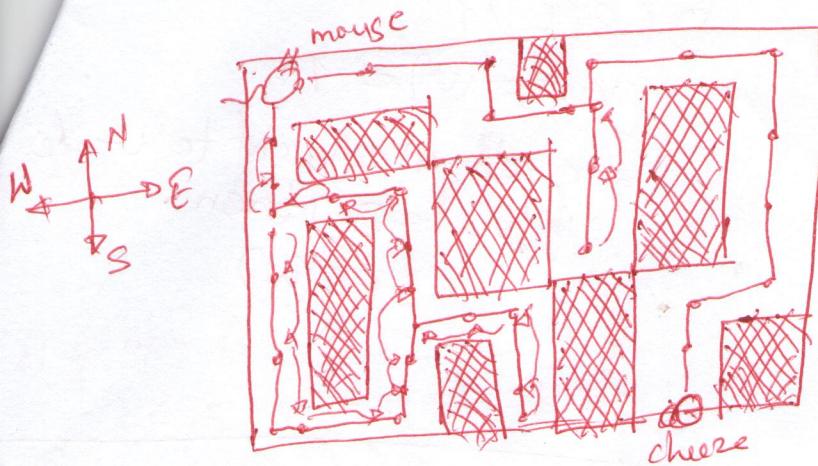
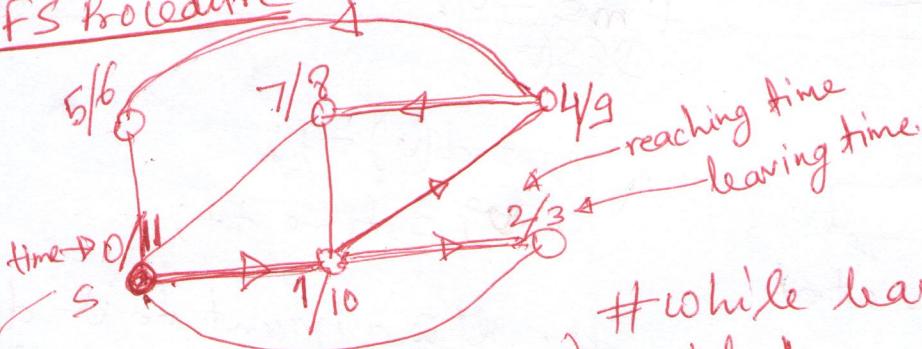


Depth First Search (DFS)



The mouse wants to find the cheese in the maze.
⇒ DFS way of exploring.

