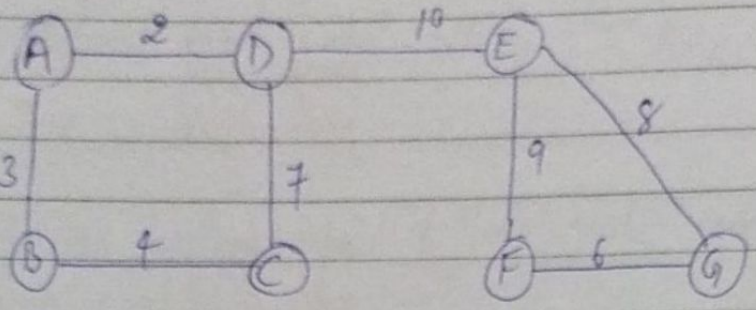
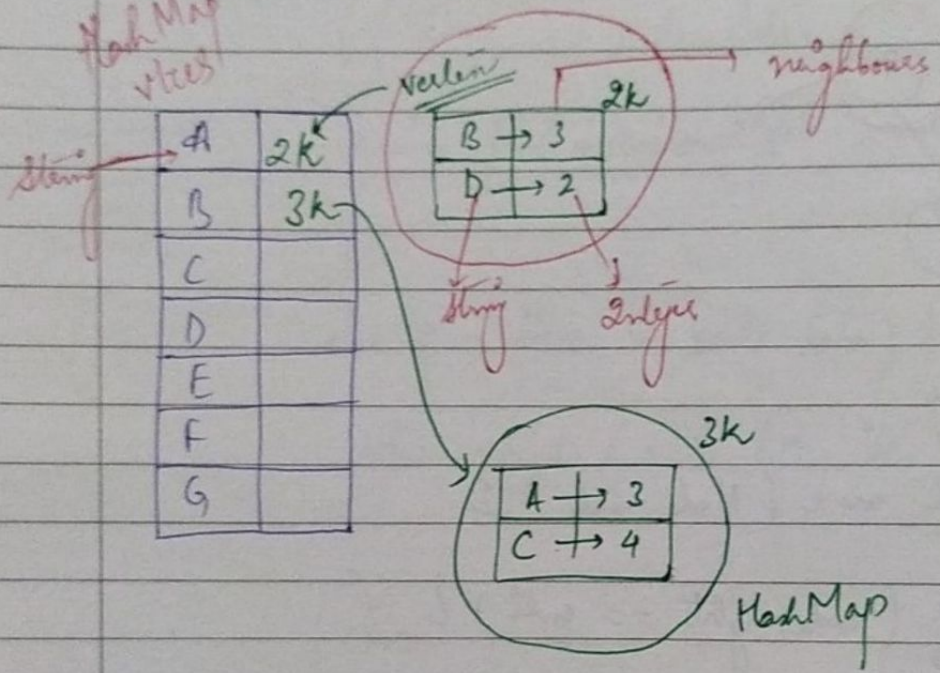
Imp **

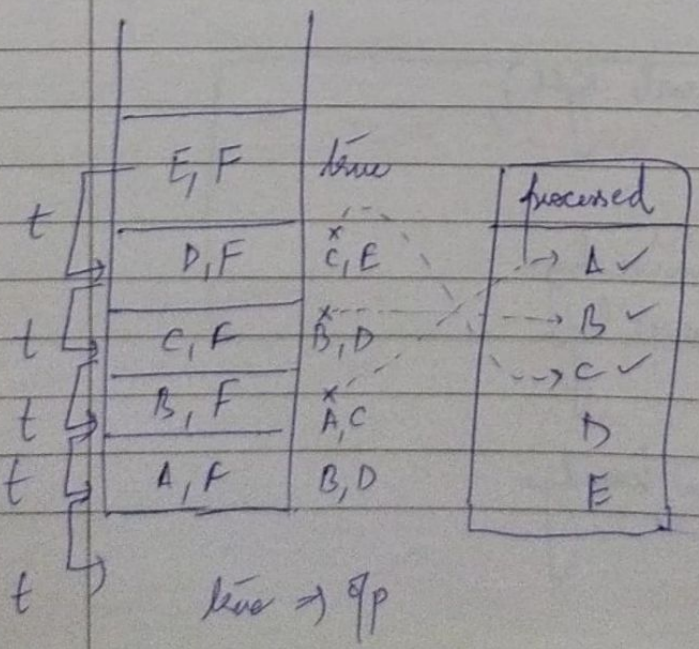
Graphs





HasPath

A - F



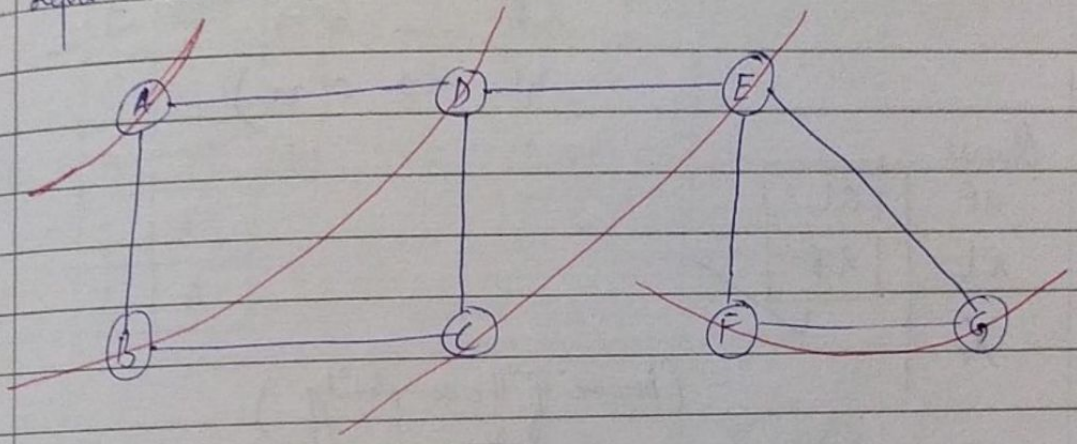
Shreya

→ Hamesha Shortest path milega

PAGE NO. icon
DATE : / /

Breadth First Search

algo for searching in graph data structure. It starts from src node and explores all neighbours nodes at the present depth prior to moving on to the nodes at next depth level.



