

Efficient Algorithms

Ekkapot Charoenwanit

Software Systems Engineering

TGGS

KMUTNB

Lecture 10: Graph Algorithms (Part II)

Single-Source Shortest Paths

Shortest Path Problem

In a shortest path problem, we are given a **weighted, directed** graph $G = (V, E, w)$ with a weight function $w: E \rightarrow \mathbb{R}$ that maps edges to **real-valued** weights.

The weight $w(p)$ of a path $p = \langle v_0, v_1, v_2, \dots, v_k \rangle$ is the sum of the weights of its constituent edges:

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

Shortest Path Problem

We define the **shortest path weight** from u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \rightsquigarrow v\} \\ \infty \end{cases} \quad (*)$$

A **shortest path** from u to v is then defined as **any path** p with weight $w(p) = \delta(u, v)$.

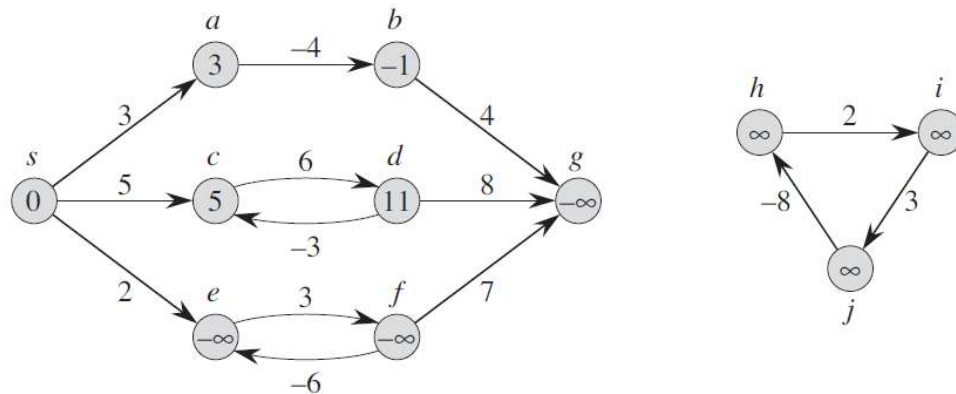
If there is no path from u to v , $\delta(u, v) = \infty$.

Even though there is a path u to v , a shortest path may not exist in the presence of **a negative-weight cycle** reachable from u .

Negative-Weight Cycle

Even though there is a path u to v , a shortest path may not exist in the presence of at least one **negative-weight cycle** reachable from u . Thus, $\delta(u, v) = -\infty$.

In the example below, a shortest path from s to f is **undefined** because we can always find a path with a smaller weight by traversing the negative-weight cycle $\langle e, f, e \rangle$ as many times as we want before reaching f .



Optimal Substructure

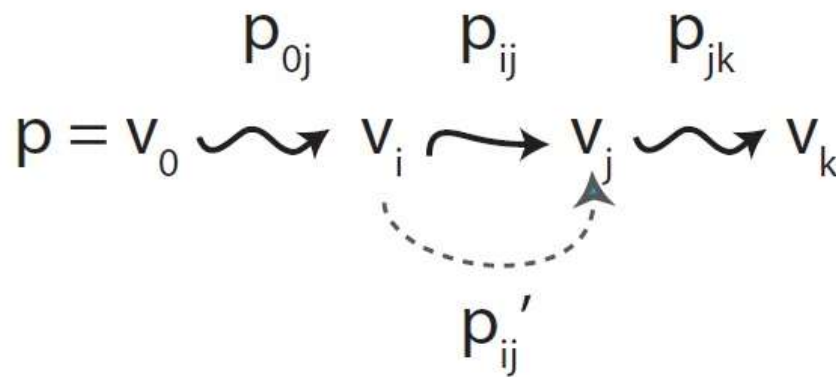
Shortest-path algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.

The following **lemma** states the **optimal substructure property** of shortest paths:

Lemma: Let $p = \langle v_0, v_1, v_2, \dots, v_k \rangle$ be a shortest path from v_0 to v_k . Then, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ be a subpath of p from v_i to v_j . Then p_{ij} is a shortest path from v_i to v_j .

