Announcements

- Homework 2 online due today
- No Homework next week
- Exam 1 Next Friday!

Exam Details

- In class
- Randomized assigned seats
- You may use 6 one-sided pages of notes
- No textbook or electronic aids
- 3 Questions in 45 minutes
  - 1^{st} straightforward implementation of algorithm
  - 2^{nd} requires some thought
  - 3^{rd} can be quite tricky

Exam Topics

- Chapter 3
  - Graph basics
  - Explore/DFS
  - Connected components
  - Pre/Post orderings
  - DAGs
  - Topological sort
  - Strongly connected components
- Chapter 4
  - Shortest path definitions
  - BFS
  - Dijkstra
  - Priority queues
  - Bellman-Ford
  - Negative weight cycles
  - Shortest paths in DAGs

Review Options

- I will produce a brief review video
- Lecture podcasts / slides
- Textbook
- OH questions
- Old exams from problem archive

Last Time

- Shortest paths with negative edge weights
- Negative weight cycles
- Computing paths with bounded numbers of edges

Negative Edge Weights

- So far we have talked about the case of non-negative edge weights.
  - The usual case (distance & time usually cannot be negative).
  - However, if “lengths” represent other kinds of costs, sometimes they can be negative.
- Problem statement same. Find path with smallest sum of edge weights.

Negative Weight Cycles

Definition: A negative weight cycle is a cycle where the total weight of edges is negative.

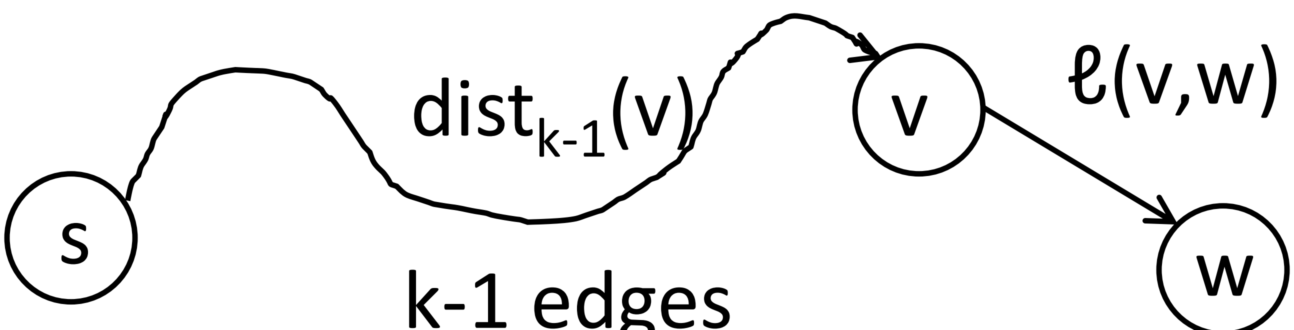
- If G has a negative weight cycle, then there are probably no shortest paths.
  - Go around the cycle over and over.
- **Note:** For undirected G, a single negative weight edge gives a negative weight cycle by going back and forth on it.

Algorithm Idea

Instead of finding shortest paths (which may not exist), find shortest paths of length at most k.

For $w \neq s$,

$$\mathrm{dist}_k(w) = \min_{(v,w)\in E} \mathrm{dist}_{k-1}(v) + \ell(v,w).$$



Algorithm

```
Bellman-Ford(G,s,ℓ)
  dist_0(v) ← ∞ for all v
    //cant reach
  dist_0(s) ← 0
  For k = 1 to n
    For w ∈ V
      dist_k(w)←min(dist_{k-1}(v)+ℓ(v,w))
    dist_k(s) ← min(dist_k(s),0)
      // s has the trivial path
```

Today

- Bellman-Ford
  - Computing shortest paths
  - Detecting negative weight cycles
- Shortest paths in DAGs
- Introduction to divide & conquer

Algorithm

```
Bellman-Ford(G,s,ℓ)
  dist_0(v) ← ∞ for all v
    //cant reach
  dist_0(s) ← 0
  For k = 1 to n
    For w ∈ V
      dist_k(w)←min(dist_{k-1}(v)+ℓ(v,w))
    dist_k(s) ← min(dist_k(s),0)
      // s has the trivial path
```

Algorithm

```
Bellman-Ford(G,s,ℓ)
  dist_0(v) ← ∞ for all v
    //cant reach
  dist_0(s) ← 0
  For k = 1 to n
    For w ∈ V
      dist_k(w)←min(dist_{k-1}(v)+ℓ(v,w))
    dist_k(s) ← min(dist_k(s),0)
      // s has the trivial path
```

O(|E|)

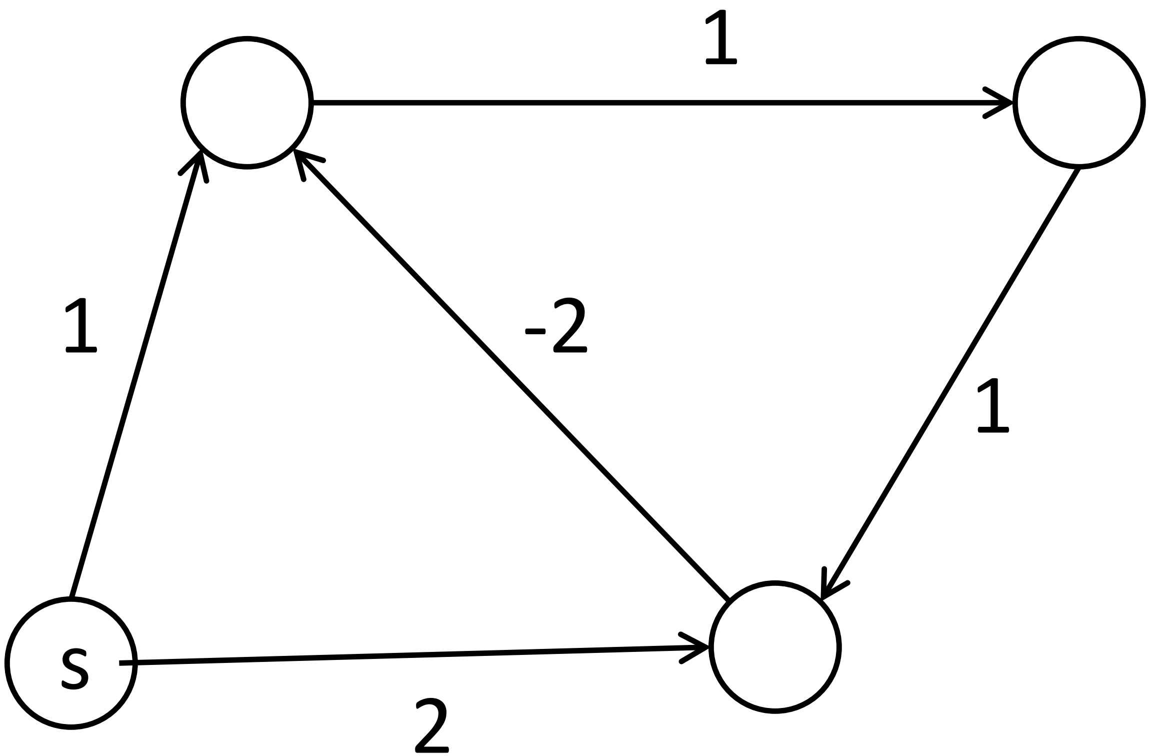
Algorithm

```
Bellman-Ford(G,s,ℓ)
  dist₀(v) ← ∞ for all v
    //cant reach
  dist₀(s) ← 0
  For k = 1 to n
    For w ∈ V
      distₖ(w)←min(distₖ₋₁(v)+ℓ(v,w))
    distₖ(s) ← min(distₖ(s),0)
      // s has the trivial path
```

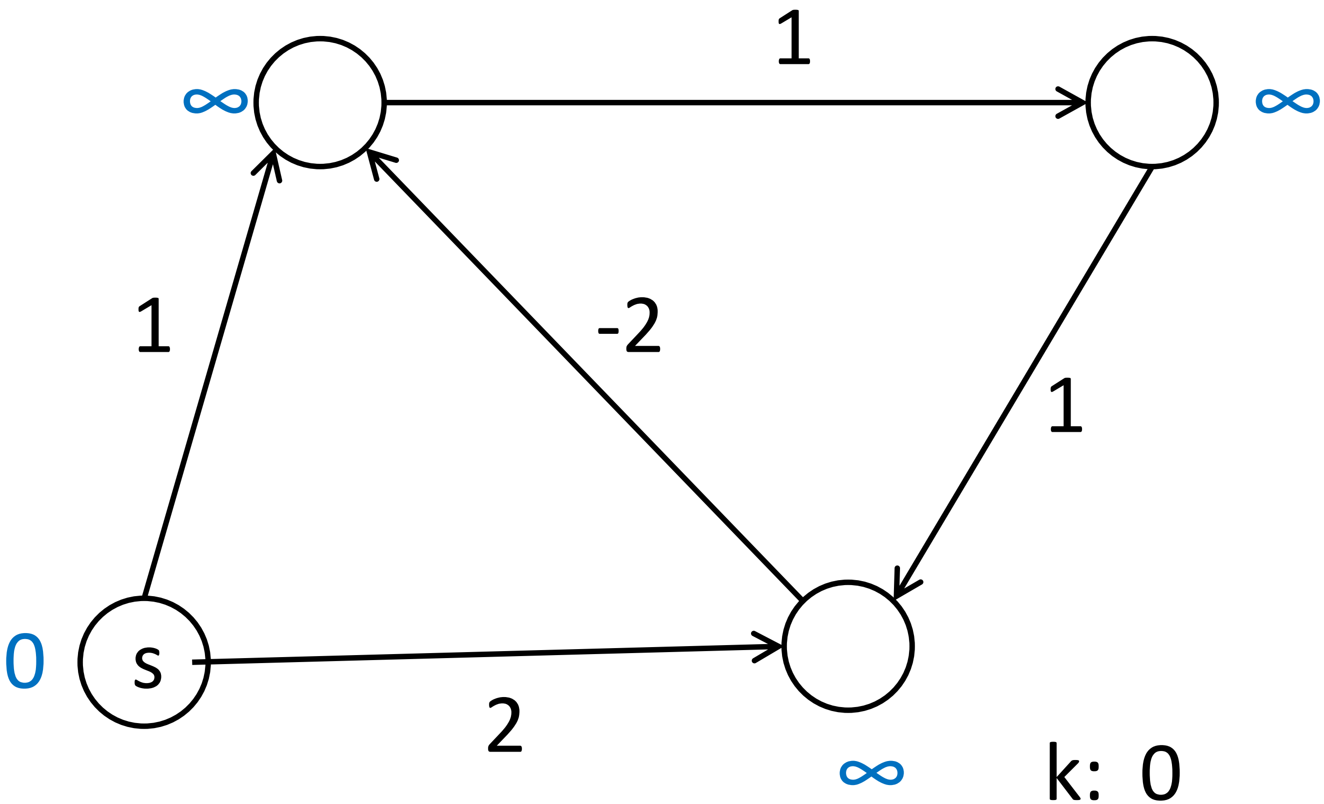
What value of k do we use?

