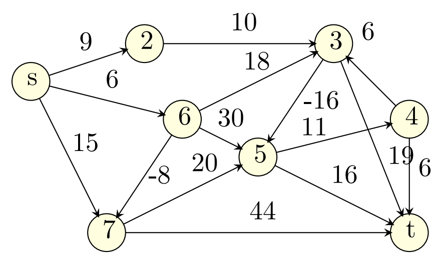
Chapter 5

Shortest Path Algorithms

OLD CS 473: Fundamental Algorithms, Spring 2015
February 3, 2015

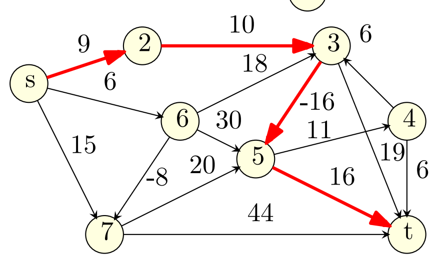
5.1 Shortest Paths with Negative Length Edges

5.1.0.1 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.

(A) Given nodes s, t find shortest path from s to t.
(B) Given node s find shortest path from s to all other nodes.

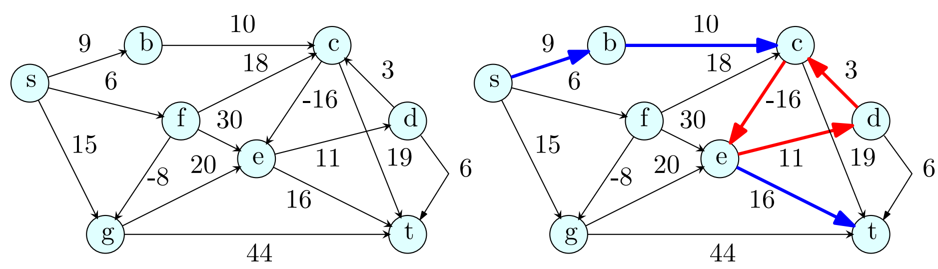


5.1.0.2 Negative Length Cycles

Definition 5.1.1. *A cycle C is a* **negative length cycle** *if the sum of the edge lengths of C is negative.*

5.1.0.3 Shortest Paths and Negative Cycles

(A) Given $G = (V, E)$ with edge lengths and s, t. Suppose
   (A) G has a negative length cycle C, and
   (B) s can reach C and C can reach t.
(B) **Question:** What is the shortest ***distance*** from s to t?



(C) Possible answers: Define shortest distance to be:
  (A) undefined, that is $-\infty$, OR
  (B) the length of a shortest ***simple*** path from s to t.

5.1.0.4 Shortest Paths and Negative Cycles

Lemma 5.1.2. *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**!

5.1.1 Shortest Paths with Negative Edge Lengths

5.1.1.1 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.

Questions:

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

5.1.2 Shortest Paths with Negative Edge Lengths

5.1.2.1 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 more involved than those for directed graphs. Beyond the scope of this class. If interested, ask instructor for references.

5.1.2.2 Why Negative Lengths?

Several Applications

(A) Shortest path problems useful in modeling many situations — in some negative lengths are natural

(B) Negative length cycle can be used to find arbitrage opportunities in currency trading

(C) Important sub-routine in algorithms for more general problem: minimum-cost flow

5.1.3 Negative cycles

5.1.3.1 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.

Questions:

(A) Is there an arbitrage opportunity?

(B) Given currencies s, t what is the best way to convert s to t (perhaps via other intermediate currencies)?

5.1.4 Negative cycles

5.1.4.1 Application to Currency Trading

Concrete example:

(A) 1 Chinese Yuan = 0.1116 Euro

(B) 1 Euro = 1.3617 US dollar

(C) 1 US Dollar = 7.1 Chinese Yuan.

As such... Thus, if exchanging 1 \$ $\rightarrow$ Yuan $\rightarrow$ Euro $\rightarrow$ \$, we get: $0.1116 * 1.3617 * 7.1 = 1.07896$\$.