Optimal Substructure

Lemma: Let $p = \langle v_0, v_1, v_2, \dots, v_k \rangle$ be a shortest path from v_0 to v_k . Then, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ be a subpath of p from v_i to v_j . Then p_{ij} is a shortest path from v_i to v_j .



Optimal Substructure

Lemma: Let $p = \langle v_0, v_1, v_2, \dots, v_k \rangle$ be a shortest path from v_0 to v_k . Then, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \dots, v_j \rangle$ is a subpath of p from v_i to v_j . Then p_{ij} is a shortest path from v_i to v_j .

Proof: We will prove using a **cut-and-paste argument**.

Assume for the purpose of contradiction that there is some path p'_{ij} shorter than p_{ij} . That is, $w(p'_{ij}) < w(p_{ij})$.

We can replace p_{ij} with p'_{ij} and obtain a new path p' from v_0 to v_k , where $w(p') < w(p)$. Hence, this contradicts with the optimality of p . ■

Estimate Distance : The D-Value

In shortest path algorithms, we typically first initialize all the **estimate distances** of all the vertices to ∞ except for the source vertex, whose estimate distance is set to 0.

That is, $d[v] = \infty$ for all $v \in V - \{s\}$ and $d[s] = 0$.

As the algorithm progresses, these d-values will gradually converge to the **actual shortest distances** $\delta(s, v)$.

Note:

- $d[v] = \infty$ and remains ∞ iff there is no path from s to v .
- $d[v]$ fails to converge iff there is a **negative-weight cycle** on a path from s to v .

Predecessor: The Pi-Value

To reconstruct a shortest path p from the source vertex s to every other vertex v , we associate each vertex with a property called the ***p-value*** denoted by $\pi[v]$ to keep track of the ***predecessor*** of v .

When we find a better path from s to v via an edge (u, v) , we update the ***pi-value*** by setting $\pi[v] = u$.

Note:

- $\pi[s] = NIL$ initially.
- $\pi[v] = NIL$ and remains NIL if there is no path from s to v .

General Procedure in SP Problems

Shortest-Path algorithms typically have the following two procedures *in common*:

- The initialization step where all the *d-values* and *pi-values* are initialized.
- The relaxation step where the *d-values* and *pi-values* are updated when a better path for each vertex is found.

Initialization

```
1: procedure INITIALIZATION( $G, s$ )  
2:   for each vertex  $v \in G.V$  do  
3:      $d[v] = \infty$   
4:      $\pi[v] = NIL$   
5:    $d[s] = 0$ 
```

The time complexity of *Initialization*(G, s) is $\Theta(V)$.

Relaxation

When a **better path** from s to v via edge (u, v) is found, we update the **d-value** and the **pi-value** of v as follows:

```
1: procedure RELAX( $u, v, w$ )
2:   if  $d[v] > d[u] + w(u, v)$  then
3:      $d[v] = d[u] + w(u, v)$ 
4:      $\pi[v] = u$ 
```

We say that the edge (u, v) is **relaxed**.

The time complexity of $Relax(u, v, w)$ is

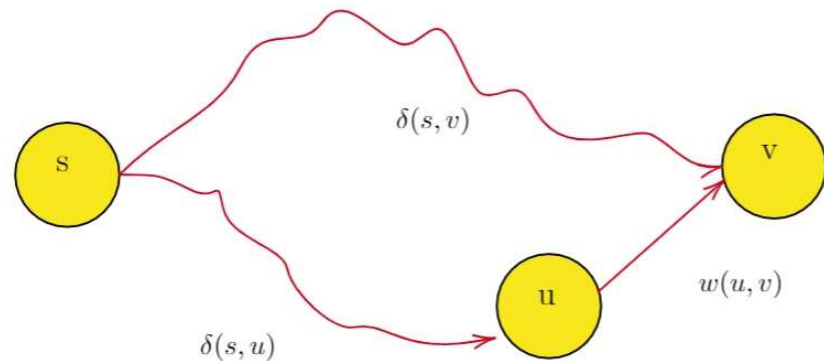
- $O(\log V)$ as the assignment in **line 3** is, in fact, a *Decrease – Key – Value* operation if a **min-priority queue** is used in **Dijkstra's algorithm**
- $\Theta(1)$ in **Bellman-Ford**

Triangle Inequality

Lemma: Triangle Inequality

Let $G = (V, E, w)$ be a weighted, directed graph with weight function $w: E \rightarrow \mathbb{R}$ and some source vertex s . Then, for all edges $(u, v) \in E$, we have

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$



Triangle Inequality

Proof:

Case I: There is at least one path from s to v .

