Parallel And Sequential Data Structures and Algorithms

Shortest Paths and Dijkstra's Algorithm

Announcements

- **Midterm Two** this Friday!
  - Probability/Randomization
  - BSTs, Treaps, Augmented BSTs
  - DFS and BFS (short answers only)
  - 80-minutes long
  - Two practice midterms are available on Ed

Learning Objectives

- Review **BFS** / unweighted shortest paths
- Define **shortest paths** in weighted graphs
- See **Dijkstra's algorithm** for shortest paths

Shortest Paths

Unweighted versus Weighted

Definition (Length): The length of a path is the number of edges in it

Definition (Path Weight): In a weighted graph, the weight of a path is the sum of the weights of its edges

Note: Some books/people use "length" to mean weight, which causes confusion, so we will use length to mean number of edges and weight to mean sum of weights.

Unweighted Shortest Paths

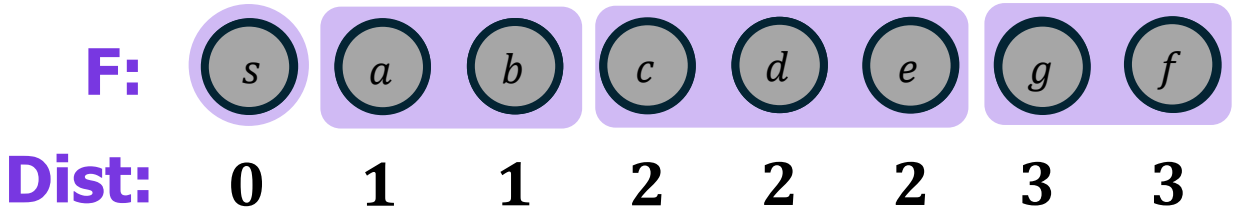
Definition (Shortest Path Length): Given a pair of vertices s and t in an unweighted graph, the shortest path length from s to t is the minimum number of edges of any path from s to t.

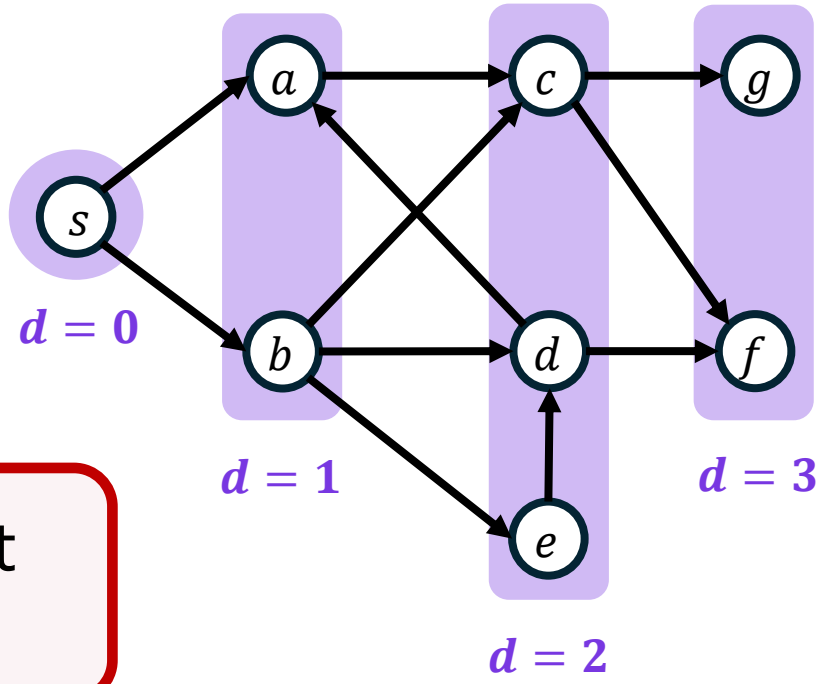
Recall breadth-first search from last lecture. BFS finds the shortest path lengths in an **unweighted** graph for a given s and all t

Parallel BFS

Recall: Parallel BFS explores vertices **layer by layer**

The current layer is called the **frontier**:





Sequential dependencies between frontiers, but substantial parallelism **within** each frontier

Weighted Shortest Paths

Definition (Shortest Path Weight): Given a pair of vertices s and t in a weighted graph, the shortest path weight from s to t is the minimum weight of any path from s to t.

- We will use the notation $\delta(s, t)$ to denote the shortest path weight for the path from s to t
- If no s-t path exists, we define $\delta(s, t) = \infty$

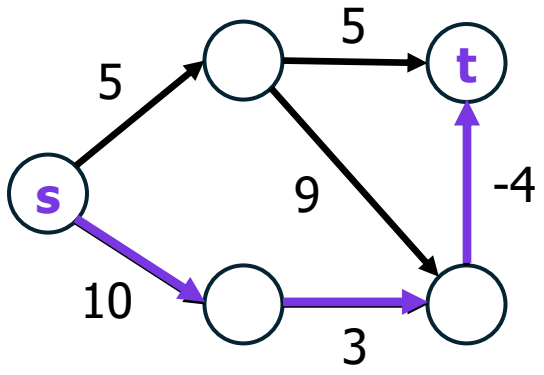
Are there any restrictions we need to place on edge weights?

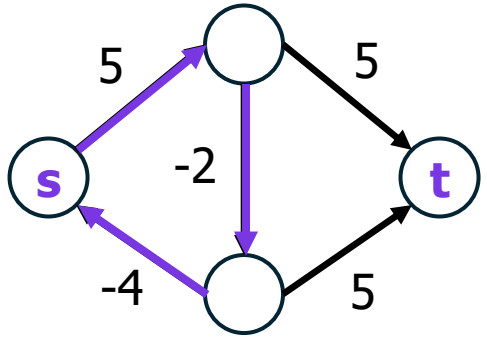
Weight Restrictions

- BFS only works finds shortest paths in **unweighted** graphs, or equivalently, graphs where every edge has the same weight

Some algorithms only work in graphs with **non-negative weights**.

