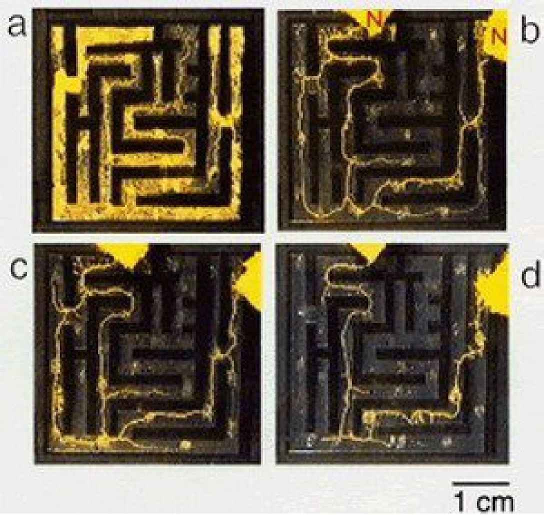
Aside: Slime Mould Solving Shortest Path



Dynamic Programming: Shortest Paths

Lecture 5

Jan 30, 2018

Most slides are courtesy Prof. Chekuri

Part I

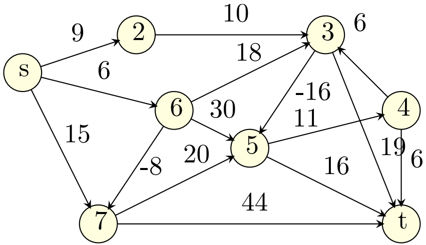
Shortest Paths with Negative Length Edges

Single-Source Shortest Paths with Negative Edge Lengths

Single-Source Shortest Path Problems

Input: A *directed* graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.

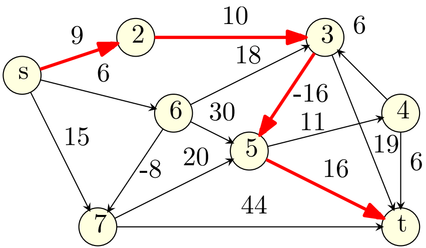


Single-Source Shortest Paths with Negative Edge Lengths

Single-Source Shortest Path Problems

Input: A *directed* graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

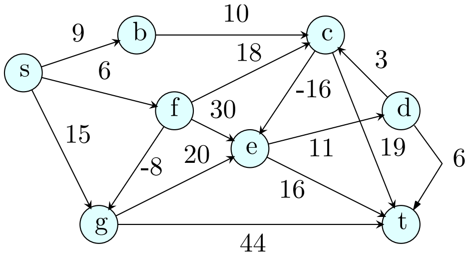
1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.



Negative Length Cycles

Definition

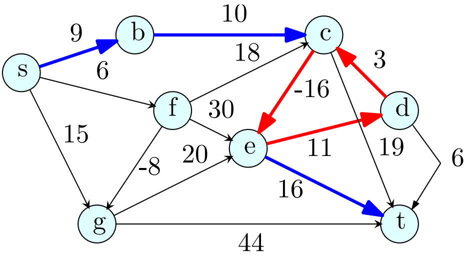
A cycle C is a negative length cycle if the sum of the edge lengths of C is negative.



Negative Length Cycles

Definition

A cycle C is a negative length cycle if the sum of the edge lengths of C is negative.



Shortest Paths and Negative Cycles

Given $G = (V, E)$ with edge lengths and s, t. Suppose

1. G has a negative length cycle C, and
2. s can reach C and C can reach t.

Question: What is the shortest **distance** from s to t?

Shortest Paths and Negative Cycles

Given $G = (V, E)$ with edge lengths and s, t. Suppose

1. G has a negative length cycle C, and
2. s can reach C and C can reach t.

Question: What is the shortest **distance** from s to t?

Possible answers: Define shortest distance to be:

1. undefined, that is $-\infty$
   OR
2. the length of a shortest **simple** path from s to t.

Shortest Paths and Negative Cycles

Given $G = (V, E)$ with edge lengths and s, t. Suppose

1. G has a negative length cycle C, and
2. s can reach C and C can reach t.

Question: What is the shortest **distance** from s to t?

Possible answers: Define shortest distance to be:

1. undefined, that is $-\infty$
   OR
2. the length of a shortest **simple** path from s to t. **NP-Hard**!

Alterantively: Finding Shortest Walks

Given a graph $G = (V, E)$:

1. A path is a sequence of *distinct* vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.

Alterantively: Finding Shortest Walks

Given a graph $G = (V, E)$:

1. A path is a sequence of *distinct* vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
2. A walk is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$. Vertices are allowed to repeat.

Alterantively: Finding Shortest Walks

Given a graph $G = (V, E)$:

1. A path is a sequence of *distinct* vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
2. A walk is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$. Vertices are allowed to repeat.

Define $dist(u, v)$ to be the length of a **shortest walk** from u to v.

Alterantively: Finding Shortest Walks

Given a graph $G = (V, E)$:

1. A path is a sequence of *distinct* vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
2. A walk is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$. Vertices are allowed to repeat.

Define $dist(u, v)$ to be the length of a **shortest walk** from u to v.

1. If there is a walk from u to v that contains negative length cycle then $dist(u, v) = -\infty$

Alterantively: Finding Shortest Walks

Given a graph $G = (V, E)$:

1. A path is a sequence of *distinct* vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$.
2. A walk is a sequence of vertices $v_1, v_2, \ldots, v_k$ such that $(v_i, v_{i+1}) \in E$ for $1 \leq i \leq k - 1$. Vertices are allowed to repeat.

Define $dist(u, v)$ to be the length of a **shortest walk** from u to v.

1. If there is a walk from u to v that contains negative length cycle then $dist(u, v) = -\infty$
2. Else there is a path with at most $n - 1$ edges whose length is equal to the length of a shortest walk and $dist(u, v)$ is finite

Helpful to think about walks

Algorithmic Problems

Input: A directed graph $G = (V, E)$ with edge lengths (could be negative). For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

Questions:

1. Given nodes s, t, either find a negative length cycle C that s can reach or find a shortest path from s to t.
2. Given node s, either find a negative length cycle C that s can reach or find shortest path distances from s to all reachable nodes.
3. Check if G has a negative length cycle or not.

Shortest Paths with Negative Edge Lengths

In Undirected Graphs

Note: Negative cycle detection in undirected graph can not be reduced to directed gaph by bi-directing edges, why?

Shortest Paths with Negative Edge Lengths

In Undirected Graphs

Note: Negative cycle detection in undirected graph can not be reduced to directed gaph by bi-directing edges, why?