Suppose that p is a **shortest path** from s to v . Then, p has no more weight than any other path from s to v . [Definition of a shortest path]

Therefore, p also has no more weight than a path $s \rightsquigarrow u \rightarrow v$ which consists of a shortest path from s to u followed by edge (u, v) . [$s \rightsquigarrow u \rightarrow v$ is one of the paths]

Note that the inequality still holds even though there is a **negative-weight cycle** reachable from s on the path p to v or the path $s \rightsquigarrow u \rightarrow v$.

Case II: There is no path from s to v .

Then, $\delta[v] = \infty$.

Then, there is also no path from s to u so $\delta[u] = \infty$.

If there were, we could otherwise go from s to v via $s \rightsquigarrow u \rightarrow v$, which is a **contradiction** to the assumption that there is no path from s to v .

Then, the inequality still holds. ■

Relaxation is safe

We want to show that the ***d-value*** of each vertex $v \in V$ never reduces below the actual shortest path $\delta(s, v)$.

That is, we want to show the following inequality holds ***at all times***:

$$d[v] \geq \delta(s, v) \quad \forall v \in V$$

Relaxation is safe

Lemma: Upper-Bound Property

(**Claim I**) Let $G = (V, E, w)$ be a weighted, directed graph with weight function $w: E \rightarrow \mathbb{R}$. Let $s \in V$ be the source vertex, and let G be initialized by $Initialization(G, s)$. Then, $d[v] \geq \delta(s, v) \forall v \in V$, and this invariant is maintained over any sequence of relaxation steps on the edges of G .

(**Claim II**) Moreover, once $d[v]$ achieves its lower bound $\delta(s, v)$, it never changes.

Relaxation is safe

Proof:

For Claim 1, we will show by induction on the number of relaxations steps.

Base case:

In *Initialization*(G, s),

$$d[v] = \infty \text{ for } v \in V - \{s\} \rightarrow d[v] = \infty \geq \delta(s, v).$$

and $d[s] = 0 \rightarrow d[s] = 0 \geq \delta(s, s).$

Note: $\delta(s, s) = -\infty$ when there is a **negative-weight cycle** reachable from s .
Otherwise, $\delta(s, s) = 0$.

Relaxation is safe

Proof: (Continued)

Inductive Step: Consider the relaxation of an edge $(u, v) \in E$.

By *I.H.*, we have $d[v] \geq \delta(s, v) \forall v \in V$ **just prior to the relaxation.**

The only **d-value** that may change is $d[v]$.

If it does not change, the invariant still holds by *I.H.*.

If it changes, we have

$$\begin{aligned} d[v] &= d[u] + w(u, v) \\ &\geq \delta(s, u) + w(u, v) \\ &\geq \delta(s, v) \end{aligned}$$

[**Code Inspection**]

[**I.H. : $d[u] \geq \delta(s, u)$**]

[**Triangle Inequality**]

Thus, the invariant is maintained. ■

Relaxation is safe

Proof: (Continued)

For Claim II, we will show that the ***d-value*** of any vertex $v \in V$ never changes once $d[v] = \delta(s, v)$.

Since we have just shown that the invariant $d[v] \geq \delta(s, v) \forall v \in V$ always holds, this means that when $d[v] = \delta(s, v)$, which is its lower bound, it cannot ***decrease*** any further.

And, relaxation never increases ***d-values***.

Hence, this concludes that once $d[v] = \delta(s, v)$, its value never changes. ■

Dijkstra's Algorithm

Dijkstra's algorithm solves the *single-source-shortest-path problem* on a weighted, directed graph $G = (V, E, w)$ for the case where all edges weights are *non-negative*.

Therefore, we assume that $w(u, v) \geq 0$ for all $(u, v) \in E$.

Dijkstra's Algorithm

Dijkstra's algorithm maintains a set S of vertices v whose **final d-values** have already been determined, that is, $d[v] = \delta(s, v)$.

