Announcements

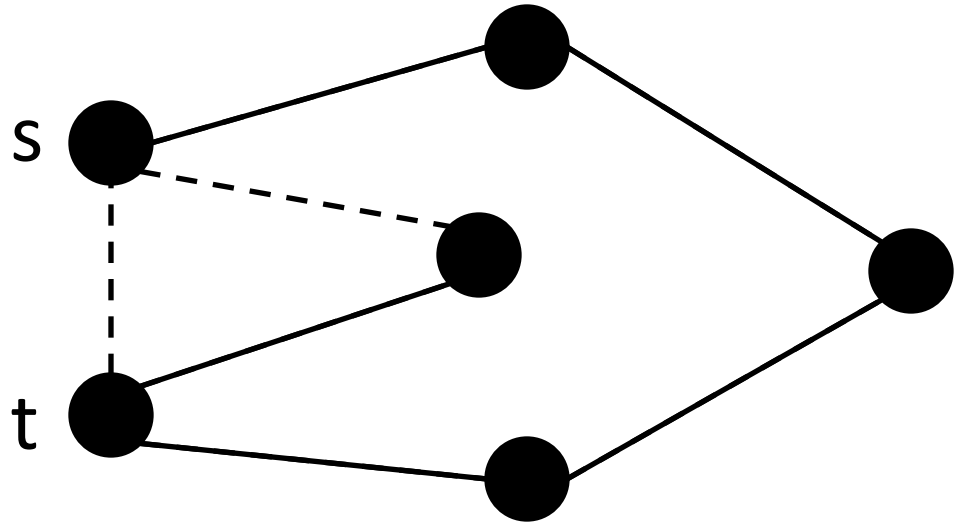
- Homework 1 Due today
- No class/discussion/OH on Monday
- Exam 1 Next Friday
  - 3 Questions in 1 hour
  - During class time unless you cannot make it
  - Cover Chapter 3 + BFS
  - Using gradescope time limited submission
  - Exam instructions assignment

Paths in Graphs (Ch 4)

- Breadth First Search
- Dijkstra
  - Priority Queues
- Bellman-Ford
- Shortest Paths in DAGs

Motivation

`DFS`/`explore` allow us to determine *if* it is possible to get from one vertex to another, and using the DFS tree, you can also find *a* path. But this often is not an efficient path.



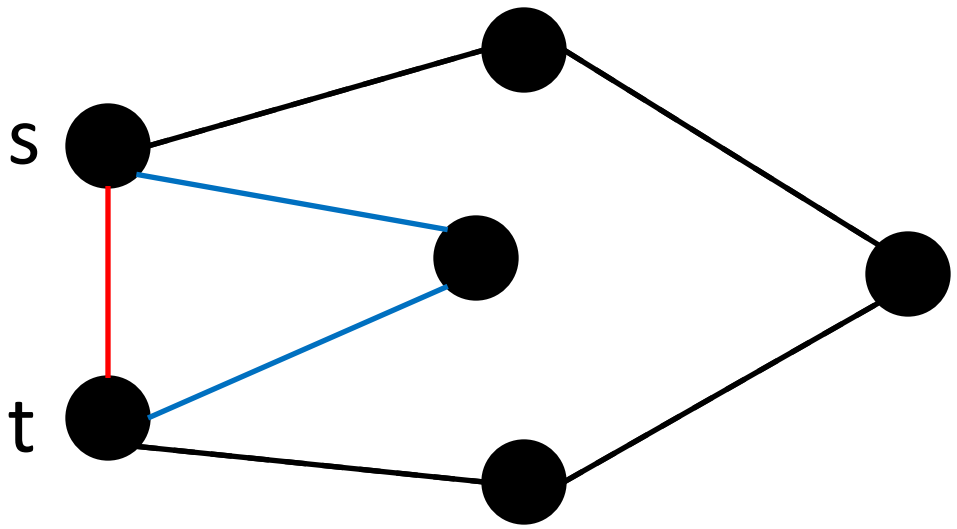
Goal

Problem: Given a graph G with two vertices s and t in the same connected component, find the *best* path from s to t.

What do we mean by best?

- Least expensive
- Best scenery
- Shortest
- For now: fewest edges

Example



4 edges

2 edges

1 edge

Best is 1 edge.