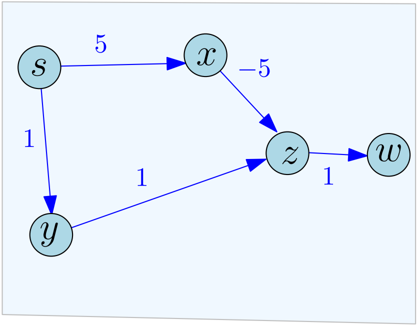
Problem can be solved efficiently in undirected graphs but algorithms are different and more involved than those for directed graphs. Need min-cost matchings which we will see later in the course.

Why Negative Lengths?

Several Applications

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

What are the distances computed by Dijkstra's algorithm?



The distance as computed by Dijkstra algorithm starting from s:

(A) $s = 0$, $x = 5$, $y = 1$, $z = 0$.

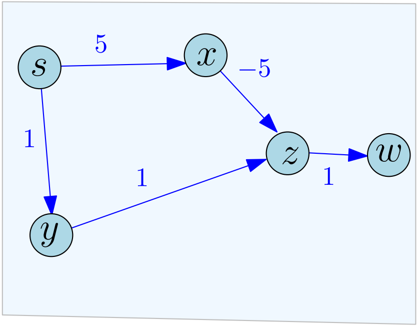
(B) $s = 0$, $x = 1$, $y = 2$, $z = 5$.

(C) $s = 0$, $x = 5$, $y = 1$, $z = 2$.

(D) IDK.

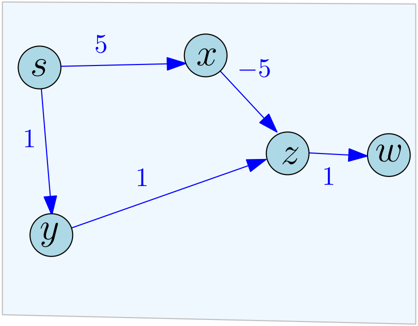
Dijkstra's Algorithm and Negative Lengths

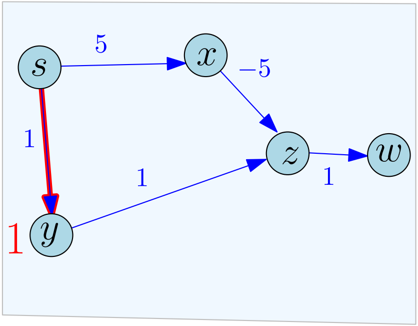
With negative length edges, Dijkstra's algorithm can fail



Dijkstra's Algorithm and Negative Lengths

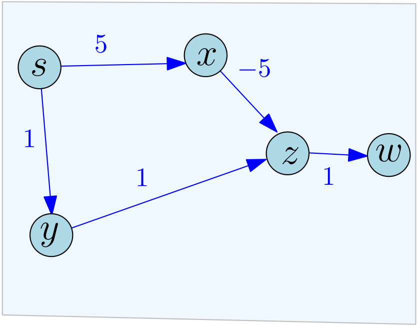
With negative length edges, Dijkstra's algorithm can fail

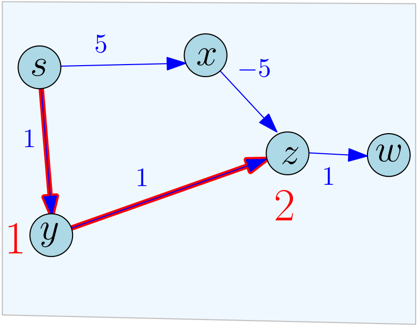




Dijkstra's Algorithm and Negative Lengths

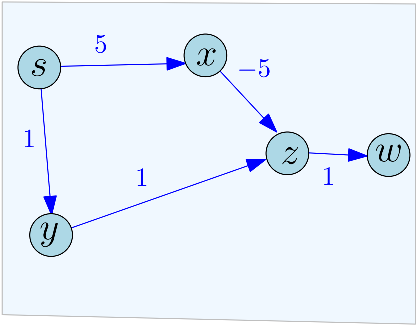
With negative length edges, Dijkstra's algorithm can fail

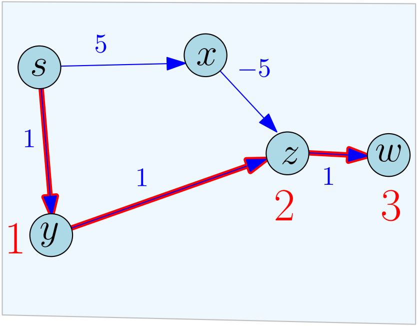




Dijkstra's Algorithm and Negative Lengths

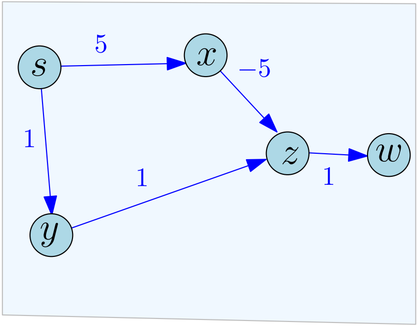
With negative length edges, Dijkstra's algorithm can fail

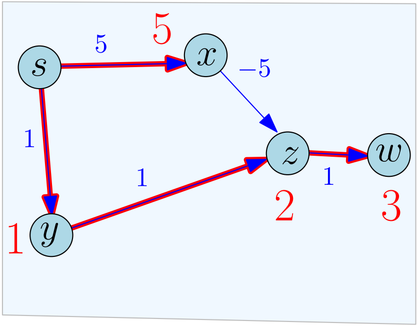




Dijkstra's Algorithm and Negative Lengths

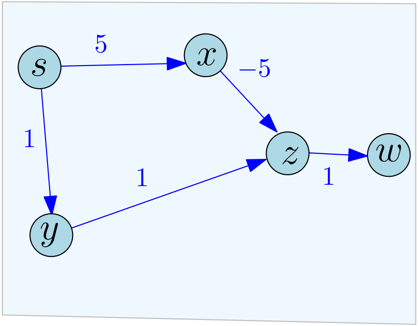
With negative length edges, Dijkstra's algorithm can fail

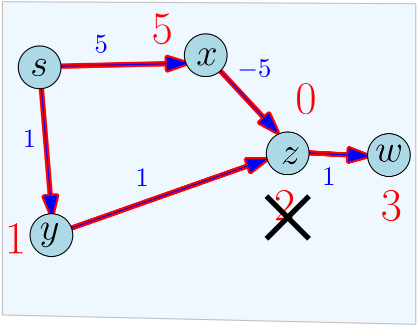




Dijkstra's Algorithm and Negative Lengths

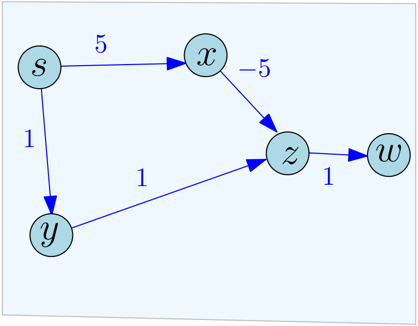
With negative length edges, Dijkstra's algorithm can fail

