CSE 100: Algorithm Design and Analysis
Chapter 24: Single-Source Shortest Paths

Sungjin Im

University of California, Merced

Last Update: 4-21-2023

Single-Source Shortest Paths

Problem definition

Single-pair shortest-path problem:

Input: Directed graph $G = (V, E)$ with weight/distance $w(u, v)$ on each edge $(u, v) \in E$, and a pair of vertices $s, t \in V$.

Output: A shortest path from s to t.

Single-destination shortest-paths problem:

Input: G and w as above, and a destination vertex t.

Output: A shortest path to t from each vertex v.

Single-source shortest-paths problem:
Input: G and w as above, and a source vertex s.
Output: A shortest path from s to each vertex v.

Preliminaries

Terminology:

- Path: a sequence of edges that connect a sequence of vertices. So a path P can be represented as $\langle v_0, v_1, ..., v_k \rangle$ where $v_0, v_1, ..., v_k \in V$ and $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k) \in E$.
- Simple Path: a path P is said to be simple if no vertex appears more than once on the path.
  * It is a convention that a path refers to a simple path; however, this is not necessarily the case in the textbook.
- The weight/distance of path P, $w(P)$ is defined as the total weight/distance of edges of the path P:
  $w(P) := w(v_0, v_1) + w(v_1, v_2) + ... + w(v_{k-1}, v_k)$

Preliminaries

$u \rightsquigarrow v$: v is reachable from u. (sometimes, a path from u to v)
$u \rightsquigarrow^{P} v$: P is a path from u to v.
Shortest-path weight $\delta(u, v)$ from u to v is defined as

$$\delta(u,v) = \begin{cases} \min\{w(P) : u \rightsquigarrow^{P} v\}, & \text{if there is a path from } u \text{ to } v \\ \infty & \textit{otherwise} \end{cases}$$

A shortest path from u to v is any path P from u to v such that $w(P) = \delta(u, v)$.

Key Lemma

Lemma (24.1. Optimality of Subpaths)

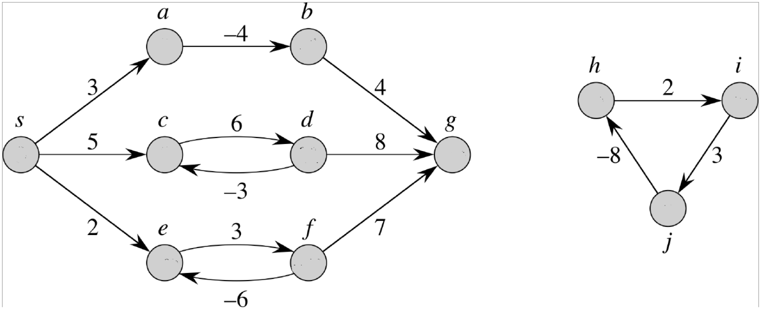
If $P = \langle v_0, v_1, ..., v_k \rangle$ is a shortest path from v_0 to v_k, then for any $0 \leq i \leq j \leq k$, $\langle v_i, v_{i+1}, ..., v_j \rangle$ is a shortest path from v_i to v_j. In other words, subpaths of shortest paths are also shortest paths.

Proof.

Cut-and-paste. Otherwise, one could get a 'better' shortest path from v_0 to v_k by replacing $\langle v_i, v_{i+1}, ..., v_j \rangle$ with a better path from v_i to v_j. $\square$

Some Issues

What is the shortest distance from s to each vertex?



Some Issues

If there is a path from s to a vertex u that includes a negative-weight cycle, then a shortest path from s to v is not well-defined, and we have $\delta(s, v) = -\infty$.

Lemma

If there is a shortest path from u to v, then there is such a path that is simple.

Proof.

If the path contains a negative-weight cycle:
If the path contains a positive-weight cycle:
If the path contains a 0-weight cycle: $\square$

We can assume without loss of generality that a shortest path contains no cycles.

Some Issues

Corollary

If there is a shortest path from u to v, then there is such a path that is simple, therefore consists of at most $|V| - 1$ edges.

Representing Shortest paths from a Single Source s

A shortest path can be encoded by π: $v.\pi$ means v's predecessor. In path $\langle v_0 = s, v_1, v_2, ..., v_k \rangle$, $v_0.\pi = NIL$, $v_1.\pi = v_0$, ..., $v_k.\pi = v_{k-1}$.

Definition

A shortest-paths tree rooted at s is a directed subraph $G' = (V', E')$ where $V' \subseteq V$ adn $E' \subseteq E$ such that

1. V' is the set of vertices reachable from s in G.
2. G' forms a rooted tree with root s, and
3. for all $v \in V'$, the unique simple path from s to $v \in G'$ is a shortest path from s to v in G.