The algorithm repeatedly selects a vertex $u \in V - S$ with the minimum **d-value**, adds u to S , and then relaxes all edges leaving u .

We can use a **min-priority queue** Q to store vertices in $V - S$, keyed by their **d-values**.

Dijkstra's Algorithm

The algorithm maintains the invariant that $Q = V - S$ at the start of each iteration of the while loop of *lines 5-9*.

```
1: procedure DJKSTRA( $G, w, s$ )
2:   INITIALIZE( $G, s$ )
3:    $S = \emptyset$ 
4:    $Q = G.V$ 
5:   while  $Q \neq \emptyset$  do
6:      $u = \text{EXTRACT-MIN}(Q)$ 
7:      $S = S \cup \{u\}$ 
8:     for each vertex  $v \in G.Adj[u]$  do
9:       RELAX( $u, v, w$ )
```

Dijkstra's Algorithm : Analysis

Claim: Dijkstra's algorithm takes $O((V + E) \log V)$ using a min-priority queue.

Proof:

Initialization(G, s) takes $\Theta(V)$ time.

Extract – Min(Q) runs $\Theta(V)$ times, each of which takes **at most** $O(\log V)$ time.

- In total, *Extract – Min*(Q) takes **at most** $O(V \log V)$ time.

Relax(u, v, w) runs exactly E times, each of which takes $O(\log V)$ time.

- In total, *Relax*(u, v, w) takes $O(E \log V)$ time.

Summing up, the total running time of Dijkstra's algorithm is **at most**

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



No-Path Property

Lemma: (No-Path Property)

Suppose that in a weighted, directed graph $G = (V, E, w)$ with weight function $w: E \rightarrow \mathbb{R}$, no path connects a source vertex $s \in V$ to a given vertex $v \in V$. Then, after the graph is initialized by *Initialization*(G, s), we have $d[v] = \delta(s, v) = \infty$, and this equality is maintained as an invariant over any sequence of relaxation steps on the edges of G .

Proof: By the upper-bound property, we always have $\infty = \delta(s, v) \leq d[v]$.

Thus, $\delta(s, v) = d[v]$. ■

Edge-Relaxation Property

Lemma: (Edge-Relaxation Property)

Let $G = (V, E, w)$ be a weighted, directed graph with weight function $w: E \rightarrow \mathbb{R}$ and let $(u, v) \in E$. Then, immediately after relaxing edge (u, v) by executing $\text{Relax}(u, v, w)$, we have $d[v] \leq d[u] + w(u, v)$.

Proof:

If, just prior to relaxing edge (u, v) , we have $d[v] > d[u] + w(u, v)$, then $d[v] = d[u] + w(u, v)$ afterwards.

If, instead, $d[v] \leq d[u] + w(u, v)$ just prior to relaxation, then neither $d[v]$ nor $d[u]$ changes, and so $d[v] \leq d[u] + w(u, v)$ afterwards. ■

Convergence Property

Lemma: (Convergence Property)

Let $G = (V, E, w)$ be a weighted, directed graph with weight function $w: E \rightarrow \mathbb{R}$, let $s \in V$ be a source vertex, and let $s \rightsquigarrow u \rightarrow v$ be a shortest path in G for some vertices $u, v \in V$.

Suppose that G is initialized by $Initialization(G, s)$ and then a sequence of relaxation steps that include the call $Relax(u, v, w)$ is executed on the edges of G .

If $d[u] = \delta(s, u)$ at any time prior to the call, then $d[v] = \delta(s, v)$ at all times after the call.

Convergence Property

Proof: By the **Upper-Bound Property**, if $d[u] = \delta(s, u)$ at some point prior to the relaxation of (u, v) , then this equality holds thereafter.

In particular , after relaxing (u, v) , we have

$$\begin{aligned} d[v] &\leq d[u] + w(u, v) && [\text{Edge-Relaxation Property}] \\ &= \delta(s, u) + w(u, v) && [d[u] = \delta(s, u)] \\ & && [s \rightsquigarrow u \rightarrow v \text{ is a shortest path in } G] \\ &= \delta(s, v) && [\text{Optimal Substructure Property}] \end{aligned}$$

Invoking the **Upper-Bound Property**, which requires that $d[v] \geq \delta(s, v)$, we have $d[v] = \delta(s, v)$ and this equality holds thereafter by the **Upper-Bound Property**. ■

Dijkstra's Algorithm : Correctness

Theorem: (Correctness of Dijkstra's Algorithm)

Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E, w)$ with non-negative weight function w and source vertex s , terminates with $d[u] = \delta(s, u)$ for all vertices $u \in V$.

