CS 473: Algorithms

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

University of Illinois, Urbana-Champaign

Fall 2010
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.

- Given nodes $s, t$ find shortest path from $s$ to $t$.
- Given node $s$ find shortest path from $s$ to all other nodes.
Single-Source Shortest Paths with Negative Edge Lengths

**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.
A cycle $C$ is a negative length cycle if the sum of the edge lengths of $C$ is negative.
A cycle $C$ is a negative length cycle if the sum of the edge lengths of $C$ is negative.
A cycle $C$ is a negative length cycle if the sum of the edge lengths of $C$ is negative.
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 $t$?
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 $t$?

- Define shortest distance to be undefined, that is $-\infty$, OR
- Define shortest distance to be 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
- $G$ 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 $t$?

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

**Lemma**

*If there is an efficient algorithm to find a shortest simple $s \rightarrow t$ path in a graph with negative edge lengths, then there is an efficient algorithm to find the longest simple $s \rightarrow t$ path in a graph with positive edge lengths.*
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 $t$?
- Define shortest distance to be undefined, that is $-\infty$, OR
- Define shortest distance to be the length of a shortest *simple* path from $s$ to $t$.

**Lemma**

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

Finding the $s \rightarrow t$ longest path is difficult. NP-Hard!
# Shortest Paths with Negative Edge Lengths: Problems

## Algorithmic 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 $s, t$, 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.
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 more involved than those for directed graphs. Beyond the scope of this class. If interested, ask instructor for references.
Why Negative Lengths?

Several Applications

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

Currency Trading

**Input**  
n currencies and for each ordered pair \((a, b)\) the *exchange rate* for converting one unit of \(a\) into one unit of \(b\).

- Is there an arbitrage opportunity?
- Given currencies \(s, t\) what is the best way to convert \(s\) to \(t\) (perhaps via other intermediate currencies)?
Observation: If we convert currency $i$ to $j$ via intermediate currencies $k_1, k_2, \ldots, k_h$ then one unit of $i$ yields $\text{exch}(i, k_1) \times \text{exch}(k_1, k_2) \ldots \times \text{exch}(k_h, j)$ units of $j$. 
Reducing Currency Trading to Shortest Paths

**Observation:** If we convert currency $i$ to $j$ via intermediate currencies $k_1, k_2, \ldots, k_h$ then one unit of $i$ yields
\[ \text{exch}(i, k_1) \times \text{exch}(k_1, k_2) \times \cdots \times \text{exch}(k_h, j) \] units of $j$.

Create currency trading graph $G = (V, E)$:
- For each currency $i$ there is a node $v_i \in V$
- $E = V \times V$: an edge for each pair of currencies
- edge length $\ell(v_i, v_j) =$
Observation: If we convert currency $i$ to $j$ via intermediate currencies $k_1, k_2, \ldots, k_h$ then one unit of $i$ yields 
$\text{exch}(i, k_1) \times \text{exch}(k_1, k_2) \ldots \times \text{exch}(k_h, j)$ units of $j$.

Create currency trading graph $G = (V, E)$:
- For each currency $i$ there is a node $v_i \in V$
- $E = V \times V$: an edge for each pair of currencies
- edge length $\ell(v_i, v_j) = \log(\text{exch}(i, j))$ can be negative

Exercise: Verify that there is an arbitrage opportunity if and only if $G$ has a negative length cycle.

The best way to convert currency $i$ to currency $j$ is via a shortest path in $G$ from $i$ to $j$. If $d$ is the distance from $i$ to $j$ then one unit of $i$ can be converted into $2^d$ units of $j$. 
Reducing Currency Trading to Shortest Paths

**Observation:** If we convert currency \( i \) to \( j \) via intermediate currencies \( k_1, k_2, \ldots, k_h \) then one unit of \( i \) yields \( \text{exch}(i, k_1) \times \text{exch}(k_1, k_2) \times \cdots \times \text{exch}(k_h, j) \) units of \( j \).

Create currency trading graph \( G = (V, E) \):

- For each currency \( i \) there is a node \( v_i \in V \)
- \( E = V \times V \): an edge for each pair of currencies
- edge length \( \ell(v_i, v_j) = \log(\text{exch}(i, j)) \) can be negative

**Exercise:** Verify that

- There is an arbitrage opportunity if and only if \( G \) has a negative length cycle.
- The best way to convert currency \( i \) to currency \( j \) is via a shortest path in \( G \) from \( i \) to \( j \). If \( d \) is the distance from \( i \) to \( j \) then one unit of \( i \) can be converted into \( 2^d \) units of \( j \).
### Algorithmic 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 $s, t$, 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.
Observation

With negative cost edges, Dijkstra’s algorithm fails

---

Dijkstra’s Algorithm and Negative Lengths
Dijkstra’s Algorithm and Negative Lengths

**Observation**

With negative cost edges, Dijkstra’s algorithm fails.

