Lecture 14

Dijkstra's Algorithm

Source: CLRS and Slides of Prof. Surender Baswana

Weighted Graphs

Weighted Graphs

Defn: A weighted graph $G = (V, E)$ is graph with a weight function $w : E \to \mathbb{R}$ mapping edges

Weighted Graphs

Defn: A weighted graph $G = (V, E)$ is graph with a weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights.

Weighted Graphs

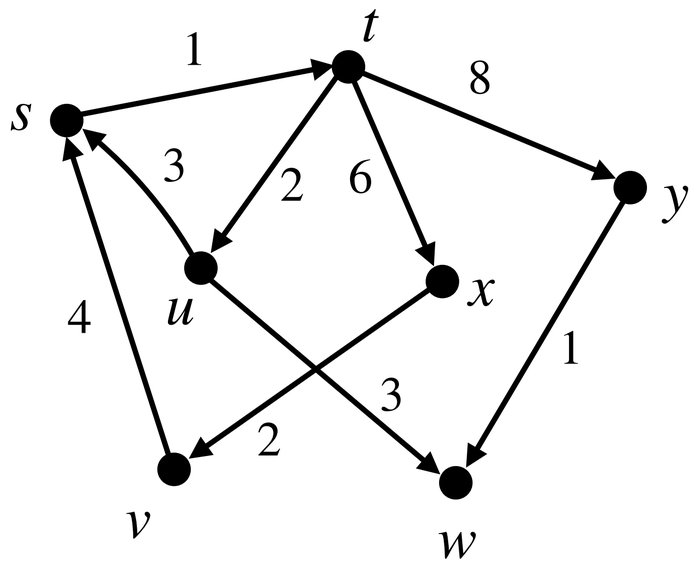
Defn: A weighted graph $G = (V, E)$ is graph with a weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights.

Example:

Weighted Graphs

Defn: A weighted graph $G = (V, E)$ is graph with a weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights.

Example:



Weighted Graphs

Weighted Graphs

Defn: Weight of a path is the sum of the weights of the edges in that path.

Weighted Graphs

Defn: Weight of a path is the sum of the weights of the edges in that path. In a weighted graph,

Weighted Graphs

Defn: Weight of a path is the sum of the weights of the edges in that path. In a weighted graph, a shortest-path between two vertices is the path with the least weight.

Weighted Graphs

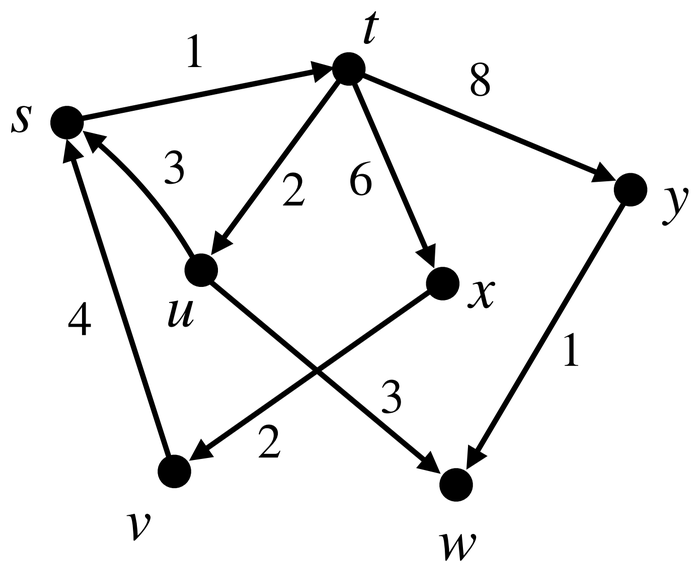
Defn: Weight of a path is the sum of the weights of the edges in that path. In a weighted graph, a shortest-path between two vertices is the path with the least weight.

Example:

Weighted Graphs

Defn: Weight of a path is the sum of the weights of the edges in that path. In a weighted graph, a shortest-path between two vertices is the path with the least weight.

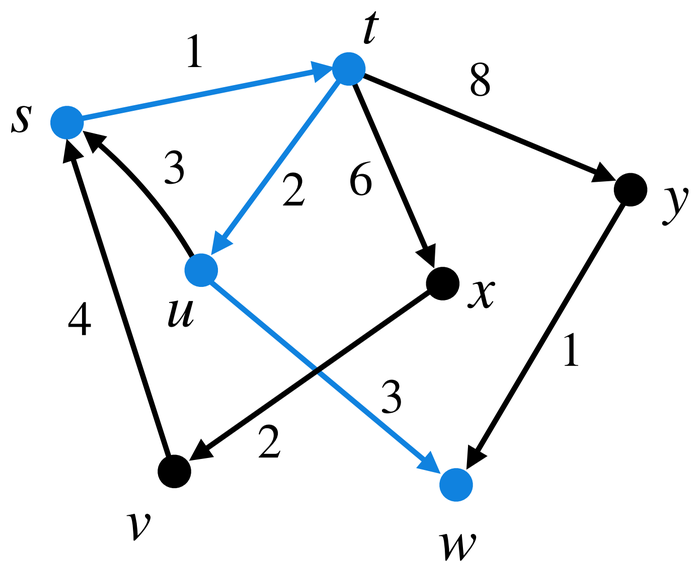
Example:



Weighted Graphs

Defn: Weight of a path is the sum of the weights of the edges in that path. In a weighted graph, a shortest-path between two vertices is the path with the least weight.

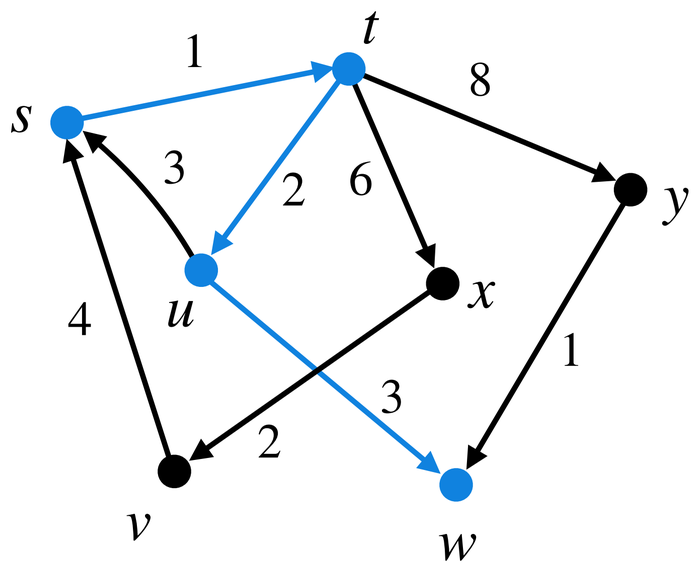
Example:



Weighted Graphs

Defn: Weight of a path is the sum of the weights of the edges in that path. In a weighted graph, a shortest-path between two vertices is the path with the least weight.

Example: In the below graph shortest path from s to w is of weight 6.



Single-Source Shortest Path Problem

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

SSSP:

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

SSSP:

Input: A weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and a vertex s.

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

SSSP:

Input: A weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and a vertex s.

Output: Compute distance from s to every other vertex. That is, an array $d[1 : |V|]$,

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

SSSP:

Input: A weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and a vertex s.

Output: Compute distance from s to every other vertex. That is, an array $d[1 : |V|]$, such that $d[i] = \delta(s, i)$.

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

SSSP:

Input: A weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and a vertex s.

Output: Compute distance from s to every other vertex. That is, an array $d[1 : |V|]$, such that $d[i] = \delta(s, i)$.

- Dijkstra's algorithm solves *SSSP* in graphs with non-negative weights.

Single-Source Shortest Path Problem

Let $\delta(s, v)$ denote the weight of a shortest path from s to v.

