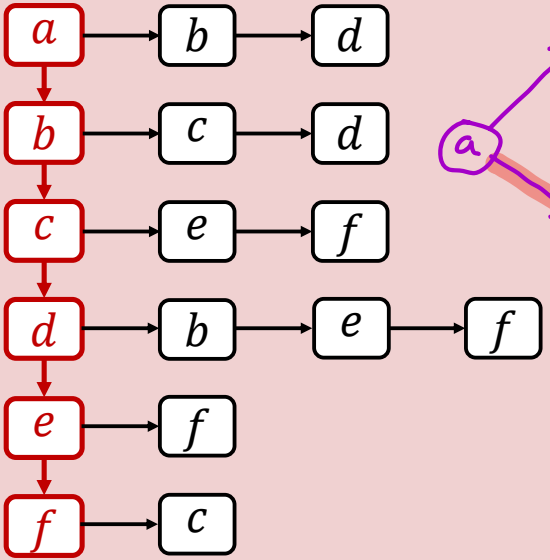
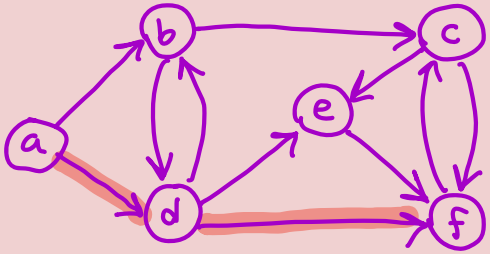
Poll Everywhere

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)

Improved Adjacency List Representation

Rather than using nested LinkedLists, which have O(|V|) lookup, use nested HashMaps, which have expected O(1) lookup.

```
public class AdjListGraph implements Graph<ALVertex, Edge<ALVertex>> {
  /** A vertex in this graph responsible for tracking its neighbors. */
  public static class ALVertex implements Vertex<Edge<ALVertex>> {
    /** The label of this vertex */
    private final String label;

    /** Map associating neighbor's labels with Edges connecting to them. */
    private final HashMap<String, Edge<ALVertex>> outEdges;
  }

  /** Map associating labels with ALVertex objects. */
  private final HashMap<String, ALVertex> vertices;
}
```

each vertex keeps track of neighborhood

graph is collection of vertices

vertices, outEdges

Space Complexity:

$O(|V| + |E|)$

Check for an edge:

O(1) using Map get()

Iterate over neighbors:

O(degree) using KeySet()

Iterate over all edges:

$O(|V| + |E|) = O(|E|)$

Add vertex/edge:

O(1) using Map put()

Comparing Representations

Representation	Memory	Time to Query Edge	Time to Iterate Neighbors	Time to Iterate all Edges
Adj Matrix	$O(\|V\|^2)$	$O(1)$	$O(\|V\|)$	$O(\|V\|^2)$
Adj List (Linked)	$O(\|V\|+\|E\|)$	$O(\|V\|)$	$O(\|V\|)$	$O(\|V\|+\|E\|)$
Adj List (Map)	$O(\|V\|+\|E\|)$	$O(1)$	$O(\text{degree})$	$O(\|V\|+\|E\|)$

Dense Graphs: have $|E| \approx O(|V|^2)$: AM + AL have similar performance
memory layout likely makes AM ops faster

Sparse Graphs: have $|E| \approx O(|V|)$: AL has much better performance
↖ most real-world graphs



Lecture 22: Graph Traversals

CS 2110

November 10, 2026

Today's Learning Outcomes

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

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

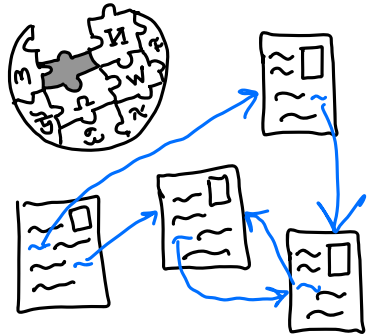
83. Visualize the state of a graph traversal at a given point of its execution, describing each vertex as either *undiscovered*, *discovered*, *visited*, and/or *settled*.