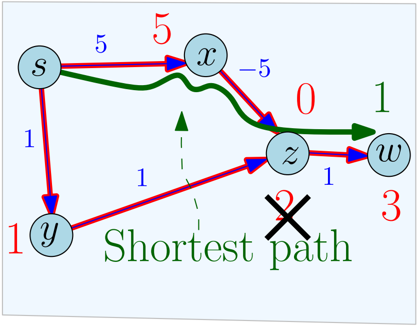




Dijkstra's Algorithm and Negative Lengths

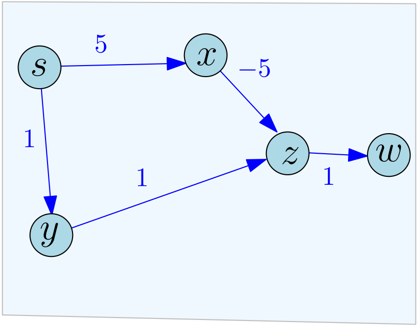
With negative length edges, Dijkstra's algorithm can fail

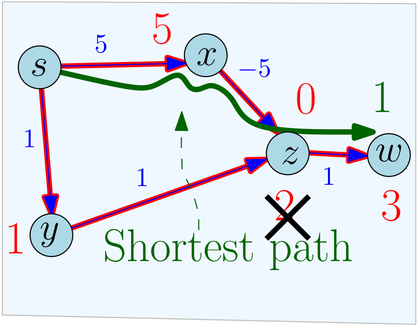




Dijkstra's Algorithm and Negative Lengths

With negative length edges, Dijkstra's algorithm can fail





False assumption: Dijkstra's algorithm is based on the assumption that if $s = v_0 \to v_1 \to v_2 \ldots \to v_k$ is a shortest path from s to v_k then $dist(s, v_i) \leq dist(s, v_{i+1})$ for $0 \leq i < k$. Holds true only for non-negative edge lengths.

Shortest Paths with Negative Lengths

Lemma

Let G *be a directed graph with arbitrary edge lengths. If* $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ *is a shortest path from* s *to* v_k *then for* $1 \leq i < k$*:*

1. $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_i$ *is a shortest path from* s *to* v_i

Shortest Paths with Negative Lengths

Lemma

Let G be a directed graph with arbitrary edge lengths. If $s \to v_1 \to v_2 \to \ldots \to v_k$ is a shortest path from s to v_k then for $1 \leq i < k$:

1. $s \to v_1 \to v_2 \to \ldots \to v_i$ *is a shortest path from* s *to* v_i
2. *False:* $dist(s, v_i) \leq dist(s, v_k)$ *for* $1 \leq i < k$. *Holds true only for non-negative edge lengths.*

Shortest Paths with Negative Lengths

Lemma

Let G be a directed graph with arbitrary edge lengths. If $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is a shortest path from s to v_k then for $1 \leq i < k$:

1. $s \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_i$ *is a shortest path from* s *to* v_i
2. *False:* $dist(s, v_i) \leq dist(s, v_k)$ *for* $1 \leq i < k$. *Holds true only for non-negative edge lengths.*

Cannot explore nodes in increasing order of distance! We need other strategies.

Shortest Paths and Recursion

1. Compute the shortest path distance from s to t recursively?
2. What are the smaller sub-problems?

Shortest Paths and Recursion

1. Compute the shortest path distance from s to t recursively?
2. What are the smaller sub-problems?

Lemma

Let G be a directed graph with arbitrary edge lengths. If $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is a shortest path from s to v_k then for $1 \leq i < k$:

1. *$s = v_0 \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_i$ is a shortest path from s to v_i*

Shortest Paths and Recursion

1. Compute the shortest path distance from s to t recursively?
2. What are the smaller sub-problems?

Lemma

Let G be a directed graph with arbitrary edge lengths. If $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ is a shortest path from s to v_k then for $1 \leq i < k$:

1. *$s = v_0 \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_i$ is a shortest path from s to v_i*

Sub-problem idea: paths of fewer hops/edges

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.
Assume that all nodes can be reached by s in G. (Remove nodes unreachable from s).

$d(v, k)$: shortest walk length from s to v using at most k edges (∞ if none exists).

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.
Assume that all nodes can be reached by s in G. (Remove nodes unreachable from s).

$d(v, k)$: shortest walk length from s to v using at most k edges (∞ if none exists).

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.
Assume that all nodes can be reached by s in G. (Remove nodes unreachable from s).

$d(v, k)$: shortest walk length from s to v using at most k edges (∞ if none exists).

Recursion for $d(v, k)$:

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.
Assume that all nodes can be reached by s in G. (Remove nodes unreachable from s).

$d(v, k)$: shortest walk length from s to v using at most k edges (∞ if none exists).

Recursion for $d(v, k)$:

$$d(v, k) = \min \begin{cases} \min_{u \in V}(d(u, k-1) + \ell(u, v)). \\ d(v, k-1) \end{cases}$$

Base case:

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.
Assume that all nodes can be reached by s in G. (Remove nodes unreachable from s).

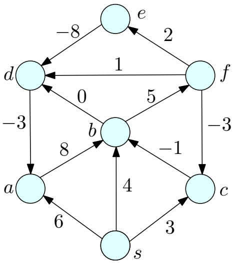
$d(v,k)$: shortest walk length from s to v using at most k edges (∞ if none exists).

Recursion for $d(v,k)$:

$$d(v,k) = \min \begin{cases} \min_{u \in V}(d(u,k-1) + \ell(u,v)). \\ d(v,k-1) \end{cases}$$

Base case: $d(s,0) = 0$ and $d(v,0) = \infty$ for all $v \neq s$.