If v is not reachable (or connected) from s, then $\delta(s, v) = \infty$.

SSSP:

Input: A weighted graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and a vertex s.

Output: Compute distance from s to every other vertex. That is, an array $d[1 : |V|]$, such that $d[i] = \delta(s, i)$.

- Dijkstra's algorithm solves *SSSP* in graphs with non-negative weights.
- Bellman-Ford's algorithm solves *SSSP* in graphs with real weights.

Optimal Subpath Property

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch:

Optimal Subpath Property

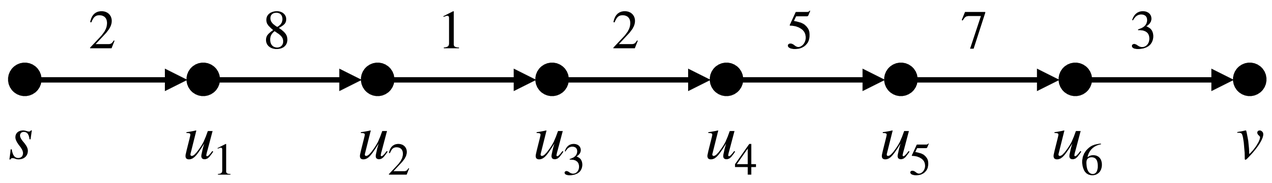
Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

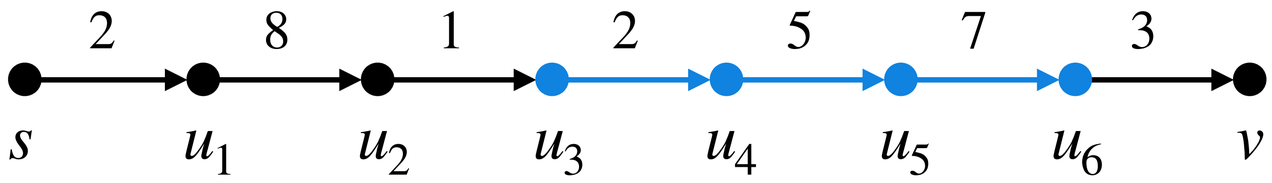
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

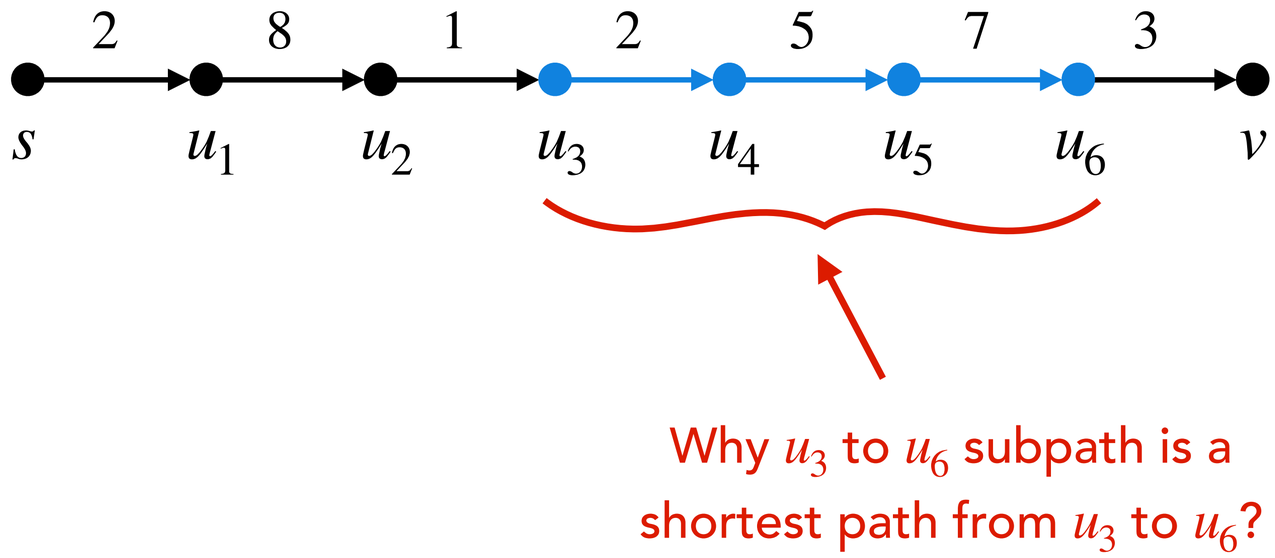
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

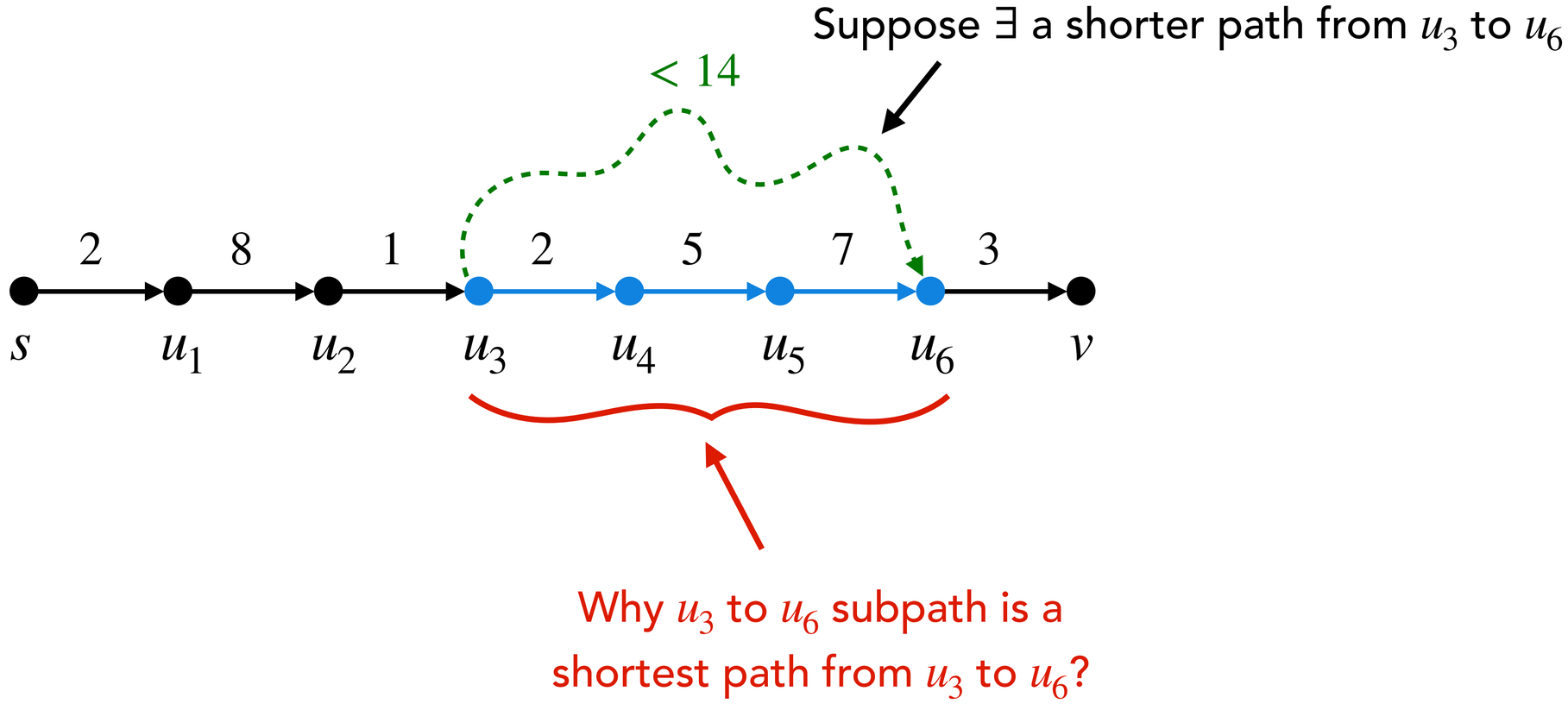
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