Similar to Level Order Traversal in BT

Queue	Vertex	A	B	D	C	C	E	
	Path	A	AB	AD	ADC	ADC	ADE	ADEF

Neighbours → B, D ; A, C ; A, G ; B, D IGNORE

remove krte hi
add to processed HashMap

nil gya !! (E-F)

1. P ✓
2. direct edge
3. nbss

processed ⟹ Now check in processed hash map ki kya usse vertex se destination tak direct edge hai? agar hai tb return true

A
B
D
C
E (direct edge nil gyi)

Queue → adapt as LL.

Ek Pair class banayo.

Linked List → aF → O(1)
aL → O(1)
rF → O(1)
rL → O(n)

Queue

aF	aL
rL	rF ✓
X	

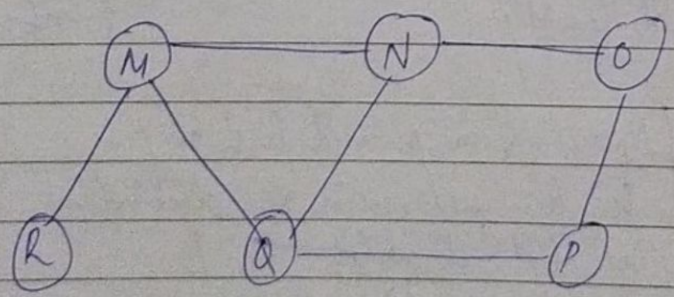
(Because of the complexity)

BFS → pehle woh ~~nodes~~ vertex visit krte hain jo source se 1 unit dist. pe hai, then 2 unit dist. & so on.....
(Just like Level Order in Tree

Stack

aF	aL
rF ✓	rL
	X

Q.



M ~
NQR
QNR
RNQ

Q- BFS is implemented using queue DS. One possible order of visiting nodes of the foll. graph

A MNOPQR X
B NQMPOR X
C QMNPRO ✓
D QMNPOR

M | NQR |
N | QMO |
Q | MNP |

correct → X

M ke nbrs ki processing ∴ RO should be the order

(C) is correct

Depth First Search

DFS → A's neighbour is B, D
Toh B pick kiya & pura traversal B ke saath kiya (going to B's neighbours).

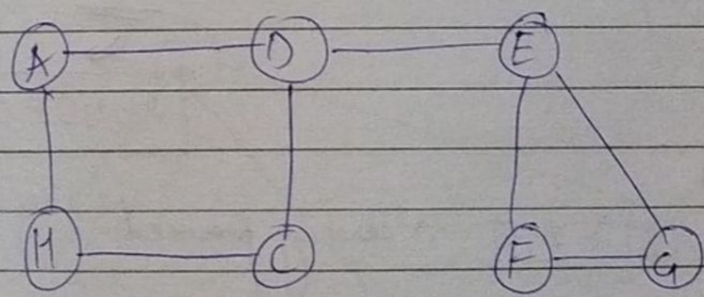
→ Use stack

~~E~~	~~ADE~~ (top)
C	ADC
~~D~~	~~AD~~ (top) ← A, C, E (A crossed)
B	AB
A	A

E–F direct edge ∴ return kara diya ∴ ADEF

processed
A ✓
D ✓
E ✓

→ Acha Now let's rename B vertex to H vertex.



E	AHCDE
~~D~~	~~AHCD~~ (top) ← A, C, E (A, C crossed)
~~C~~	~~AHC~~ (top) ← H, D (H crossed)
~~H~~	~~AH~~ (top)
D	AD
A	A ← D, H

→ E–F direct edge ∴ return AHCDEF

processed
A ✓
H ✓
C ✓
D ✓

keyset
↓
ASCII order

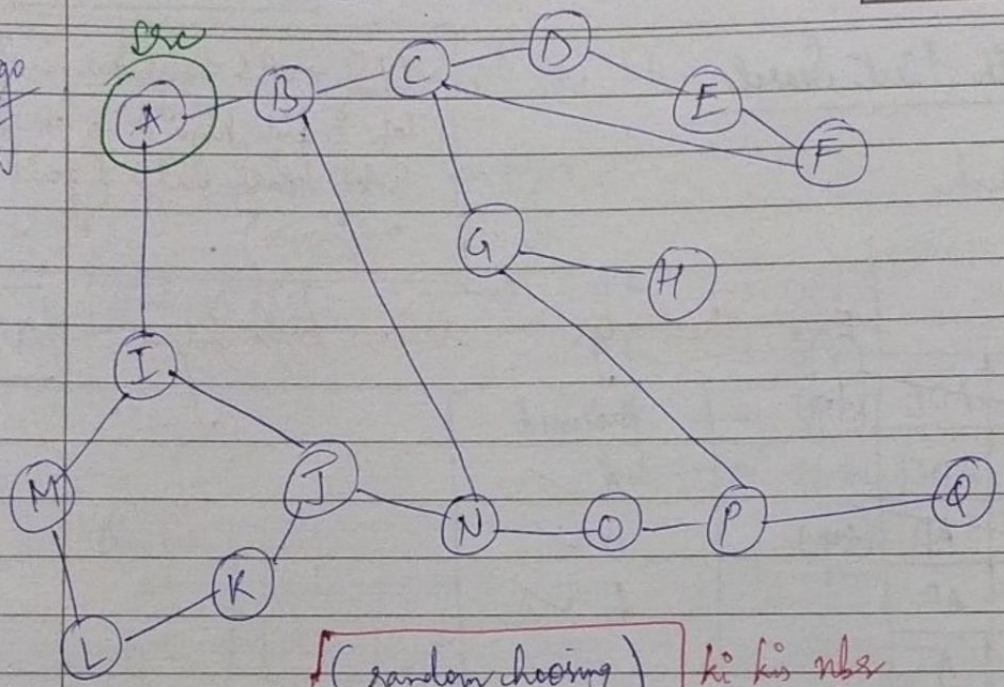
* Imp
acc to ASCII DFS is dependent

DFS

PAGE NO.
DATE : / /

Algo

src



dst.
F
first path

(random choosing) ki kis nbs ki fastest jaana hai

A : [B, I]
B : [C, N]
N : [J, O]

k
L
M
I → A, J → already processed.