Representing Shortest paths from a Single Source s

Lemma

Let $G = (V, E)$ be a weighted, directed graph with no negative-weight cycles reachable from source vertex $s \in V$. Then, there exists a shortest-paths tree rooted at s (over all reachable vertices from s).

Representing Shortest paths from a Single Source s

Lemma

Let $G = (V, E)$ be a weighted, directed graph with no negative-weight cycles reachable from source vertex $s \in V$. Then, there exists a shortest-paths tree rooted at s (over all reachable vertices from s).

Proof.

Use the optimality of subpaths lemma. $\square$

* Not necessarily unique.

Representing Shortest paths from a Single Source s

To compute the shotest path from source s to each vertex v (along with its distance), we only need to compute $v.d$ and $v.\pi$.
* $v.d = \infty$ implies that v is not reachable from s.

Key Subroutine: edge relaxation

First, initialize:

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

Think of $v.d$ as the shortest path (distance) estimate to v from s.

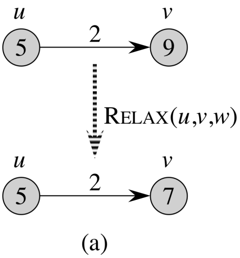
Key Subroutine: edge relaxation

Relaxing edge (u, v): if we can improve the (current) shortest path to v by replacing it with the (current) shortest path to u followed by (u, v), then update $v.d$ and $v.\pi$ accordingly.

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

Key Subroutine: edge relaxation

Relaxing edge (u, v): if we can improve the (current) shortest path to v by replacing it with the (current) shortest path to u followed by (u, v), then update $v.d$ and $v.\pi$ accordingly.



Key Subroutine: edge relaxation

Most algorithms are based on relaxing edges (after the initialization).

- Dijkstra's algorithm and the shortest-paths algorithm for DAGs relax each edge exactly once.
- Bellman-Ford algorithm relaxes each edge $|V| - 1$ times.

Useful Properties

Lemma (24.10. Triangle Inequality)

For any edge (u, v), $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Suppose we first set $v.d = \infty$ for all $v \in V$ except the source vertex, and update $v.d$ only via relaxing some edges. Then we have the following properties:

Lemma (24.11. Upper-bound Property)

We always have $v.d \geq \delta(s, v)$ *for all* $v \in V$*, and once* $v.d$ *achieves value* $\delta(s, v)$*, it never changes.*

Corollary (24.12. No-path Property)

If there is no path from s *to* v*, then we always have* $v.d = \delta(s, v) = \infty$.

Useful Properties

Lemma (24.14. Convergence Property)

If $s \leadsto u \to v$ is a shortest path in G for some $u, v \in G$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.

Lemma (24.15. Path-relaxation Property)

If $P = \langle v_0, v_1, ..., v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and the sequence of relaxations includes $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$ as a subsequence, then $v_k.d = \delta(s, v_k)$.

Lemma (24.17. Predecessor-subgraph Property)

Once $v.d = \delta(s, v)$ for all $v \in V$, the predcessor subgraph is a shortest-paths tree rooted at s.

The Bellman-Ford Algorithm

- Solves the single-source shortest-paths problem in the general case in which edges can have negative weights.
- Returns false if there is a negative-weight cycle reachable from s. Otherwise returns true along with the shortest paths and their distances.

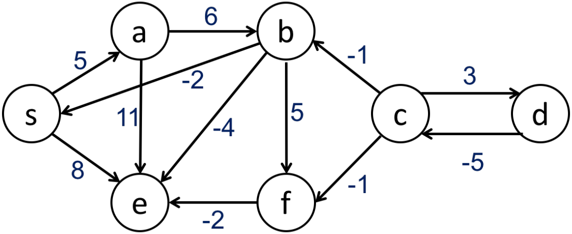
The Bellman-Ford Algorithm

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

The Bellman-Ford Algorithm

Example

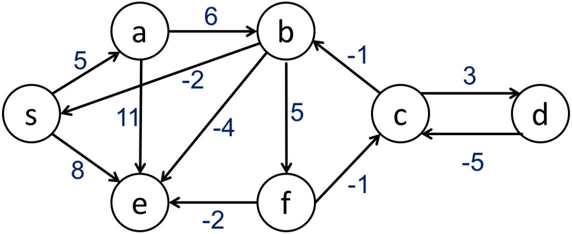
At any point in time,
$v.d = D$ means there's a path of distance D from s to v.
if $v.d = \infty$ means we haven't found a path from s to v.



