

COL106: Data Structures and Algorithms

Lecture 30

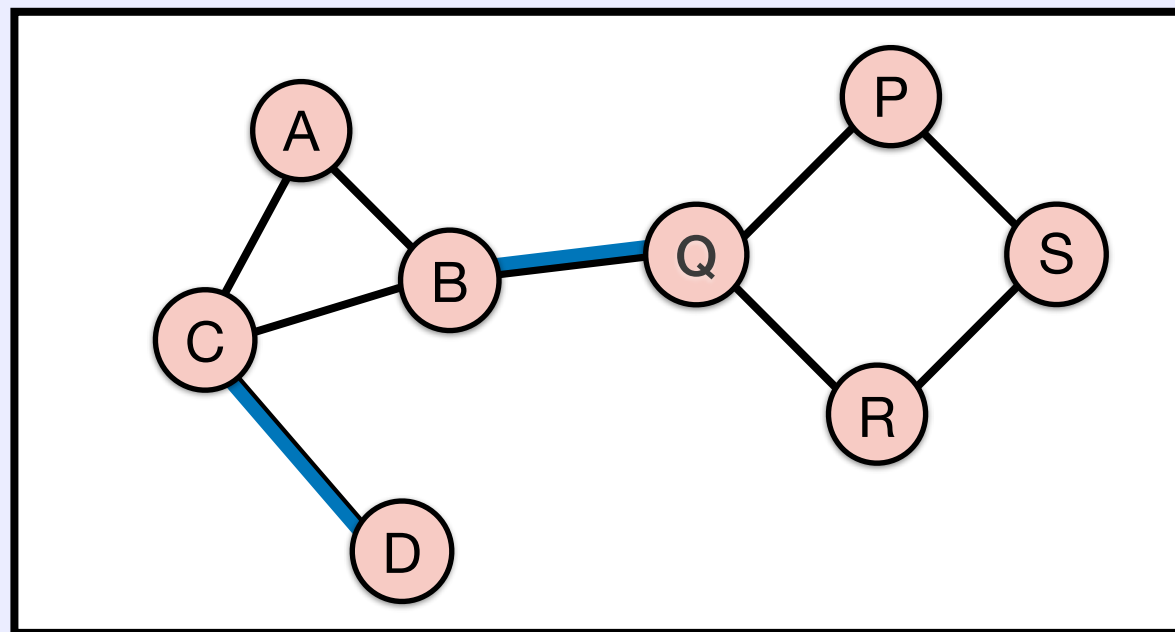
Keerti Choudhary
Department of Computer Science, IIT Delhi

Today's Lecture

1. Bridge Edges
2. DFS Trees

Bridge Edge

Definition: An edge (x, y) in an undirected graph is said to be a *bridge edge* if there is **no path** from x to y in $G - (x, y)$.



Trivial algorithm:

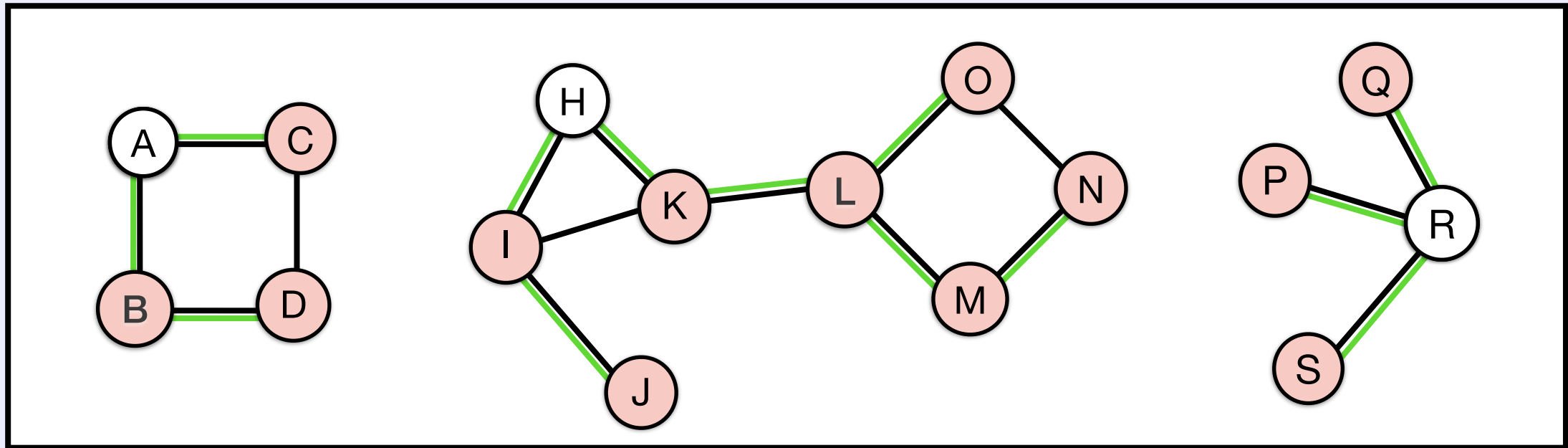
For **each** edge (x, y) in G , run $\text{BFS}(x)$ in $G - (x, y)$ to check if x and y are disconnected.

$$\text{Time} = O(m \cdot (m + n))$$

*How to improve
the running
time?*



Slightly improved algorithm for Bridge Edges



Algorithm:

