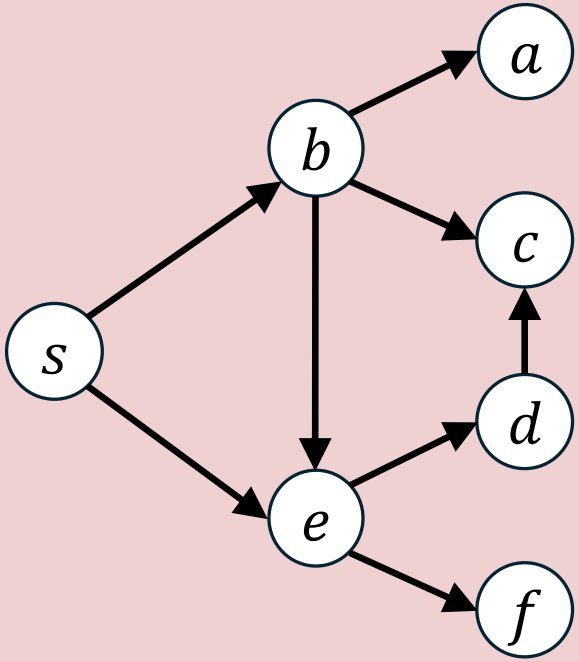
Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

Which of the following *cannot* be the visitation order of a BFS of the following graph starting from source vertex s?



s, b, e, a, c, d, f **(A)**

s, e, b, f, d, c, a **(B)**

s, b, e, c, d, a, f **(C)**

s, e, b, d, f, a, c **(D)**

Lecture 23: Shortest Paths

CS 2110

November 13, 2025

Today's Learning Outcomes

89. Identify substructures in graphs such as neighborhoods, paths, and cycles both visually and programmatically.

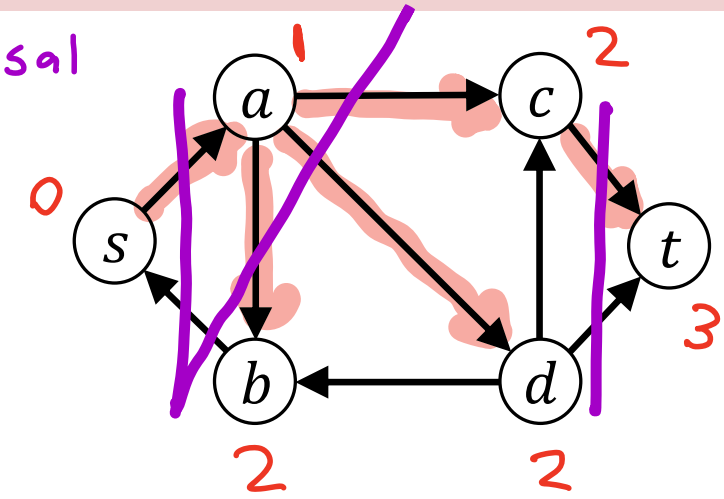
92. Visualize the state of a graph traversal at a given point of its execution, describing each node as either undiscovered, discovered, visited, and/or settled.

93. Describe Dijkstra's shortest path invariant and use it to establish properties on the unvisited portions of a graph at a given point in the execution of Dijkstra's algorithm.

94. Implement Dijkstra's shortest path algorithm and analyze its time/space complexities.

Recall: Traversal Levels

The level of vertex v, $\ell(v)$, in a traversal starting at s is the length of the shortest $s \rightsquigarrow v$ path.



BFS visits vertices in level order.

Think in phases:

1. visiting source (level 0) discovers all vertices at level 1
2. visiting vertices at level 1 discovers vertices at level 2
⋮

Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

Suppose (u, v) is an edge in a directed graph. What is the strongest statement we can make about the traversal levels of u and v, $\ell(u)$ and $\ell(v)$?

Imagine point when u was visited.

- If v undiscovered it will be in level $\ell(u)+1$
- If v discovered it was labeled earlier, so will be in level $\leq \ell(u)+1$