The Bellman-Ford Algorithm

Example

At any point in time,
$v.d = D$ means there's a path of distance D from s to v.
if $v.d = \infty$ means we haven't found a path from s to v.



The Bellman-Ford Algorithm

Correctness: True case

At any point in time,
$v.d = D$ means there's a path of distance D from s to v.
if $v.d = \infty$ means we haven't found a path from s to v.

As claimed before all algorithms in this chapter are based on edge relaxation:

Lemma (24.11. Upper-bound Property)

We always have $v.d \geq \delta(s, v)$ for all $v \in V$, and once $v.d$ achieves value $\delta(s, v)$, it never changes.

Corollary (24.12. No-path Property)

If there is no path from s to v, then we always have $v.d = \delta(s, v) = \infty$.

The Bellman-Ford Algorithm

Correctness: True case

For any shortest (simple) path $P = \langle v_0 = s, v_1, v_2, ..., v_k = u$ from s to u, there is a relaxation subsequence of $(v_0, v_1), (v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)$ generated by the BF.

The Bellman-Ford Algorithm

Correctness: True case

For any shortest (simple) path $P = \langle v_0 = s, v_1, v_2, ..., v_k = u$ from s to u, there is a relaxation subsequence of $(v_0, v_1), (v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)$ generated by the BF. Note that $k \leq |V| - 1$.

The Bellman-Ford Algorithm

Correctness: True case

For any shortest (simple) path $P = \langle v_0 = s, v_1, v_2, ..., v_k = u$ from s to u, there is a relaxation subsequence of $(v_0, v_1), (v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)$ generated by the BF. Note that $k \leq |V| - 1$. Do you see why?

The Bellman-Ford Algorithm

Correctness: True case

For any shortest (simple) path $P = \langle v_0 = s, v_1, v_2, ..., v_k = u$ from s to u, there is a relaxation subsequence of $(v_0, v_1), (v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)$ generated by the BF. Note that $k \leq |V| - 1$. Do you see why?

Lemma (24.15. Path-relaxation Property)

If $P = \langle v_0, v_1, ..., v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and the sequence of relaxations includes $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$ as a subsequence, then $v_k.d = \delta(s, v_k)$.

Thanks to the observation and Lemma 24.15, we have $u.d = \delta(s, u)$ for all $u \in V$ at the end.

The Bellman-Ford Algorithm

Correctness: True case

We have $u.d = \delta(s, u)$ for all $u \in V$ at the end. So, $v.d \leq u.d + w(u, v)$ for all (u, v) by Triangle Inequality. So the BF returns True.

The Bellman-Ford Algorithm

Correctness: False case

For the sake of contradiction, suppose the BF returns True for the False case.
Say $c = \langle v_0, v_1, ..., v_k = v_0 \rangle$ be a negative-weight cycle (reachable from s), so we have,

$$w(v_0, v_1) + w(v_1, v_2) + ... + w(v_k, v_0) < 0.$$

But we know

$$\begin{array}{rcl} v_1.d & \leq & v_0.d + w(v_0, v_1) \\ v_2.d & \leq & v_1.d + w(v_1, v_2) \\ & & ... \\ v_k.d & \leq & v_{k-1}.d + w(v_{k-1}, v_k) \\ v_0.d & \leq & v_k.d + w(v_k, v_0) \end{array}$$

So, we have $0 \leq w(v_0, v_1) + w(v_1, v_2) + ... + w(v_k, v_0)$, a contradiction.

The Bellman-Ford Algorithm

Running Time

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

The Bellman-Ford Algorithm

Running Time

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

$O(EV)$

The Bellman-Ford Algorithm

Questions

Q. Suppose that there is an integer $m > 0$ such that there is a shortest path from s to each vertex v consisting of at most m edges. Further, we know the value of m. How many times do you need to iterate Lines 3 and 4?

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| - 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

The Bellman-Ford Algorithm

Questions

Q. We would like to find a shortest path from the single source to each vertex. If we see no change of $v.d$ for any vertex v, we can stop. True or False?

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

Single-source Shortest Paths in DAGs

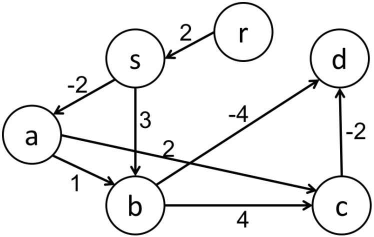
Questions

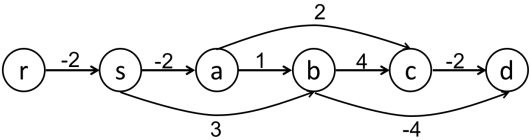
A DAG has no cycle. So no worries about negative-weight cycles.