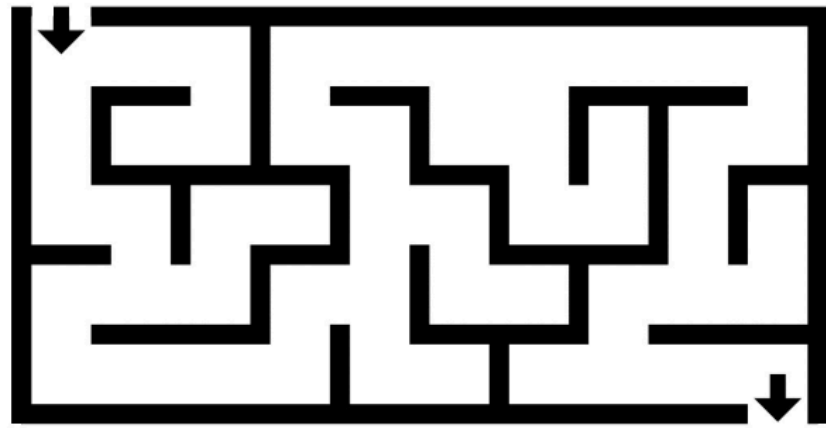
1. Find BFS trees $T_1, \dots, T_k$, of connected components in $O(m + n)$ time.
2. For each edge $(x, y) \in (T_1 \cup \dots \cup T_k)$, run $\text{BFS}(x)$ in graph $G - (x, y)$ to check if x and y are disconnected.

*How can you find
all bridge edges
in just
 $O(m + n)$ time?*

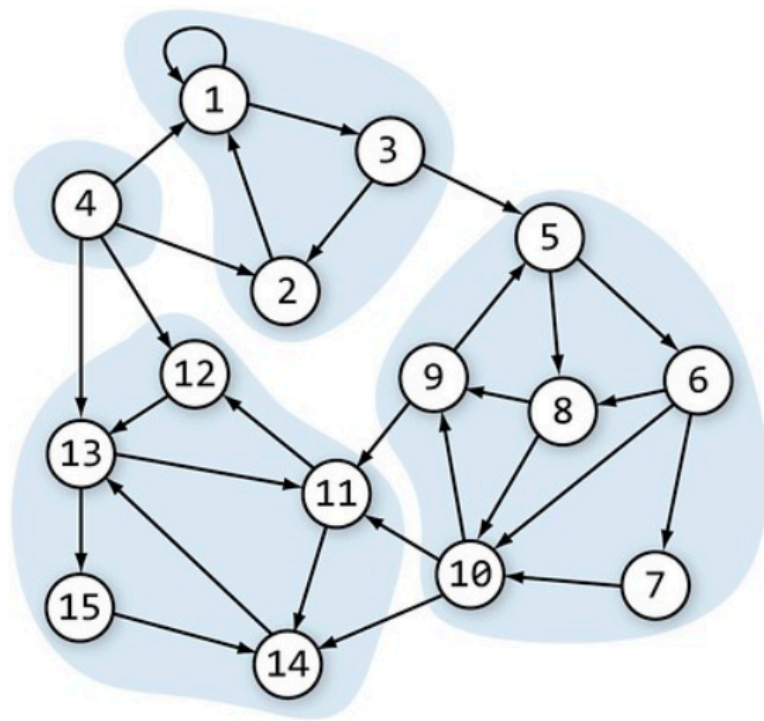
Look at new type of
trees: “DFS tree”

