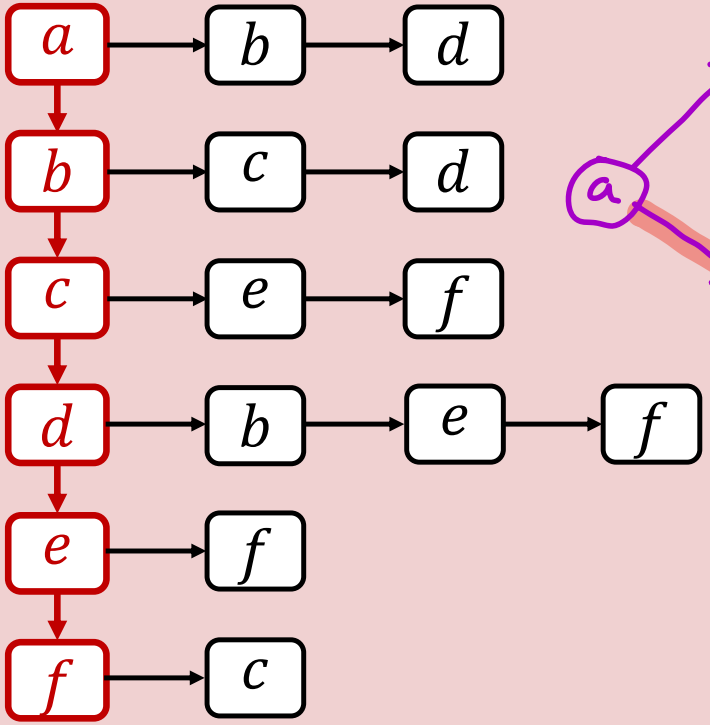
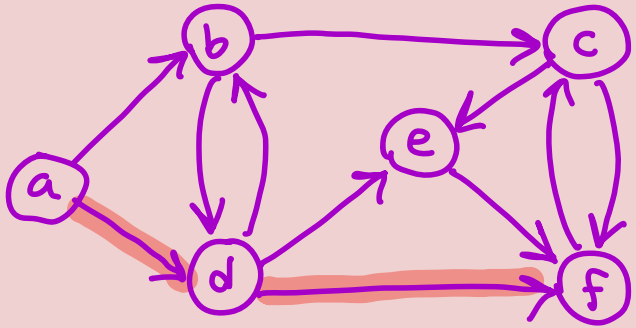
# Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

What is the length of the shortest $a \rightsquigarrow f$ path in the graph whose adjacency list representation is below?




1 (A)

2 (B)

3 (C)

4 (D)


# Lecture 22: Graph Traversals

CS 2110

April 14, 2026

# Today's Learning Outcomes

90. Describe adjacency list and matrix representations of a graph and compare the space/time complexities of operations on each of these representations.

91. Perform BFS and DFS traversals of a given graph by hand and in code.

92. Visualize the state of a graph traversal at a given point of its execution, describing each node as either undiscovered, discovered, visited, and/or settled.

# (Improved) Adjacency List Representation

```
class AdjListGraph implements Graph<...> {
   record AdjListEdge(...) implements Edge<...> { }

   static class AdjListVertex implements Vertex<...> {
      String label;
      HashMap<String, AdjListEdge> outEdges;
   }

   private HashMap<String, AdjListVertex> vertices;

   // methods
}
```

**Space Complexity:** $O(|V| + |E|) = O(|E|)$ for connected graphs

vertices map ↗ ↑ all outEdges maps

**Runtime Complexities:**

Check for a vertex: $O(1)$ vertices.containsKey()