DFS Procedure

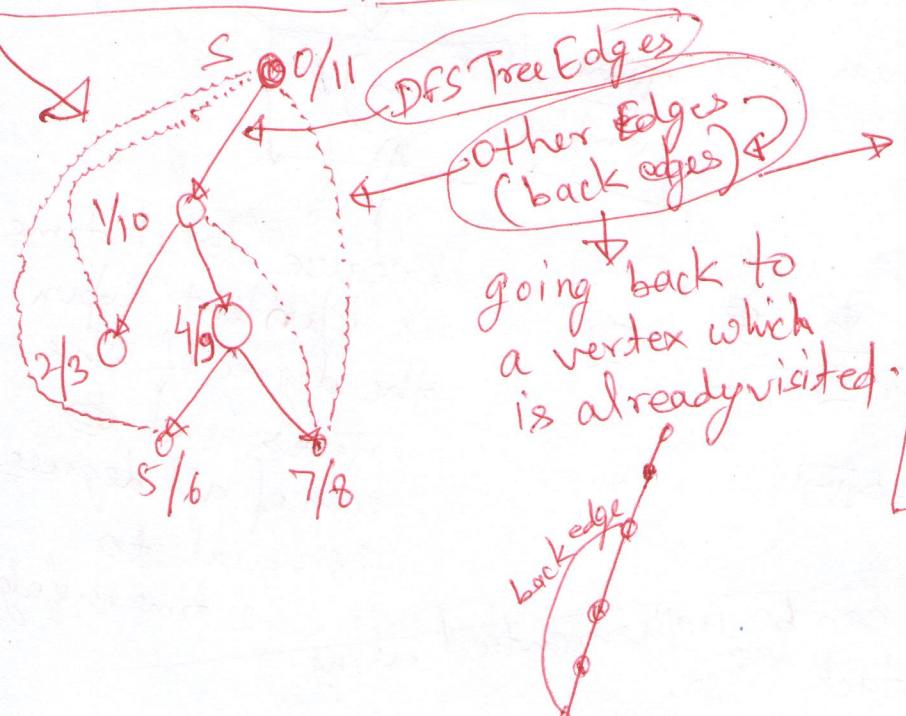


⇒ The no. of edges in DFS Tree is $n-1$.

form a connected subgraph

- # while leaving a node, color it black.
- # Unexplored nodes are white.
- # Gray nodes are in process and its ~~all children~~ all children needs to be explored.
- # If a node is white, then only it will be explored.

⇒ The visited edges form a DFS Tree.

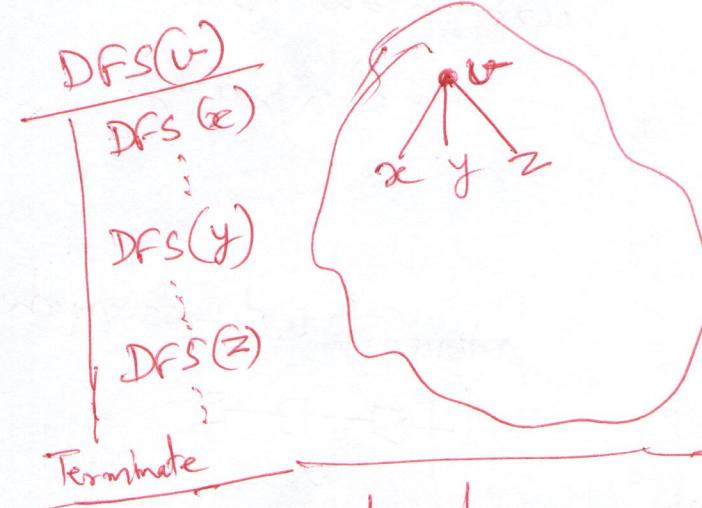
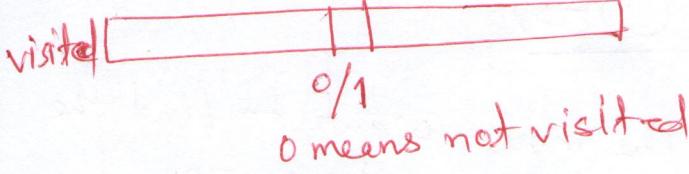


An edge from a node to an ancestor is called the back edges except parent.

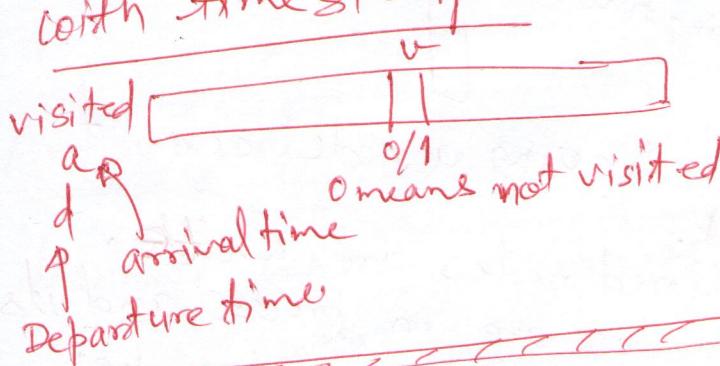
⇒ DFS will divide all edges into either tree edges or back edges.

DFS will divide all edges into either tree edges or back edges.

⑧ DFS Implementation



with timestamp.



