CS 473ug: Algorithms

Chandra Chekuri
chekuri@cs.uiuc.edu
3228 Siebel Center

University of Illinois, Urbana-Champaign

Fall 2007

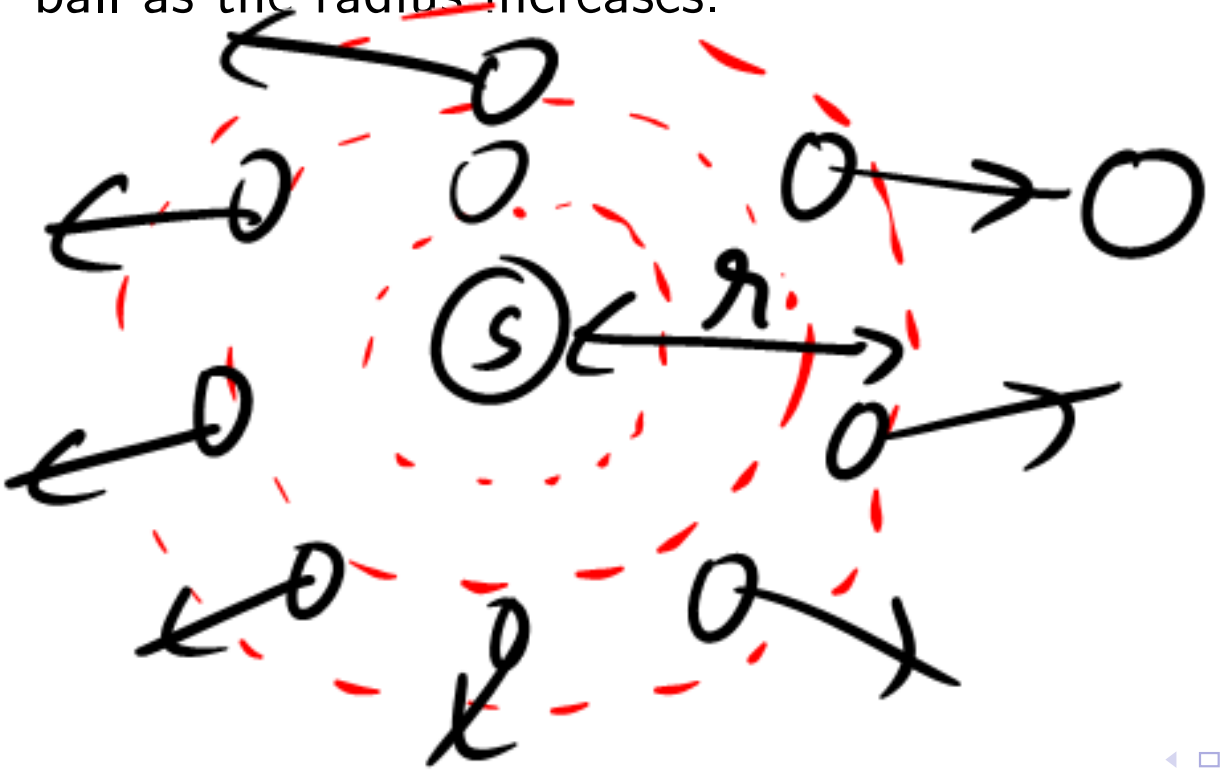
Part I

Dijkstra's Algorithm Recap

An Alternate View

Previous class: view Dijkstra's algorithm as an efficient implementation of BFS in expanded graph.

Another view: Grow a ball around s and update the vertices in the ball as the radius increases.



An Alternate View

```
for all u ∈ V do
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0
S = {s}

While S ≠ V do
    pick u ∈ V-S with smallest dist value
    S = S ∪ {u}
    for all (u,v) in Adj(u) do
        dist(v) = min(dist(v), dist(u) + ℓ(u, v))
```

An Alternate View

```
for all u ∈ V do
    dist(u) = ∞
    prev(u) = nil
dist(s) = 0
S = {s}

While S ≠ V do
    pick u ∈ V-S with smallest dist value
    S = S ∪ {u}
    for all (u,v) in Adj(u) do
        dist(v) = min(dist(v), dist(u) + ℓ(u, v))
```

Do you see similarity with Prim's algorithm for MST?

Correctness

$d(s, u)$: shortest path distance from s to u.