Proof:

We will prove correctness by showing that the following **loop invariant** holds:

Prior to each iteration of the while loop, $d[v] = \delta(s, v)$ for each vertex $v \in S$.

Dijkstra's Algorithm : Correctness

Proof: (Continued)

It suffices to show that for each vertex $u \in V$, we have $d[u] = \delta(s, u)$ at the time when u is added to S . Once we show that $d[u] = \delta(s, u)$, we can invoke the **Upper-Bound Property** to show that the equality $d[u] = \delta(s, u)$ holds **at all times thereafter**.

Initialization: Initially, $S = \emptyset$. Therefore, the invariant vacuously holds.

Dijkstra's Algorithm : Correctness

Proof: (Continued)

Maintenance:

We will show that in each iteration $d[u] = \delta(s, u)$ for the vertex added to S .

Assume for the purpose of contradiction that u is the **first** vertex for which $d[u] \neq \delta(s, u)$ when added to S .
---(1)

We must have that $u \neq s$ because we can be certain that $\delta(s, s) = d[s] = 0$ at the time it is added to S .

Because $u \neq s$, we know that $S \neq \emptyset$ at the time when u is added to S .

Thus, there must be some path from s to u . Otherwise, $d[u] = \delta(s, u) = \infty$ by the **No-Path Property**, which would violate the assumption (1) that $d[u] \neq \delta(s, u)$.

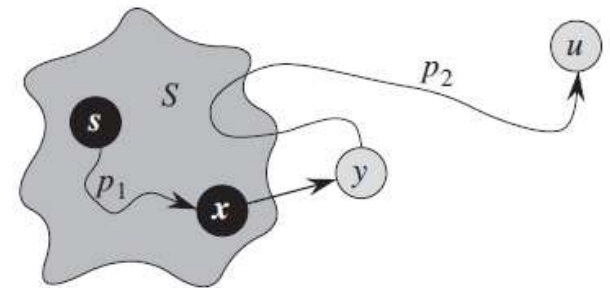
Dijkstra's Algorithm : Correctness

Proof: (Continued)

Maintenance: Because there is at least one path, there must be a shortest path p from s to u .

Prior to adding u to S , path p connects a vertex in S , namely s , to a vertex in $V - S$, namely, u .

Let us consider the first vertex y along p such that $y \in V - S$ and $x \in S$ be the immediate predecessor of y along p .



Dijkstra's Algorithm : Correctness

Proof: (Continued)

We can break down path p into $s \rightsquigarrow x \rightarrow y \rightsquigarrow u$, where $p_1 = s \rightsquigarrow x$ and $p_2 = y \rightsquigarrow u$. (Either p_1 or p_2 may contain no edges.)

[**Claim I**] We **claim** that $d[y] = \delta(s, y)$ when u is added to S .

To prove **Claim I**, notice that $x \in S$.

Recall that we chose u such that it is the **first** vertex for which $d[u] \neq \delta(s, u)$ when it is added to S .

Thus, we had $d[x] = \delta(s, x)$ when x was added to S .

Edge (x, y) was relaxed at the time, and the claim follows from the **Convergence Property**.

Dijkstra's Algorithm : Correctness

Proof: (Continued)

Because y appears before u on a shortest path from s to u and all edge weights are **non-negative** (notably those on path p_2), we have

$$\delta(s, y) \leq \delta(s, u) \quad [\text{Monotonicity}]$$

and thus

$$\begin{aligned} d[y] &= \delta(s, y) && [\text{Claim 1}] \\ &\leq \delta(s, u) && [\text{Monotonicity}] \\ &\leq d[u] && [\text{Upper-Bound Property}] \end{aligned} \quad \text{---(1)}$$

Dijkstra's Algorithm : Correctness

Proof: (Continued)

But because both vertices u and y were in $V - S$ when u was chosen in **line 6** ($u = \text{Extract} - \text{Min}(Q)$), we have

[Min-Priority Queue implies Greedy Choice]

$$d[u] \leq d[y] \quad \text{---(2)}$$

By **(1)** & **(2)**, we have $d[y] = \delta(s, y) = \delta(s, u) = d[u]$.

