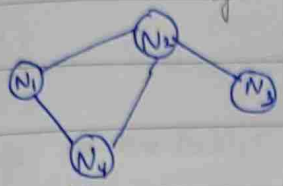
GRAPHS

Graphs class-1 (LIVE)

25sep2024

Data Structure → edges (made from interlinking of) → nodes / vertex



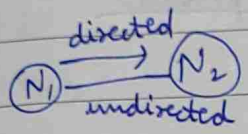
→ int
→ char
→ string ...

→ networking

uses : google maps, facebook mutual connection, shortest distance

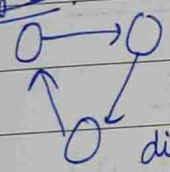
~~every graph is tree, but~~

→ every tree is graph, but every graph is not tree



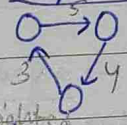
imp → clone a graph, clone a tree, check whether a graph is a tree or not.

Terminologies

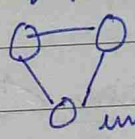


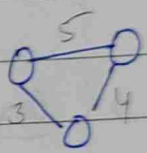
directed graph

weighted directed graph



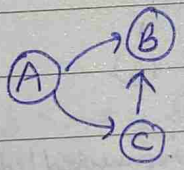
undirected graph





weighted undirected graph

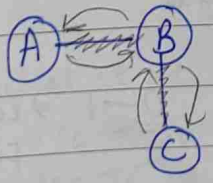
→ Degree
- indegree
- outdegree



A : indegree → 0
outdegree → 2

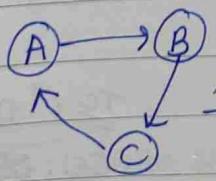
B → in : 1
out : 0

C → in : 1
out : 1



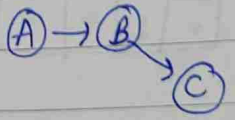
A : in → 1
B : out : 2

→ cycle



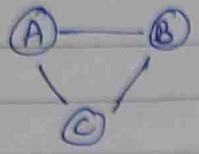
cyclic graph

(A → B → C → A)



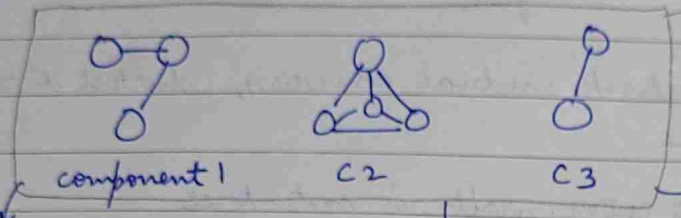
acyclic graph

- **Path**

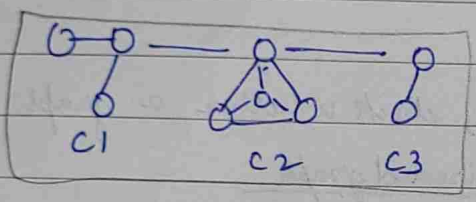


A-B-C : path → (ek node ek hi baar ayegi)
A-B-C-A : cycle, not path

- **Components**



→ need not be connected
→ forest (aka disconnected components)
graph g

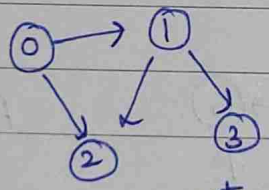


connected components

- **Graph Implementation** → lallu → adjacency matrix
  → mentos → adjacency list