Check for an edge/neighbor: $O(1)$

check for tail, get tail vertex, check for head

Iterate over all vertices: $O(|V|)$

Iterate over all edges: $O(|V| + |E|) = O(|E|)$

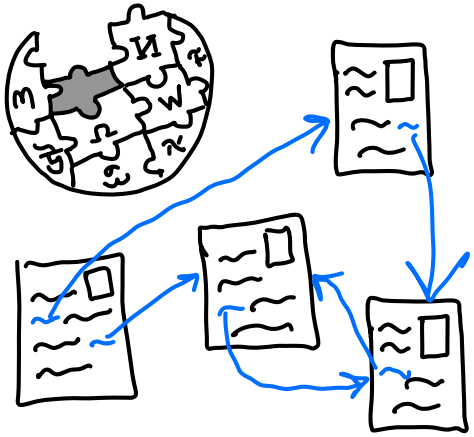
Iterate over neighborhood: $O(|V|)$

# Graph Traversals

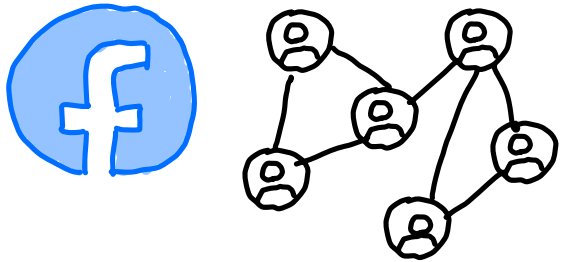
Procedure to systematically visit vertices/edges in a graph in an order that respects "local structure".

- Follow outgoing edge from one vertex to its neighbor.

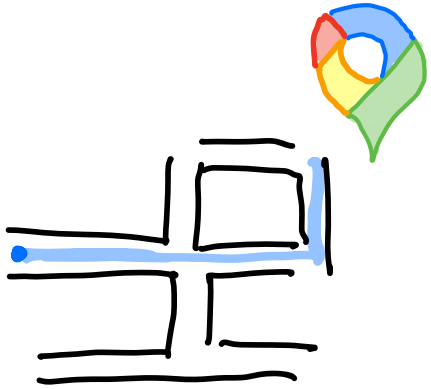
Why? Local structure holds rich information



