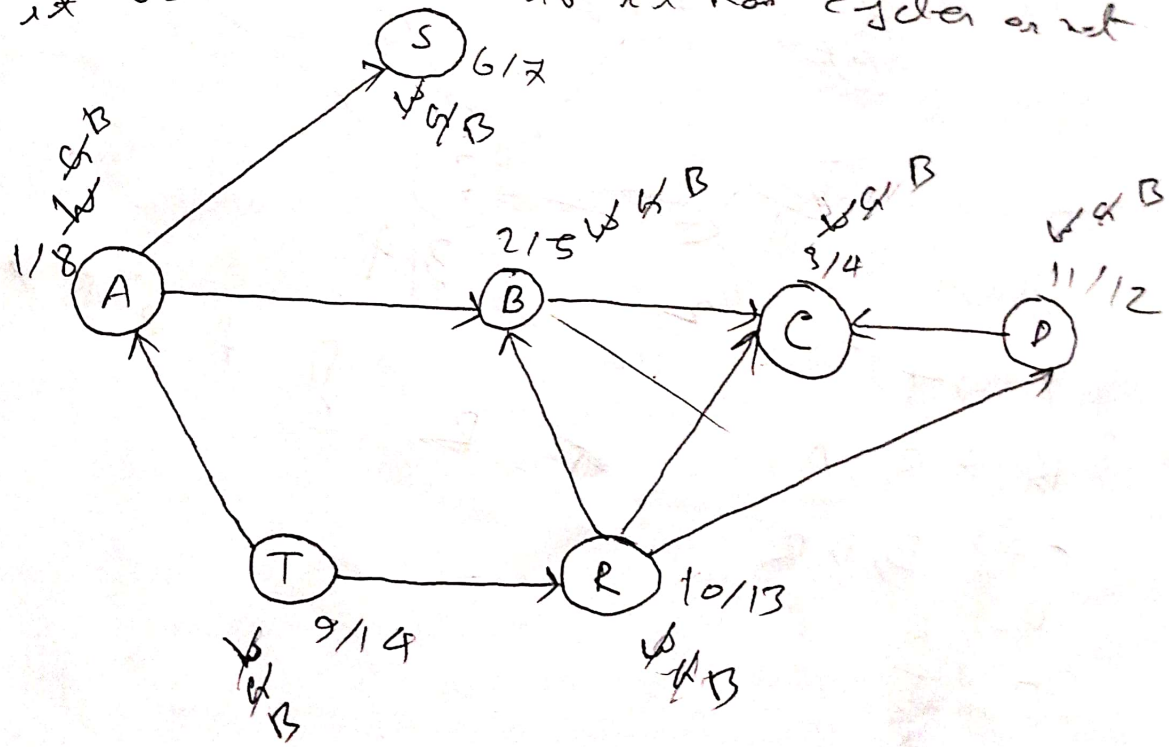
1) Given graph,

Here we have to do DFS because only by it we can check it it has cycles or not



Here,

W means not visited

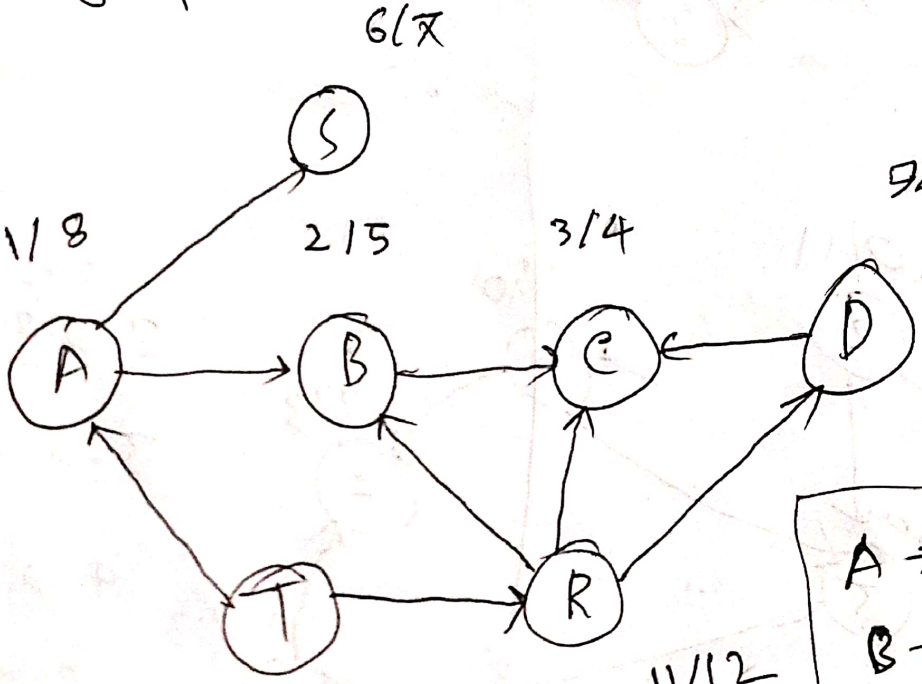
G means visited

and B = all child nodes are visited.

In the first comparison we see, A → B(in first traverse A → B for which (G = W) so we conclude not cycle. Then, for B → C it shows (G = W) which indicates it's not cycle. As C has no other remaining so C turns to B. Now as B's all childs are visited B to turns to B (Black). Now, As A has a one child left which is S so now A → S and G = W and so it's not a cycle. As S has no