Key Property: If $P = s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is a shortest path from s to v_k then $d(s, v_i) \leq d(s, v_k)$ for $1 \leq i < k$.

True only because lengths are positive!

Dijstra's algorithm works for both undirected and directed graphs.

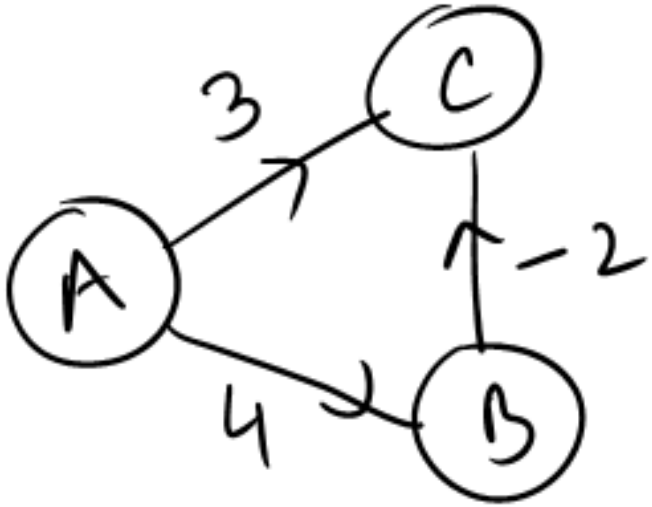
Part II

Single Source Shortest Paths with Negative Length Edges

Negative Lengths

Suppose we allow edges to have negative lengths.

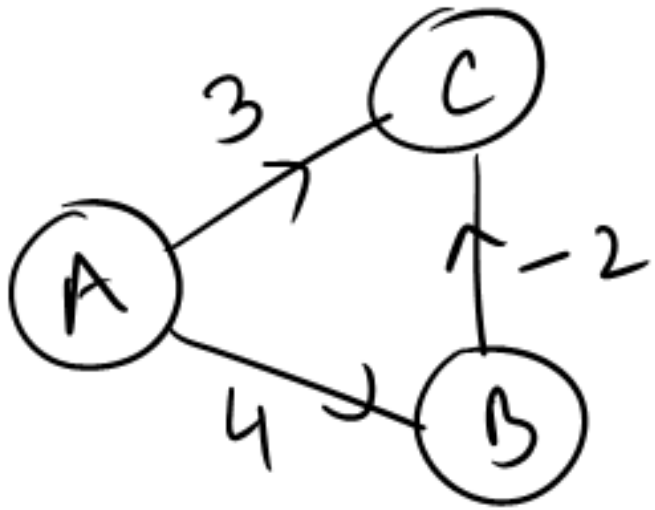
Example:



Negative Lengths

Suppose we allow edges to have negative lengths.

Example:

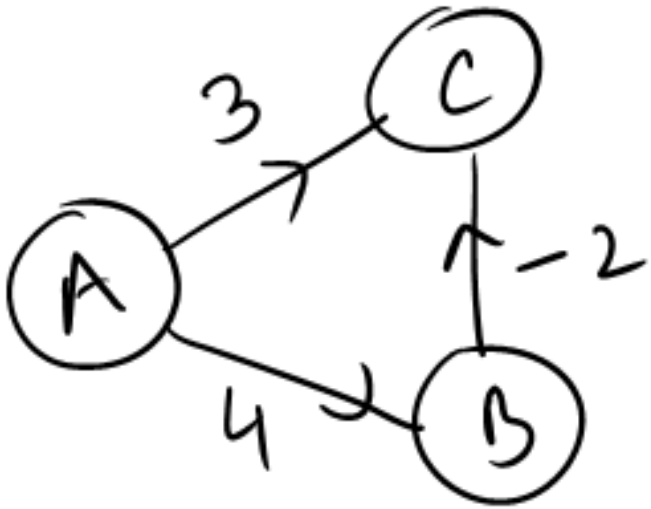


Dijkstra's algorithm can fail! Why?

Negative Lengths

Suppose we allow edges to have negative lengths.

Example:



Dijkstra's algorithm can fail! Why?
If $s \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$ is a shortest path from s to v_k then it is no longer true that $d(s, v_i) \leq d(s, v_k)$ for all $i < k$.

