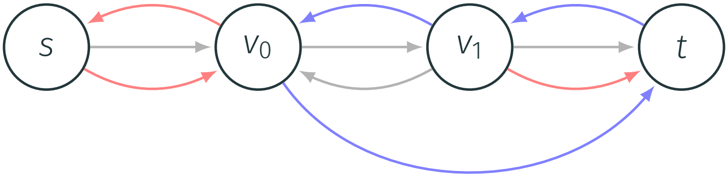
Pre-lecture brain teaser

You have a graph G(V,E). Some of the edges are red, some are white and some are blue. You are given two distinct vertices u and v and want to find a walk $[u \to v]$ such that:

- a white edge must be taken after a red edge only.
- a blue edge must be taken after a white edge only.
- and a red edge may be taken after a blue edge only.
- must start on red edge



ECE-374-B: Lecture 17 - Bellman-Ford and Dynamic Programming on Graphs

Instructor: Nickvash Kani

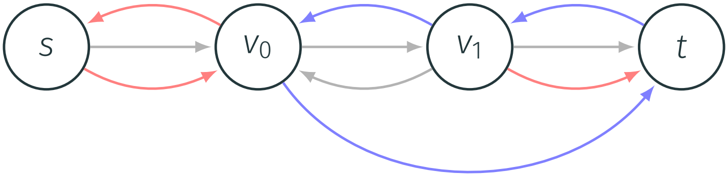
October 28, 2025

University of Illinois Urbana-Champaign

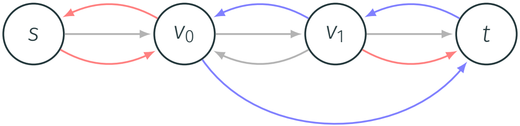
Pre-lecture brain teaser

You have a graph G(V,E). Some of the edges are red, some are white and some are blue. You are given two distinct vertices u and v and want to find a walk $[u \to v]$ such that:

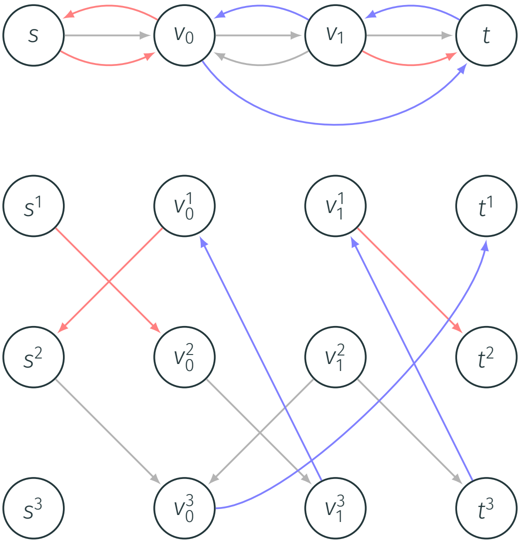
- a white edge must be taken after a red edge only.
- a blue edge must be taken after a white edge only.
- and a red edge may be taken after a blue edge only.
- must start on red edge



Pre-lecture brain teaser



Pre-lecture brain teaser



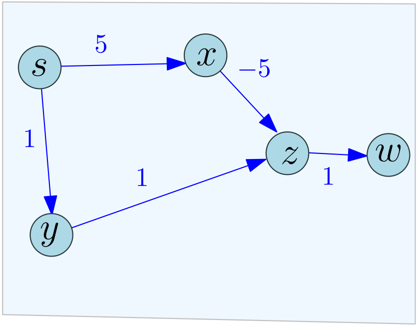
Shortest Paths with Negative Length Edges

Why Dijkstra’s algorithm fails with negative edges

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.

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

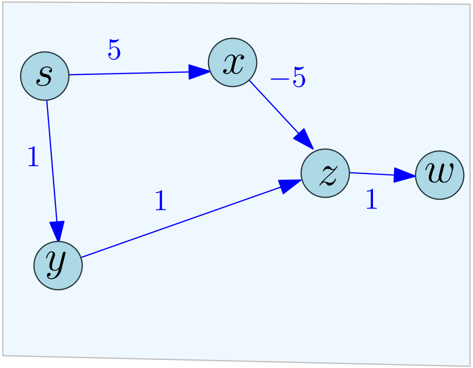
What are the distances computed by Dijkstra's algorithm?



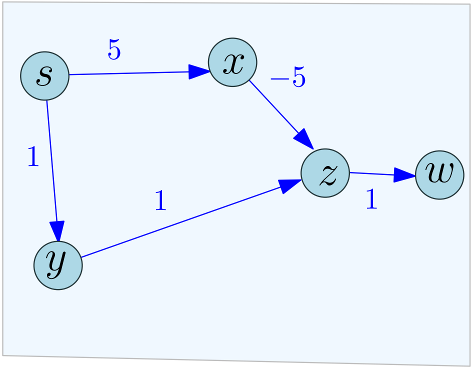
The distance as computed by Dijkstra algorithm starting from s:

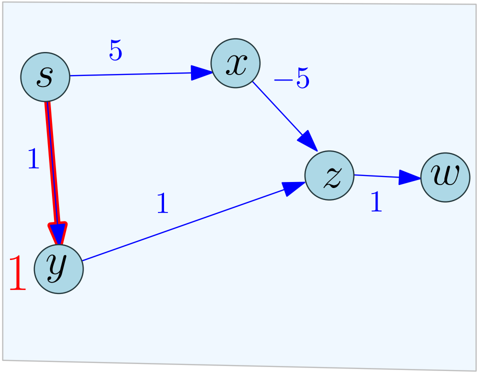
1. $s = 0$, $x = 5$, $y = 1$, $z = 0$.
2. $s = 0$, $x = 1$, $y = 2$, $z = 5$.
3. $s = 0$, $x = 5$, $y = 1$, $z = 2$.
4. IDK.