∴ Path

A B N O P G C D E F

Remember!!

So, come back!!! to N

N → [J, O]

O
P
Q → can't explore further

DFS does not give you the shortest path

So, come back to P !!!

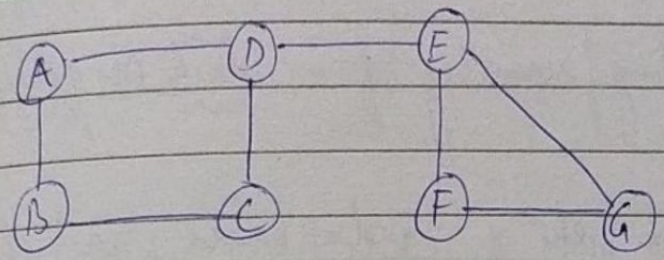
P
G
C
D
E
F

Shreya

PAGE NO. icon
DATE : / /

→ covers each & every node of the graph in BF manner

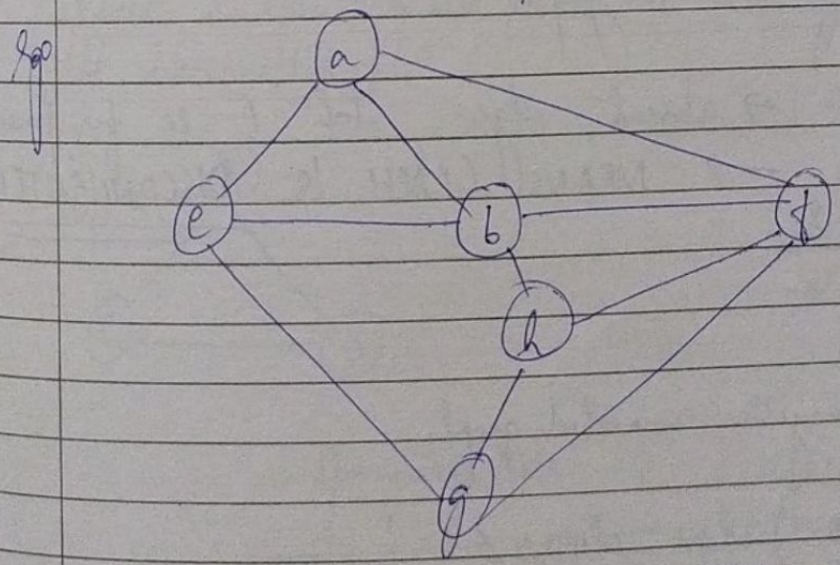
Breadth First Traversal



① Traverse through all vertices
② print vertex & the path followed to reach from that vertex to another.

(Queue Implementation)

DFT → Stack Implementation



Among the following sequences, which are DFT of above graph?

I ✓ a b e g h f
II ✗ a b f e h g
III ✓ a b f h g e
IV ✗ a f g h b e

PAGE NO.
DATE : / /

Is Cyclic ?

* If, while adding / removing from stack or queue
  ↓
  ek vertex ek se zyada baar na aaya hai
  toh graph me cycle hai

Is Connected ?

* Agar graph connected hoga → to hamare BFS me
  saare nodes ko A se hi traverse kar lenge,
  B, C, D, E, F, G me continue karenge
  ∴ Connected graph

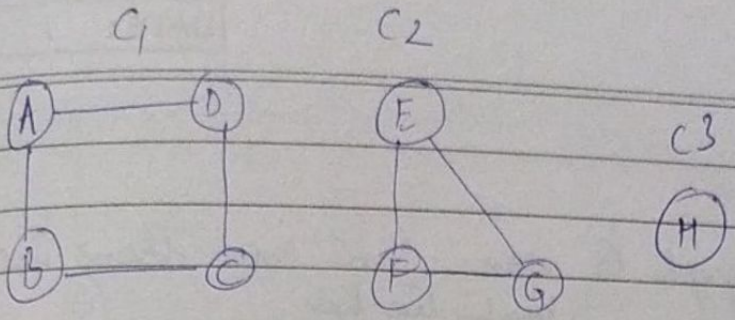
* Otherwise, DE → absent edge, toh 'E' se fir journey
  start karenge → MEANS GRAPH IS DISCONNECTED

isTree ?

Tree is an acyclic connected graph

Tree → (n-1) edges always

Get Connected Components



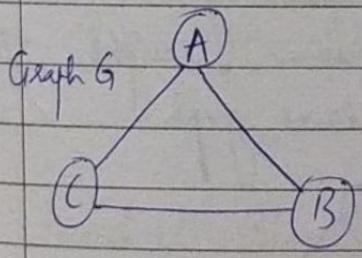
[[A, B, C, D], [E, F, G], [H]]

C1 C2 C3

Spanning Tree

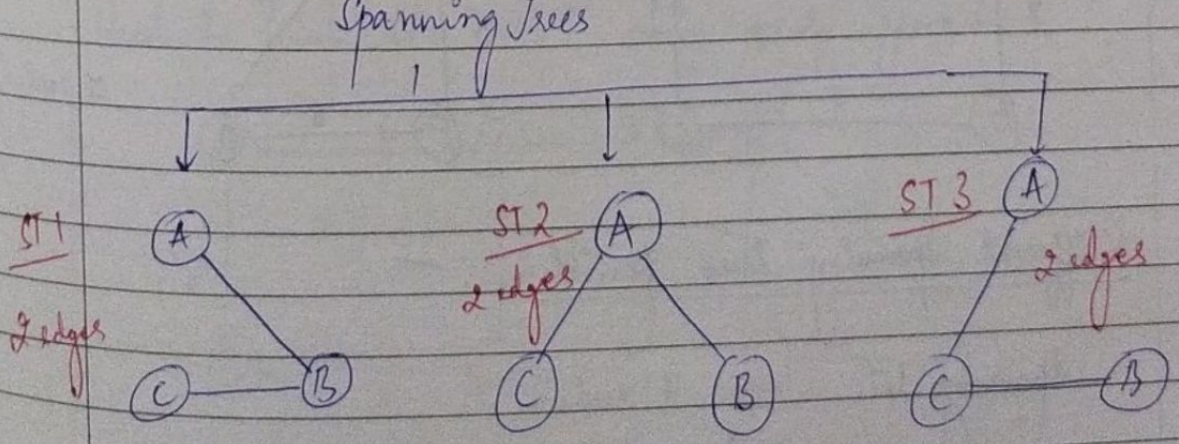
A spanning tree is a subset of graph G, which has all the vertices covered with minimum possible number of edges.