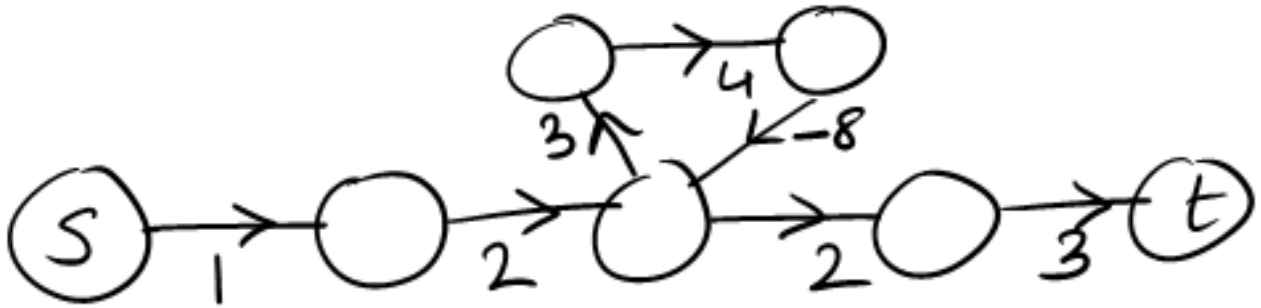
Negative Length Cycles

Cycle C is a negative length cycle if sum of edge lengths is < 0.

Negative Length Cycles

Cycle C is a negative length cycle if sum of edge lengths is < 0.

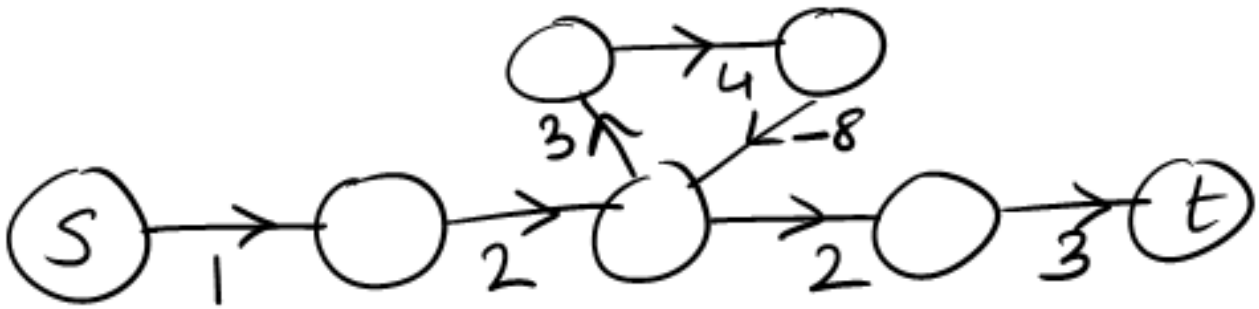
If s can reach a negative length cycle C and C can reach t then shortest path length from s to t is $-\infty$.



Negative Length Cycles

Cycle C is a negative length cycle if sum of edge lengths is < 0.

If s can reach a negative length cycle C and C can reach t then shortest path length from s to t is $-\infty$.

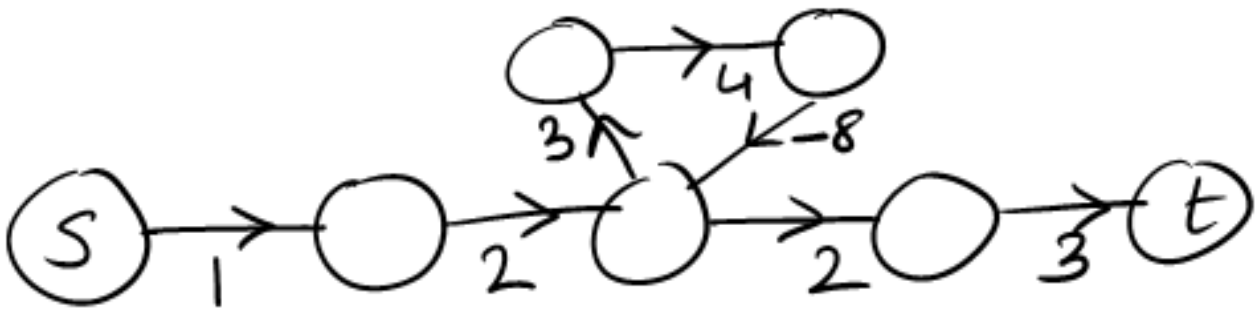


What about shortest length *simple* path? Implies we can find *longest* simple path in a graph with positive edge lengths.