![Graph with negative edges](image)

- False assumption: Dijkstra’s algorithm is based on the assumption that 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
  
  \[ \text{dist}(s, v_i) \leq \text{dist}(s, v_i+1) \]
  
  for \( 0 \leq i < k \). Holds true only for non-negative edge lengths.
Observation

With negative cost edges, Dijkstra’s algorithm fails

False assumption: Dijkstra’s algorithm is based on the assumption that if \( s = v_0 \rightarrow v_1 \rightarrow v_2 \ldots \rightarrow v_k \) 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 $v_i$
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 $1 \leq i < k$:

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

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

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

Cannot explore nodes in increasing order of distance! We need a more basic strategy.
A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to $\infty$
- Maintain the invariant that there is an $s \rightarrow u$ path of length $d(s, u)$. Hence $d(s, u) \geq dist(s, u)$.
- Iteratively refine $d(s, \cdot)$ values until they reach the correct value $dist(s, \cdot)$ values at termination

**Question**: How do we make progress?
A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to $\infty$
- Maintain the invariant that there is an $s \rightarrow u$ path of length $d(s, u)$. Hence $d(s, u) \geq \text{dist}(s, u)$.
- Iteratively refine $d(s, \cdot)$ values until they reach the correct value $\text{dist}(s, \cdot)$ values at termination

**Question:** How do we make progress?

**Definition**

Given distance estimates $d(s, u)$ for each $u \in V$, an edge $e = (u, v)$ is tense if $d(s, v) > d(s, u) + \ell(u, v)$. 
A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to $\infty$
- Maintain the invariant that there is an $s \rightarrow u$ path of length $d(s, u)$. Hence $d(s, u) \geq dist(s, u)$.
- Iteratively refine $d(s, \cdot)$ values until they reach the correct value $dist(s, \cdot)$ values at termination

**Question:** How do we make progress?

**Definition**

Given distance estimates $d(s, u)$ for each $u \in V$, an edge $e = (u, v)$ is **tense** if $d(s, v) > d(s, u) + \ell(u, v)$.

**Relax(e=(u,v))**

if $(d(s,v) > d(s,u) + \ell(u,v))$ then

$d(s,v) = d(s,u) + \ell(u,v)$
A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to $\infty$
- Maintain the invariant that there is an $s \rightarrow u$ path of length $d(s, u)$. Hence $d(s, u) \geq dist(s, u)$.
- Iteratively refine $d(s, \cdot)$ values until they reach the correct value $dist(s, \cdot)$ values at termination

**Question:** How do we make progress?

**Definition**

Given distance estimates $d(s, u)$ for each $u \in V$, an edge $e = (u, v)$ is tense if $d(s, v) > d(s, u) + \ell(u, v)$.

Relax(e=(u,v))

if $(d(s,v) > d(s,u) + \ell(u,v))$ then
$\quad d(s,v) = d(s,u) + \ell(u,v)$

**Proposition**

*Relax() maintains the invariant on $d(s, u)$ values.*
A Generic Shortest Path Algorithm

d(s,s) = 0
for each node u ≠ s do
    d(s,u) = ∞

While there is a tense edge do
    Pick a tense edge e
    Relax(e)

Output d(s,u) values

Technical assumption: If e = (u, v) is an edge and d(s, u) = d(s, v) = ∞ then edge is not tense.
Properties of the generic algorithm

**Proposition**

*If* \( u \) *is not reachable from* \( s \) *then* \( d(s, u) \) *remains at* \( \infty \) *throughout the algorithm.*

**Proposition**

*If a negative length cycle* \( C \) *is reachable by* \( s \) *then there is always a tense edge and hence the algorithm never terminates.*

**Corollary**

*If the algorithm terminates then there is no negative length cycle* \( C \) *that is reachable from* \( s \).