Hence, a spanning tree does not have cycles and it cannot be disconnected.



3 vertex
2 edges

Spanning Trees



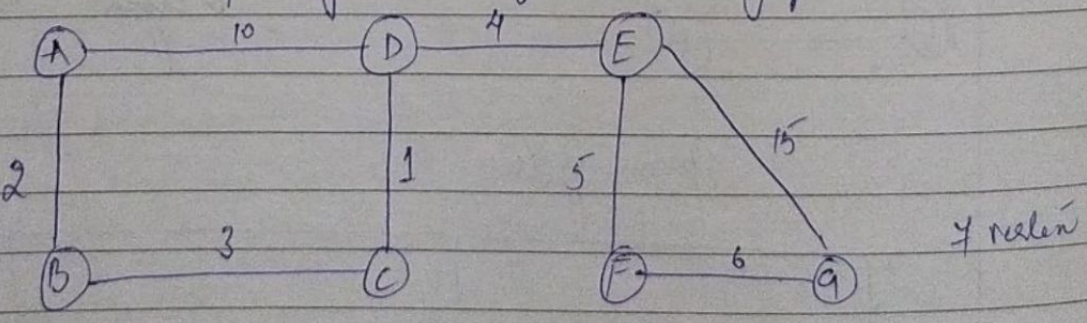
PAGE NO. icon
DATE : / /

Properties :

1. A connected graph G can have more than one spanning tree
2. Spanning tree has n−1 edges → n is no. of vertices (kyunki tree hai)
3. All possible spanning trees of graph G, have the same number of edges & vertices
4. The spanning tree does not have any cycle (loops)
5. Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is minimally connected.
6. Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is maximally cyclic

Minimum Spanning Tree (MST)

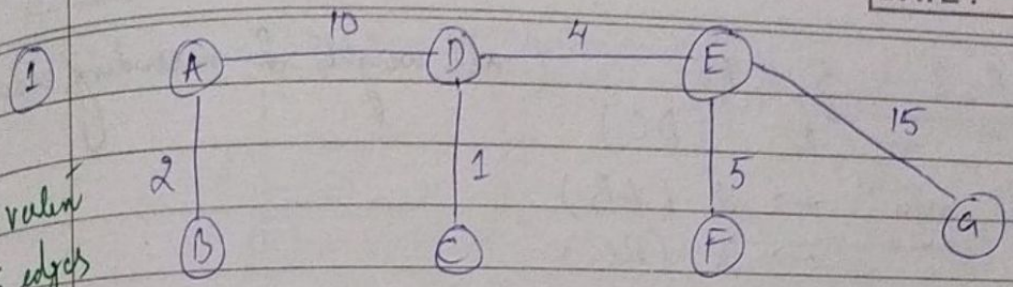
In a weighted graph a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.



Different spanning trees possible :—

Spanning tree → 7 vertex
→ minimum edges

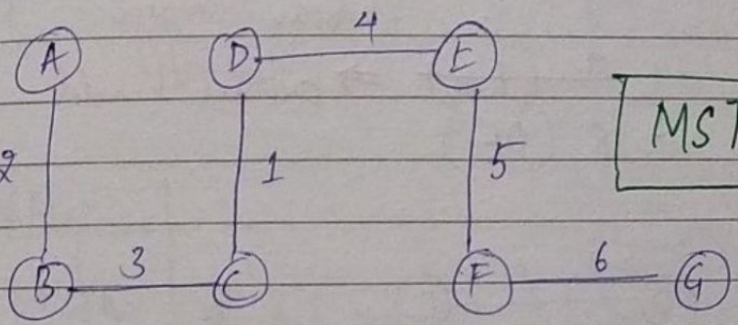
1



7 vertex
6 edges

Cost = 2 + 10 + 1 + 4 + 5 + 15 = 37

2

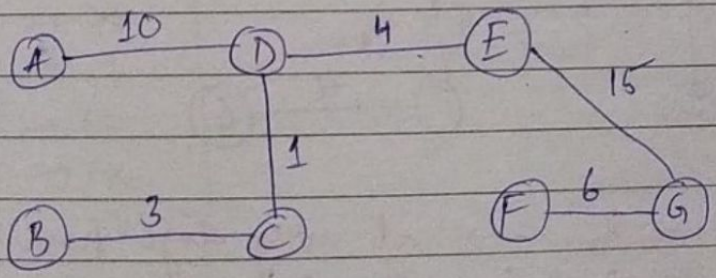


MST

7 vertex
6 edges

Cost = 21
MINIMUM

3

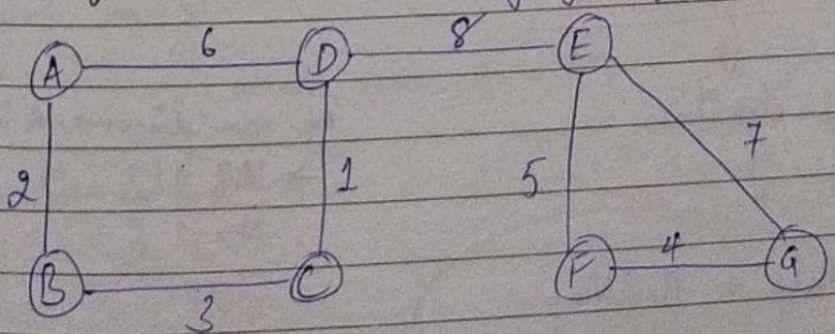


Cost = 39

7 vertex
6 edges

Kruskal Algorithm

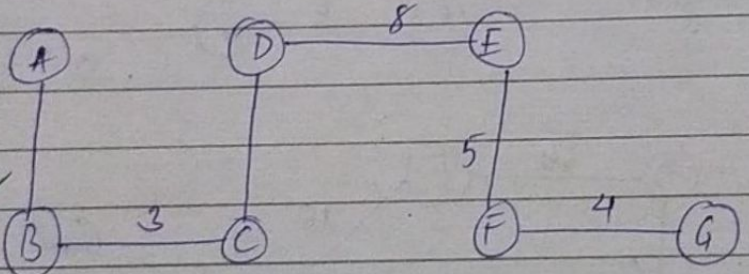
Used to find the MST using greedy approach