Negative Length Cycles

Cycle C is a negative length cycle if sum of edge lengths is < 0.

If s can reach a negative length cycle C and C can reach t then shortest path length from s to t is $-\infty$.



What about shortest length *simple* path? Implies we can find *longest* simple path in a graph with positive edge lengths. NP-Hard!

Single Source Shortest Paths with Negative Lengths

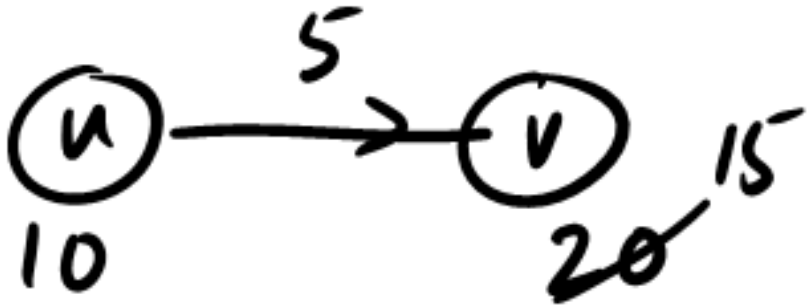
Problem: Given G with edge lengths (can be negative) and s, t either find a shortest length path or show that it is $-\infty$ by exhibiting a negative length cycle.

Assume No Negative Length Cycles

Then there exists a shortest path
$s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$

Assume No Negative Length Cycles

Then there exists a shortest path
$s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$



Keep track of distances from s: dist(u) is an overestimate and is initially ∞ for all u
Basic operation: given edge $e = (u, v)$, update(e)

```
update(e=(u,v))
    if (dist(v) > dist(u) + ℓ(u,v)) then
        dist(v) = dist(u) + ℓ(u,v)
```

Assume No Negative Length Cycles

Then there exists a shortest path
$s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$

Keep track of distances from s: dist(u) is an overestimate and is initially ∞ for all u
Basic operation: given edge $e = (u, v)$, update(e)

```
update(e=(u,v))
    if (dist(v) > dist(u) + ℓ(u,v)) then
        dist(v) = dist(u) + ℓ(u,v)
```

update(e) is always safe and valid even with negative edge lengths!

Order of Updates

Suppose $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ is a shortest path.

Observation: If update((s, v_1)), update((v_1, v_2)), ..., update((v_{k-1}, v_k)) are done in *order* then dist(v_k) is the correct shortest path length!

Order of Updates

Suppose $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ is a shortest path.

Observation: If update$((s, v_1))$, update$((v_1, v_2))$, ..., update$((v_{k-1}, v_k))$ are done in *order* then dist(v_k) is the correct shortest path length!

How do we know the order?

Order of Updates

Suppose $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$ is a shortest path.

Observation: If update$((s, v_1))$, update$((v_1, v_2))$, ..., update$((v_{k-1}, v_k))$ are done in *order* then dist(v_k) is the correct shortest path length!

How do we know the order?

We don't. So update all edges and iterate $|V| - 1$ times!

Bellman-Ford Algorithm

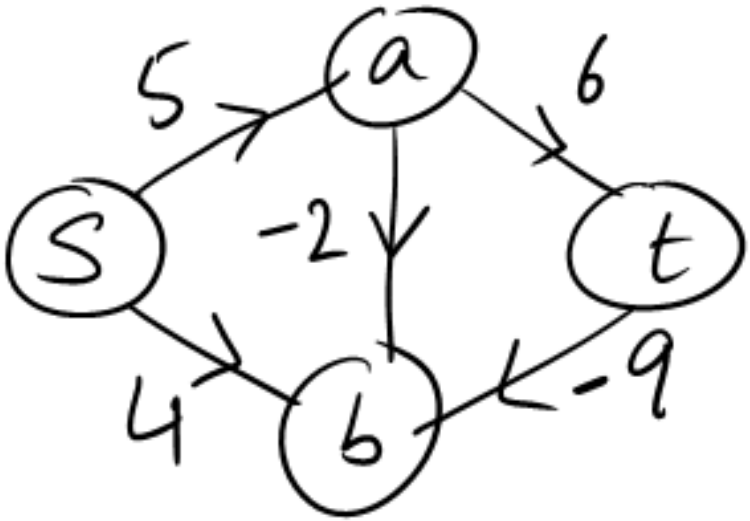
Given $G = (V, E)$ and edge lengths $\ell(e)$ (potentially negative) and a source node s.