O(|E|)

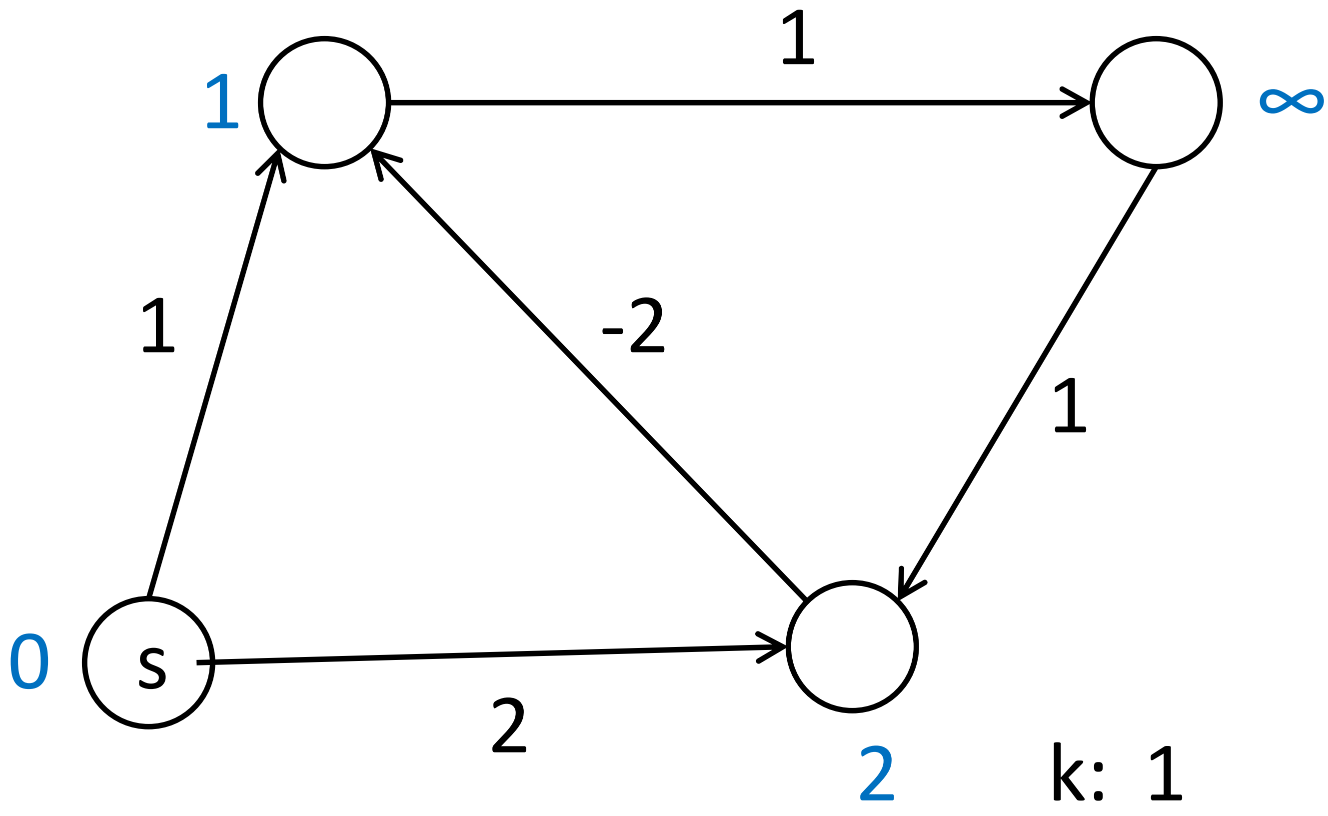
Example



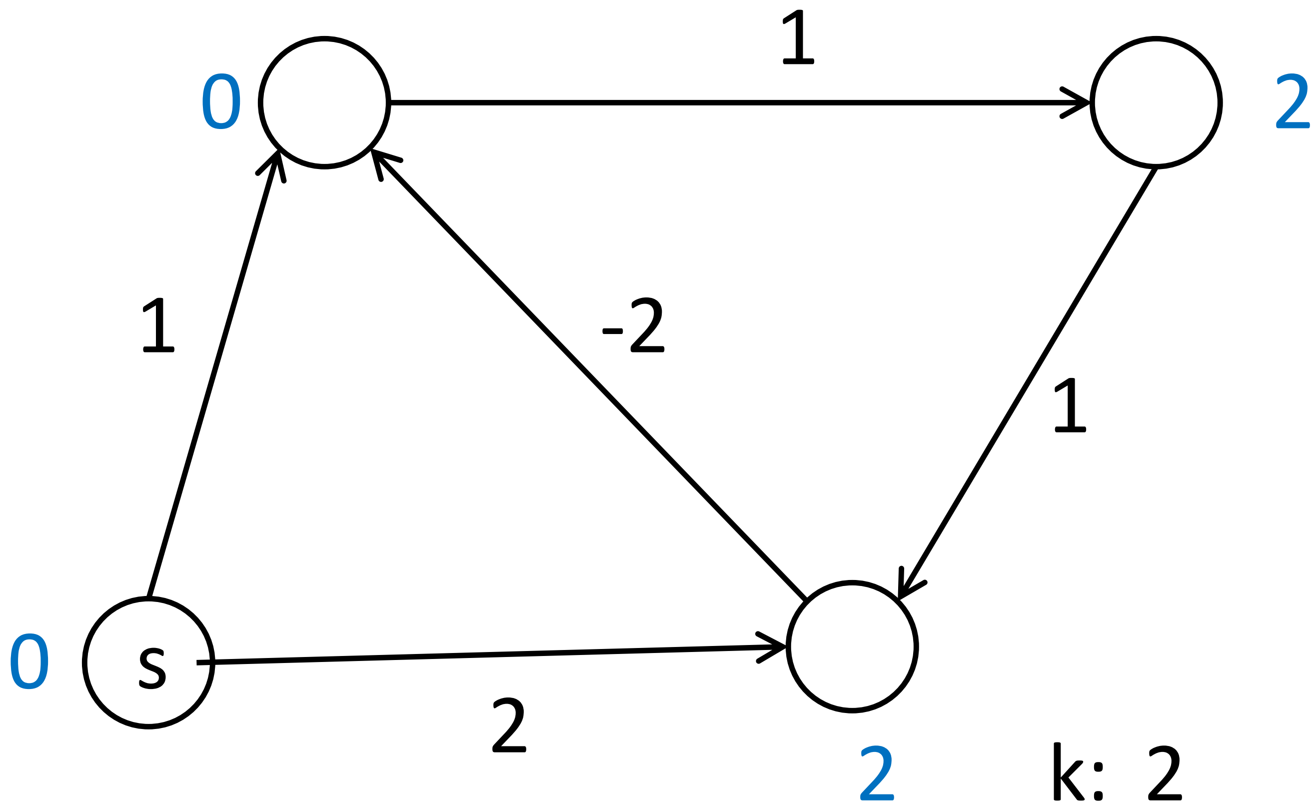
Example



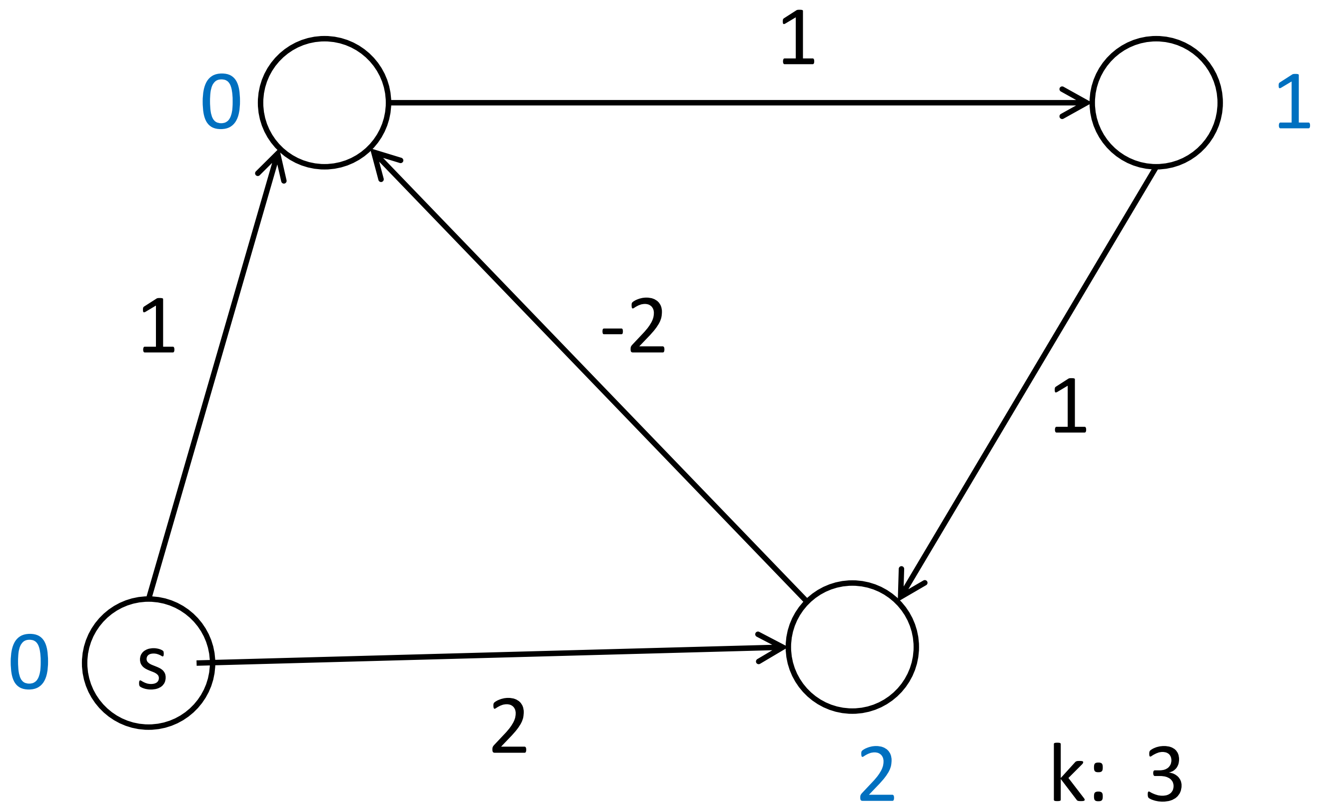
Example



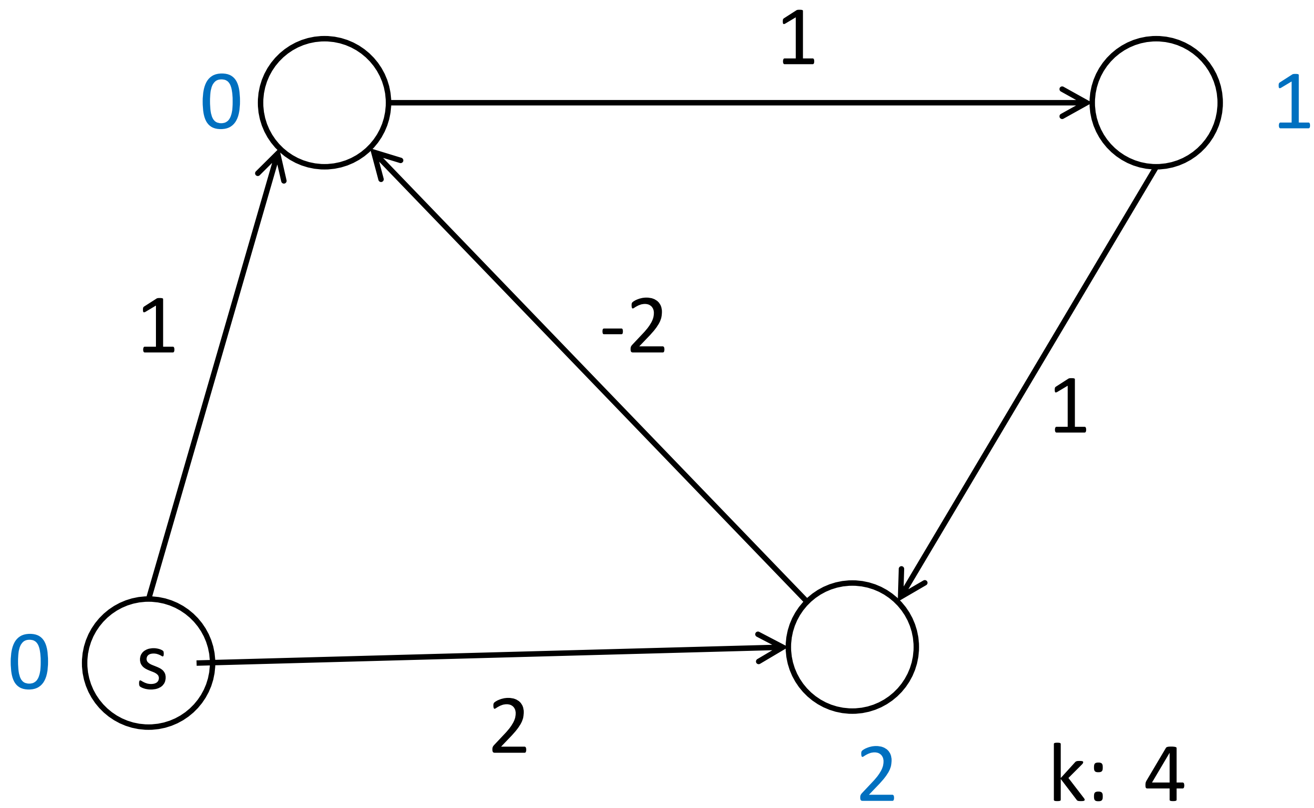
Example



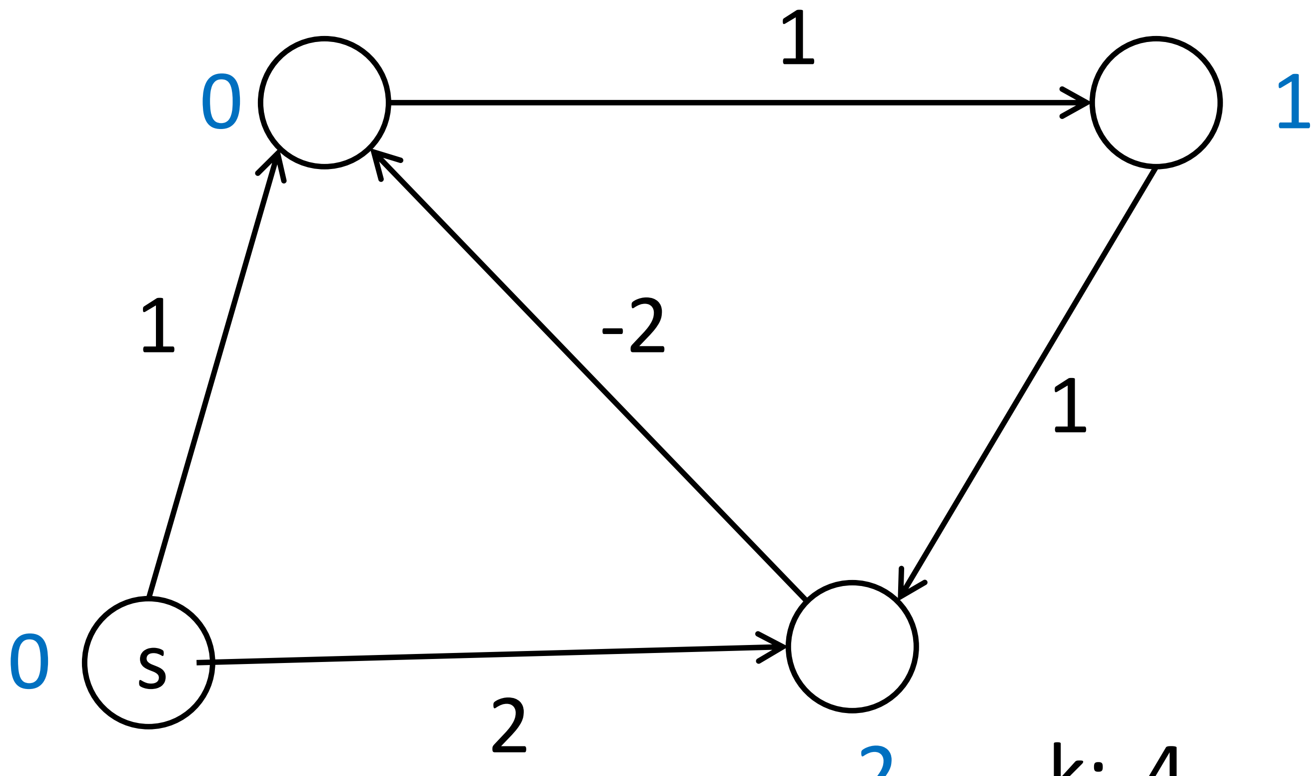
Example



Example



Example



k: 4

Stabalizes

Analysis

Proposition: If $n \geq |V|-1$ and if G has no negative weight cycles, then for all v,

$$\text{dist}(v) = \text{dist}_n(v).$$

Analysis

Proposition: If $n \geq |V|-1$ and if G has no negative weight cycles, then for all v,

$$\mathrm{dist}(v) = \mathrm{dist}_n(v).$$

- If there is a negative weight cycle, there probably is no shortest path.

Analysis

<u>Proposition:</u> If $n \geq |V|-1$ and if G has no negative weight cycles, then for all v,

$$\text{dist}(v) = \text{dist}_n(v).$$

- If there is a negative weight cycle, there probably is no shortest path.
