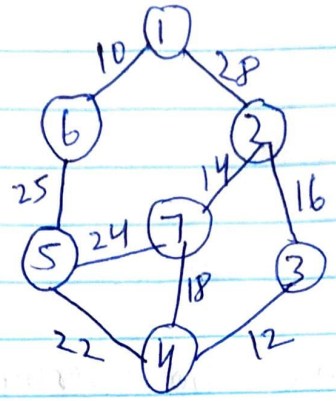
Exam 3

Mahesh Koppula
1001764522

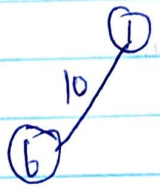
1. Using prim's algorithm, we will find the minimum spanning tree starting at 1.



Step1 : Starting from vertex 1

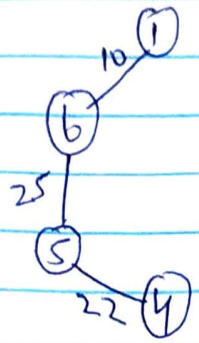
(1)

Step2: Minimum distance from (1) is to (6)

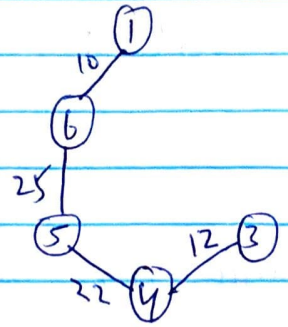


Step3: Then go for minimum distance from any visited vertex i,e (1) or (6). Here it is (5)

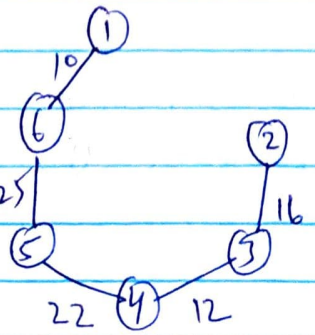
step4: Minimum distance from vertices ①, ⑥ & ⑤ is ④



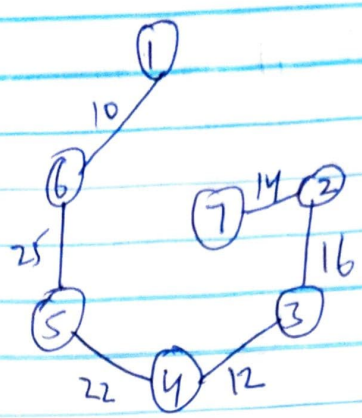
step5: Minimum distance from vertices ①, ⑥, ⑤ & ④ is ③



step6: Minimum distance from vertices ①, ⑥, ⑤, ④ & ③ is ②



step7: Minimum distance from vertices ①, ⑥, ⑤, ④, ③ & ② is ⑦

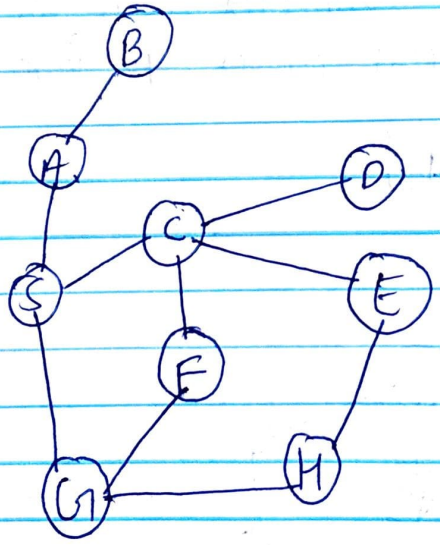


← minimum Spanning tree

All nodes are visited, so this will be the minimum spanning tree.

Cost = 10+25+22+12+16+14
= 99

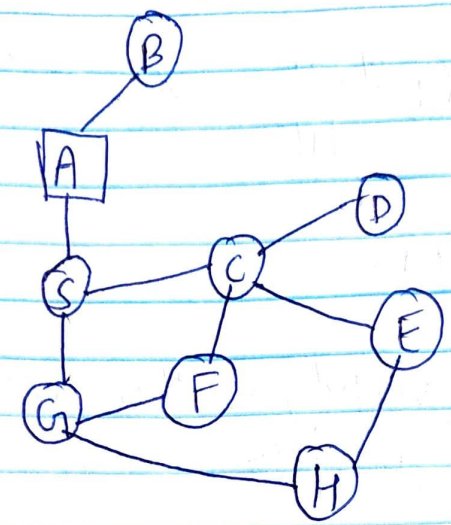
(2)



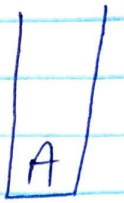
(a) Depth First Search, use alphabetical order to push onto stack.

☐ – visited.