Consequently, $\delta(s, u) = d[u]$, which contradicts our choice of u .

We can conclude that $d[u] = \delta(s, u)$ when u was added to S , and this equality is maintained at all times thereafter.

Dijkstra's Algorithm : Correctness

Proof: (Continued)

Termination: At termination, we have $Q = \emptyset$, which means that $V - S = \emptyset$, implying that $V = S$.

Plugging $V = S$ into the loop invariant, we have:

$$d[v] = \delta(s, v) \text{ for each vertex } v \in V$$

,which proves the correctness of Dijkstra's algorithm. ■

Bellman-Ford

The **Bellman-Ford** algorithm solves the single-source shortest-path problem in the general case where edge weights may be **negative**.

Given a weighted, directed graph $G = (V, E, w)$ with a source s and weight function $w: E \rightarrow \mathbb{R}$, Bellman-Ford **returns a Boolean value** indicating whether or not there is a **negative-weight cycle** reachable from s .

If there is such a cycle, the algorithm reports that **no solution exists**. Otherwise, it produces **shortest paths** and **their weights** for all the vertices $v \in V$.

Bellman-Ford

The algorithm proceeds by relaxing edges, hence progressively decreasing the ***d-value*** of each vertex $v \in V$ until it achieves the actual shortest-path values $\delta(s, v)$.

```
1: procedure BELLMAN-FORD( $G, w, s$ )
2:   INITIALIZE( $G, s$ )
3:   for  $i = 1 \rightarrow |G.V| - 1$  do
4:     for each edge  $(u, v) \in G.E$  do
5:       RELAX( $u, v, w$ )
6:   for each edge  $(u, v) \in G.E$  do
7:     if  $d[v] > d[u] + w(u, v)$  then
8:       return FALSE
9:   return TRUE
```

Bellman-Ford

The **Bellman-Ford** algorithm proceeds as follows:

It first initializes the **d-value** and the **pi-value** of each vertex $v \in V$ by calling *Initialization*(G, s).

The algorithm then makes exactly $|V| - 1$ passes over the edges of G .

Each pass consists of relaxing each edge of the graph once.

After making $|V| - 1$ passes, the algorithm checks for a **negative-weight cycle** by making **one extra pass** over the edges of G and returns the appropriate Boolean value.

Bellman-Ford: Analysis

Claim: Bellman-Ford takes $\Theta(V E)$ time.

Proof:

Initialization(G, s) takes $\Theta(V)$ time.

Each pass takes $\Theta(E)$ time.

- In total, there are $|V| - 1$ passes so it takes $\Theta(V E)$ time.

The final extra pass takes $\Theta(E)$ time.

Summing up all the contributions, the running time of Bellman-Ford is

$$\Theta(V) + \Theta(V E) + \Theta(E) = \Theta(V E). \blacksquare$$

Path-Relaxation Property

Lemma: (Path-Relaxation Property)

Let $G = (V, E, w)$ be a weighted, directed graph with a source s and weight function $w: E \rightarrow \mathbb{R}$. Consider any shortest path $p = \langle v_0, v_1, \dots, v_k \rangle$ from $s = v_0$ to v_k . If G is initialized with $\text{Initialization}(G, s)$ and then a sequence of relaxation steps occurs that includes, in order, relaxing the edges $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$, then $d[v_k] = \delta(s, v_k)$ after these relaxations and at all times afterward.

This property holds no matter what other edge relaxations occur, including relaxations that are intermixed with relaxations of the edges of p .

Path-Relaxation Property

Proof: We will show by induction that after the i^{th} edge of p is relaxed, we have $d[v_i] = \delta(s, v_i)$.

Base Case: $i = 0$

Before any edge of p is relaxed, we have $d[v_0] = d[s] = 0 = \delta(s, s)$. $d[s]$ never changes by the **Upper-Bound Property**.

Note: $\delta(s, s) = 0$ because p is assumed to be a shortest path so p cannot contain a negative-weight cycle reachable from s .

Induction Hypothesis: Assume that $d[v_{i-1}] = \delta(s, v_{i-1})$.