Wikipedia Links

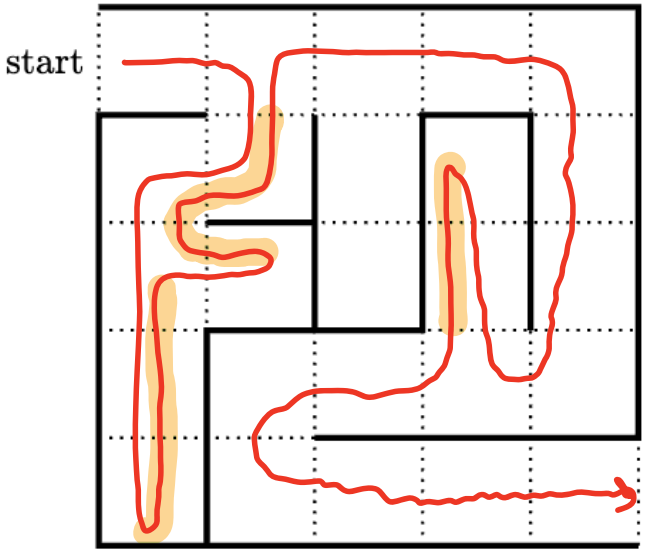


Social Networks

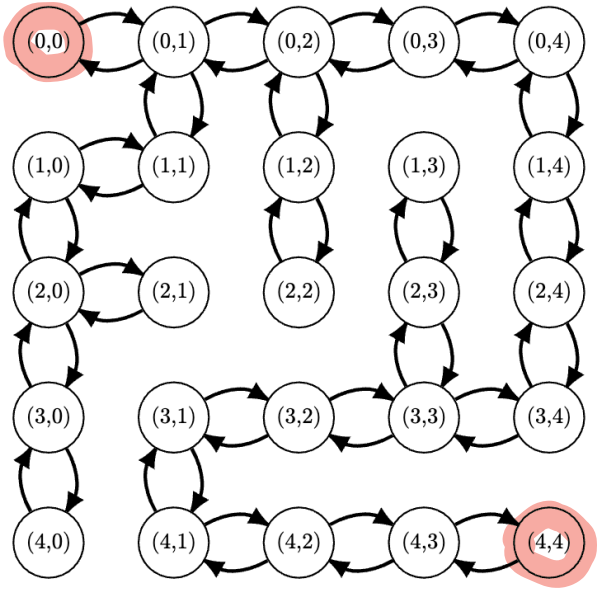


Route Planning

# Navigating a Maze




= is there a path from source → dest in this graph



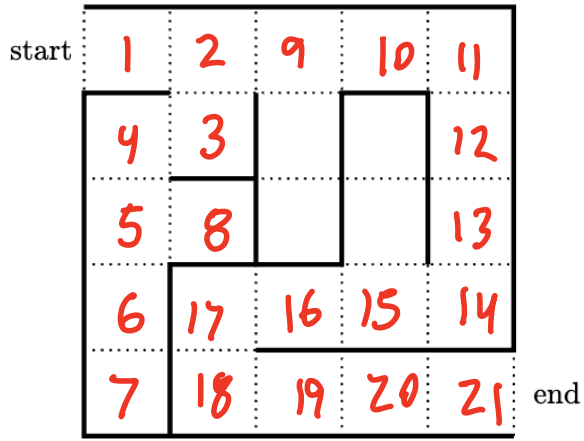

"Are these vertices connected"

Rule: Always in one square, can only see whether it's possible to move to adjacent squares

Idea: "Blindly" follow one path, backtrack if we hit dead end.

# Depth-First Search (DFS)

A search procedure that fully explores one path away from source before backtracking to consider other paths.




- begin at source
- follow an edge to a neighbor
- set off on a DFS from that neighbor
  * Recursion *
- stop if we reach destination * base case *
- unwind (backtrack) if vertex has no neighbors, continue when we find another alternate neighbor to visit

# A First Attempt

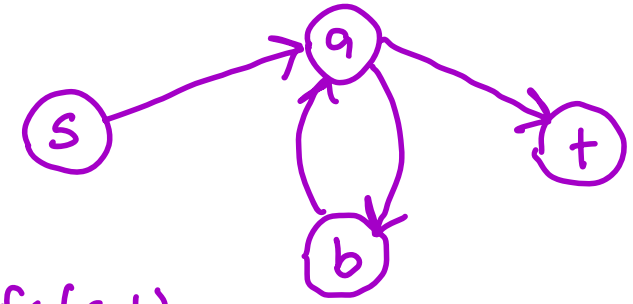
```
/** Returns whether there is a path from `source`
 *  to `dest` in this graph. */
public static <...> boolean dfs(V source, V dest) {
  if (source == dest) {  return true;  }
  for (E edge : source.outgoingEdges()) {
    if (dfs(edge.head(), dest)) {  return true;  }
  }
  return false;
}
```

base case

iterate over neighbors

step to neighbor and initiate new search

What's wrong?




dfs(s,t)

↘ dfs(a,t)

↘ dfs(b,t)