5.1.4.2 Reducing Currency Trading to Shortest Paths

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

(B) Create currency trading *directed* graph $G = (V, E)$:

    (A) For each currency i there is a node $v_i \in V$

    (B) $E = V \times V$: an edge for each pair of currencies

    (C) edge length $\ell(v_i, v_j) = -\log(exch(i, j))$ **can be negative**

(C) **Exercise:** Verify that

    (A) There is an arbitrage opportunity if and only if G has a negative length cycle.

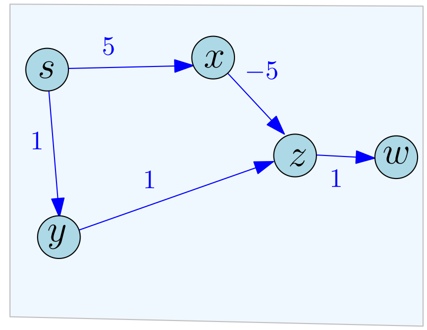
    (B) The best way to convert currency i to currency j is via a shortest path in G from i to j. If d is the distance from i to j then one unit of i can be converted into 2^d units of j.

5.1.5 Reducing Currency Trading to Shortest Paths

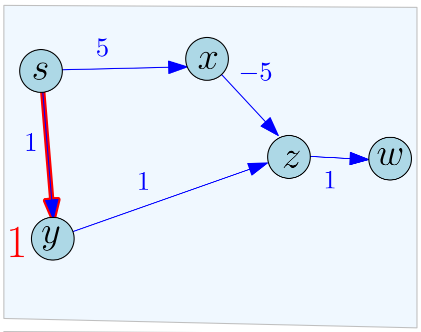
5.1.5.1 Math recall - relevant information

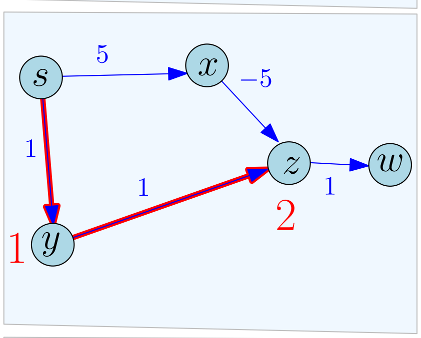
(A) $\log(\alpha_1 * \alpha_2 * \cdots * \alpha_k) = \log \alpha_1 + \log \alpha_2 + \cdots + \log \alpha_k$.
(B) $\log x > 0$ if and only if $x > 1$.

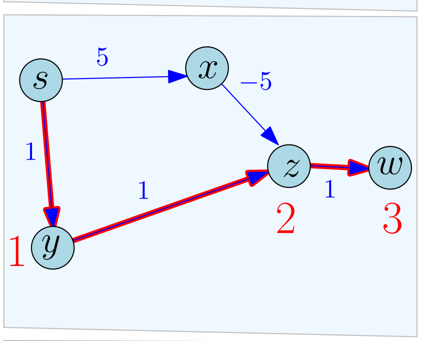
5.1.5.2 Dijkstra's Algorithm and Negative Lengths

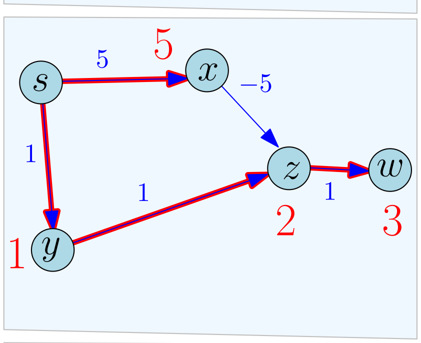


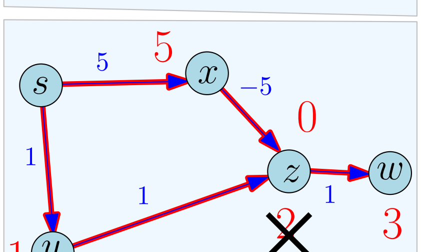
With negative cost edges, Dijkstra's algorithm fails











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 $dist(s, v_i) \le dist(s, v_{i+1})$ for $0 \le i < k$. Holds true only for non-negative edge lengths.