**Lemma**

*If the algorithm terminates then* \( d(s, u) = \text{dist}(s, u) \) *for each node* \( u \) *and* \( s \) *cannot reach a negative cycle.*

Proof of last lemma; see future slides.
for each node $u \neq s$
  $d(s,u) = \infty$
$d(s,s) = 0$
$S = \emptyset$
While ($S \neq V$) do
  Let $v$ be node in $V - S$ with min $d$ value
  $S = S \cup \{v\}$
  For each edge $e$ in $\text{Adj}(v)$ do
    Relax($e$)
Generic Algorithm: Ordering Relax operations

\[
\begin{align*}
d(s,s) &= 0 \\
\text{for each node } u \neq s \text{ do} \\
\quad d(s,u) &= \infty
\end{align*}
\]

While there is a tense edge do
\[
\begin{align*}
\text{Pick a tense edge } e \\
\text{Relax}(e)
\end{align*}
\]

Output \(d(s,u)\) values

**Question:** How do we pick edges to relax?
Generic Algorithm: Ordering Relax operations

d(s, s) = 0
for each node u \neq s do
    d(s, u) = \infty

While there is a tense edge do
    Pick a tense edge e
    Relax(e)

Output d(s, u) values

**Question:** How do we pick edges to relax?

**Observation:** Suppose s \rightarrow v_1 \rightarrow \ldots \rightarrow v_k is a shortest path.

If Relax(s, v_1), Relax(v_1, v_2), \ldots, Relax(v_{k-1}, v_k) are done in order
then \( d(s, v_k) = dist(s, v_k) \)!
Observation: Suppose $s \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$ is a shortest path.

If $\text{Relax}(s, v_1)$, $\text{Relax}(v_1, v_2)$, $\ldots$, $\text{Relax}(v_{k-1}, v_k)$ are done in order then $d(s, v_k) = \text{dist}(s, v_k)$! (Why?)
Ordering Relax operations

**Observation:** Suppose $s \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$ is a shortest path.

If $\text{Relax}(s, v_1), \text{Relax}(v_1, v_2), \ldots, \text{Relax}(v_{k-1}, v_k)$ are done in order then $d(s, v_k) = \text{dist}(s, v_k)!$ (Why?)

We don’t know the shortest paths so how do we know the order to do the Relax operations?
Observation: Suppose $s \rightarrow v_1 \rightarrow \ldots \rightarrow v_k$ is a shortest path.

If Relax($s, v_1$), Relax($v_1, v_2$), $\ldots$, Relax($v_{k-1}, v_k$) are done in order then $d(s, v_k) = \text{dist}(s, v_k)$! (Why?)

We don’t know the shortest paths so how do we know the order to do the Relax operations?

We don’t!

- Relax all edges (even those not tense) in some arbitrary order
- Iterate $|V| - 1$ times
- First iteration will do Relax($s, v_1$) (and other edges), second round Relax($v_1, v_2$) and in iteration $k$ we do Relax($v_{k-1}, v_k$).
Bellman-Ford Algorithm

for each $u \in V$
\[ d(s,u) = \infty \]
\[ d(s,s) = 0 \]

for $i = 1$ to $|V| - 1$ do
  for each edge $e = (u, v)$ do
    Relax(e)

for each $u \in V$
\[ \text{dist}(s,u) = d(s,u) \]
Bellman-Ford Algorithm: Scanning Edges

One possible way to scan edges in each iteration.

Q is an empty queue
for each $u \in V$
    $d(s,u) = \infty$
    enqueue(Q, u)
d(s,s) = 0

for $i = 1$ to $|V| - 1$ do
    for $i = 1$ to $|V|$ do
        u = dequeue(Q)
        for each edge $e$ in Adj(u) do
            Relax(e)
        enqueue(Q,u)

for each $u \in V$
    dist(s,u) = d(s,u)
Figure: One iteration of Bellman-Ford that Relaxes all edges by processing nodes in the order $s, a, b, c, d, e, f$. Red edges indicate the prev pointers (in reverse)
Figure: 6 iterations of Bellman-Ford starting with the first one from previous slide. No changes in 5th iteration and 6th iteration.
Correctness of Bellman-Ford Algorithm

Lemma

Let $G$ be a directed graph with arbitrary edge lengths. Let $v$ be a node in $V$ such that there is a shortest path in $G$ from $s$ to $v$ with $i$ hops/edges. Then, after $i$ iterations of the for loop in the Bellman-Ford algorithm, $d(s, v) = \text{dist}(s, v)$

Proof.

By induction on $i$.

- Base case: $i = 0$. $d(s, s) = 0$ and $d(s, s) = \text{dist}(s, s)$.
Lemma

Let $G$ be a directed graph with arbitrary edge lengths. Let $v$ be a node in $V$ such that there is a shortest path in $G$ from $s$ to $v$ with $i$ hops/edges. Then, after $i$ iterations of the for loop in the Bellman-Ford algorithm, $d(s, v) = \text{dist}(s, v)$.

Proof.

By induction on $i$.