Question: Shortest Path Length

What is the shortest path length from s to t in the graph below?

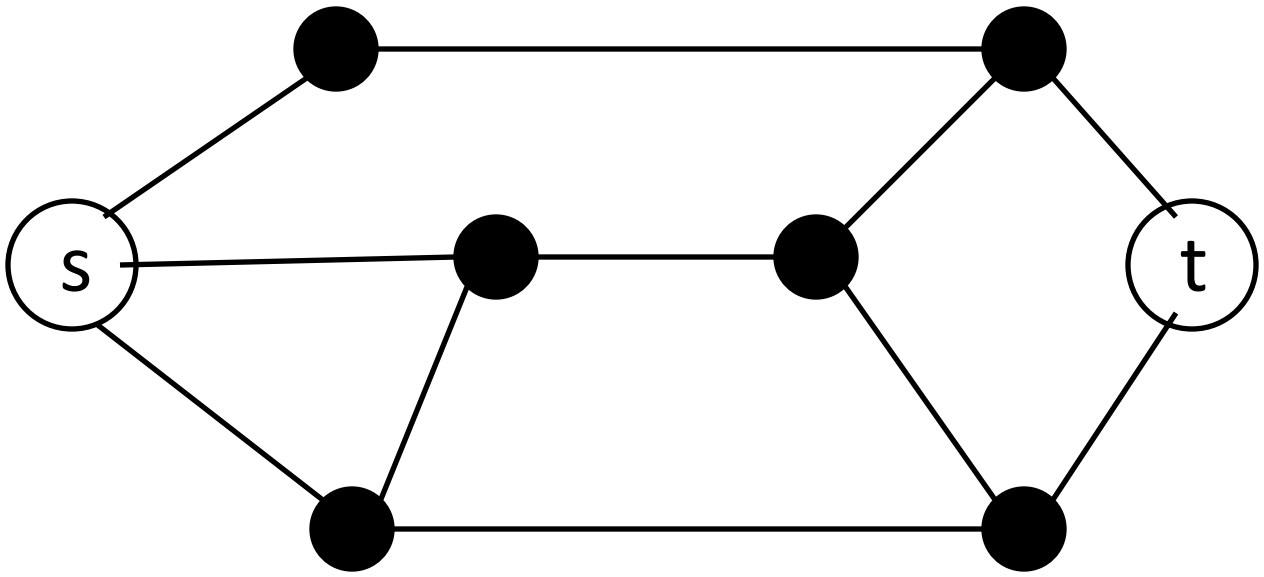
A) 1

B) 2

C) 3

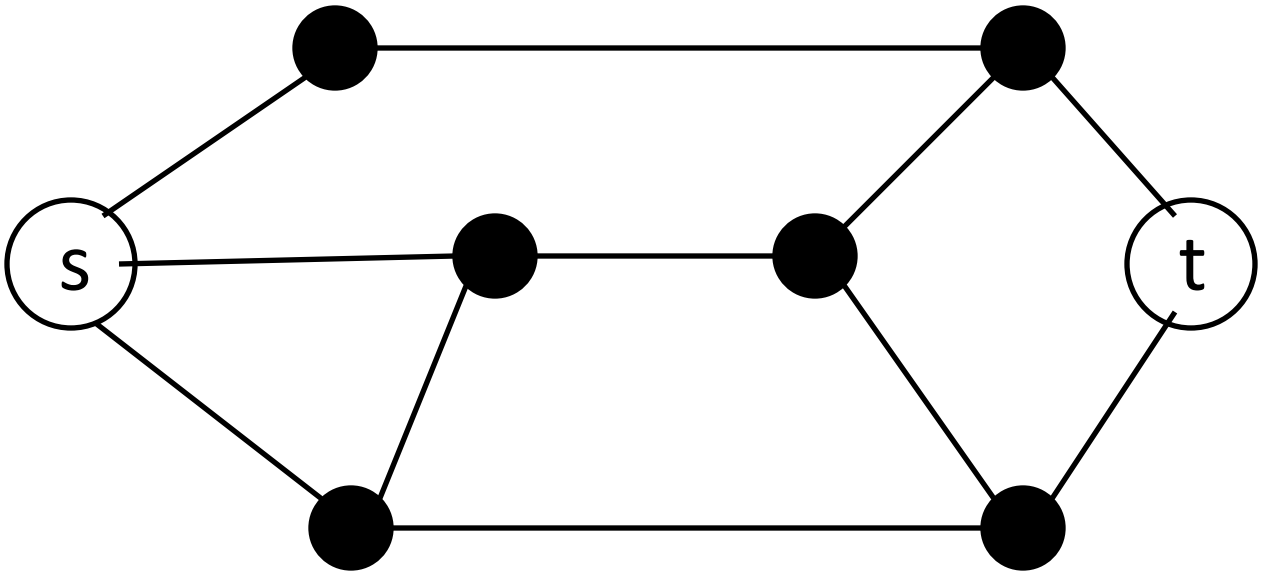
D) 4

E) 5



How do you *know*?

It is not hard to convince yourself that the shortest s-t path below has 3 edges, but how do we *know* there is nothing better?