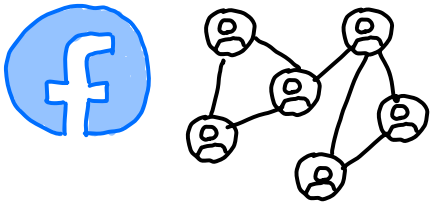
Graph Traversals

Procedures to systematically visit all vertices/edges in a graph.

Typically, we do this in an order that respects the graph's "local structure," following outgoing edges from one vertex to its neighbor.

** Local structure encodes meaningful information about the graph.

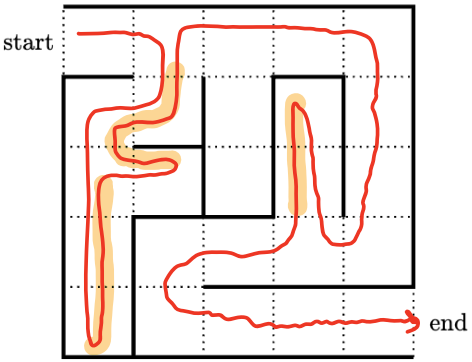




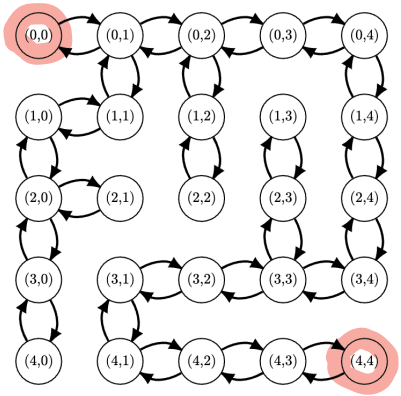


Navigating a Maze

Rule: At any point, we're "standing" in one maze square and can only see whether it's possible to move to adjacent squares.



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

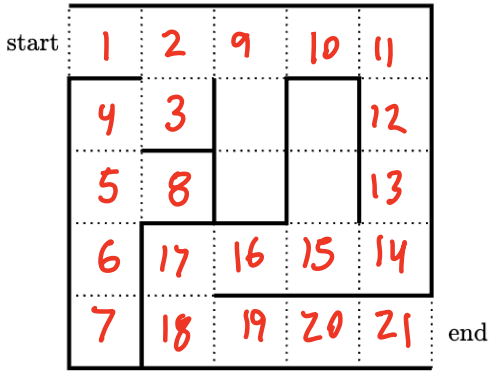


"Are these vertices connected?"

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 the source *before* backtracking to consider other paths.



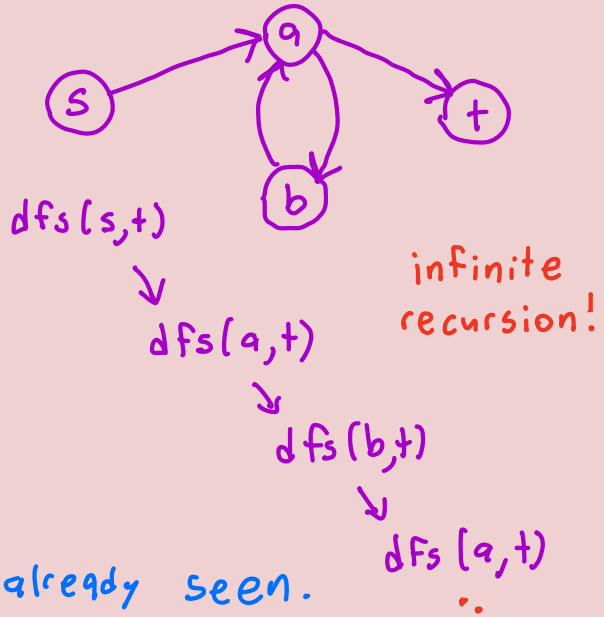
- begin at source
- follow an edge to a neighbor
- set off on a DFS from that neighbor
  * Recursion *
- stop if we reach destination * base case *
- unwind (backtrack) if vertex has no neighbors, continue when we find another alternate neighbor to visit

Poll Everywhere

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

What can go wrong with this implementation?