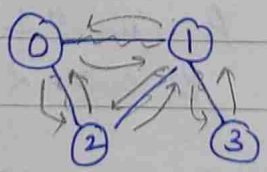
- **Adjacency matrix**
  ↳ 2D array

	0	1	2	3
0	0	1	1	0
1	0	0	1	1
2	0	0	0	0
3	0	0	0	0



0 → not connected/edge
1 → connected edges

	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	0
3	0	1	0	0



0—1 ⇒ [0][1], [1][0] } 1
1-2 ⇒ [1][2], [2][1] } 1

TC : $O(V^2)$
SC : ~~adjacency~~ ⇒ $O(V^2)$

no. of nodes : V

→ Padosio Ki list

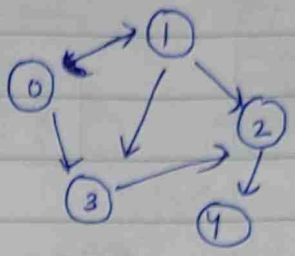
1) Adjacency list

list / vector

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

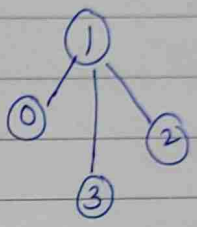
order doesn't matter

Key → int
value → list / vector



unordered map
Key: int
value: list / vector

TC : O(V + E) (E → no. of edges)
SC : O(V + E)

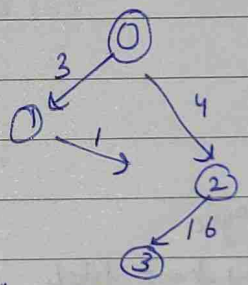


eg: 0 → 1
1 → 0, 3, 2
2 → 1
3 → 1

directed
1 → 2
1 → {2}

undirected
1 — 2
1 → {2}
2 → {1}

eg: 0 → {(1, 3), (2, 4)}
1 → {(2, 1)}
2 → {(3, 16)}
3 → { }

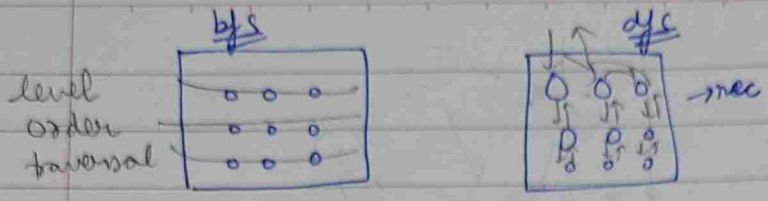


unordered_map < int , list < pair < int, int > > > adjlist
(int → u; pair → v, weight)

HW → ① class repeat at 2x
② code → TC → O(E)
↳ SC → O(E)

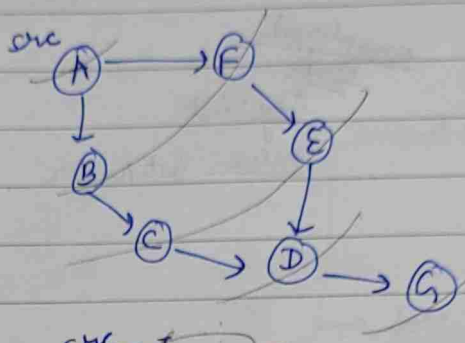
Traversal

↳ bfs (breadth first search)
↳ dfs (depth " ")



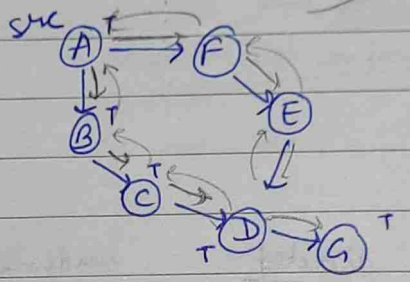
BFS.

A
BF
CE
D
G



dfs

A
B
C
D
G
F
E



mark visited.

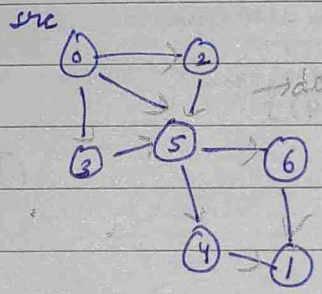
2) queue (bfs).

↓ initial state

q ← {0}

↳ ~~xxx~~ main logic

↳ front fetch → 0



→ doesn't matter if directed or undirected (working on adj list)

q ← ~~3~~ 5, 2, 5 same node → can't come twice

visited.