Assume no negative length cycle

Find shortest path lengths from s to all nodes reachable from s.

```
for each u ∈ V
    dist(u) = ∞
dist(s) = 0

for i = 1 to |V| − 1 do
    for each edge e = (u, v) do
        update(e)

return dist values
```

Example



	s	a	b	t
	0	∞	∞	∞
1	0	5	4	∞
2	0	5	3	11
3	0	5	2	11

Correctness

Consider shortest $s \longrightarrow t$ path.

$$s \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_k = t$$

Observation: for each $i < k$,

$$s \longrightarrow v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_i$$

is shortest $s \longrightarrow v_i$ path. Why?

Correctness

Consider shortest $s \rightarrow t$ path.

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k = t$$

Observation: for each $i < k$,

$$s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_i$$

is shortest $s \rightarrow v_i$ path. Why?

- After iteration 1, dist$(v_1) = d(s, v_1)$ because of update((s,v_1))
- After iteration 2, dist$(v_2) = d(s, v_2)$ because dist(v_1) was correct in previous iteration *and* because of update((v_1,v_2))
- So on by induction

Correctness

Formally

Lemma

If G has no negative-length cycle that s can reach, then $dist(u)$ is the correct shortest path length from s to v after $|V| - 1$ iterations of Bellman-Ford algorithm.

Finding the Paths

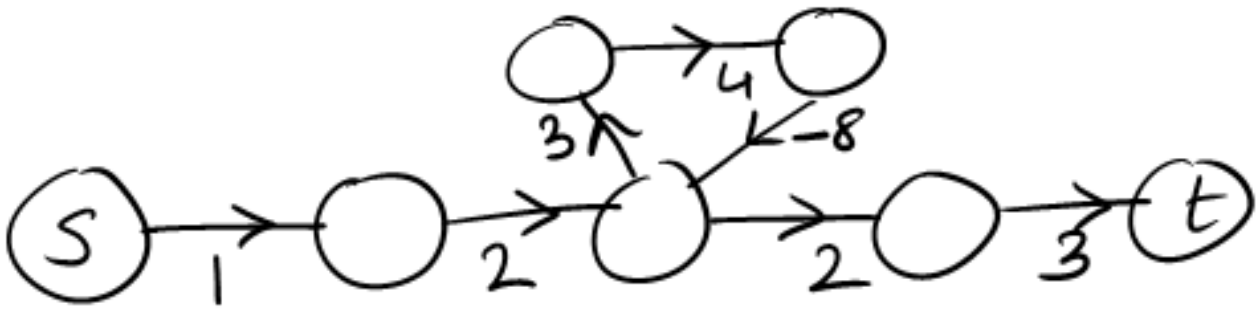
How do we find the paths themselves?

```
for each u ∈ V
    dist(u) = ∞
    prev(u) = nil dist(s) = 0

for i = 1 to |V| − 1 do
    for each edge e = (u, v) do
        update(e)
        if dist(v) changed
            prev(v) = u

return dist values
```

To find shortest path for v follow prev links to s.

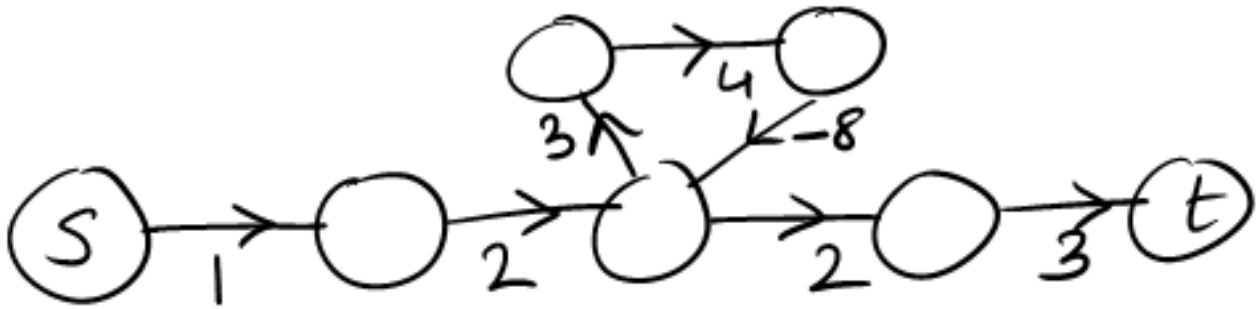
Detecting Negative Cycles

What if G has a negative cycle that s can reach?



