CS 473: Algorithms

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

University of Illinois, Urbana-Champaign

Fall 2009

Part I

Dijkstra's Algorithm Recap

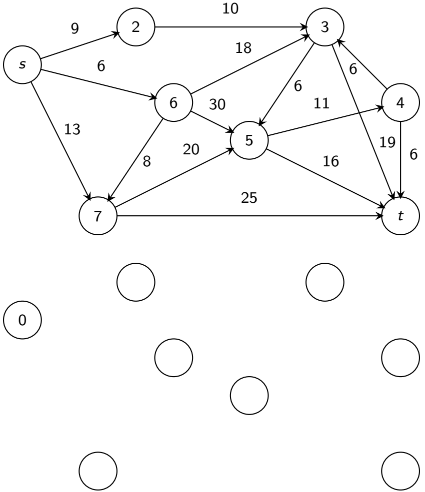
Dijkstra's Algorithm using Priority Queues

Shortest paths from node s to all nodes in V:

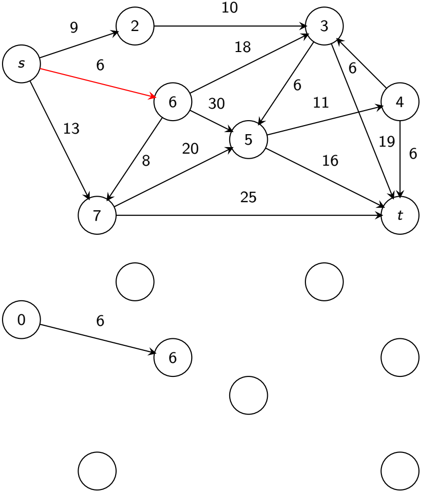
```
Q = makePQ()
insert(Q, (s,0))
for each node u ≠ s
    insert(Q, (u,∞))
S = ∅
for i = 1 to |V| do
    (v, dist(s,v)) = extractMin(Q)
    S = S ∪ {v}
    For each u in Adj(v) do
        decreaseKey(Q, (u, min (dist(s,u), dist(s,v) + ℓ(v,u))))
```