Example



A Basic Lemma

Lemma

Assume s *can reach all nodes in* $G = (V, E)$*. Then,*

1. *There is a negative length cycle in* G *iff* $d(v, n) < d(v, n-1)$ *for some node* $v \in V$*.*
2. *If there is no negative length cycle in* G *then* $dist(s, v) = d(v, n-1)$ *for all* $v \in V$*.*

Bellman-Ford Algorithm

for each $u \in V$ **do**
    $d(u, 0) \leftarrow \infty$
$d(s, 0) \leftarrow 0$

Bellman-Ford Algorithm

```
for each u ∈ V do
    d(u, 0) ← ∞
d(s, 0) ← 0

for k = 1 to n do
        for each v ∈ V do
            d(v, k) ← d(v, k − 1)
            for each edge (u, v) ∈ In(v) do
                d(v, k) = min{d(v, k), d(u, k − 1) + ℓ(u, v)}
```

Bellman-Ford Algorithm

```
for each u ∈ V do
    d(u,0) ← ∞
d(s,0) ← 0

for k = 1 to n do
        for each v ∈ V do
            d(v,k) ← d(v,k − 1)
            for each edge (u,v) ∈ In(v) do
                d(v,k) = min{d(v,k), d(u,k − 1) + ℓ(u,v)}


for each v ∈ V do
        dist(s,v) ← d(v,n − 1)
        If d(v,n) < d(v,n − 1)
            Return ‘‘Negative Cycle in G’’
```

Bellman-Ford Algorithm

for each $u \in V$ do
    $d(u,0) \leftarrow \infty$
$d(s,0) \leftarrow 0$

for $k = 1$ to n do
        for each $v \in V$ do
            $d(v,k) \leftarrow d(v,k-1)$
            for each edge $(u,v) \in In(v)$ do
                $d(v,k) = \min\{d(v,k), d(u,k-1) + \ell(u,v)\}$

for each $v \in V$ do
        $\mathrm{dist}(s,v) \leftarrow d(v,n-1)$
        If $d(v,n) < d(v,n-1)$
            Return ''Negative Cycle in G''

Running time:

Bellman-Ford Algorithm

```
for each u ∈ V do
    d(u, 0) ← ∞
d(s, 0) ← 0

for k = 1 to n do
        for each v ∈ V do
            d(v, k) ← d(v, k − 1)
            for each edge (u, v) ∈ In(v) do
                d(v, k) = min{d(v, k), d(u, k − 1) + ℓ(u, v)}


for each v ∈ V do
        dist(s, v) ← d(v, n − 1)
        If d(v, n) < d(v, n − 1)
            Return ‘‘Negative Cycle in G’’
```

Running time: $O(mn)$

Bellman-Ford Algorithm

```
for each u ∈ V do
    d(u,0) ← ∞
d(s,0) ← 0

for k = 1 to n do
        for each v ∈ V do
            d(v,k) ← d(v,k − 1)
            for each edge (u,v) ∈ In(v) do
                d(v,k) = min{d(v,k), d(u,k − 1) + ℓ(u,v)}


for each v ∈ V do
        dist(s,v) ← d(v,n − 1)
        If d(v,n) < d(v,n − 1)
            Return ‘‘Negative Cycle in G’’
```

Running time: $O(mn)$ Space:

Bellman-Ford Algorithm

```
for each u ∈ V do
    d(u, 0) ← ∞
d(s, 0) ← 0

for k = 1 to n do
        for each v ∈ V do
            d(v, k) ← d(v, k − 1)
            for each edge (u, v) ∈ In(v) do
                d(v, k) = min{d(v, k), d(u, k − 1) + ℓ(u, v)}


for each v ∈ V do
        dist(s, v) ← d(v, n − 1)
        If d(v, n) < d(v, n − 1)
            Return ‘‘Negative Cycle in G’’
```

Running time: $O(mn)$ Space: $O(m + n^2)$

Bellman-Ford Algorithm

```
for each u ∈ V do
    d(u,0) ← ∞
d(s,0) ← 0

for k = 1 to n do
        for each v ∈ V do
            d(v,k) ← d(v,k − 1)
            for each edge (u,v) ∈ In(v) do
                d(v,k) = min{d(v,k), d(u,k − 1) + ℓ(u,v)}


for each v ∈ V do
        dist(s,v) ← d(v,n − 1)
        If d(v,n) < d(v,n − 1)
            Return ``Negative Cycle in G''
```

Running time: $O(mn)$ Space: $O(m + n^2)$

Space can be reduced to $O(m + n)$.

Bellman-Ford with Space Saving

```
for each u ∈ V do
    d(u) ← ∞
d(s) ← 0

for k = 1 to n − 1 do
        for each v ∈ V do
            for each edge (u, v) ∈ In(v) do
                d(v) = min{d(v), d(u) + ℓ(u, v)}
```

Bellman-Ford with Space Saving

```
for each u ∈ V do
    d(u) ← ∞
d(s) ← 0

for k = 1 to n − 1 do
        for each v ∈ V do
            for each edge (u, v) ∈ In(v) do
                d(v) = min{d(v), d(u) + ℓ(u, v)}
(* One more iteration to check if distances change *)
for each v ∈ V do
    for each edge (u, v) ∈ In(v) do
        if (d(v) > d(u) + ℓ(u, v))
            Output ‘‘Negative Cycle’’

for each v ∈ V do
        dist(s, v) ← d(v)
```

Exercise: Show that this algorithm achieves same result.

Correctness of the Bellman-Ford Algorithm

Via induction show: For each v, $d(v, k)$ is the length of a shortest walk from s to v with *at most* k hops.

Correctness of the Bellman-Ford Algorithm

Via induction show: For each v, $d(v, k)$ is the length of a shortest walk from s to v with *at most* k hops.
And for each $1 \leq k \leq n - 1$, $d(v, k) \leq d(v, k - 1)$.

Correctness of the Bellman-Ford Algorithm

Via induction show: For each v, $d(v, k)$ is the length of a shortest walk from s to v with *at most* k hops.
And for each $1 \leq k \leq n - 1$, $d(v, k) \leq d(v, k - 1)$.

Lemma

Assume s can reach all nodes in $G = (V, E)$. Then,

1. *There is a negative length cycle in G iff $d(v, n) < d(v, n - 1)$ for some node $v \in V$.*
2. *If there is no negative length cycle in G then $dist(s, v) = d(v, n - 1)$ for all $v \in V$.*

Correctness of the Bellman-Ford Algorithm

Via induction show: For each v, $d(v, k)$ is the length of a shortest walk from s to v with *at most* k hops.
And for each $1 \leq k \leq n - 1$, $d(v, k) \leq d(v, k - 1)$.

Lemma

Assume s can reach all nodes in $G = (V, E)$. Then,

1. *There is a negative length cycle in G iff $d(v, n) < d(v, n - 1)$ for some node $v \in V$.*
2. *If there is no negative length cycle in G then $dist(s, v) = d(v, n - 1)$ for all $v \in V$.*

Exercise: Prove algorithm correctness from above two.

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G *then* $d(v, h) \geq d(v, n-1)$ *for all* $h \geq n$ *and for all* $v \in V$.

Proof.

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G *then* $d(v,h) \geq d(v,n-1)$ *for all* $h \geq n$ *and for all* $v \in V$.

Proof.

By contradiction. Suppose $d(v,h) < d(v,n-1)$ for some $h \geq n$ and some v.