With negative length edges, Dijkstra's algorithm can fail

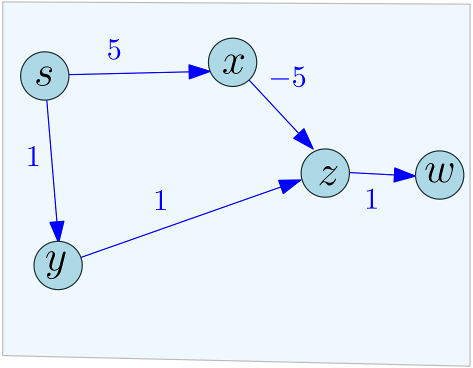


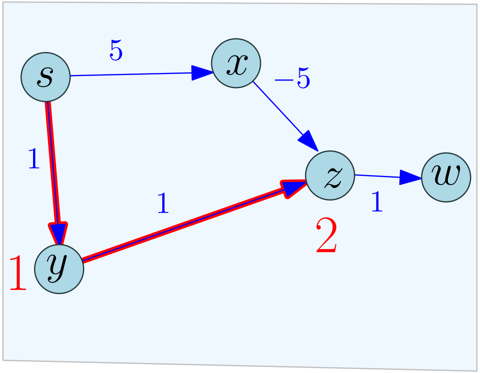
With negative length edges, Dijkstra's algorithm can fail



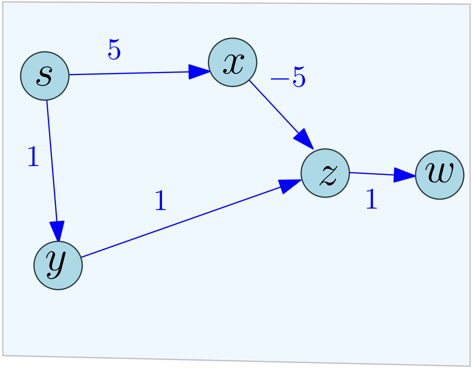


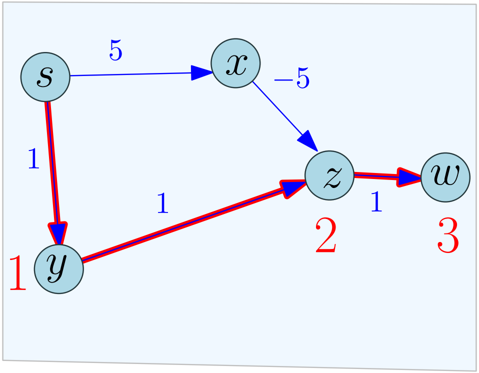
With negative length edges, Dijkstra's algorithm can fail



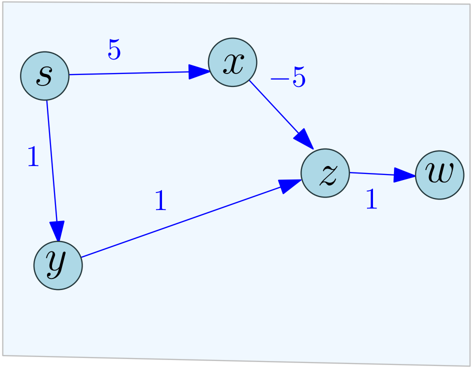


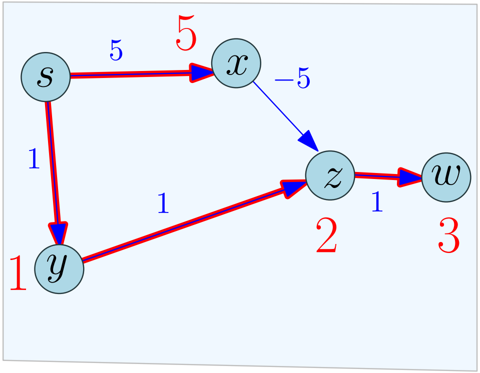
With negative length edges, Dijkstra's algorithm can fail



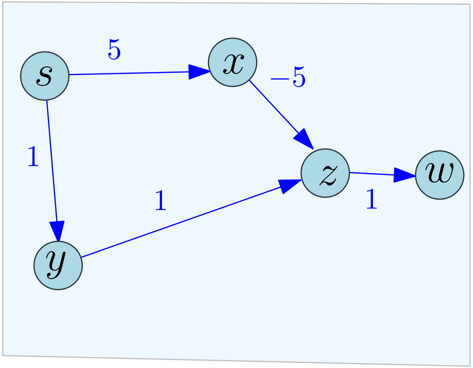


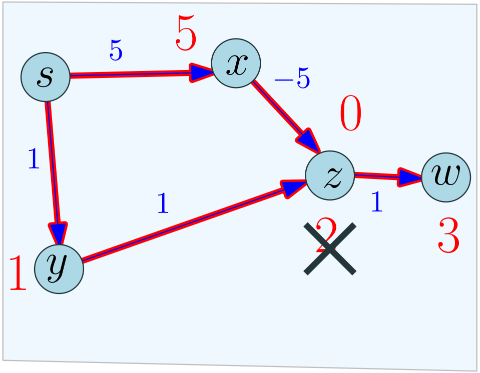
With negative length edges, Dijkstra's algorithm can fail