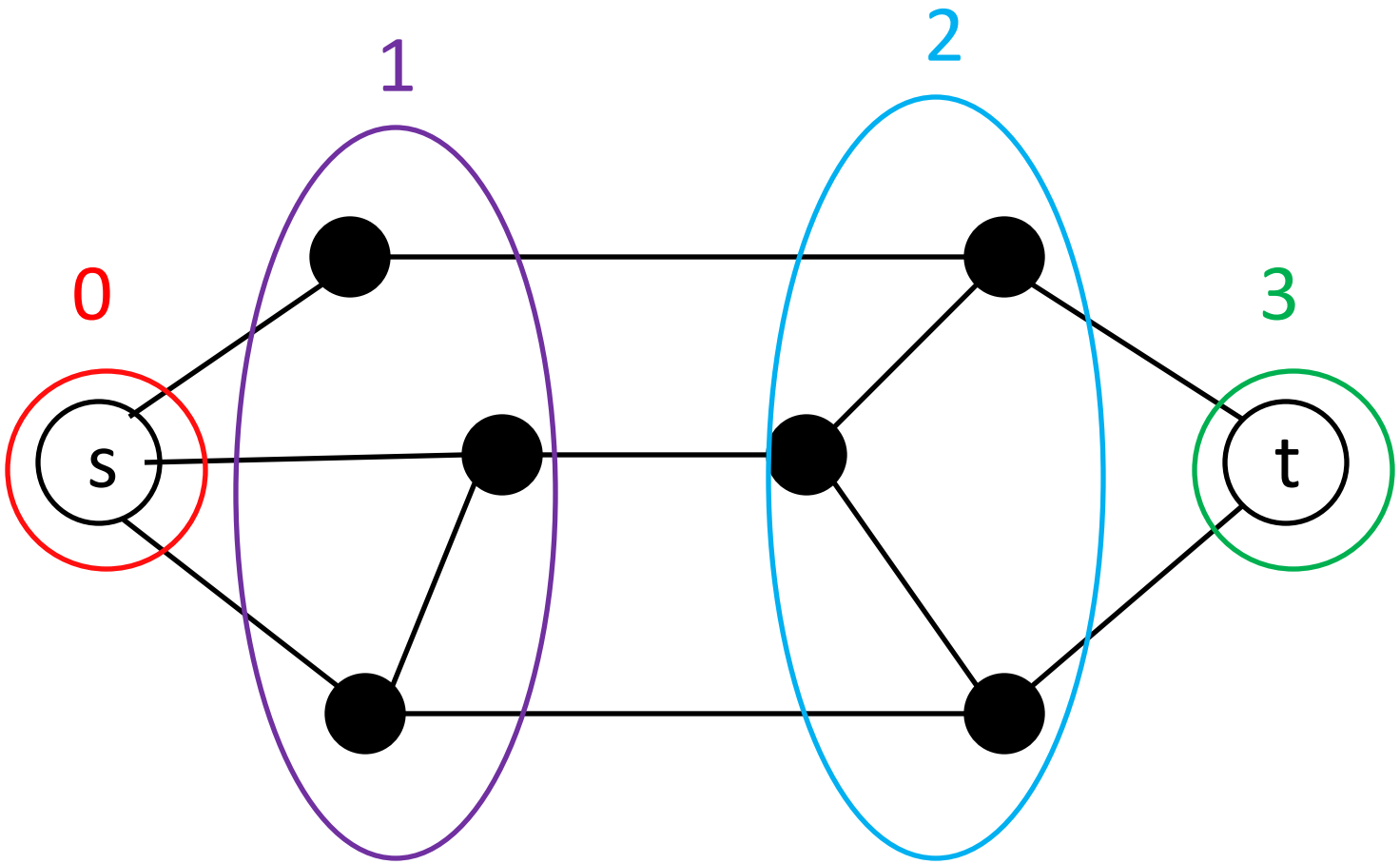
Observation

If there is a length ≤d s-v path, then there is some w adjacent to v with a length ≤(d-1) s-w path.

Proof: w is the next to last vertex on the path.

This means that if we know all of the vertices at distance ≤(d-1), we can find all of the vertices at distance ≤d.

Example



Algorithm Idea

For each d create a list of all vertices at distance d from s.

- For d=0, this list is just {s}.
- For larger d, we want all new vertices adjacent to vertices at distance d-1.

Breadth First Search

```
ShortestPaths(G,s)
  For v ∈ V, dist(v) ← ∞
  Initialize Array A
  A[0] ← {s}
  dist(s) ← 0
  For d = 0 to n
    For u ∈ A[d]
      For (u,v) ∈ E