- Should we allow negative weights?





Negative-Weight Cycles

In the presence of a **negative-weight cycle**, a shortest s-t path **may** not exist (if the cycle is reachable from s and can reach t)

- A shortest-path algorithm for non-negative weights will always fine a valid answer (provided that s can reach t)
- For negative weights, we have a choice:
  - Detect a negative-weight cycle and report invalid paths
  - Just silently return bogus answers

Shortest Path Properties

Question: Can a shortest path contain a cycle?

- It can not contain a negative-weight cycle, because then there would not exist a shortest path
- It can not contain a positive-weight cycle, because that would make the weight larger
- It **can** contain a zero-weight cycle, because that doesn't change the weight

If a path contains a zero-weight cycle, we can delete it, and the weight remains unchanged

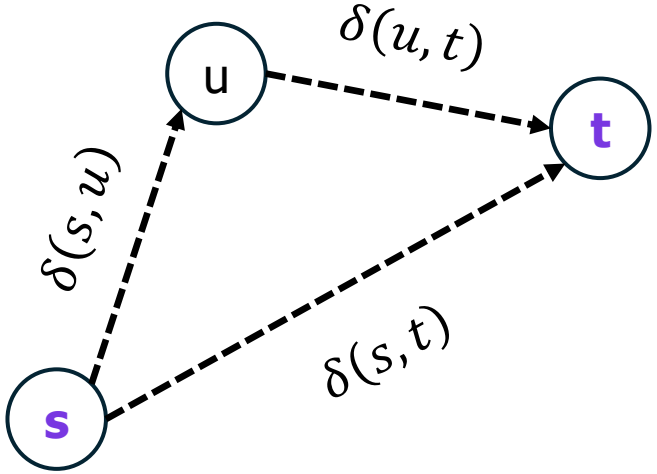
Theorem: If a shortest s-t path exists, then a simple (cycle-free) shortest s-t path exists.

The Triangle Inequality

- What is the relationship between $\delta(s,t)$ and $\delta(s,u)$ and $\delta(u,t)$?

Theorem (Triangle Inequality):
For any vertices s, u, t in G, we have

$$\delta(s,t) \leq \delta(s,u) + \delta(u,t)$$



Proof: The path $s \rightsquigarrow u$ appended with the path $u \rightsquigarrow t$ is a path from s to t. So, either it's the shortest path or the shortest path is shorter.

Shortest Path Problems

Shortest Path Problems

Problem (Single-Pair Shortest Path): Given a pair of vertices s and t we want to find a shortest path from s to t

Problem (Single-Source Shortest Path): Given a vertex s, we want to find, for all $v \in V$, a shortest path from s to v

Problem (All-Pairs Shortest Path): We want to find, for all $u, v \in V$, a shortest path from u to v

The last two can be solved by n invocations of the one above it. But we can be more efficient by developing algorithms for each individually.

So Many Possibilities

- **Problem**: Single-pair, single source, all pairs
- **Weights**: Non-negative versus negatives, integers versus real
- **Direction**: Directed versus undirected
- **Properties**: Acyclic graphs, planar graphs, …
- **Density**: Dense versus sparse graphs

There are shortest path algorithms for most combinations of all of these.

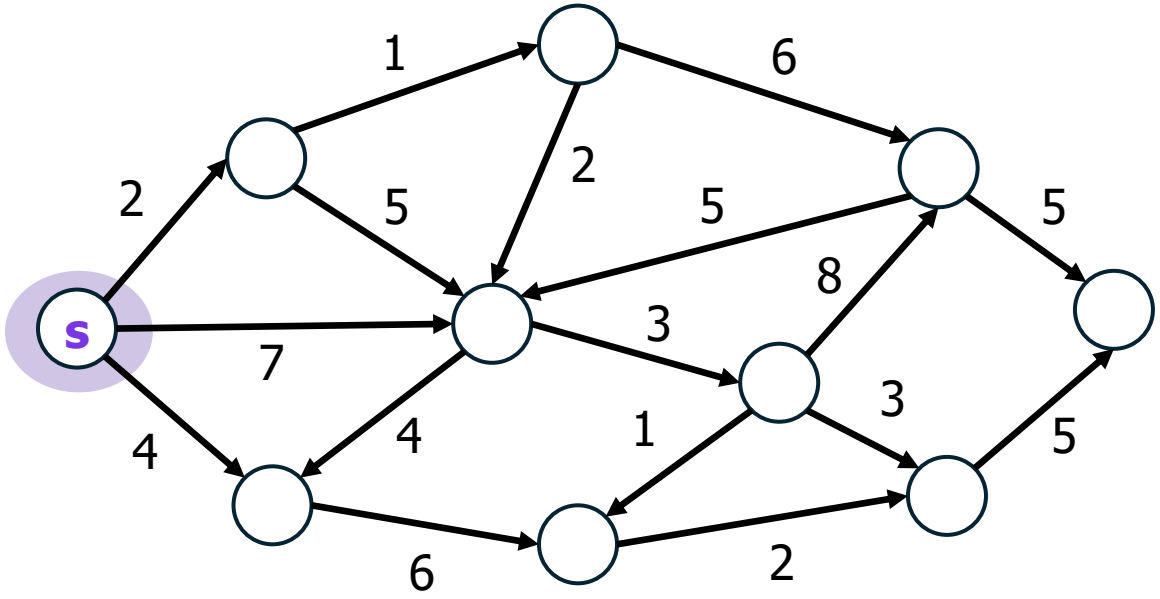
New algorithms are still being discovered! Last year: a new algorithm for SSSP on directed graphs with real non-negative edge weights.