- Base case: $i = 0$. $d(s, s) = 0$ and $d(s, s) = \text{dist}(s, s)$.
- Induction Step: Let $s \rightarrow v_1 \ldots \rightarrow v_{i-1} \rightarrow v$ be a shortest path from $s$ to $v$ of $i$ hops.
Lemma

Let $G$ be a directed graph with arbitrary edge lengths. Let $v$ be a node in $V$ such that there is a shortest path in $G$ from $s$ to $v$ with $i$ hops/edges. Then, after $i$ iterations of the for loop in the Bellman-Ford algorithm, $d(s, v) = \text{dist}(s, v)$

Proof.

By induction on $i$.

- Base case: $i = 0$. $d(s, s) = 0$ and $d(s, s) = \text{dist}(s, s)$.
- Induction Step: Let $s \rightarrow v_1 \ldots \rightarrow v_{i-1} \rightarrow v$ be a shortest path from $s$ to $v$ of $i$ hops.
  - $v_{i-1}$ has a shortest path from $s$ of $i - 1$ hops or less. (Why?). By induction, $d(s, v_{i-1}) = \text{dist}(s, v_{i-1})$ after $i - 1$ iterations.
**Correctness of Bellman-Ford Algorithm**

**Lemma**

Let $G$ be a directed graph with arbitrary edge lengths. Let $v$ be a node in $V$ such that there is a shortest path in $G$ from $s$ to $v$ with $i$ hops/edges. Then, after $i$ iterations of the for loop in the Bellman-Ford algorithm, $d(s, v) = dist(s, v)$

**Proof.**

By induction on $i$.

- **Base case:** $i = 0$. $d(s, s) = 0$ and $d(s, s) = dist(s, s)$.
- **Induction Step:** Let $s \rightarrow v_1 \ldots \rightarrow v_{i-1} \rightarrow v$ be a shortest path from $s$ to $v$ of $i$ hops.
  - $v_{i-1}$ has a shortest path from $s$ of $i - 1$ hops or less. (Why?). By induction, $d(s, v_{i-1}) = dist(s, v_{i-1})$ after $i - 1$ iterations.
  - In iteration $i$, Relax($v_{i-1}, v_i$) sets $d(s, v_i) = dist(s, v_i)$. 

Note: Relax does not change $d(s, u)$ once $d(s, u) = dist(s, u)$. 
Correctness of Bellman-Ford Algorithm

Lemma

Let $G$ be a directed graph with arbitrary edge lengths. Let $v$ be a node in $V$ such that there is a shortest path in $G$ from $s$ to $v$ with $i$ hops/edges. Then, after $i$ iterations of the for loop in the Bellman-Ford algorithm, $d(s, v) = \text{dist}(s, v)$

Proof.

By induction on $i$.

- **Base case**: $i = 0$. $d(s, s) = 0$ and $d(s, s) = \text{dist}(s, s)$.
- **Induction Step**: Let $s \rightarrow v_1 \ldots \rightarrow v_{i-1} \rightarrow v$ be a shortest path from $s$ to $v$ of $i$ hops.
  - $v_{i-1}$ has a shortest path from $s$ of $i - 1$ hops or less. (Why?). By induction, $d(s, v_{i-1}) = \text{dist}(s, v_{i-1})$ after $i - 1$ iterations.
  - In iteration $i$, $\text{Relax}(v_{i-1}, v_i)$ sets $d(s, v_i) = \text{dist}(s, v_i)$.
  - Note: $\text{Relax}$ does not change $d(s, u)$ once $d(s, u) = \text{dist}(s, u)$.
Correctness of Bellman-Ford Algorithm

**Corollary**

After $|V| - 1$ iterations of Bellman-Ford, $d(s, u) = \text{dist}(s, u)$ for any node $u$ that has a shortest path from $s$. 

Note: If there is a negative cycle $C$ such that $s$ can reach $C$ then we do not know whether $d(s, u) = \text{dist}(s, u)$ or not even if $\text{dist}(s, u)$ is well-defined.

Question: How do we know whether there is a negative cycle reachable from $s$?
Correctness of Bellman-Ford Algorithm

**Corollary**

After $|V| - 1$ iterations of Bellman-Ford, $d(s, u) = \text{dist}(s, u)$ for any node $u$ that has a shortest path from $s$.

**Note:** If there is a negative cycle $C$ such that $s$ can reach $C$ then we do not know whether $d(s, u) = \text{dist}(s, u)$ or not even if $\text{dist}(s, u)$ is well-defined.
Correctness of Bellman-Ford Algorithm

Corollary

After $|V| - 1$ iterations of Bellman-Ford, $d(s, u) = dist(s, u)$ for any node $u$ that has a shortest path from $s$.