5.1.5.3 Shortest Paths with Negative Lengths

Lemma 5.1.3. *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 $1 \le i < k$:*

(A) $s = v_0 \to v_1 \to v_2 \to \ldots \to v_i$ is a shortest path from s to v_i

*(B) **False:** $dist(s, v_i) \le dist(s, v_k)$ for $1 \le i < k$. **Holds true only for non-negative edge lengths.***

Cannot explore nodes in increasing order of distance! We need a more basic strategy.

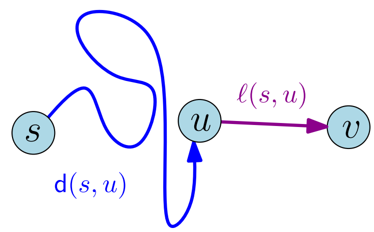
5.1.5.4 A Generic Shortest Path Algorithm

(A) Start with distance estimate for each node $d(s, u)$ set to ∞

(B) Maintain the invariant that there is an $s \to u$ path of length $d(s, u)$. Hence $d(s, u) \ge dist(s, u)$.

(C) Iteratively refine $d(s, \cdot)$ values until they reach the correct value $dist(s, \cdot)$ values at termination

Must hold that... $d(s, v) \le d(s, u) + \ell(u, v)$



5.1.5.5 A Generic Shortest Path Algorithm

Question: How do we make progress?

Definition 5.1.4. *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**
    $d(s, v) \Leftarrow d(s, u) + \ell(u, v)$

5.1.5.6 A Generic Shortest Path Algorithm

Invariant If a vertex u has value $d(s, u)$ associated with it, then there is a $s \rightsquigarrow u$ walk of length $d(s, u)$.

Proposition 5.1.5. **Relax** *maintains the invariant on $d(s, u)$ values.*

Proof: Indeed, if **Relax**$((u, v))$ changed the value of $d(s, v)$, then there is a walk to u of length $d(s, u)$ (by invariant), and there is a walk of length $d(s, u) + \ell(u, v)$ to v through u, which is the new value of $d(s, v)$. ∎

5.1.5.7 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 $d(s, u) = d(s, v) = \infty$ then edge is not tense.

5.1.5.8 Properties of the generic algorithm

Proposition 5.1.6. *If u is not reachable from s then $d(s, u)$ remains at ∞ throughout the algorithm.*

5.1.5.9 Properties of the generic algorithm

Proposition 5.1.7. *If a negative length cycle C is reachable by s then there is always a tense edge and hence the algorithm never terminates.*

Proof Let $C = v_0, v_1, \ldots, v_k$ be a negative length cycle. Suppose algorithm terminates. Since no edge of C was tense, for $i = 1, 2, \ldots, k$ we have $d(s, v_i) \leq d(s, v_{i-1}) + \ell(v_{i-1}, v_i)$ and $d(s, v_0) \leq d(s, v_k) + \ell(v_k, v_0)$. Adding up all the inequalities we obtain that length of C is non-negative!

5.1.5.10 Proof in more detail...

$$
\begin{aligned}
d(s, v_1) &\leq d(s, v_0) + \ell(v_0, v_1) \\
d(s, v_2) &\leq d(s, v_1) + \ell(v_1, v_2) \\
&\cdots \\
d(s, v_i) &\leq d(s, v_{i-1}) + \ell(v_{i-1}, v_i) \\
&\cdots \\
d(s, v_k) &\leq d(s, v_{k-1}) + \ell(v_{k-1}, v_k) \\
d(s, v_0) &\leq d(s, v_k) + \ell(v_k, v_k)
\end{aligned}
$$

$$
\sum_{i=0}^{k} d(s, v_i) \leq \sum_{i=0}^{k} d(s, v_i) + \sum_{i=1}^{k} \ell(v_{i-1}, v_i) + \ell(v_k, v_0)
$$

$$0 \leq \sum_{i=1}^{k} \ell(v_{i-1}, v_i) + \ell(v_k, v_0) = \text{len}(C).$$

C is a not a negative cycle. Contradiction.