SC : O(V)

TC : O(V+E)

adj list

0 → {3, 5, 2}
1 → { }
2 → {5}
3 → {5, 5}
4 → {1}
5 → {4, 6}
6 → {1}

visited array

0 → ~~F~~ T
1 → ~~F~~ T
2 → ~~F~~ T
3 → ~~F~~ T
4 → ~~F~~ T
5 → ~~F~~ T
6 → ~~F~~ T

q → | ~~0~~ | ~~3~~ | ~~5~~ | ~~2~~ | ~~4~~ | ~~6~~ | ~~1~~ |

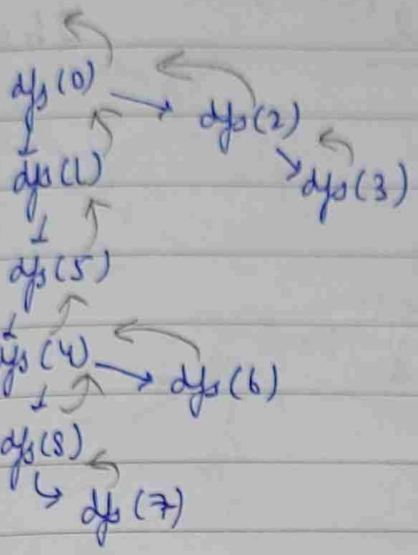
0 3 5 2 4 6 1

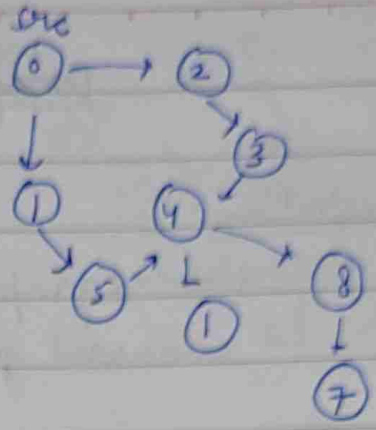
① initial state : q ← src

↳ vis[src] = T

② main logic : front neighbour

(dfs)





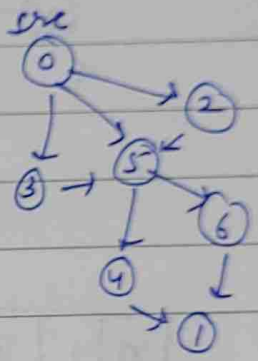
0 1 5 4 8 7 6
2 3

adjlist:

0 → 3, 5, 2
1 → {}
2 → {5}
3 → {5}
4 → {1}
5 → {4, 6}
6 → {1}

visited

0 → ~~F~~ R	T
1 → ~~F~~ F	T
2 → ~~F~~ R	T
3 → ~~F~~ F	T
4 → ~~F~~ R	T
5 → ~~F~~ F	T
6 → ~~F~~ F	T



0 3 5 4 1 6 2

dfs(0) → dfs(3) → dfs(5) → dfs(4) → dfs(1)
dfs(2)
dfs(6)

(HW)

SC: O(V)

TC: O(V+E)

DOMS | Page No.
Date / /

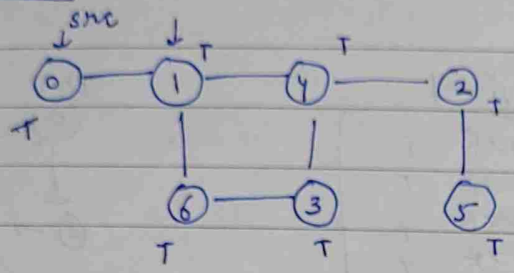
Graphs class-2 (LIVE)

26 sep 2024

* Cycle detection
  - directed graph → BFS
  - directed graph → DFS
  - undirected graph → BFS
  - undirected graph → DFS

(1) undirected

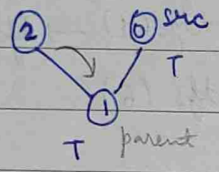
* BFS



nbr → already visited → cycle present ✓ → visited node should not be parent

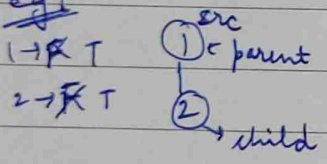
q: 0 | 1 | 4 | 6 | 3 | 2

Spcl case :



⇒ parent par wapas aana is not considered as cycle

eg!