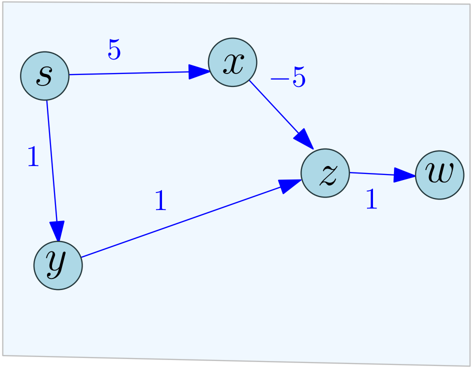


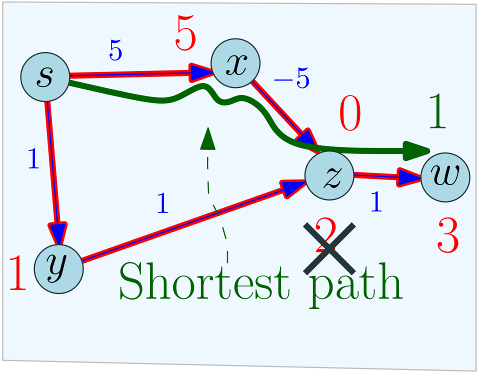
With negative length edges, Dijkstra's algorithm can fail





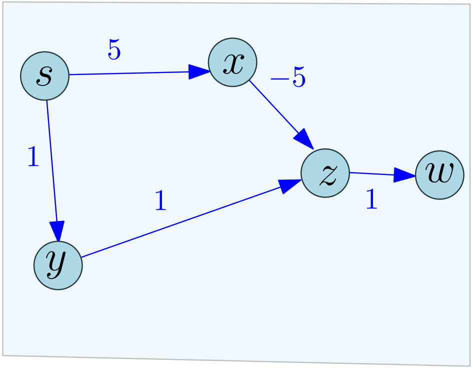
With negative length edges, Dijkstra's algorithm can fail

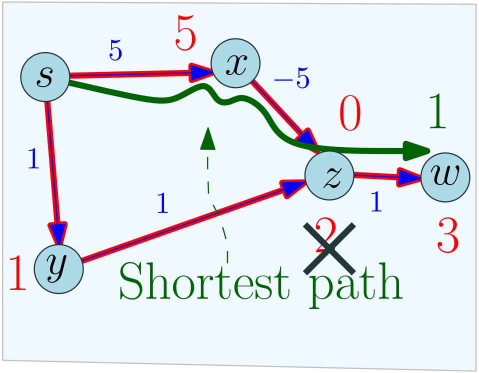




Dijkstra's Algorithm and Negative Lengths

With negative length edges, Dijkstra's algorithm can fail





False assumption: Dijkstra's algorithm assumes that if $s \to v_0 \to v_1 \to v_2 \ldots \to v_k$ 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.

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

- $s = v_0 \to v_1 \to v_2 \to \ldots \to v_i$ *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 $s = v_0 \to v_1 \to v_2 \to \ldots \to v_k$ is a shortest path from s to v_k then for $1 \le i < k$:

- $s = v_0 \to v_1 \to v_2 \to \ldots \to v_i$ *is a shortest path from s to v_i*
- *False:* $dist(s, v_i) \le dist(s, v_k)$ *for $1 \le i < k$. 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 $s = v_0 \to v_1 \to v_2 \to \ldots \to v_k$ is a shortest path from s to v_k then for $1 \leq i < k$:

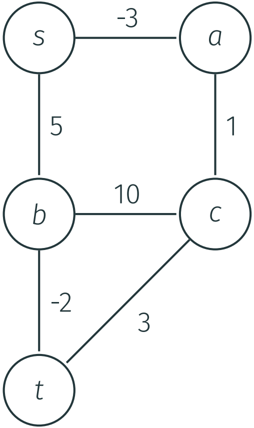
- $s = v_0 \to v_1 \to v_2 \to \ldots \to v_i$ *is a shortest path from s to v_i*
- *False:* $\mathit{dist}(s, v_i) \leq \mathit{dist}(s, v_k)$ *for $1 \leq i < k$.* *Holds true only for non-negative edge lengths.*

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

Why can’t we just re-normalize the edge lengths!?

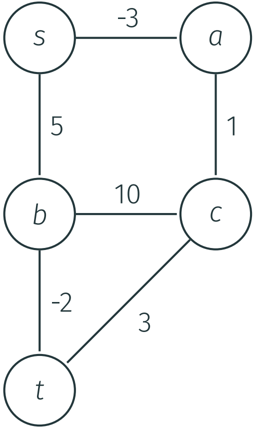
Instinctual thought

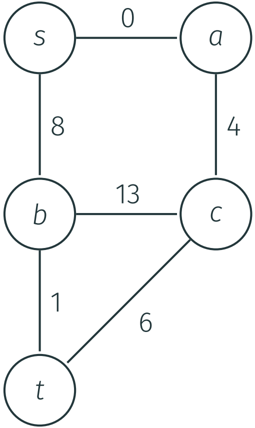
Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive).