Inductive Step: We shall investigate what happens when (v_{i-1}, v_i) is relaxed.

By the **Convergence Property**, after relaxing this edge, we have $d[v_i] = \delta(s, v_i)$ and this equality holds at all times thereafter. ■

Bellman-Ford: Correctness

Lemma 1:

Let $G = (V, E, w)$ be a weighted, directed graph with a source s and weight function $w: E \rightarrow \mathbb{R}$ and assume that G contains **no negative-weight cycles** reachable from s . Then, after $|V| - 1$ iterations, we have $d[v] = \delta(s, v)$ for all vertices v that are reachable from s .

Proof: Consider any vertex v that is reachable from s , and let $p = \langle v_0, v_1, \dots, v_k \rangle$, where $v_0 = s$ and $v_k = v$, be any shortest path from s to v . Because shortest paths are **simple**, p has **at most** $|V| - 1$ edges, and so $k \leq |V| - 1$.

Each of the $|V| - 1$ iterations relaxes all $|E|$ edges. Among the edges relaxed in the i^{th} iteration, for $i = 1, 2, \dots, k$, is (v_{i-1}, v_i) .

By the **Path-Relaxation Property**, $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$. ■

Bellman-Ford: Correctness

Lemma II: Let $G = (V, E, w)$ be a weighted, directed graph with a source s and weight function $w: E \rightarrow \mathbb{R}$. Then, for each vertex $v \in V$, there is a path from s to v if and only if the algorithm terminates with $d[v] < \infty$.

Proof:

\Rightarrow : If there is a path from s to v , then the algorithm terminates with $d[v] < \infty$.

Base Case: Initially, $V_\pi = \{s\}$. There is a trivial path of weight 0 from s to itself and we set $d[s] = 0 < \infty$.

Observe that $d[s] < \infty$ at all times thereafter because relaxation **never increases** d-values.

Induction Hypothesis: Assume true for any k^{th} relaxation.

Bellman-Ford: Correctness

Proof: (Continued)

Inductive Step: Suppose we are relaxing an edge $(u, v) \in E$.

Case I: $u \in V_\pi$: there is a path from s to u . Then $d[u] < \infty$ by **I.H.**.

If $d[v] > d[u] + w(u, v)$, then

$$\pi[v] = u \text{ and } d[v] = d[u] + w(u, v)$$

Thus, there is a path from s to v via u and $d[v] < \infty$.

If $d[v] \leq d[u] + w(u, v)$, then nothing changes so I.H. is reestablished by the I.H. from the previous relaxation step.

We have $v \in V_\pi$.

Bellman-Ford: Correctness

Proof: (Continued)

Inductive Step: Suppose we are relaxing an edge $(u, v) \in E$.

Case II: There is no path from s to u .

Thus, $d[u] = \infty = \delta(s, u)$ *by the **No-Path Property**.*

Since $d[v] > d[u] + w(u, v)$ does not hold, nothing changes so I.H. is reestablished by the I.H. from the previous relaxation step.

Bellman-Ford: Correctness

Proof: (Continued)

\Leftarrow : If the algorithm terminates with $d[v] < \infty$, there is a path from s to v .

Base Case: Initially, $V_\pi = \{s\}$.

We set $d[s] = 0 < \infty$.

There is a path of weight 0 from s to itself.

Observe that $d[s] < \infty$ at all times thereafter because relaxation ***never increases*** d-values.

Induction Hypothesis: Assume true for any k^{th} relaxation.

Bellman-Ford: Correctness

Inductive Step: Suppose we are relaxing an edge $(u, v) \in E$.

Case I: $d[v] > d[u] + w(u, v)$

Thus, $d[u]$ must be a finite value so $d[u] < \infty$.

Then, there is a path from s to u by ***I.H.***

Therefore, $d[v] = d[u] + w(u, v)$ and $\pi[v] = u$.

This establishes a path from s to v via u and $d[v]$ is now a finite value so $d[v] < \infty$.

Bellman-Ford: Correctness

Inductive Step: Suppose we are relaxing an edge $(u, v) \in E$.

Case II: $d[v] \leq d[u] + w(u, v)$

Thus, nothing changes so I.H. is reestablished by the I.H. from the previous relaxation step.