$$\text{Time} = O(n \cdot (m + n))$$

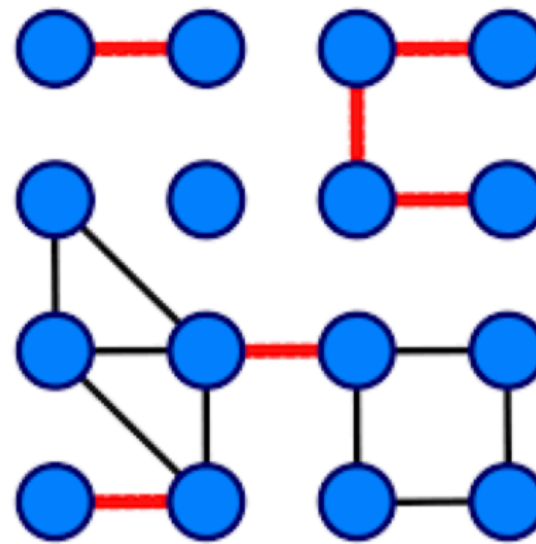
DFS Tree Applications



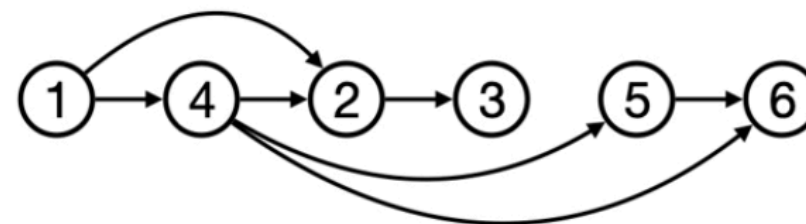
Solving Mazes



Strongly connected components



Bridge edges



Topological sort

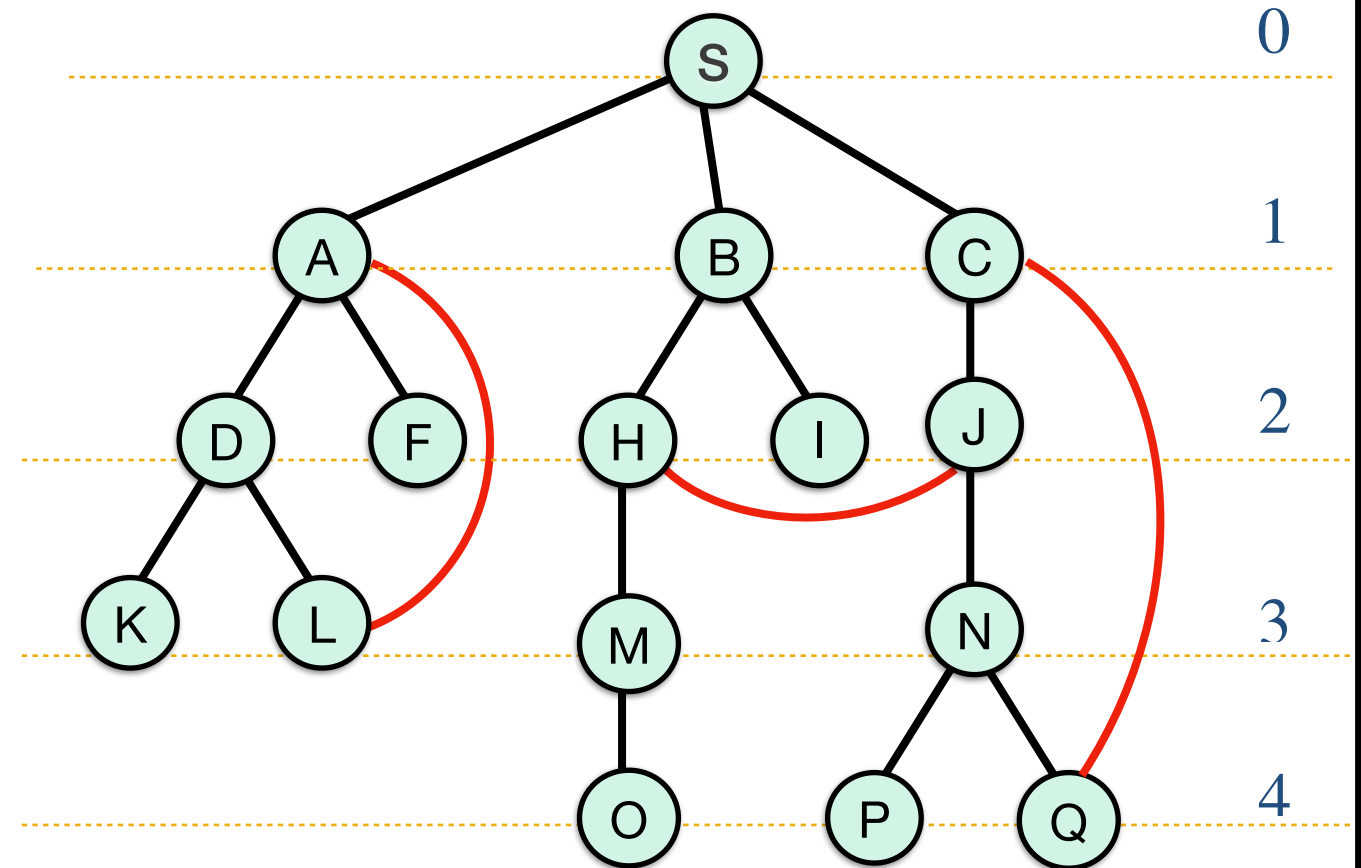
Applications of DFS tree include solving mazes, finding bridge edges, finding SCCs, finding topological ordering, etc.

New structure: DFS Tree

Definition (characterisation)

For an undirected connected graph, a rooted spanning tree is said to be DFS tree if

For each non-tree edge (a, b) ,
either
 a is ancestor of b , or vice-versa.