Step1: Starting at A

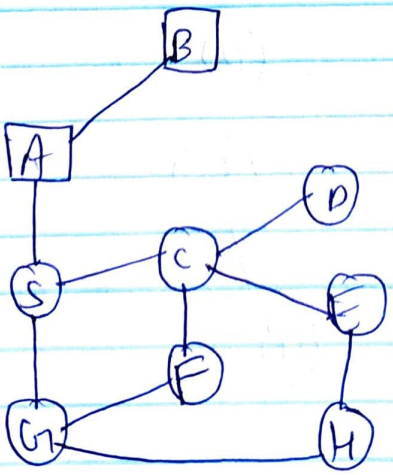


stack



Output: A

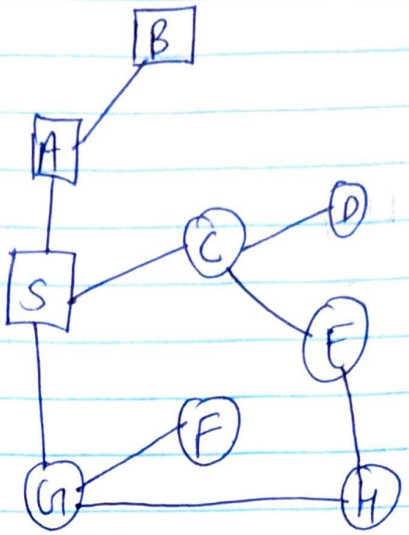
Step2: Nodes connected to A are B and S
Alphabetical order go for B



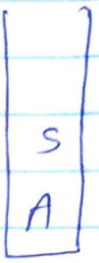
stack

output: A B

step 3: Remaining node connected to A is S



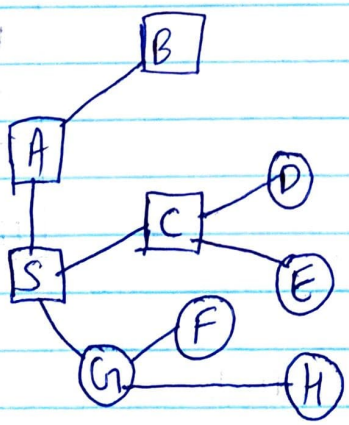
stack



B is connected only to A, which is already visited so pop up B from the stack

output: A B S

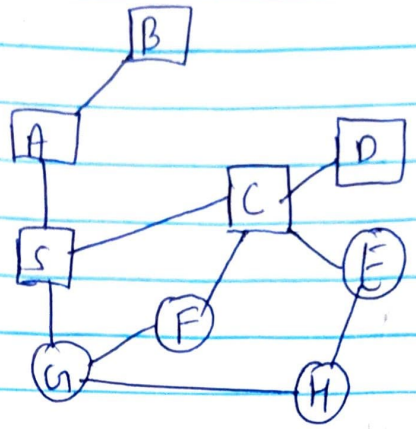
<u>step4</u>: Nodes connected to S are A, C, G. A is visited or go for C



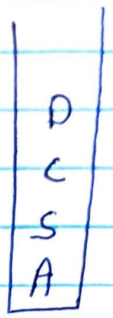
stack

<u>Output</u>: A B S C

step5: Nodes connected to C are D, E, F, S unvisited and alphabetical order - go for D.

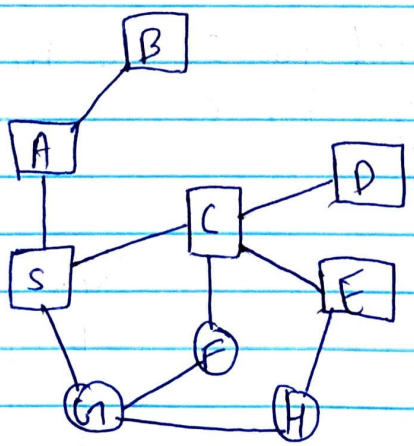


stack

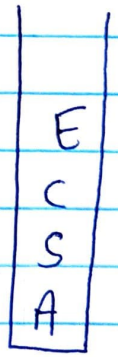


output: A B S C D

step6: No more unvisited, connected nodes for D, so go for unvisited, connected nodes of C i,e E.

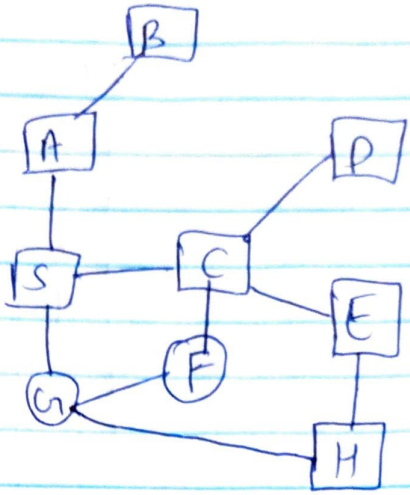


stack



output: A B S C D E

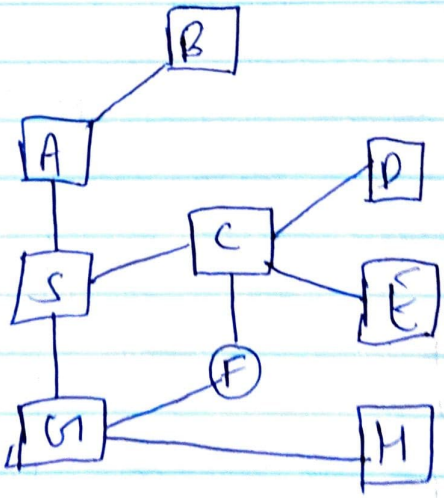
step7: E is on top of the stack, has connected, unvisited node H.