PAGE NO. icon
DATE : / /

1. 1, 2, 3, 4, 5, 6, 7, 8 (all weights in ascending order)
2. Min^m → 1 (DC)
   Next min^m → 2 (AB)
   " " → 3 (BC)
   → 4 (FG)
   → 5 (EF)
   → ~~6 (AD)~~ ⇒ cannot add because cycle ban jaayegi
   → ~~7 (EG)~~ ⇒ cannot add
   → 8 (DE)

Ye process tab tak karenge jab tak (n−1) number of edges fill nhi ho jaate



Greedy algorithm → at every stage hum aisi edge pick kar rhe jo minimum cost ki ho

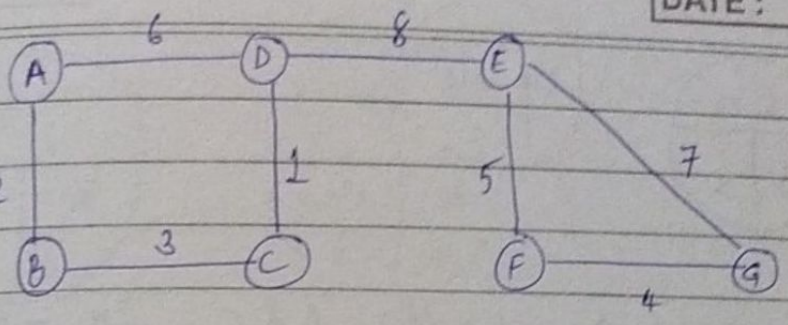
(optimising each step → final bhi optimised hoga)

Prims Algorithm

→ Greedy algorithm.

We can discover A by 2 paths -
→ AB : 2
AD : 6

Heap + HashMap

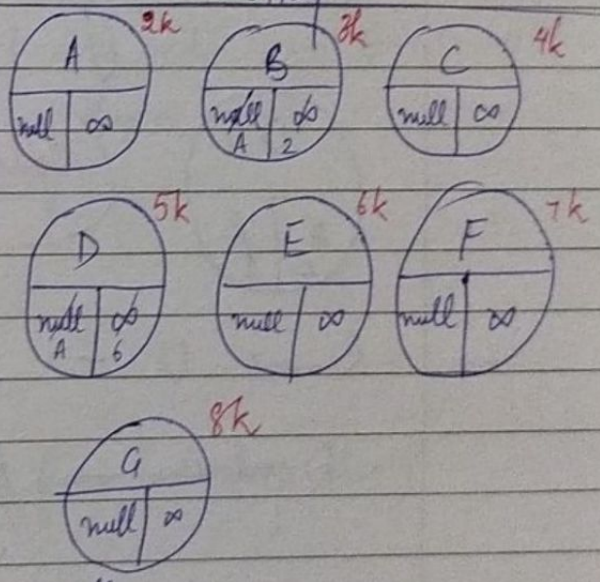


1. Make a pair → put in heap

Heap ⇒ 1. vname] string
2. avg name
3. cost] int

→ with pair fehle remove hoga jisko cost sabse kam hoga

Min Heap



HashMap

A → 2k
B → 3k
C → 4k
D → 5k
E → 6k
F → 7k
G → 8k

2. pair remove
→ Add in MST
→ See (A) ke nbrs

→ Remove first pair (A) → acq vname = null
→ • Now remove B : A–B edge → of cost 2
(kyunki B ke liye hum A ke through aaye the)
(remove from HashMap also)

• B nbrs ⇒ ~~A~~, C
(already removed)

We discovered (C) via (B)
↓
4k
map.get(C)

→ C | B | 3 — Update

→ • Remove C → edge b/w B & C with cost 3 (from hash map also)
• C nbrs → ~~B~~, D → ye pehle bhi discover kiya tha A → D with cost 6
↳ already removed

• Now we are again discovering (D) ⇒ CD cost 1

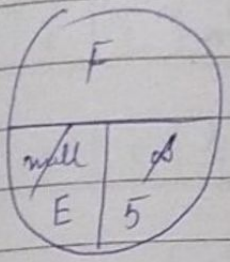
→ D | ~~A~~ C | ~~6~~ 1 — Update

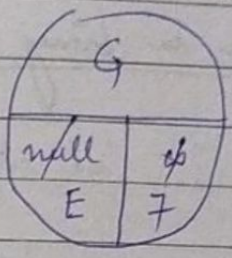
→ • Remove D → add in MST → CD with cost 1 (remove D from hash map also)
• D nbrs → ~~A~~, ~~C~~, E
↳ already removed

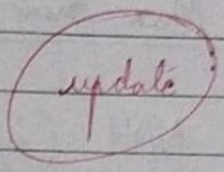
→ E | null | ∞ / D | 8 — update

→ • Remove E → add in ST → edge DE cost 8 (remove E from hash map also)
• E nbrs → ~~D~~, F, G
↳ already removed

→





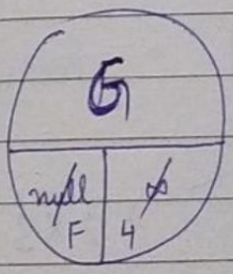


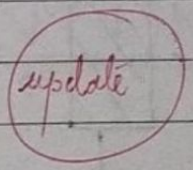
→ • Remove F (min^m cost) → add F in MST
→ edge E-F cost 5
(remove F from hash map also)

• F nbrs → ~~F~~, G
already removed

F - G cost ⇒ 4
previous cost ⇒ E - G ⇒ 7
Hence, update

→





→ Remove G (remove G from hash map also)