$\ell(v) = \ell(u) + 1$ **(A)**

$\ell(v) \geq \ell(u) + 1$ **(B)**

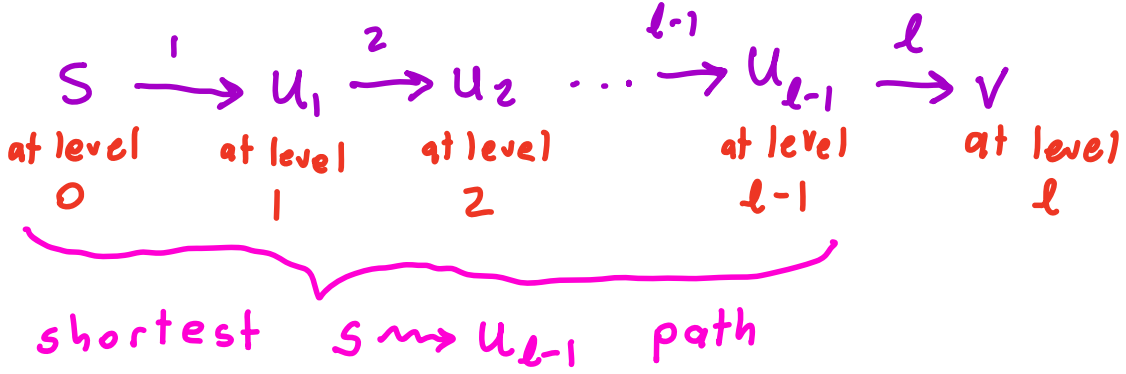
$\ell(v) \leq \ell(u) + 1$ **(C)**

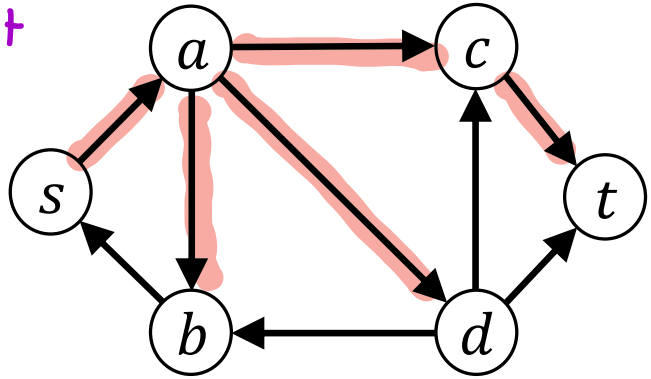
$\ell(v) \geq \ell(u)$ **(D)**

Coding Demo: Level-Augmented BFS

Shortest Paths in Unweighted Graphs

For a vertex v at level ℓ, its shortest $s \rightsquigarrow v$ path will include ℓ edges





"shortest path tree"

To recover all shortest paths from s, enough to keep track of last edge into each vertex, which was an incoming edge from previous level.

For BFS, the edge that discovered the vertex works.

Coding Demo: BFS with `PathInfo`

Reconstructing Shortest Paths

Complete the definition of this method to reconstruct the shortest path.

```
/** Stores the `level` of this vertex in the BFS and the label of the `prev` vertex whose outgoing
  *  edge discovered this vertex. */
record PathInfo(int level, String prev) { ... }

/** Reconstructs and returns the shortest path from the vertex with label `srcLabel` to the vertex
  *  with label `dstLabel` using the given `info` map produced by BFS. */
static List<String> reconstructPath(Map<String,PathInfo> info, String srcLabel, String dstLabel) {




}
```

Reconstructing Shortest Paths

```
record PathInfo(int level, String prev) { ... }

static List<String> reconstructPath(Map<String,PathInfo> info, String srcLabel, String dstLabel) {
    List<String> path = new LinkedList<>();
    path.add(dstLabel);
    while (!path.getFirst().equals(srcLabel)) {
        path.addFirst(info.get(path.getFirst()).prev());
    }
    return path;
}
```

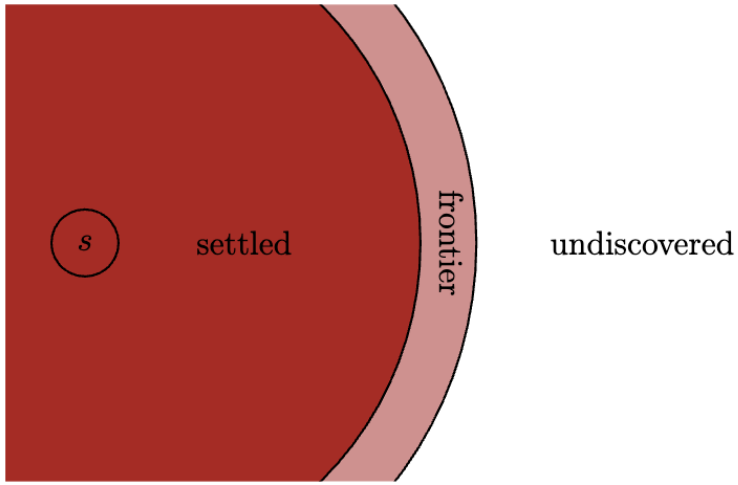
loop inv: path is a shortest path from path.getFirst() to dst vertex

incoming edge to path.getFirst()

Key Ideas of Unweighted Shortest Paths

1. At any point in the algorithm, our `discovered` map records the shortest (known) path distance to every node that we've discovered.