1 → {2}
2 → {1}

q: 1 | 2

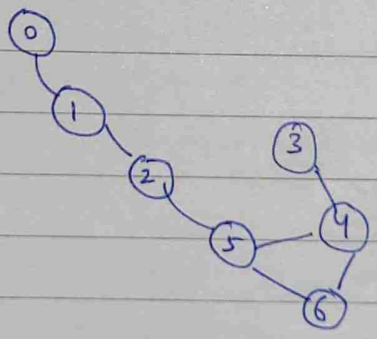
eg:

adj list

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

visited

0 → T
1 → T
2 → T
3 → T
4 → T
5 → T
6 → F (T)

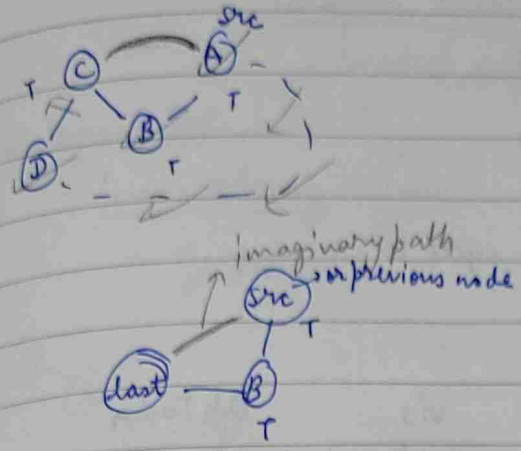


parent

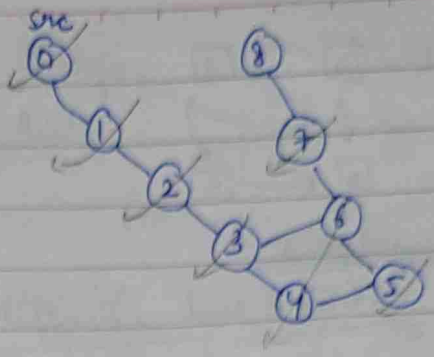
0 → -1
1 → 0
2 → 1
3 → 4
4 → 5
5 → 2
6 → 5

visited and not parent → cycle

q: 0 | X | 2 | 5 | 4 | 6 | 3



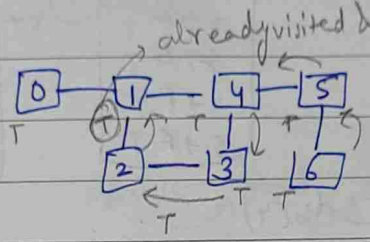
bfs



TC : O(V+E)

SC : O(V)

a) DFS



HW. famous (interview)

Q. find the no. of islands.

↳ using dfs.

```
for() {
    if(!vis) {
        rec call
        cnt++;
    }
}
```

HW. famous

Q. rotten oranges / tomatoes

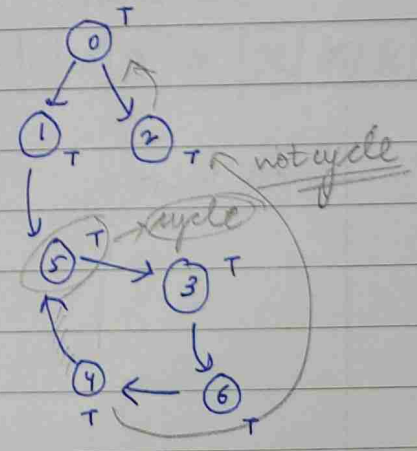
↳ apply both bfs & dfs

② Directed graph

a) DFS

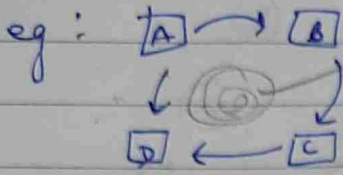
cycle : path → 1 node repeated.

