LECTURE #11

Elementary Graph Algorithms:

Search → Explore graph by traveling over the edges to discover the nodes.

Why? → Discover the graphs structure

Tonight → BFS & DFS

Breadth-First Search

L Named that bc it expands its fronteer of discovered nodes uniformly

- Given a graph G, & source vertex $S \in V$

  Explore the edges of G to discover every node reachable from S

  (if the graph is connected, then this will discover every other node)

  → Computes distance (# of edges)

  → Generates a BFS-Tree that stores shortest path at any node from S

  (works on directed or undirected graphs, <u>NO WEIGHTS.</u>)

- Color each node white, grey, or Black — <u>initially all white</u>

  Undiscovered (white), In process (grey), complete (Black)

- A vertex is discovered a first time it is encountered & becomes non-white

  (Black = itself & all adjacent nodes have been discovered.)

- BFS constructs a BFS-Tree, initially only contains the source nodes.

- BFS-Tree:

Whenever a white vertex v is discovered, the vertex v & edge (u, v) are added to the tree.

u is a parent of v.

- To implement BFS:

  need to store

  - For each vertex
    - Color
    - Parent (AKA predecessor node)
    - Distance (from s)
  - Queue. $\boxed{x}$ $\boxed{x+1}$

RUN TIME:

$O(V+E)$

"start"

```
BFS ( graph G,  node s) {
    for each vertex u ∈ V - {s} (or all nodes) {     O(V)
        color[u] = white
        p[u] = Ø         // parent[u] = null
        δ[u] = ∞         // distance(u) = flag value
    }
    color[s] = gray
    δ[s] = 0
    p[s] = Ø
    Q.insert(s)

    While ( ! Q.isEmpty()) {
        t = Q.dequeue()          // ?
        for each node v ∈ Adj[u] {
            if (color[v] == white) {
                color[v] = grey;
                p[v] = u;
                δ[v] = δ[u] + 1;
                Q.insert(v);
            } // if
```

Init

Init Source

```
        } // for
        color[U] = Black;
    } // while
} // BFS
```

EX:

A = start Node

δ[ ]
color[ ]

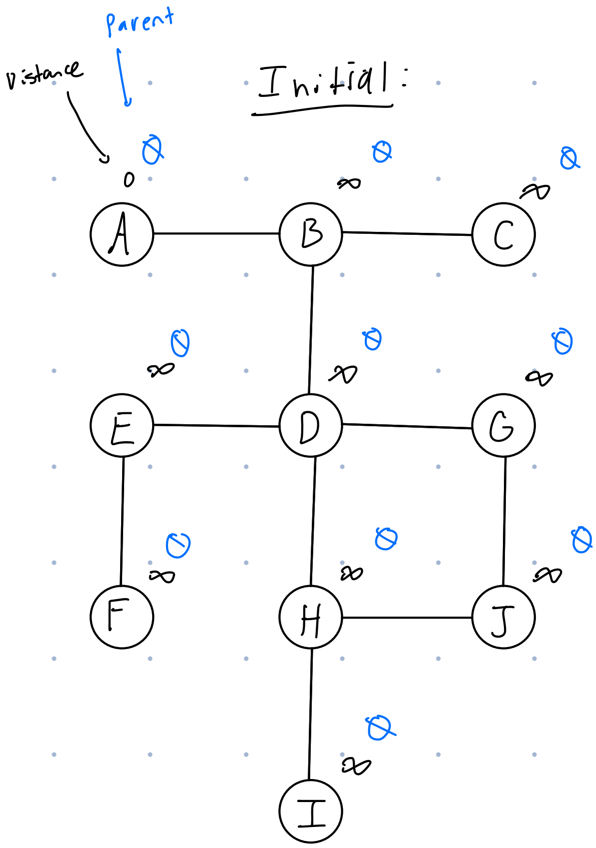
on graph

Highlight edges to show parents

Q : Empty when done

U over time :