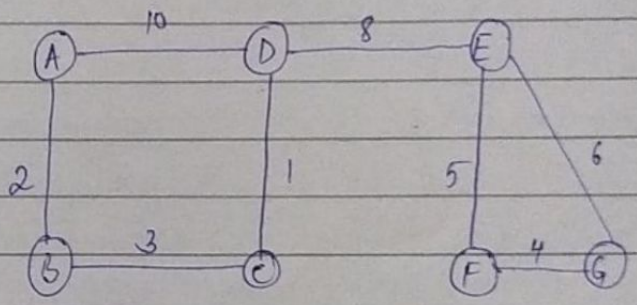
Graphs Continue from ② Notebook

DIJKSTRA ALGORITHM

→ Single Source shortest path Algorithm

Find out minimum distance from source vertex to every other vertex

src : A



(very similar to Prims algorithm)

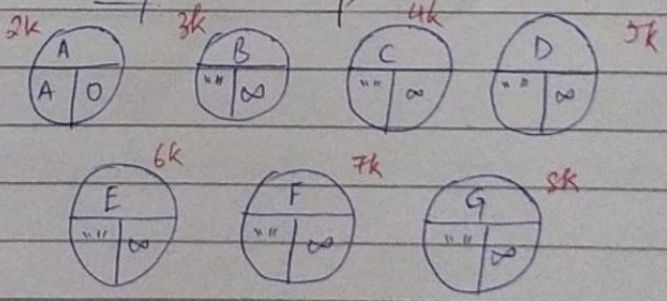
Dijkstra pair
- → vname
- → psf
- → cost

src se leke har vertex tak ka min^m cost

ans map

A → 2k
B → 3k
C → 4k
D → 5k
E → 6k
F → 7k
G → 8k

heap (min heap)



Step 1: remove the pair
Step 2: Add it to ans
Step 3: nbrs

① a) Remove A ($\because$ min^m cost = 0)

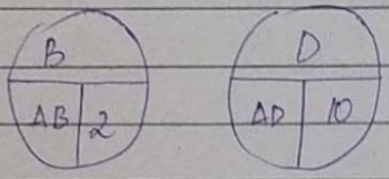
b) ans

A → 0

c) Nbrs of A : B, D

Update B : AA + AB = 0 + 2 = 2

Update D : 10



② a) Remove B ($\because$ min^m cost = 2)

b) ans

B → 2 (2 cost ke saath remove hua)

c) Nbrs of B : A, C

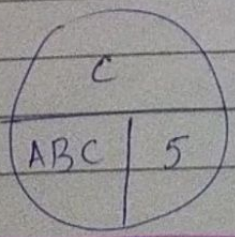
X

↳ already removed

Update C : 2 + 3 = 5 (source se leke C tak ki total cost is 5)

↑ B tak ki cost

edge cost



(3) a) Remove C (∵ min^n cost : 5)

b) ans

C → 5

c) Nbrs : B, D

X

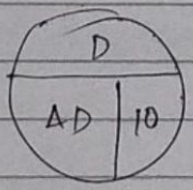
↳ already removed

Update D : 5 + 1 = 6

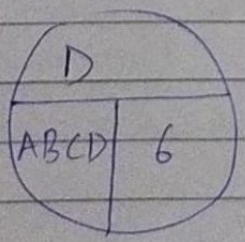
↓ source se C-tak ki cost

↓ edge cost (CD)

Now, we noticed ki D pe pehle se value rakhi hui hai



But 6 < 10, toh hame value update karni chahiye



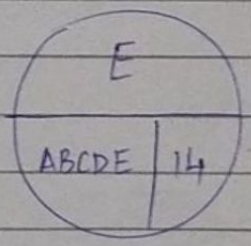
④ a) Remove D

b) ans

D → 6

c) Nbrs : A, C, E
x x
↳ already removed

Update E : D tak ki cost + DE edge cost
6 + 8 = 14



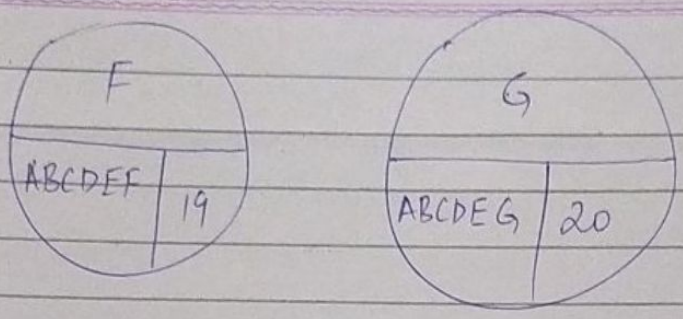
⑤ a) Remove E

b) ans

E → 14

c) Nbrs : D, F, G
X ↳ already removed

Update F : 14 + 5 = 19
Update G : 14 + 6 = 20



(6) a) Remove F (∵ min^m cost : 19)

b) ans

F → 19

c) Nbrs : E, G

X

↳ already removed

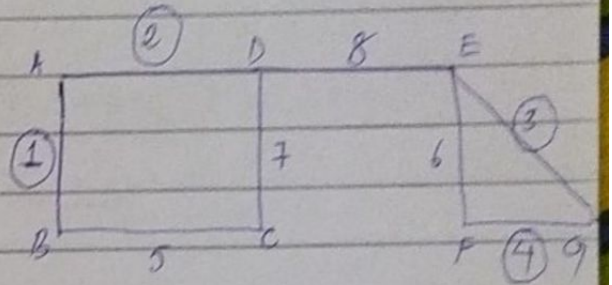
G : 19 + 4 = 23

(No need to update bcoz 20 < 23)

(7) a) Remove G

KRUSKAL

- Place vertex in different sets → create set

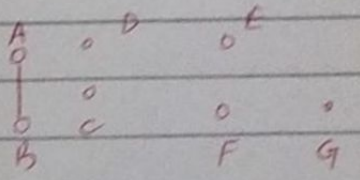


① {A} {B} {C} {D} {E} {F} {G}

union

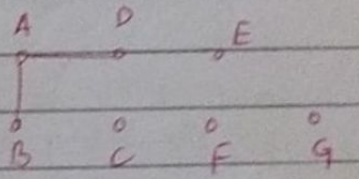
1 → min^m cost edge b/w vertex A, B

② {A,B} {C} {D} {E} {F} {G} → represent



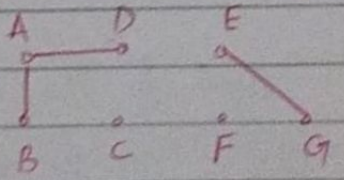
Next → 2 min^m cost edge b/w vertex A, D