5-3-6-4-5



path: 0 1 5 3 6 4 0

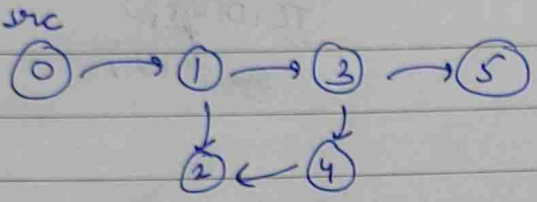
0
1
5
3
6
4

eg: 

→ cycle ✗
A – B – C – D

directed cond"

i) vis [True] and currentpath ⇒ cycle ✓

eg: 

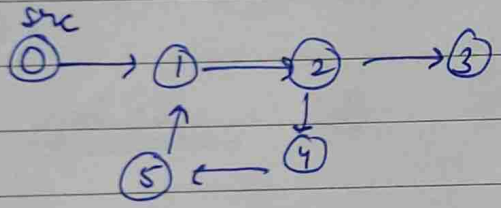
vis

0 →	F	T
1 →	F	T
2 →	F	T
3 →	F	T
4 →	F	T
5 →	F	T

dfsTracker

0 →	F	F
1 →	F	F
2 →	F	F
3 →	F	F
4 →	F	F
5 →	F	F

dfs(0)
dfs(1)
dfs(2)
dfs(3)
dfs(4)
backtrack
dfs(5)

eg: 

vis

0 →	F	T
1 →	F	T
2 →	F	T
3 →	F	T
4 →	F	T
5 →	F	T

dfsTracker

0 →	F	T
1 →	F	T
2 →	F	T
3 →	F	F
4 →	F	T
5 →	F	T

True
cycle

dfs(0)
dfs(1)
dfs(2)
dfs(4)
dfs(3)
dfs(5)

TC : O(V+E)

SC : O(V)

DOMS | Page No.
Date / /

Graphs class-3 (LIVE)

30sep2024

directed acyclic graphs

Topological Sort {DAG}

(dfs) ↳ linear ordering + rule

eg: independent (indegree=0) most dependent

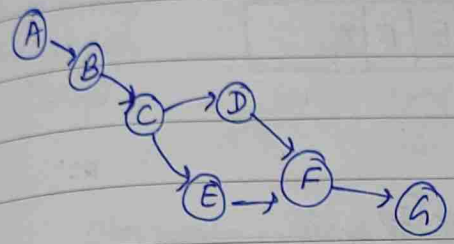
notes: (namrata) → (love) → (gateway) → (kunal)

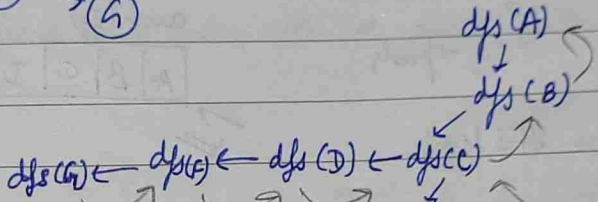
x, y, z, u, v

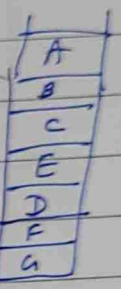
edge list

x→y
y→z
y→u
u→v

agar edge list me x, y se pehle hai toh ordering me bi pehle aana chiye







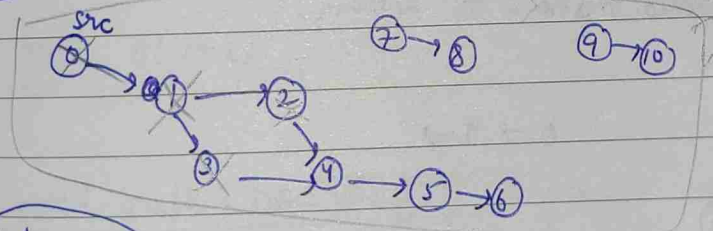
backtrack krte time stack me insert kardena