Running Time (DFS)

DFS(v){
visited[v] = 1;
for all w adjacent to v do
if !visited[w] then
DFS(w)}

DFS(v){
visited[v] = 1
for all w adjacent to v do
if !visited[w] then
DFS(w)}

time = 0;

DFS(v){

visited[v] = 1;

a[v] = time ++;

for all w adjacent to v do

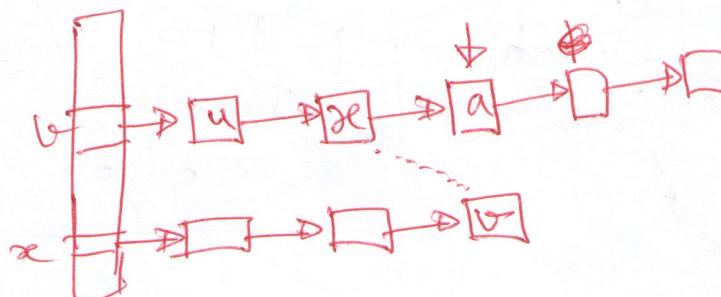
if !visited[w] then
(v,w) is a tree edge
DFS(w)

d[v] = time ++;

}

Assume graph is connected.

Time Complexity $O(m)$



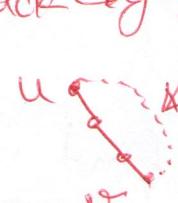
Adjacency list

It can be implemented using stack also.

Because total time is dependent upon the degree of the nodes and sum of all degrees is equal to $2 \times \text{no. of edges}$

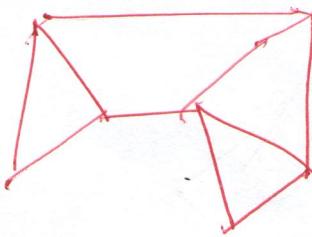
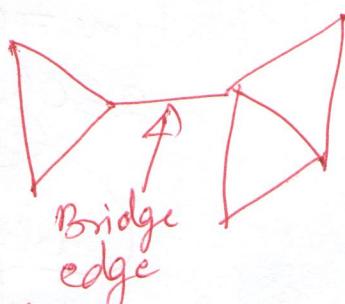
(u, v) tree edges


$\text{arr}[u] < \text{arr}[v]$
 $\text{dep}[u] > \text{dep}[v]$

(u, v) back edges


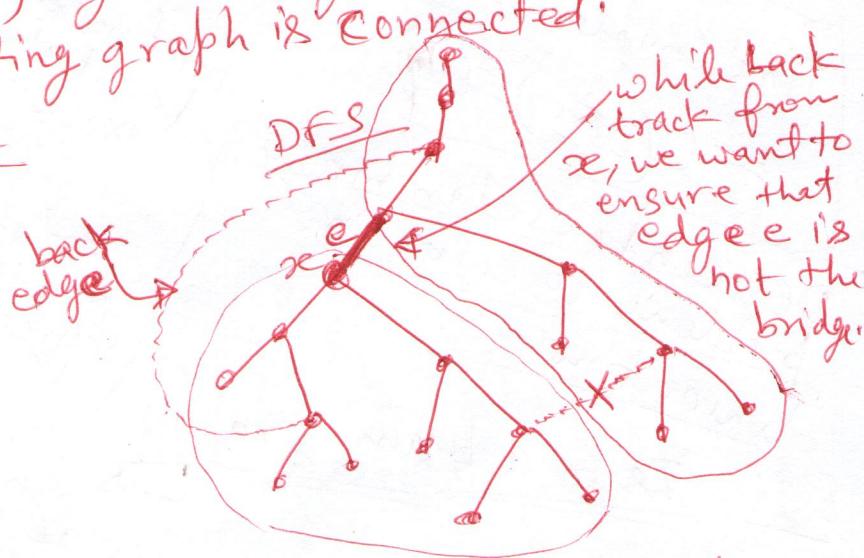
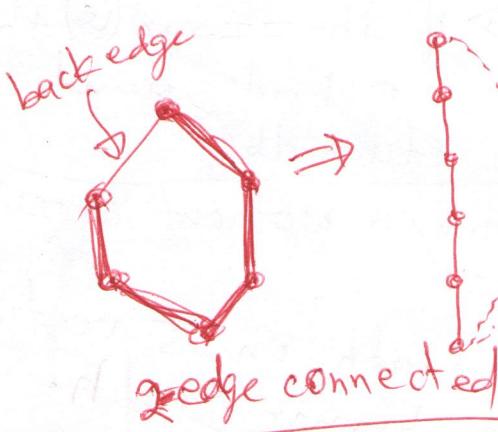
$\text{arr}[u] < \text{arr}[v]$
 $\text{dep}(u) > \text{dep}[v]$