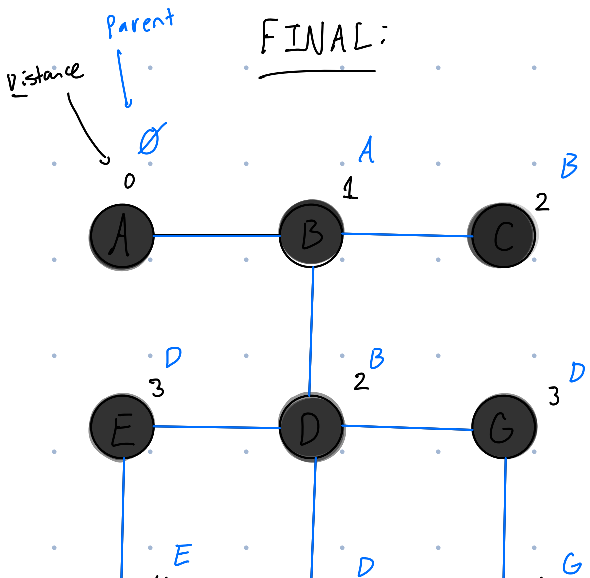
· · · · ·



A = start Node

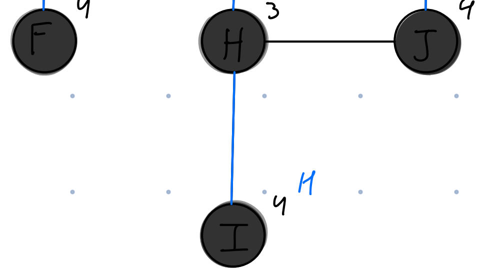
δ[ ]
color[ ]

on graph



Highlight edges to show

parents



Q : Empty when done

V over time :

V = A Adj = B

V = B Adj = A, C, D

V = C Adj = B

V = D Adj = B, E, G, H

V = E Adj = D, F

V = G Adj = D, J

V = H Adj = D, J, I

We have discovered the optimal route to each node given starting point A!

Depth - First Search

- Edges are are explored out of the most recently discovered node that still has unexplored edges
- When all of a nodes edges have been explored, backtrack to a predecessor

White gray black

each node is timestamped (NO DUPLICATE TIMES)

with 1 & 2 |V|

- both δ - discovery time

  F - finish time

  δ[v] < F[v]

  p - parent

Whole Algorithm :

O(V + E)

DFS ( ) {

for each vertex in U {

color [u] = white ;

p[u] = ∅

```
    p[U] = ∅
    }

    time = 0
    for each vertex in U {
        if (color[U] == white) {
            DFS - visit(U);
        } // if color
    } // for each
} // DFS


DFS-Visit(node U) {
    color[U] = gray;
    δ[U] = ++time;
    for each vertex ∈ Adj[U] {
        if (color[V] == white) {
            p[V] = U;
            DFS-visit(V);
        } // if
    } // for each

    color[U] = black;
    f[U] = ++time;
} // DFS-Visit
```

EX:

INITIAL:

- **Node :**

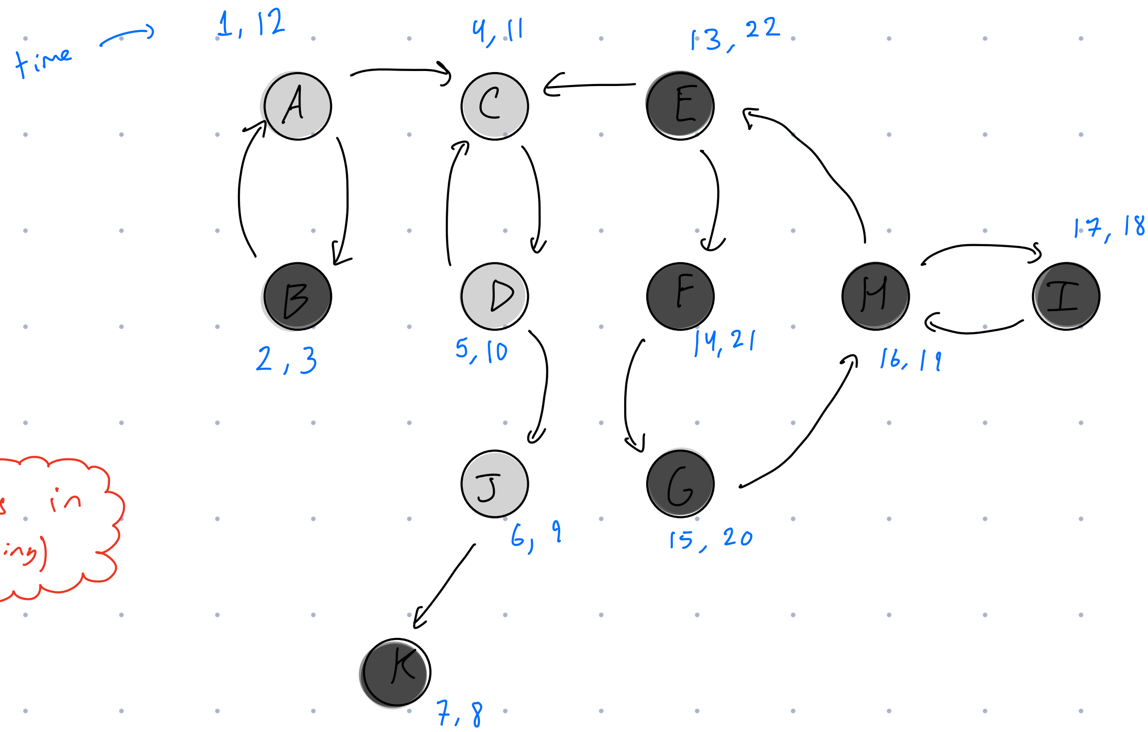
  color

  p - parent

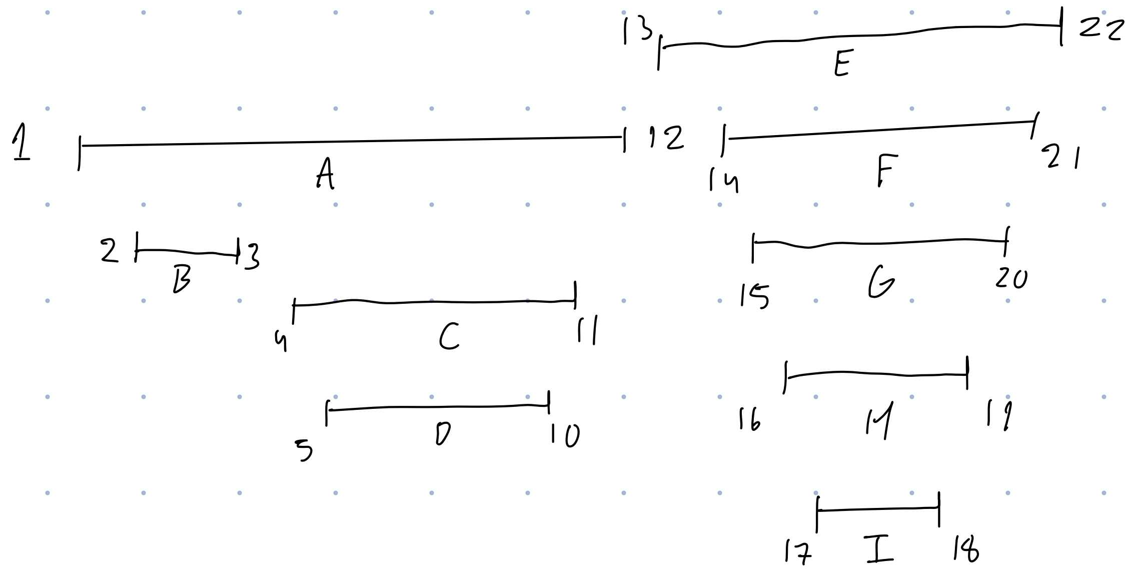
  δ - discovery time

  f - finish time

Note: Consider all nodes in alphabetical order (ascending)



So What Does This Tell Us???