        If dist(v) = ∞
          dist(v) ← d+1
          add v to A[d+1]
          v.prev ← u
```

Initialize Queue Q

Q.enqueue(s)

While Q not empty

u ← Q.dequeue()

dist(v) ← dist(u)+1

Q.enqueue(v)

What if dist(v) undefined at end?

Algorithm goes through A[0], A[1],… in order. Can just use a queue.

Keep track of paths

Runtime

```
BFS(G,s)
   For v ∈ V, dist(v) ← ∞
   Initialize Queue Q
   Q.enqueue(s)
   dist(s) ← 0
   While(Q nonempty)
      u ← front(Q)
      For (u,v) ∈ E
         If dist(v) = ∞
            dist(v) ← dist(u)+1
            Q.enqueue(v)
            v.prev ← u
```

O(|V|)

O(|V|) iterations

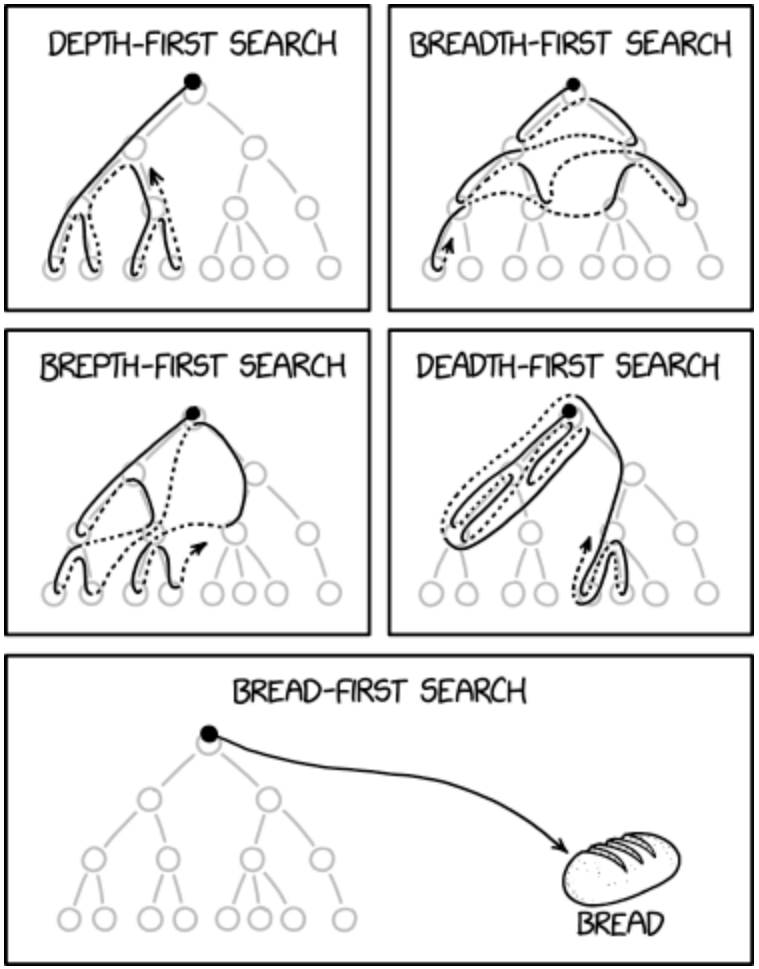
O(|E|) total iterations

Total runtime:
O(|V|+|E|)