A→B
B→C
C→D
D→F
E→F
F→G

A→B→C→E→D→F→G

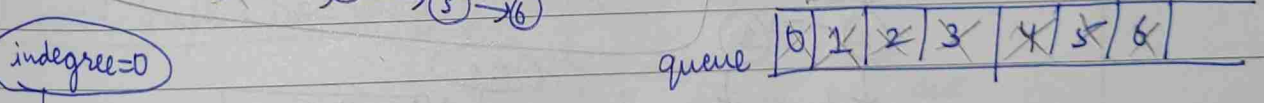
TC: O(V+E)

BFS (topological sort)



eg: multiple indeg node can be there

indegree=0 → independent

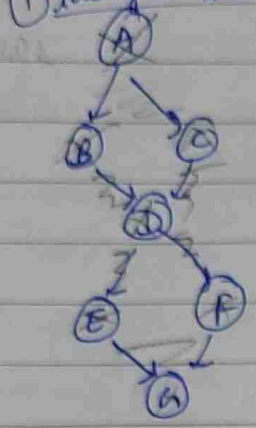


0→1→2→3→4→5→6

indegree

0→ ~~0~~
1→ ~~1~~ 0
2→ ~~1~~ 0
3→ ~~1~~ 0
4→ ~~2~~ ~~1~~ 0
5→ ~~1~~ 0
6→ ~~1~~ 0

① indegree calculate



indegree

A → 0
B → 1 → 0
C → 1 → 0
D → 2 → 1 → 0
E → 1 → 0
F → 1 → 0
G → 2 → 1 → 0

q

A	B	C	D	E	F	G

② travel indegree

⇓

==0

→ q → push

③ main BFS logic

ans (I)

A	B	C	D	E	F	G

TC : linear time

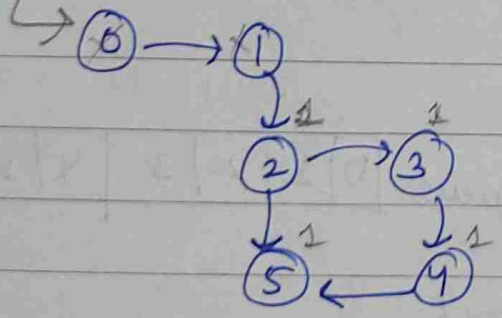
disconnected component? → handled while maintaining initial component

Q. cycle detection

↳ directed graph → BFS ✓

DAG → valid topological sort (TS)

cyclic graph → valid T.S ✗



0 → 1 →

wrong ans ✗

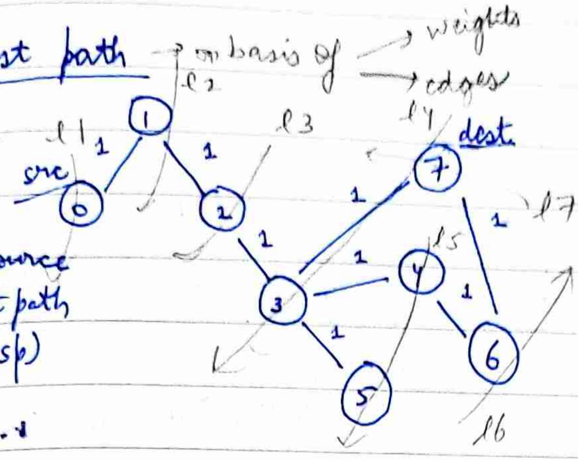
T.S → length != V

↳ cyclic graph

Graphs Class-4 (LIVE) — 1 oct 2024

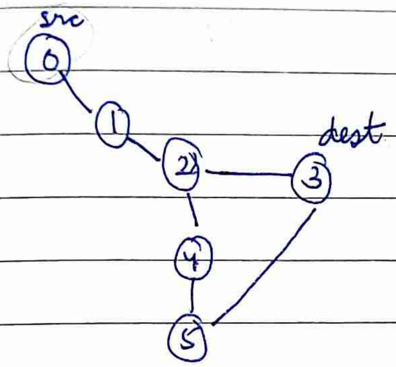
Shortest path → on basis of → weights, → edges

① BFS (undirected) → single source shortest path (sssp)



Observation: first time node visited ⇒ shortest path ✓

eg:



initial state
queue: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~

Parent
0 → -1
1 → 0
2 → 1
3 → 2
4 → 2
5 → 3

visited
0 → ~~F~~ T
1 → ~~F~~ T
2 → ~~F~~ T
3 → ~~F~~ T
4 → ~~F~~ T
5 → ~~F~~ T

↳ helpful in finding path b/w src & dest.

