# Lecture 12 - Oct 20

## *Graphs*

***Adapting DFS for Graph Questions***
***BFS: Marking Vertices and Edges***
***BFS: First Example on a Tree***

## Announcements/Reminders

- Today's class: notes template posted
- Assignment 1 due on Wednesday, October 22
- Test 1 next Monday, October 27:
  + Guide released not yet (Tuesday).
  + Review Session (slides, notes): Wednesday
  + Review Session (A1, more Q&A): Friday
- Tutorial Exercises so far:
  + Tutorial Week 1 (2D arrays)
  + Tutorial Week 2 (2D arrays, Proving Big-O)
  + Tutorial Week 3 (avg case analysis on doubling strategy)
  + Tutorial Week 4 (Trinode restructuring after deletions)

# Test 1 (WSC, 4:30 PM to 5:20 PM)

Monday, Oct 27

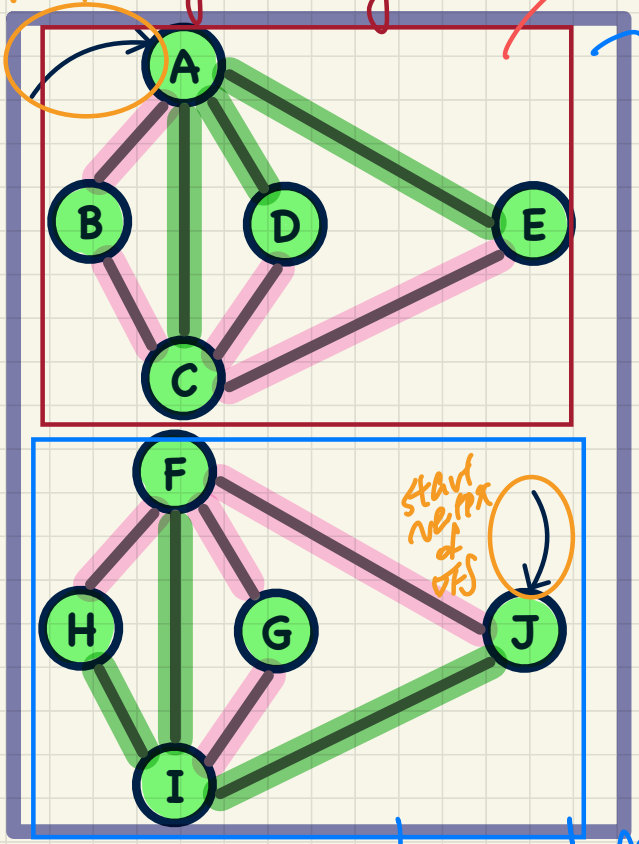
- **Coverage**
  - + Lecture materials (slides, notes, example code) up to and including Monday, October 20
  - + Tutorials 1 to 4
  - + Assignment 1
- **Format**

eClass: 100 marks
programming part: X cases

  - + **Programming** Part (Eclipse):
    - * Import a Java starter project (like A1)
    - * Implement Java classes/methods to pass test cases
  - + **Written** Part (eClass):

Two visible

    - * Primarily MCQs
    - * Written questions (e.g., short answers, justifications, proofs)




subgraph traversed by 1st DFS

↳ a DFS starting at vertex A is guaranteed to visit all vertices in the C.C. that A belongs to

↳ not connected

- > 1 connected components

↳ multiple passes of traversal

↳ - DFS
- BFS

RT of DFS

$O(m + n)$ → assumption getting a vertex's incident edges is $O(1)$

$|V|$ $|E|$

not the case for edge list strategy

↓ connected component traversed by 2nd DFS.