Instinctual thought

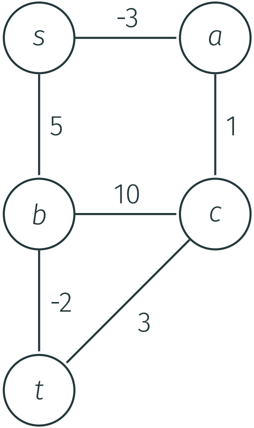
Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive).



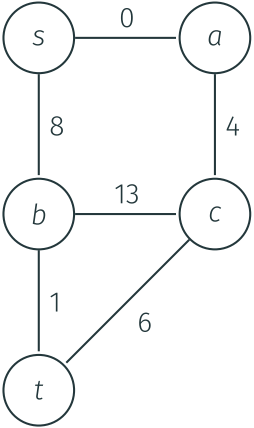


Instinctual thought

Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive).



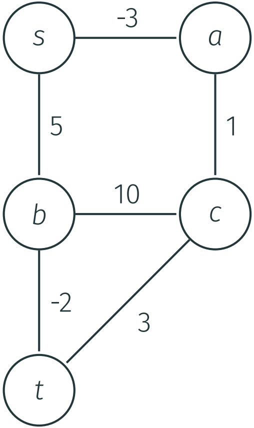
Shortest Path: $s \to a \to c \to t$



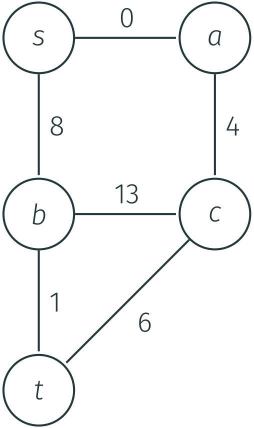
Shortest Path: $s \to b \to t$

Instinctual thought

Why can't we simply add a weight to each edge so that the shortest length is 0 (or positive).



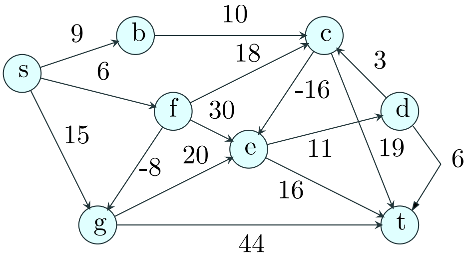
Shortest Path: $s \to a \to c \to t$



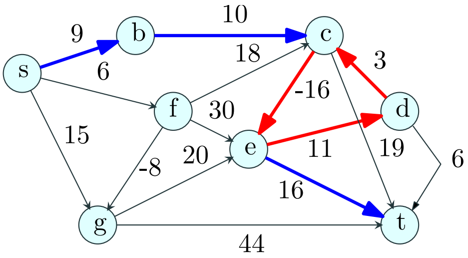
Shortest Path: $s \to b \to t$

But wait! Things get worse: Negative cycles

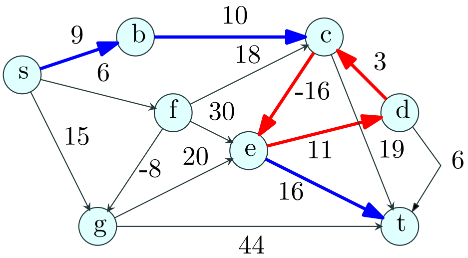
Definition
A cycle C is a negative length cycle if the sum of the edge lengths of C is negative.



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



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



What is the shortest path distance between s and t?

Reminder: Paths have to be simple...

Shortest Paths and Negative Cycles

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

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

Shortest Paths and Negative Cycles

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

- G has a negative length cycle C, and
- 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:

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

Really bad new about negative edges, and shortest path...

Lemma
If there is an efficient algorithm to find a shortest simple $s \to t$ path in a graph with negative edge lengths, then there is an efficient algorithm to find the longest simple $s \to t$ path in a graph with positive edge lengths.

Finding the $s \to t$ longest path is difficult. NP-HARD!

Restating problem of Shortest path with negative edges

Alternatively: Finding Shortest Walks

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

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

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

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

- Given nodes s, t, either find a negative length cycle C that s can reach or find a shortest path from s to t.
- 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.
- Check if G has a negative length cycle or not.

Shortest Paths with Negative Edge Lengths - In Undirected Graphs

Note: With negative lengths, shortest path problems and negative cycle detection in undirected graphs cannot be reduced to directed graphs by bi-directing each undirected edge. Why?

Problem can be solved efficiently in undirected graphs but algorithms are different and significantly more involved than those for directed graphs. One need to compute T-joins in the relevant graph. Pretty painful stuff.

Bellman Ford Algorithm

Shortest path via number of hops

Shortest Paths and Recursion

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

Shortest Paths and Recursion

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

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

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

Shortest Paths and Recursion

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

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

- *$s = v_0 \to v_1 \to v_2 \to \ldots \to v_i$ 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

Assume G has no negative-length cycle (for now).

$d(v, k)$: shortest walk length from s to v using at most k edges.

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.

Assume that all nodes can be reached by s in G

Assume G has no negative-length cycle (for now).

$d(v, k)$: shortest walk length from s to v using at most k edges.

Note: $dist(s, v) = d(v, n - 1)$.

Hop-based Recursion: Bellman-Ford Algorithm

Single-source problem: fix source s.

Assume that all nodes can be reached by s in G

Assume G has no negative-length cycle (for now).

$d(v, k)$: shortest walk length from s to v using at most k edges.