5.1.5.11 Properties of the generic algorithm

Corollary 5.1.8. *If the algorithm terminates then there is no negative length cycle C that is reachable from s.*

5.1.5.12 Properties of the generic algorithm

Lemma 5.1.9. *If the algorithm terminates then $d(s, u) = dist(s, u)$ for each node u (and s cannot reach a negative cycle).*

Proof of lemma; see future slides.

5.1.6 Properties of the generic algorithm

5.1.6.1 If estimate distance from source too large, then ∃ tense edge...

Lemma 5.1.10. *Assume there is a path $\pi = v_1 \to v_2 \to \cdots \to v_k$ from $v_1 = s$ to $v_k = u$ (not necessarily simple!): $\ell(\pi) = \sum_{i=1}^{k-1} \ell(v_i, v_j) < d(s, u)$.*

Then, there exists a tense edge in G.

Proof: Assume π is the shortest (in number of edges) such path, and observe that it must be that $\ell(v_1 \to \cdots v_{k-1}) \geq d(s, v_{k-1})$. But then, we have that $d(s, v_{k-1}) + \ell(v_{k-1}, v_k) \leq \ell(v_1 \to \cdots v_{k-1}) + \ell(v_{k-1}, v_k) = \ell(\pi) < d(s, v_k)$. Namely, $d(s, v_{k-1}) + \ell(v_{k-1}, v_k) < d(s, v_k)$ and the edge (v_{k-1}, v_k) is tense. ■

$\implies$ If for any vertex u: $d(s, u) > \text{dist}(s, u)$ then the algorithm will continue working!

5.1.6.2 Generic Algorithm: Ordering Relax operations

```
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 for u ∈ V(G)
```

Question: How do we pick edges to relax?

Observation: Suppose $s \to v_1 \to \ldots \to v_k$ 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 $d(s, v_k) = dist(s, v_k)$!

5.1.6.3 Ordering Relax operations

(A) **Observation:** Suppose $s \to v_1 \to \ldots \to v_k$ is a shortest path.

If **Relax**(s, v_1), **Relax**(v_1, v_2), ..., **Relax**(v_{k-1}, v_k) are done in *order* then $d(s, v_k) = dist(s, v_k)$! (Why?)
(B) We don't know the shortest paths so how do we know the order to do the Relax operations?

5.1.6.4 Ordering Relax operations

(A) We don't know the shortest paths so how do we know the order to do the Relax operations?
(B) We don't!
   (A) Relax *all* edges (even those not tense) in some arbitrary order
   (B) Iterate $|V| - 1$ times
   (C) 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).

5.1.6.5 The Bellman-Ford (BellmanFord) Algorithm

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

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

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

5.1.6.6 BellmanFord Algorithm: Scanning Edges

One possible way to scan edges in each iteration.