A graph is 2-Edge Connected \Rightarrow iff it remains connected after the removal of any one edge.
A graph G is 2-edge connected iff $G - \{e\}$ is connected $\forall e \in E$

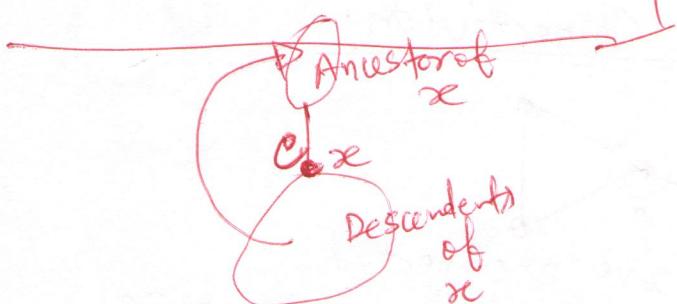


How to check if a given graph is 2-edge connected?

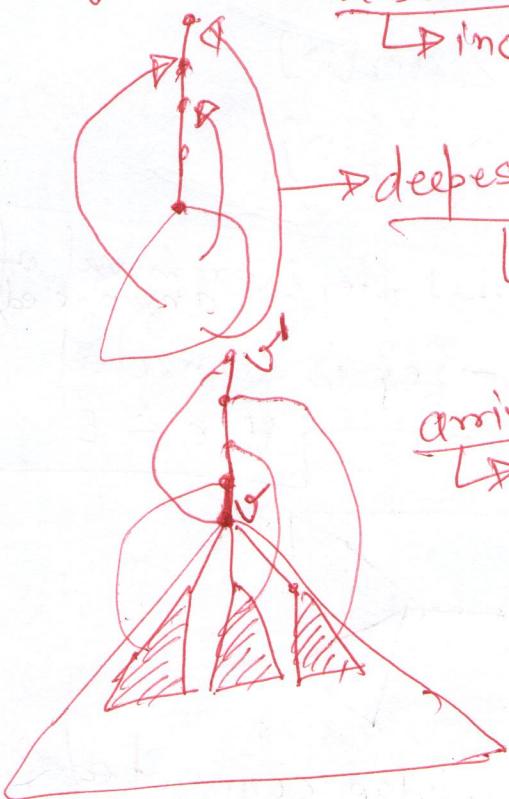
We can do it in $O(m^2)$ by removing every edge & checking if resulting graph is connected.
Costly procedure



If there is any back edge from ~~any~~ any descendant of x to any ancestor of x , then edge e can not be the bridge edge.



⑩ When back tracking from a node v , we need to ensure that there is a back edge from some descendant of v to some ancestor of v .
 ↳ including node v itself ↳ parent or above.

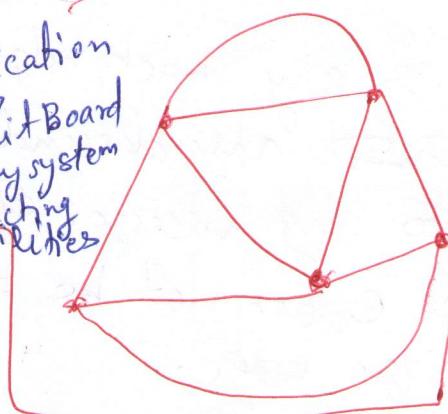


Time Complexity
 $= O(DFS) = O(M)$

DFS can be used in many problems to solve in linear time.