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Single-source Shortest Paths in DAGs

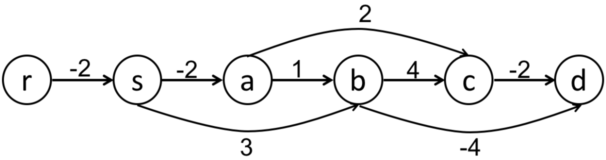
Eaxmple





Single-source Shortest Paths in DAGs

Eaxmple



Single-source Shortest Paths in DAGs

Correctness

Say $P = \langle v_0 = s, v_1, v_2, ..., v_k = v \rangle$ is a shortest path from s to v.
Then $v_0, v_1, ..., v_k$ must appear in the topological ordering in this order.
The relaxation sequence includes $(v_0, v_1), (v_1, v_2), ...(v_{k-1}, v_k)$ as a subsequence.
Hence if a vertex v is reachable from s, then $v.d = \delta(s, v)$ by the path-relaxation property.

Single-source Shortest Paths in DAGs

Running time

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Single-source Shortest Paths in DAGs

Running time

```
DAG-SHORTEST-PATHS(G, w, s)
topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u, taken in topologically sorted order
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Topological sort: $O(E + V)$.
Each edge is relaxed exactly once. So, $O(E)$ for all relaxations.
Thus, $O(E + V)$.

Single-source Shortest Paths when Edges Have Non-negative Weights

No worries about negative weight cycles since we have no negative-weight edges.

The Dijkstra algorithm

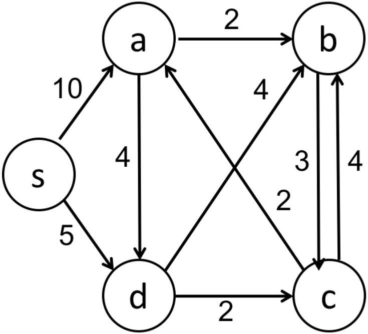
- maintains a set S of vertices whose shortest distances have been determined.
- grows S by adding a vertex $u \in V - S$ with the shortest distance estimate, $u.d$ (and relaxing all edges leaving u).
- is similar to Prim's algorithm.

Dijkstra Algorithm's Pseudocode

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

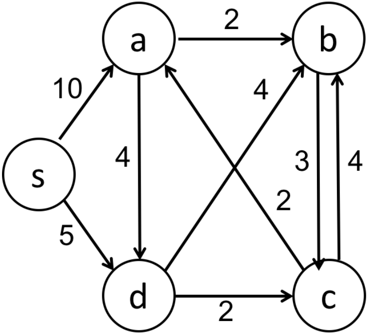
The Dijkstra algorithm: conceptual illustration

Example



The Dijkstra algorithm: actual illustration

Example



Dijkstra Algorithm's correctness

Proof sketch:
Want to show: when u is extracted from Q, we have $u.d = \delta(s, u)$, assuming that S consists of vertices whose shortest distances have been determined.

If $u = s$ or u is unreachable from s:
Otherwise, what happens if $u.d \neq \delta(s, u)$?

Where do we use the fact that edge weights are non-negative?

Dijkstra Algorithm's running time

Running Time:
Dijkstra Algorithm performs $O(V)$ Extract-Min and $O(E)$ Decrease-key.

DIJKSTRA(G, w, s)

```
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

Dijkstra Algorithm's running time

Running Time:
Dijkstra Algorithm performs $|V|$ Extract-Min and $|E|$ Decrease-key.
If min-priority queue is implemented by array,

- Extract-Min: $O(V)$.
- Decrease-Key: $O(1)$.

$O(V^2 + E) = O(V^2)$.

Dijkstra Algorithm's running time

Running Time:
Dijkstra Algorithm performs $|V|$ Extract-Min and $|E|$ Decrease-key.
If min-priority queue is implemented by binary heap,

- Extract-Min: $O(\log V)$.
- Decrease-Key: $O(\log V)$.

$O(V \log V + E \log V) = O(E \log V)$.

Dijkstra Algorithm's running time

Running Time:
Dijkstra Algorithm performs $|V|$ Extract-Min and $|E|$ Decrease-key.
If min-priority queue is implemented by Fibonacci heap,

- Extract-Min: $O(\log V)$.
- Decrease-Key: $O(1)$ (in an amortized sense).

$O(V \log V + E)$.