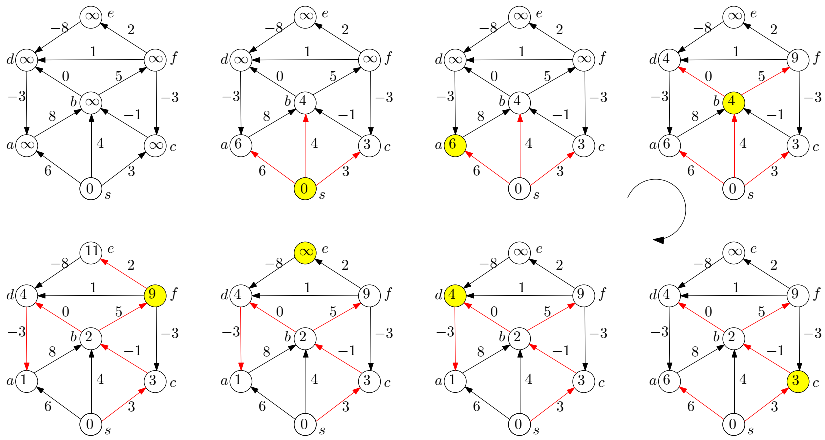


Figure 5.1: One iteration of **BellmanFord** that Relaxes all edges by processing nodes in the order s, a, b, c, d, e, f. Red edges indicate the prev pointers (in reverse)

```
Q is an empty queue
for each u ∈ V do
    d(s,u) = ∞
    enq(Q, u)
d(s,s) = 0

for i = 1 to |V| − 1 do
    for j = 1 to |V| do
        u = deq(Q)
        for each edge e in Adj(u) do
            Relax(e)
        enq(Q, u)

for each u ∈ V do
    dist(s,u) = d(s,u)
```

5.1.6.7 Example

5.1.6.8 Example

5.1.6.9 Correctness of the BellmanFord Algorithm

Lemma 5.1.11. *G: a directed graph with arbitrary edge lengths, v: a node in V s.t. there is a shortest path from s to v with i edges. Then, after i iterations of the loop in* **BellmanFord**, $d(s,v) = dist(s,v)$

Proof: By induction on i.

(A) Base case: $i = 0$. $d(s,s) = 0$ and $d(s,s) = dist(s,s)$.

(B) Induction Step: Let $s \to v_1 \ldots \to v_{i-1} \to v$ be a shortest path from s to v of i hops.
   (A) 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.

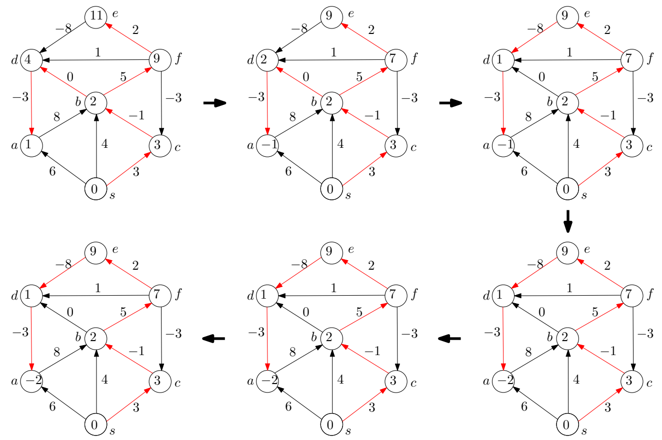


Figure 5.2: 6 iterations of **BellmanFord** starting with the first one from previous slide. No changes in 5th iteration and 6th iteration.

(B) In iteration i, **Relax**(v_{i-1}, v_i) sets $d(s, v_i) = \text{dist}(s, v_i)$.
(C) Note: Relax does not change $d(s, u)$ once $d(s, u) = \text{dist}(s, u)$.

∎

5.1.6.10 Correctness of BellmanFord Algorithm

Corollary 5.1.12. *After $|V| - 1$ iterations of* **BellmanFord**, *$d(s, u) = dist(s, u)$ 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?

5.1.6.11 BellmanFord to detect Negative Cycles

```
for each u ∈ V do
        d(s,u) = ∞
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 tense then
            Stop and output that s can reach
                    a negative length cycle

Output for each u ∈ V:  d(s,u)
```

5.1.6.12 Correctness

Lemma 5.1.13. *G has a negative cycle reachable from s if and only if there is a tense edge e after $|V| - 1$ iterations of* **BellmanFord**.