child so it will turn to B. Now Beside, A also turns to B B (Black) for visiting all childs. Now for T it will go to R so T → R and (G = W) so there is no cycle and R will go to D for which (G = W) and there will be no cycle. In conclusion we can say as there is no (G = G) between the vertices so there is no cycle in the graph.

2) b) For topological sort we need to do DFS in the graph by only looking into the starting time and ending time.

Given graph



A → B, S
B → C
C →
D → C
R → B, C, D
S →
T → A, R

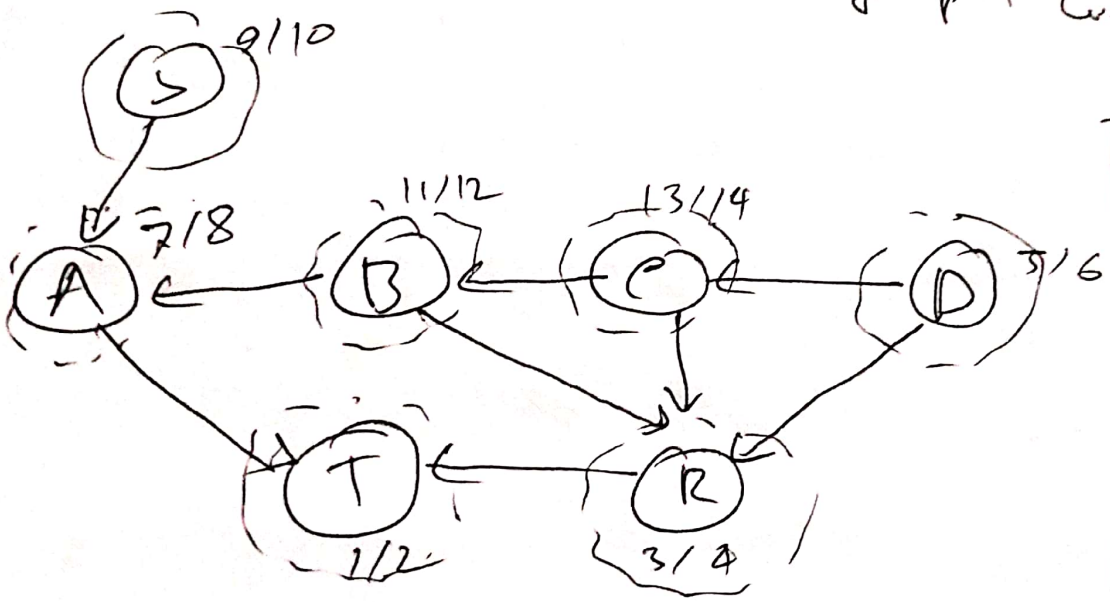
For ordering we have to sort it in descending order according to the finishing time.