Algorithm adds nodes to S in order of increasing distance from s

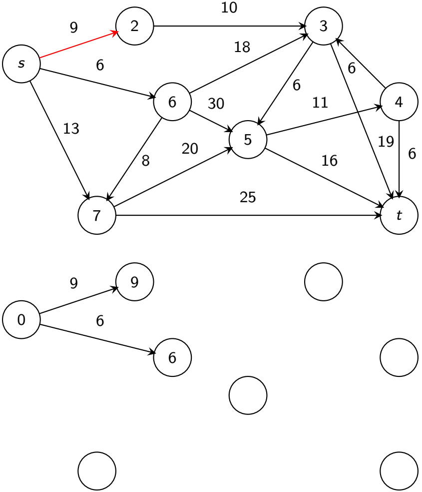
Example



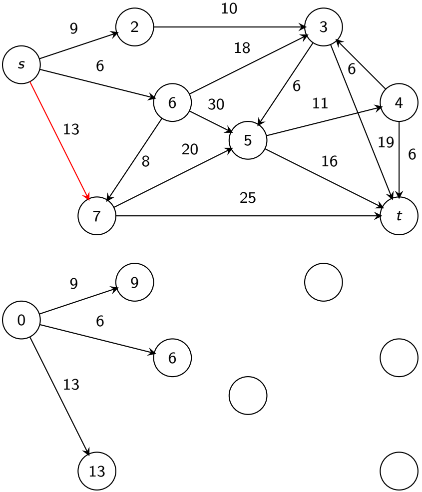
Example



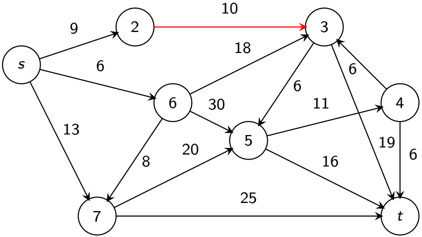
Example

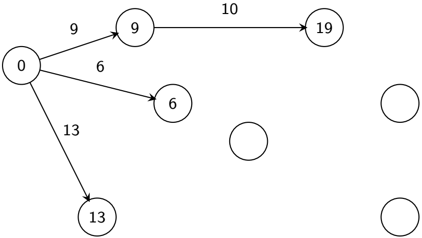


Example

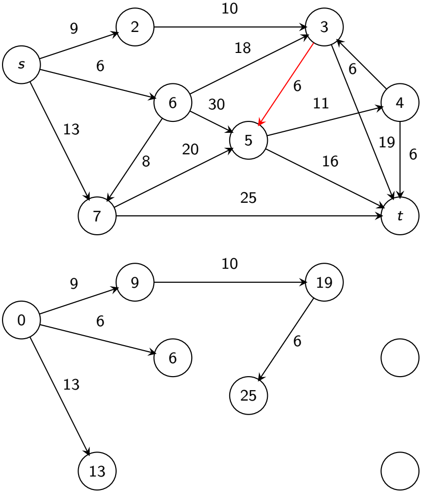


Example

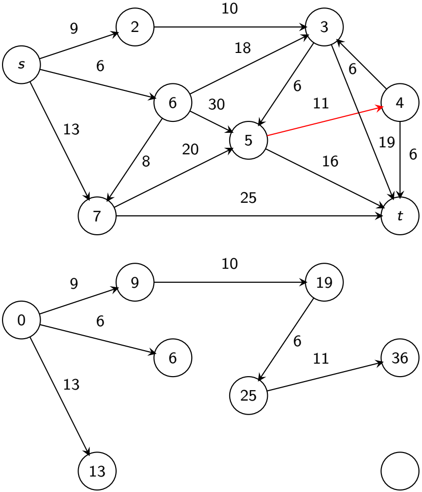




Example

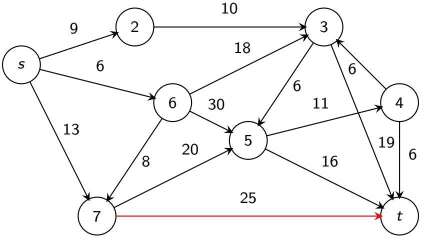


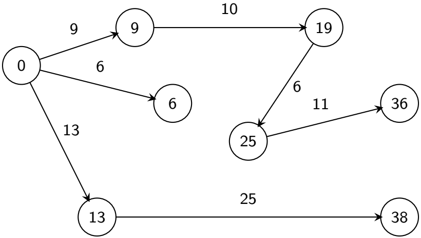
Example



s
2
3
4
5
6
7
t
9
10
6
18
13
30
6
6
11
8
20
19
6
16
25
0
9
19
6
36
25
13
10
9
6
6
11
13

Example


9
10
6
18
13
6
30
6
11
8
20
19
6
16
25
s
2
3
6
5
4
7
t


9
10
6
6
11
13
25
0
9
19
6
36
25
13
38

Shortest Path Tree

Dijkstra's algorithm finds the shortest path distances from s to V.
Question: How do we find the paths themselves?

Shortest Path Tree

Dijkstra's algorithm finds the shortest path distances from s to V.
Question: How do we find the paths themselves?

```
Q = makePQ()
insert(Q, (s,0))
prev(s) = null
for each node u ≠ s
    insert(Q, (u,∞))
    prev(u) = null

S = ∅
for i = 1 to |V| do
    (v, dist(s,v)) = extractMin(Q)
    S = S ∪ {v}
    For each u in Adj(v) do
        if (dist(s,v) + ℓ(v,u) < dist(s,u) ) then
            decreaseKey(Q, (u, dist(s,v) + ℓ(v,u)))
            prev(u) = v
```

Shortest Path Tree

Lemma

The edge set $(u, prev(u))$ is the reverse *of a shortest path tree rooted at s. For each u, the reverse of the path from u to s in the tree is a shortest path from s to u.*

Proof Sketch.