SSSP with Non-Negative Weights (Dijkstra's Algorithm)

The Problem Today

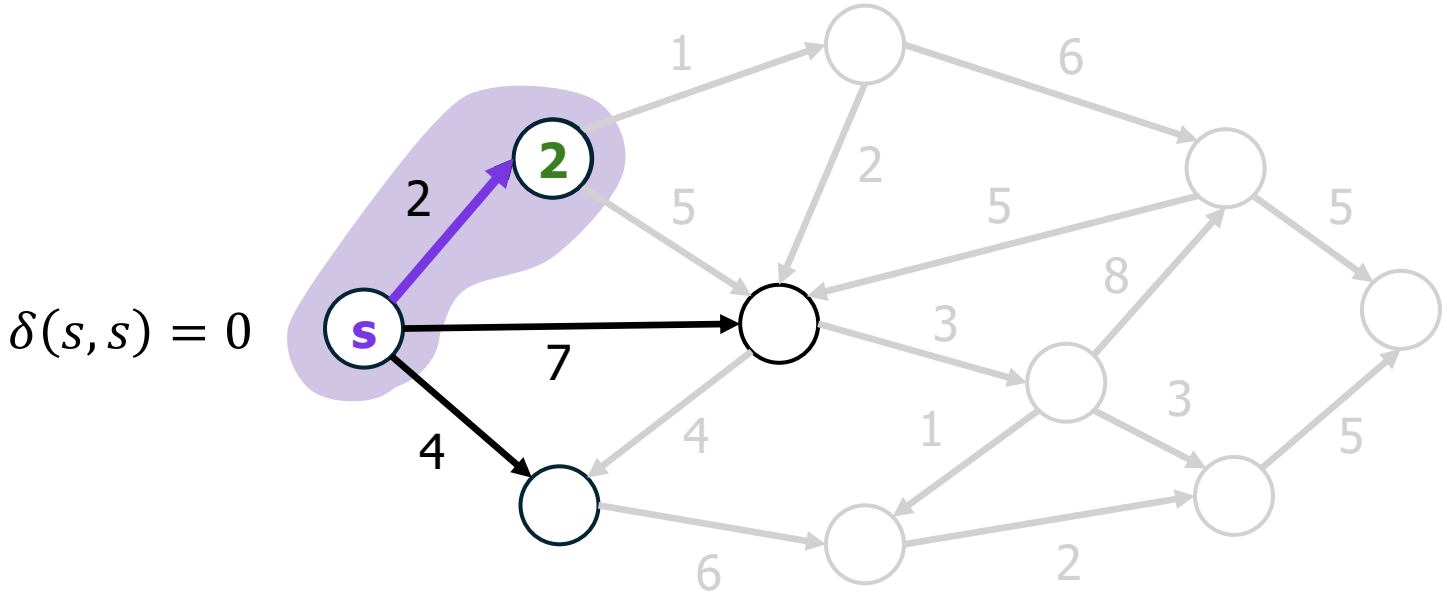
- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

$\delta(s, s) = 0$



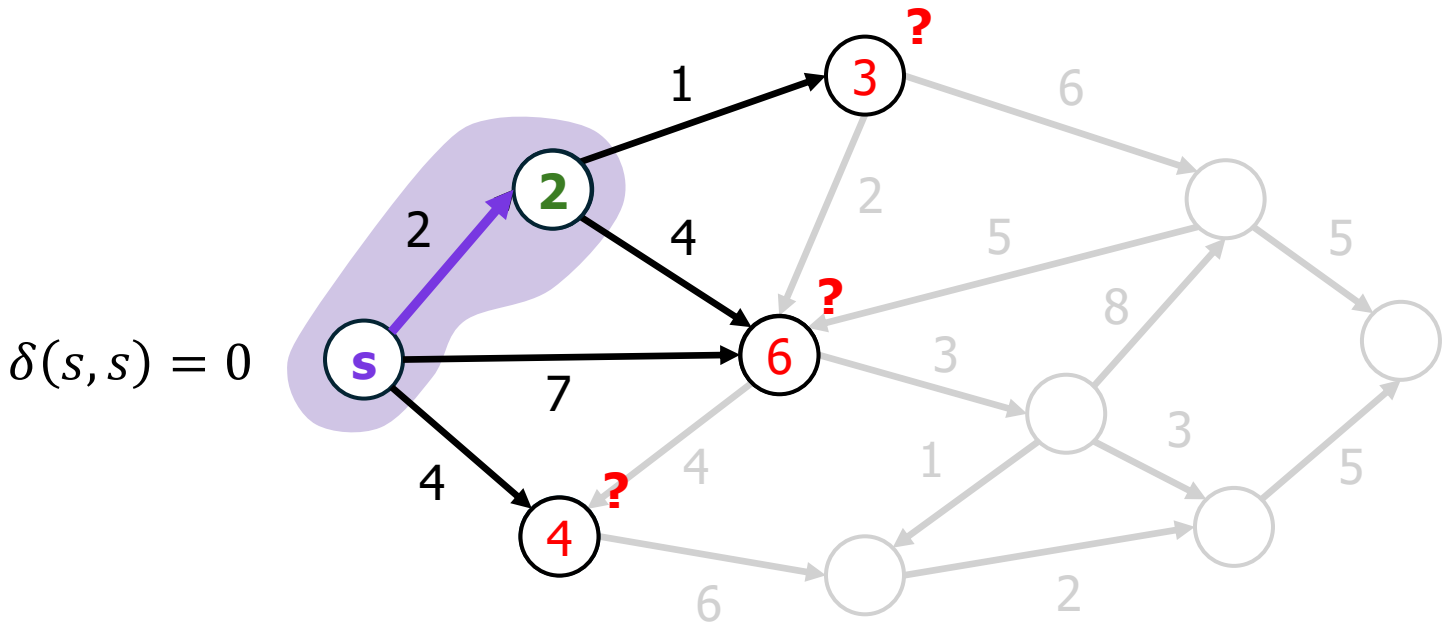
The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights



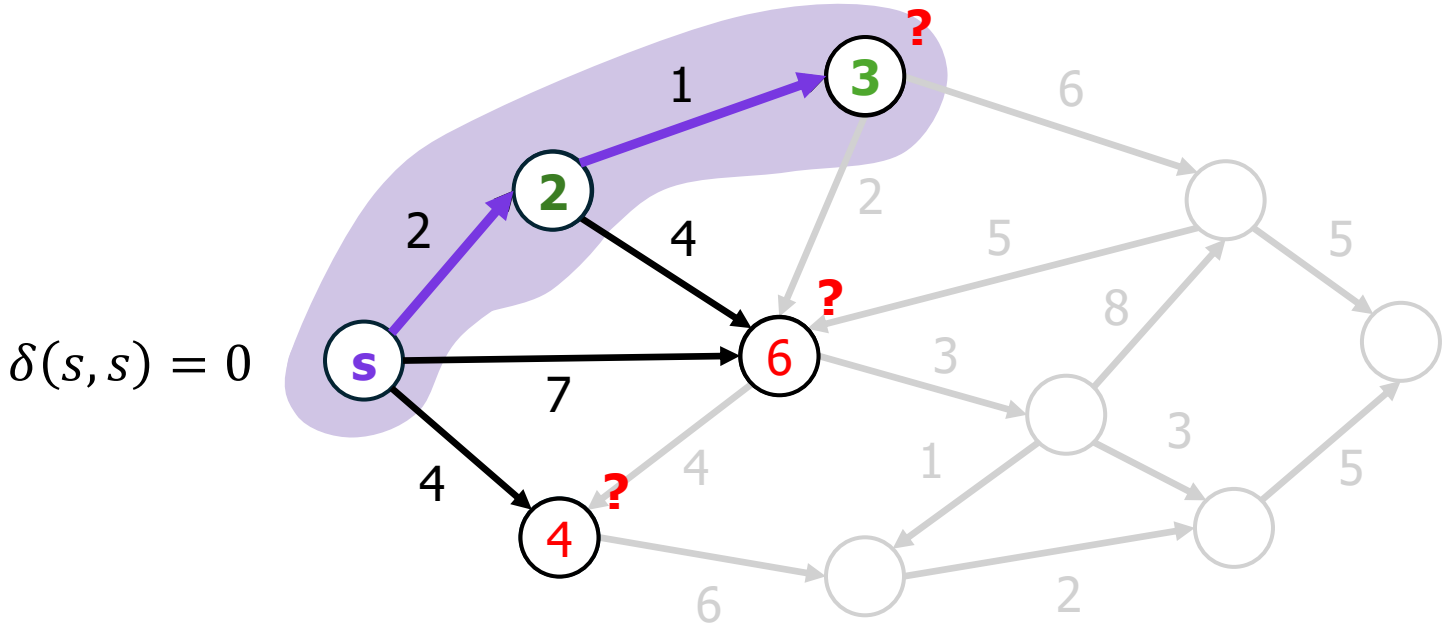
The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights



The Problem Today

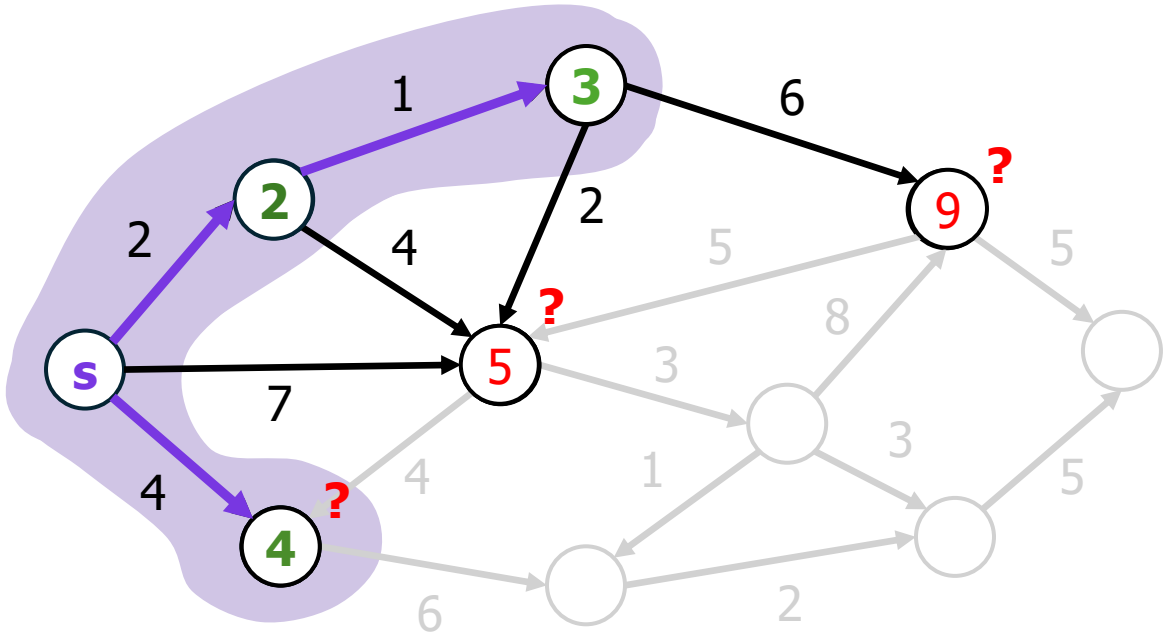
- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