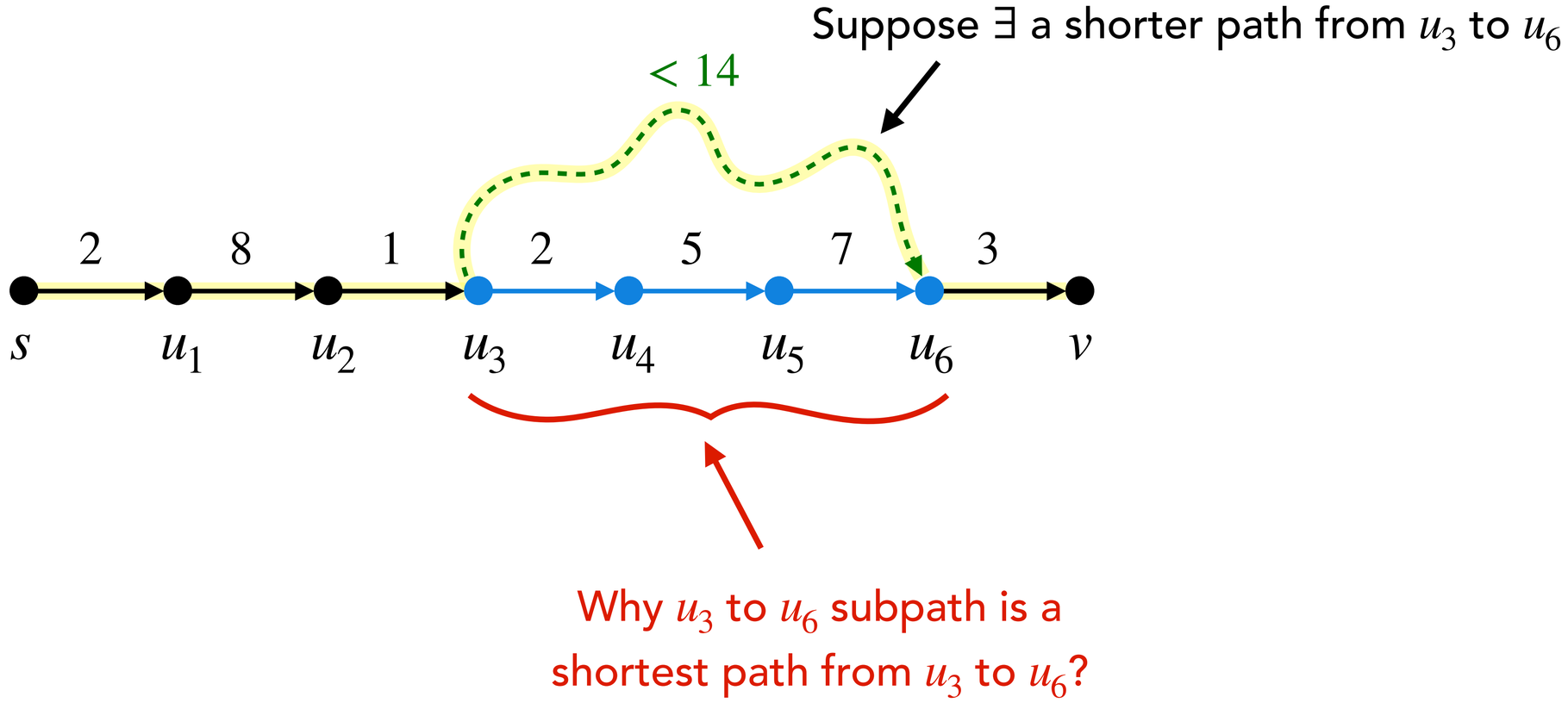
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

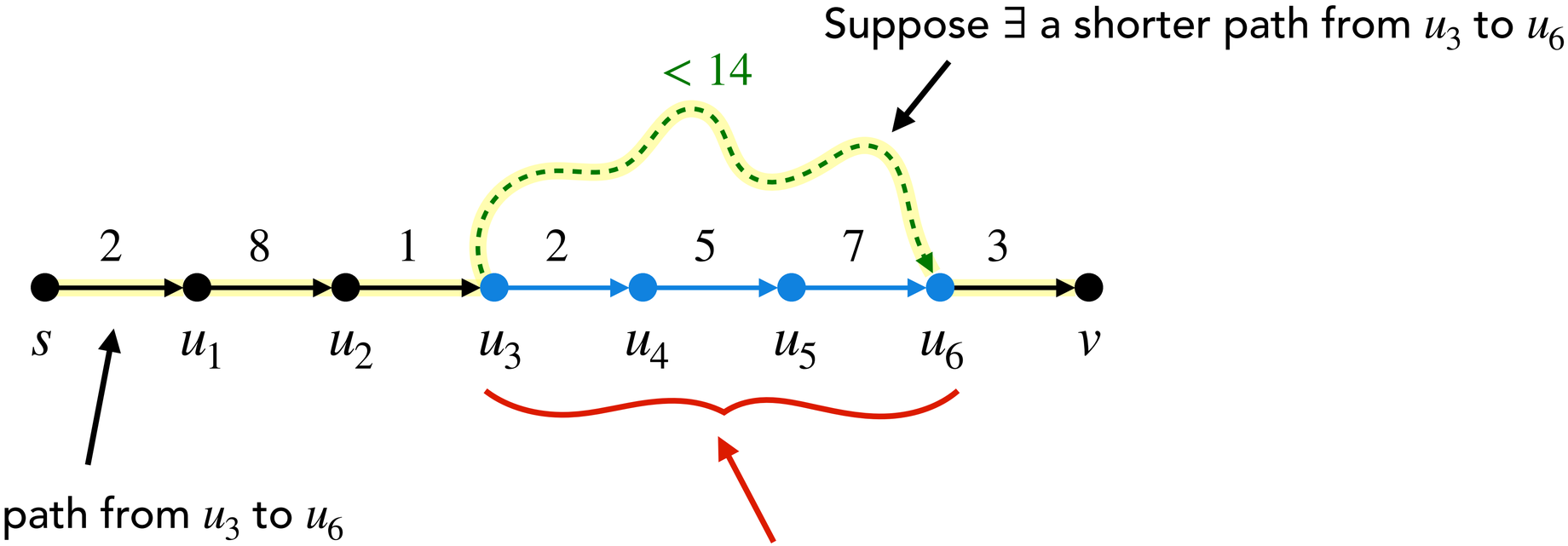
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.