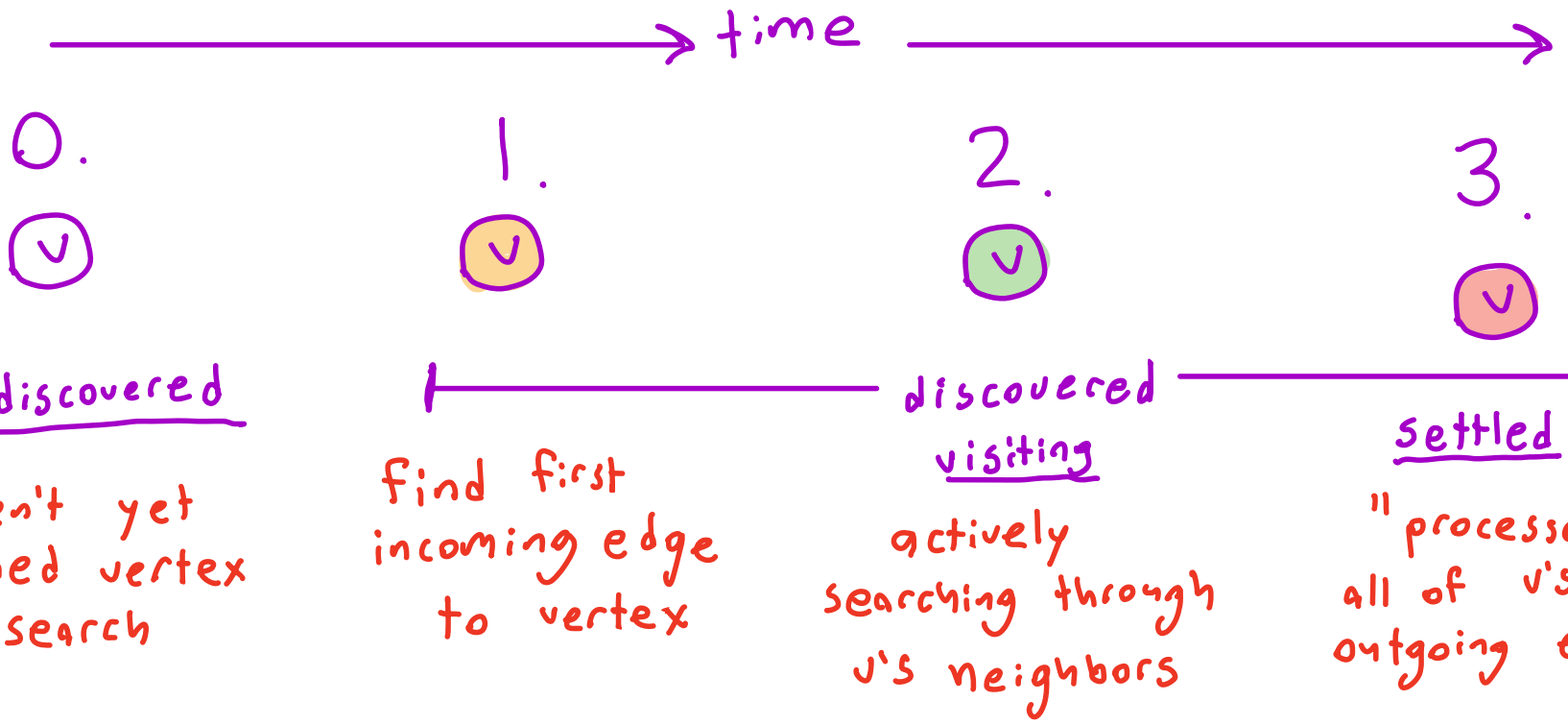
↘ dfs(a,t)

infinite recursion!

Need a way to avoid infinite cycling

Idea: Keep track of which vertices we've already seen.

# Vertex Status During Search



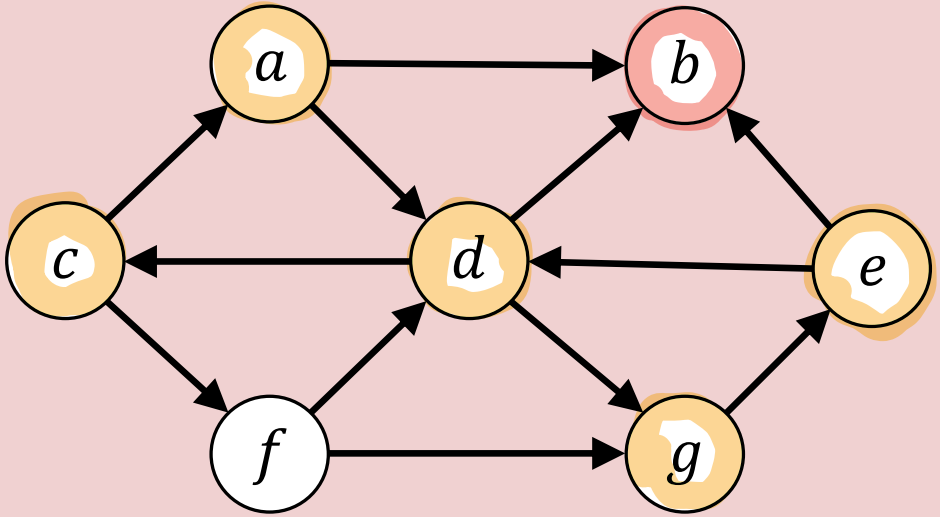

To prevent infinite cycling, DFS keeps track of which vertices have been discovered and only visits newly discovered ones.