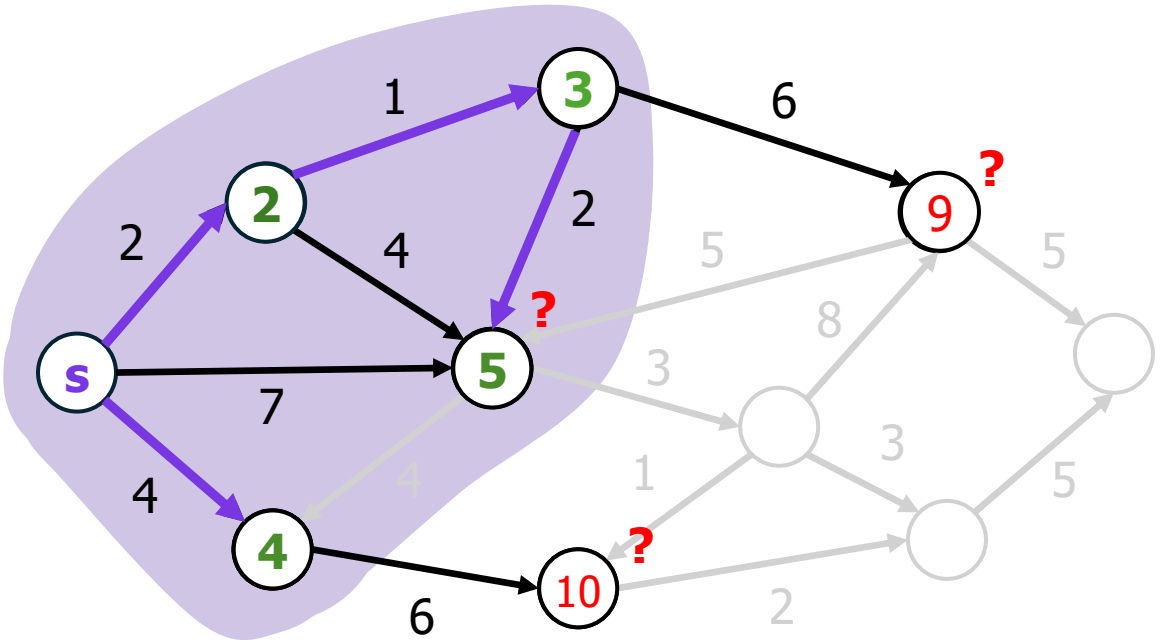
$\delta(s, s) = 0$



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

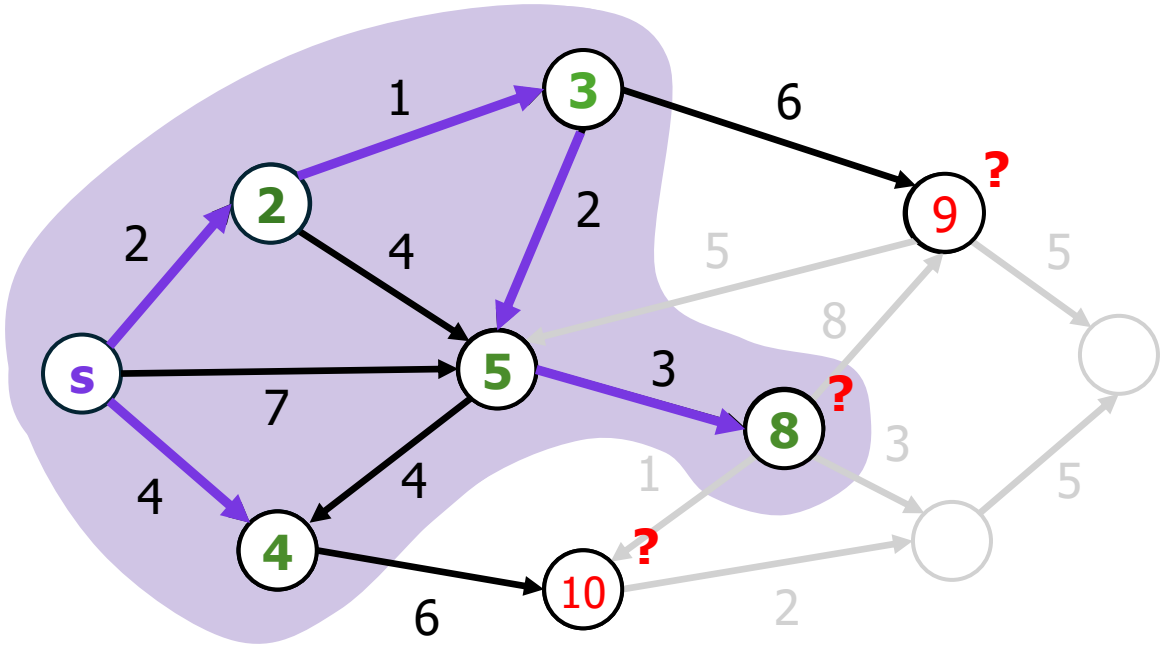
$\delta(s, s) = 0$



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

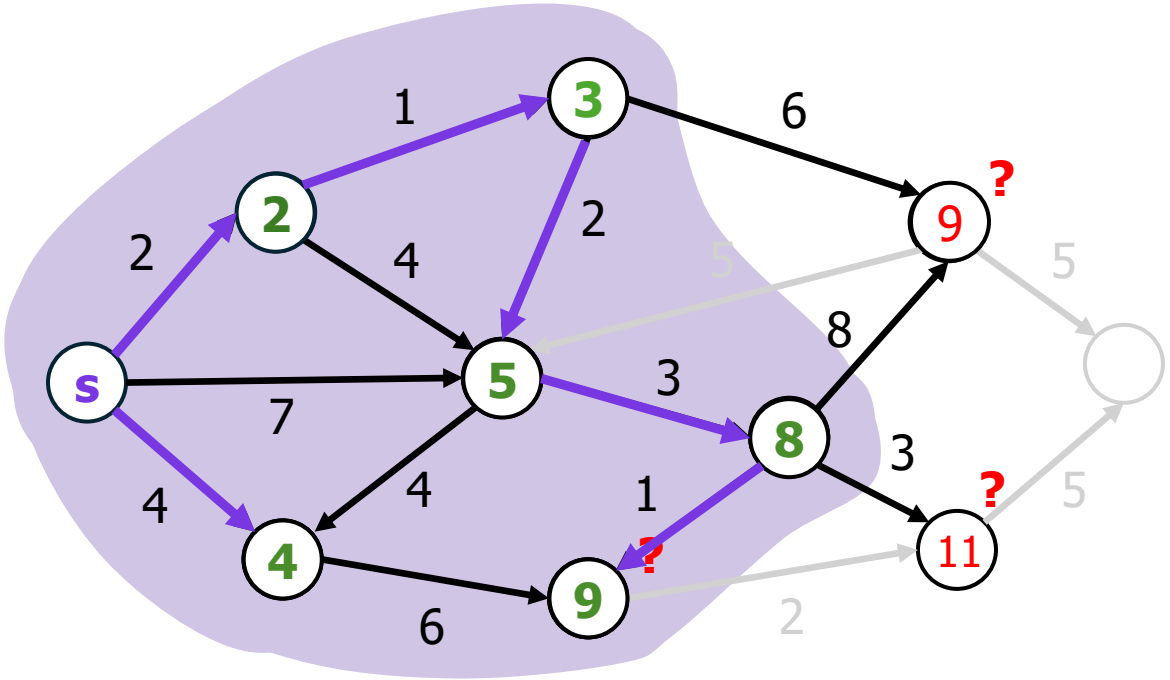
$\delta(s, s) = 0$



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

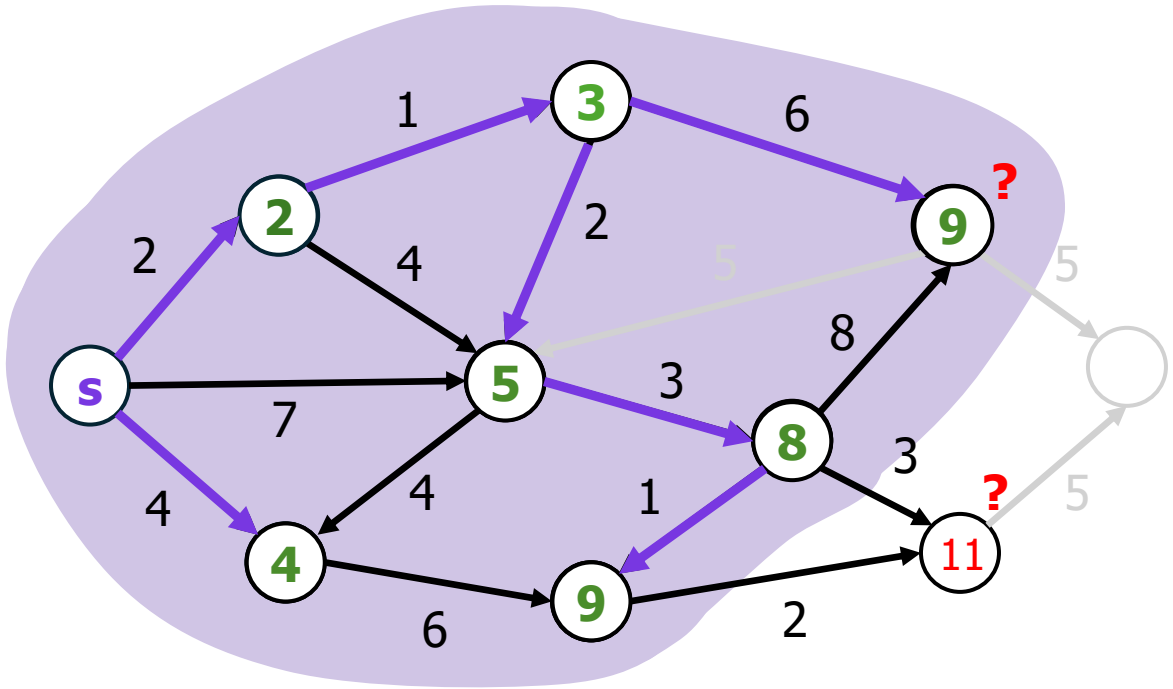
$\delta(s, s) = 0$



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

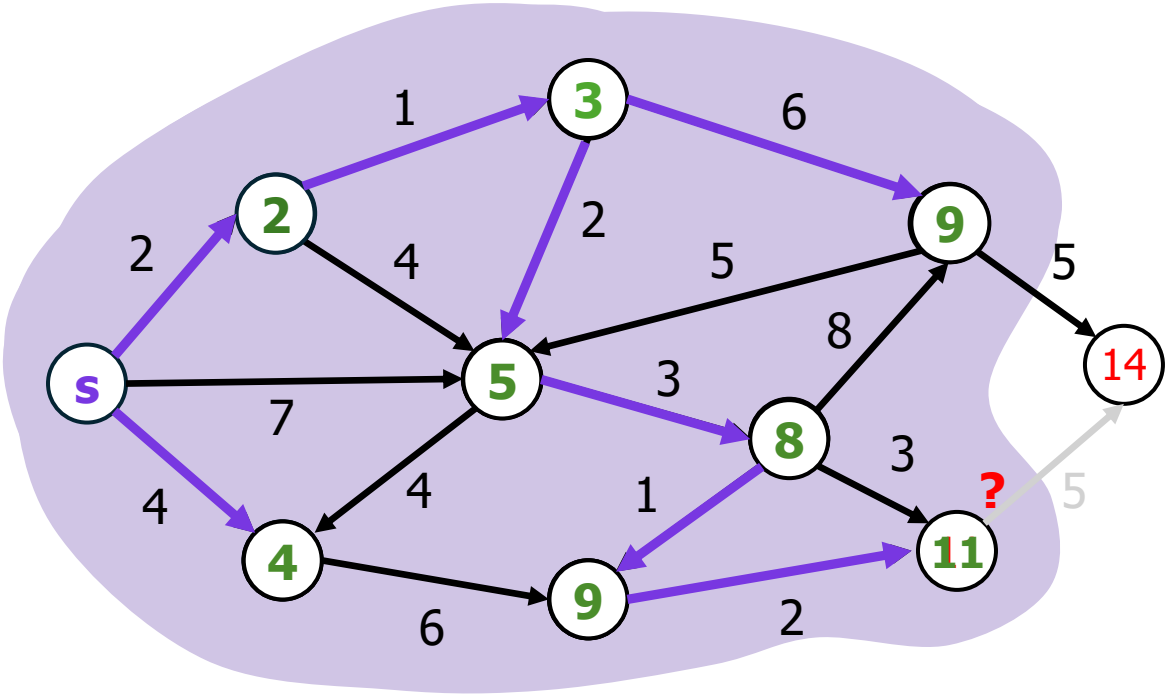
$\delta(s, s) = 0$



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

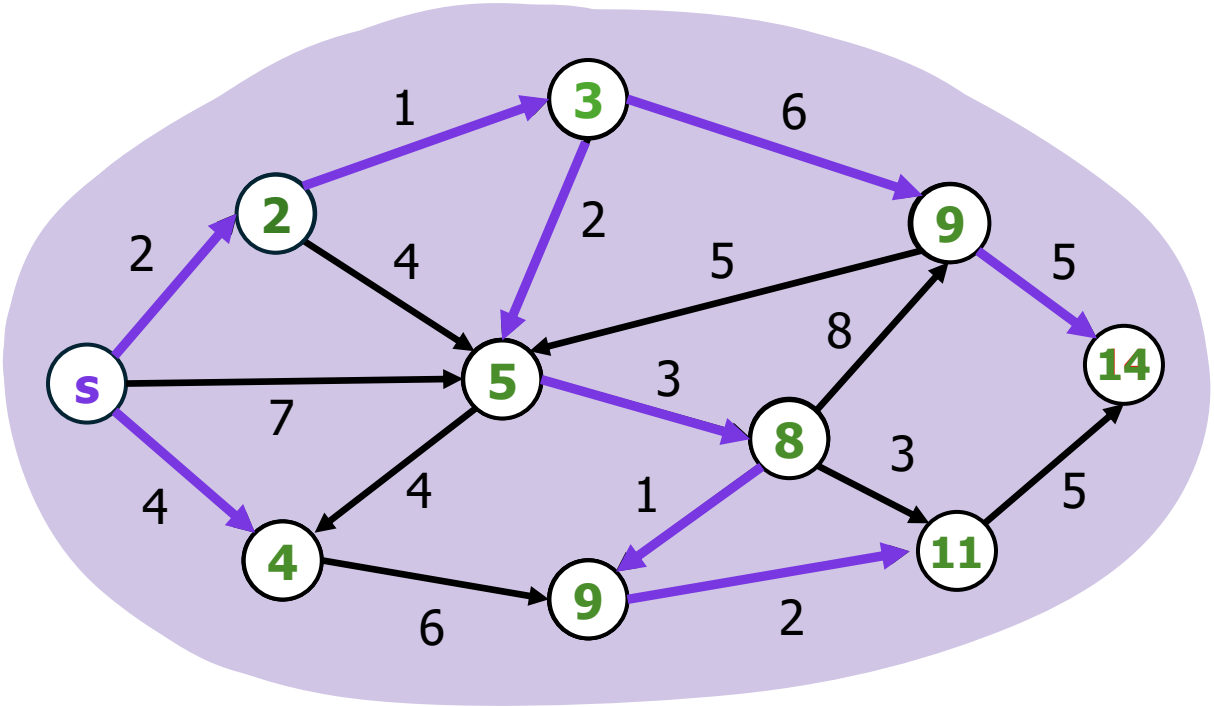