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G then $d(v, h) \geq d(v, n-1)$ for all $h \geq n$ and for all $v \in V$.

Proof.

By contradiction. Suppose $d(v, h) < d(v, n-1)$ for some $h \geq n$ and some v. Choose smallest such h. Let P be a s-v walk with h edges of length $d(v, h)$. Since $h \geq n$, P has a cycle C.

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G then $d(v, h) \geq d(v, n-1)$ for all $h \geq n$ and for all $v \in V$.

Proof.

By contradiction. Suppose $d(v, h) < d(v, n-1)$ for some $h \geq n$ and some v. Choose smallest such h. Let P be a s-v walk with h edges of length $d(v, h)$. Since $h \geq n$, P has a cycle C. Since $\ell(C) \geq 0$, the walk P' obtained by removing C from P satisfies: $\ell(P') \leq \ell(P)$

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G *then* $d(v, h) \geq d(v, n-1)$ *for all* $h \geq n$ *and for all* $v \in V$.

Proof.

By contradiction. Suppose $d(v, h) < d(v, n-1)$ for some $h \geq n$ and some v. Choose smallest such h. Let P be a s-v walk with h edges of length $d(v, h)$. Since $h \geq n$, P has a cycle C. Since $\ell(C) \geq 0$, the walk P' obtained by removing C from P satisfies: $\ell(P') \leq \ell(P) = d(v, h) < d(v, n-1)$.

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G then $d(v, h) \geq d(v, n-1)$ for all $h \geq n$ and for all $v \in V$.

Proof.

By contradiction. Suppose $d(v, h) < d(v, n-1)$ for some $h \geq n$ and some v. Choose smallest such h. Let P be a s-v walk with h edges of length $d(v, h)$. Since $h \geq n$, P has a cycle C. Since $\ell(C) \geq 0$, the walk P' obtained by removing C from P satisfies: $\ell(P') \leq \ell(P) = d(v, h) < d(v, n-1)$. If P' has $\leq n-1$ edges then $d(v, n-1) \leq \ell(P') < d(v, n-1)$ which is a contradiction.

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G then $d(v, h) \geq d(v, n-1)$ for all $h \geq n$ and for all $v \in V$.

Proof.

By contradiction. Suppose $d(v, h) < d(v, n-1)$ for some $h \geq n$ and some v. Choose smallest such h. Let P be a s-v walk with h edges of length $d(v, h)$. Since $h \geq n$, P has a cycle C. Since $\ell(C) \geq 0$, the walk P' obtained by removing C from P satisfies: $\ell(P') \leq \ell(P) = d(v, h) < d(v, n-1)$. If P' has $\leq n-1$ edges then $d(v, n-1) \leq \ell(P') < d(v, n-1)$ which is a contradiction.

Therefor P' has $h' \geq n$ edges

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G then $d(v, h) \geq d(v, n-1)$ for all $h \geq n$ and for all $v \in V$.

Proof.

By contradiction. Suppose $d(v, h) < d(v, n-1)$ for some $h \geq n$ and some v. Choose smallest such h. Let P be a s-v walk with h edges of length $d(v, h)$. Since $h \geq n$, P has a cycle C. Since $\ell(C) \geq 0$, the walk P' obtained by removing C from P satisfies: $\ell(P') \leq \ell(P) = d(v, h) < d(v, n-1)$. If P' has $\leq n-1$ edges then $d(v, n-1) \leq \ell(P') < d(v, n-1)$ which is a contradiction.

Therefor P' has $h' \geq n$ edges but P' is a shorter walk than P with $\ell(P') < d(v, n-1)$ contradicting choice of h. □

Proof of Lemma

Proposition

Suppose there is no negative length cycle in G *then* $d(v, h) \geq d(v, n-1)$ *for all* $h \geq n$ *and for all* $v \in V$.

Proof of Lemma cond

Proposition

If G has a negative length cycle then there is some v such that $d(v, n) < d(v, n-1)$.

Proof.

Proof of Lemma cond

Proposition

If G has a negative length cycle then there is some v such that $d(v, n) < d(v, n-1)$.

Proof.

Suppose not.

Proof of Lemma cond

Proposition

If G has a negative length cycle then there is some v such that $d(v, n) < d(v, n-1)$.

Proof.

Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_h \rightarrow v_1$ be negative length cycle reachable from s. $d(v_i, n-1)$ is finite for $1 \leq i \leq h$ since C is reachable from s.

Proof of Lemma cond

Proposition

If G has a negative length cycle then there is some v such that $d(v, n) < d(v, n-1)$.

Proof.

Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_h \rightarrow v_1$ be negative length cycle reachable from s. $d(v_i, n-1)$ is finite for $1 \leq i \leq h$ since C is reachable from s. By assumption $d(v, n) \geq d(v, n-1)$ for all $v \in C$; this means

Proof of Lemma cond

Proposition

If G has a negative length cycle then there is some v such that $d(v, n) < d(v, n-1)$.

Proof.

Suppose not. Let $C = v_1 \to v_2 \to \ldots \to v_h \to v_1$ be negative length cycle reachable from s. $d(v_i, n-1)$ is finite for $1 \leq i \leq h$ since C is reachable from s. By assumption $d(v, n) \geq d(v, n-1)$ for all $v \in C$; this means
$d(v_i, n-1) \leq d(v_{i-1}, n-1) + \ell(v_{i-1}, v_i)$ for $2 \leq i \leq h$ and
$d(v_1, n-1) \leq d(v_n, n-1) + \ell(v_n, v_1)$.

Proof of Lemma cond

Proposition

If G has a negative length cycle then there is some v such that $d(v, n) < d(v, n-1)$.

Proof.

Suppose not. Let $C = v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_h \rightarrow v_1$ be negative length cycle reachable from s. $d(v_i, n-1)$ is finite for $1 \leq i \leq h$ since C is reachable from s. By assumption $d(v, n) \geq d(v, n-1)$ for all $v \in C$; this means
$d(v_i, n-1) \leq d(v_{i-1}, n-1) + \ell(v_{i-1}, v_i)$ for $2 \leq i \leq h$ and $d(v_1, n-1) \leq d(v_n, n-1) + \ell(v_n, v_1)$. Adding up all these inequalities results in the inequality $0 \leq \ell(C)$ which contradicts the assumption that $\ell(C) < 0$. □

Proof of Lemma contd

Exercise: Finish proof of lemma using the two propositions.

Finding the Paths and a Shortest Path Tree

How do we find a shortest path tree in addition to distances?

- For each v the $d(v)$ can only get smaller as algorithm proceeds.
- If $d(v)$ becomes smaller it is because we found a vertex u such that $d(v) > d(u) + \ell(u, v)$ and we update $d(v) = d(u) + \ell(u, v)$. That is, we found a shorter path to v through u.