So, the topological order is

T R D A S B C

1(c) As in 1(b) we have done DFS and finished sorting in decreasing order. Now we have to make transpose of the graph and do DFS on it according to the ~~tran~~ topological order we got from 1(b)

So, The transpose of the graph will be,



A → T
B → R, A
C → B, D, R
D → R
R → T
S → A
T →

~~B~~ The order from 1(b)

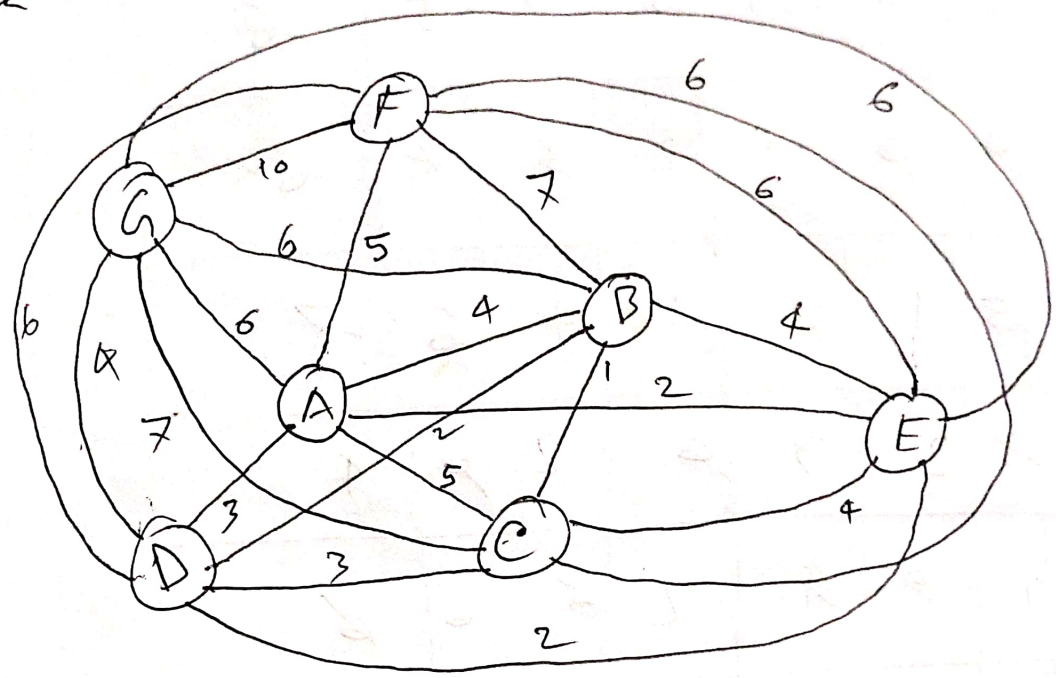
~~T R D A S B C~~

So, strongly connected components are

(T) (R) (D) (A) (S) (B) (C)

So, we can say that each node is strongly connected to its only self.

2) From the given table the graph we will get is,



So, the priority queue will be,

As A is the source the so nothing A=0.

A→0
B→∞ 4 2
C→∞ 5 4 3 2
D→∞ 3 2
E→∞ 2
F→∞ 5
G→∞ 6 4

So the stack = AEDBCFG

	A	B	C	D	E	F	G
A	0	4	5	3	2	5	6
B	4	0	1	2	4	7	6
C	5	1	0	3	4	6	7
D	3	2	3	0	2	6	4
E	2	4	4	2	0	6	6
F	5	7	6	6	6	0	10
G	6	6	7	4	6	10	0

those which are deleted.

	A	B	C	D	E	F	G
A					2	5	
B			1	2			
C		1					
D		2		2			4
E	2			2			
F	5						
G				4			

those which ~~are taken~~ are chosen.

Now,

Order in which the rows are deleted:

From A = A D are G are deleted.

From B = A to → B and E → B G are deleted.