Note: $dist(s, v) = d(v, n-1)$. 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

Assume G has no negative-length cycle (for now).

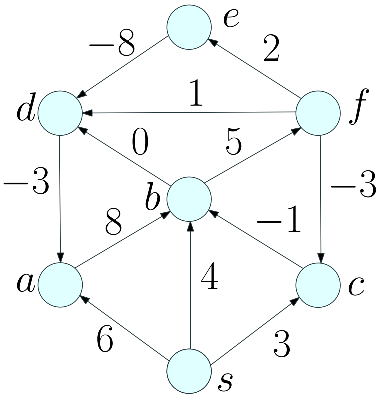
$d(v, k)$: shortest walk length from s to v using at most k edges.

Note: $dist(s, v) = d(v, n-1)$. Recursion for $d(v, k)$:

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

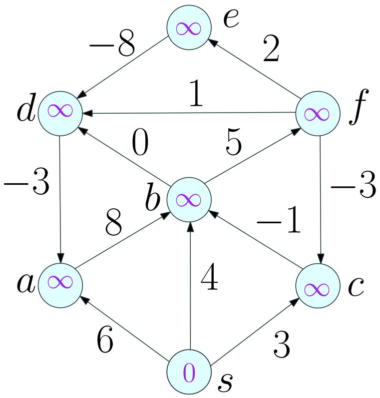
Base case: $d(s, 0) = 0$ and $d(v, 0) = \infty$ for all $v \neq s$.

Example



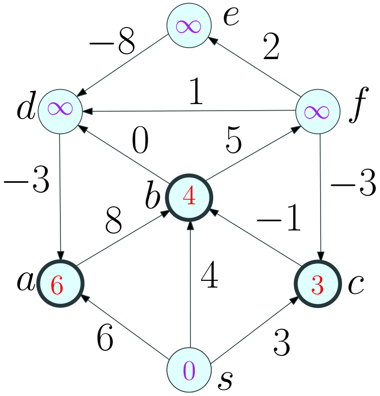
round	s	a	b	c	d	e	f

Example



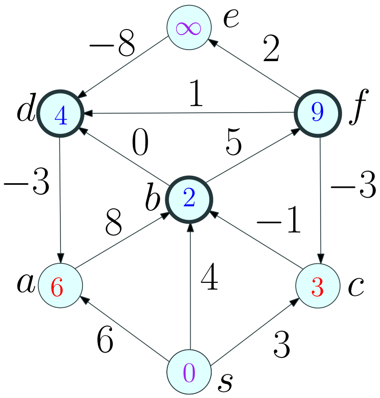
round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞

Example



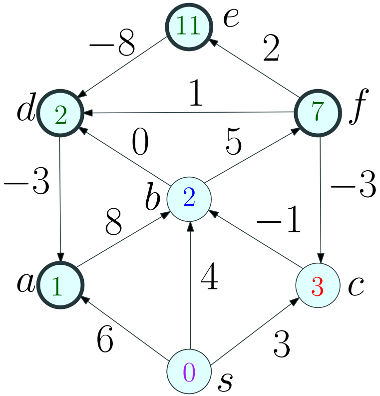
round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞

Example



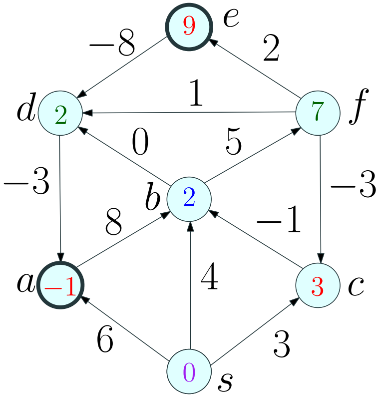
round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9

Example



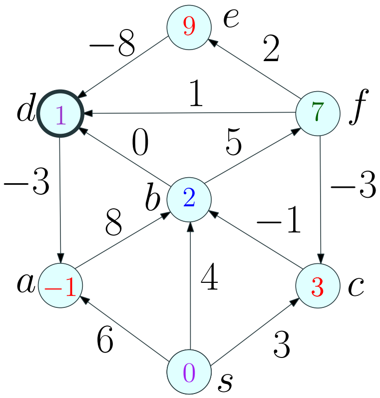
round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7

Example



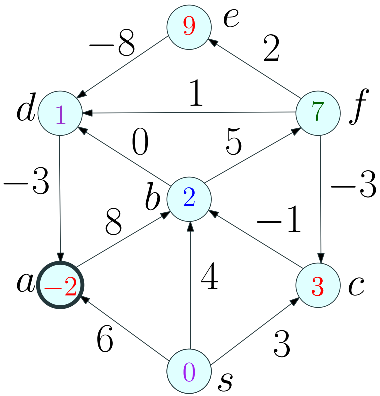
round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7

Example



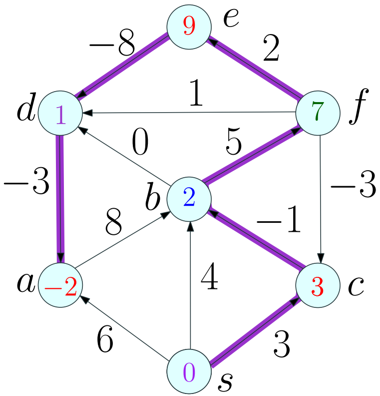
round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5	0	-1	2	3	1	9	7

Example



round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5	0	-1	2	3	1	9	7
6	0	-2	2	3	1	9	7

