COL 351: Analysis and Design of Algorithms

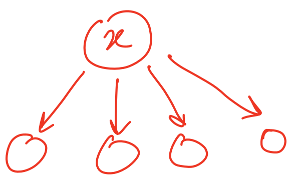
Lecture 7

Graph Traversals

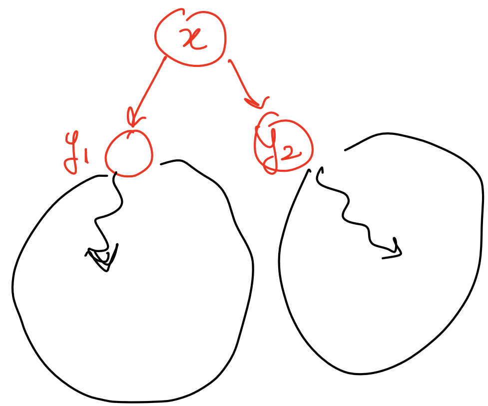
Process of visiting vertices of a graph (directed/undirected).

Standard Types
- BFS (Breadth First Search) — $O(m+n)$ Queues.
- DFS (Depth First Search) — $O(m+n)$ Stacks

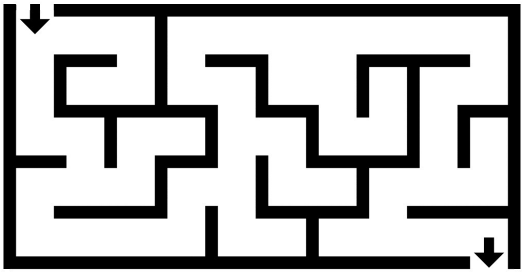
BFS



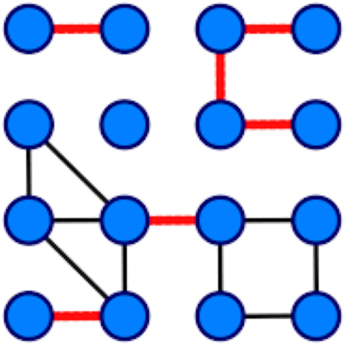
DFS



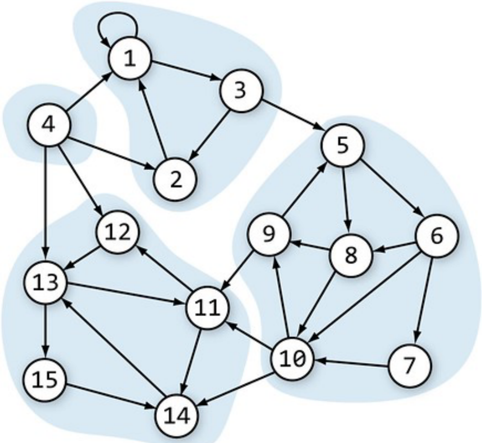
DFS Applications



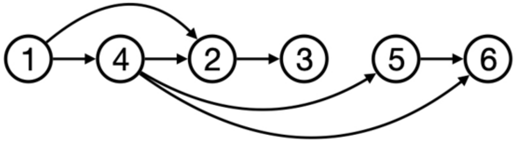
Solving Mazes



Bridge edges



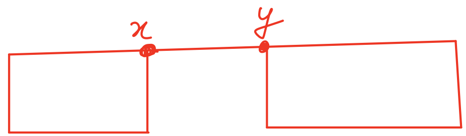
Strongly connected components

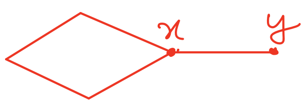


Topological sort

Bridge Edges

Defⁿ: An edge (x, y) is Bridge-edge if there is NO path from x to y in $G - (x,y)$.

