

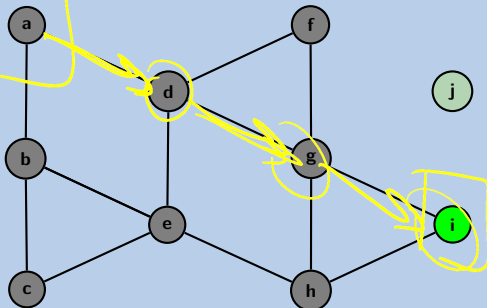
Any reason we shouldn't use a Queue?

When we use a Queue:

- 1 This algorithm is called BFS (breadth-first search)
- 2 We increase our horizon away from the starting node

worklist ← i ←

$\{a: \text{null}, d: a, g: d, i: g\}$



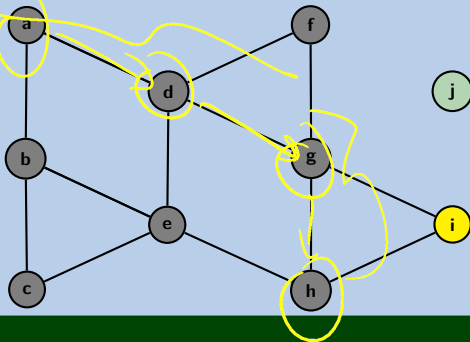
Any reason we shouldn't use a Queue?

When we use a Queue:

- 1 This algorithm is called ~~BFS~~ (breadth-first search)
- 2 We increase our horizon away from the starting node

worklist ← i ←

DFS



Any reason we shouldn't use a Queue?

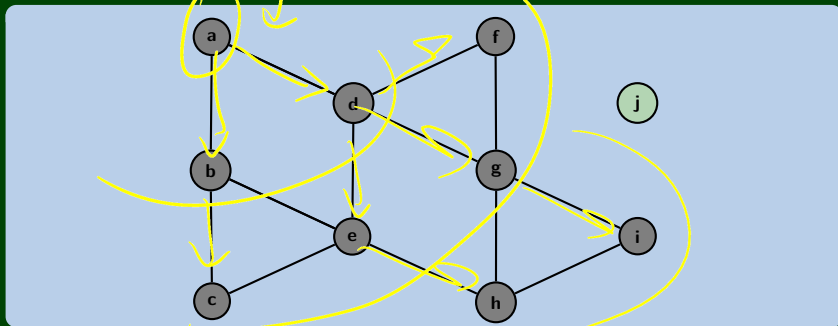
(Continued!)

When we use a Queue:

- 1 This algorithm is called BFS (breadth-first search)
- 2 We increase our horizon away from the starting node

worklist ←

BFS



## Trade-Offs

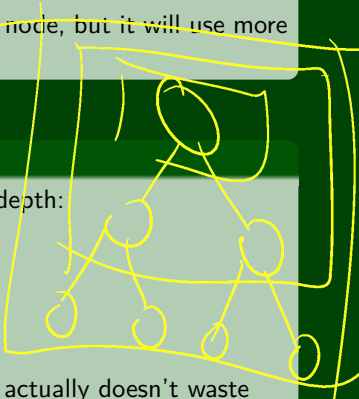
- DFS has better space usage, but it might find a circuitous path
- BFS will always find the shortest path to a node, but it will use more memory

## Iterative Deepening

Iterative Deepening is a DFS that **bounds** the depth:

```
1  int depth = 1;  
2  while (there are nodes to explore) {  
3      dfs(v, depth);  
4      depth++;  
5  }
```

Since **most of the vertices are “leaves”**, this actually doesn't waste much time!

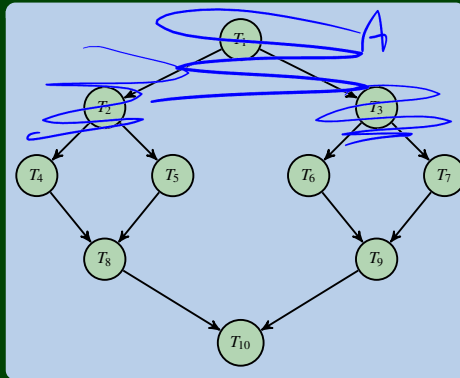


## Topological Sort

Given a DAG ( $G = (V, E)$ ), output all the vertices in an order such that no vertex appears before any vertex that has an edge to it.


“Output an order to process the graph that meets all dependencies”

**This is how we can allocate work in the ForkJoin model!**



## Implementing Topological Sort

Throw all the **in-degrees** in a priority queue. `removeMin()` repeatedly.

- This works, but it's **too slow**.
  - Insight: PriorityQueues must deal with negative numbers; indegree will never be negative!
  - Instead: Split ready vs. not ready (0 vs. non-zero) sets
  - The “ready set” is a worklist!
- 

## Setup

```
1 output = []
2 deps = {}
3 worklist = []
4 for (v : vertices) {
5     deps[v] = in-degree(v);
6     if (deps[v] == 0) {
7         worklist.add(v);
8     }
9 }
```

## Do Work

```
1 while (worklist.hasWork()) {
2     v = worklist.next();
3     output.add(v);
4     for (w : neighbors(v)) {
5         deps[w] -= 1
6         if (deps[w] == 0) {
7             worklist.add(w);
8         }
9     }
10 }
```