- If not, we only need to run our algorithm for $|V|$ rounds, for a final runtime $O(|V||E|)$.

Proof

- We need to show that the shortest path has fewer than |V| edges.

Proof

- We need to show that the shortest path has fewer than |V| edges.
- If a path has at least |V| edges, it must contain the same vertex twice (by the pigeonhole principle).

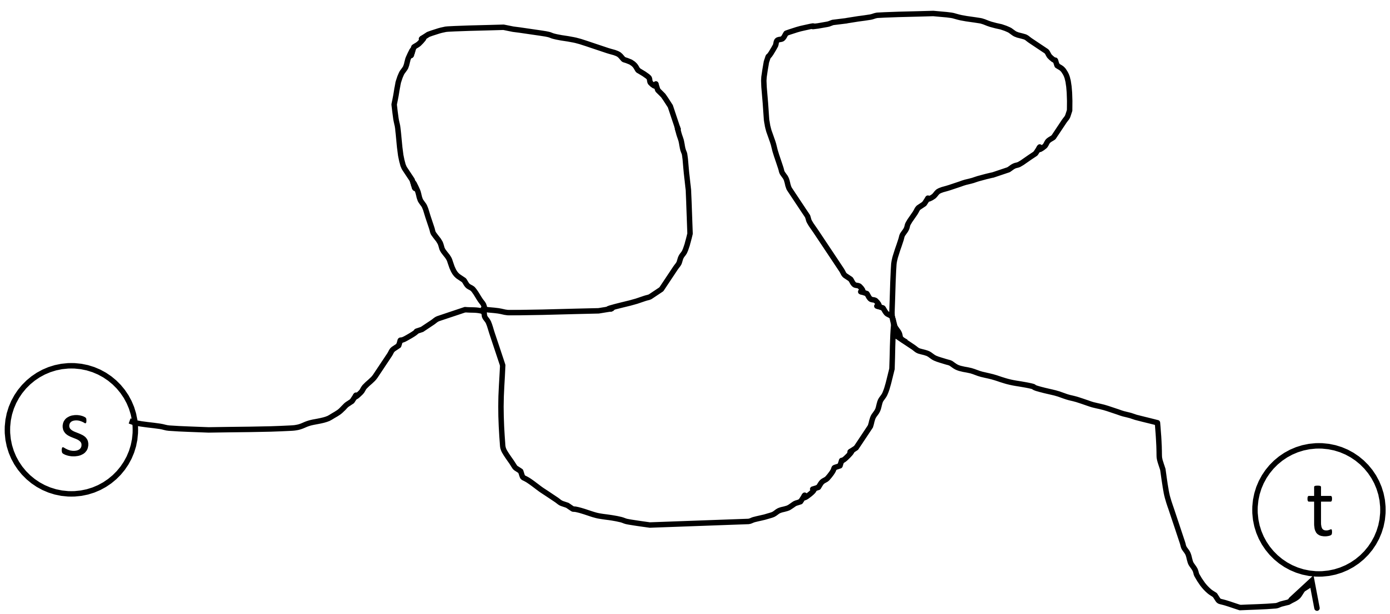
Proof

- We need to show that the shortest path has fewer than $|V|$ edges.
- If a path has at least $|V|$ edges, it must contain the same vertex twice (by the pigeonhole principle).
- This means it has a loop.

Proof

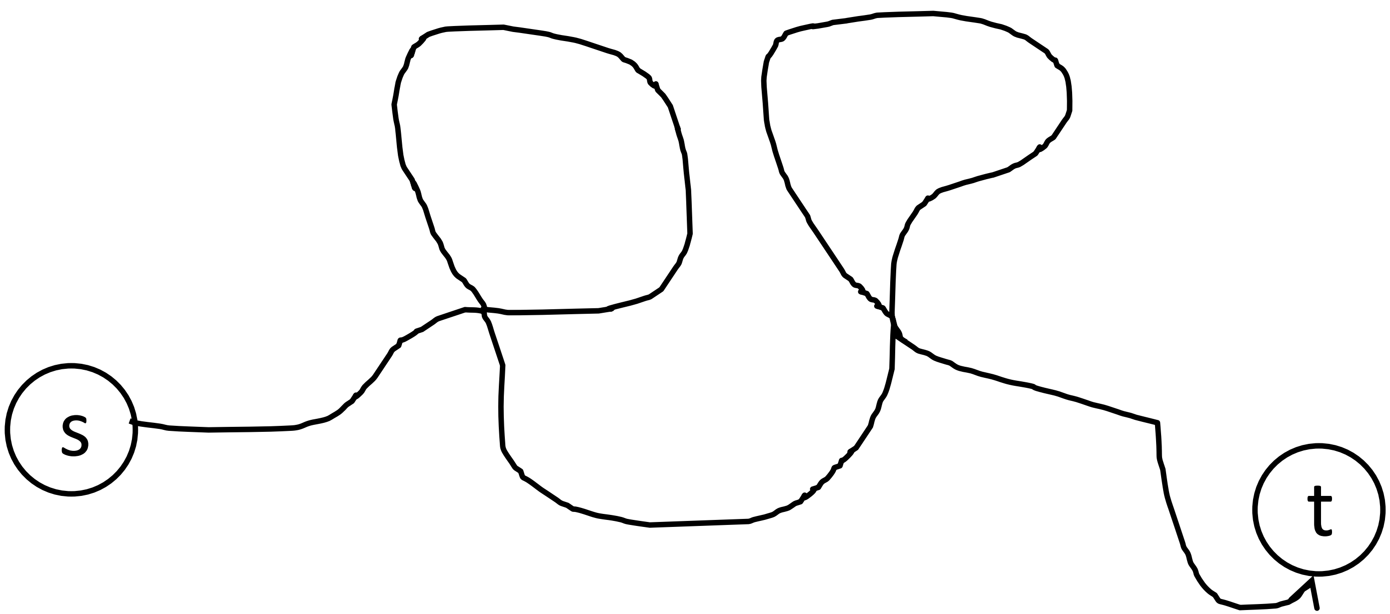
- We need to show that the shortest path has fewer than |V| edges.
- If a path has at least |V| edges, it must contain the same vertex twice (by the pigeonhole principle).
- This means it has a loop.
- Removing the loop gives a shorter path.