Therefore, we have proved the invariant:

there is a path from s to v if and only if $d[v] < \infty$.

Termination: If $d[v] < \infty$ just after $|V| - 1$ iterations, then , $d[v] < \infty$ remains true thereafter because relaxation ***never increases d-values***. ■

Bellman-Ford: Correctness

Theorem: Let Bellman-Ford be run on a weighted, directed graph $G = (V, E, w)$ with source s and weight function $w: E \rightarrow \mathbb{R}$.

(**Claim I**) If G contains **no negative-weight cycles** that are reachable from s , the algorithm returns **TRUE**, we have $d[v] = \delta(s, v)$ for all vertices $v \in V$.

(**Claim II**) If G does contain a **negative-weight cycle** reachable from s then the algorithm returns **FALSE**.

Bellman-Ford: Correctness

Proof: (Claim I)

Suppose G contains no negative-weight cycles that are reachable from s .

(**Claim III**) We first prove the claim that $d[v] = \delta(s, v)$ for all vertices $v \in V$.

By **Lemma I**, we prove **Claim III** for those vertices v reachable from s .

By the **No-Path Property**, we prove **Claim III** for those vertices v not reachable from s .

Bellman-Ford: Correctness

Proof: (Claim I)

At termination, for all edges $(u, v) \in E$, we have

$$\begin{aligned} d[v] &= \delta(s, v) && [\text{Claim III: } d[v] = \delta(s, v)] \\ &\leq \delta(s, u) + w(u, v) && [\text{Triangle Inequality}] \\ &= d[u] + w(u, v) && [\text{Claim III: } d[u] = \delta(s, u)] \end{aligned}$$

Therefore, we have $d[v] \leq d[u] + w(u, v)$ so it does not pass the **if condition** in the extra pass. Therefore, the algorithm returns **TRUE**. ■

Bellman-Ford: Correctness

Proof: (Claim II)

Suppose that G contains a **negative-weight cycle** reachable from s and let this cycle be $c = \langle v_0, v_1, \dots, v_k \rangle$, where $v_0 = v_k$.

Then, we have the sum of all the edge weights in this cycle

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0. \quad \text{---(1)}$$

Assume for the purpose of contradiction that the algorithm return **TRUE**.

Thus, we must have

$$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i) \text{ for } i = 1, 2, \dots, k. \quad \text{---(2)}$$

Bellman-Ford: Correctness

Proof: (Claim II)

$$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i) \text{ for } i = 1, 2, \dots, k. \quad \text{---(2)}$$

Summing **Eq.(2)** around the cycle c , we have

$$\sum_{i=1}^k d[v_i] \leq \sum_{i=1}^k d[v_{i-1}] + \sum_{i=1}^k w(v_{i-1}, v_i) \quad \text{---(3)}$$

Observe since $v_0 = v_k$, each vertex appears in c exactly once in each of the summations $\sum_{i=1}^k d[v_i]$ and $\sum_{i=1}^k d[v_{i-1}]$.

Thus, $\sum_{i=1}^k d[v_i] = \sum_{i=1}^k d[v_{i-1}]$.

We can rewrite **Eq.(3)** as

$$\sum_{i=1}^k d[v_i] \leq \sum_{i=1}^k d[v_i] + \sum_{i=1}^k w(v_{i-1}, v_i) \quad \text{---(4)}$$

Bellman-Ford: Correctness

Proof: (Claim II)

By ***Lemma II***, $d[v_i] < \infty$, i.e., ***the d-value is finite***,

The terms $\sum_{i=1}^k d[v_i]$ on both sides legitimately cancel out and we have

$$0 \leq \sum_{i=1}^k w(v_{i-1}, v_i)$$

,which contradicts our assumption in ***Eq.(1)***.

Hence, the algorithm must return ***FALSE*** in the ***presence of a negative-weight cycle*** that is reachable from s . ■

Thus, we can conclude that Bellman-Ford returns ***TRUE*** if G contains ***no negative-weight cycles*** reachable from s .

Otherwise, it returns ***FALSE***. ■

Summary

In this lecture, we have covered the topic of single-source shortest path problems:

- Dijkstra's Single-Source Shortest Path
- Bellman-Ford

In the next lecture, we will cover more on *shortest path problems*.