0 1 2 3 4 5

3 → 2 → 1 → 0

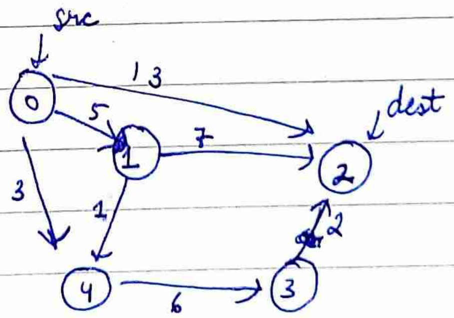
reverse ⇒ 0 → 1 → 2 → 3

TC = O(V+E)

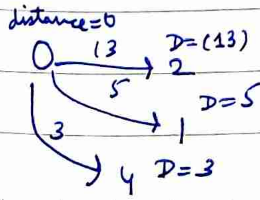
② **Directed graph : DFS** → sssp

min distance

↳ next distance update if new min dist is found



min dist
~~16~~ 2
~~18~~ 11 ✓



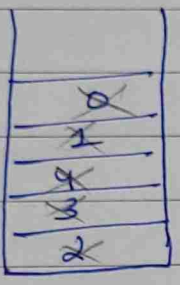
Approach

↳ distance update : min

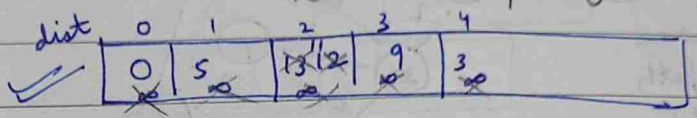
Topological sort ✓

⇒ 0 ← 1 ← 4 ← 3 ← 2

↳ distance baar baar update nhi karna padega.

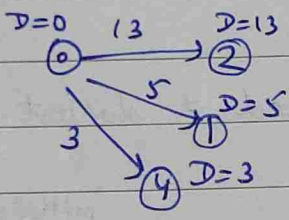


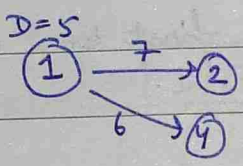
stack

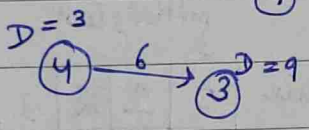


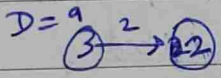
dist[src] = 0

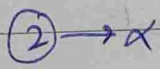
↳ initial state maintain











TC : O(V+E)

Graphs Class-5 (LIVE)

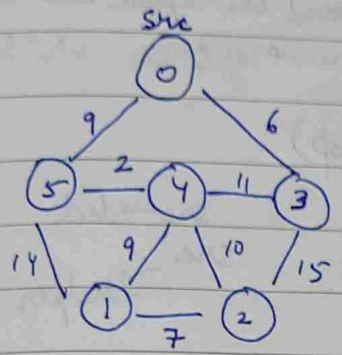
30 Oct 2024

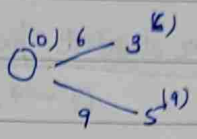
(Virup)

Dijkstra Algo → can be applied to directed & undirected graph

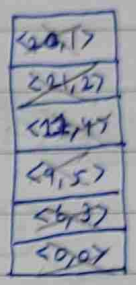
: single ~~start~~ source shortest path (sssp)

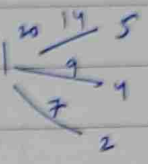
↳ based on greedy approach → e.g. set ... or pq





set





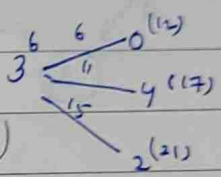
pair <int, int>

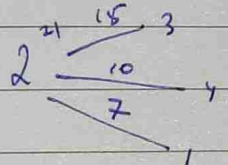
dist → node

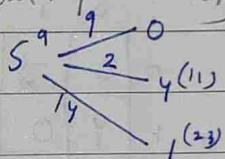
	0	1	2	3	4	5
dist	0	~~20~~ 20	21	6	~~11~~ 11	9