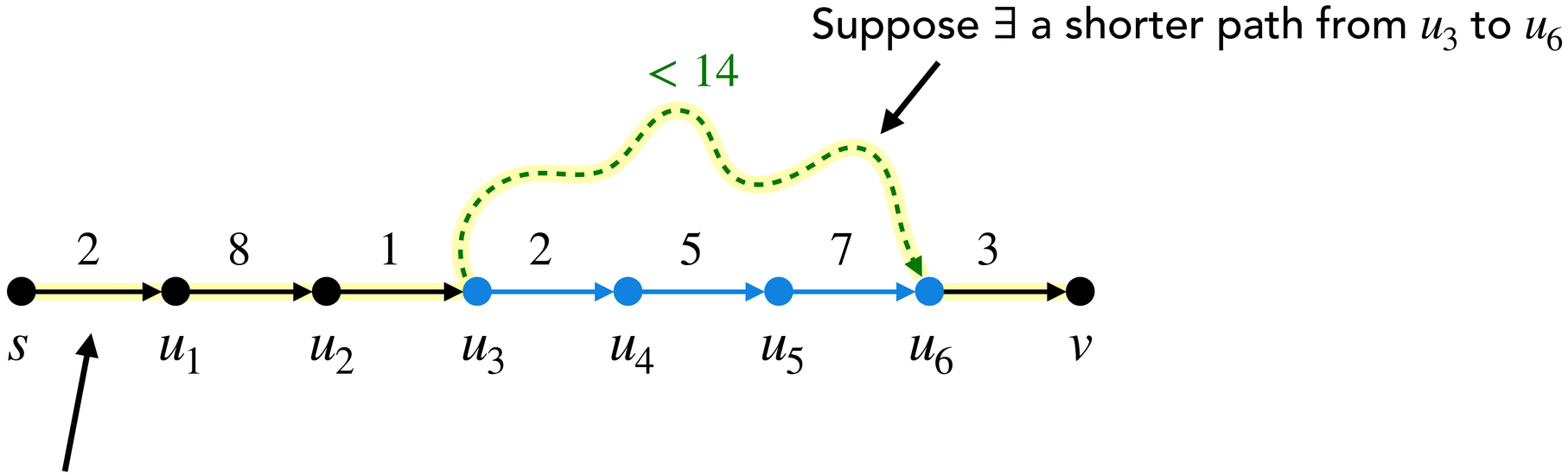
Then using shorter path from u_3 to u_6
we can get a shorter path than P,
which is a contradiction.

Why u_3 to u_6 subpath is a
shortest path from u_3 to u_6?

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.



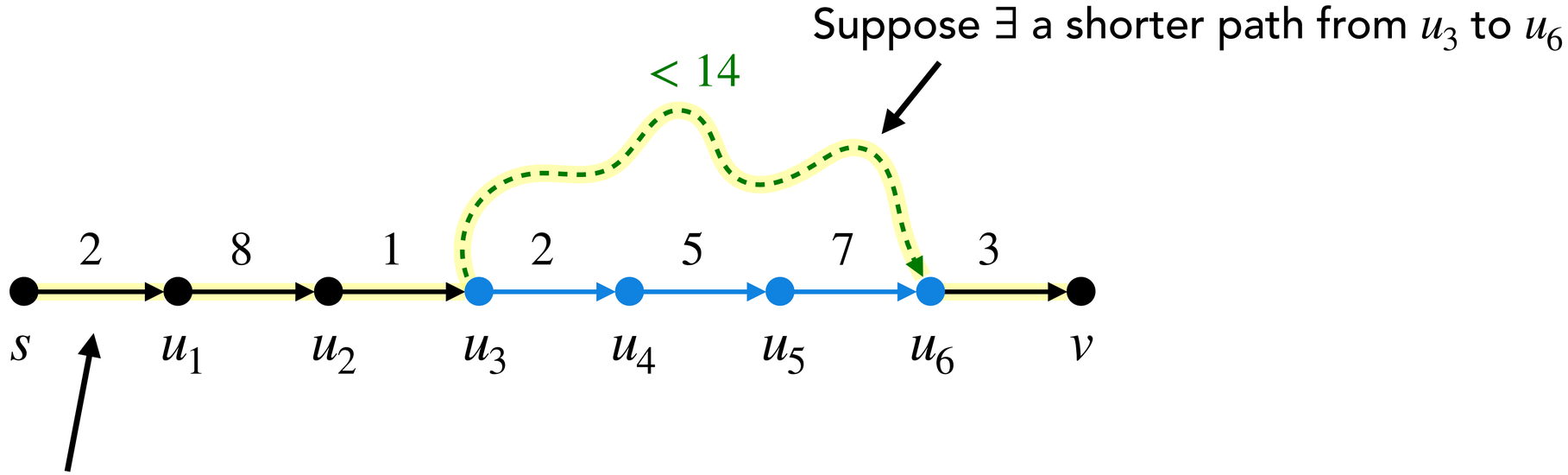
Then using shorter path from u_3 to u_6
we can get a shorter path than P,
which is a contradiction.

Is the proof correct?

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.



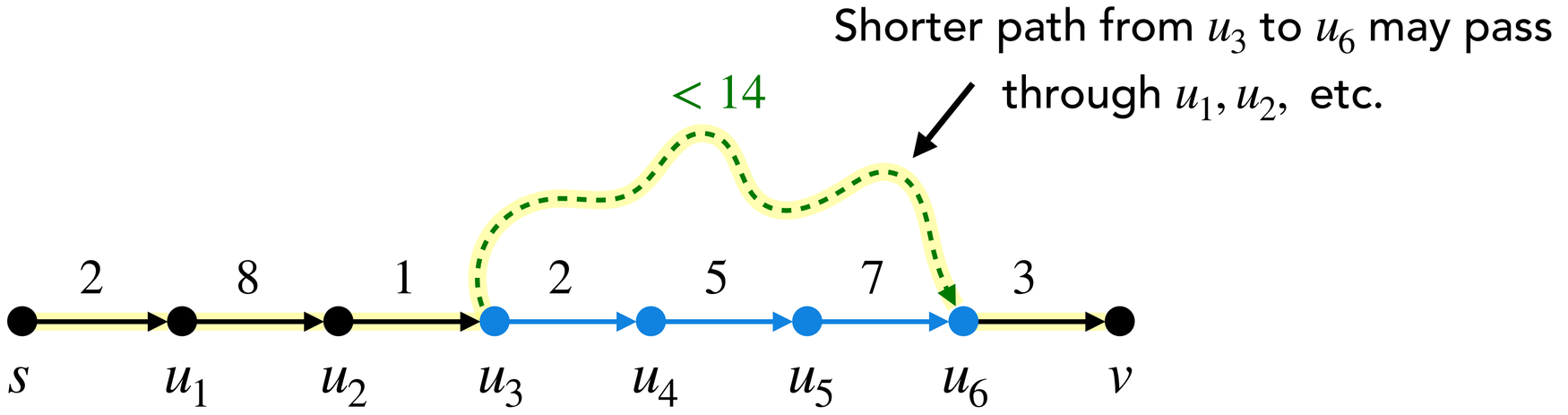
Then using shorter path from u_3 to u_6
we can get a shorter path than P,
which is a contradiction.

Is the proof correct? No.