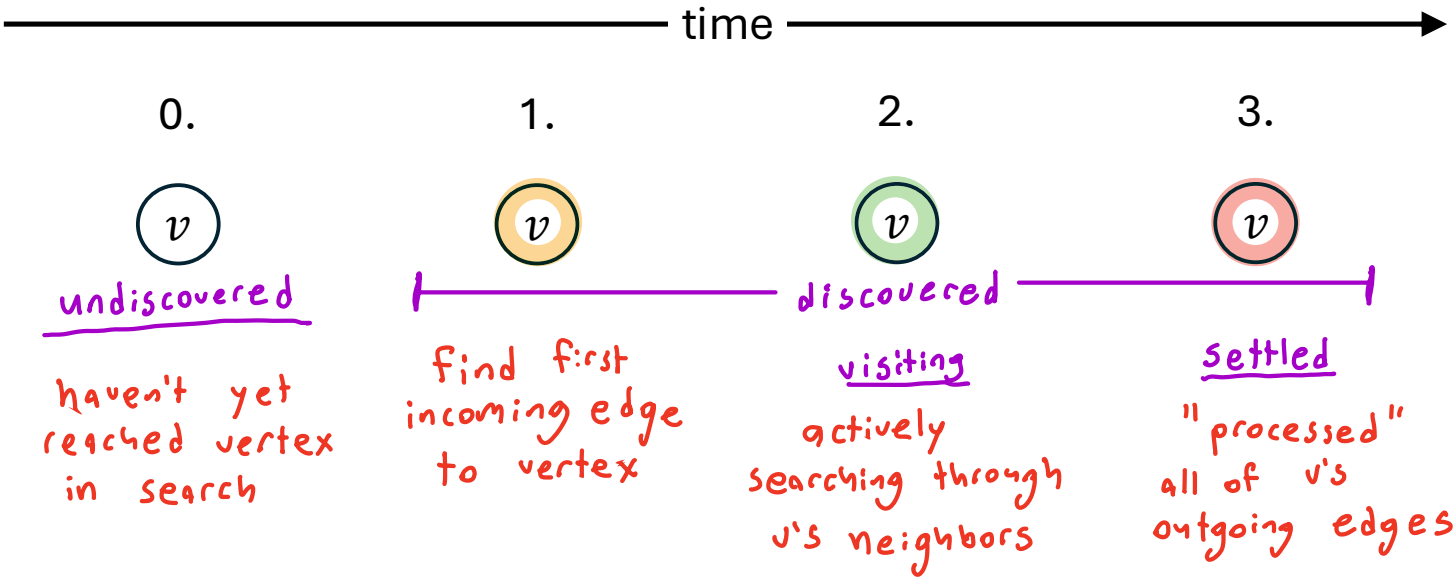
```
/** Returns whether there is a path from `src`
 *  to `dst` in this graph. */
public static <...> boolean dfs(V src, V dst) {
  if (src == dst) { return true; }  base case
  iterate over neighbors
  for (E edge : src.outgoingEdges()) {
    if (dfs(edge.head(), dest)) { return true; }
  } step to neighbor and initiate new
                                search
  return false;
}
```



Need to avoid infinite cycling

Idea: Keep track of 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.