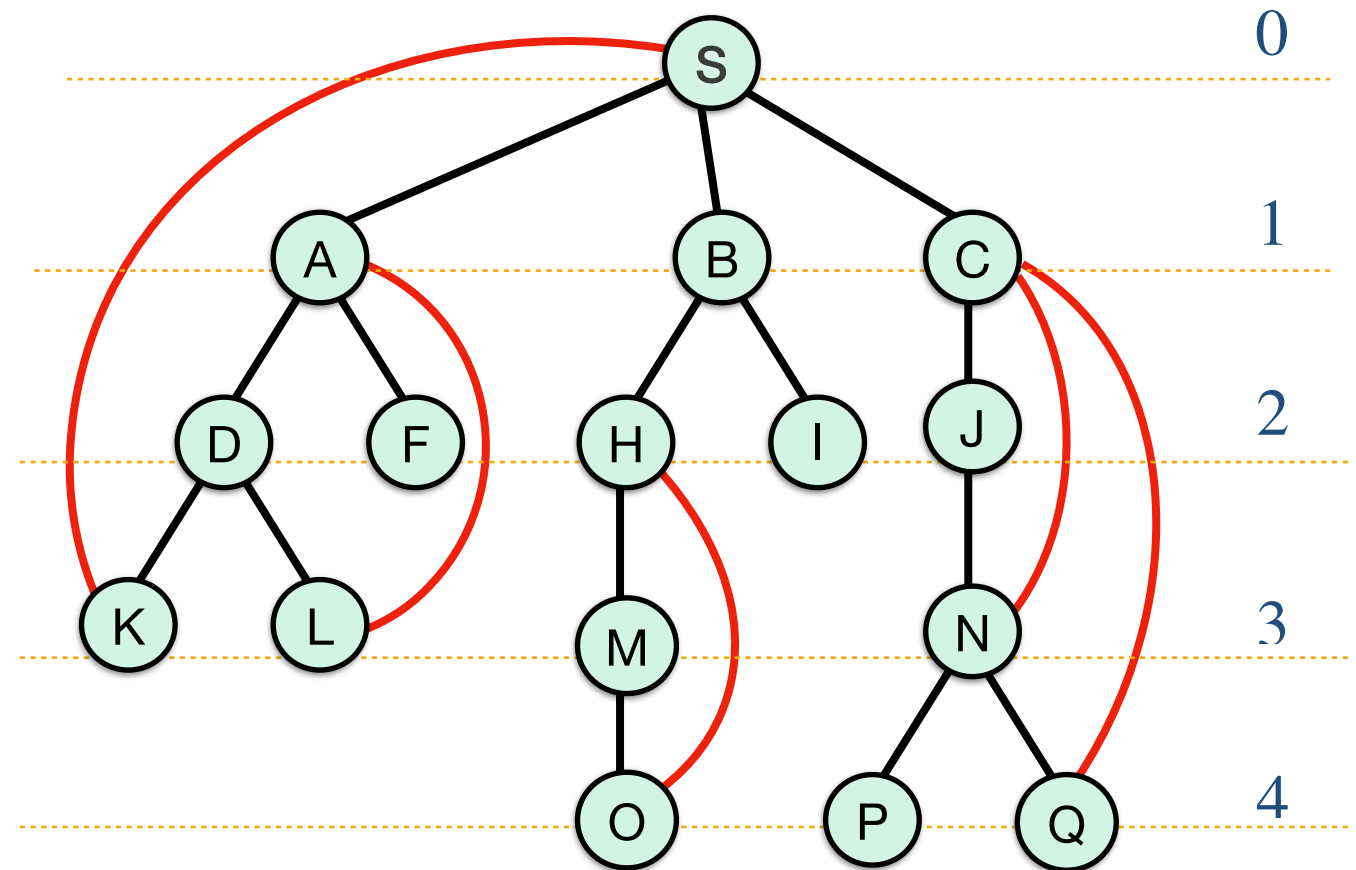
Not a DFS tree

Bridge Edges

Property

A tree edge $(par(x), x)$ is a bridge iff

there is no non-tree edge going out from subtree $T(x)$.



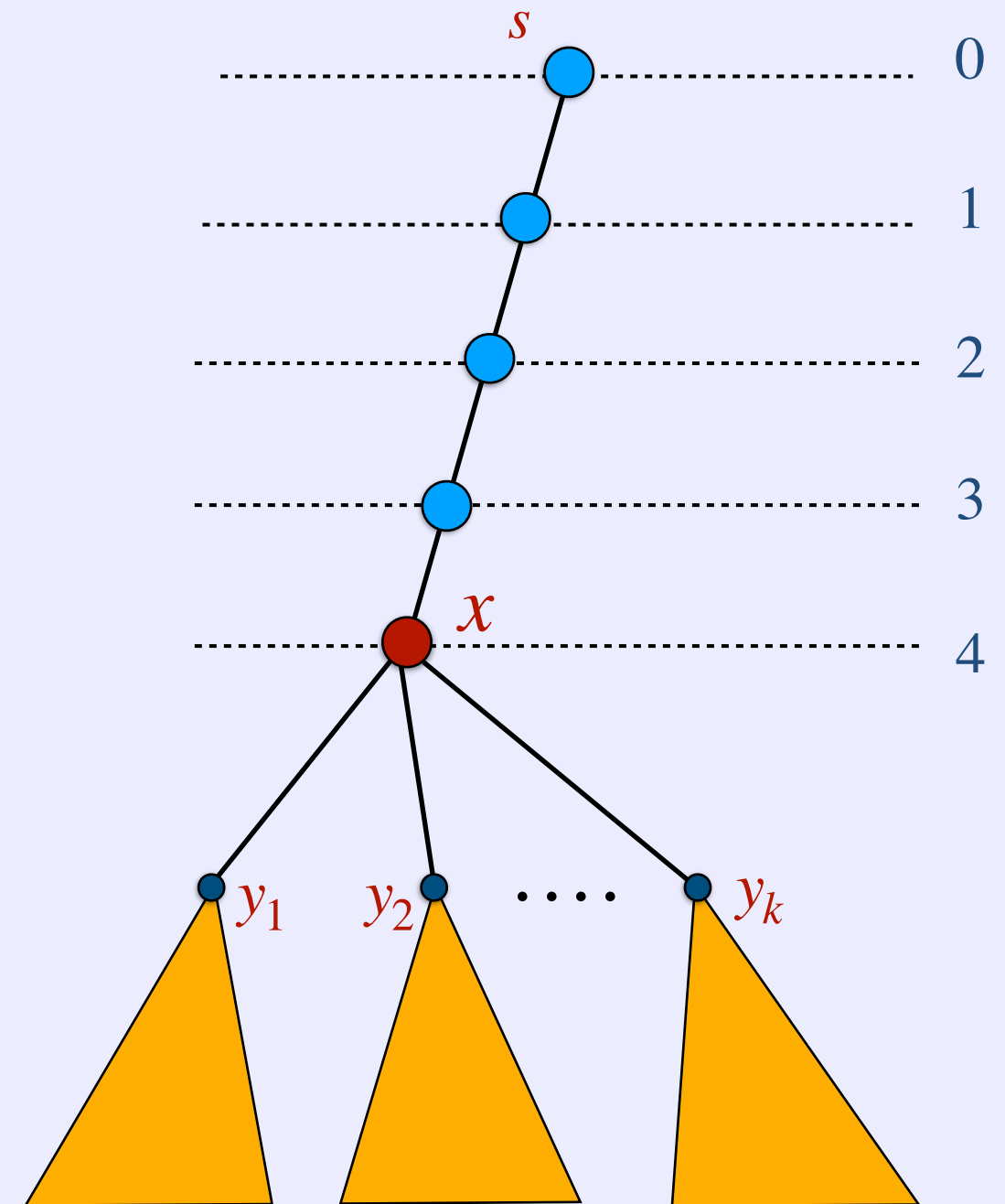
Bridge edges:

$(A, F), (S, B), (B, H), (B, I), (S, C), (N, P)$

High point (HP) of a vertex

High-point(x):

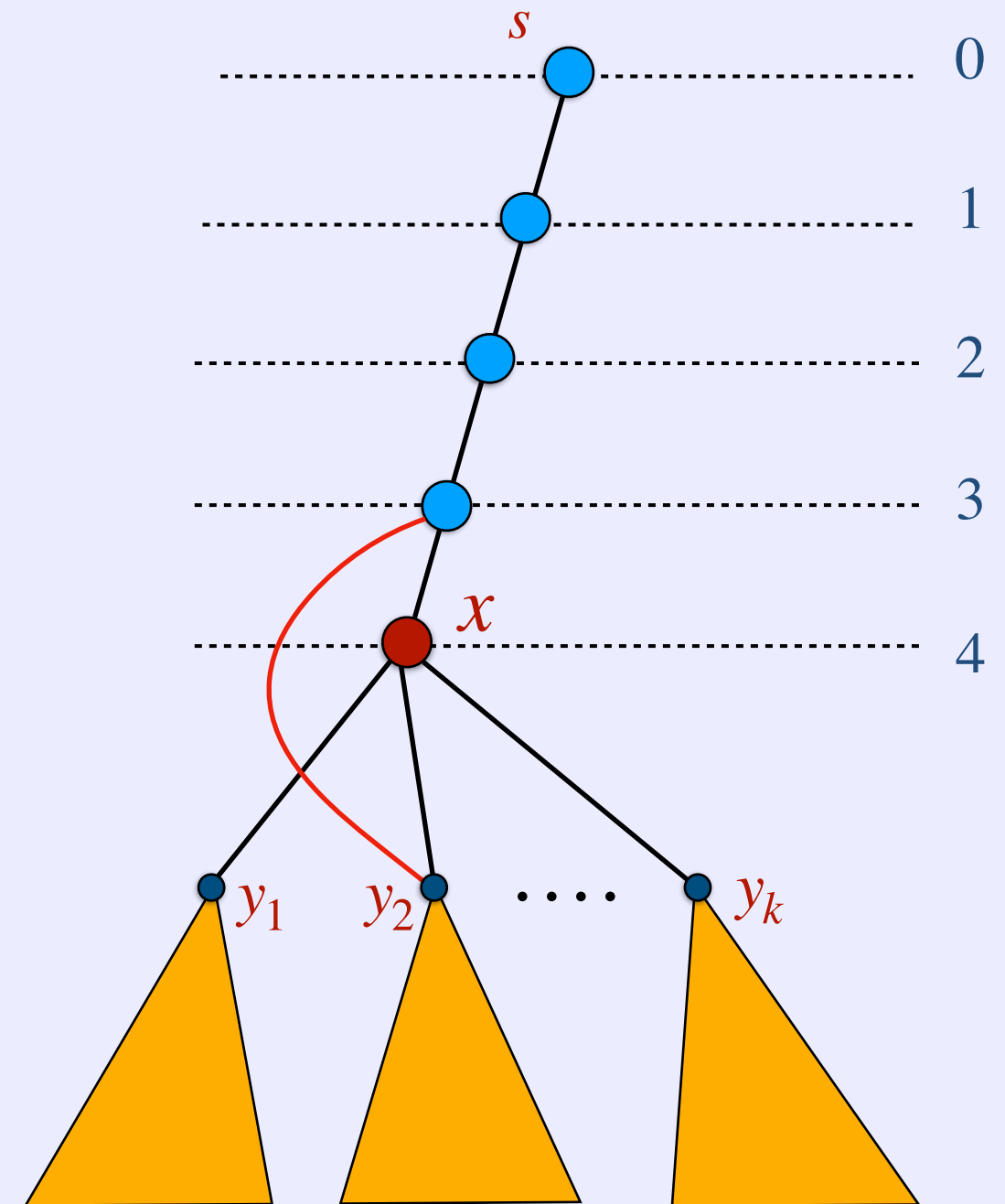
The level of the highest ancestor of x to which there is a non-tree edge from subtree $T(x)$.



High point (HP) of a vertex

High-point(x):

The level of the highest ancestor of x to which there is a non-tree edge from subtree $T(x)$.