# Graph Traversal

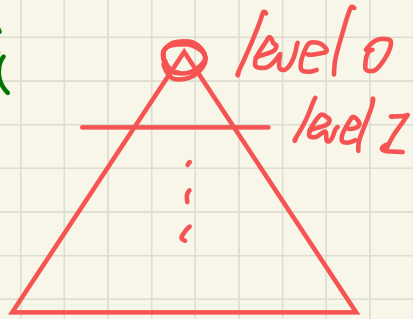
Assume G is connected

a DFS gives a spanning tree of G

DFS tree

a BFS gives a spanning tree of G

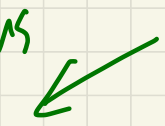
BFS tree, level tree



If input graph $G$ is not assumed to be connected.

```
boolean done = false;
while( ! done. ) {
    dfs(some unvisited vertex)
    if( # visited nodes so far = |V| ){
        done = true
    }
}
```

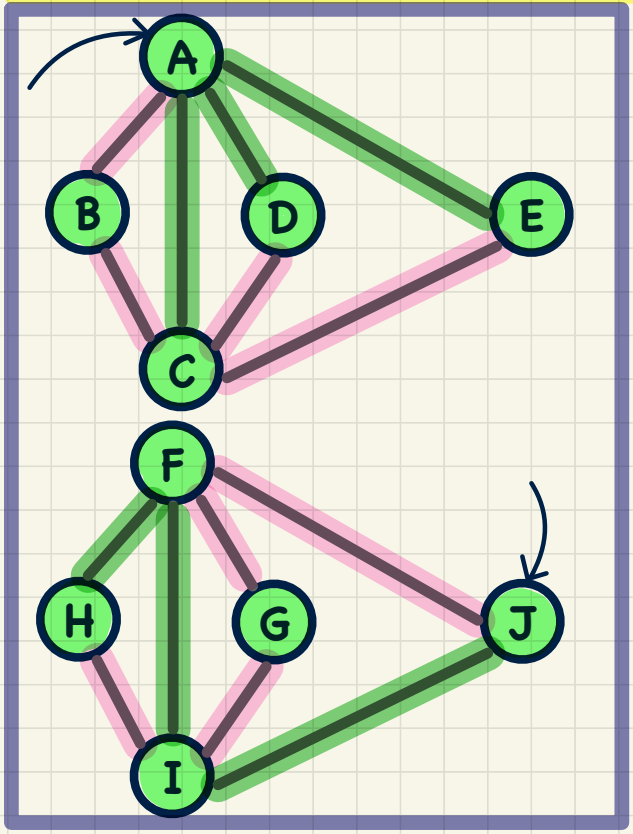
# iterations = # connected components.



* a single DFS can return a single C.C.

# Graph Traversals: Adapting DFS

## Efficient Traversal of Graph G:




4. if v is never encountered, then path does not exist!

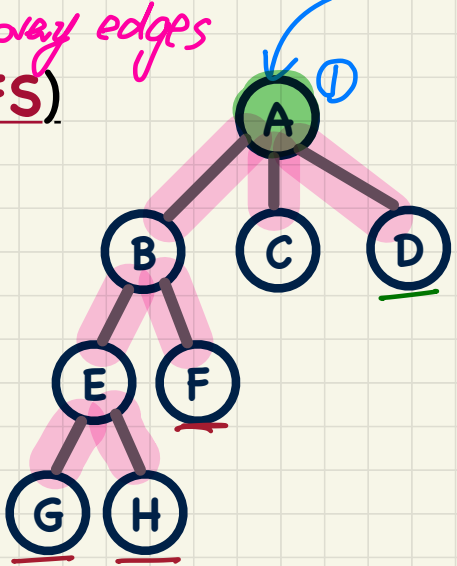
### Graph Questions:

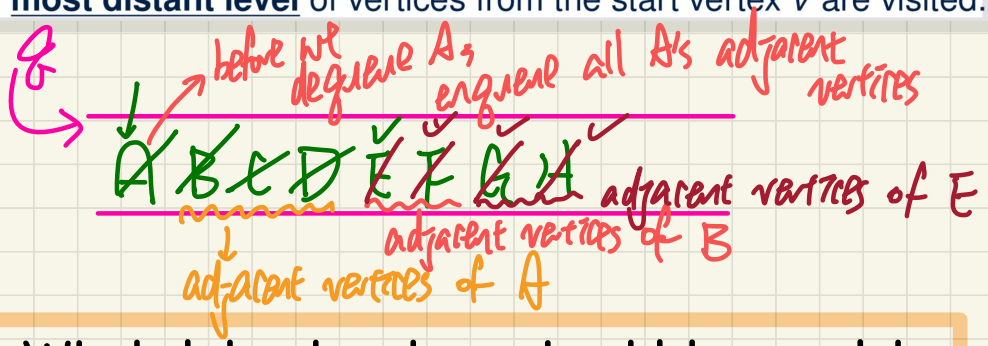
- Fina a path between {u, v} ⊆ V
  1. start a DFS from u.
  2. Maintain a list to store the visited vertices.
  3. if v is visited, return the list so far.
- Is v reachable from u
  similar to calculating a path
- * Find all connected components of G.
- Compute a spanning tree of a connected G.
  ↳ # discovery edges = # edges in G − 1
- Is G connected?
  ↳ # visited nodes $\stackrel{?}{=} |V|$
- If G is cyclic, return a cycle.
  ↳ return the path that leads to a back edge.