Finding the Paths and a Shortest Path Tree

How do we find a shortest path tree in addition to distances?

- For each v the $d(v)$ can only get smaller as algorithm proceeds.
- If $d(v)$ becomes smaller it is because we found a vertex u such that $d(v) > d(u) + \ell(u, v)$ and we update $d(v) = d(u) + \ell(u, v)$. That is, we found a shorter path to v through u.
- For each v have a $prev(v)$ pointer and update it to point to u if v finds a shorter path via u.
- At end of algorithm $prev(v)$ pointers give a shortest path tree oriented towards the source s.

Negative Cycle Detection

Negative Cycle Detection

Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

Negative Cycle Detection

Negative Cycle Detection

Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

1. Bellman-Ford checks whether there is a negative cycle C that is reachable from a specific vertex s. There may negative cycles not reachable from s.

Negative Cycle Detection

Negative Cycle Detection

Given directed graph G with arbitrary edge lengths, does it have a negative length cycle?

1. Bellman-Ford checks whether there is a negative cycle C that is reachable from a specific vertex s. There may negative cycles not reachable from s.
2. Run Bellman-Ford $|V|$ times, one from each node u?

Negative Cycle Detection

1. Add a new node s' and connect it to all nodes of G with zero length edges. Bellman-Ford from s' will fill find a negative length cycle if there is one. Exercise: why does this work?
2. Negative cycle detection can be done with one Bellman-Ford invocation.

Part II

Shortest Paths in DAGs

Shortest Paths in a DAG

Single-Source Shortest Path Problems

Input A directed acyclic graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.

Shortest Paths in a DAG

Single-Source Shortest Path Problems

Input A directed acyclic graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.

Simplification of algorithms for DAGs

1. No cycles and hence no negative length cycles! Hence can find shortest paths even for negative edge weights.
2. Can order nodes using topological sort.

Algorithm for DAGs

1. Want to find shortest paths from s. Ignore nodes not reachable from s.
2. Let $s = v_1, v_2, v_{i+1}, \ldots, v_n$ be a topological sort of G

Algorithm for DAGs

1. Want to find shortest paths from s. Ignore nodes not reachable from s.
2. Let $s = v_1, v_2, v_{i+1}, \ldots, v_n$ be a topological sort of G

Observation:

1. shortest path from s to v_i cannot use any node from $v_{i+1}, \ldots, v_n$

Algorithm for DAGs

1. Want to find shortest paths from s. Ignore nodes not reachable from s.
2. Let $s = v_1, v_2, v_{i+1}, \ldots, v_n$ be a topological sort of G

Observation:

1. shortest path from s to v_i cannot use any node from $v_{i+1}, \ldots, v_n$, since no path from s to v_i uses any of them.
2. can find shortest paths in topological sort order.

Algorithm for DAGs

for $i = 1$ `to` n **do**
    $d(s, v_i) = \infty$
$d(s, s) = 0$

for $i = 1$ `to` $n - 1$ **do**
    for `each edge` (v_i, v_j) `in` $\mathrm{Adj}(v_i)$ **do**
        $d(s, v_j) = \min\{d(s, v_j), d(s, v_i) + \ell(v_i, v_j)\}$

return $d(s, \cdot)$ `values computed`

Algorithm for DAGs

```
for i = 1 to n do
        d(s, v_i) = ∞
d(s, s) = 0

for i = 1 to n − 1 do
        for each edge (v_i, v_j) in Adj(v_i) do
            d(s, v_j) = min{d(s, v_j), d(s, v_i) + ℓ(v_i, v_j)}

return d(s, ·) values computed
```

Correctness by induction: If by the end of ith round $d(s, v_j)$ is the shortest path length from s to v_j for each $1 \leq i \leq j$, then after $(i+1)$th round $d(s, v_{i+1})$ is the shortest path length from s to v_{i+1}.

Algorithm for DAGs

```
for i = 1 to n do
        d(s, v_i) = ∞
d(s, s) = 0

for i = 1 to n − 1 do
        for each edge (v_i, v_j) in Adj(v_i) do
            d(s, v_j) = min{d(s, v_j), d(s, v_i) + ℓ(v_i, v_j)}

return d(s, ·) values computed
```

Correctness by induction: If by the end of ith round $d(s, v_j)$ is the shortest path length from s to v_j for each $1 \leq i \leq j$, then after $(i + 1)$th round $d(s, v_{i+1})$ is the shortest path length from s to v_{i+1}. Use observation in the previous slide.

Algorithm for DAGs

```
for i = 1 to n do
        d(s, v_i) = ∞
d(s, s) = 0

for i = 1 to n − 1 do
        for each edge (v_i, v_j) in Adj(v_i) do
            d(s, v_j) = min{d(s, v_j), d(s, v_i) + ℓ(v_i, v_j)}

return d(s, ·) values computed
```

Correctness by induction: If by the end of ith round $d(s, v_j)$ is the shortest path length from s to v_j for each $1 \leq i \leq j$, then after $(i + 1)$th round $d(s, v_{i+1})$ is the shortest path length from s to v_{i+1}. Use observation in the previous slide.

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

Part III

All Pairs Shortest Paths

Shortest Path Problems

Shortest Path Problems

Input A (undirected or directed) graph $G = (V, E)$ with edge lengths (or costs). For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.
3. Find shortest paths for *all* pairs of nodes.

Single-Source Shortest Paths

Single-Source Shortest Path Problems

Input A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.

Single-Source Shortest Paths

Single-Source Shortest Path Problems

Input A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Given nodes s, t find shortest path from s to t.
2. Given node s find shortest path from s to all other nodes.

Dijkstra's algorithm for non-negative edge lengths. Running time: $O((m + n) \log n)$ with heaps and $O(m + n \log n)$ with advanced priority queues.

Bellman-Ford algorithm for arbitrary edge lengths. Running time: $O(nm)$.

All-Pairs Shortest Paths

All-Pairs Shortest Path Problem

Input A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Find shortest paths for all pairs of nodes.

All-Pairs Shortest Paths

All-Pairs Shortest Path Problem

Input A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Find shortest paths for all pairs of nodes.

Apply single-source algorithms n times, once for each vertex.

1. Non-negative lengths. $O(nm \log n)$ with heaps and $O(nm + n^2 \log n)$ using advanced priority queues.
2. Arbitrary edge lengths: $O(n^2 m)$.
   $\Theta(n^4)$ if $m = \Omega(n^2)$.