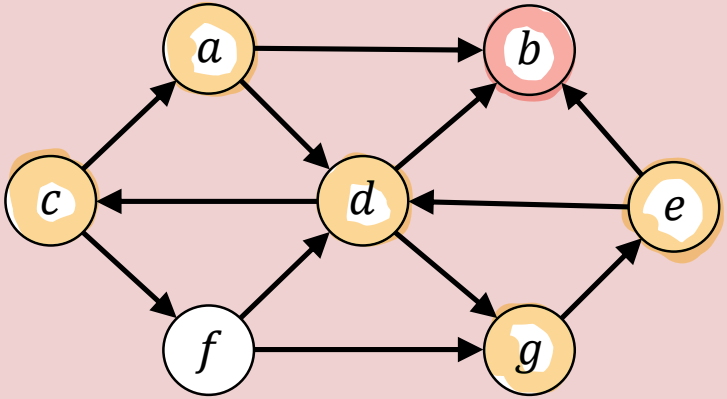
Coding Demo: Fixing our DFS Code



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 dst, Set<V> discovered) {
  if (current == dst) { return true; }

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

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

O(1) · O(|V|) = O(|V|)

O(|V|) iterations per call

but: total # iterations across all calls is O(|E|)
(tighter bound than O(|V|²))

O(1) · O(|E|) = O(|E|)

O(1) · O(|E|) = O(|E|)

O(1) · O(|V|) = O(|V|)

enter if body once per discovery

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

Great! Linear in size of graph!

Poll Everywhere

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



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

grows to O(|V|) size

```
boolean dfsRec(V current, V dst, Set<V> discovered) {
  if (current == dst) { return true; }

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

O(1) for local vars

O(|V|) recursive depth

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

Search vs. Traversal

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

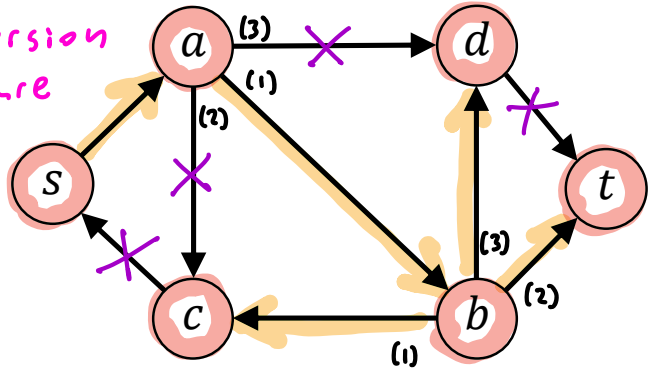
- Often, we won't need to traverse the entire graph to do this.

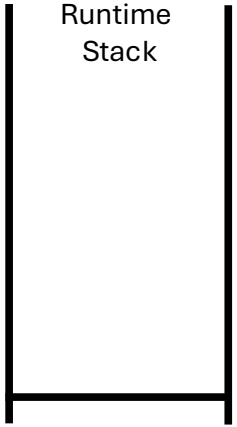
Traversal: Systematically visit every vertex and edge in the graph, taking some action during the visit

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

Visitation vs Settlement Orders

* see the animated version in the lecture notes





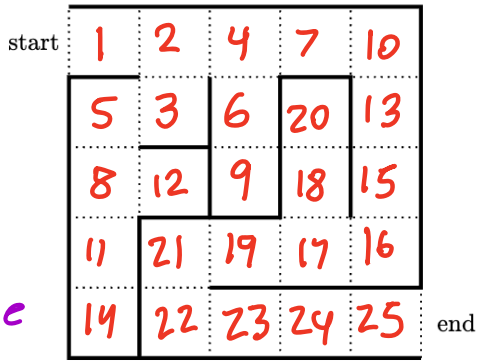
Visitation: s a b c t d

Settlement: c t d b a s

Breadth-First Search

DFS "focuses in" on one path at a time.

Alternative (BFS): "Fan out" a search party of "scouts" who follow all paths simultaneously



- 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



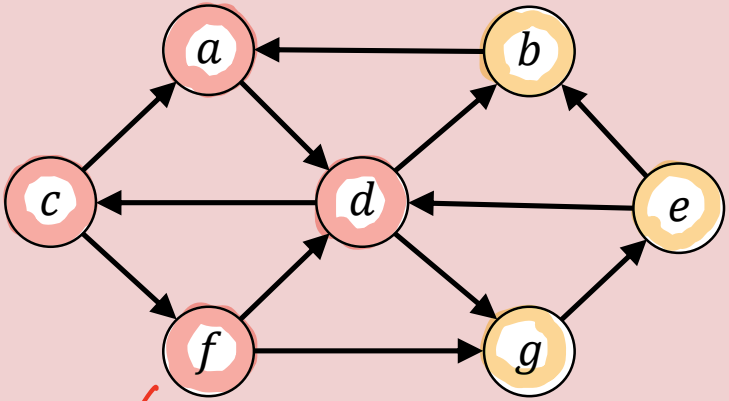
Coding Demo: BFS Traversals



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, 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)**

BFS Complexity