# **Coding Demo:** Fixing our DFS Code

# Poll Everywhere

PollEv.com/javabear  text **javabear** to 22333

In which order will the vertices be *discovered* during a DFS from $c$ to $e$ of the following graph? Assume neighbors are discovered in alphabetical order.




skipped b

$c, a, d, g, e$ **(A)**

don't rediscover a

$c, a, b, a, d, g, e$ **(B)**

only backtrack to a

$c, a, b, f, d, g, e$ **(C)**

$c, a, b, d, g, e$ **D**

# DFS Complexity

```
boolean dfsRec(V current, V dest, Set<V> discovered) {
  if (current == dest) {  return true; }
  for (E edge : current.outgoingEdges()) {
     V neighbor = edge.head();
     if (!discovered.contains(neighbor)) {
        discovered.add(neighbor);
        if (dfsRec(neighbor, dest, discovered)) {  return true; }
     }
  }
  return false;
}
```

O(1)

O(|V|) iterations

O(1) · O(|E|) = O(|E|)

O(|E|)

O(1) · O(|V|) = O(|V|)

O(|V|) calls – one per discovered vertex

per call * but only O(|E|) total iterations across all calls (better bound than $O(|V|^2)$)

only enter if block once per vertex discovery

Overall: O(|V| + |E|)

Great! linear in graph size

# Poll Everywhere

PollEv.com/javabear  text **javabear** to 22333

What is the tightest worst-case space complexity of our DFS implementation?

```
boolean dfsRec(V current, V dest, Set<V> discovered) {
  if (current == dest) {  return true; }
  for (E edge : current.outgoingEdges()) {
     V neighbor = edge.head();
     if (!discovered.contains(neighbor)) {
        discovered.add(neighbor);
        if (dfsRec(neighbor, dest, discovered)) {  return true; }
     }
  }
  return false;
}
```

grows to O(|V|) size

O(1) extra memory per call

O(|V|) recursive depth

| | |
|---|---|
| $O(1)$ | **(A)** |
| $O(\|V\|)$ | **(B)** |
| $O(\|E\|)$ | **(C)** |
| $O(\|V\| + \|E\|)$ | **(D)** |

# Search vs. Traversal

Search: Given source s and destination t, locate an $s \rightsquigarrow t$ path

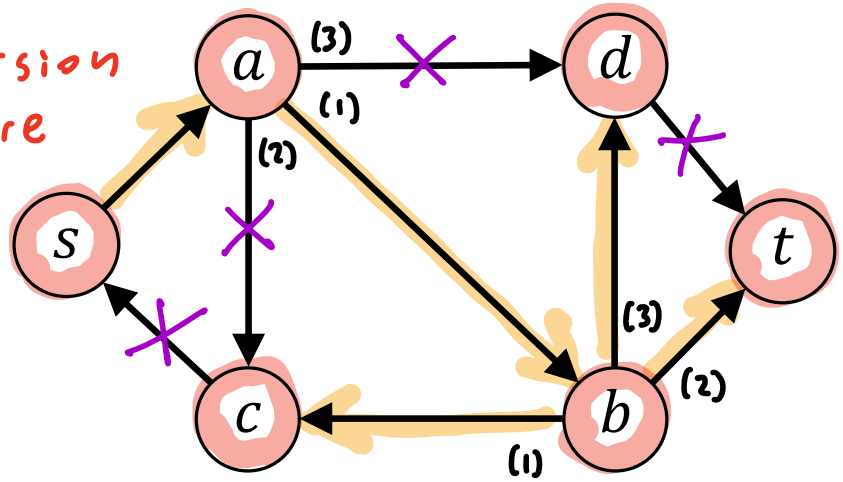
- Often, we won't need to explore whole graph

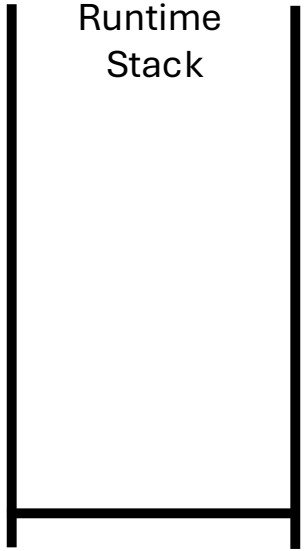
Traversal: Systematically visit every vertex and edge in the graph, taking some action during the visit