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

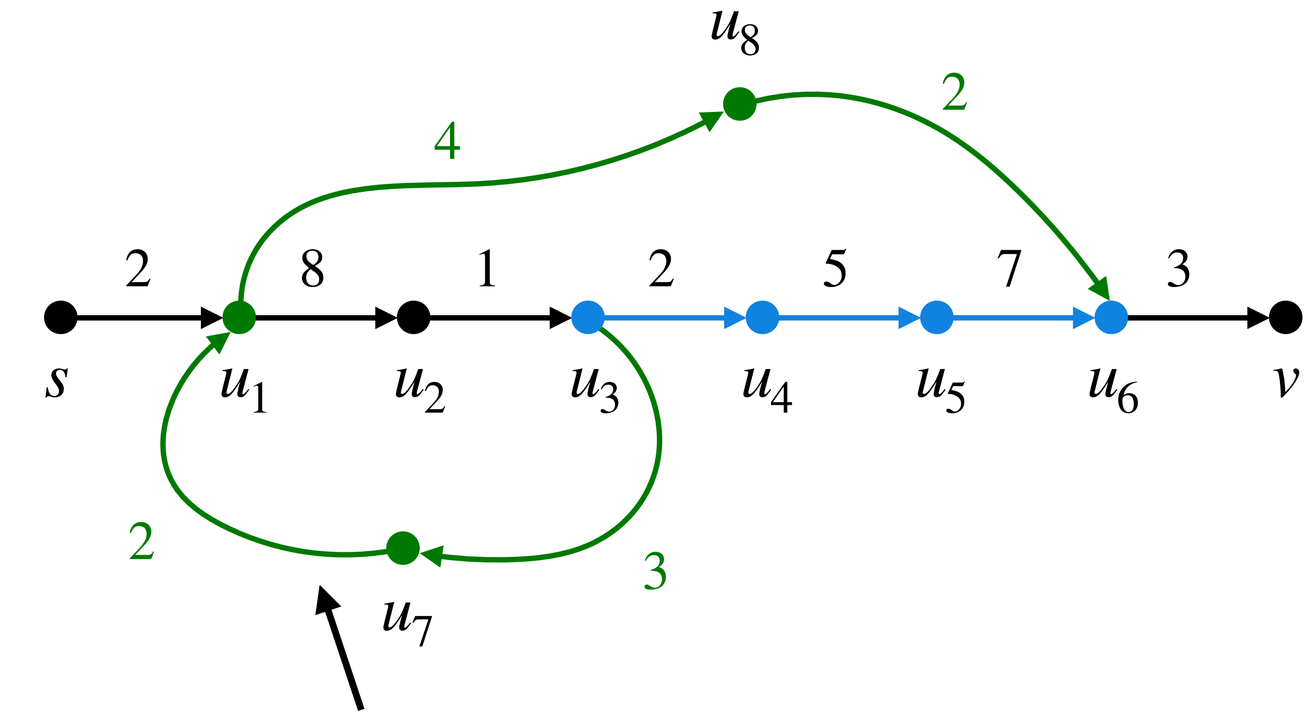
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.

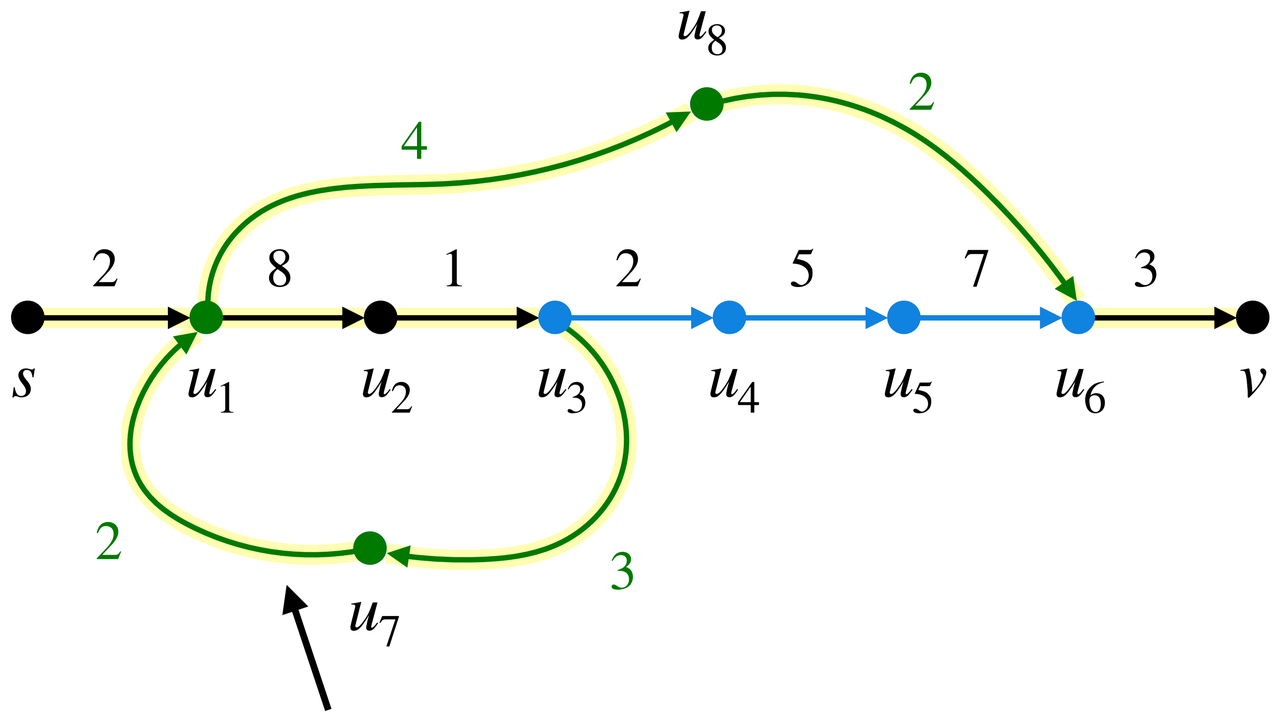


Shorter path from u_3 to u_6 going via u_1.

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

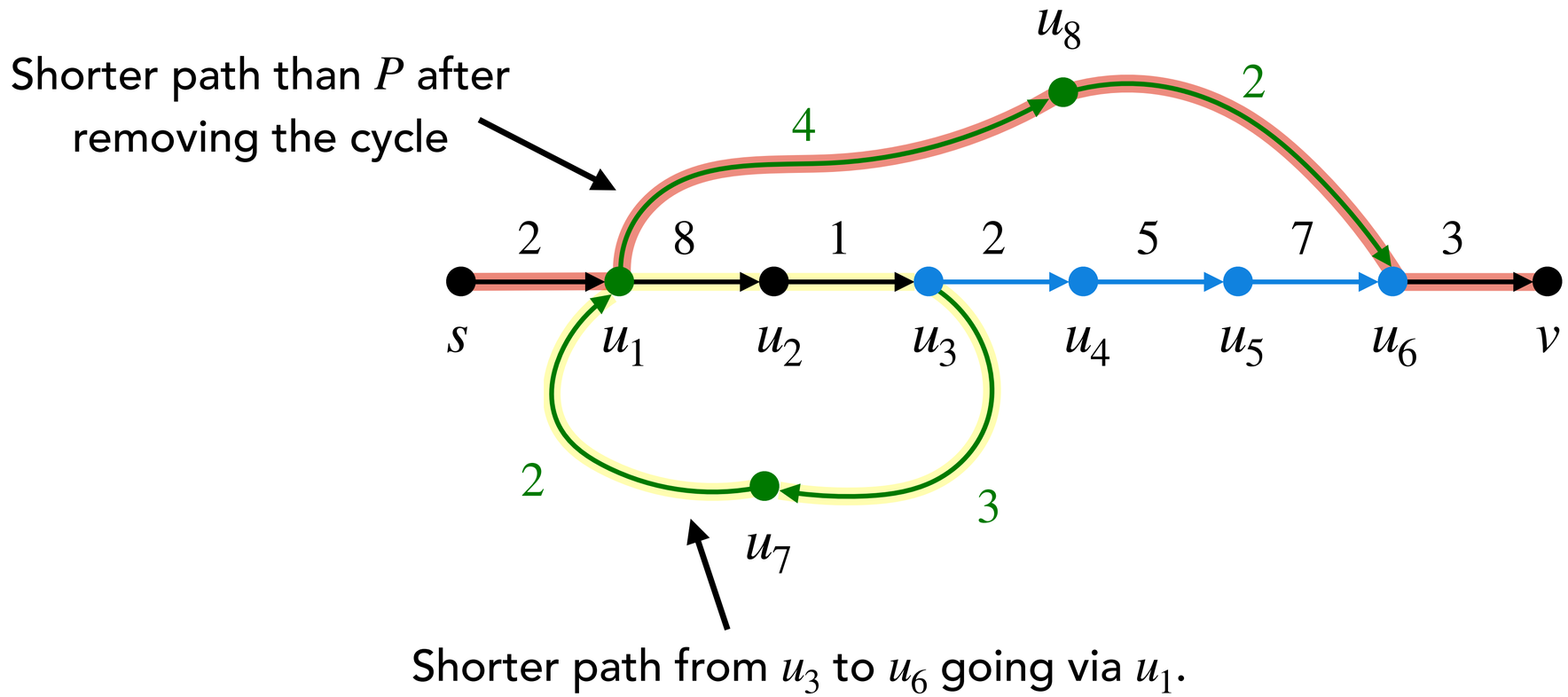
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