Example



round	s	a	b	c	d	e	f
0	0	∞	∞	∞	∞	∞	∞
1	0	6	4	3	∞	∞	∞
2	0	6	2	3	4	∞	9
3	0	1	2	3	2	11	7
4	0	-1	2	3	2	9	7
5	0	-1	2	3	1	9	7
6	0	-2	2	3	1	9	7

The Bellman-Ford Algorithm

Bellman-Ford Algorithm

```
Create in(G) list from adj(G)

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

for k = 1 to n − 1 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)
```

Bellman-Ford Algorithm

```
Create in(G) list from adj(G)

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

for k = 1 to n − 1 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)
```

Running time:

Bellman-Ford Algorithm

```
Create in(G) list from adj(G)

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

for k = 1 to n − 1 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)
```

Running time: $O(n(n+m))$

Bellman-Ford Algorithm

```
Create in(G) list from adj(G)

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

for k = 1 to n − 1 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)
```

Running time: $O(n(n+m))$ Space:

Bellman-Ford Algorithm

```
Create in(G) list from adj(G)

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

for k = 1 to n − 1 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)
```

Running time: $O(n(n+m))$ Space: $O(m+n^2)$ (Space can be reduced to $O(m+n)$).

Bellman-Ford Algorithm: Cleaner version

```
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)}

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

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

Bellman-Ford Algorithm: Cleaner version

```
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)}

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

Running time: $O(mn)$ Space: $O(m+n)$ Do we need the in(V) list?

Bellman-Ford Algorithm: Cleaner version

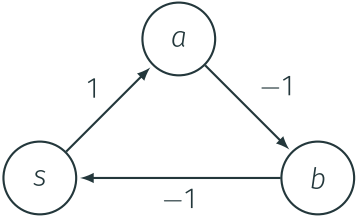
```
for each u ∈ V do
    d(u) ← ∞
d(s) ← 0

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

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

Running time: $O(mn)$ Space: $O(n)$

Bellman-Ford: Detecting negative cycles

Negative cycles

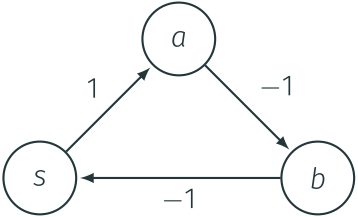
What happens if we run this on a graph with negative cycles?



round	s	a	b

Negative cycles

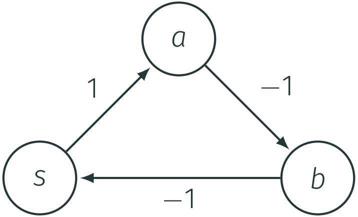
What happens if we run this on a graph with negative cycles?



round	s	a	b
0	0	∞	∞

Negative cycles

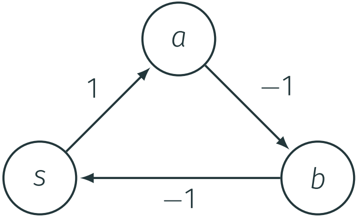
What happens if we run this on a graph with negative cycles?



round	s	a	b
0	0	∞	∞
1	0	1	∞

Negative cycles

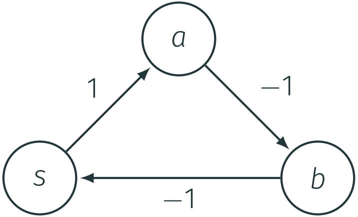
What happens if we run this on a graph with negative cycles?



round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0

Negative cycles

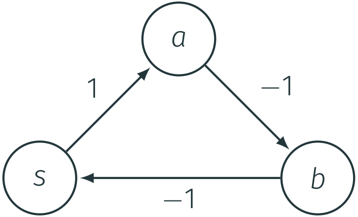
What happens if we run this on a graph with negative cycles?



round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0
3	-1	1	0

Negative cycles

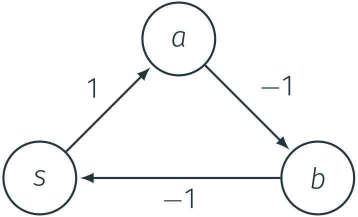
What happens if we run this on a graph with negative cycles?



round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0
3	-1	1	0
4	-1	0	0

Negative cycles

What happens if we run this on a graph with negative cycles?



round	s	a	b
0	0	∞	∞
1	0	1	∞
2	0	1	0
3	-1	1	0
4	-1	0	0
5	-1	0	-1

Negative cycles can not hide

Lemma restated
If G does not has a negative length cycle reachable from $s \implies \forall v$: $d(v, n) = d(v, n-1)$.

Also, $d(v, n-1)$ is the length of the shortest path between s and v.

Put together are the following:

Lemma
G has a negative length cycle reachable from $s \iff$ there is some node v such that $d(v, n) < d(v, n-1)$.

Bellman-Ford: Negative Cycle Detection - final version