E.g. Is G a planar graph?

Application
 ↳ Circuit Board
 ↳ Subway system
 ↳ Connecting utilities



arrival time of deepest back edge
 ↳ minimum needs to be traced

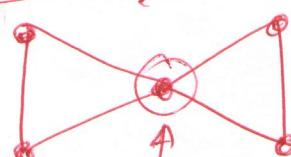
deepest back edge
 time = 0
 $2EC(v)$
 $dbe = \min\{arr[w] \mid w \text{ is adjacent to } v\}$
 $arr(v) = time + 1$
 $visited(v) = 1$

for all w adjacent to v
 if $\neg visited(w)$ then
 $dbe = \min(dbe, 2EC(w))$
 else
 $dbe = \min(dbe, arr[w])$
 if $dbe == arr(v)$ then
 ↳ bridge found.
 return dbe

Can be drawn without crossing of edges.

⇒ A complete graph on 5 vertices is not a planar graph.

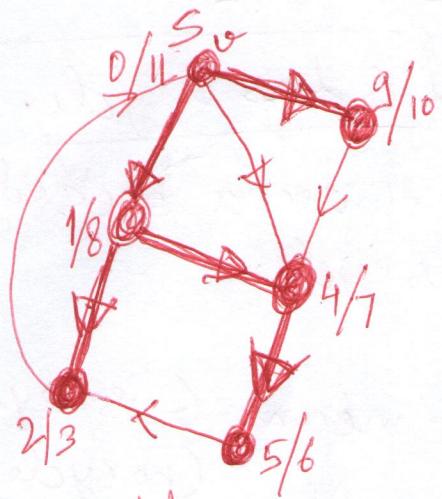
↳ 2-Vertex Connectivity:



not 2 vertex connected

Can be checked in linear time using DFS.

DFS in Directed Graphs



$\text{DFS}(v) \{$
 visited $[v] = 1$ basically outadjacent
 for all w adjacent to v do
 if !visited $[w]$ then
 $\text{DFS}(w)$

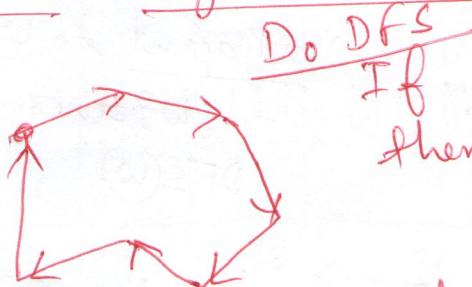
v w
 x y
 z t
 s r
 u p
 q m
 n o
 f e
 d c
 b a
 g h
 i j
 k l
 m n
 o p
 q r
 s t
 u v
 w x
 y z
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u v
 v w
 w x
 x y
 y z
 z t
 t u
 u <span style="border: 1px solid black; border-radius: 50%; padding

(12)

Cross Edge

$$(x, y) \Rightarrow \text{arr}[y] < \text{dep}[y] < \text{arr}[x] < \text{dep}[x]$$

Given a directed graph G, check if G has a cycle.