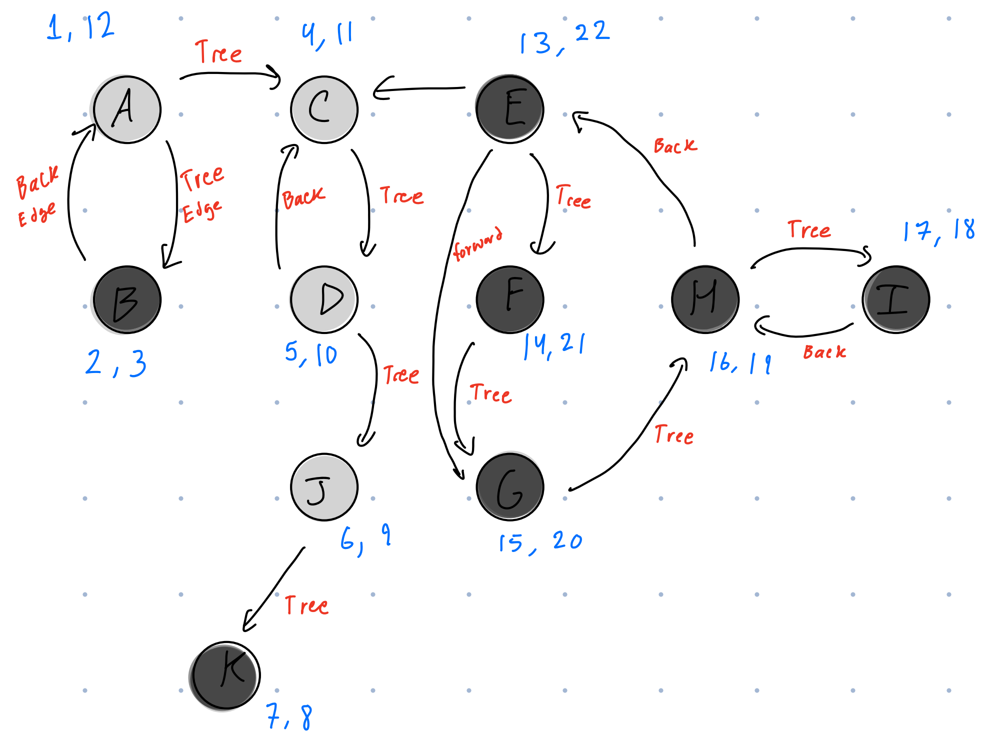
- None of these times overlap!

  =

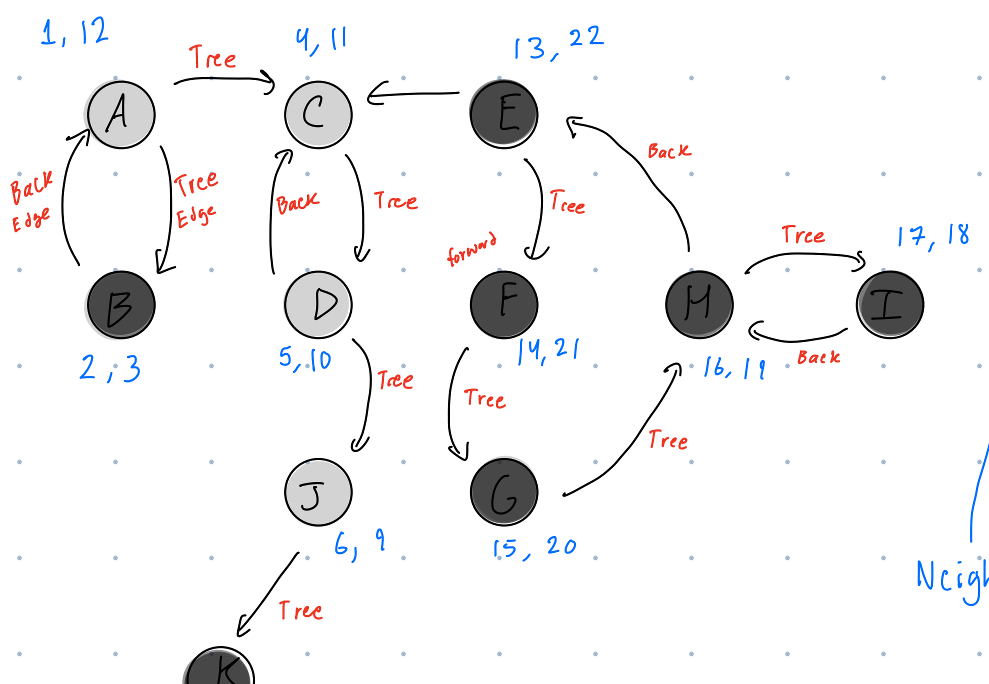
  Parenthesis structure

- Edges can be classified (tree, back,

forward, and cross edges)



- Ultimately, this structure allows us to find strongly connected components
  - A part of directed graphs: Sets of nodes that are all mutually reachable. Every node is in a set minimum set size is one.



SCC:

{A, B}

{C, D}

{J}

{K}

{E, F, G, H, I}

Neighborhoods!

7, 8

How do we use DFS to find SCC?

we will use DFS twice!