$\delta(s, s) = 0$



The Problem Today

- **Single-source** shortest paths
- Directed, weighted graph, **non-negative** real weights

$\delta(s, s) = 0$



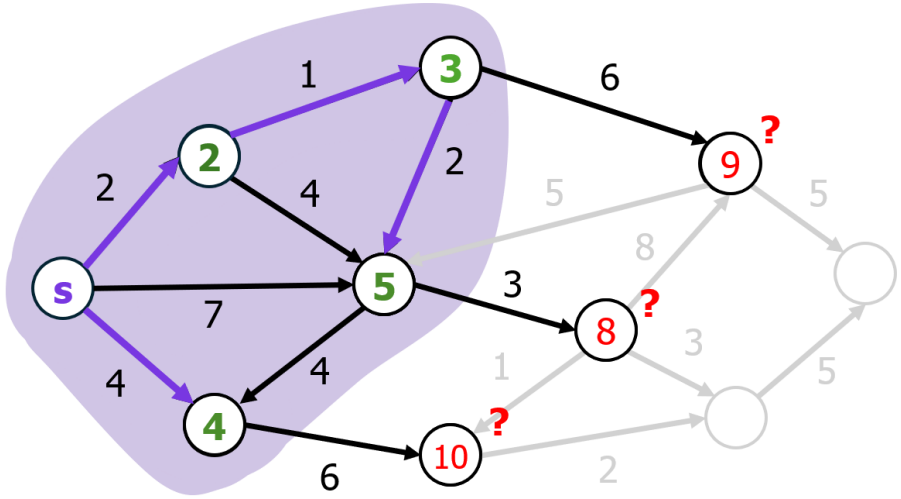
What Does That Algorithm Do?

We just inferred a **greedy property** for single-source shortest paths. The vertex with the shortest "candidate path" must be the next shortest path, because any alternate path would only have higher weight.

- This algorithm, when implemented with appropriate data structures for efficiency, is called **Dijkstra's Algorithm**.

High-Level Overview

- Our algorithm implicitly maintains two sets of vertices:
  - Vertices whose distance from s is known for certain (the purple blob)
  - Vertices whose distance from s is not known for certain (the others)



- For the second kind, we maintain a **distance estimate**: the weight of the shortest path to that vertex that we have discovered so far (say ∞ for undiscovered vertices)
- At each step, the vertex u with the lowest distance estimate is in fact the correct distance (due to the greedy property).

Data Structures

- Recall the key step: *At each step, the vertex u with the lowest distance estimate is in fact the correct distance*

? **What data structure** would make this efficient?