③ {A,B,D} {C} {E} {F} {G} → represent



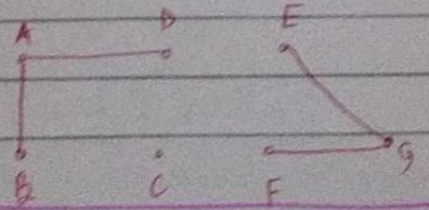
Next → 3 min^m cost edge b/w vertex E, G

④ {ABD} {C} {EG} {F} → represent



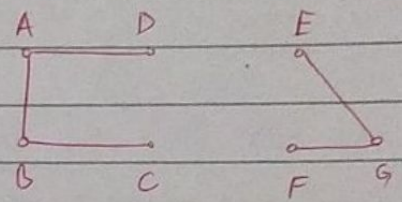
Next → 4 min^m cost edge b/w F, G

⑤ {ABD} {C} {EFG} → represent



Next → 5 min^m cost
edge b/w B, C

⑥ {ABDC} {EFG} → represent



find set → if same set, no need to union the sets

Next → 6 min^m cost
edge b/w E, F

same set in ⑥
if we connect E, F toh cycle aa jaayegi
and mst can't have cycles

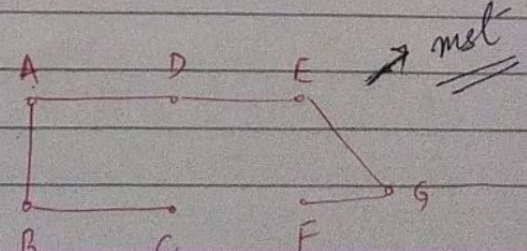
∴ Do nothing & move forward

⑦ Next → 7 min^m cost
edge b/w C, D

Again in the same set in ⑥
∴ do nothing & move forward

Next → 8 min^m cost
edge b/w D, E

⑧ {ABDCEFG} → represents



It represents all nodes are connected

Requirements

⇒ We've represented sets as Disjoint Sets

↓

jin 2 sets me koi element common nhi ho eg {AB} {CD}

⇒ create set : vertex given {A}

⇒ Union set : {ABCD} {EF} {G}

C, F

U

{ABCD EF}

⇒ Find set : S1 {ABCD} S2 {EF} S3 {G}

C : konse set me present hai ? : S1
F : S2

⇒ Ab, we can't name the sets S1, S2, S3, kyunki sunaine pe set naye bante ho sakte hain

So, to deal with this → we need a representative element in a set.

{(A)BCD} {(E)F} {(G)}

C : konse set me present hai ? : A
F : E

⇒ C, D same set ?

| A | A |

Yes

⇒ C, F same set ?

| A | E |

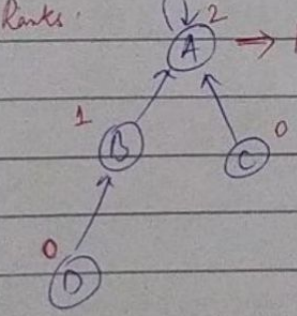
No

Disjoint Set Implementation

Tree — S1 {A, B, C, D} S2 {E, F, G}

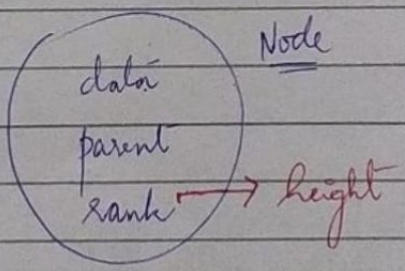
S1

Ranks

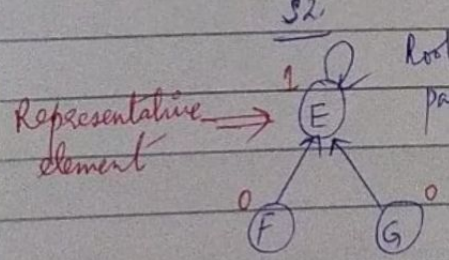


→ Representative element

node : parent



S2.

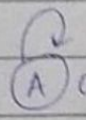


Root node ka parent khud root node hoga

Root Node

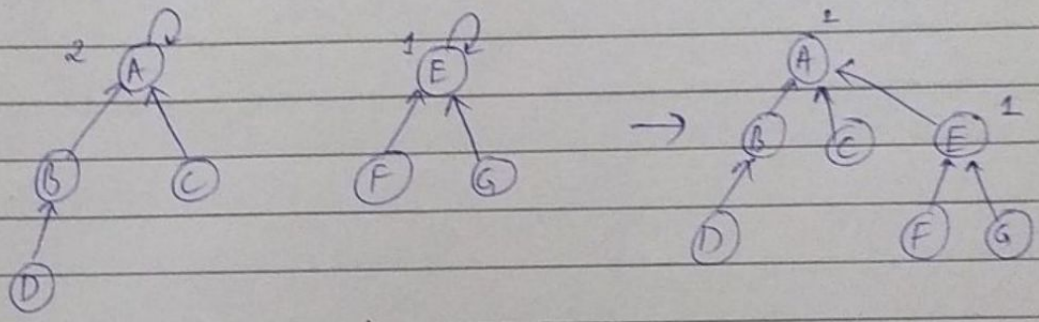
Representative element

⇒ create set — 'A' value is created



⇒ Union set

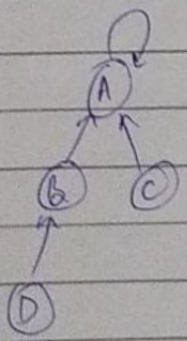
{A,B,C,D} ∪ {E,F,G} → {ABCDEFG}



Prefer this way, bcoz rank will remain unchanged

(chhoti rank wale ka parent badi rank wale ko banado)