Detecting Negative Cycles

What if G has a negative cycle that s can reach?



We run Bellman-Ford algorithm for one more iteration. If any dist value changes then there is a negative length cycle!

Bellman-Ford to detect Negative Cycles

```
for each u ∈ V
    dist(u) = ∞
dist(s) = 0

for i = 1 to |V| − 1 do
    for each edge e = (u, v) do
        update(e)

for each edge e = (u, v) do
    update(e)
    if dist(v) changed
        STOP: G has a negative cycle

G has no negative cycle
```

Correctness

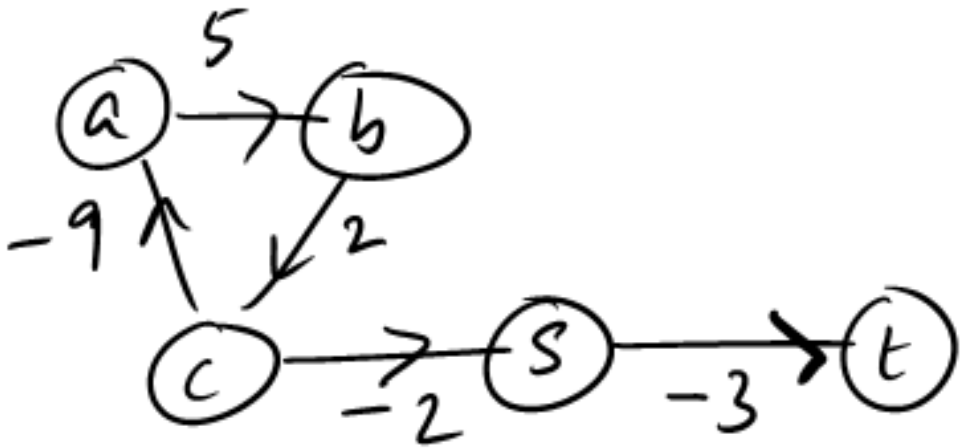
- If G has no negative length cycle then dist values are correct for all vertices after $|V| - 1$ iterations.
- If dist values are correct implies they won't change in any subsequent iteration
- Therefore dist value changed in iteration $|V|$ implies G has a negative length cycle

Can use prev links to find negative length cycle.

Any Negative Length Cycle?

Problem: Given G and edge lengths, does it have a negative length cycle?

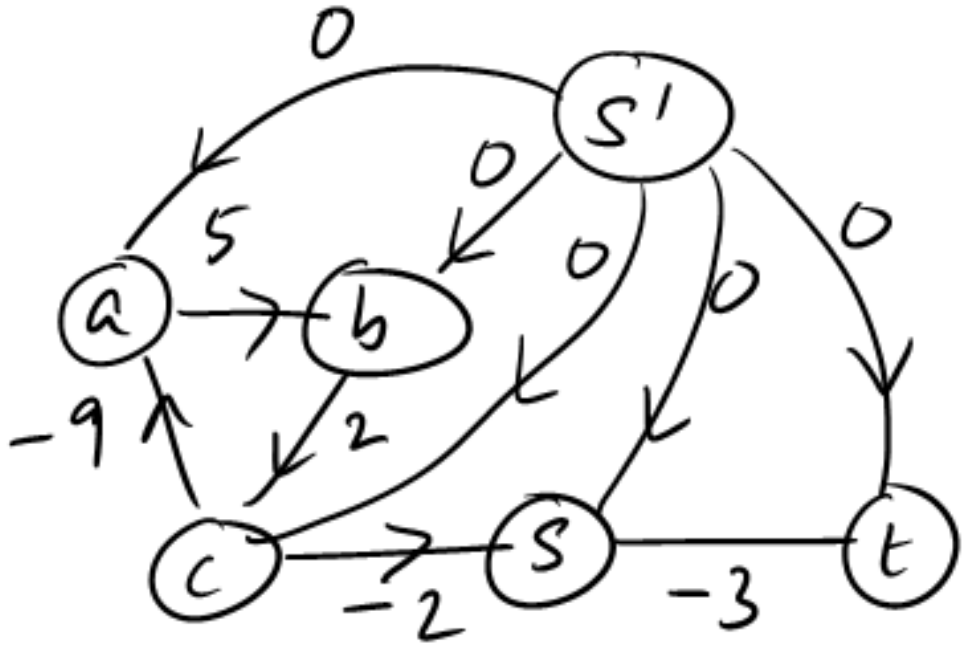
We say that if we do Bellman-Ford with s as source then we find negative length cycle if s can reach it. What if s cannot reach the cycle? (Only relevant for directed graphs)