Proof:[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 lemma about properties of the generic shortest path algorithm. ■

5.2 Negative cycle detection

5.2.0.13 Finding the Paths and a Shortest Path Tree

```
BellmanFord:
    for each u ∈ V do
        d(s, u) = ∞
        prev(u) = null
    d(s, s) = 0
    for i = 1 to |V| − 1 do
        for each edge e = (u, v) do
            Relax(e)
    if there is a tense edge e then
        Output that s can reach a negative cycle C
    else
        for each u ∈ V do
            output d(s, u)
```

```
Relax(e = (u, v)):
    if (d(s, v) > d(s, u) + ℓ(u, v)) then
        d(s, v) = d(s, u) + ℓ(u, v)
        prev(v) = u
```

Note: prev pointers induce a shortest path tree.

5.2.0.14 Negative Cycle Detection

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

(A) **BellmanFord** checks whether there is a negative cycle C that is reachable from a specific vertex s. There may negative cycles not reachable from s.

(B) Run **BellmanFord** $|V|$ times, once from each node u?

5.2.0.15 Negative Cycle Detection

(A) Add a new node s' and connect it to all nodes of G with zero length edges.
(B) **BellmanFord** from s' will fill find a negative length cycle if there is one.
(C) **Exercise:** why does this work?
(D) Negative cycle detection can be done with one **BellmanFord** invocation.

5.2.0.16 Running time for BellmanFord

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

5.2.0.17 Dijkstra's Algorithm with Relax()

```
for each node u ≠ s do
    d(s, u) = ∞
d(s, s) = 0
S = ∅
while (S ≠ V) do
    Let v be node in V − S with min d value
    S = S ∪ {v}
    for each edge e in Adj(v) do
        Relax(e)
```

5.3 Shortest Paths in DAGs

5.3.0.18 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.

(A) Given nodes s, t find shortest path from s to t.
(B) Given node s find shortest path from s to all other nodes.

Simplification of algorithms for DAGs

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

5.3.0.19 Algorithm for DAGs

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

Observation:

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

5.3.0.20 Algorithm for DAGs

(A) Code:

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

    for i = 1 to n − 1 do
        for each edge e in Adj(v_i) do
            Relax(e)

    return d(s, ·) values computed
```

(B) **Correctness:** induction on i and observation in previous slide.
(C) **Running time:** $O(m + n)$ time algorithm! Works for negative edge lengths and hence can find *longest* paths in a DAG.

5.3.0.21 Takeaway Points

(A) Shortest paths with potentially negative length edges arise in a variety of applications.
(B) Longest simple path problem is difficult (no known efficient algorithm and **NP-Hard**).
(C) Restrict attention to shortest walks. Well defined only if there are no negative length cycles reachable from the source.
(D) In this case shortest walk = shortest path.
(E) Generic shortest path algorithm starts with distance estimates to the source. Iteratively relaxes the edges one by one.
(F) ...Guaranteed to terminate with correct distances if no negative length cycle reachable from s.
(G) If negative length cycle reachable from $s \implies$ no termination.
(H) Dijkstra's algorithm also instantiation of generic algorithm.

5.3.0.22 Points continued

(A) **BellmanFord** is instantiation of generic algorithm.
(B) ...in each iteration relaxes all the edges.
(C) Discovers negative length cycles if there is tense edge in the nth iteration.
(D) For vertex u with a shortest path to the source with i edges the algorithm has the correct distance after i iterations.
(E) Running time of **BellmanFord** algorithm is $O(nm)$.
(F) **BellmanFord** can be adapted to find a negative length cycle in the graph by adding a new vertex.
(G) If we have a DAG then it has no negative length cycle and hence shortest paths exists even with negative lengths.
(H) Can compute single-source shortest paths in DAG in linear time.
(I) Implies one can compute longest paths in a DAG in linear time.

5.3.0.23 Questions for a possible written quiz...

(A) Given a directed graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices and m edges, describe how to compute a cycle in G if such a cycle exist. What is the running time of your algorithm?
(B) As above, but assume edges have weights (negative or positive). Describe how to detect a negative cycle in G?
(C) Describe how to modify your algorithm from (B) so that it outputs the negative cycle.

5.4 Not for lecture

5.4.1 A shortest walk that visits all vertices...

5.4.1.1 ... in a graph might have to be of length $\Omega(n^2)$