stack

output: A B S C D E H

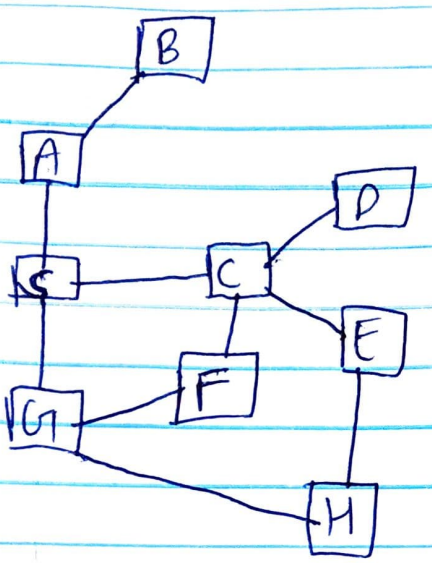
step8: H is on top of stack, has connected, unvisited node G.

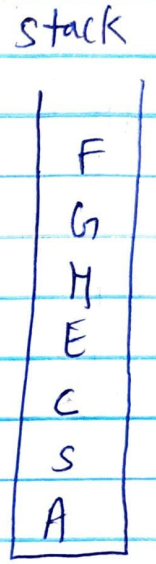


stack

output: A B S C D E H G

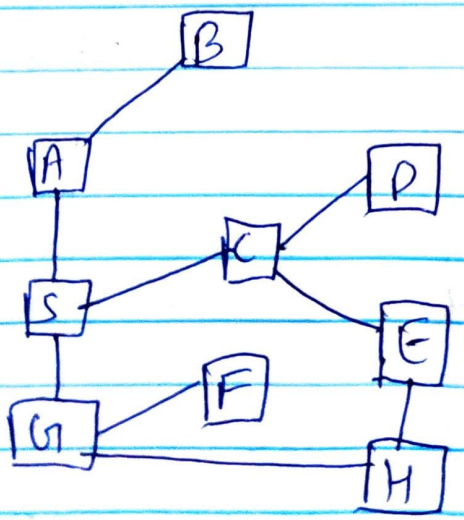
Step9: G is on top of stack, has connected, unvisited node F.





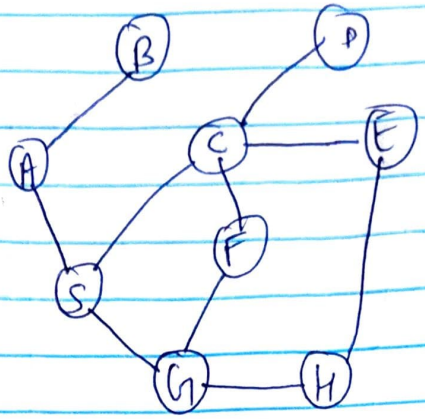
Output: A B S C D E H G F

Step10: F is on top of stack, no unvisited, connected nodes to pop out. All nodes are visited. so we pop out - all elements in stack similarly.



Output: A B S C D E H G F

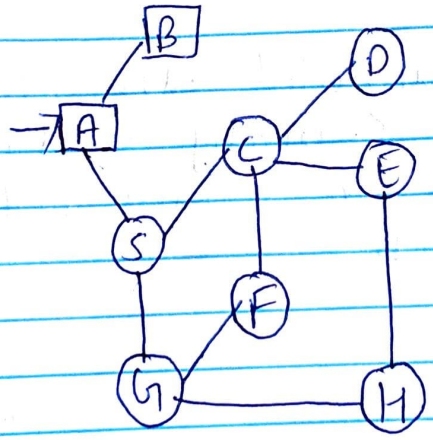
b) Breadth First Search, use alphabetical order to push onto Queue:



□ → visited

"→" → currently working node

step1: Starting at A, it is currently working node.



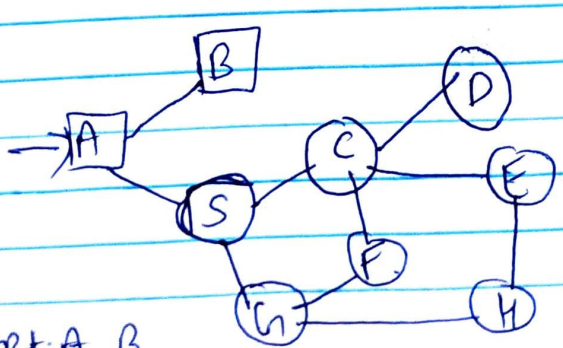
queue

output: A

step2: Adjacent, unvisited nodes of A - B and S go for B.