```
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)
```

Variants on Bellman-Ford

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

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

Negative Cycle Detection

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

- 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.
- Run Bellman-Ford $|V|$ times, once from each node u?

Negative Cycle Detection

- 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.
- Negative cycle detection can be done with one Bellman-Ford invocation.

Shortest Paths in DAGs

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.

- Given nodes s, t find shortest path from s to t.
- 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.

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

Simplification of algorithms for DAGs

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

Algorithm for DAGs

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

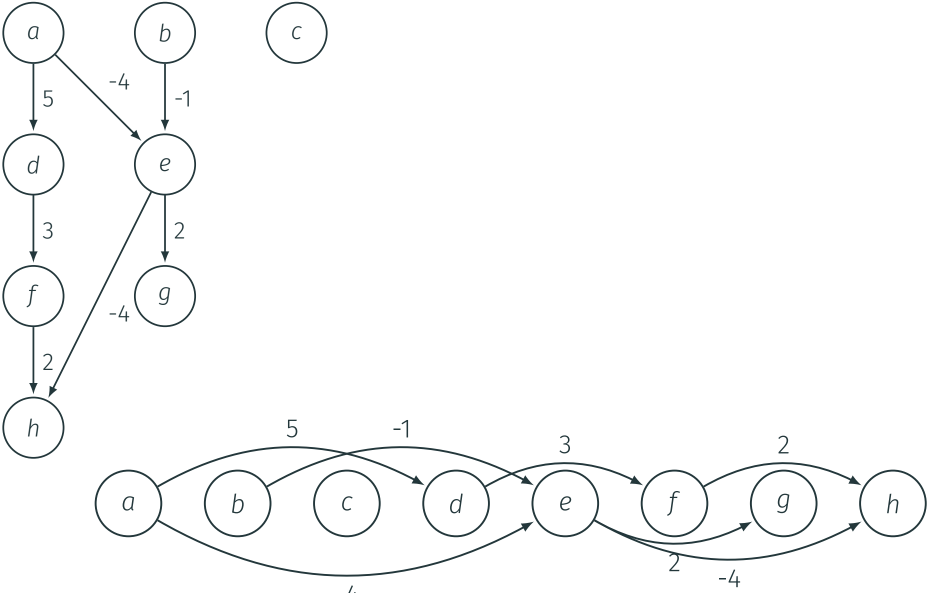
Algorithm for DAGs

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

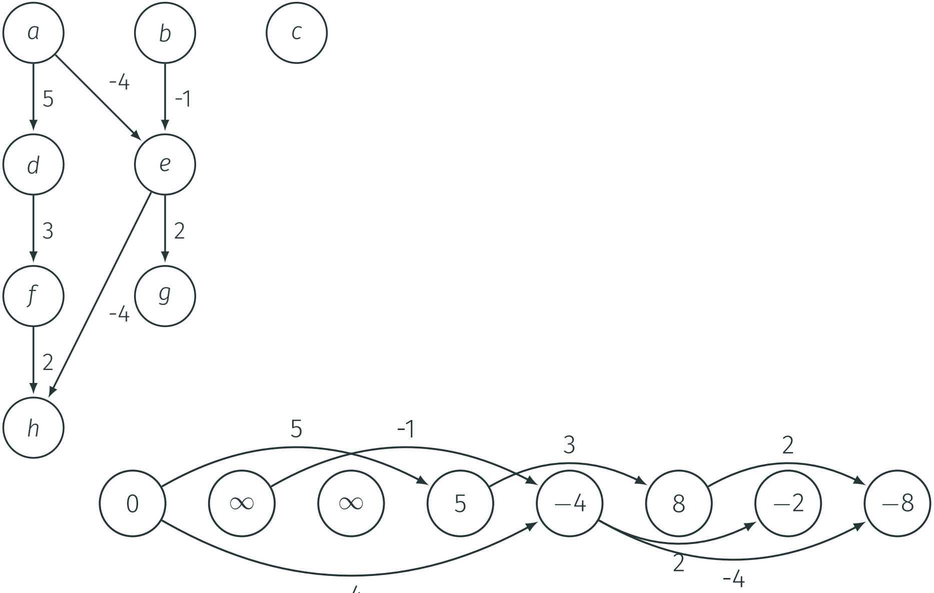
Observation:

- shortest path from s to v_i cannot use any node from $v_{i+1}, \ldots, v_n$
- can find shortest paths in topological sort order.

Shortest Paths for DAGs - Example



Shortest Paths for DAGs - Example



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: induction on i and observation in previous slide.

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

All Pairs Shortest Paths

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.

- Given nodes s, t find shortest path from s to t.
- Given node s find shortest path from s to all other nodes.
- Find shortest paths for all pairs of nodes.

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.

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

SSSP: 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.

- Given nodes s, t find shortest path from s to t.
- 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 - Using known algorithms...

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.

- Find shortest paths for all pairs of nodes.

All-Pairs Shortest Paths - Using known algorithms...

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.

- Find shortest paths for all pairs of nodes.

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

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

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.

- Find shortest paths for all pairs of nodes.

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

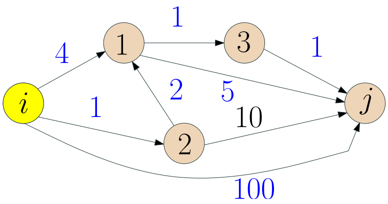
- Non-negative lengths. $O(nm \log n)$ with heaps and $O(nm + n^2 \log n)$ using advanced priority queues.
- Arbitrary edge lengths: $O(n^2 m)$.
  $\Theta(n^4)$ if $m = \Omega(n^2)$.

Can we do better?

All Pairs Shortest Paths: A recursive solution

All-Pairs: Recursion on index of intermediate nodes

- Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
- $dist(i, j, k)$: 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).



$dist(i, j, 0) =$

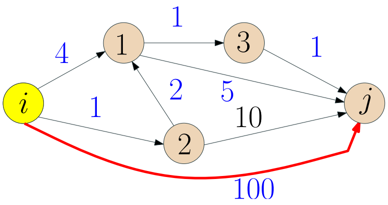
$dist(i, j, 1) =$

$dist(i, j, 2) =$

$dist(i, j, 3) =$

All-Pairs: Recursion on index of intermediate nodes

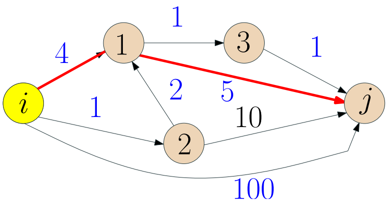
- Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
- $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



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

All-Pairs: Recursion on index of intermediate nodes

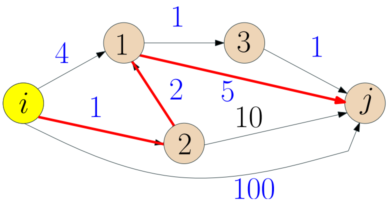
- Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
- $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



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

All-Pairs: Recursion on index of intermediate nodes

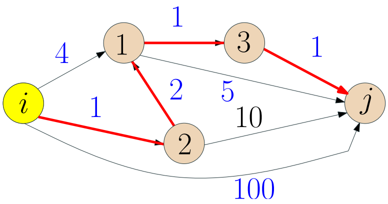
- Number vertices arbitrarily as $v_1, v_2, \ldots, v_n$
- $dist(i, j, k)$: length of shortest walk from v_i to v_j among all walks in which the largest index of an intermediate node is at most k (could be $-\infty$ if there is a negative length cycle).



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

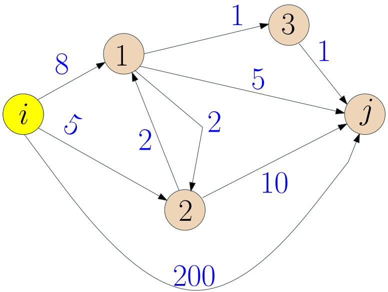
All-Pairs: Recursion on index of intermediate nodes

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



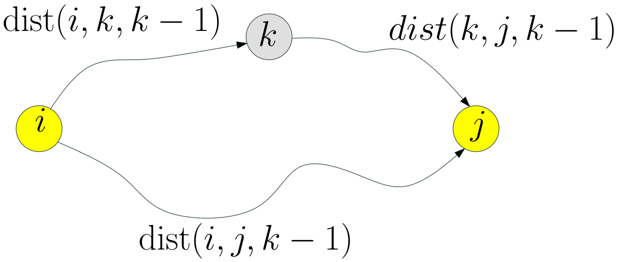
$$dist(i, j, 0) = 100$$
$$dist(i, j, 1) = 9$$
$$dist(i, j, 2) = 8$$
$$dist(i, j, 3) = 5$$

For the following graph, $\text{dist}(i, j, 2)$ is...



1. 9
2. 10
3. 11
4. 12
5. 15

All-Pairs: Recursion on index of intermediate nodes



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

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

All-Pairs: Recursion on index of intermediate nodes

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

Recursion below is valid only if $dist(k, k, k-1) \geq 0$. We can detect this during the algorithm or wait till the end.

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

Floyd-Warshall algorithm

Floyd-Warshall Algorithm - for All-Pairs Shortest Paths

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