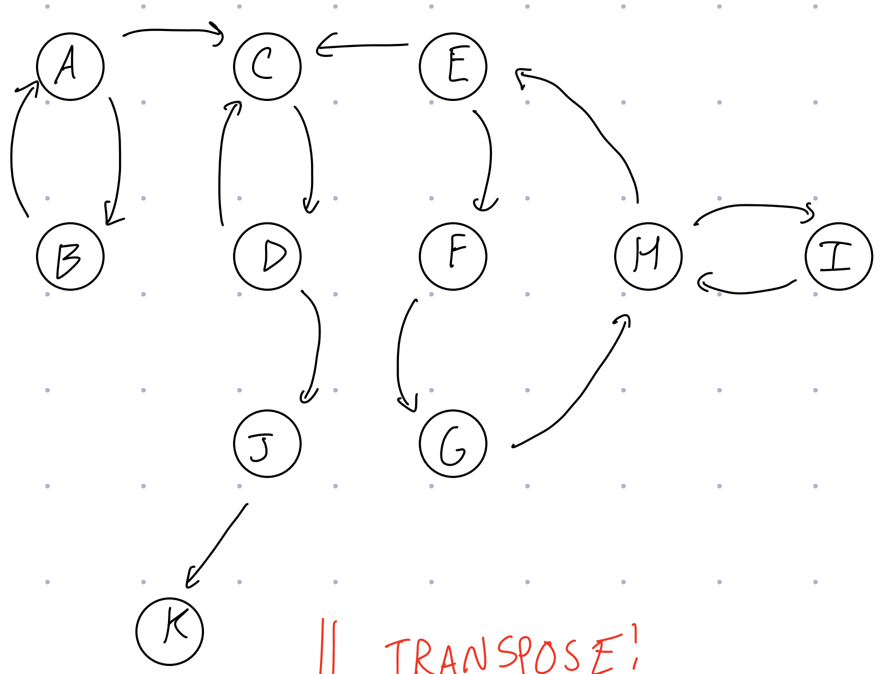
$G = (V, E)$

$G^T = (V, E^T)$ transpose

$(a, b) \in E \Rightarrow (b, a) \in E^T$

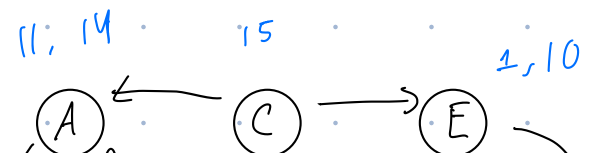
Note: G & G^T have same SCC

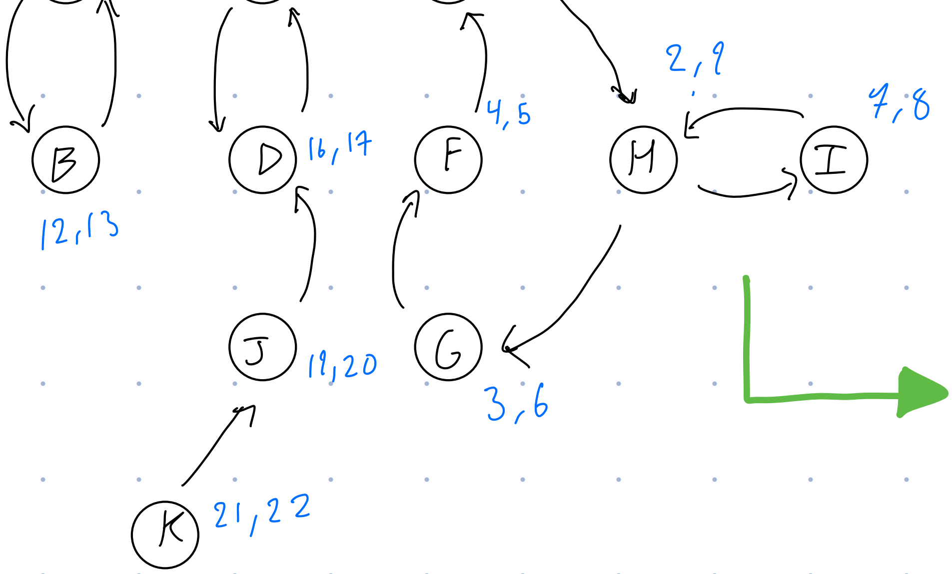
1. DFS(G) remember $f[\ ]$ finish time
2. Generate G^T
3. DFS (G^T) considering nodes in decreasing $F[\ ]$



E - 22
F - 21
G - 20
H - 19
I - 18
A - 12
C - 11
D - 10
J - 9
K - 8
B - 3

$\Downarrow$ TRANSPOSE!
(Reverse arrow directions)





SCC:

{A, B}

{C, D}

{J}

{K}

{E, F, G, H, I}