Proof



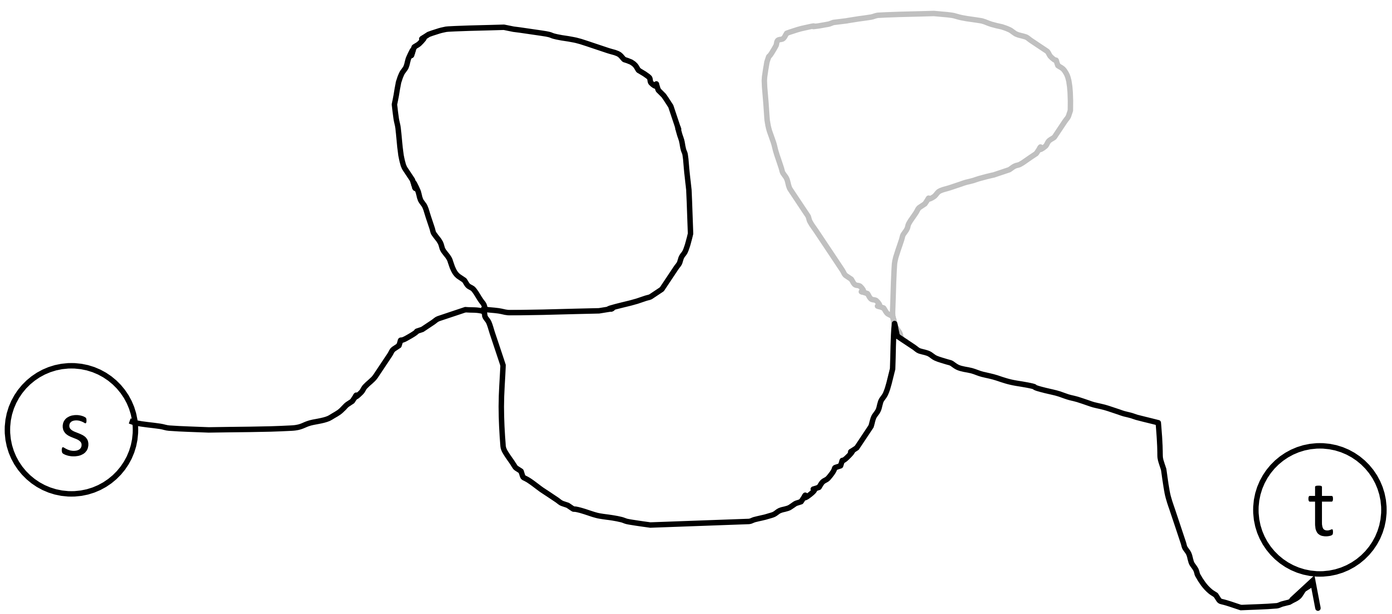
Proof

Non-negative total weight
(no negative weight cycles)



Proof

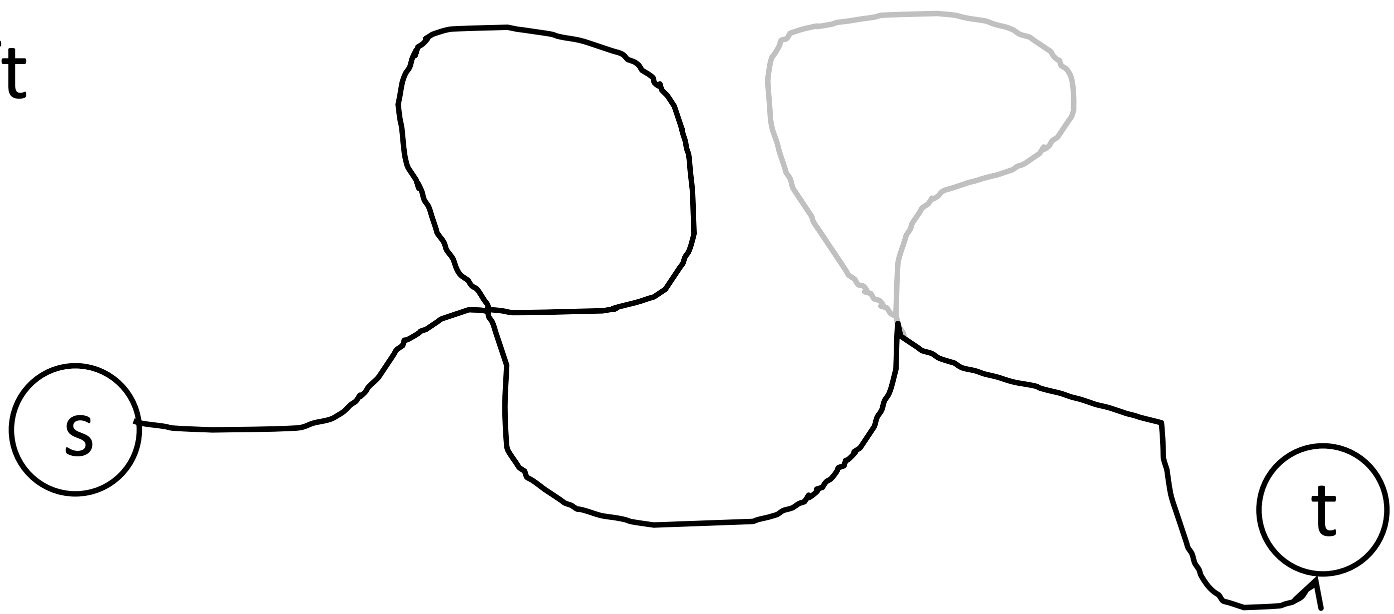
Non-negative total weight
(no negative weight cycles)



Proof

Remove other cycles until none left

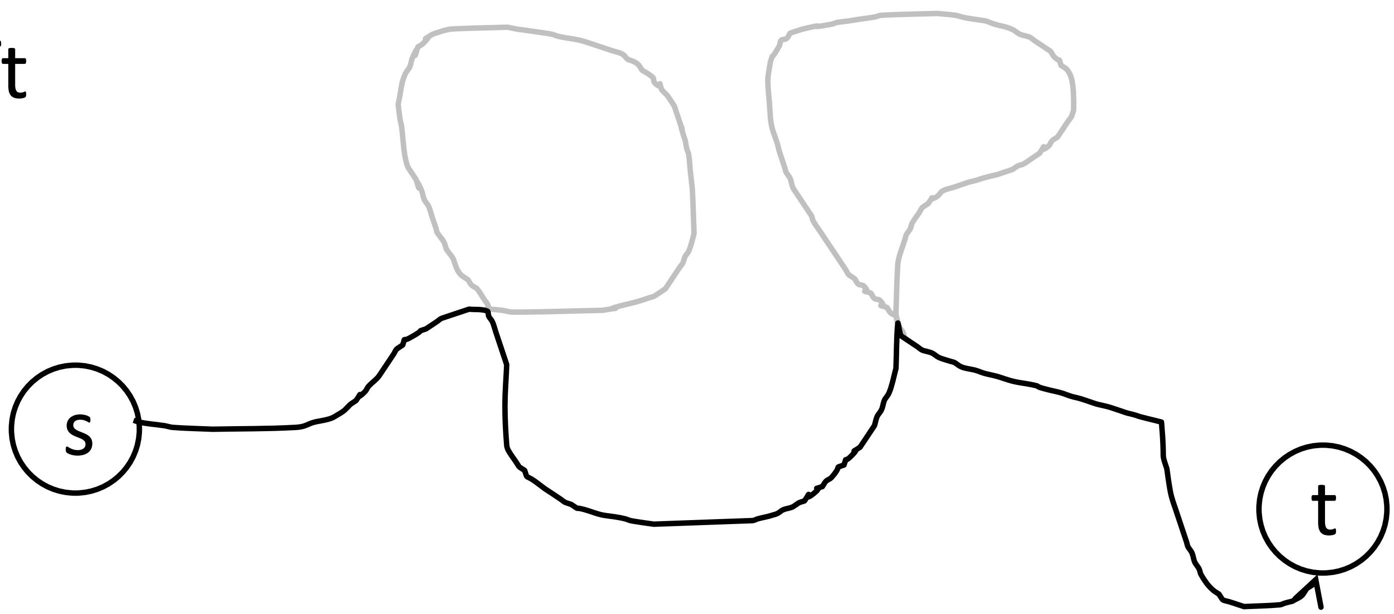
Non-negative total weight (no negative weight cycles)



Proof

Remove other cycles until none left

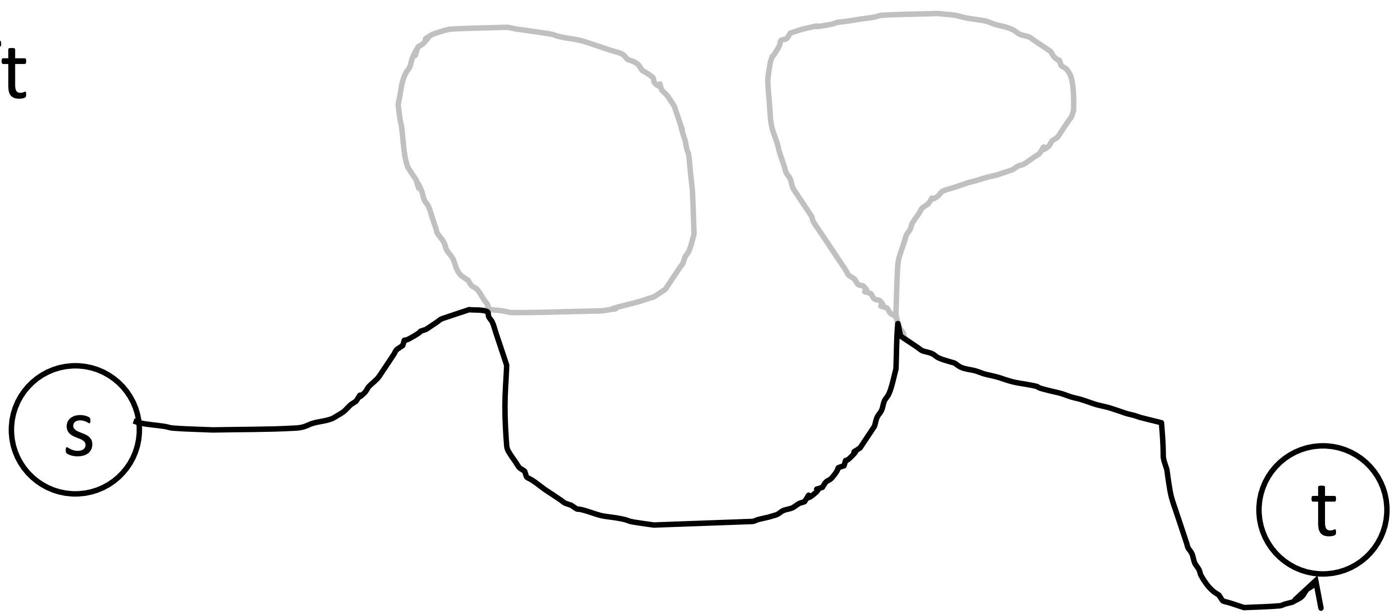
Non-negative total weight
(no negative weight cycles)



Proof

Remove other cycles until none left

Non-negative total weight (no negative weight cycles)



At most |V|-1 edges

New Algorithm

While Bellman-Ford computes shortest paths in time $O(|V||E|)$, it is possible to do better. A recent breakthrough gave an algorithm that runs in time

$$O(\log^8|V|\log(W)(|V|+|E|))$$

where W is the most negative edge weight.