# Graph Traversal: Breadth-First Search (BFS)

A ***breadth-first search (BFS)*** of graph G = (V, E), starting from some vertex $v \in V$:

- Visits every vertex ***adjacent*** to $v$ before visiting any other (**more distant**) vertices
  - ***BFS*** attempts to stay as **close** as possible, whereas ***DFS*** attempts to move as **far** as possible
  - ***BFS*** proceeds in rounds and divides the vertices into ***levels***
- **No backtracking** in ***BFS***: it is completed **as soon as** the **most distant level** of vertices from the start vertex $v$ are visited.




Q. What data structure should be used to keep track of the visited nodes?

FIFO queue

# Breadth-First Search (BFS): Marking Vertices & Edges

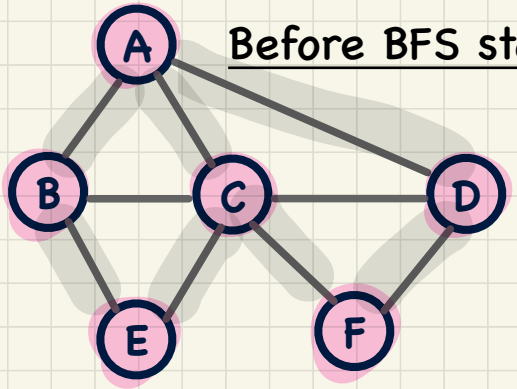
Before the ***BFS*** starts:

- All vertices are ***unvisited***.
- All edges are **unexplored/unmarked**.

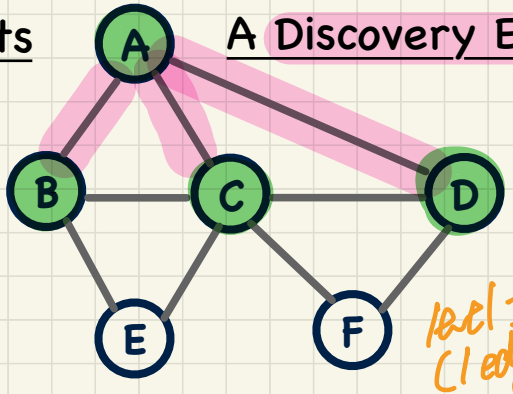
Over the course of a ***BFS***, we **mark** vertices and edges:

- A vertex is marked ***visited*** when it is **first** encountered.
- Then, we iterate through each of v's **incident edges**, say *e*:
  - If edge *e* is already **marked**, then skip it.
  - Otherwise, for an *undirected* graph, an edge is marked as:
    - A ***discovery*** edge if it leads to an ***unvisited*** vertex
    - A ***cross*** edge if it leads to a ***visited*** vertex
      (i.e., from a different ***branch*** at the same ***level***).

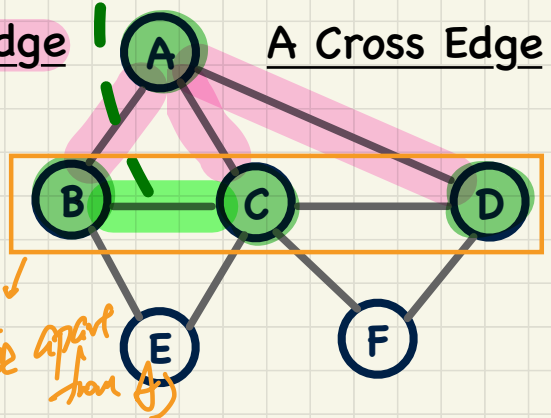
Cross edge: connecting nodes at the same level




Before BFS starts




A Discovery Edge




A Cross Edge