- The edgeset $\{(u, prev(u)) \mid u \in V\}$ induces a directed in-tree rooted at s (Why?)
- Use induction on $|S|$ to argue that the tree is a shortest path tree for nodes in V.

$\square$

Shortest paths to s

Dijkstra's algorithm gives shortest paths from s to all nodes in V.

How do we find shortest paths from all of V to s?

Shortest paths to s

Dijkstra's algorithm gives shortest paths from s to all nodes in V.

How do we find shortest paths from all of V to s?

- In undirected graphs shortest path from s to u is a shortest path from u to s so there is no need to disjtinguish.
- In directed graphs, use Dijkstra's algorithm in G^{rev}!

Part II

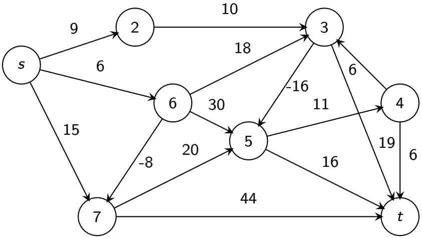
Shortest Paths with Negative Length Edges

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.

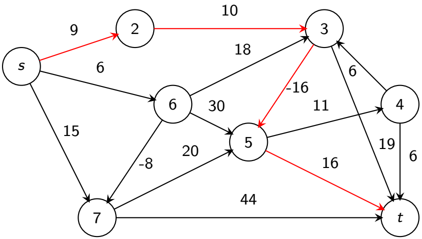


Single-Source Shortest Paths with Negative Edge Lengths

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.

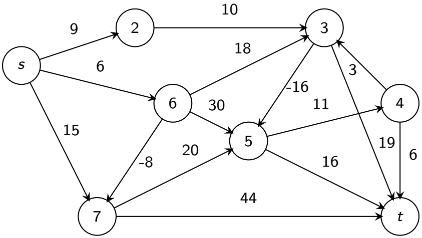
- Given nodes s, t find shortest path from s to t.
- Given node s find shortest path from s to all other nodes.



Negative Length Cycles

Definition

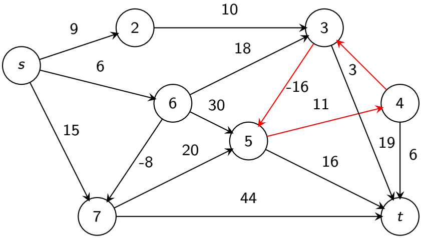
A cycle C is a negative length cycle if the sum of the edge lengths of C is negative.



Negative Length Cycles

Definition

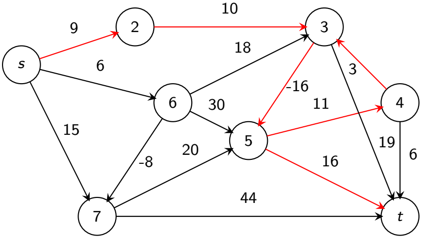
A cycle C is a negative length cycle if the sum of the edge lengths of C is negative.



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

- 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 lenth 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 lenth of a shortest *simple* path from s to t.

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.*

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 lenth of a shortest *simple* path from s to t.

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!

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 and t can reached from, 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 other nodes.
- Check if G has a negative length cycle or not.

Why Negative Lengths?

Several Applications

- Shortest path problems useful in modeling many situations — in some negative lenths are natural
- Negative length cycle can be used to find arbitrage opportunities in currency trading
- 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)?

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 $exch(i, k_1) \times exch(k_1, k_2) \ldots \times exch(k_h, j)$ 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 $exch(i, k_1) \times exch(k_1, k_2) \ldots \times exch(k_h, j)$ 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 $\ell(v_i, v_j) =$

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 $exch(i, k_1) \times exch(k_1, k_2) \ldots \times exch(k_h, j)$ 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 $\ell(v_i, v_j) = \log(exch(i, j))$ can be negative

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 $exch(i, k_1) \times exch(k_1, k_2) \ldots \times exch(k_h, j)$ 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 $\ell(v_i, v_j) = \log(exch(i, j))$ 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.