Note: If there is a negative cycle $C$ such that $s$ can reach $C$ then we do not know whether $d(s, u) = dist(s, u)$ or not even if $dist(s, u)$ is well-defined.

Question: How do we know whether there is a negative cycle $C$ reachable from $s$?
Bellman-Ford to detect Negative Cycles

for each \( u \in V \)
\[
d(s,u) = \infty
\]
\[
d(s,s) = 0
\]

for \( i = 1 \) to \(|V| - 1\) do

for each edge \( e = (u,v) \) do

Relax(e)

for each edge \( e = (u,v) \) do

If \( e=(u,v) \) is tensé then

Stop and output that \( s \) can reach a negative length cycle

Output for each \( u \in V \) dist(s,u) as \( d(s,u) \)
Correctness

Lemma

G has a negative cycle reachable from s if and only if there is a tense edge e after |V| − 1 iterations of Bellman-Ford.

Proof Sketch.

G has no negative length cycle reachable from s implies that all nodes u have a shortest path from s. Therefore \( d(s, u) = dist(s, u) \) after the |V| − 1 iterations. Therefore, there cannot be any tense edges left.
Correctness

Lemma

\( G \) has a negative cycle reachable from \( s \) if and only if there is a tense edge \( e \) after \(|V| - 1\) iterations of Bellman-Ford.

Proof Sketch.

\( G \) has no negative length cycle reachable from \( s \) implies that all nodes \( u \) have a shortest path from \( s \). Therefore \( d(s, u) = dist(s, u) \) after the \(|V| - 1\) iterations. Therefore, there cannot be any tense edges left.

If there is a negative cycle \( C \) then there is a tense edge after \(|V| - 1\) (in fact any number of) iterations. See Proposition (proof is exercise) about properties of the generic shortest path algorithm.
Finding the Paths and a Shortest Path Tree

for each \( u \in V \)
\[
d(s,u) = \infty
\]
\[
\text{prev}(u) = \text{null}
\]
endfor

\( d(s,s) = 0 \)

for \( i = 1 \) to \(|V| - 1\) do
  
  for each edge \( e = (u,v) \) do
    
    Relax(e)
  
endfor

If there is a tense edge \( e \) then
  
  Output that \( s \) can reach a negative cycle \( C \)

Else

  for each \( u \in V \)
  
  dist(s,u) = d(s,u)

Relax(e=(u,v))

  if \( d(s,v) > d(s,u) + \ell(u,v) \) then
    
    d(s,v) = d(s,u) + \ell(u,v)
    
    prev(v) = u

Note: prev pointers induce a shortest path tree.
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 be negative cycles not reachable from $s$.
- Run Bellman-Ford $|V|$ times, once from each node $u$?
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?

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

- Input graph $G = (V, E)$ with $m = |E|$ and $n = |V|$.
- $n$ outer iterations and $m$ Relax() operations in each iteration. Each Relax() operation is $O(1)$ time.
- Total running time: $O(mn)$. 
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.

- 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
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.
Algorithm for DAGs

\[
\text{for } i = 1 \text{ to } n \\
\quad d(s, v_i) = \infty \\
\quad d(s, s) = 0 \\
\text{for } i = 1 \text{ to } n-1 \text{ do} \\
\quad \text{for each edge } e \text{ in } \text{Adj}(v_i) \text{ do} \\
\qquad \text{Relax}(e) \\
\text{return } d \text{ values}
\]

**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.
Takeaway Points

- Shortest paths with potentially negative length edges arise in a variety of applications. Longest simple path problem is difficult (no known efficient algorithm and NP-Hard). We restrict attention to shortest walks and they are well defined only if there are no negative length cycles reachable from the source.

- A generic shortest path algorithm starts with distance estimates to the source and iteratively refines them by considering edges one at a time. The algorithm is guaranteed to terminate with correct distances if there are no negative length cycle. If a negative length cycle is reachable from the source it is guaranteed not to terminate.

- Dijkstra’s algorithm can also be thought of as an instantiation of the generic algorithm.
Bellman-Ford algorithm is an instantiation of the generic algorithm that in each iteration relaxes all the edges. It recognizes negative length cycles if there is a tense edges in the $n$'th iteration. For a vertex $u$ with a shortest path to the source with $i$ edges the algorithm has the correct distance after $i$ iterations. Running time of Bellman-Ford algorithm is $O(nm)$.

Bellman-Ford can be adapted to find a negative length cycle in the graph by adding a new vertex.

If we have a DAG then it has no negative length cycle and hence shortest paths exists even with negative lengths. One can compute single-source shortest paths in a DAG in linear time. This implies that one can also compute longest paths in a DAG in linear time.