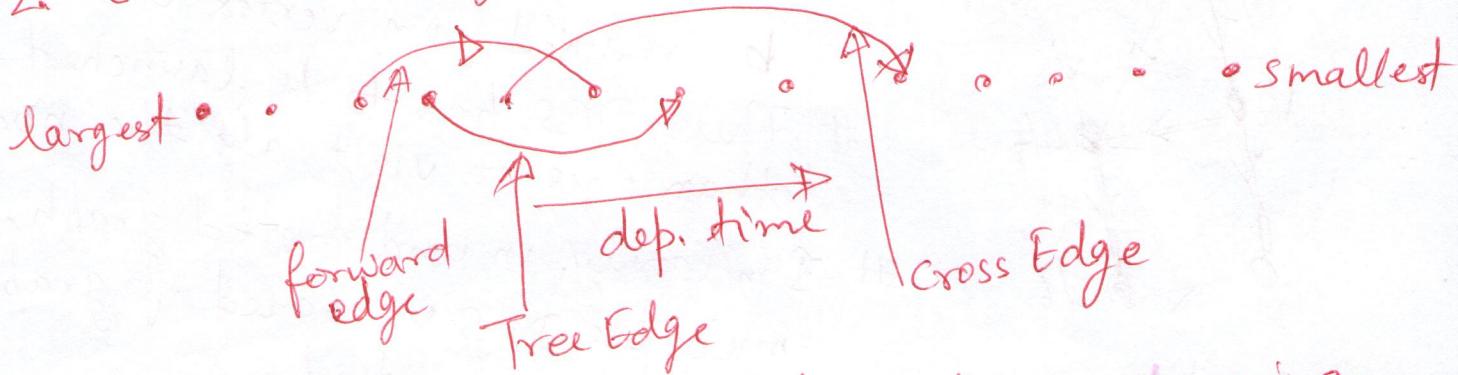
Do DFS
If we encounter a back edge
then G has cycle.

\Rightarrow If there is no back edge, does that mean G is acyclic (no cycle)?

statement : No Back Edge \equiv No Cycle.

1. Do DFS

2. order vertices by their departure times



Because, all forward, cross and tree edges are going from left to right, so cycle can not be possible if back edge is not present.

Given an acyclic graph, we can order the vertices of G, so that every edge goes from left to right.

\hookrightarrow This ordering is also called a topological sort.

\rightarrow maximum departure time = $2n-1$

A directed acyclic graph is ~~not~~ also called

~~a DAG~~

It can be computed in $O(m+n)$ time.

Undirected graphs:

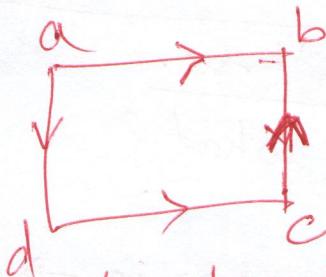
\hookrightarrow Connected \Rightarrow There is a path b/w every pair of vertices.

~~corresponding~~

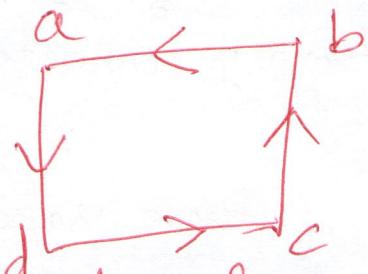
Directed graphs:

\hookrightarrow Strongly connected \Rightarrow There is a path b/w every ordered pair of vertices.

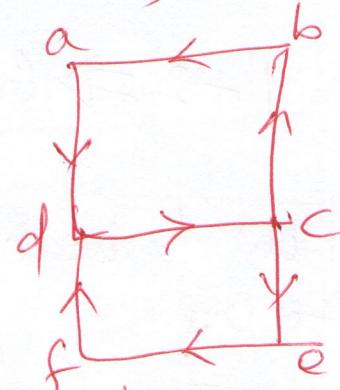
(means if a path is b/w (a,b) then there is another path (b,a) also.)



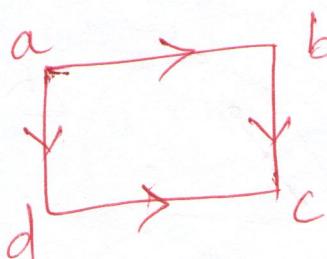
not a strongly connected graph



strongly connected graph



strongly connected graph.