Shortest Paths with Negative Edge Lengths: Problems

Algorithmic Problems

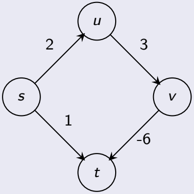
Input A directed graph $G = (V, E)$ with arbitrary (including negative) edge lengths. For edge $e = (u, v)$, $\ell(e) = \ell(u, v)$ is its length.

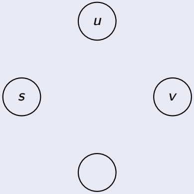
- Given nodes s, t, either find a negative length cycle C that s can reach and t can reached from, or find a shortest path from s to t.
- Given node s, either find a negative length cycle C that s can reach or find shortest path distances from s to all other nodes.
- Check if G has a negative length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

Dijkstra's Algorithm and Negative Lengths

Observation

With negative cost edges, Dijkstra's algorithm fails

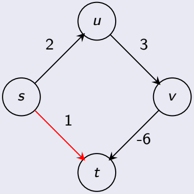


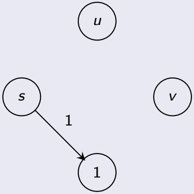


Dijkstra's Algorithm and Negative Lengths

Observation

With negative cost edges, Dijkstra's algorithm fails

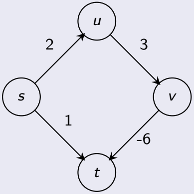


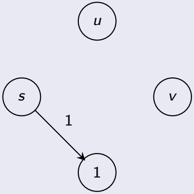


Dijkstra's Algorithm and Negative Lengths

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.

Shortest Paths with Negative 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*

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: $dist(s, v_i) \leq dist(s, v_k)$ 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 $s = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k$ 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: $dist(s, v_i) \leq dist(s, v_k)$ for $1 \leq i < k$. Holds true only for non-negative edge lengths.*

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

A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to ∞
- 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 ∞
- 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 $d(s, v) > d(s, u) + \ell(u, v)$.

A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to ∞
- 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) + ℓ(u,v)) then
        d(s,v) = d(s,u) + ℓ(u,v)
```

A Generic Shortest Path Algorithm

- Start with distance estimate for each node $d(s, u)$ set to ∞
- 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) + ℓ(u,v)) then
        d(s,v) = d(s,u) + ℓ(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
```

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
```

Lemma

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

Proof is left as an exercise after seeing future slides.

Dijkstra's Algorithm with Relax()

```
for each node u ≠ s
    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 edeg e in Adj(v) do
        Relax(e)
```

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
```

Question: How do we pick edges to relax?

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
```

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), …, Relax(v_{k-1}, v_k) are done in *order* then $d(s, v_k) = dist(s, v_k)$!

Ordering Relax operations

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?)

Ordering 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), ..., Relax(v_{k-1}, v_k) are done in *order* then $d(s, v_k) = dist(s, v_k)$! (Why?)

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

Ordering Relax operations

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?)

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 ∈ V
    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
    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 ∈ V
    d(s,u) = ∞
    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 ∈ V
    dist(s,u) = d(s,u)
```