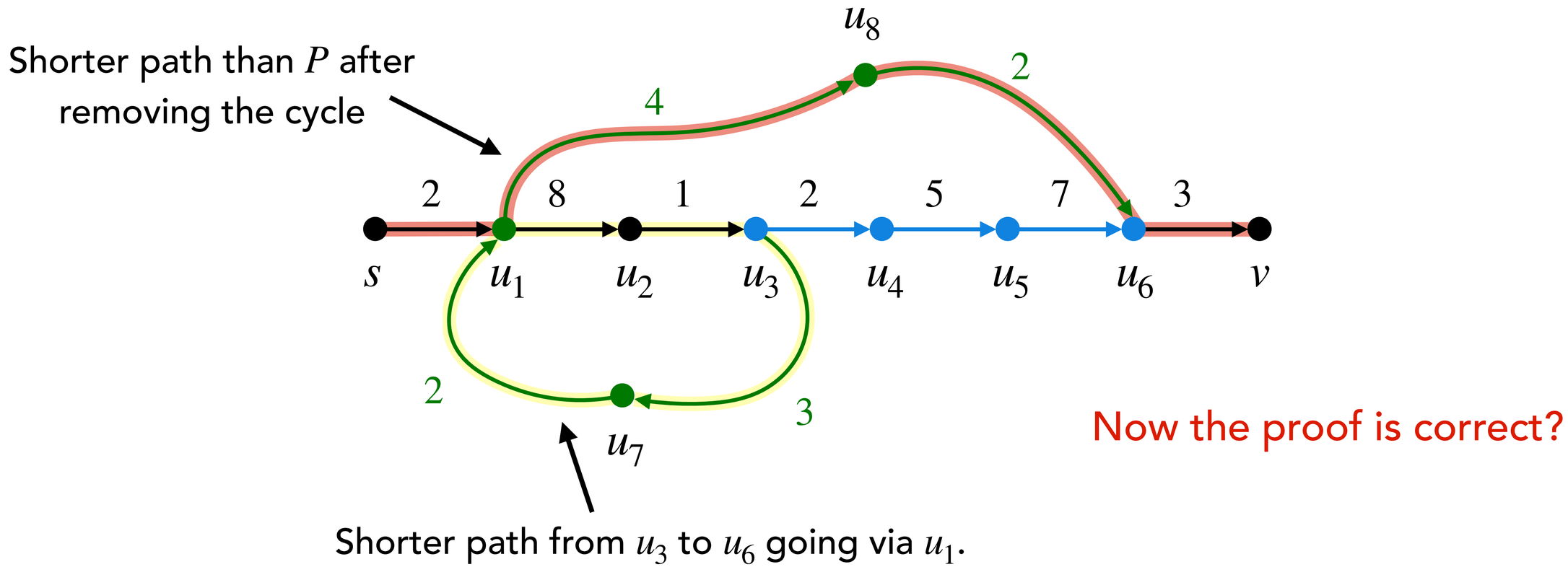
Proof Sketch: Let P be a shortest path from s to v.



Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.

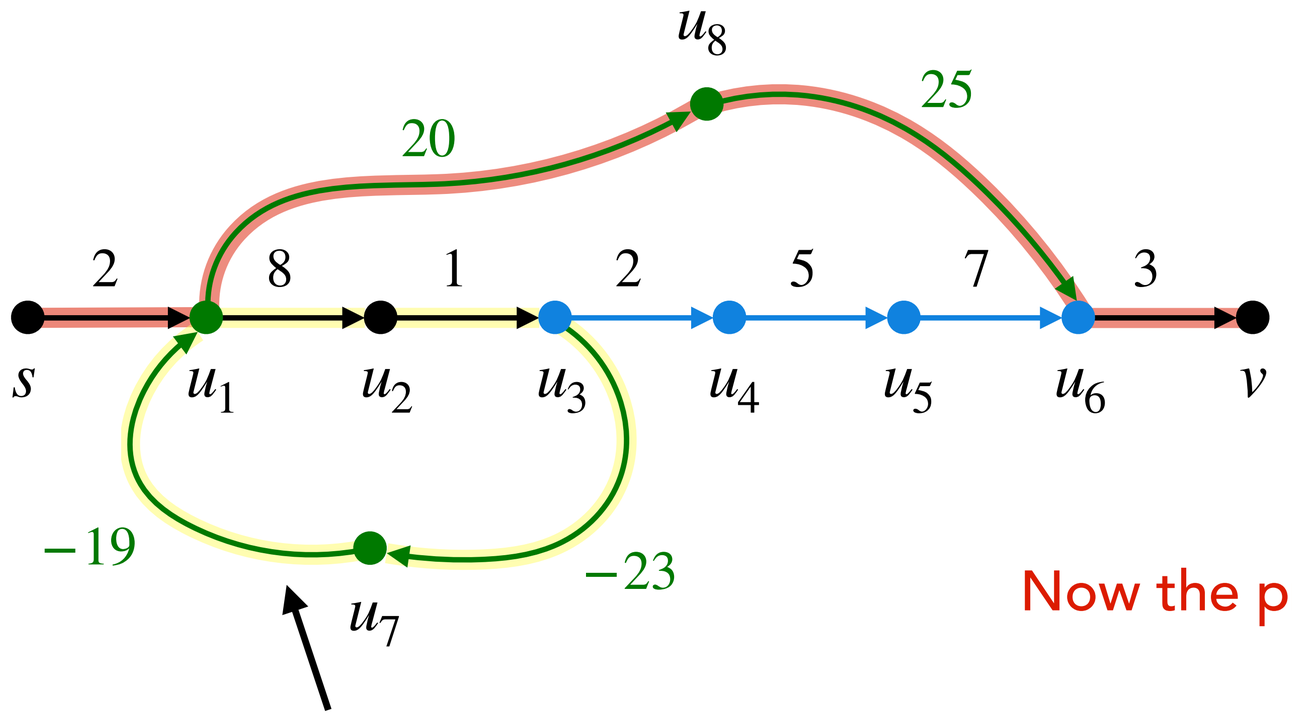


Now the proof is correct?

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.



Shorter path from u_3 to u_6 going via u_1.

