, 3)

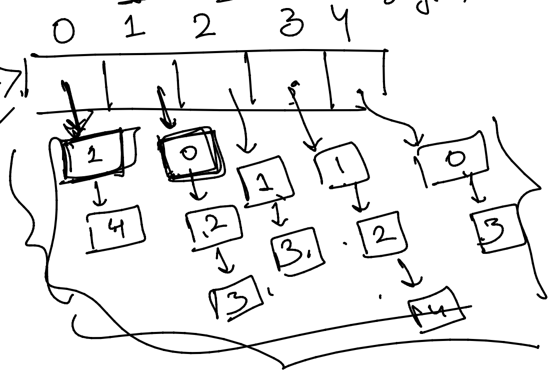


we've a list of neighbours of vertex u.

to matrix

deg(0) = deg(1) deg(2) deg(3) deg(4)

<forward-list> SLL or DLL <list>



$2|E| = 12$

$$\underbrace{\sum_{i=0}^{|V|-1} deg(i) = 2|E|}_{\text{Handshaking lemma}}$$

$O(V + 2E)$

$O(V + E)$

$O(V + V^2)$ X

$\sim E \sim V^2$

int * arr = new int[V];

list<int>* arr = new list<int>[V];

array of list

time

1) u → list all neighbours ⇒ $O(deg(u))$
$\sim O(V)$

2) add an edge (u, v) ⇒ $O(1)$

3) remove an edge (u, v) ⇒
$O(deg(u) + deg(v))$

4) check if (u,v) exist? $O(V+V) \sim O(V)$ $= O(min(deg(u), deg(v)))$
$O(deg(u))$ or $O(deg(v))$
$\sim O(V)$

5) list all edges $\sim O(2E) \sim$
$O(E)$

Adjacency Matrix

$V \times V$ matrix of 0's and 1's

$A_{V \times V}$ $\quad A[i][j]$

for undirected graph

$A(i, j) = 1$ iff $\{i, j\}$

for directed graph

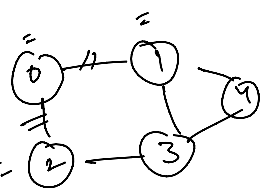
$A(i, j) =$ iff $i \to j$

Undirected

$V \Rightarrow O(V^2)$

sparse $\Rightarrow$ inefficient

dense $\Rightarrow$ efficient



$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 \\ 4 & 0 & 1 & 0 & 1 & 0 \end{array}$$

5×5

symmetric $> 50\%$

✓ check if edge $(i, j) = O(1)$

✓ add edge $= O(1)$

✓ remove edge $= O(1)$

✓ list neighbours $= O(V)$ $\quad A(i, j) = A(j, i)$

× list edges $= O(V^2)$ $\quad A(i, i) = 0$

---

1. Basic Introduction
2. Implementation of G $\to$ adj. list / $\hookrightarrow$ adj. mat

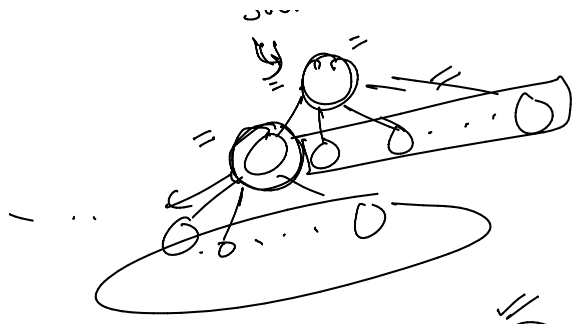
---

DFS and BFS $\Rightarrow$ Roman G, 8 $\quad$ Roman Empire (1000 years) $\quad$ How?