All-Pairs Shortest Paths

All-Pairs Shortest Path Problem

Input A (undirected or directed) graph $G = (V, E)$ with edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

1. Find shortest paths for all pairs of nodes.

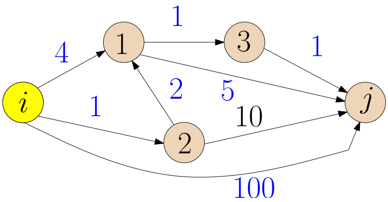
Apply single-source algorithms n times, once for each vertex.

1. Non-negative lengths. $O(nm \log n)$ with heaps and $O(nm + n^2 \log n)$ using advanced priority queues.
2. Arbitrary edge lengths: $O(n^2 m)$.
   $\Theta(n^4)$ if $m = \Omega(n^2)$.

Can we do better?

All-Pairs: Recursion on index of intermediate nodes

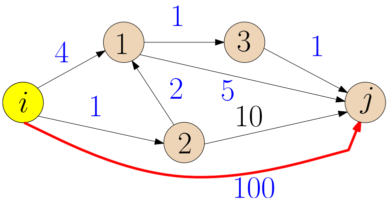
1. Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
2. $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an *intermediate node* is at most k (could be $-\infty$ if there is a negative length cycle).



$$dist(i, j, 0) =$$
$$dist(i, j, 1) =$$
$$dist(i, j, 2) =$$
$$dist(i, j, 3) =$$

All-Pairs: Recursion on index of intermediate nodes

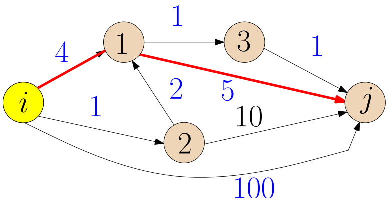
1. Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
2. $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an *intermediate node* is at most k (could be $-\infty$ if there is a negative length cycle).



$$dist(i, j, 0) = 100$$
$$dist(i, j, 1) =$$
$$dist(i, j, 2) =$$
$$dist(i, j, 3) =$$

All-Pairs: Recursion on index of intermediate nodes

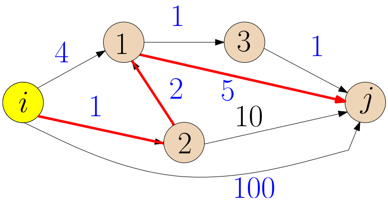
1. Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
2. $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an *intermediate node* is at most k (could be $-\infty$ if there is a negative length cycle).



$$dist(i, j, 0) = 100$$
$$dist(i, j, 1) = 9$$
$$dist(i, j, 2) =$$
$$dist(i, j, 3) =$$

All-Pairs: Recursion on index of intermediate nodes

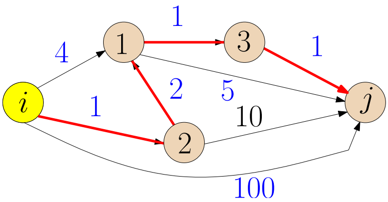
1. Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
2. $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an *intermediate node* is at most k (could be $-\infty$ if there is a negative length cycle).



$$dist(i, j, 0) = 100$$
$$dist(i, j, 1) = 9$$
$$dist(i, j, 2) = 8$$
$$dist(i, j, 3) =$$

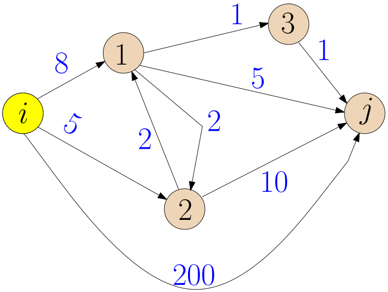
All-Pairs: Recursion on index of intermediate nodes

1. Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
2. $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an *intermediate node* is at most k (could be $-\infty$ if there is a negative length cycle).



$$dist(i, j, 0) = 100$$
$$dist(i, j, 1) = 9$$
$$dist(i, j, 2) = 8$$
$$dist(i, j, 3) = 5$$

For the following graph, dist(i, j, 2) is...



(A) **9**

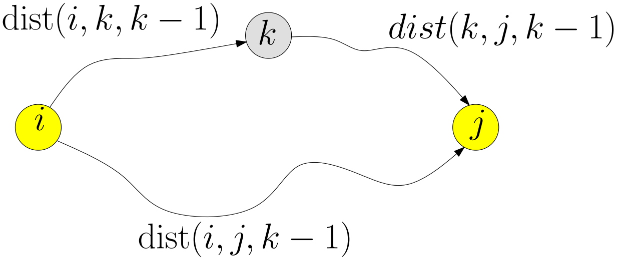
(B) **10**

(C) **11**

(D) **12**

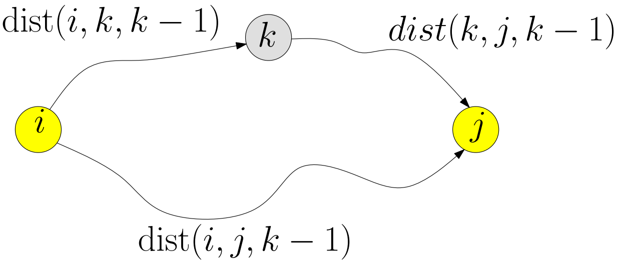
(E) **15**

All-Pairs: Recursion on index of intermediate nodes



$$dist(i,j,k) = \min \begin{cases} dist(i,j,k-1) \\ dist(i,k,k-1) + dist(k,j,k-1) \end{cases}$$

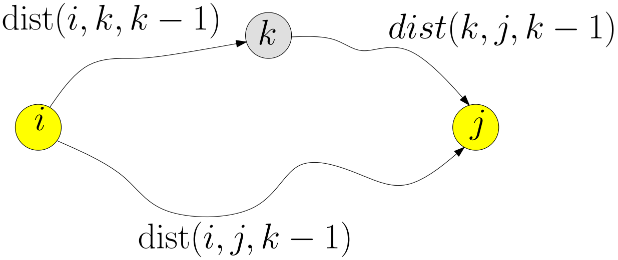
All-Pairs: Recursion on index of intermediate nodes



$$dist(i,j,k) = \min \begin{cases} dist(i,j,k-1) \\ dist(i,k,k-1) + dist(k,j,k-1) \end{cases}$$

Base case: $dist(i,j,0) = \ell(i,j)$ if $(i,j) \in E$, otherwise ∞

All-Pairs: Recursion on index of intermediate nodes



$$dist(i,j,k) = \min \begin{cases} dist(i,j,k-1) \\ dist(i,k,k-1) + dist(k,j,k-1) \end{cases}$$

Base case: $dist(i,j,0) = \ell(i,j)$ if $(i,j) \in E$, otherwise ∞

Correctness: If $i \to j$ shortest walk goes through k then k occurs only once on the path — otherwise there is a negative length cycle.

All-Pairs: Recursion on index of intermediate nodes

If $dist(k, k, k-1) < 0$ then G has a negative length cycle containing k.