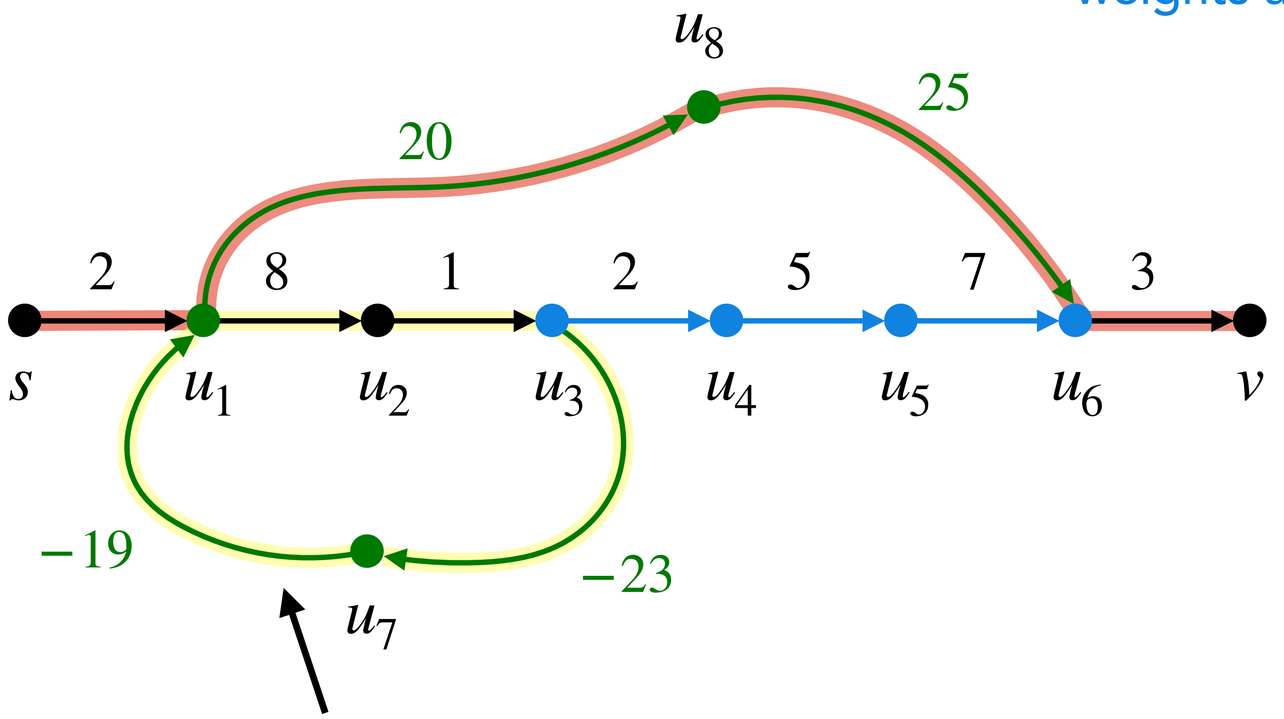
Now the proof is correct?

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.

Lemma is true only when weights are non-negative.



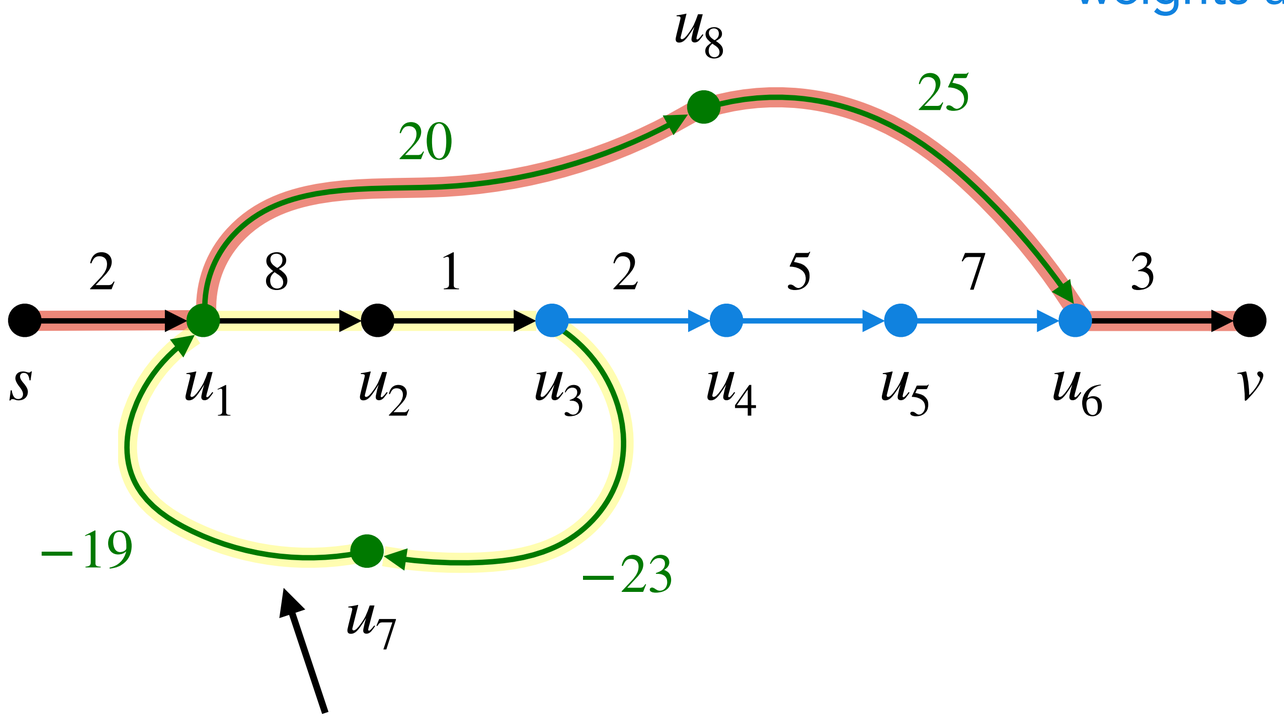
Shorter path from u_3 to u_6 going via u_1.

Optimal Subpath Property

Lemma: Every subpath of a shortest is also a shortest path.

Proof Sketch: Let P be a shortest path from s to v.

Lemma is true only when weights are non-negative.



Shorter path from u_3 to u_6 going via u_1.

■

Dijkstra’s Algorithm: Idea

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s.

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Dijkstra's Algorithm: Idea

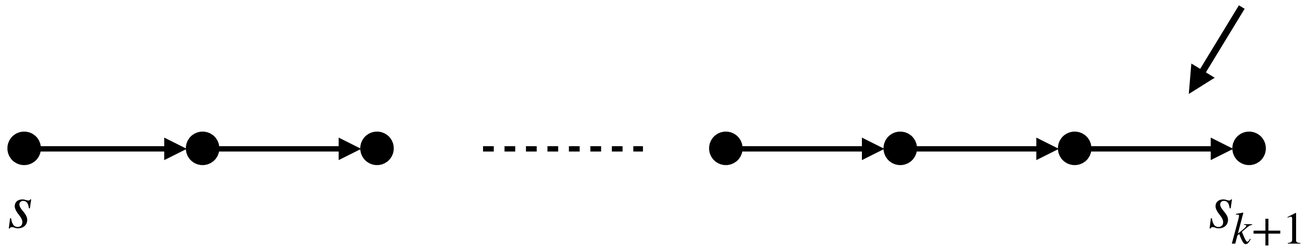
Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Shortest path to s_{k+1} from s.



Dijkstra's Algorithm: Idea

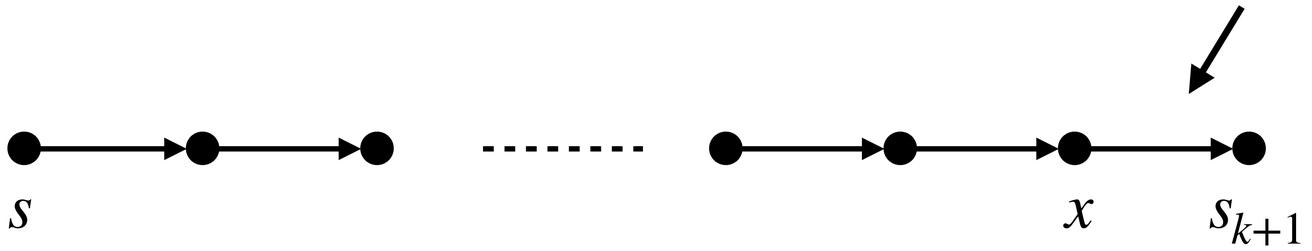
Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Shortest path to s_{k+1} from s.