```
void bfs(V src, Consumer<V> action) {
  Set<V> discovered = new HashSet<>();
  Queue<V> frontier = new LinkedList<>();
  discovered.add(src); frontier.add(src);

  while(!frontier.isEmpty()) {
    V v = frontier.remove();
    action.accept(v);
    for (E edge : v.outgoingEdges()) {
      V neighbor = edge.head();
      if (!discovered.contains(neighbor)) {
        discovered.add(neighbor);
        frontier.add(neighbor);
}}}}
```

O(1)

discovery adds to frontier
one vertex removed per iteration

O(|V|) iterations, one per discovered vertex

O(1) · O(|V|) = O(|V|)

O(|V|) iterations per outer loop iteration, but O(|E|) total iterations

O(1) · O(|E|) = O(|E|)

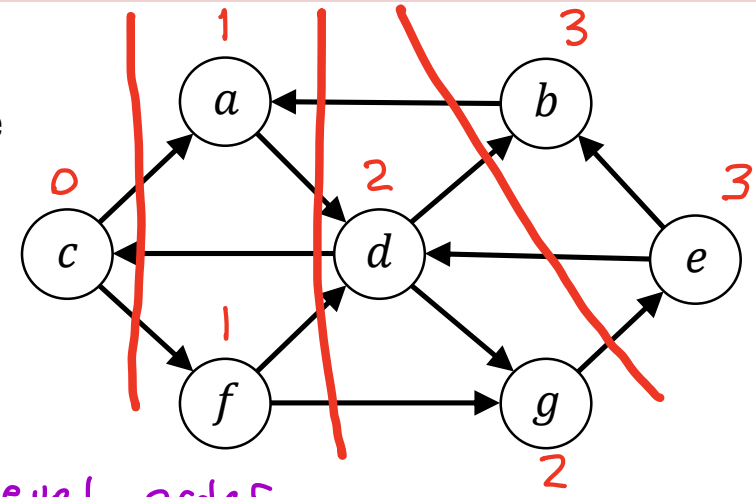
O(1) · O(|V|) = O(|V|)

Overall: O(|V|+|E|)
does same work as DFS, just in different order

Space Complexity O(|V|) for discovered, frontier

Traversal Levels

BFS sweeps across the graph in layers (or *levels*), expanding outward from the source.



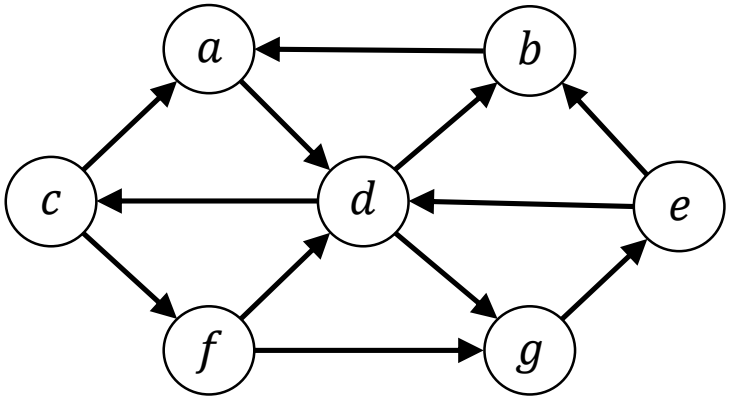
level of vertex v = length of shortest $s \rightsquigarrow v$ path

Vertices are traversed in level order

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

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?