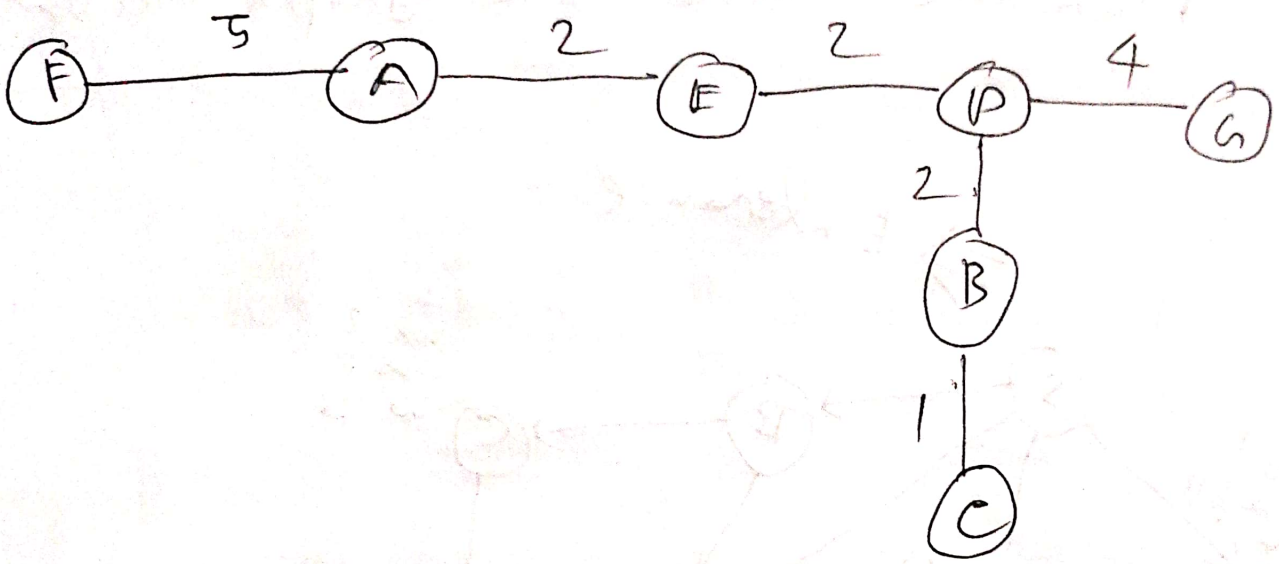
From C = A and C to the G will be Deleted.

From D = A, C and E to the F node will be deleted.

From E = B to E to G and B to C will be deleted.

From F = B to the G will be deleted.