- Another iteratee pattern
- Use a modified DFS that doesn't stop at destination

# Visitation vs Settlement Orders

* see the animated version in the lecture notes




**Visitation:** s a b c t d

**Settlement:** c t d b a s

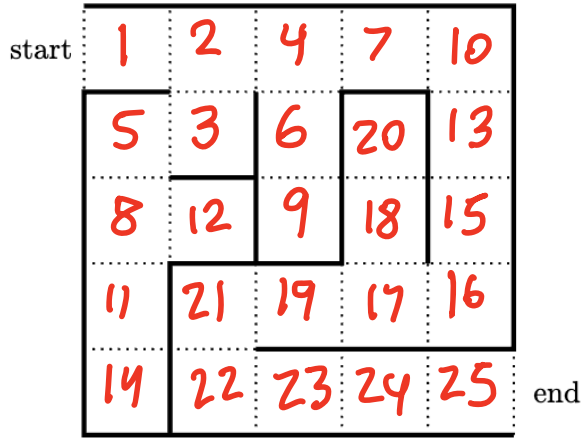
# Breadth-First Search

DFS focuses in on one particular path

Alternate approach: Fan out and have "scouts" walk all paths simultaneously

Breadth-First Search (BFS) strategy

- Always locates shortest path
- Fanning out can be expensive if we have a good heuristic to guess shortest path



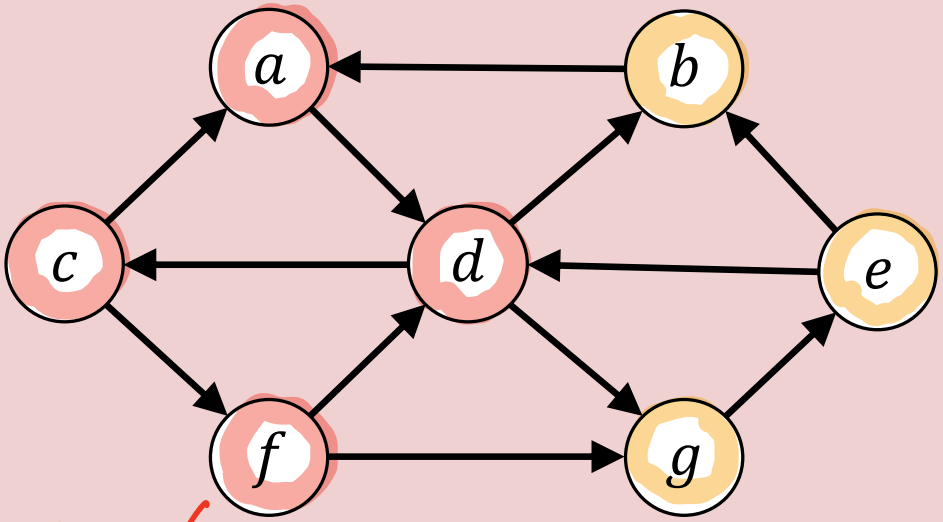

- Need to organize incoming messages from scouts
- Queue up their opportunities to report back

# **Coding Demo:** BFS Traversals

# Poll Everywhere

PollEv.com/javabear text **javabear** to 22333

In which order will the vertices be visited in a BFS traversal of the following graph starting from $c$? Assume neighbors are discovered in alphabetical order.