Eg 1: 

Eg 2: 

Goal: Compute ALL bridges of G

$O(m^2)$ – Naive

$O(m \cdot n)$ – Easy

Aim: $O(m+n)$ time

Sketch of $O(m \cdot n)$ time algo:

1. Compute a spanning forest of G in $O(m+n)$ time.
2. Let X = set of edges in spanning forest
3. $Y = \emptyset$
4. For $e = (a,b) \in X$:

   Add e to Y iff there is no $a \sim b$ path in $G \setminus e$
5. Return Y

Time $= O(m|X|) = O(m \cdot n)$

H.W.: Prove correctness

DFS Traversal

Preprocessing :

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

Set VISITED(v) = False

DFS(x)

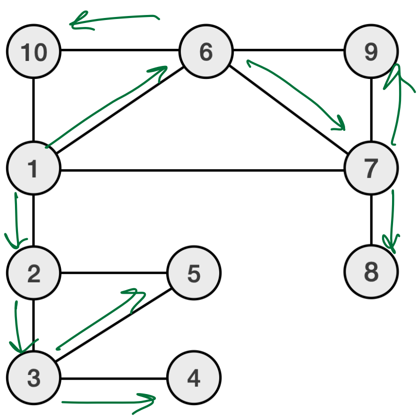
1. Set VISITED(x) = True
2. For each $y \in N(x)$:

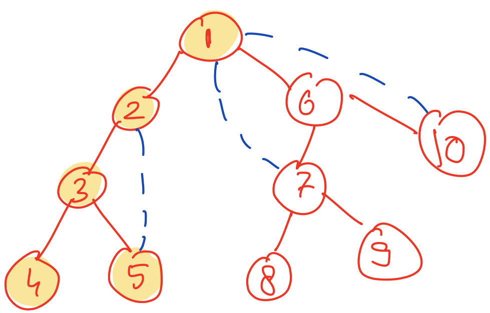
   If VISITED(y) = False :

   DFS(y)

   PARENT$(y) = x$

   LEVEL$(y) = 1 +$ LEVEL(x)



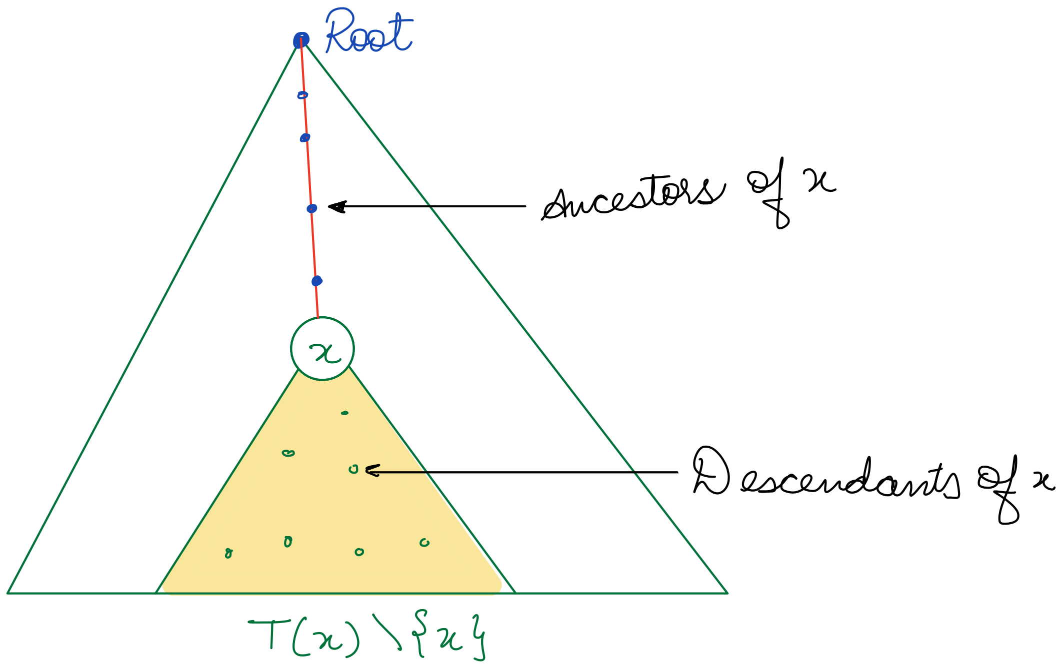


Edges in - - - (non-tree edges)

have ancestor-descendant relationship

Property

Lemma: Let G be a undirected, connected graph and $T = DFS(G)$.
Then for any edge (x,y) in G, x & y have ANCESTOR-DESCENDANT relationship in T