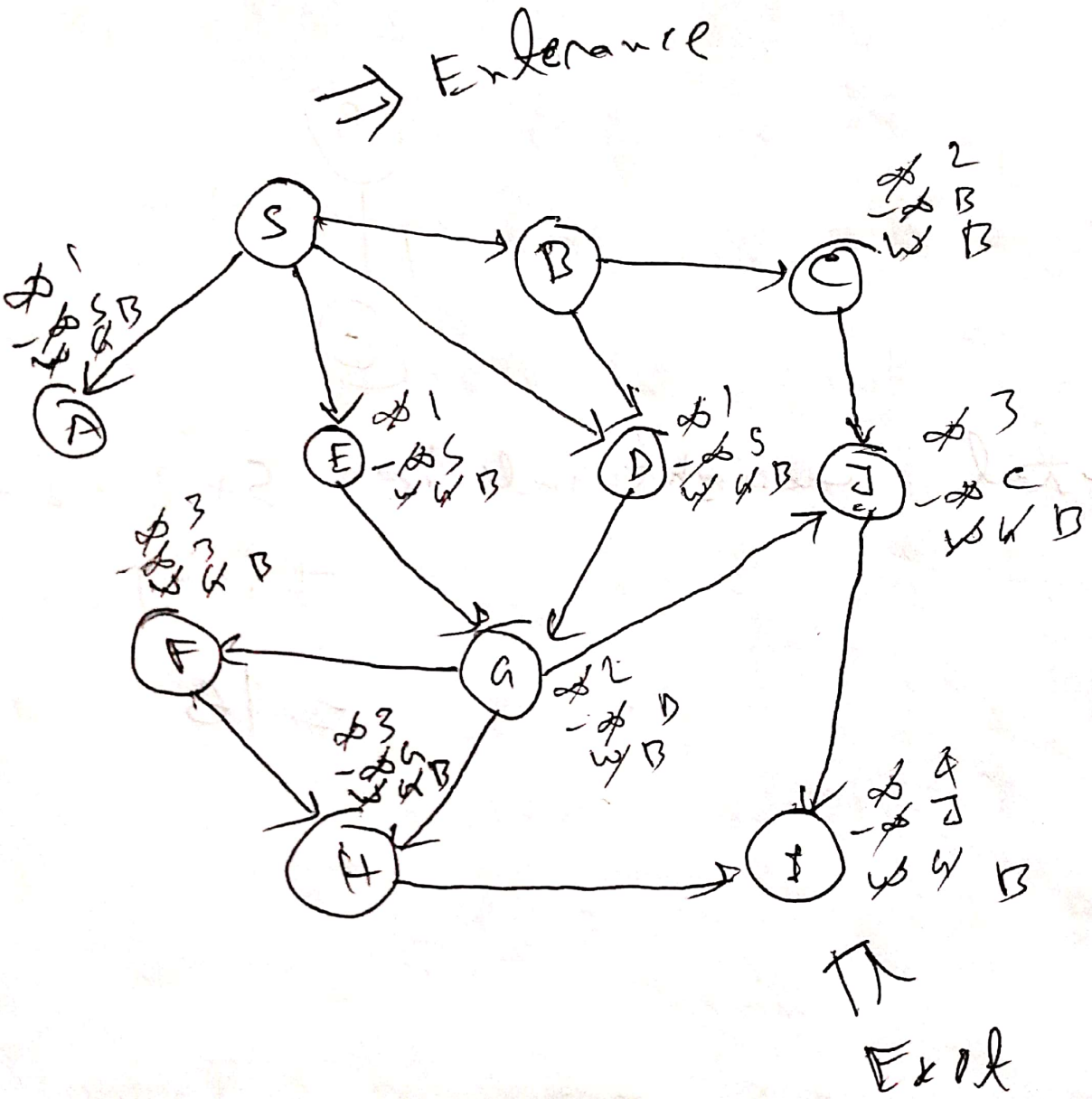
From G = A to the C and E to the G will be deleted.



It's total weight will be: $5+2+2+4+2+1$

$= 16$

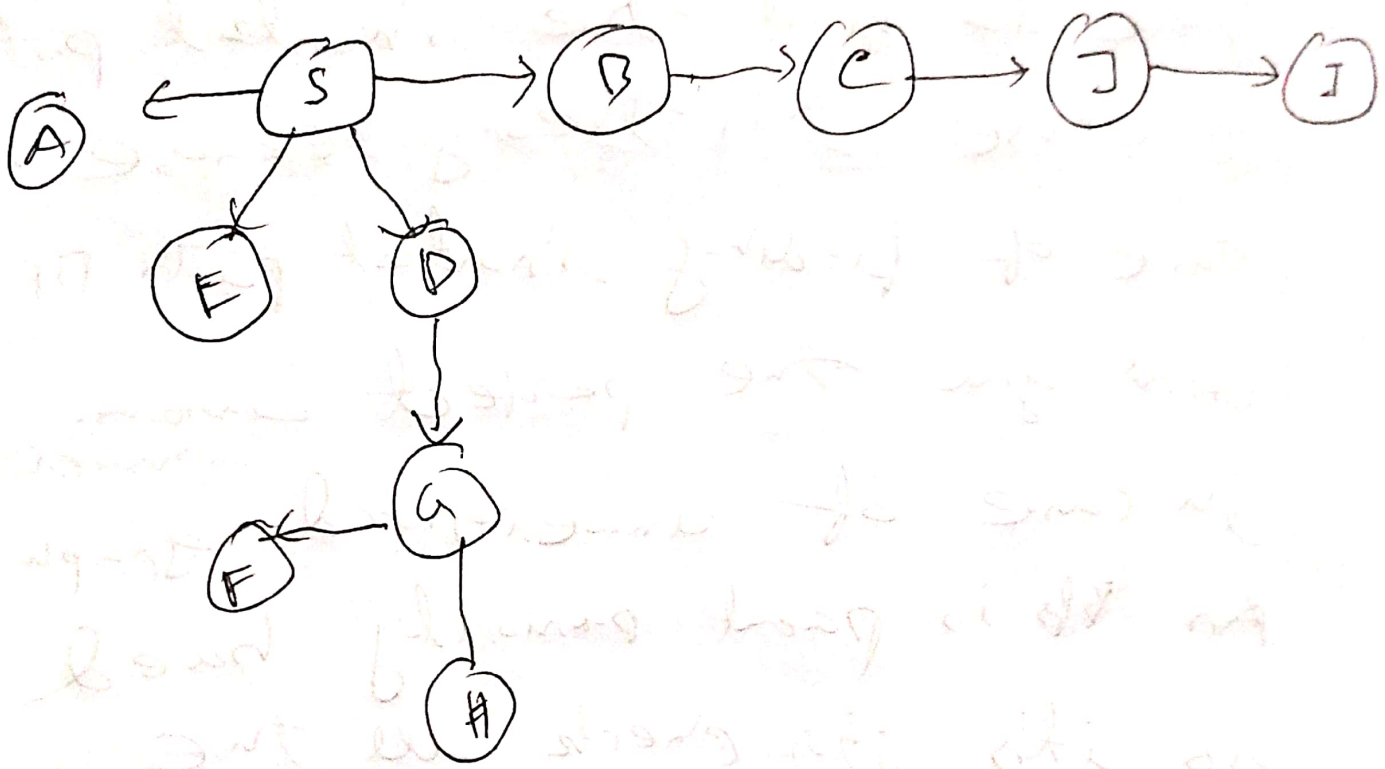
3) From the maze the graph will be:



From BFS we will get,

Q ⇒ S A B D E C G J F H I

After BFS the graph will be;



So the shortest path from Entrance to exit will be.

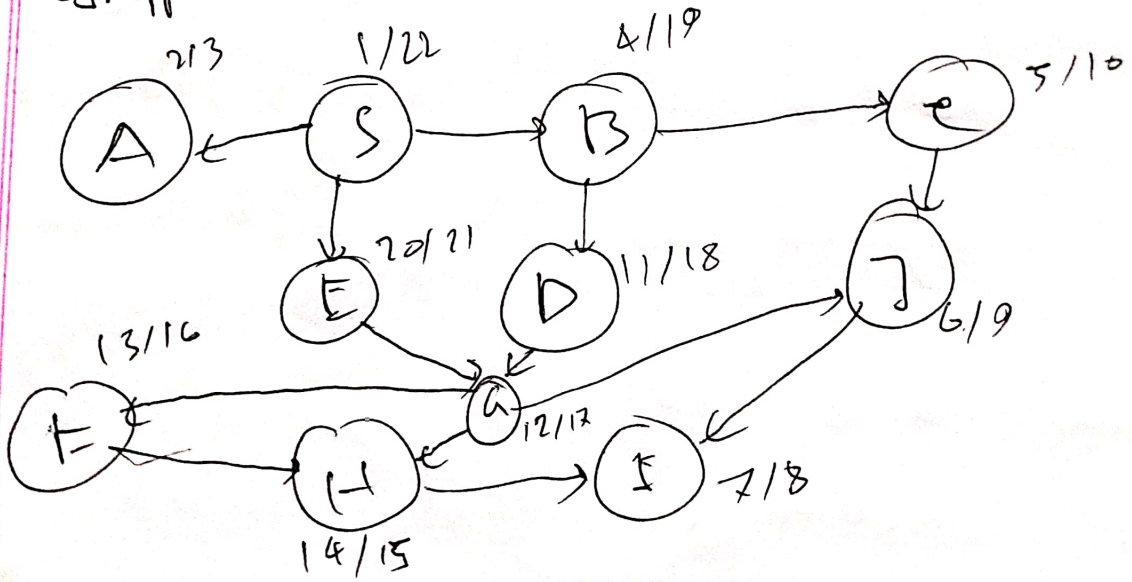
S → B → C → J → I.

I chose the BFS algorithm have to find the shortest path. I chose this theory because in case of finding shortest path BFS will give the perfect answer. in case of unweighted graph. An BFS is part parionity based so it's its check all the nodes on the finest step. So finding shonest path in unweighted graph ida is method.