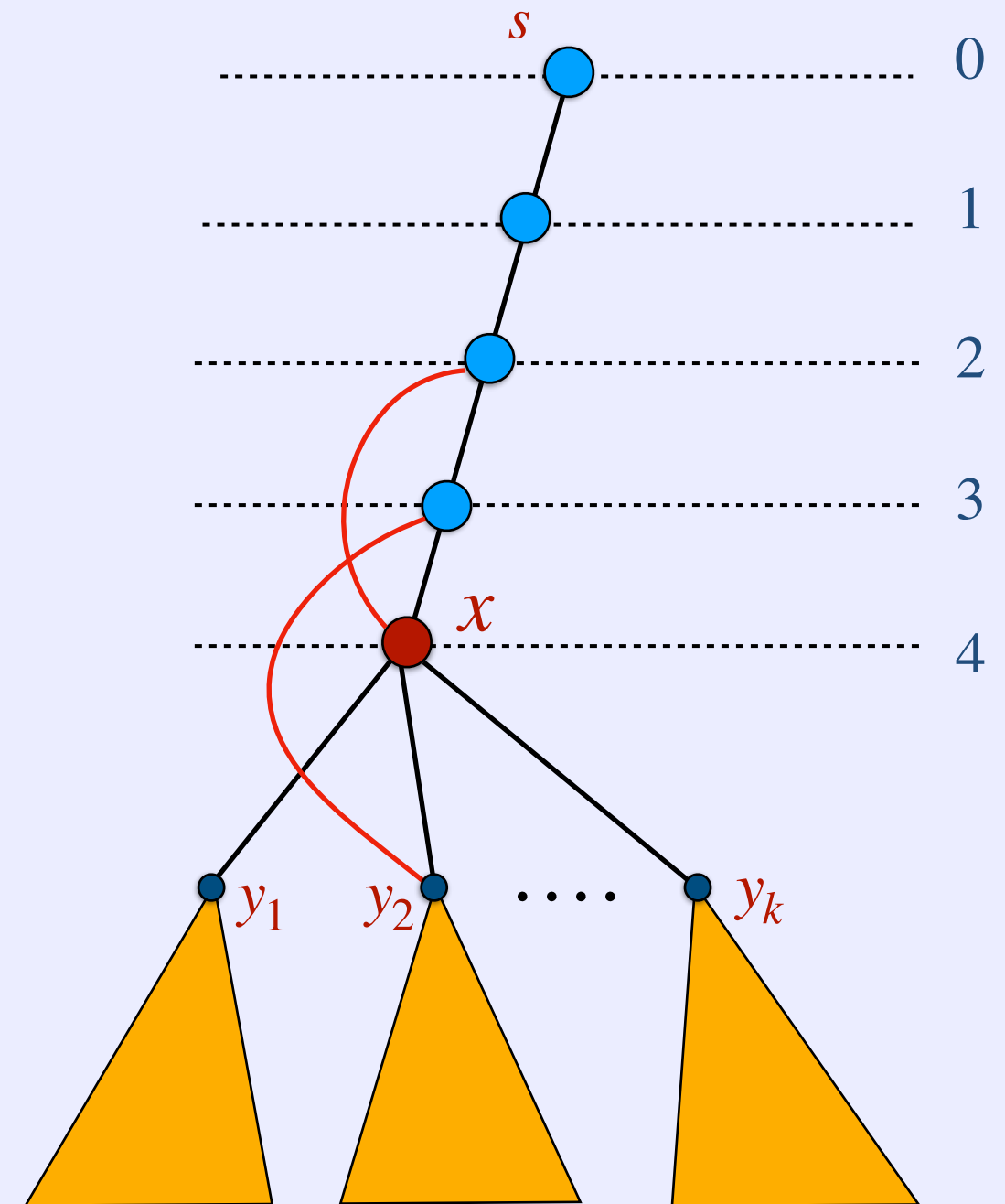


$$\text{HP}(x) = 3$$

High point (HP) of a vertex

High-point(x):

The level of the highest ancestor of x to which there is a non-tree edge from subtree $T(x)$.