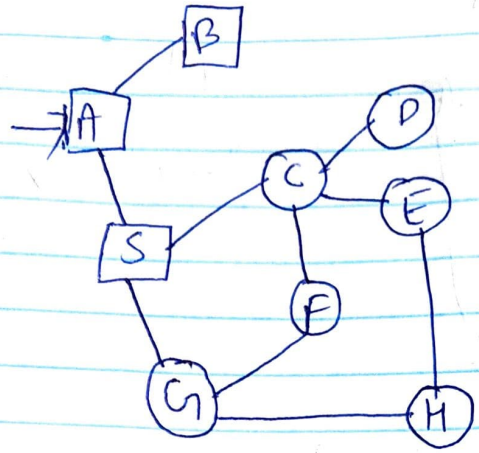
queue

B

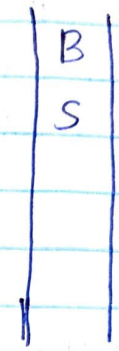


output: A B

Step3:
Now go for S. enqueue it

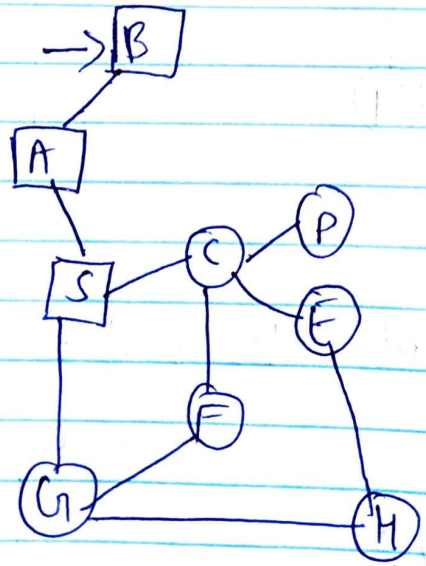


Queue

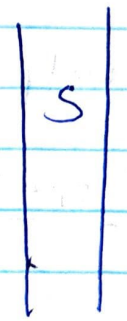


Output: A B S

Step4: No more unvisited adjacent nodes for A, so update currently working node to first element in queueue and dequeue it. Here it is, B.

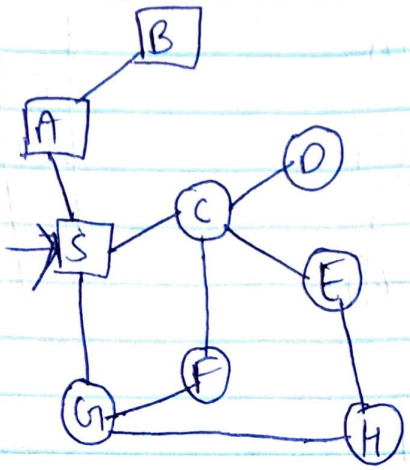


Queue



output: A B S

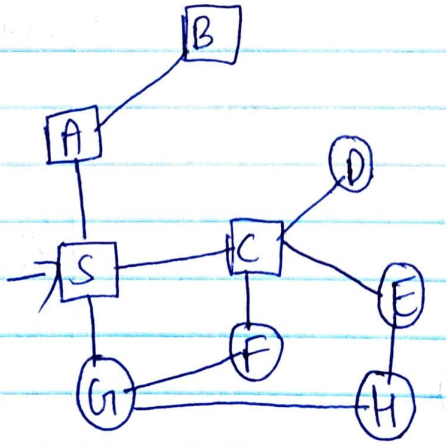
Step5: No more unvisited adjacent nodes for B, so update currently working node to S and dequeue it.



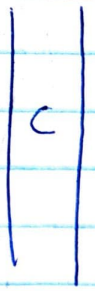
Queue

Output: A B S

Step6: S has three unvisited nodes C, F and G. Enqueue C.

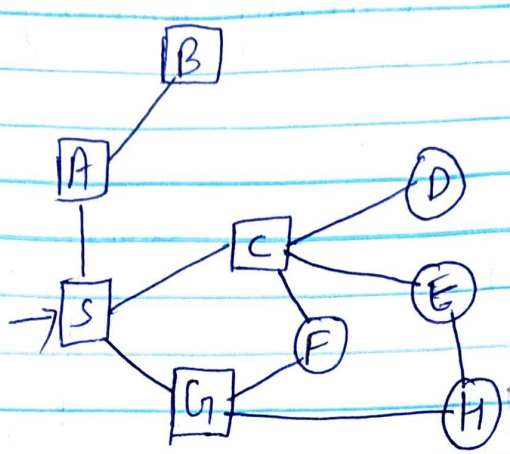


Queue

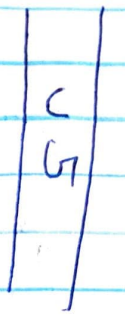


Output: A B S C

Step 7: Enqueue G.

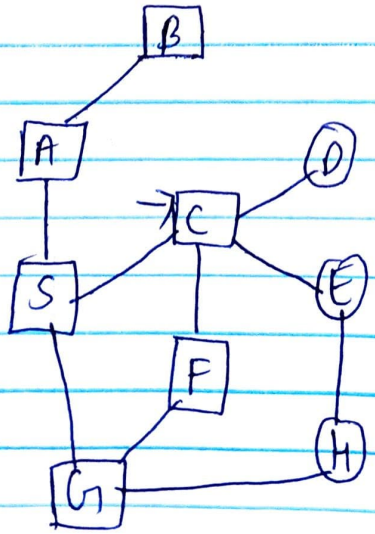


Queue

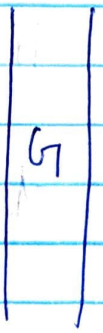


Output: A B S C G

Step 8: No more unvisited adjacent node for S, so update currently working node to C and dequeue it.