From ~~comparing~~ comparing we can say oft DFS here. DFS is a child prionity based algorithm. DFS will not ~~with~~ not make sure that. It will go deepa to node to from the source node. It does not give surity that

it is one node is visited before another node. starting from a source vertex, then that node is closer to the source than the another node. So using BFS is a better option here.

From using DFS we will get from the graph.



From DFS the shortest path will be:

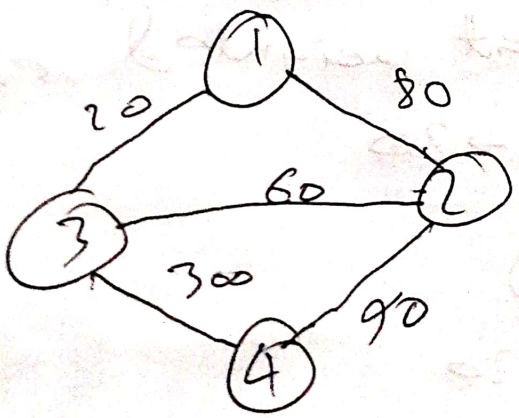
$S \rightarrow B \rightarrow C \rightarrow J \rightarrow J$

Though BFS and DFS gives same answer in this scenario for shortest path

but planning BFS will give the easiest way to find the shortest path for this type of graph.

4) a) For the given scenario kruskal algorithm will be appropriate. Besides, kruskal algorithm is a faster algorithm than prims because its time complexity is less than prims. where as prim's complexity is O((E+V) logV) and kruskal's kruskal's complexity is O(ElogV). So using kruskal is a better option.

b) The graph will be,



Ans) From the given inputs if we arrange it we ascending order in term of weight we will get,

(3,1) → 20

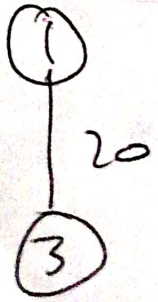
(2,3) → 60

(1,2) → 80

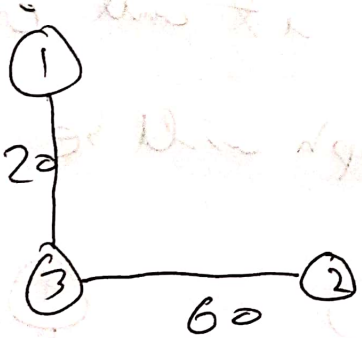
(2,4) → 90

(3,4) → 300

So we have to first take the smallest weighted connection which is (3,1) → 20

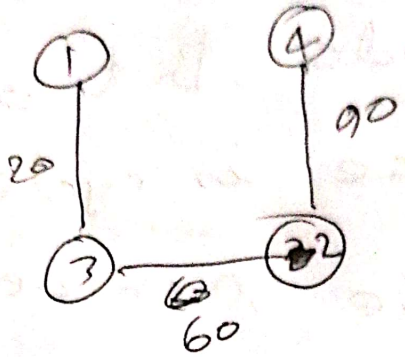


Now we have take next connection from the order. But we have to check if there is cycle or not. If there is cycle then we can't add. Here we add (2,3) → 60 as it does not create cycle.



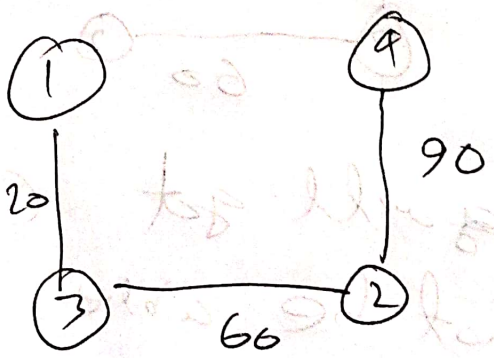
Now we will get (1,2) → 80 but we will not be able add as it will create cycle.

Now, for (2,4) → 90 we can add it in the graph as it will not create cycle.



Now lastly we have (3,4) → 300.
But we will not be able to add it as it will create a cycle.

So, the graph will be after the simulation,



Here we have E = 3 which is (V-1) meaning (4-1).

So, here the maximum speed is 90.