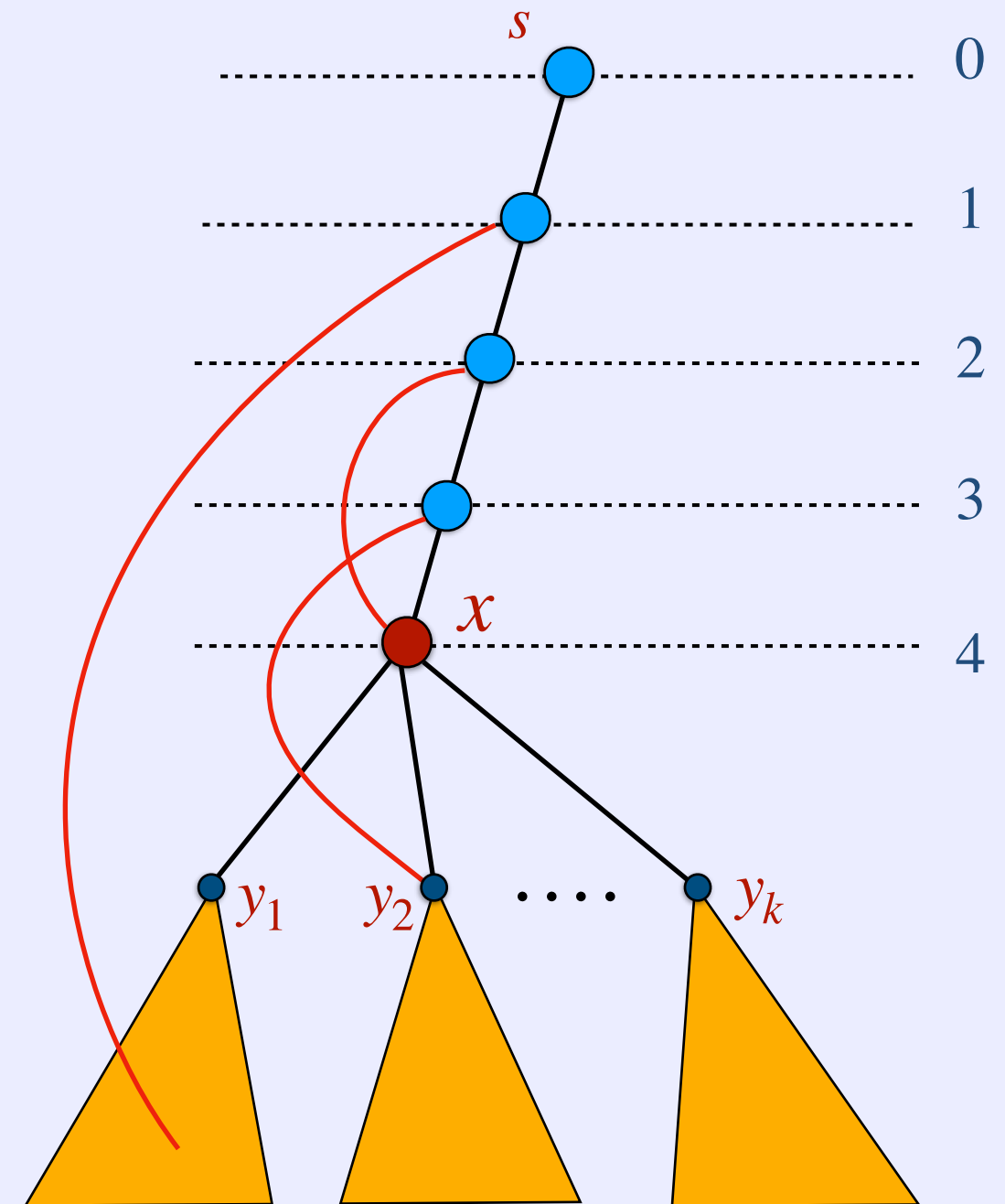


$$\text{HP}(x) = 2$$

High point (HP) of a vertex

High-point(x):

The level of the highest ancestor of x to which there is a non-tree edge from subtree $T(x)$.



$$\text{HP}(x) = 1$$

High point (HP) of a vertex

High-point(x):

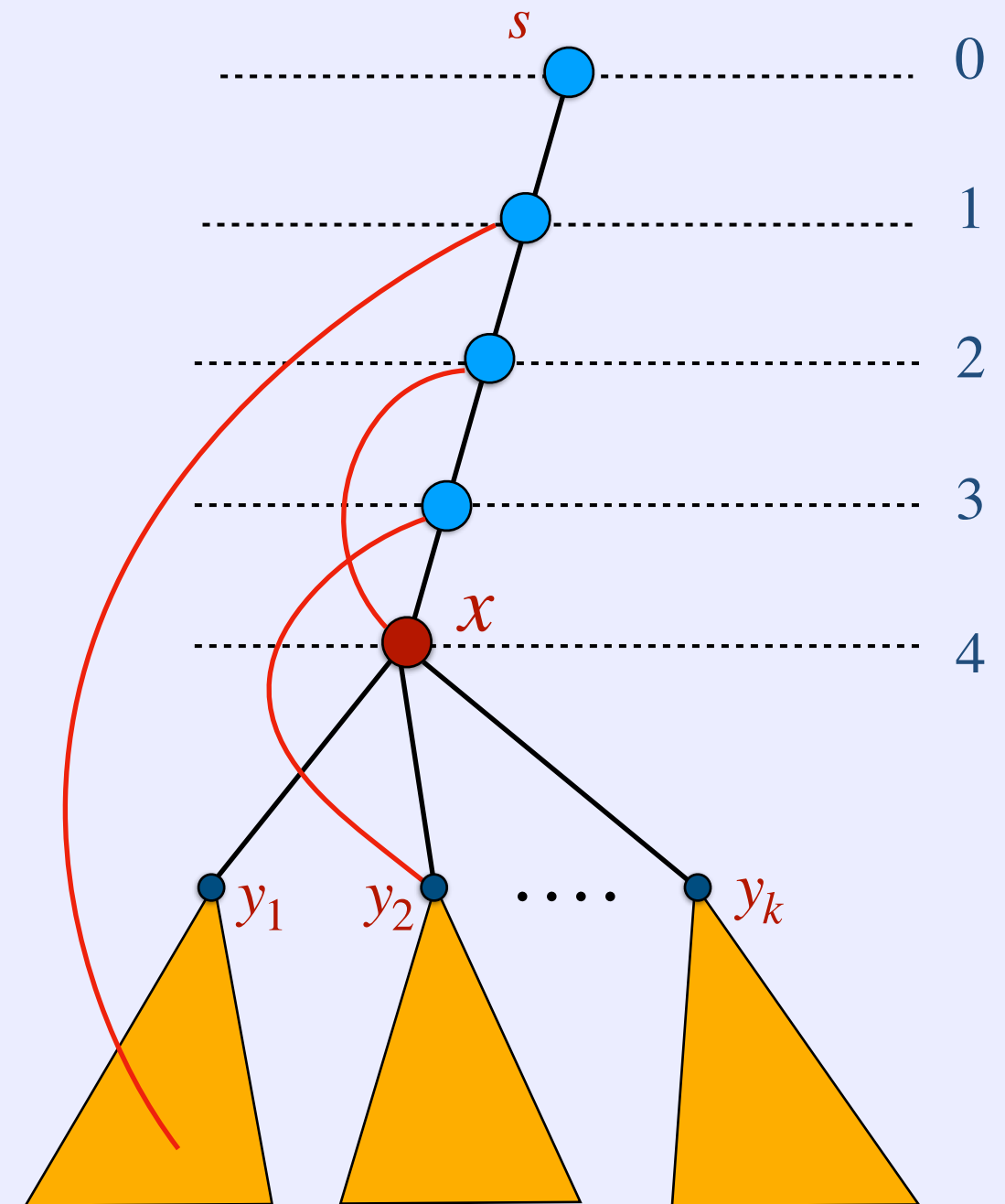
The level of the highest ancestor of x to which there is a non-tree edge from subtree $T(x)$.

