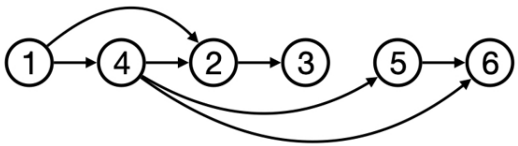
COL 351:
Analysis and Design of Algorithms

Lecture 8

DFS in Directed Graphs

- Topological sort in DAGs (directed acyclic graphs)



Topological sort

- Unique Path Graph

(Checking if $\forall x, y$, there is unique $x \rightsquigarrow y$ path in G.)

- Finding SCCs

DFS Algorithm

Preprocessing :

For each $v \in V(G)$:

Set VISITED(v) = False

count = 1

DFS(x)

1. ST(x) = count ++
2. Set VISITED(x) = True
3. For each $y \in N(x)$:

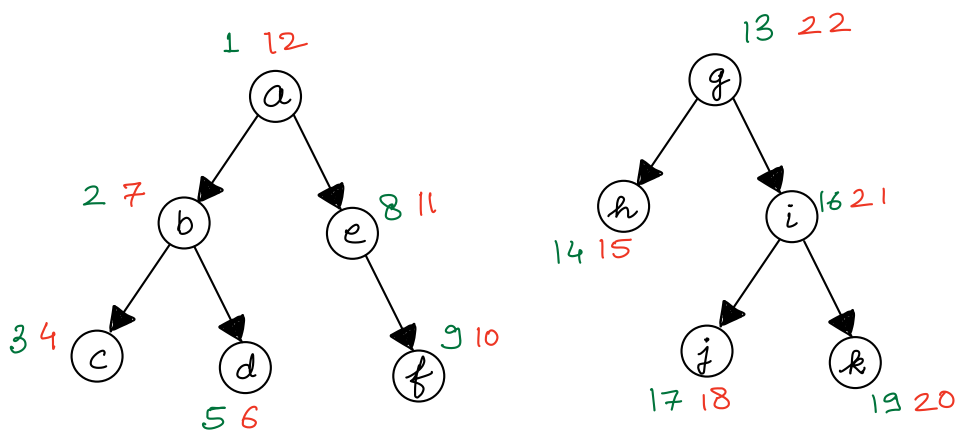
   If VISITED(y) = False :

   DFS(y)

4. FT(x) = count ++

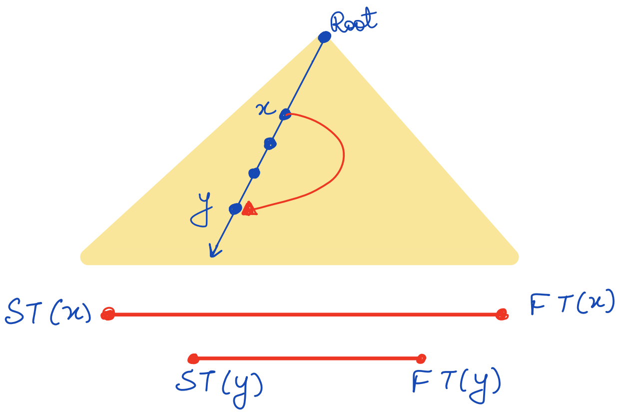
ST : Start Time

FT : Finish Time

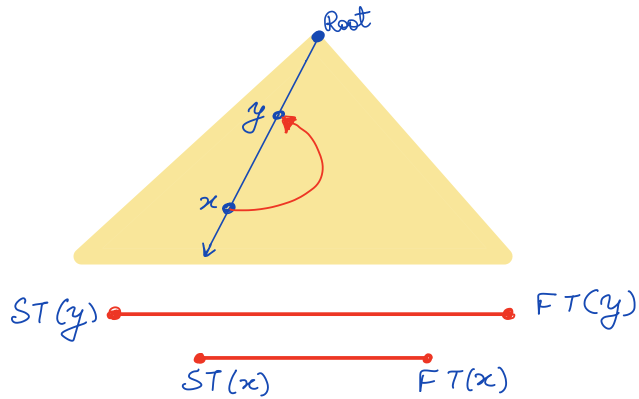


Classification of non-tree edges wrt DFS tree

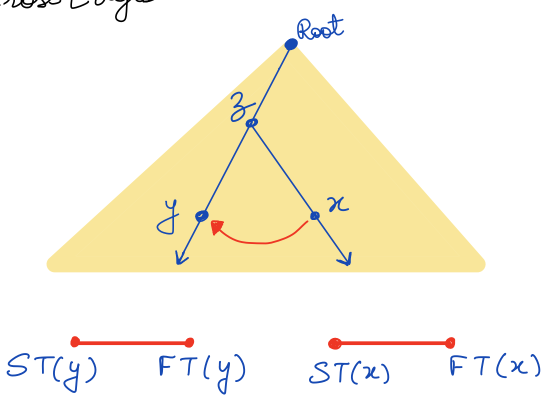
- Forward Edges



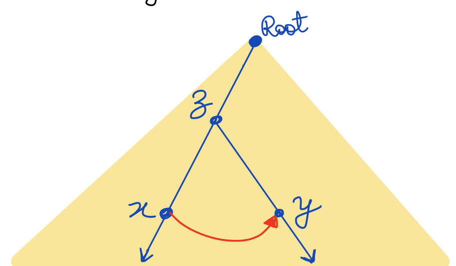
- Back Edges



- Cross Edges



- Anti cross edges



Not possible in DFS tree

Topological Sort in DAGs

Lemma: For any edge (x, y) in a DAG, we have FT(y) < FT(x).