DFS vs BFS

- Processed vertices (visited, dist < ∞)
- For each vertex, process all unprocessed neighbors
- Difference:
  - DFS uses a stack to store vertices waiting to be processed
  - BFS uses a queue
- Big effect
  - DFS goes depth first – very long path
  - BFS is breadth first – visits all side paths

DFS vs BFS vs Others?



https://xkcd.com/2407/

Edge Lengths

The number of edges in a path is not always the right measure of distance. Sometimes, taking several shorter steps is preferable to taking a few longer ones.

We assign each edge (u,v) a non-negative length $\ell(u,v)$. The length of a path is the sum of the lengths of its edges.

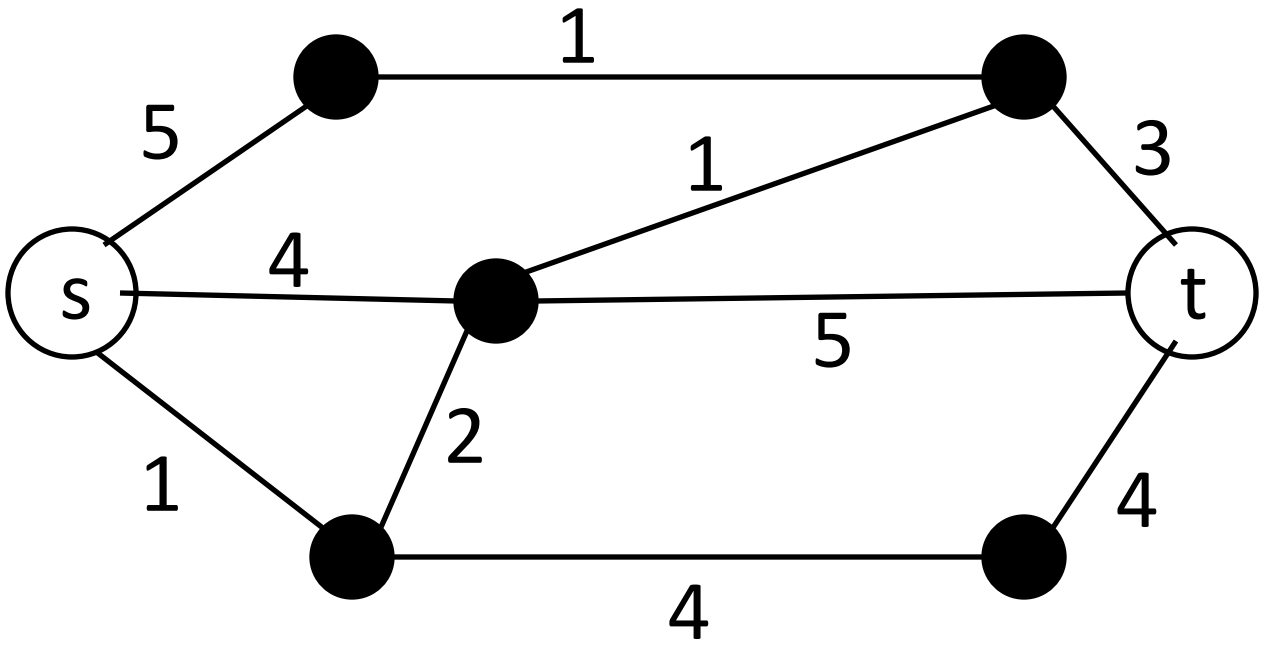
Problem: Shortest Paths

Problem: Given a Graph G with vertices s and t and a length function ℓ, find the shortest path from s to t.

Question: Shortest Path

What is the length of the shortest s-t path below?

A) 5
B) 6
C) 7
D) 8
E) 9



1+2+1+3 = 7