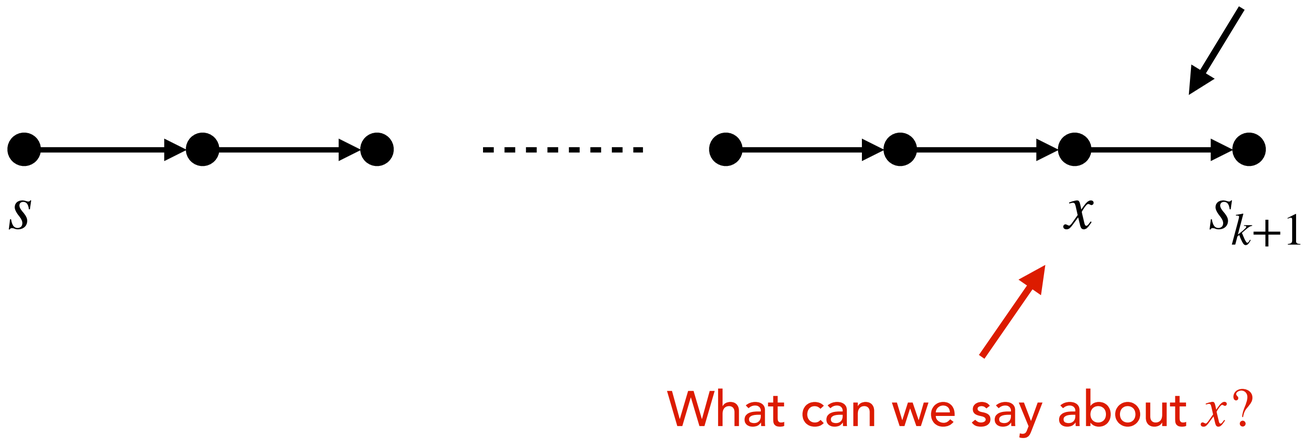
Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?



Dijkstra's Algorithm: Idea

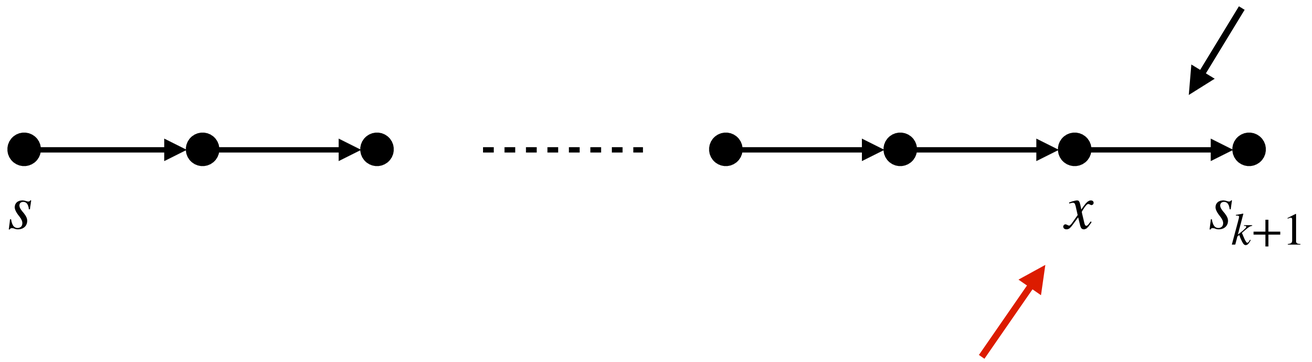
Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Shortest path to s_{k+1} from s.



What can we say about x?

x should be s_i for some $i \leq k$.

Dijkstra's Algorithm: Idea

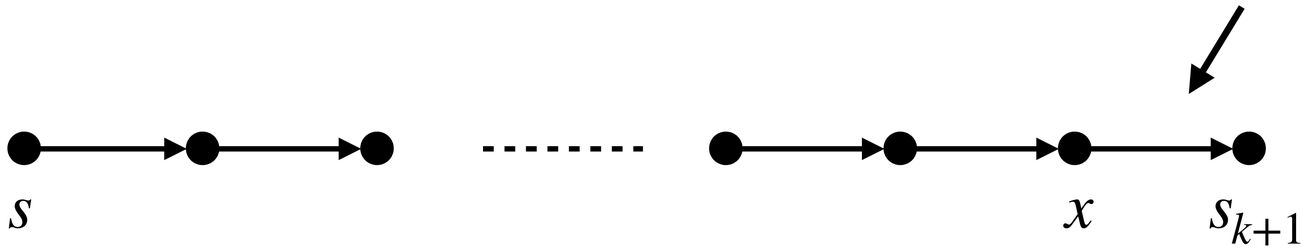
Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Shortest path to s_{k+1} from s.



Clearly, $d[s_{k+1}] = d[x] + w(x, s_{k+1})$.

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Shortest path to s_{k+1} from s.

Clearly, $d[s_{k+1}] = d[x] + w(x, s_{k+1})$.

We don't know x, so we compute this value over all $x \in S$ and pick the minimum one.

Dijkstra's Algorithm: Idea

Let s_i be the ith nearest vertex to s. (Assume there are no ties and all the weights are positive.)

Dijkstra's algorithm first computes distance of s_1, then s_2's, then s_3's, then s_4's, and so on.

Suppose we have calculated the distances of $S = \{s_1, s_2, \ldots, s_k\}$.

How can we compute distance of s_{k+1}?

Shortest path to s_{k+1} from s.

Clearly, $d[s_{k+1}] = d[x] + w(x, s_{k+1})$.

We don't know s_{k+1}, so we compute this value over all $v \in V \backslash S$ and the one with minimum value will be s_{k+1}.

We don't know x, so we compute this value over all $x \in S$ and pick the minimum one.

Dijkstra's Algorithm: Sketch

Dijkstra's Algorithm: Sketch

Maintain a set of explored vertices S for which algorithm has found $d[u] = \delta(s, u)$:

Dijkstra's Algorithm: Sketch

Maintain a set of explored vertices S for which algorithm has found $d[u] = \delta(s, u)$:

Step 1: Initialise $S = \{s\}$, $d[s] = 0$.

Dijkstra's Algorithm: Sketch

Maintain a set of explored vertices S for which algorithm has found $d[u] = \delta(s, u)$:

Step 1: Initialise $S = \{s\}$, $d[s] = 0$.

Step 2: Choose an unexplored vertex v from $V(G) \backslash S$ which minimizes:

Dijkstra's Algorithm: Sketch

Maintain a set of explored vertices S for which algorithm has found $d[u] = \delta(s, u)$:

Step 1: Initialise $S = \{s\}$, $d[s] = 0$.

Step 2: Choose an unexplored vertex v from $V(G) \backslash S$ which minimizes:

$$\pi[v] = \min_{(u,v) \in E,\ u \in S} d[u] + w(u, v)$$

Dijkstra's Algorithm: Sketch

Maintain a set of explored vertices S for which algorithm has found $d[u] = \delta(s, u)$:

Step 1: Initialise $S = \{s\}$, $d[s] = 0$.

Step 2: Choose an unexplored vertex v from $V(G) \backslash S$ which minimizes:

$$\pi[v] = \min_{(u,v) \in E,\ u \in S} d[u] + w(u, v)$$

Add v to S and set $d[v] = \Pi[v]$.

Dijkstra's Algorithm: Sketch

Maintain a set of explored vertices S for which algorithm has found $d[u] = \delta(s, u)$:

Step 1: Initialise $S = \{s\}$, $d[s] = 0$.

Step 2: Choose an unexplored vertex v from $V(G) \backslash S$ which minimizes:

$$\pi[v] = \min_{(u,v) \in E,\ u \in S} d[u] + w(u, v)$$

Add v to S and set $d[v] = \Pi[v]$.

Step 3: Go to **Step 2** if it can be performed.