Answer*: A* ***priority queue***

- We will use a priority queue that keeps vertices sorted by their distance estimate (smallest first).
- At each step, we pop the minimum element and update the distance estimates of its neighbours

Dijkstra's Algorithm

```
type WeightedAdjList = sequence<sequence<(int, real)>>

fun dijkstra(adj : WeightedAdjList, s : int) -> sequence<real>:
  dist = [0 if u == s else ∞ for u in 0...|adj|-1]   // current best distance estimates
  done = [False for _ in 0...|adj|-1]                // vertices whose distances are finalized
  pq : priority_queue<(real, int)> = empty
  pq.insert((0, s))                                  // stores (distance estimate, vertex)

  while not pq.is_empty():
    d, u = pq.delete_min()
    if done[u]: continue
    done[u] ← True
    for v, w in adj[u]:
      new_dist = d + w
      if new_dist < dist[v]:
        dist[v] ← new_dist
        pq.insert((new_dist, v))

  return dist
```

Cost Analysis

Theorem (Dijkstra's Cost): On a simple graph with n vertices and m edges, Dijkstra's algorithm costs $O(m \log n)$ (work and span).

- *The cost is dominated by the priority queue operations*
- *Each edge of the graph can contribute one entry to the PQ*
- *So, the PQ has size at most m, so operations cost $O(\log m)$*
- *The graph is simple so $m \leq n^2$ and hence the cost is*

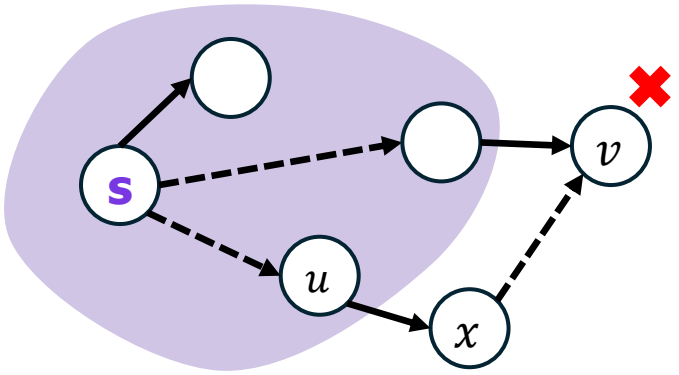
$$O(m \log m) = O(m \log n)$$

Correctness

Theorem (Dijkstra's Correctness): On a graph with no negative weights, Dijkstra's algorithm outputs correct distances from s in G.

Proof: AFSOC Dijkstra's algorithm gives a wrong distance

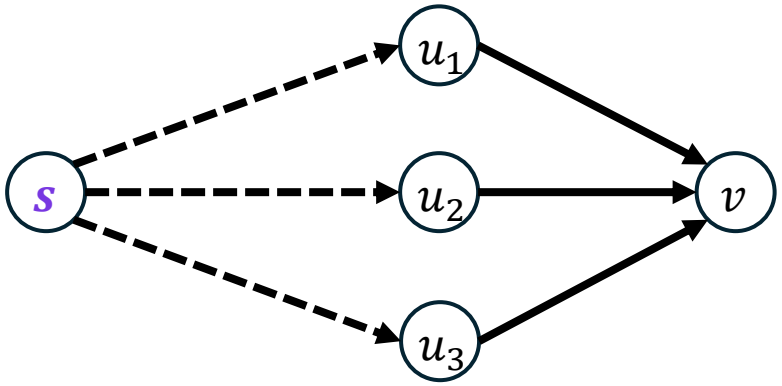
- *Let v be the first vertex discovered that has the wrong distance*
- *There must be a shorter path to v!*
- *Let u be the last visited vertex on that path*
- ***Since the weights are non-negative**, the distance estimate of x must be lower than v.*
- *If $x = v$, contradiction. Otherwise, Dijkstra would have visited x, not v, a contradiction.*



Shortest Paths in Directed Acyclic Graphs

Acyclic Graphs

- Dijkstra's Algorithm works for graphs with non-negative weights, and it can handle cycles just fine
- Assuming the graph has no cycles makes shortest paths simpler



$$\delta(s,v) = \min \begin{cases} \delta(s,u_1) + w(u_1,v) \\ \delta(s,u_2) + w(u_2,v) \\ \delta(s,u_3) + w(u_3,v) \end{cases}$$

In general, $\delta(s,v) = \min_{u \in N^-(v)} \big(\delta(s,u) + w(u,v)\big)$

Shortest Paths as a Recurrence

In general, $\delta(s, v) = \min_{u \in N^-(v)} \big(\delta(s, u) + w(u, v)\big)$

- This equation is still true even for cyclic graphs
- But there's a problem…
- You can't evaluate it because trying to evaluate the recursion would cycle forever (infinite recursion!)
- This problem does not exist on an acyclic graph, so in theory we can just evaluate this recurrence and we are done!

Evaluating the Recurrence

- Let $\mathrm{Dist}(v)$ denote the weight of the shortest path from s to v for all $v \in V$. Then consider the following recurrence

$$\mathrm{Dist}(v) = \begin{cases} 0, & \text{if } v = s \\ \min\limits_{u \in N^-(v)} \big(\mathrm{Dist}(u) + w(u,v)\big), & \text{otherwise} \end{cases}$$

- Evaluating this naively could take exponential time since it may enumerate every path in the graph!

Use **dynamic programming!**

Cost Analysis

Theorem (SSSP in a DAG): On a directed acyclic graph on n vertices and m edges, shortest paths can be found in $O(n+m)$ cost

Proof: There are n subproblems (one per vertex)

- *For each subproblem, the cost of evaluating it is the in-degree of that vertex, i.e., $|N^-(v)|$*
- *Therefore, the sum of costs of every subproblem is the sum of the in-degrees in the graph*
- *Each in-degree comes from one edge, so that cost is $O(m)$*
- *Therefore, the total cost is $O(n+m)$*

Summary

- Shortest paths in weighted graphs have many algorithms depending on the constraints of the graph
- **Dijkstra's Algorithm** costs $O(m \log n)$ (completely sequential) and only works for graphs with non-negative weights
- For directed graphs with no cycles (**directed acyclic graphs**), we can find shortest paths in $O(n + m)$ cost, and this works on graphs with negative weights