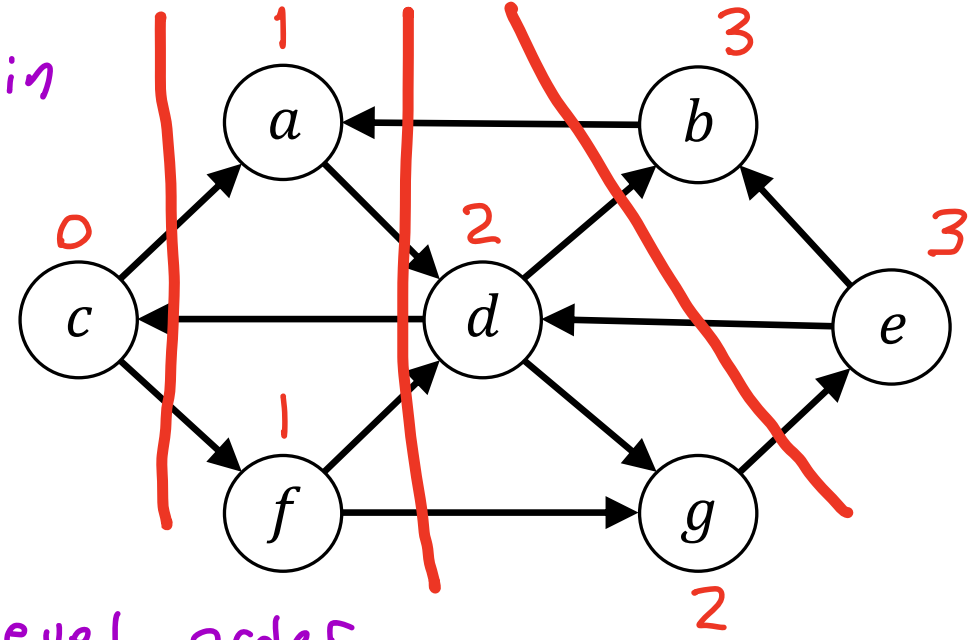

c d d f g b e

- $c, a, f, d, g, b, e$ **(A)**
- $c, a, d, b, f, g, e$ **(B)**
- $c, a, d, f, g, b, e$ **(C)**
- $c, a, f, d, g, e, b$ **(D)**

# Traversal Levels

BFS sweeps across graph in layers (or levels), moving outward from source

level of vertex v = length of shortest s ⇝ v path




Vertices are traversed in level order

We'll take inspiration from this next lecture when we discuss shortest path algorithms.

# BFS Complexity

```
while(!frontier.isEmpty()) {
    V v = frontier.remove();
    if (v == dest) { return true; }

    for (E edge : v.outgoingEdges()) {
        V neighbor = edge.head();
        if (!discovered.contains(neighbor)) {
            discovered.add(neighbor);
            frontier.add(neighbor);
        }
    }
}
return false;
```

O(|V|) iterations

O(1) · O(|V|) = O(|V|)

O(|V|) iterations

O(1) · O(|E|)

O(1) · O(|E|)

O(1) · O(|V|)

O(1) · O(|V|)

- each vertex added when discovered
  one removed per iteration

per outer loop iteration

but O(|E|) total iterations

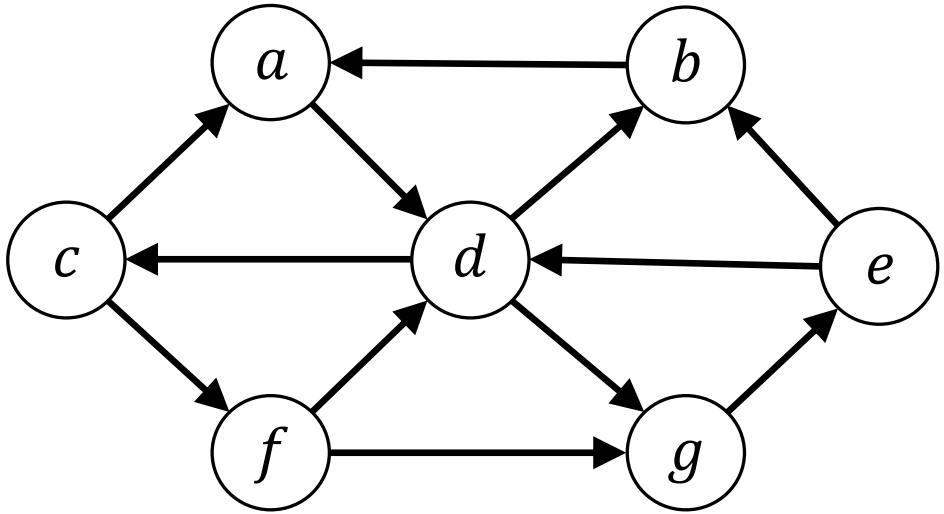
Another O(|V| + |E|) algorithm

Space complexity O(|V|)

discovered + frontier

# Setting up Next Lecture

Highlight the edges that discovered new vertices during BFS




What structure do these edges form?

How do these edges help us find paths from the source vertex?