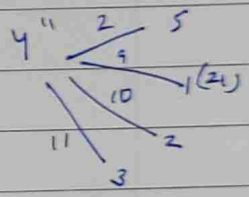
initial state → dis[src] = 0

TC: E log V → where we are updating (in our implementation)

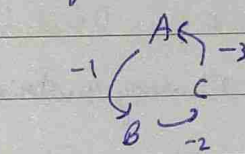








→ this algo doesn't work for -ve weights & -ve cycle.



∞ loop me jaa jaaye

TC: O(V²)

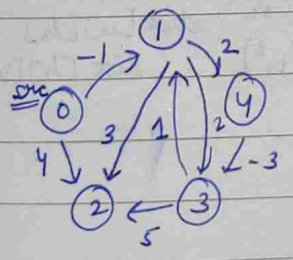
standard dijkstra

revaluation not allowed → cannot update again & again

0) Bellman ford Algo (sssp) → -ve weight handle and check -ve cycle exists or not.

relaxation step → (N-1) times → why?

Result → shortest distance



(dist) wt → V
u

relaxation step

dist[u] + wt < dist[v] → update if true

edgelist

Not V=5

0 -1→ 1
1 -2→ 4
0 -4→ 2
3 -5→ 2
4 -3→ 3
1 -3→ 2
1 -2→ 3

Relaxation #1

TC: N X E

dist	0	-1	2	-2	1
	0	1	2	3	4

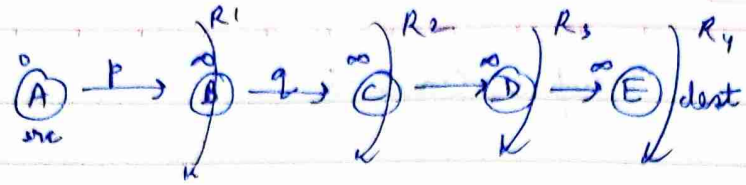
Relaxation #2

0	-1	2	-2	1
		2	3	4

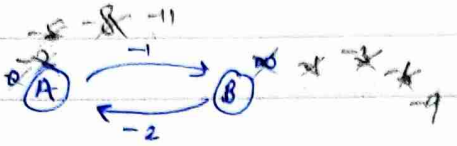
→ no change (break)

#3 → no change
#4 → two change

eg: N=5

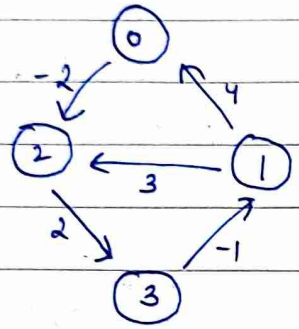


eg:

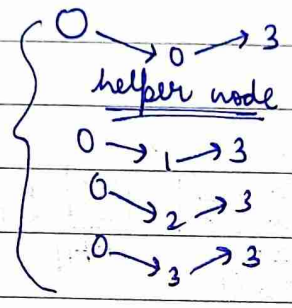


-ve cycle: relaxation (N-1)-times ke baad bhi update hota hai.
(non-ve cycle me nhi hota)

Floyd Warshal Algo → multiple source shortest path (mssp)



0 → 3



src → dest
src → helper → dest

	0	1	2	3
0	0	-1	-2	0
1	4	0	2	4
2	5	1	0	2
3	3	-1	1	0

① (i, i) → 0 (1 se 1, 2 se 2 ... 0 dist me pahuch jaoge)
② grab → dist → copy in 2D array/matrix
③ rest mark as ∞

```
for (h → 0 → n) {
    for (src → 0 → n) {
        for (dest → 0 → n)
            → logic
```

dis[u][v] =
min → dist[u][v]
or
dist[u][h] + dist[h][v]

TC: $O(n^3)$