Run Bellman-Ford from each node v? n times! Can we do better?

Run Bellman-Ford from each node v? n times! Can we do better?

Add a new dummy node s and connect it to all nodes with positive lengths. Bellman-Ford from s will fill find a negative length cycle if there is one.



Running time for Bellman-Ford

$O(n)$ iterations each updating $O(m)$ edges.
Total running time is $O(nm)$.

Why Negative Lengths?

Several Applications

- Shortest path problems useful in modeling many situations — in some negative lenths are natural
- Negative length cycle can be used to find arbitrage opportunities in currency trading (see book)
- Important sub-routine in algorithms for more general problem: minimum-cost flow

Part III

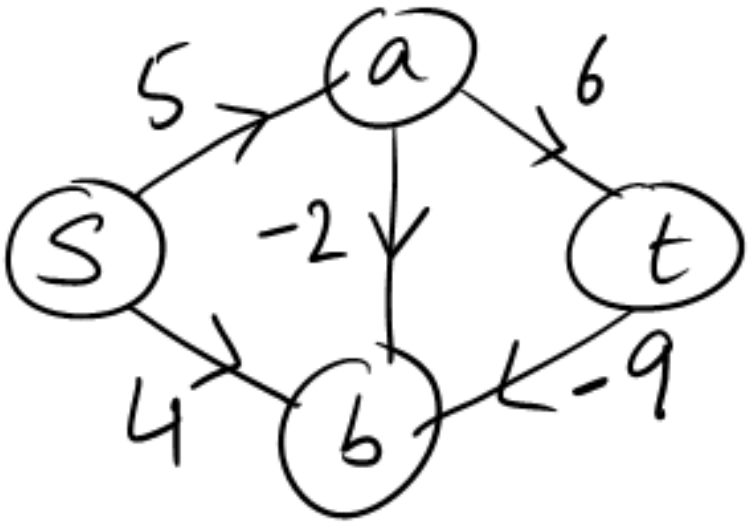
Shortest Paths in DAGs

Shortest Paths in a DAG

Given DAG $G = (V, E)$ and edge lengths ℓ, source s. Find shortest path lengths from s to all nodes reachable from s.

What is the advantage of a DAG?

- No cycles and hence no negative length cycles! Hence can find shortest paths even for negative length edges
- Can order nodes using topological sort



s a t b

Shortest Paths in a DAG

Given DAG $G = (V, E)$ and edge lengths ℓ, source s. Find shortest path lengths from s to all nodes reachable from s.

What is the advantage of a DAG?

- No cycles and hence no negative length cycles! Hence can find shortest paths even for negative length edges
- Can order nodes using topological sort

Suppose a topological order of G is $s = v_1, v_2, v_{i+1}, \ldots, v_n$
(eliminate all nodes not reachable from s)
Observation: shortest path from s to v_i cannot use any node from $v_{i+1}, \ldots, v_n$

Algorithm for DAGs

$s = v_1, v_2, v_{i+1}, \ldots, v_n$ is a topological sort of G

```
for i = 1 to n
    dist(v_i) = ∞
dist(s) = 0

for i = 2 to n do
    for each edge e = (u, v_i) do (* edges entering v_i *)
        update(e)

return dist values
```

$O(m + n)$ time algorithm! Works for negative edge lengths and hence can find *longest* paths in a DAG