Detecting Negative Cycles

If there are no negative weight cycles, Bellman-Ford computes shortest paths (and they might not exist otherwise).

Detecting Negative Cycles

If there are no negative weight cycles, Bellman-Ford computes shortest paths (and they might not exist otherwise).

How do we know whether or not there are any?

Cycle Detection

Proposition: For any $n \geq |V| - 1$, there are no negative weight cycles reachable from s if and only if for every $v \in V$

$$\mathrm{dist}_n(v) = \mathrm{dist}_{n+1}(v)$$

Cycle Detection

Proposition: For any $n \geq |V| - 1$, there are no negative weight cycles reachable from s if and only if for every $v \in V$

$$dist_n(v) = dist_{n+1}(v)$$

- Detect by running one more round of Bellman-Ford.
- Need to see if *any* v's distance changes.

Proof of "Only If"

- Suppose no negative weight cycles.

Proof of “Only If”

- Suppose no negative weight cycles.
- For any $n \geq |V| - 1$, $dist_n(v) = dist(v)$.

Proof of "Only If"

- Suppose no negative weight cycles.
- For any $n \geq |V| - 1$, $dist_n(v) = dist(v)$.
- So

$$dist_n(v) = dist(v) = dist_{n+1}(v)$$

Proof of “If”

Suppose $dist_n(v) = dist_{n+1}(v)$ for all v.

Proof of "If"

Suppose $\text{dist}_n(v) = \text{dist}_{n+1}(v)$ for all v.

$$\begin{aligned}
\text{dist}_{n+2}(w) &= \min_{(v,w)\in E} (\text{dist}_{n+1}(v) + \ell(v,w)) \\
&= \min_{(v,w)\in E} (\text{dist}_{n}(v) + \ell(v,w)) \\
&= \text{dist}_{n+1}(w).
\end{aligned}$$

Proof of "If"

Suppose $\text{dist}_n(v) = \text{dist}_{n+1}(v)$ for all v.

$$\begin{aligned}\text{dist}_{n+2}(w) &= \min_{(v,w)\in E}(\text{dist}_{n+1}(v) + \ell(v,w)) \\ &= \min_{(v,w)\in E}(\text{dist}_{n}(v) + \ell(v,w)) \\ &= \text{dist}_{n+1}(w).\end{aligned}$$

So

$$\text{dist}_n(v) = \text{dist}_{n+1}(v) = \text{dist}_{n+2}(v) = \text{dist}_{n+3}(v) = \ldots$$

Proof of "If"

Suppose $\text{dist}_n(v) = \text{dist}_{n+1}(v)$ for all v.

$$\begin{aligned}\text{dist}_{n+2}(w) &= \min_{(v,w)\in E} (\text{dist}_{n+1}(v) + \ell(v,w)) \\ &= \min_{(v,w)\in E} (\text{dist}_{n}(v) + \ell(v,w)) \\ &= \text{dist}_{n+1}(w).\end{aligned}$$

So

$$\text{dist}_n(v) = \text{dist}_{n+1}(v) = \text{dist}_{n+2}(v) = \text{dist}_{n+3}(v) = \ldots$$

But if there were a negative weight cycle, distances would decrease eventually.

Alternative Proof

- Assume $dist_n(v) = dist_{n+1}(v) = d(v)$ for all v.

Alternative Proof

- Assume $dist_n(v) = dist_{n+1}(v) = d(v)$ for all v.
- $d(w) = \min(d(v)+\ell(v,w))$.
  - $d(w) \leq d(v)+\ell(v,w)$ for all $(v,w) \in E$
  - $\ell(v,w) \geq d(w) - d(v)$

Alternative Proof

- Assume $\text{dist}_n(v) = \text{dist}_{n+1}(v) = d(v)$ for all v.
- $d(w) = \min(d(v)+\ell(v,w))$.
  - $d(w) \leq d(v)+\ell(v,w)$ for all $(v,w) \in E$
  - $\ell(v,w) \geq d(w) - d(v)$
- Given cycle $v_1,v_2,v_3,\ldots,v_m$ total length of cycle is

$$\ell(v_1, v_2) + \ell(v_2, v_3) + \ldots + \ell(v_m, v_1)$$

$$\geq \; -d(v_1) + d(v_2) - d(v_2) + d(v_3) - \ldots - d(v_m) + d(v_1) = 0.$$

Potential Function

- Let $\ell'(v,w) = \ell(v,w) - d(v) + d(w) \geq 0$

Potential Function

- Let $\ell'(v,w) = \ell(v,w) - d(v) + d(w) \geq 0$
  - Imagine somebody lending you d(w) when you arrive at w, but having to pay it back when you leave.

Potential Function

- Let ℓ'(v,w) = ℓ(v,w)-d(v)+d(w) ≥ 0
  - Imagine somebody lending you d(w) when you arrive at w, but having to pay it back when you leave.
- For any s-t path P, s,v_1,v_2,…,t

$$\begin{aligned}
\ell'(P) &= \ell'(s,v_1) + \ell'(v_1,v_2) + \ldots + \ell'(v_m,t) \\
&= \ell(s,v_1) + \ell(v_1,v_2) + \ldots + \ell(v_m,t) \\
&\qquad - d(s) + d(v_1) - d(v_1) + d(v_2) + \ldots - d(v_m) + d(t) \\
&= \ell(P) - d(s) + d(t).
\end{aligned}$$

Potential Function

- Let $\ell'(v,w) = \ell(v,w) - d(v) + d(w) \geq 0$
  - Imagine somebody lending you d(w) when you arrive at w, but having to pay it back when you leave.
- For any s-t path P, $s, v_1, v_2, \ldots, t$

$$\begin{aligned}
\ell'(P) &= \ell'(s, v_1) + \ell'(v_1, v_2) + \ldots + \ell'(v_m, t) \\
&= \ell(s, v_1) + \ell(v_1, v_2) + \ldots + \ell(v_m, t) \\
&\qquad - d(s) + d(v_1) - d(v_1) + d(v_2) + \ldots - d(v_m) + d(t) \\
&= \ell(P) - d(s) + d(t).
\end{aligned}$$

- Shortest paths same. Non-negative edges.

Shortest Paths in DAGs

We saw that shortest paths is harder when we needed to deal with negative weight cycles. For general graphs, we needed to use Bellman-Ford, which is much slower than our other algorithms.

Shortest Paths in DAGs

We saw that shortest paths is harder when we needed to deal with negative weight cycles. For general graphs, we needed to use Bellman-Ford, which is much slower than our other algorithms.

One way to avoid this was to make edge weights non-negative. In this case, we could use Dijkstra.

Shortest Paths in DAGs

We saw that shortest paths is harder when we needed to deal with negative weight cycles. For general graphs, we needed to use Bellman-Ford, which is much slower than our other algorithms.

One way to avoid this was to make edge weights non-negative. In this case, we could use Dijkstra.

Another way to get rid of negative weight cycles, is to get rid of cycles. If G is a DAG, there are better algorithms.

Fundamental Shortest Paths Formula

For $w \neq s$,

$$\mathrm{dist}(w) = \min_{(v,w) \in E} \mathrm{dist}(v) + \ell(v, w).$$

Hard to apply in general because there's no order to solve equations in.

Fundamental Shortest Paths Formula

$$\text{For } w \neq s,$$
$$\text{dist}(w) = \min_{(v,w)\in E} \text{dist}(v) + \ell(v,w).$$

Hard to apply in general because there's no order to solve equations in.

DAG gives topological order!

Algorithm

```
ShortestPathsInDAGs(G,s,ℓ)
  TopologicalSort(G)
  For w ∈ V in topological order
    If w = s, dist(w) ← 0
    Else
     dist(w)←min(dist(v)+ℓ(v,w))

\\ dist(v) for all upstream v
 already computed
```

Algorithm

```
ShortestPathsInDAGs(G,s,ℓ)
   TopologicalSort(G)
   For w ∈ V in topological order
      If w = s, dist(w) ← 0
      Else
        dist(w)←min(dist(v)+ℓ(v,w))

\\ dist(v) for all upstream v
  already computed
```

O(|V|+|E|)

Algorithm

```
ShortestPathsInDAGs(G,s,ℓ)
   TopologicalSort(G)
   For w ∈ V in topological order
      If w = s, dist(w) ← 0
      Else
        dist(w)←min(dist(v)+ℓ(v,w))
 \\ dist(v) for all upstream v
   already computed
```

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

$O(|V|)$ total

Algorithm

```
ShortestPathsInDAGs(G,s,ℓ)
   TopologicalSort(G)                          } O(|V|+|E|)
   For w ∈ V in topological order
      If w = s, dist(w) ← 0
      Else                                     O(|E|) total
       dist(w)←min(dist(v)+ℓ(v,w))             }
O(|V|) total

 \\ dist(v) for all upstream v
   already computed
```