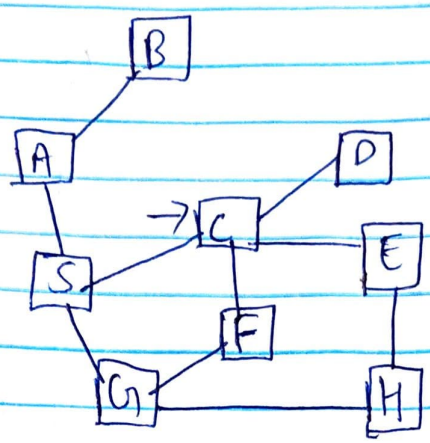


Queue



output: A B S C G

Step 9: Adjacent, unvisited nodes for C are D, E and F. Enqueue them.

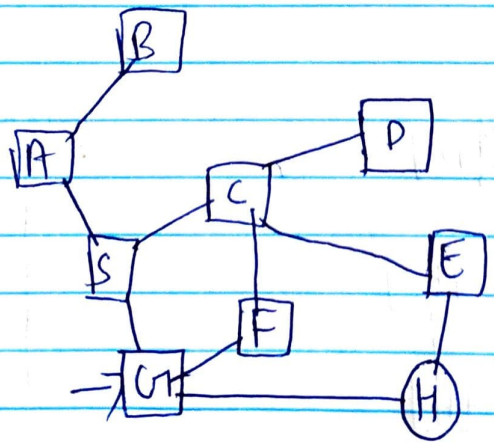


Queue

G
D
E
F

Output: A B S C G D E F

Step 10: update currently working node to G and dequeue it.

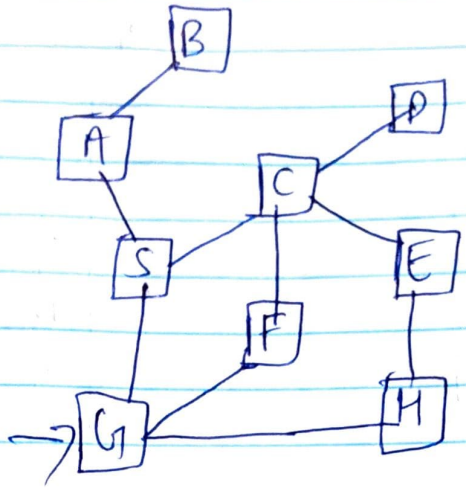


Queue

D
E
F

Output: A B S C G D E F

Step11: G has adjacent unvisited node H. Enqueue it.

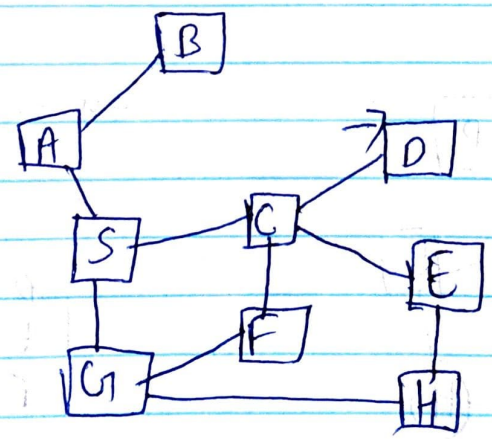


Queue

D
E
F
H

Output: A B S C G D E F H

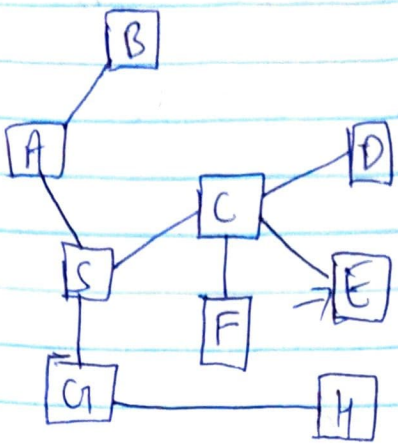
Step12: update currently working node to D and dequeue it.



Queue

E
F
H

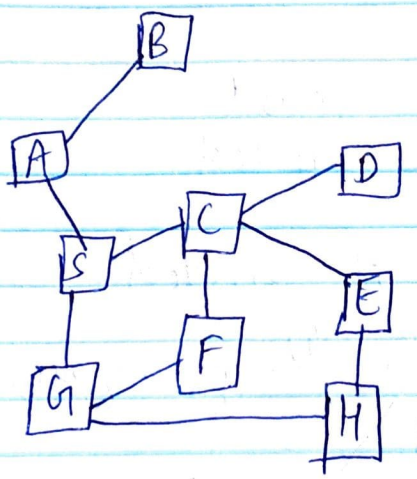
Step13: No more unvisited nodes for D. So update currently working node to E and dequeue it



Queue

F
H

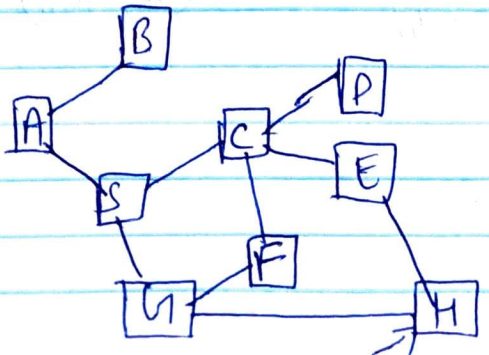
Step14: No more unvisited nodes for E. So update currently working node to F and dequeue it.