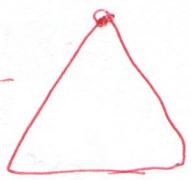
Weakly connected Graph

Not a weakly connected graph
(there no path from $b \rightarrow d$ & $d \rightarrow b$)

\Rightarrow A graph is weakly connected, if for every pair of vertices (u,v) there is a path from u to v or a path from v to u (or both).

Given a graph G , is G strongly connected?

Do DFS for each vertex.



If all these DFS visit all nodes every time,

then graph is strongly connected.

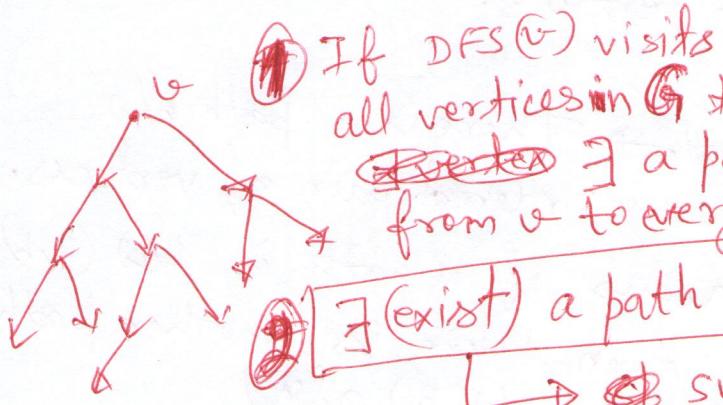
\Rightarrow Complexity $O(mn)$

Costly procedure

\Rightarrow We want to do in linear time $O(mn)$.

⑭ Applications of DFS in connected Graphs

⇒ Is a given graph G strongly connected?



There should be a path between every pair of ordered vertices.

exists a path from every vertex in G to v .

exists a path from every vertex in G to v .

suppose this is true.

It means what?

$1+2 \Rightarrow G$ is strongly connected.

$$x, y \Rightarrow x \xrightarrow{(2)} v \xrightarrow{(1)} y$$

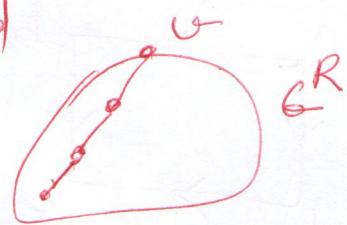
How do we check if there is a path from every vertex to v ?

Complexity:
 $= O(DFS)$

$G \xrightarrow[\text{edges}]{\text{reverse}} G^R \rightarrow$ Do DFS(v) on G^R

In G , there is
a path from
every vertex to v

If all ~~visited as~~ vertices
are visited



⇒ Equivalent to DFS in G by considering in-adjacent vertices instead of out-adjacent vertices.

Procedure

Pick an arbitrary vertex v

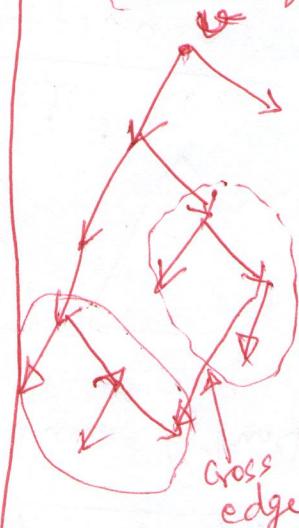
Do DFS(v) on G

Reverse(G)

Do DFS(v) on G^R

if all vertices are visited in both DFS's then G is strongly connected else G is not strongly connected.

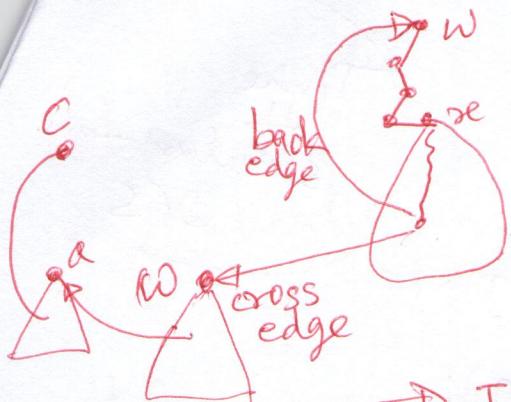
Solving in single DFS



It is better to have the edges outgoing from subtrees.