All-Pairs: Recursion on index of intermediate nodes

If $dist(k, k, k-1) < 0$ then G has a negative length cycle containing k.
Now if i can reach k *and* k can reach j then $dist(i, j, k) = -\infty$.

Therefore, recursion below is valid only if $dist(k, k, k-1) \geq 0$.

$$dist(i, j, k) = \min \begin{cases} dist(i, j, k-1) \\ dist(i, k, k-1) + dist(k, j, k-1) \end{cases}$$

All-Pairs: Recursion on index of intermediate nodes

If $dist(k, k, k-1) < 0$ then G has a negative length cycle containing k.
Now if i can reach k *and* k can reach j then $dist(i, j, k) = -\infty$.

Therefore, recursion below is valid only if $dist(k, k, k-1) \geq 0$.

$$dist(i, j, k) = \min \begin{cases} dist(i, j, k-1) \\ dist(i, k, k-1) + dist(k, j, k-1) \end{cases}$$

We can detect this during the algorithm or wait till the end.

Floyd-Warshall Algorithm

for All-Pairs Shortest Paths

for $i = 1$ to n **do**
    for $j = 1$ to n **do**
        $dist(i, j, 0) = \ell(i, j)$ (* $\ell(i, j) = \infty$ if $(i, j) \notin E$, 0 if $i = j$ *)

Floyd-Warshall Algorithm

for All-Pairs Shortest Paths

for $i = 1$ to n **do**
  for $j = 1$ to n **do**
    $dist(i, j, 0) = \ell(i, j)$ (* $\ell(i, j) = \infty$ if $(i, j) \notin E$, 0 if $i = j$ *)

for $k = 1$ to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**

$$dist(i, j, k) = \min \begin{cases} dist(i, j, k-1), \\ dist(i, k, k-1) + dist(k, j, k-1) \end{cases}$$

Floyd-Warshall Algorithm

for All-Pairs Shortest Paths

```
for i = 1 to n do
    for j = 1 to n do
        dist(i, j, 0) = ℓ(i, j)  (* ℓ(i, j) = ∞ if (i, j) ∉ E, 0 if i = j *)

for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            dist(i, j, k) = min { dist(i, j, k − 1),
                                  dist(i, k, k − 1) + dist(k, j, k − 1)
for i = 1 to n do
    if (dist(i, i, n) < 0) then
        Output that there is a negative length cycle in G
```

Floyd-Warshall Algorithm

for All-Pairs Shortest Paths

```
for i = 1 to n do
    for j = 1 to n do
        dist(i,j,0) = ℓ(i,j)  (* ℓ(i,j) = ∞ if (i,j) ∉ E, 0 if i = j *)

for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
```

$$dist(i,j,k) = \min \begin{cases} dist(i,j,k-1), \\ dist(i,k,k-1) + dist(k,j,k-1) \end{cases}$$

```
for i = 1 to n do
    if (dist(i,i,n) < 0) then
        Output that there is a negative length cycle in G
```

Running Time:

Floyd-Warshall Algorithm

for All-Pairs Shortest Paths

```
for i = 1 to n do
    for j = 1 to n do
        dist(i,j,0) = ℓ(i,j)  (* ℓ(i,j) = ∞ if (i,j) ∉ E, 0 if i = j *)

for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
```

$$dist(i,j,k) = \min \begin{cases} dist(i,j,k-1), \\ dist(i,k,k-1) + dist(k,j,k-1) \end{cases}$$

```
for i = 1 to n do
    if (dist(i,i,n) < 0) then
        Output that there is a negative length cycle in G
```

Running Time: $\Theta(n^3)$, Space: $\Theta(n^3)$.

Floyd-Warshall Algorithm
for All-Pairs Shortest Paths

```
for i = 1 to n do
    for j = 1 to n do
        dist(i, j, 0) = ℓ(i, j)  (* ℓ(i, j) = ∞ if (i, j) ∉ E, 0 if i = j *)

for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            dist(i, j, k) = min { dist(i, j, k − 1),
                                  dist(i, k, k − 1) + dist(k, j, k − 1)
for i = 1 to n do
    if (dist(i, i, n) < 0) then
        Output that there is a negative length cycle in G
```

Running Time: $\Theta(n^3)$, Space: $\Theta(n^3)$.

Correctness: via induction and recursive definition

Floyd-Warshall Algorithm: Finding the Paths

Question: Can we find the paths in addition to the distances?

1. Create a $n \times n$ array Next that stores the next vertex on shortest path for each pair of vertices
2. With array Next, for any pair of given vertices i, j can compute a shortest path in $O(n)$ time.

```
for i = 1 to n do
    for j = 1 to n do
        dist(i, j, 0) = ℓ(i, j)
(* ℓ(i, j) = ∞ if (i, j) not edge, 0 if i = j *)
        Next(i, j) = −1
for k = 1 to n do
    for i = 1 to n do
        for j = 1 to n do
            dist(i, j, k) = dist(i, j, k − 1)
            if (dist(i, j, k − 1) > dist(i, k, k − 1) + dist(k, j, k − 1)) then
                dist(i, j, k) = dist(i, k, k − 1) + dist(k, j, k − 1)
                Next(i, j) = k

for i = 1 to n do
    if (dist(i, i, n) < 0) then
        Output that there is a negative length cycle in G
```

Floyd-Warshall Algorithm

Finding the Paths

Exercise: Given $\textit{Next}$ array and any two vertices i, j describe an $O(n)$ algorithm to find a i-j shortest path.

Johnson's Algorithm

- Bellman-Ford gives $O(nm)$ time algorithm to solve single-source shortest paths when G has no negative lengths.
- To compute APSP running Bellman-Ford n times will give a run time of $O(n^2m)$.
- However, if G has no negative length cycle, after computing shortest paths from one vertex using Bellman-Ford, one can use "reduced" costs to convert the graph into one with *non-negative* edge lengths. And then one can run n Dijkstra's on this new graphs to solve APSP. This gives a run time of $O(nm + n^2 \log n)$ for APSP.

See notes for more details.

Summary of results on shortest paths

Single Source Shortest Paths

No negative edges	Dijkstra	$O(n \log n + m)$
Edge lengths can be negative	Bellman Ford	$O(nm)$

All Pairs Shortest Paths

No negative edges	n * Dijkstra	$O(n^2 \log n + nm)$
No negative cycles	n * Bellman Ford	$O(n^2 m) = O(n^4)$
No negative cycles	BF + n * Dijkstra	$O(nm + n^2 \log n)$
No negative cycles	Floyd-Warshall	$O(n^3)$
Unweighted	Matrix multiplication	$O(n^{2.38})$, $O(n^{2.58})$