Queue

H

Step15: No more unvisited nodes for F. So update currently working node to H and dequeue it.

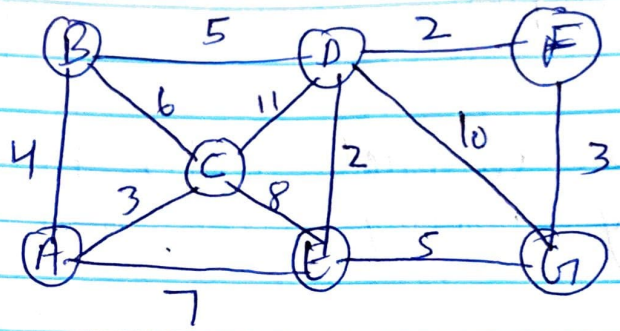


Queue

Queue Empty (No unvisited nodes)

output: A B S C G D E F H

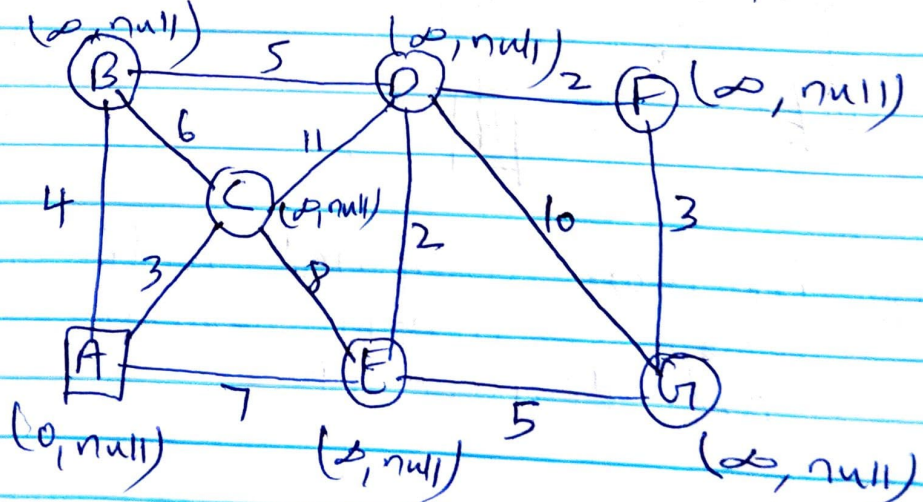
(3) Dijkstra's algorithm



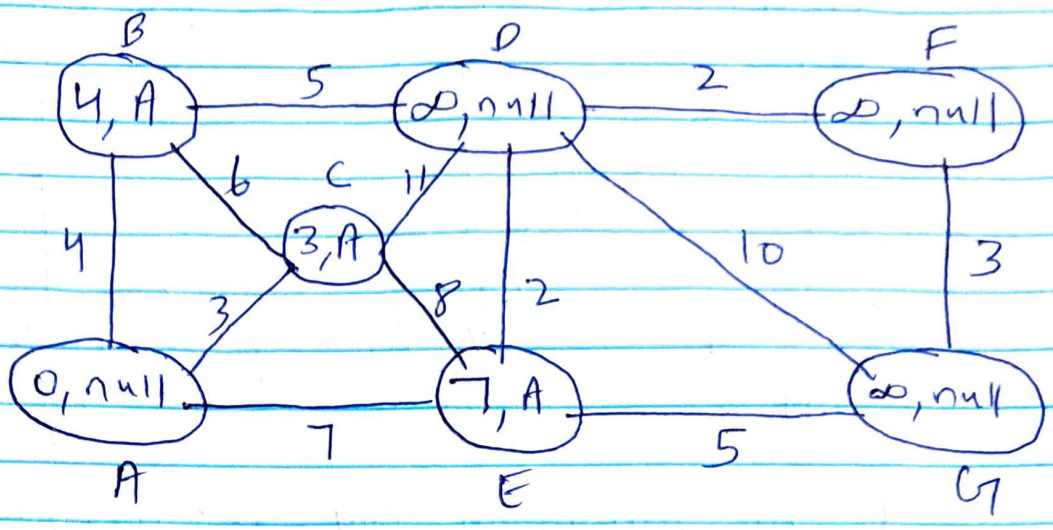
consider ○ — unused

□ — visited

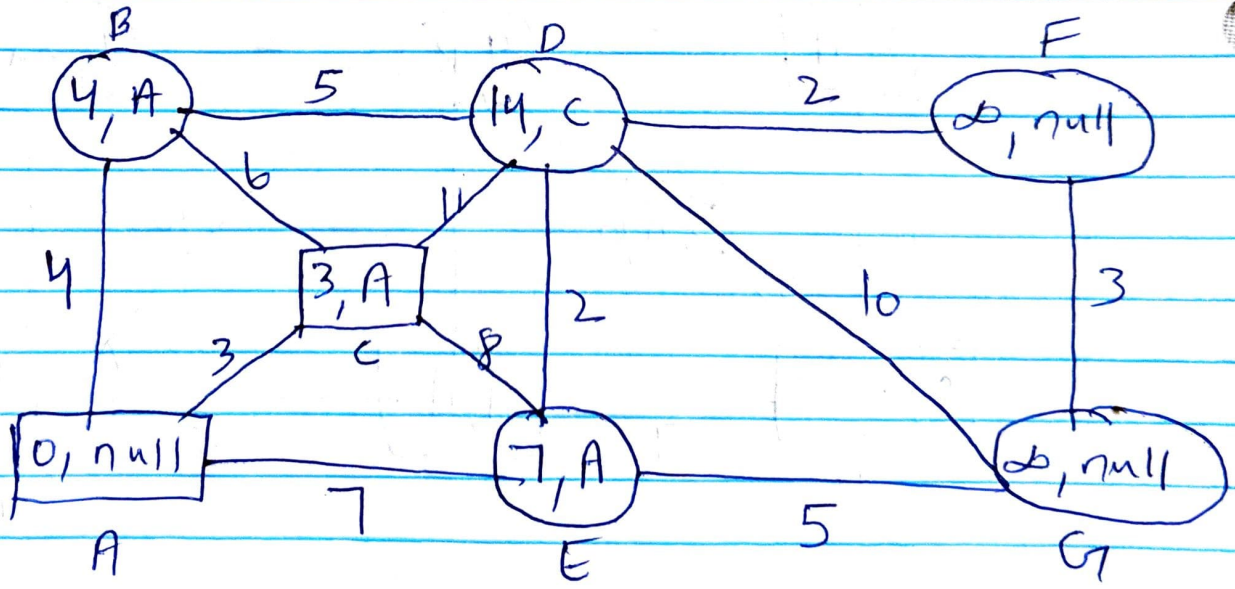
Source — A, make it visited



Minimum cost is from A to C.