Algorithm

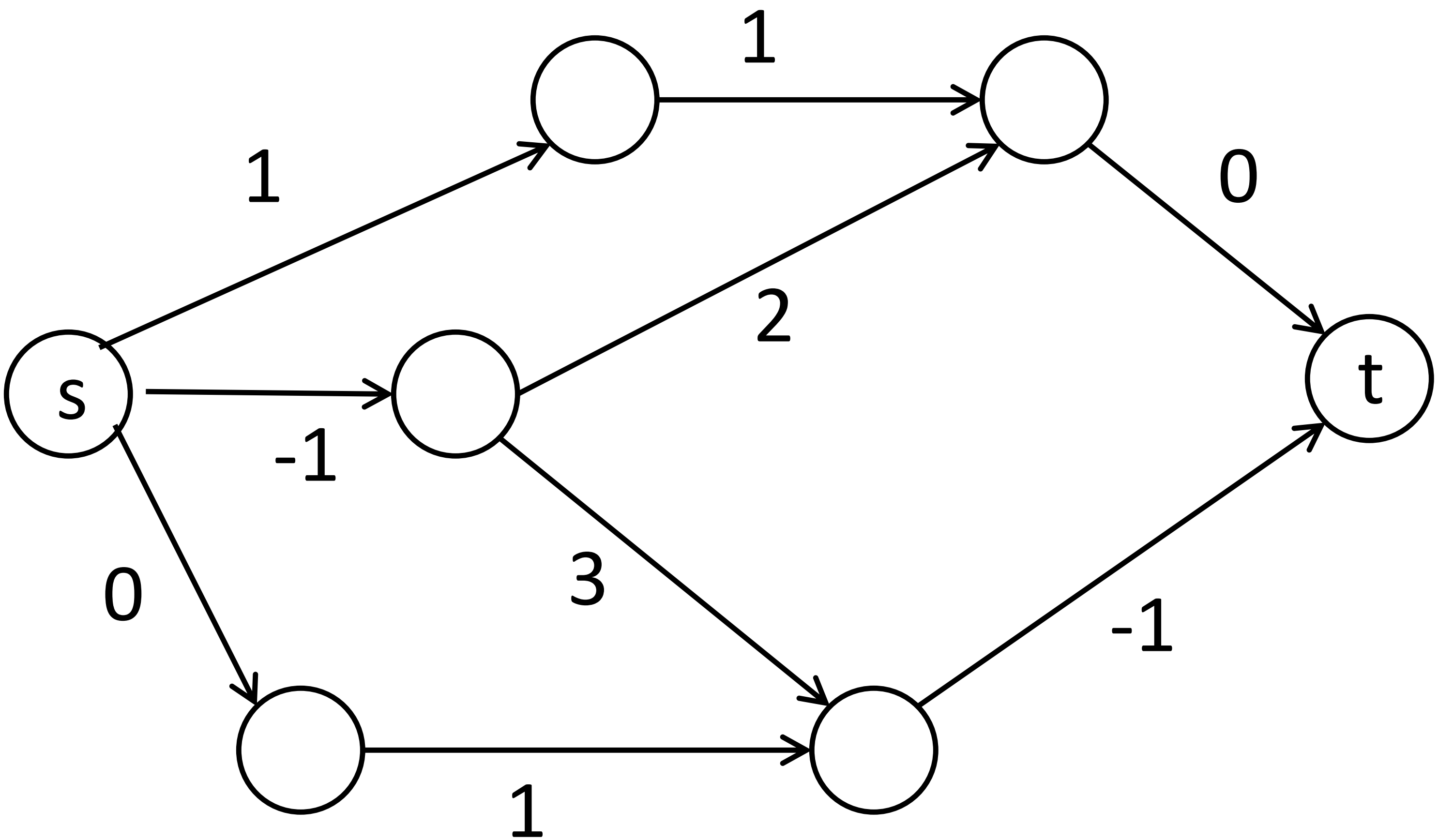
Runtime O(|V|+|E|)

```
ShortestPathsInDAGs(G,s,ℓ)
   TopologicalSort(G)                      O(|V|+|E|)
   For w ∈ V in topological order
      If w = s, dist(w) ← 0
      Else                                 O(|E|) total
        dist(w)←min(dist(v)+ℓ(v,w))
```

