H6

Page No.
Date

1] ① → let i represent rows and j represent columns
② → iterate through all rows
③ → If there exist i for which
matrix [i] [j] = 0
for all the values of j
then
(3.1) → ~~For~~ j = i
(3.2) → for all the value of i ~~(except i = j)~~
if matrix [i] [j] = 1 or Non-zero
then return true
else
continue the 3rd step from i+1

④ Return False

Proof of correctness :

Because if kth node satisfies the condition then all the entries in kth row will be '0' and all the entries in kth column will be 'Non-zero'

Run-time ⇒ $O(n^2)$

2] Algo 1:

Note: (1): If DFS of node is completed then it will be black.
(2): If it is in DFS then Grey
(3): If Not visited then white.

(1) → Start the DFS on the graph from any point.
(2) → At every step we will maintain parent-child relation
Note: We can use Parent array for this purpose index as parent and at that index we will add it's child, at a time one parent will always have one child
(3) → while performing DFS if we came accross a parent-child relation in which child is already grey and it is not the parent of current parent (in the case of graph have just 2 node) Then there exist a cycle in the graph
(4) → To print that cycle store the child which was already grey is C, let P = C

```
while (1)
{
    if (Parent [P] == C)
    Break;
    prin ("-", Parent [P], "-")
    P = Parent [P]
}
```

Algo 2:

Same as Algo 1: along with following changes

In step 3 rather than starting from any vertex start from one of the vertex of e
ie (e_1 or e_2) → let these be vertex forming e
# while doing DFS
if (e_1 becomes C and e_2 is still grey ||
e_2 become C and e_1 is still grey)
Go the step (4) // If G contain cycle with e
if (e_1 become black || e_2 become black.)
retur G does not contain cycle with e

Date

3] let us assume that there is an Edge E(x,y) that is not in T but is in G, where x and y are the vertex forming the Edge E

During BFS:

There has to be a time when one of x and y was discovered first time and other was not, let without loss of generality it be x, as BFS is running it will visit all the adjacent vertics of x including y so E(x,y) will be the part of BFS tree — (1)

During DFS

There has to be a time when one of x and y was discovered first time and other was not, let without loss of generality it be x, when we reach and x we will perform DFS, which will perform the DFS on it's adjacent vertices including y, then E(x,y) will be in the DFS tree — (2)

From (1) and (2) our assumption is wrong

∴ G = T

Second logic for proof:

As BFS tree = DFS tree the graph G cannot have any cycle as if it does the DFS and BFS tree may not be ~~equal~~ same, if a graph does not contain a cycle then it is a tree and BFS and DFS ~~tree~~ of the tree is tree itself

$\therefore$ $\boxed{G = T}$

4]

① → Initialize a counter at 0 and initialize a variable tip
② → start DFS at any vertex of the graph
③ → when-ever a node becomes grey counter++
when-ever a node become black counter--
④ → when-ever counter becomes greater than it's previous maximum value store that node in tip
⑤ → At the end of this we will get one end of longest path in tip
⑥ → Start the DFS from the tip reset the counter to 0 and repeate ③ to ④
⑦ → Now in tip will get the second end of the longest path
⑧ → path between the first end and the second end will be the longest path.

5] ① → let c = 0

② → V = { $v_1, v_2 \ldots v_n$ } be set of all the vertic which have and edge $s \rightarrow v_i$

③ → for (i = 1 to n)

{

③.① → Mark s as visited

③.② → Perform DFS from v_i

③.③ → while doing DFS if the visting vertex = t

then ⇒

c++

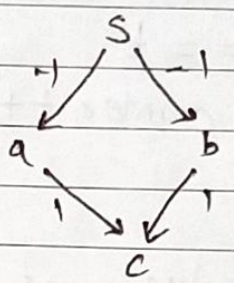
}

④ Return the value of c As the No. of paths

6] No it doesn't

Example:

Consider a graph:

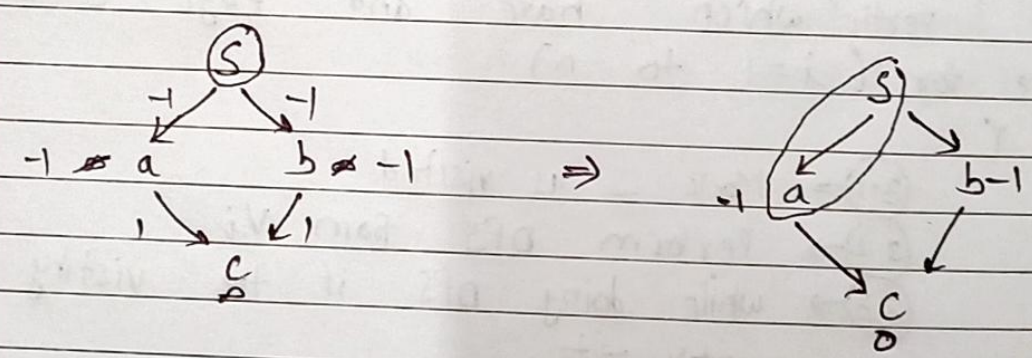


Actual distance : $s \to a \Rightarrow -1$

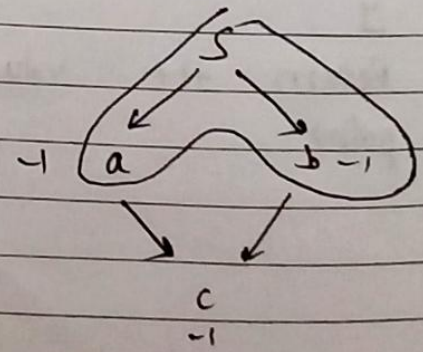
$s \to b \Rightarrow -1$

$s \to c \Rightarrow 0$

How Dijkstra's work $\Rightarrow$



$\Downarrow$



∴ Distance Accordig to Dijkstras

$s \to a \Rightarrow -1$

$s \to b \Rightarrow -1$

$s \to c \Rightarrow -1$

which is not correct.

Page No.
Date

7. ① → shortest path tree rooted at v is a tree that spans from v to all other vertices
② → At the same time it is important to keep the summation of all the edges present in the tree minimum
③ → To do so it is important to select the path ~~edge~~ which has minimum weight while goin from one vertex to another
④ → while using prims algorithm we keep two section visited graph and unvisited and one by one we bring and vertex from unvisited section to visted
⑤ → while selecting this vertex we see all the edges that connect visited and unvisited graph and select the one which is minimum
⑥ → This exactly what we have done in step (3)
⑦ Hence the postulate is Correct