Lecture 23: Shortest Paths

CS 2110

April 16, 2026

Today's Learning Outcomes

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

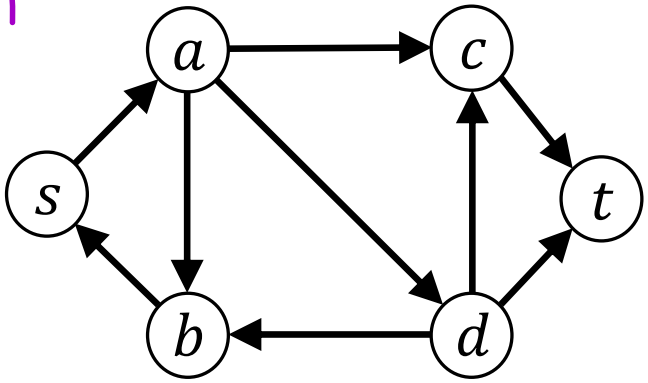
92. Visualize the state of a graph traversal at a given point of its execution, describing each node as either undiscovered, discovered, visited, and/or settled.

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.

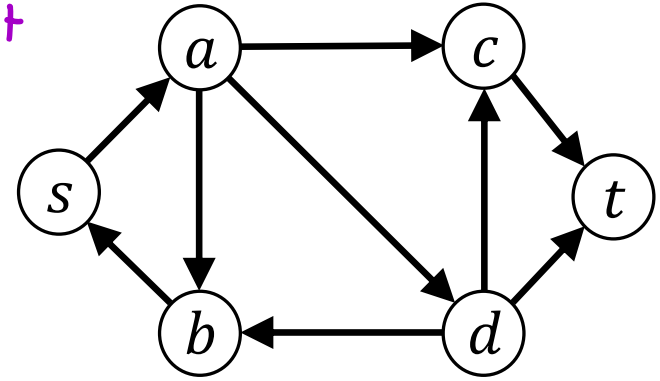
Think in phases!

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

$$s \xrightarrow{1} u_1 \xrightarrow{2} u_2 \cdots \xrightarrow{\ell-1} u_{\ell-1} \xrightarrow{\ell} v$$



Coding Demo: BFS with `PathInfo`

Reconstructing Shortest Paths

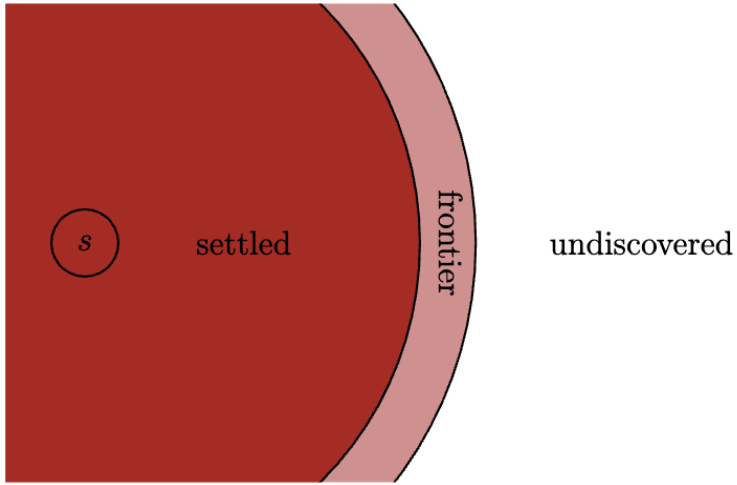
Complete the definition of this method to reconstruct the shortest path.

```
/** 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) {




}
```

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.

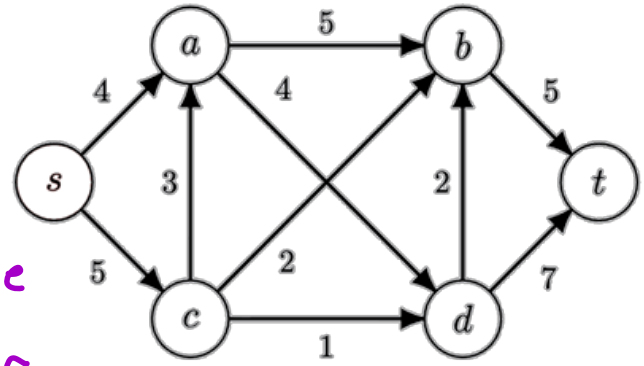
Adapting to Weighted Graphs (non-negative weights)

Some similar ideas to BFS:

- track discovered, frontier vertices

- visit vertices in distance order from source

- during visit, use outgoing edges to discover new vertices



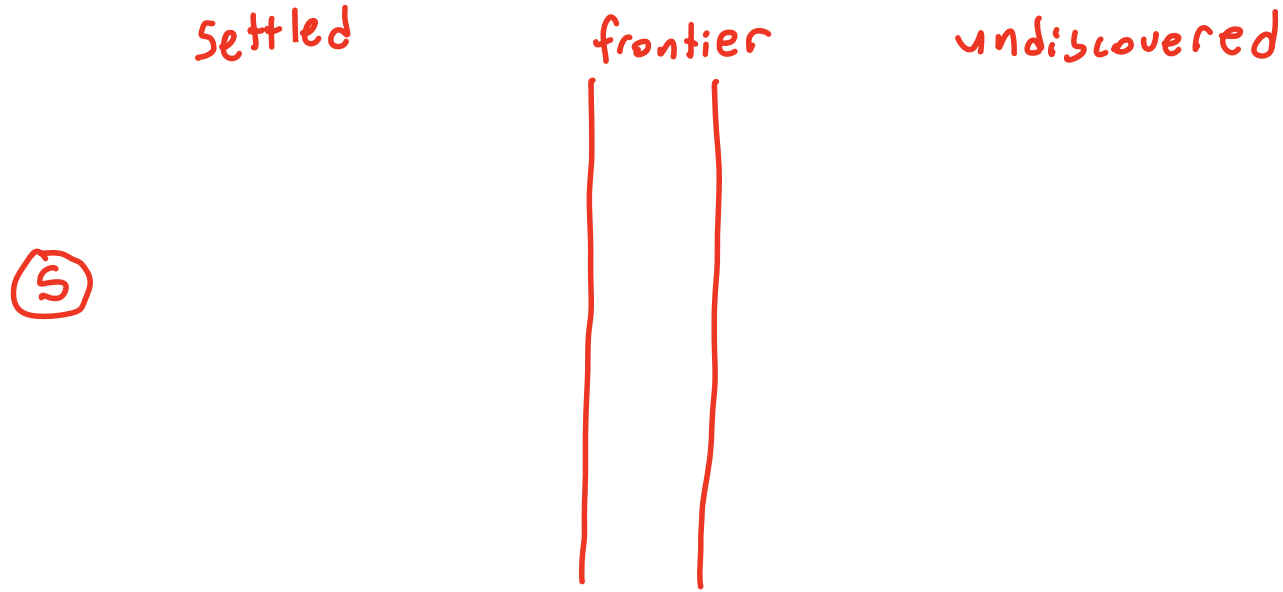
Some modifications needed:

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
   }
}
```

Dijkstra's Invariant

At the start of each main loop iteration:



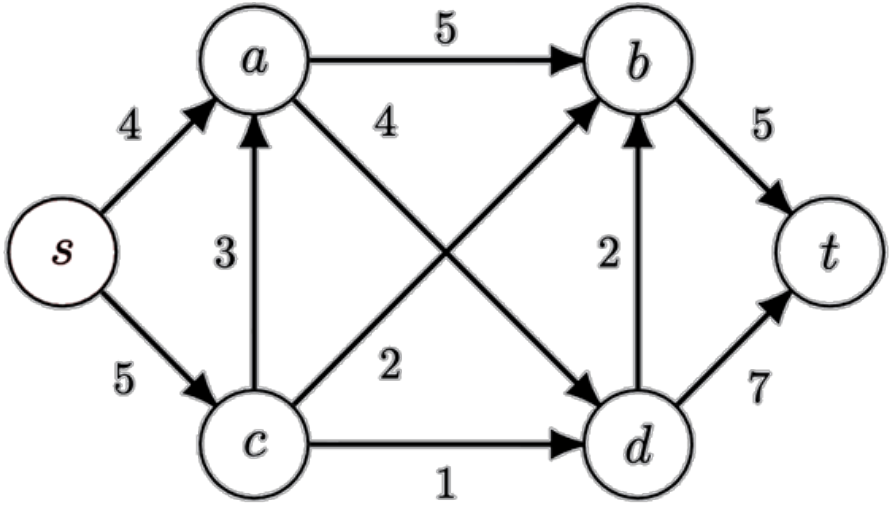
Properties of Dijkstra's Algorithm

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

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

Coding Demo: Dijkstra's Algorithm

Dijkstra's Algorithm Walkthrough



discovered

vertex	s	a	b	c	d	t
distance						
prev						

frontier

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);
}}}
```