```
for i = 1 to n do
    for j = 1 to n do
        d(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
            d(i,j,k) = min { d(i,j,k−1),
                           { d(i,k,k−1) + d(k,j,k−1)
for i = 1 to n do
    if (dist(i,i,n) < 0) then
        Output ∃ negative cycle in G
```

Floyd-Warshall Algorithm - for All-Pairs Shortest Paths

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

for $i = 1$ to n do
    for $j = 1$ to n do
        $d(i,j,0) = \ell(i,j)$
  (* $\ell(i,j) = \infty$ 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
            $d(i,j,k) = \min \begin{cases} d(i,j,k-1), \\ d(i,k,k-1) + d(k,j,k-1) \end{cases}$
for $i = 1$ to n do
    if $(dist(i,i,n) < 0)$ then
        Output $\exists$ negative cycle in G

Running Time:

Floyd-Warshall Algorithm - for All-Pairs Shortest Paths

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

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

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

for $i = 1$ `to` n **do**
    if $(dist(i,i,n) < 0)$ **then**
        `Output` $\exists$ `negative cycle in` G

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

Floyd-Warshall Algorithm: Finding the Paths

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

Floyd-Warshall Algorithm: Finding the Paths

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

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

Floyd-Warshall Algorithm - Finding the Paths

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

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

Summary of shortest path algorithms

Summary of results on shortest paths

Single source		
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	$O\left(n^2 \log n + nm\right)$
No negative cycles	n * Bellman Ford	$O\left(n^2 m\right) = O\left(n^4\right)$
No negative cycles	Johnson's [1]	$O\left(nm + n^2 \log n\right)$
No negative cycles	Floyd-Warshall	$O\left(n^3\right)$
Unweighted	Matrix multiplication [2]	$O(n^{2.38})$, $O(n^{2.58})$

Summary of results on shortest paths

(1): The algorithm for the case that there are no negative cycles, and doing all shortest paths, works by computing a potential function using Bellman-Ford and then doing Dijkstra. It is mentioned for the sake of completeness, but it outside the scope of the class.

(2): `https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss12/AdvancedGraphAlgorithms/Slides14.pdf`

Fin