DFS

source

DFS



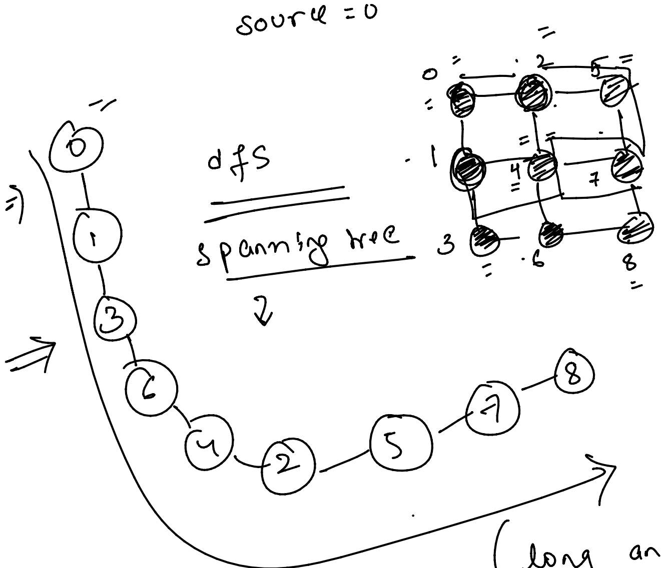
Graphs

source = 0

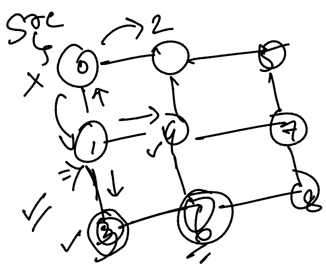
dfs

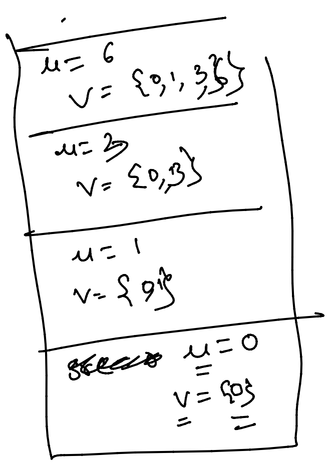
spanning tree

DFS → recursive
iterative



(long and skinny)

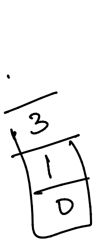


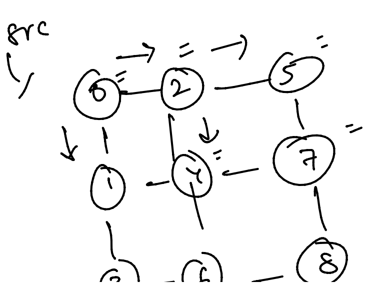


visited = {0, 1, 3, 6}

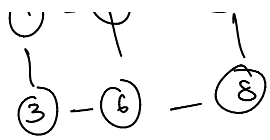
dfsHelp(0, {})

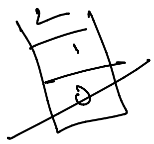
O/p: 0 1 3 6 . . .

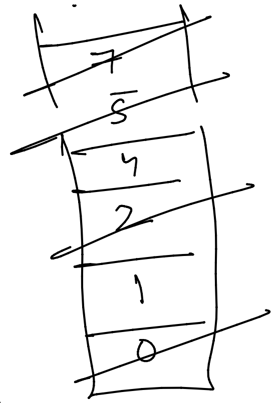




$\boxed{0}$







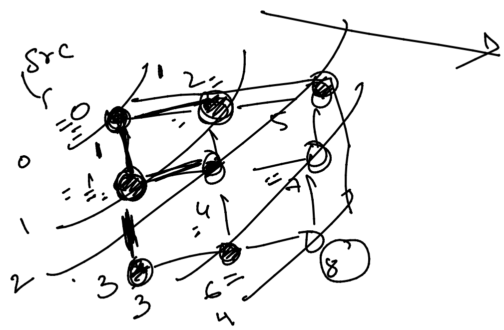
v = { 0, 1, 2, 4, 5, 7 }

o/p : 0 2 5 7   top = ~~0~~ ~~2~~ ~~5~~ 7

stk

BFS ⇒ shortest

level order



0   1 2   3 4 5   6 7 8

BFS spanning tree ⇒ shortest path from src to any other vertex