Example

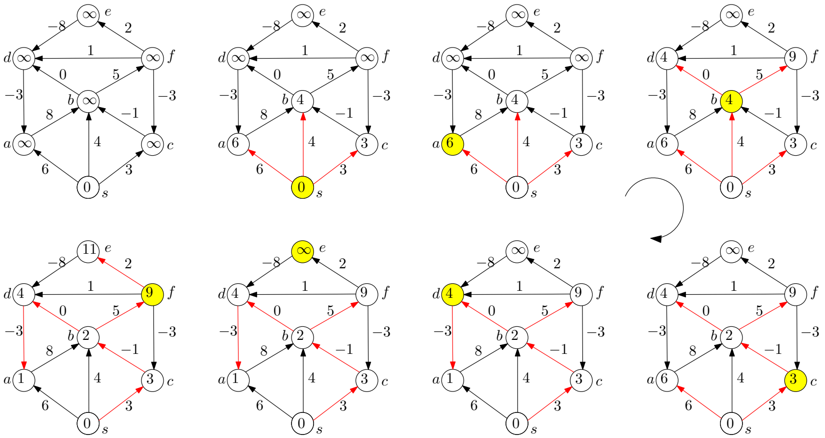


Figure: One iteration of Bellman-Ford that Relaxes all edges by processing nodes in the order s, a, b, c, d, e, f. Red edges indicate the prev pointers (in reverse)

Example

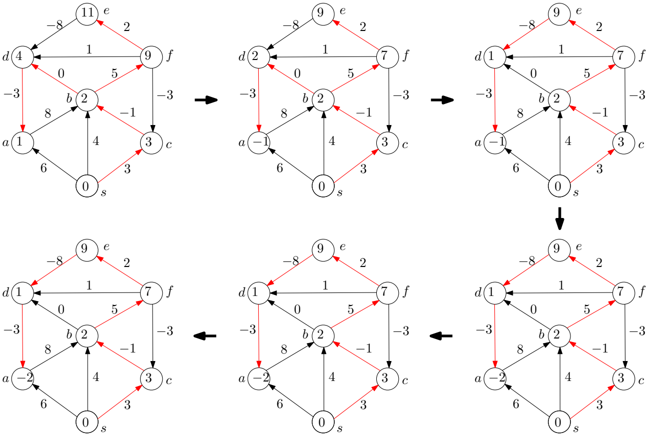


Figure: 6 iterations of Bellman-Ford starting with the first one from previous slidefigure. No changes in 5th iteration and 6th iteration.

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, $d(s, v) = dist(s, v)$

Proof.

By induction on i.

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

□

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.

□

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 $s \to v_1 \ldots \to v_{i-1} \to v$ 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.

□

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 $s \to v_1 \ldots \to v_{i-1} \to v$ 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 $d(s, v_i) = dist(s, v_i)$.

□

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 $s \to v_1 \ldots \to v_{i-1} \to v$ 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

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.

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 and C can reach s then $dist(s, u)$ is not 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 and C can reach s then $dist(s, u)$ is not defined.

Question: How do we know whether there is a negative cycle C reachable from s and whether $d(s, u)$ is valid or not?

Bellman-Ford to detect Negative Cycles

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

for each edge e = (u, v) do
    If e=(u,v) is tense then
        (* s can reach a negative length cycle C and C can reach v *)
        dist(s,v) = −∞
```

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 are no tense edges after $|V| - 1$ iterations, then $d(s, u) = dist(s, u)$ for all nodes. See exercise after the generic shortest path algorithm. $\square$

Finding the Paths

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

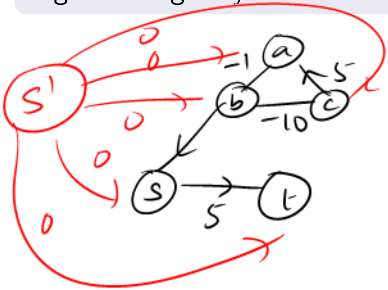
Modified Relax()

```
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
```

Negative Cycle Detection

Negative Cycle Detection

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



Negative Cycle Detection

Negative Cycle Detection

Given directed graph G with arbirary 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. Exercise: why does this work?
- Negative cycle detection can be done with one Bellman-Ford invocation.

Running time for Bellman-Ford

- $O(|V|)$ iterations and $O(|E|)$ Relax() operations in each iteration. Each Relax() operation is $O(1)$ time.
- Total running time: $O(|V||E|)$.

Part III

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

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.

Algorithm for DAGs

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

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

return d 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