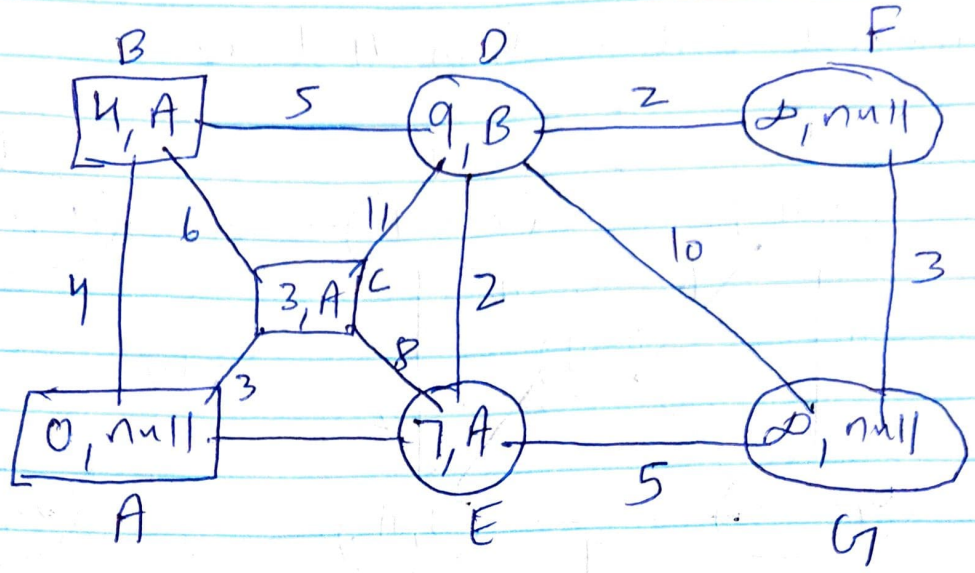


Make C as visited and change adjacent nodes

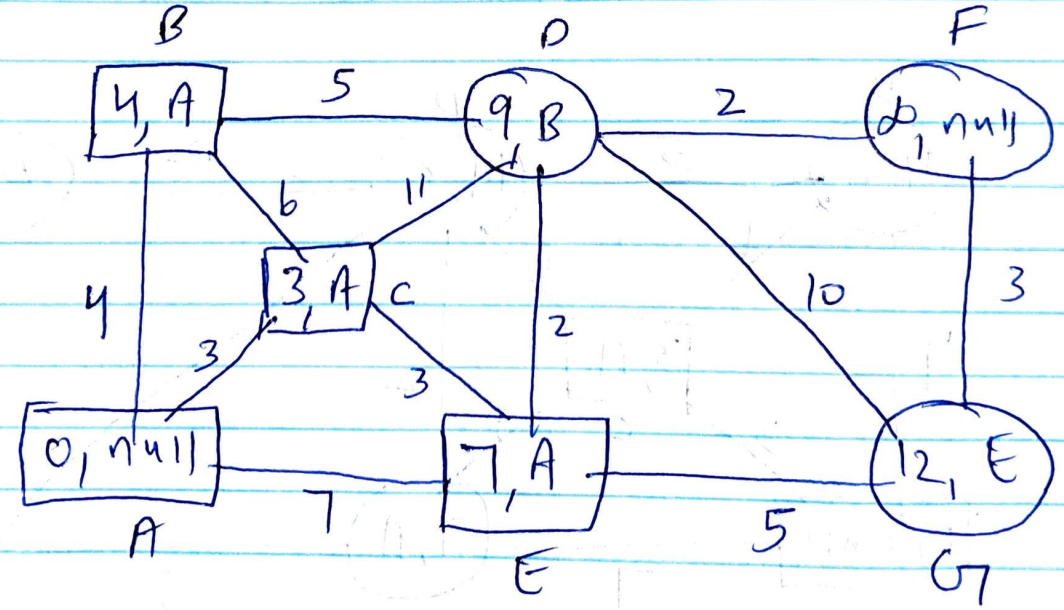


Here (14, C) is greater than (4, A). So cost of D from B is 4+5=9. D=(9, B)

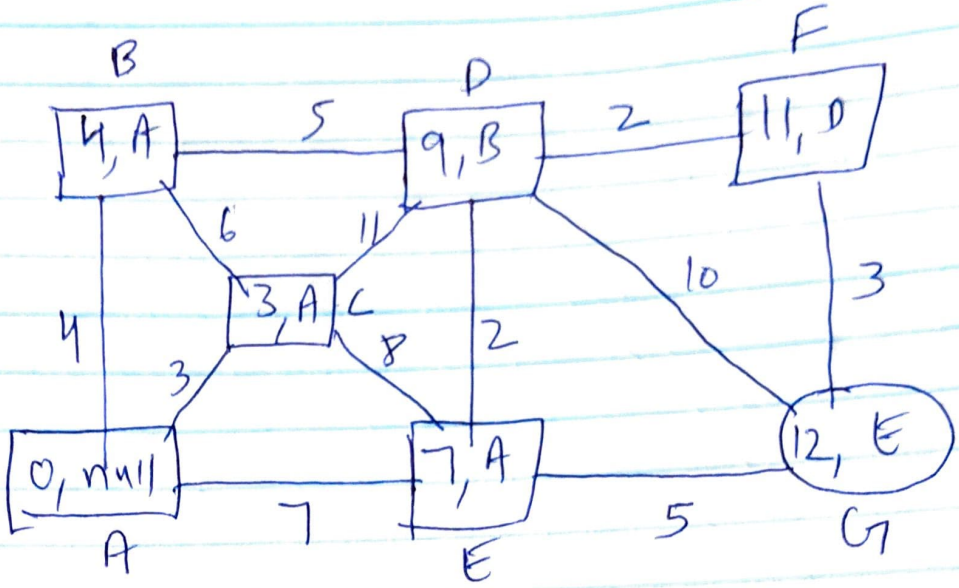
Make B as visited



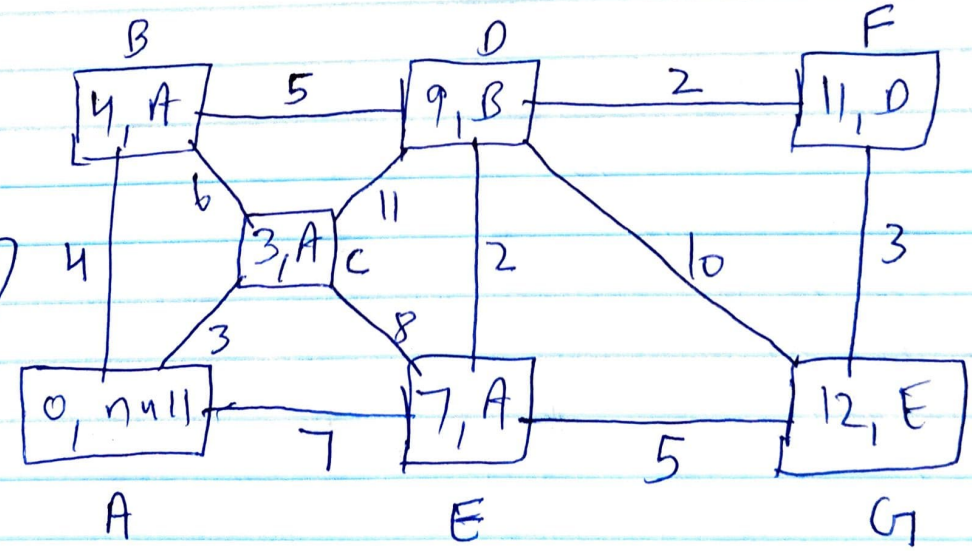
E has minimum cost, so we check it's adjacent node distance with D.



node D has least cost among D and G, so we select node D be minimum and calculate cost from D to F, making node D visited.



Next least cost is 12 from node E to G, change G node to visited.



final → Graph

Distance

A	C	B	E	D	F	G
0	3	4	7	9	11	12

Path A → C → B → E → D → F → G