⇒ Find set {A B C D}



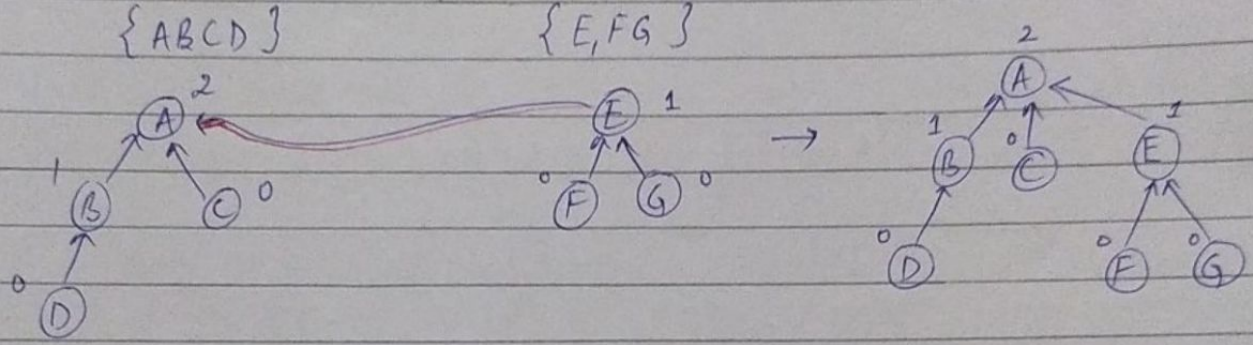
* Traverse in upward dirⁿ
* Root node find karo
* Find root node (coz that's the representative element)

⇒ Complexity :
1. create set : O(1)
2. union set : O(1) (just change the parent)
3. find set : O(h) ↳ height of tree

$O(\log_2 n)$

UNION

① Union By Rank (Method 1)

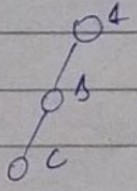


Height

ht 1 ⟹ 2 nodes

ht 2 need min^m 4 nodes

ht 3 ⟹ min^m 8 nodes



Not possible

ht	nodes (min^m)
1	2
2	4
3	8
⋮	⋮
$\log_2 n \rightarrow h$	$2^h \rightarrow n$

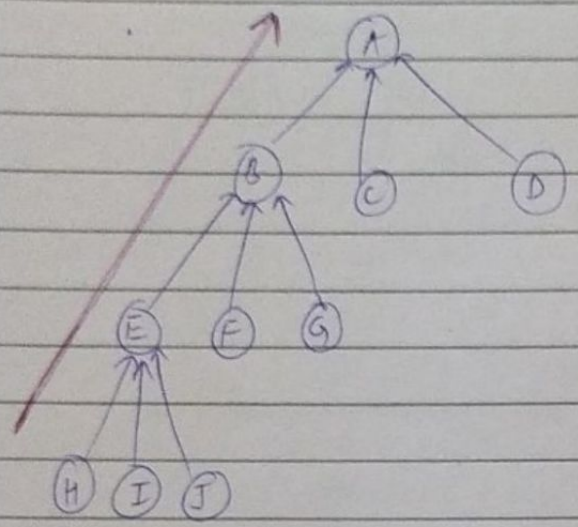
$O(h) \Rightarrow O(\log_2 n)$

We would also create a hash map to store addresses of nodes

map

A	1k
B	2k
C	3k
D	4k

(2) Union by Path Compression (Method 2)



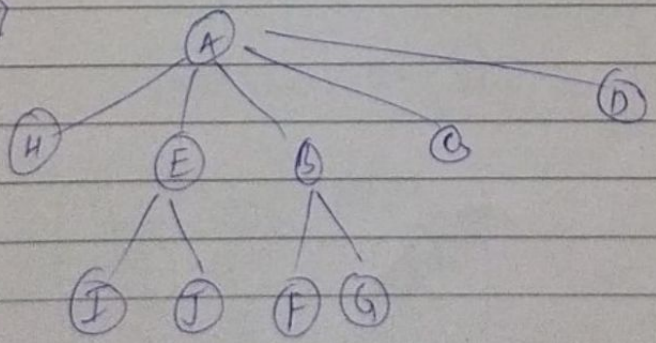
find(H) : A

find(E) : A

find(B) : A

find(H) : Time O(h) ← Right of tree

Lekin uss path ke baaki elements like 'E' 'B' unke liye bhi parent same hai, toh path compress karna chahiye



E, B : O(1) lagega ab.

node parent = recresult // path compression

Write this line in find(Node node) fx^n

Time Complexity (Graph Algos)

1. BFS : $2E+V$ $O(E+V)$
2. DFS : $O(E+V)$
3. Prims : $O(E \log V)$
4. Dijkstra : $O(E \log V)$
5. Kruskal : $O(E \log E)$

$E : V^2$ worst case

$= 2E \log V$

$= O(E \log V)$

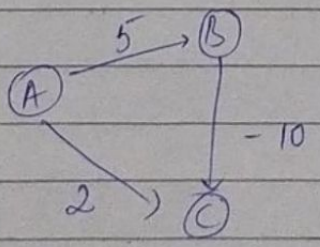
Bellman Ford algorithm

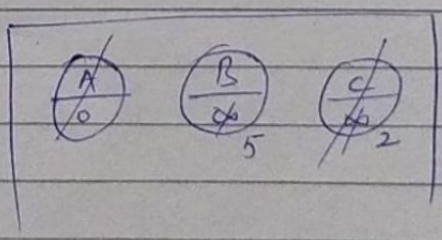
→ Single Source Shortest Path algo

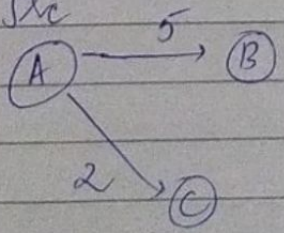
→ Drawback of Dijkstra algorithm :-

when weight of the edge is -ve, it won't work.

Eg.







But $A \to B \to C : -5$ ↳ shortest

∴ It does not work

→ To overcome this problem, Bellman Ford algo comes into play.

Bellman Ford

⊙ -ve wt edges present — O(E·V)

⊙ -ve wt cycle ✗

Floyd Warshall

→ Saari nodes se saari nodes tak ka shortest path nikaalo

→ $O(V^3)$

→ Agar hum ye kaam Bellman Ford se karte

$O(E \cdot V^2)$

↓ → saari vertex ke liye code sum karte

V^2

$O(V^4)$

Dijkstra $O(E \cdot V \cdot \log V)$