Question:

If $y_1, \dots, y_k$ are children of x , then how to compute $HP(x)$ from $HP(y_1), \dots, HP(y_k)$?

$$HP(x) = \min \left(\min_{i \in [1, k]} HP(y_i), \min_{z \in N(x) \setminus \text{par}(x)} \text{Level}(z) \right)$$

- Time to process $x = O(\deg(x))$
- Time to process all vertices in bottom-up manner = $O(m + n)$



$$HP(x) = 1$$

High point (HP) of a vertex

High-point(x):

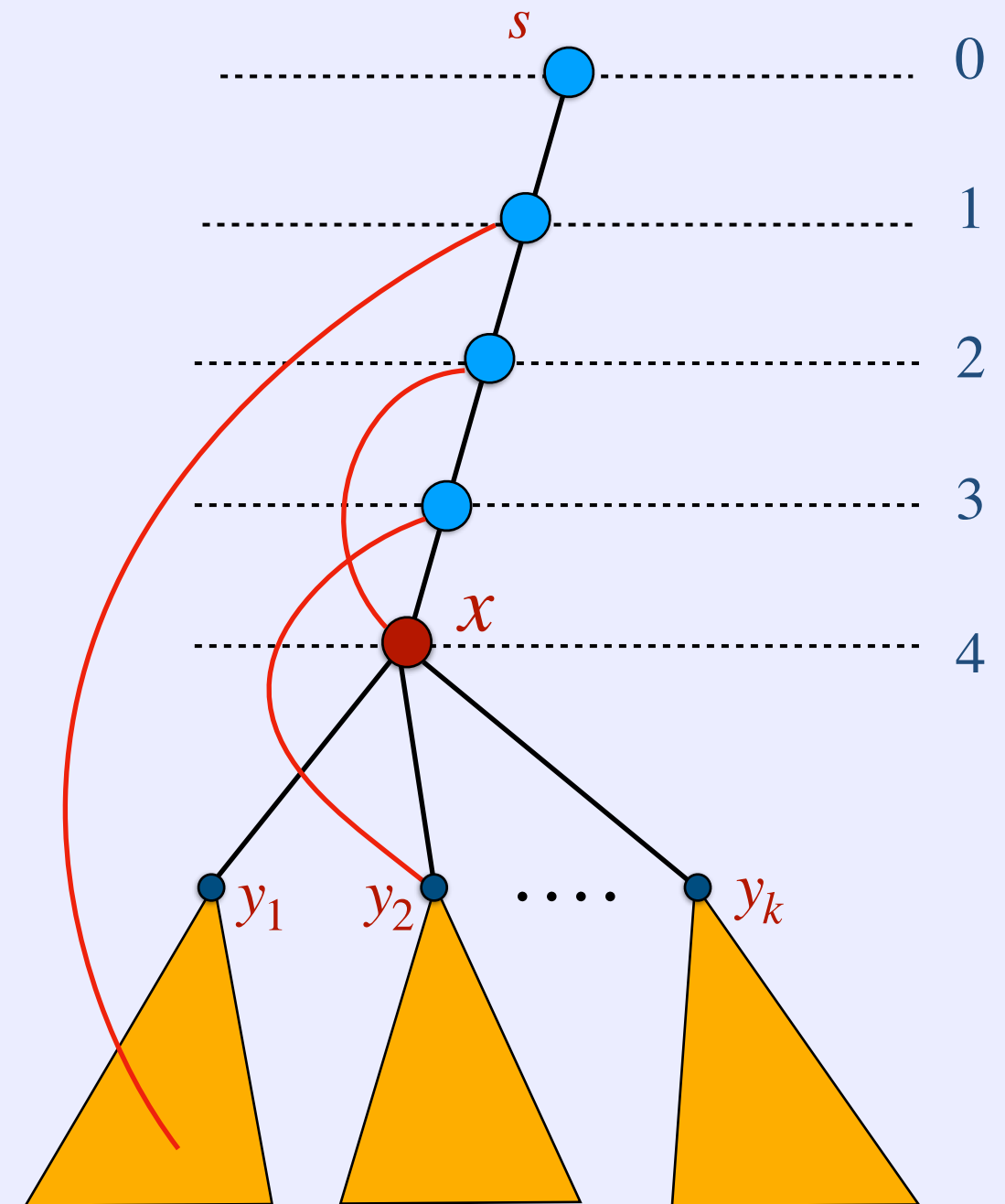
The level of the highest ancestor of x to which there is a non-tree edge from subtree $T(x)$.

Lemma 1

A tree edge $(par(x), x)$ is a bridge iff $HP(x) \geq Level(x)$.

Lemma 2

Given the DFS tree one can compute all bridge edges in $O(m + n)$ time by processing tree in bottom-up manner.



$$HP(x) = 1$$

Homework Question

Problem: Design a recursive algorithm to compute a rooted tree (i.e. DFS tree) of a graph with following property.

For each non-tree edge (a, b) ,
either
 a is ancestor of b , or vice-versa.