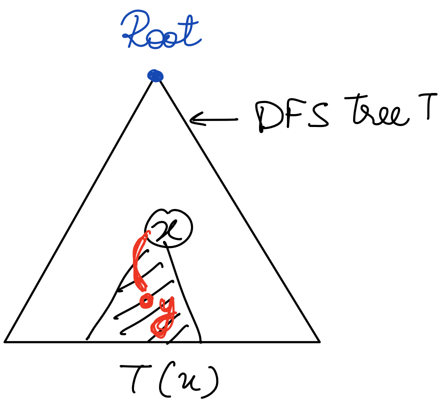
Lemma: Let G be a undirected connected graph and $T = DFS(G)$. Then for any edge (x, y) in G, x & y have ANCESTOR-DESCENDANT relationship in T

Proof Sketch:

Suppose x is visited before y. We need to show $y \in T(x)$.

Claim 1: vertices in $T(x)$ $\equiv$ vertices visited in $DFS(x)$

Claim 2: $DFS(x)$ should visit y.

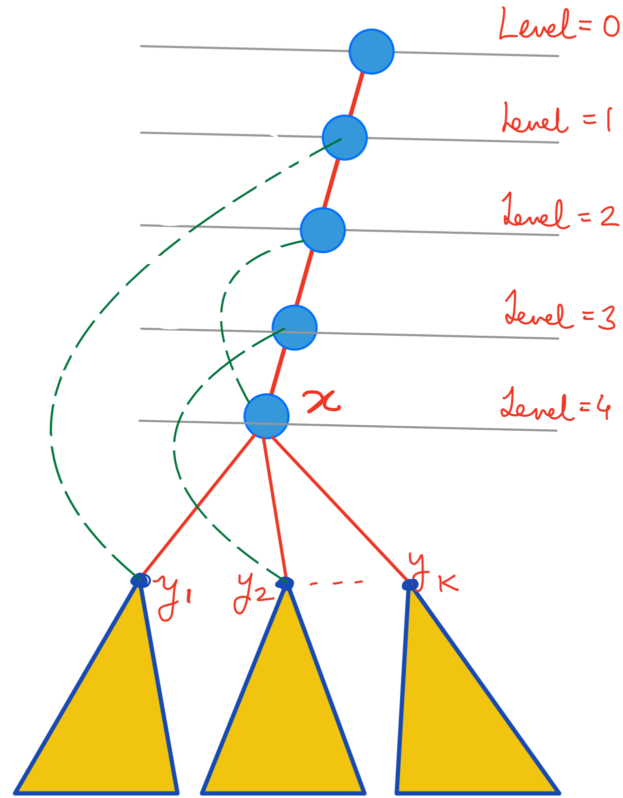


Claim 1 & claim 2 $\Rightarrow$ $y \in T(x)$

High Point 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).

How to compute High-point for all vertices of G?



$$H.P.(x) = \min\left(\min_{y_i = child(x)} H.P.(y_i) \, , \, \min_{z \in N(x)} Level(z) \right)$$

\# Time to process one vertex = O(degree)

In above example HP(x) = 1

\# Time to compute high point of ALL vertices in Bottom-up manner = O(m+n)

Theorem

Theorem: A tree edge (x, y), with x being parent of y in DFS tree, is a **bridge** edge iff

$$\text{High-point}(y) \geq \text{Level}(y).$$

Proof: If $HP(y) \geq \text{Level}(y)$ $\iff$ $\exists$ no edge from subtree $T(y)$ to ancestors of y, other than (x,y)

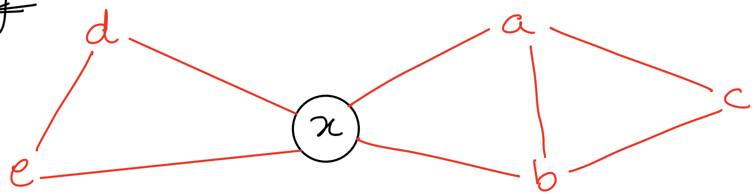
(Reasoning: All non-tree edges of DFS have ancestor descendant relationship)

$\iff$ (x,y) is a cut-edge.

CHALLENGE PROBLEM

Definition(Articulation point):
A vertex x is said to be articulation point if there are u,v different from x, such that u and v are disconnected in G\x.

Eg.



"x" is a cut-vertex in G
as c,d are disconnected in G\x

Exercise: Design an O(m+n) time algorithm to find all the articulation points of a graph.