2. The next vertex to be removed from the frontier queue is always the *unvisited* vertex with the lowest level.

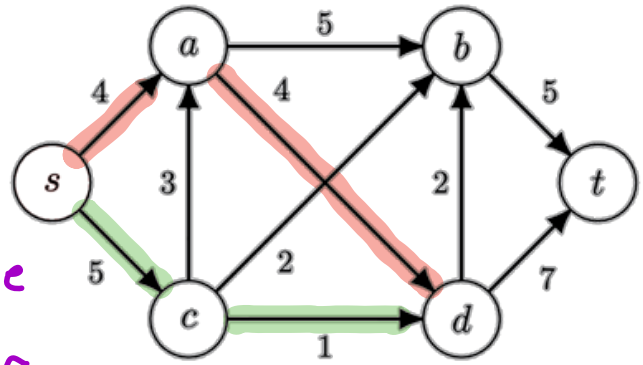


3. As soon as a vertex is visited/settled, we are guaranteed that we have located the shortest path to it from the source vertex, and this path contains only vertices that were previously settled.

Adapting to Weighted Graphs (non-negative weights)

Some similar ideas to BFS:

- track discovered, frontier vertices
  - record distance from s rather than level
- visit vertices in distance order from source
- during visit, use outgoing edges to discover new vertices and update distances in discovered map



Some modifications needed:

- short paths may contain more edges than longer paths
  - discovery order may be different than distance order
- we may locate a path later in the traversal that is shorter than an earlier path, need a way to update

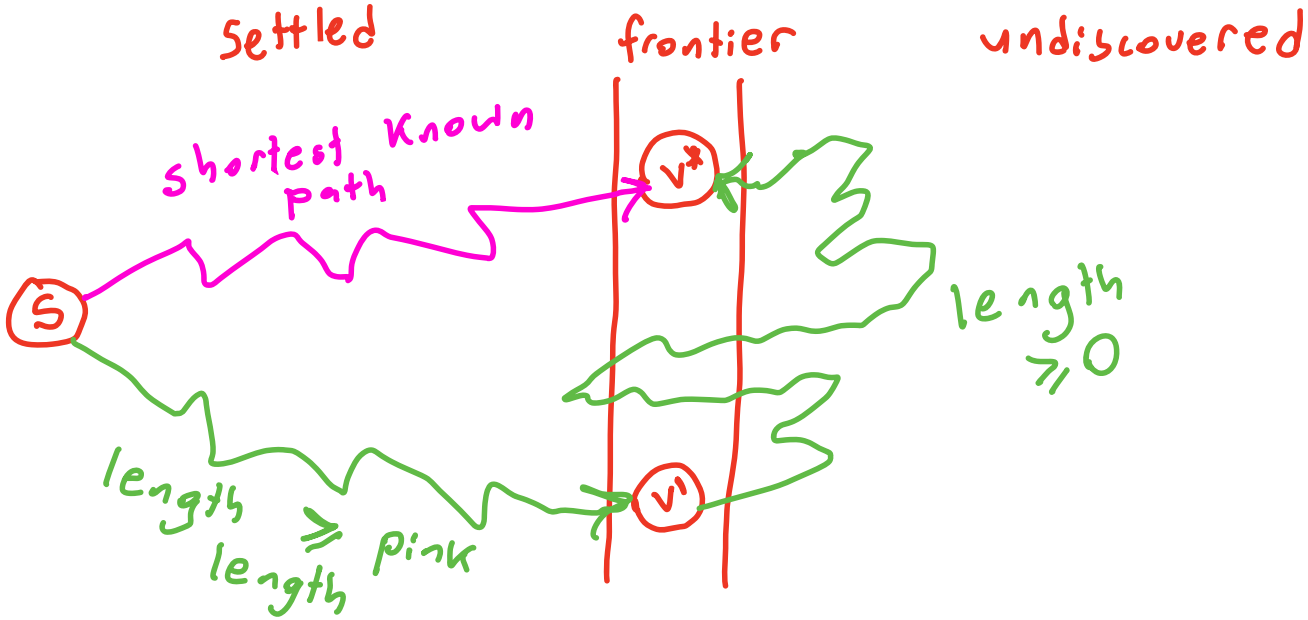
Dijkstra's Invariant

At the start of each main loop iteration:

- have located shortest path to all settled vertices
  * shortest known path to closest frontier vertex v^* is the true shortest path.

Proof sketch:

Suppose there were another (yet unknown) shorter path. It would need to cross through some other frontier vertex v'.