Proof: By discussion on previous slide, we have:

- For any tree / fwd edge (x,y): $ST(x) < ST(y) < FT(y) < FT(x)$
- For any cross edge (x,y): $ST(y) < FT(y) < ST(x) < FT(x)$

In both cases $FT(y) < FT(x)$

Theorem: Vertices arranged in decreasing order of their finish time during DFS is a topological ordering of G.

Proof: Let $v_1 \ldots v_n$ be such that $FT(v_1) > FT(v_2) > \cdots > FT(v_n)$.

For any edge (v_i, v_j) we have $FT(v_j) < FT(v_i)$

$\Rightarrow i < j$

$\Rightarrow$ All edges are from Left to Right.

Topological Sort in DAGs

Algorithm

1. Perform DFS traversal on G

2. Sort vertices $v_1 \ldots v_n$ such that

$$FT(v_1) > FT(v_2) > \cdots > FT(v_n)$$

3. Return $(v_1 \ldots v_n)$

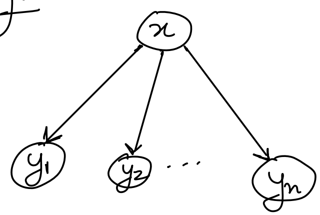
Time complexity $= \underbrace{O(m+n)}_{\text{For step 1}} + \underbrace{O(n)}_{\text{For Bucket sort}} = O(m+n)$

Unique Path graph

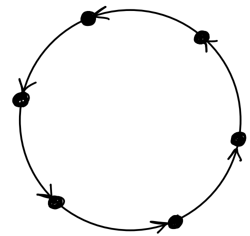
Definition: A directed graph G is said to be a unique path graph if for each pair (x,y), we have:

if there there is an x->y path in G then there is a unique path from x to y in G.

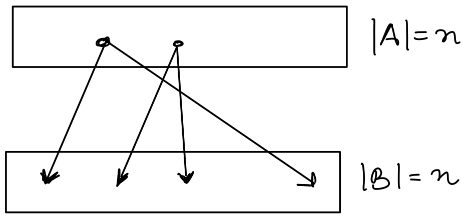
Eg.



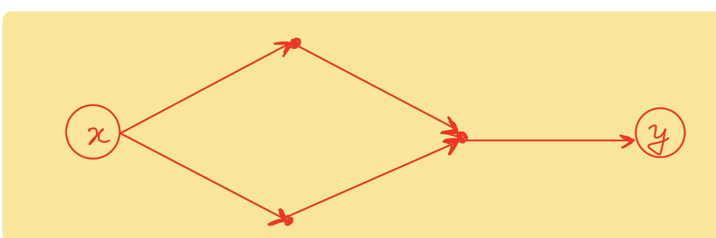
Star



Cycle



$K_{n,n}$



Not a unique path graph

Unique Paths from source "x"

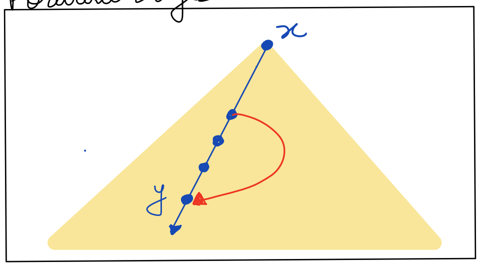
Simpler Question: Given a vertex x, how to check that for all y, there is a unique path from x to y?

UNIQUE(x)

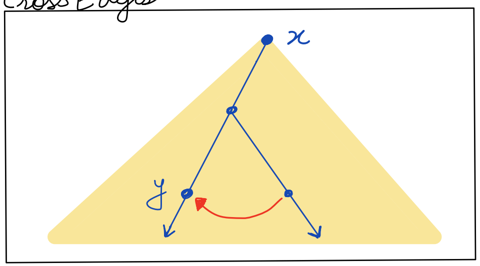
1. Compute DFS(x)
2. If you encounter Forward / Cross edge then return "False"
   Else return "True".

Proof:
Forward / cross edges results in Non-unique paths from x to some vertex y

Forward Edges



Cross Edges



Back edges in DFS(x) aren't problematic because any simple path starting from x can't contain back edges of DFS(x)

Time = $O(m+n)$

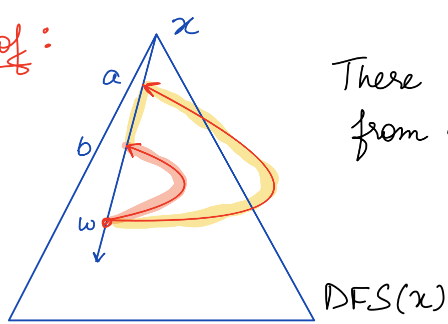
Unique Path graph

Simpler Question: Given a vertex x, how to check that for all y, there is a unique path from x to y?

Main Question: Is G a unique path graph?

Lemma If in DFS(x) we encounter 2 back edges from same vertex (say w), then G is not unique path graph for $\{w\} \times V(G)$.

Proof:



There are 2 paths from w to b.

Implication of Lemma:

- → Directly applying algorithm from previous slide to each x will take $O(m \cdot n)$ time
- → We can bring down "m" to $O(n)$ value. This is because we can abort DFS(x) if we find 2 back edges from same vertex
- → This modified DFS will take $O(n)$ time per vertex.

Unique Path graph

Algorithm Implementation

① For each x, we compute DFS with x as root.

② While computing DFS(x) we keep track of count of non tree edges. If count $\geq n$, then we ABORT as it would imply:

- either a FWD edge
- or a CROSS edge
- or 2 back edges from same vertex

③ Now if we encountered less than n non-tree edges then time for DFS(x) is $O(n)$.

Moreover after computing DFS(x) we can check using ST/FT if we encountered FWD/cross edge.
If not, then Unique Path property is satisfied for $x \times V$.