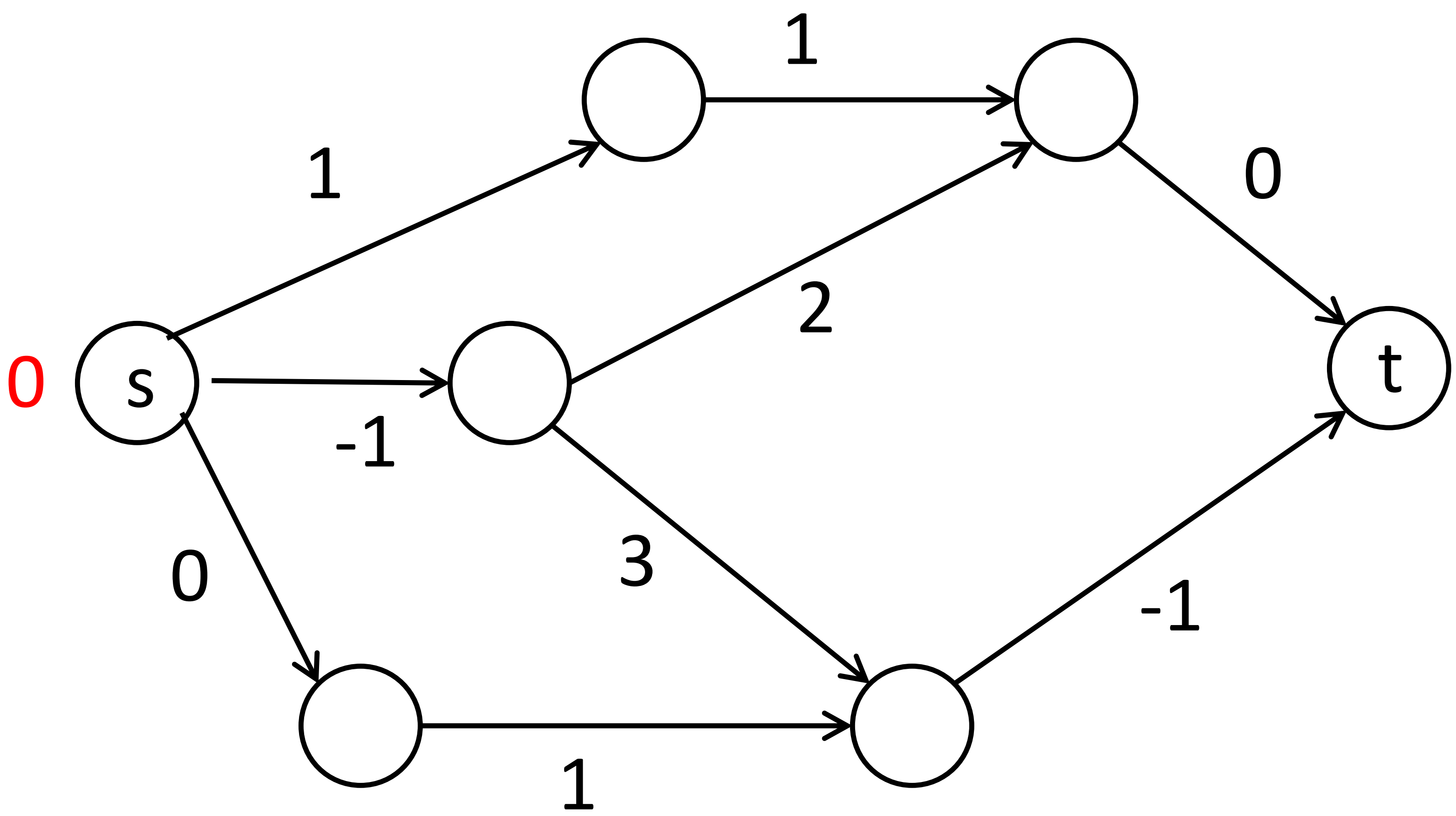
O(|V|) total

\\ dist(v) for all upstream v already computed

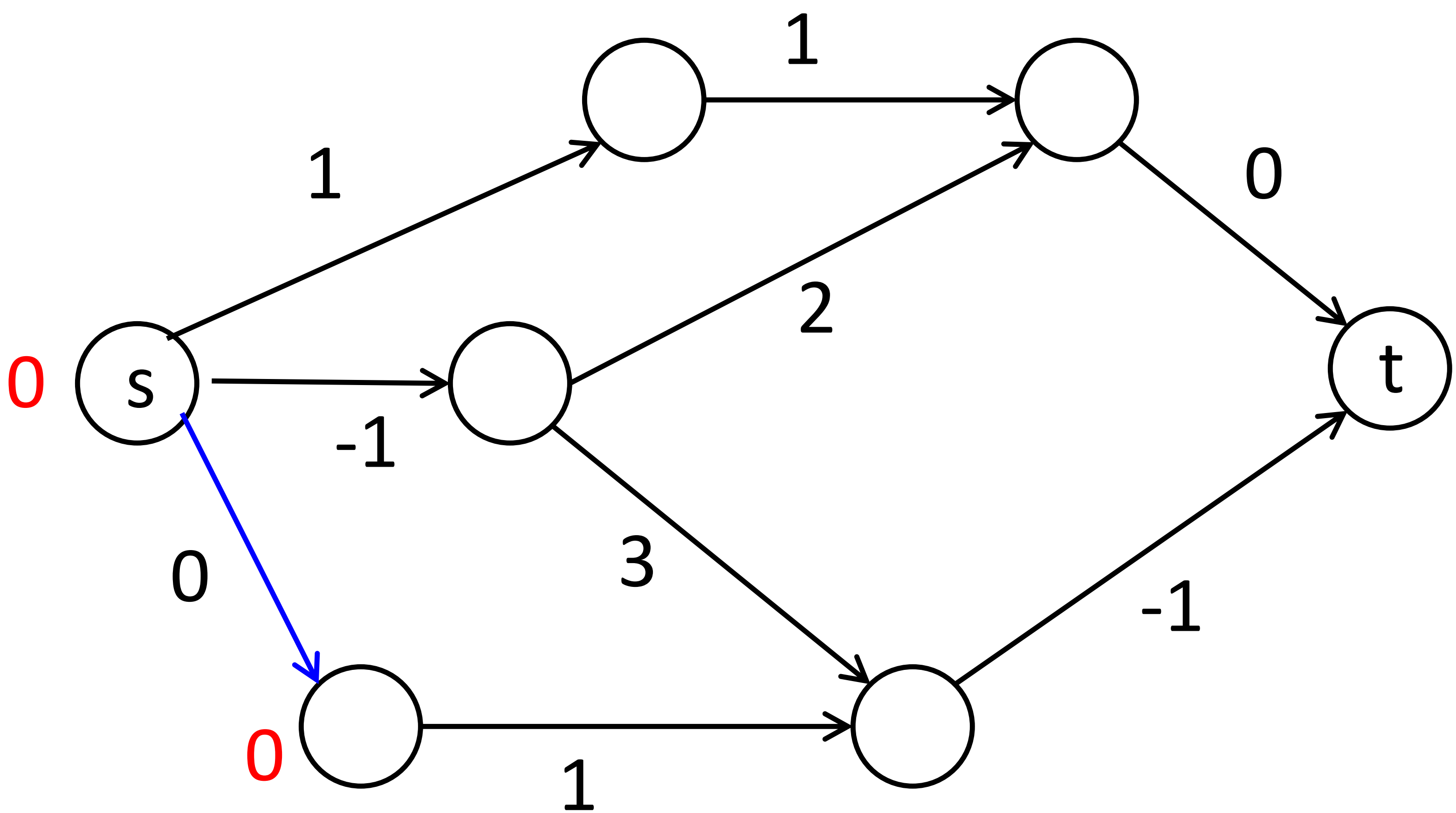
Example



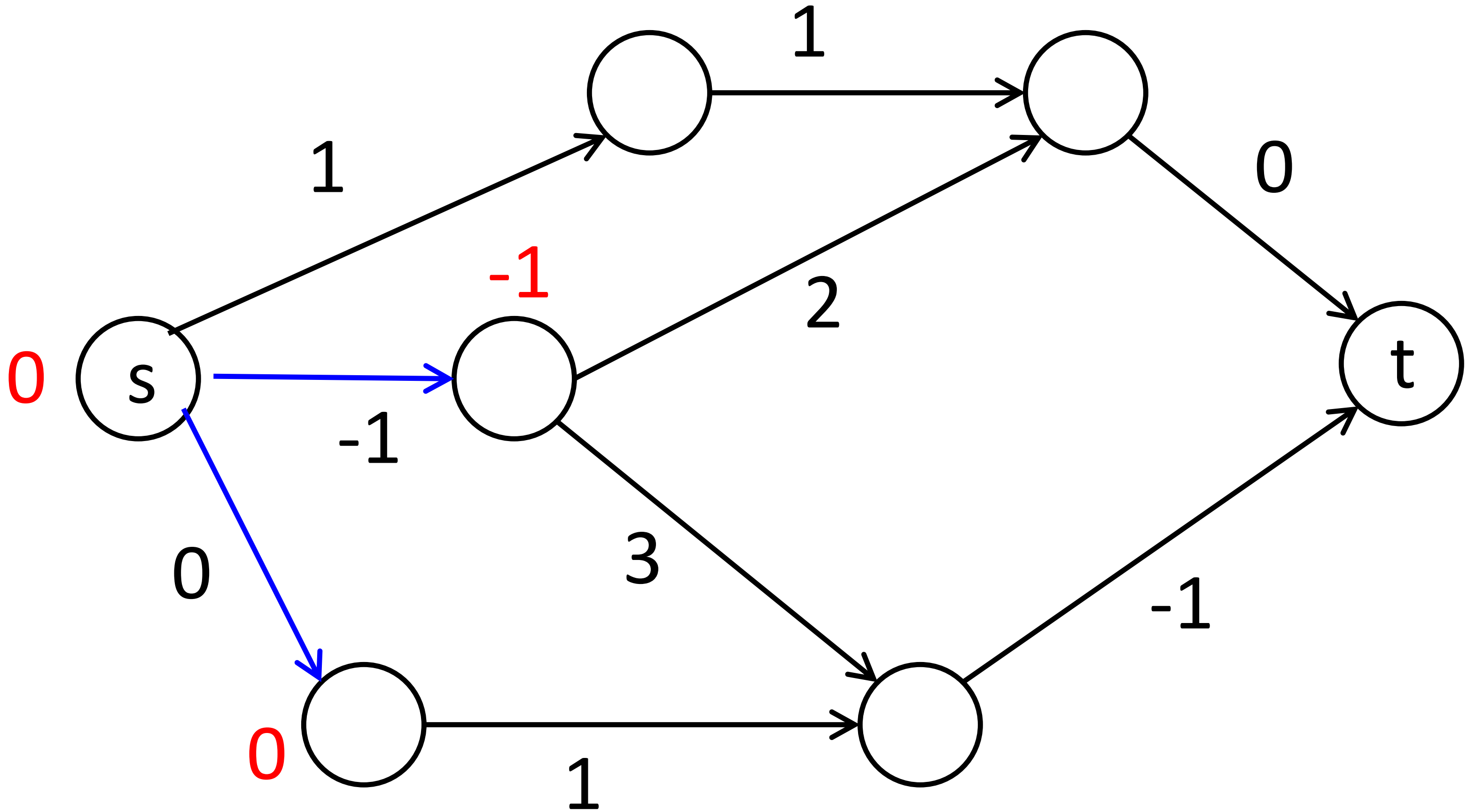
Example



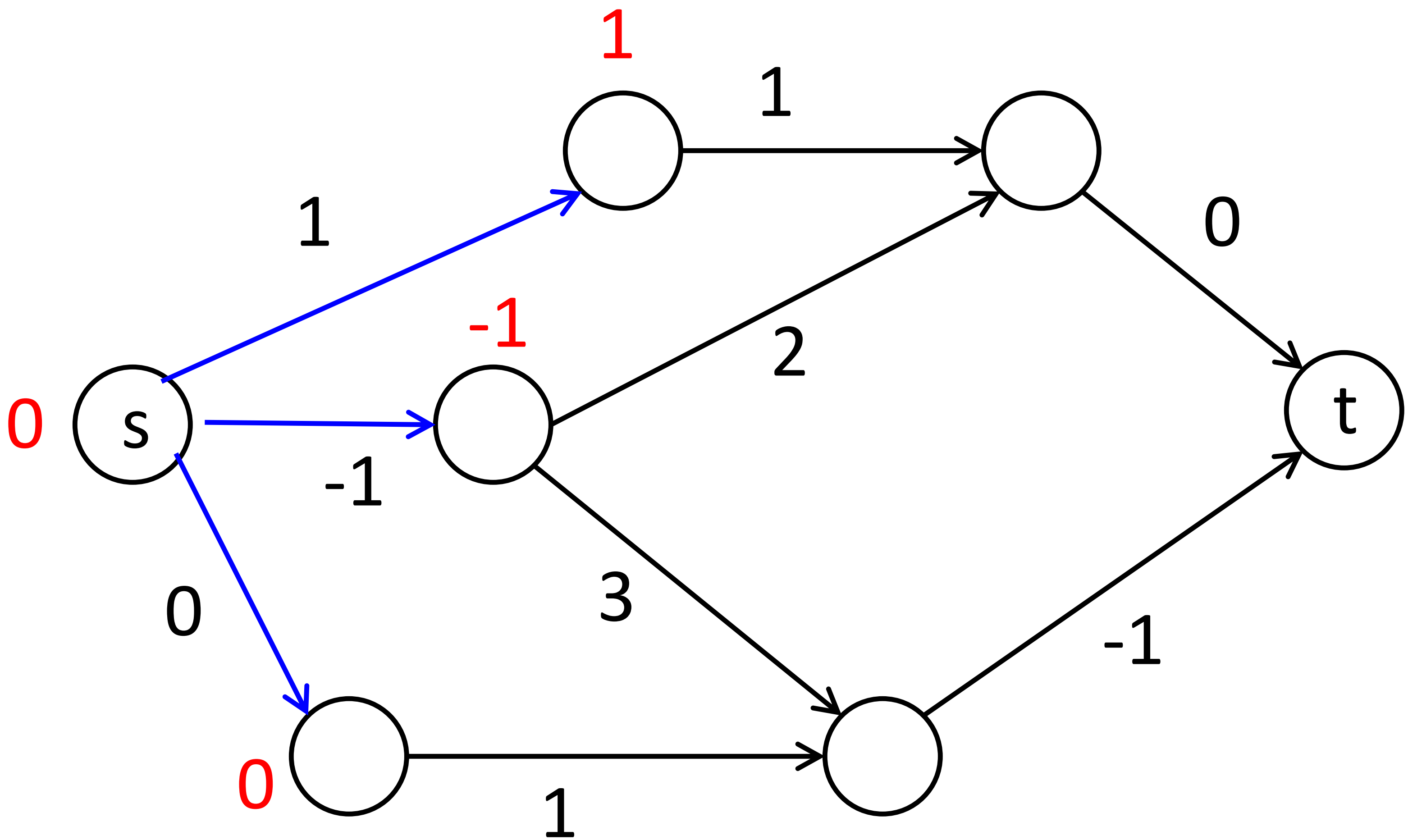
Example



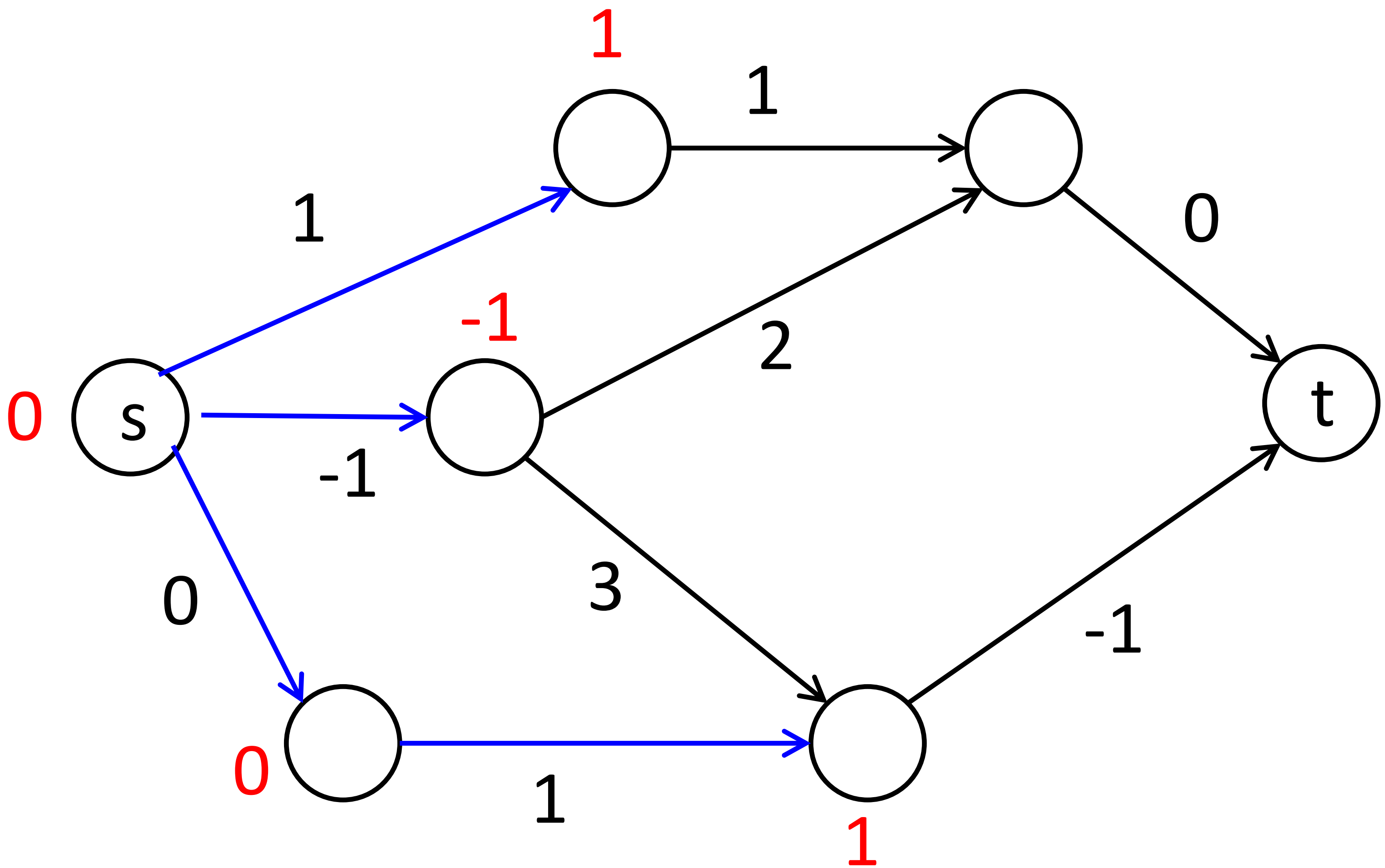
Example



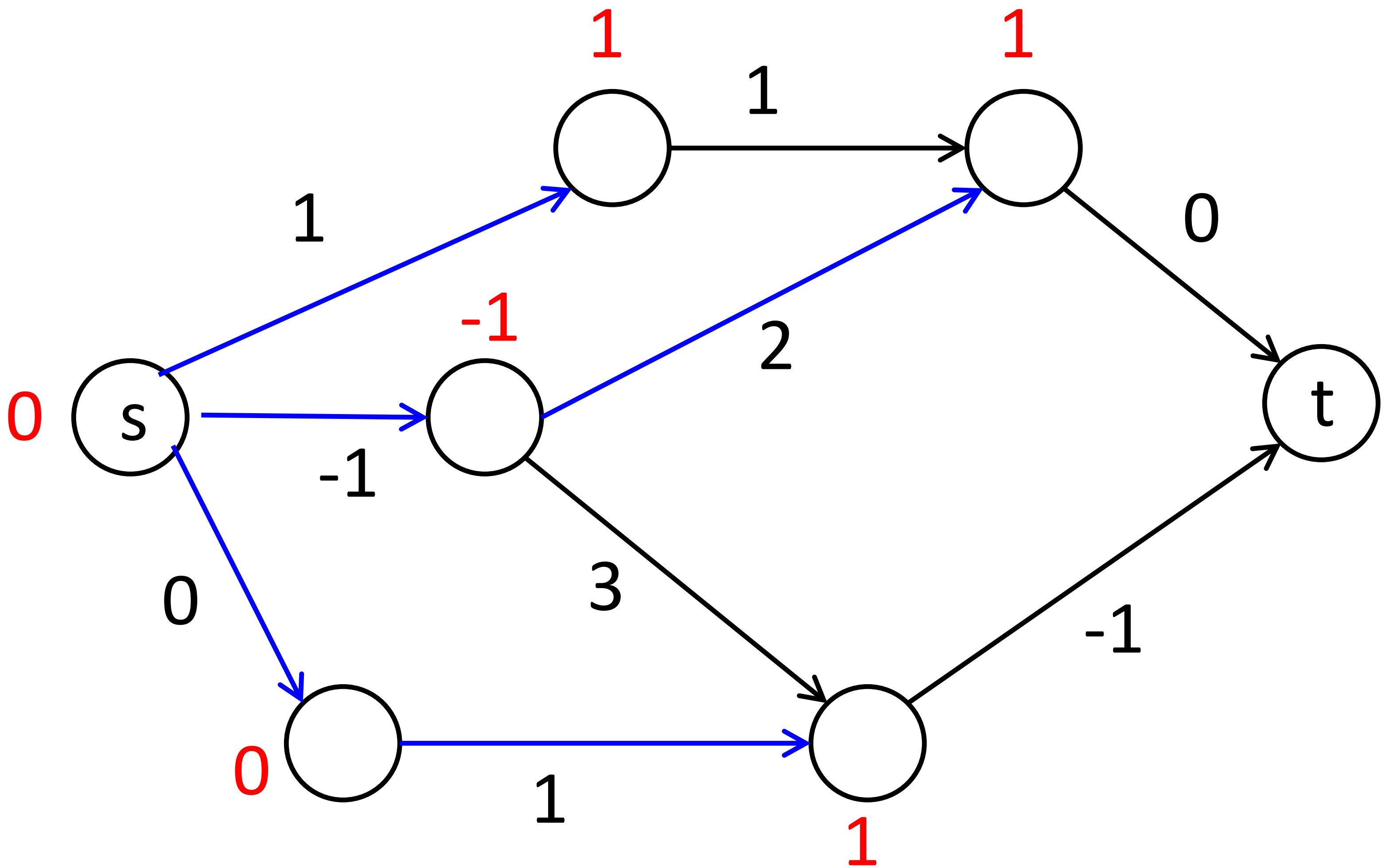
Example



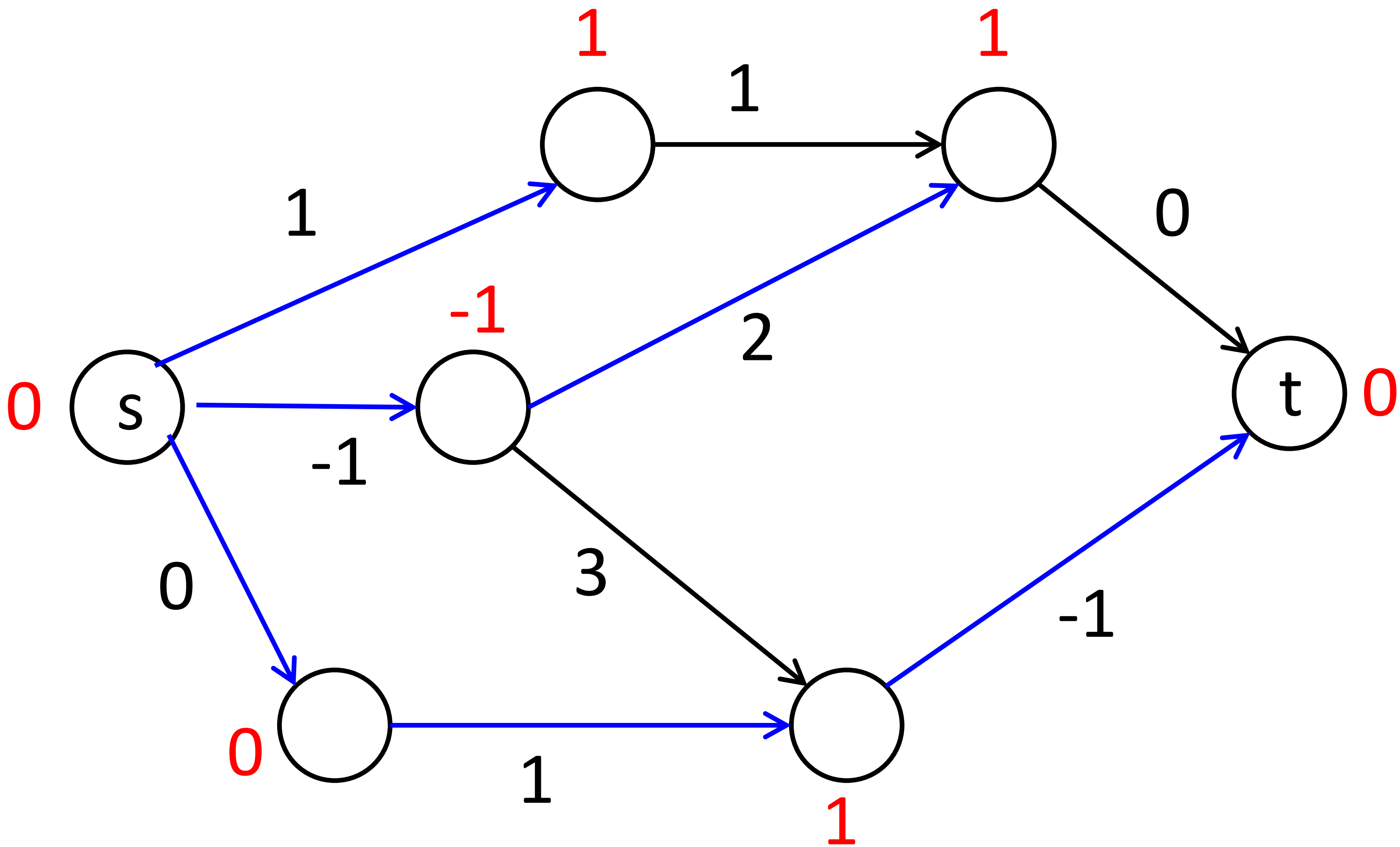
Example



Example



Example



Shortest Path Algorithms Summary

Unit Weights: `Breadth First Search`
$O(|V|+|E|)$

Non-negative Weights: `Dijkstra`
$O(|V|\log|V|+|E|)$

Arbitrary Weights: `Bellman-Ford` $O(|V||E|)$

Arbitrary Weights, graph is a DAG:
`Shortest-Paths-In-DAGs` $O(|V|+|E|)$

Divide & Conquer (Ch 2)

- General Technique
- Master Theorem
- Karatsuba Multiplication
- Strassen’s Algorithm
- Merge Sort
- Order Statistics
- Binary Search
- Closest Pair of Points

Divide and Conquer

This is the first of our three major algorithmic techniques.

Divide and Conquer

This is the first of our three major algorithmic techniques.

1. Break problem into pieces
2. Solve pieces recursively
3. Recombine pieces to get answer

Example: Integer Multiplication

Problem: Given two n-bit numbers find their product.

Example: Integer Multiplication

Problem: Given two n-bit numbers find their product.

Naïve Algorithm: Schoolboy multiplication. The binary version of the technique that you probably learned in elementary school.

Schoolboy Multiplication

$$
\begin{array}{cccccc}
 & a_1 & a_2 & \dots & a_{n-1} & a_n \\
\text{x} & b_1 & b_2 & \dots & b_{n-1} & b_n \\
\hline
\end{array}
$$

Schoolboy Multiplication

$$
\begin{array}{cccccccc}
 & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
x & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
\hline
a_1 b_n & a_2 b_n & a_3 b_n & \ldots & a_{n-1} b_n & a_n b_n
\end{array}
$$

Schoolboy Multiplication

$$\begin{array}{ccccccc}
 & & a_1 & a_2 & \dots & a_{n-1} & a_n \\
 & \times & b_1 & b_2 & \dots & b_{n-1} & b_n \\
\hline
 & a_1 b_n & a_2 b_n & a_3 b_n & \dots & a_{n-1} b_n & a_n b_n \\
a_1 b_{n-1} & a_2 b_{n-1} & a_3 b_{n-1} & a_4 b_{n-1} & \dots & a_n b_{n-1} & 0
\end{array}$$

Schoolboy Multiplication

			a_1	a_2	…	a_{n-1}	a_n
			x b_1	b_2	…	b_{n-1}	b_n
			------------	------------	------------	------------	------------
		a_1b_n	a_2b_n	a_3b_n	…	$a_{n-1}b_n$	a_nb_n
	a_1b_{n-1}	a_2b_{n-1}	a_3b_{n-1}	a_4b_{n-1}	…	a_nb_{n-1}	0
	…	…	…	…	…	…	…
$+a_1b_n$	a_2b_n	a_3b_n	…	a_nb_n	0	0	0

Schoolboy Multiplication

$$
\begin{array}{rccccccc}
 & & & a_1 & a_2 & \ldots & a_{n-1} & a_n \\
 & & \mathrm{x} & b_1 & b_2 & \ldots & b_{n-1} & b_n \\
\hline
 & & a_1b_n & a_2b_n & a_3b_n & \ldots & a_{n-1}b_n & a_nb_n \\
 & a_1b_{n-1} & a_2b_{n-1} & a_3b_{n-1} & a_4b_{n-1} & \ldots & a_nb_{n-1} & 0 \\
 & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
+a_1b_n & a_2b_n & a_3b_n & \ldots & a_nb_n & 0 & 0 & 0 \\
\hline
\end{array}
$$

ANSWER

Question: Runtime

What is the asymptotic runtime of the schoolboy algorithm?

A) $O(n)$
B) $O(n \log(n))$
C) $O(n^2)$
D) $O(n^3)$
E) $O(2^n)$

Question: Runtime

What is the asymptotic runtime of the schoolboy algorithm?

A) $O(n)$
B) $O(n \log(n))$
C) $O(n^2)$
D) $O(n^3)$
E) $O(2^n)$

Need to write down $O(n^2)$ bits of numbers to add. Addition can be done in linear time.