Dijkstra's Algorithm (High Level)

```
Initialize  discovered ← {source}  frontier ← {source}
while ( frontier is not empty) {
   v ← frontier vertex with min known distance from s
   for each outgoing edge (v,w) {
     if w is undiscovered, discover it and add to frontier
        with best known distance d(s,w) = d(s,v) + weight of
                                          edge (v,w)
     if w is discovered but we've found a shorter
      path to it, update its best known distance
   }
}
```

Properties of Dijkstra's Algorithm

Vertices are visited/settled in increasing order of distance (just like BFS).

Why? shortest path to any unsettled vertex v must pass through frontier, so v has distance $\geq$ any settled vertex

Best known distance to vertex only decreases as algorithm runs (when we find better path)

length of shortest path to closest frontier vertex $\leq$ length of shortest path to unsettled vertex v $\leq$ length of shortest known path to v

Poll Everywhere

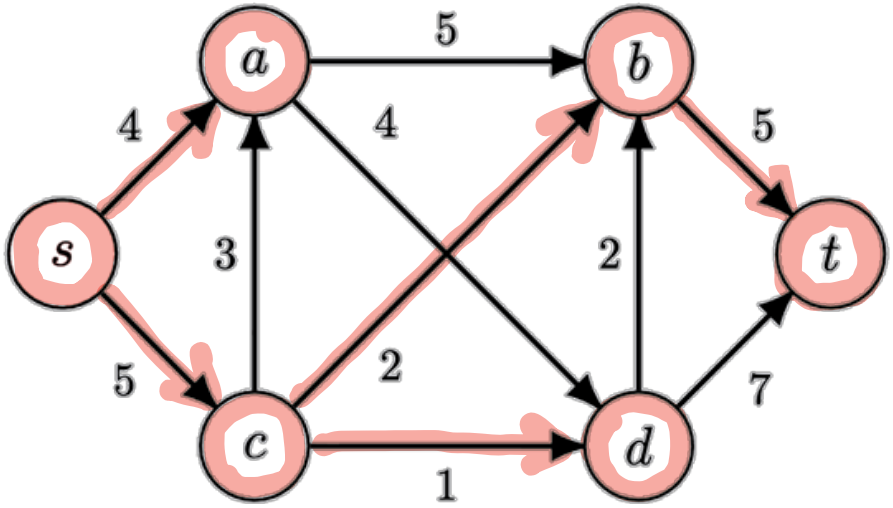
PollEv.com/2110fa25 text `2110fa25` to `22333`

Which data structure should we use to model the `frontier` in Dijkstra's algorithm?

- Queue **(A)**
- Stack **(B)**
- Priority Queue **(C)**
- Dynamic Priority Queue **(D)**

Coding Demo: Dijkstra's Algorithm

Dijkstra's Algorithm Walkthrough

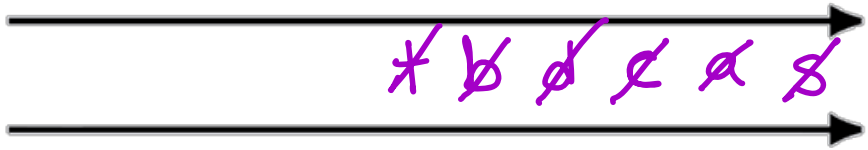


shortest path tree from s

discovered

vertex	s	a	b	c	d	t
distance	0	4	7	5	6	12
prev	null	s	c	s	c	b

frontier



Poll Everywhere

PollEv.com/2110fa25 text 2110fa25 to 22333

Which is the tightest bound on the space complexity of Dijkstra's algorithm?

discovered map and frontier Dynamic PQueue

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

Dijkstra's Algorithm Complexity

```
while(!frontier.isEmpty()) {
    V v = frontier.remove();
    for (WeightedEdge<V> edge : v.outgoingEdges()) {
        V neighbor = edge.head();
        double dist = discovered.get(v.label()).distance() + edge.weight();
        if (!discovered.containsKey(neighbor.label())) {
            discovered.put(neighbor.label(), new PathInfo(dist, v.label()));
            frontier.add(neighbor, dist);
        } else if (discovered.get(neighbor.label()).distance > dist) {
            discovered.put(neighbor.label(), new PathInfo(dist, v.label()));
            frontier.updatePriority(neighbor, dist);
}}}
```

O(|V|) iterations

O(log|V|) · O(|V|)

O(|E|) total iterations (across all outer loop iterations)

O(1) · O(|E|)

O(1) · O(|E|)

O(1) · O(|V|)

O(log|V|) · O(|V|)

O(1) · O(|E|)

O(1) · O(|E|)

O(log|V|) · O(|E|)

O(|E| · log|V|)