It is necessary that an edge go out of every subtree.

We want to find a path from every vertex to the root.

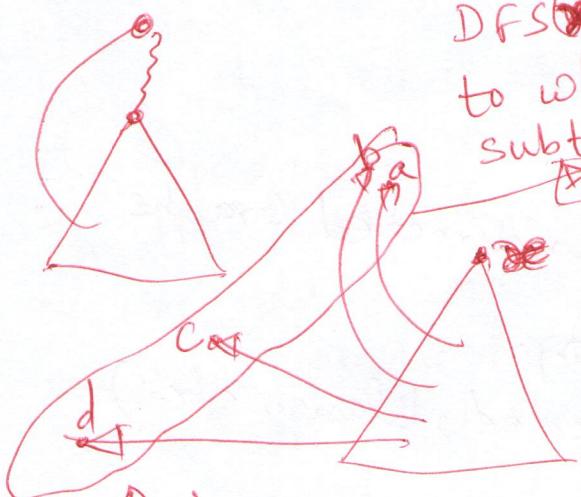


$\Rightarrow \text{arr}[x] > \text{arr}(w) > \text{arr}(a) > \text{arr}(c)$

\Rightarrow Thus, by following the combination of back edges and cross edges, If we reach to root v , it means there is a path from ~~node~~ to v .

\Rightarrow It means, it is necessary and sufficient, that an edge go out of every subtree rooted at any vertex x .

Q How should we modify DFS so that we can check if there is an edge going out of every subtree?

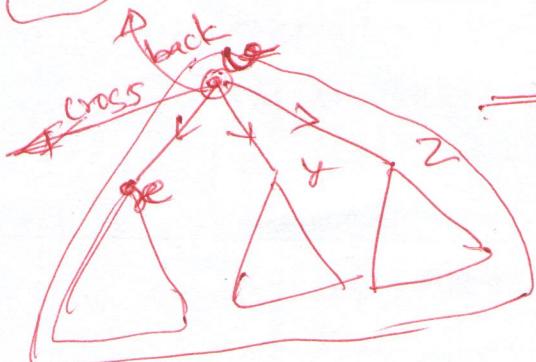


DFS(x) returns the smallest arrival time to which there is an edge from the subtree rooted at (x) .

\Rightarrow smallest arrival time of all these

DFS(x) returns

$\min(\text{arr}[a], \text{arr}[b], \text{arr}[c], \text{arr}[d])$



$\Rightarrow \text{DFS}(v)$

\Downarrow

min of

all outgoing edge from

$v, \text{DFS}(x), \text{DFS}(y)$
and $\text{DFS}(z)$

If any edge is going out of subtree rooted at v , then arrival time of destination of that edge is less than arrival time of v .

\Downarrow
given that $\text{DFS}(x), \text{DFS}(y)$ and $\text{DFS}(z)$ should also go out of subtree rooted at v .

(16)

 $SC(v) \{$ $arr[v] = \text{time}++$ $\alpha_{yz} = arr[v]$ $\text{visited}[v] = 1$ for all w outadjacent to v doif $\text{!visited}[w]$ then $\alpha_{yz} = \min(\alpha_{yz}, SC[w]);$ else $\alpha_{yz} = \min(\alpha_{yz}, arr[w]);$ needs to
be checked
for rootif $\alpha_{yz} = arr[v]$ then STOP (not strongly connected)Time
similar to
DFS

Applications of DFS in Directed Graphs

- Strongly connected
- Acyclic
- Topological sort

Applications of DFS in Undirected Graphs

- 2-edge connectivity.
- others (vertex connected, planar etc.)

Applications of BFS

- Connected components
- Bipartite