

Part V: Graph Algorithms

Lecture 13: Dijkstra's Shortest Path Algorithm

Ahmed Khademzadeh
<http://khademzadeh.mshdiau.ac.ir>
Azad University of Mashhad



The slides are borrowed from Ke (Kevin) Yi. Thanks to him for his benevolence.

Objective and Outline

Objective: Discuss a third graph problem: single-source shortest paths

Reference: Section 24.3 of CLRS

Outline:

- The single-source shortest paths problem
- Dijkstra's algorithm
- Example
- Correctness
- Running time

Shortest Path Problem for Weighted Graphs

- Let $G = (V, E)$ be a **weighted** digraph (directed graph), with weight function $w : E \mapsto \mathbb{R}$ mapping edges to real-valued weights
- If $e = (u, v)$, we write $w(u, v)$ for $w(e)$.

Definition

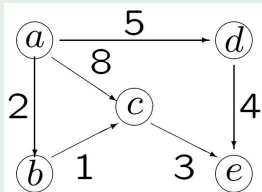
The **length** of a path $p = \langle v_0, v_1, \dots, v_k \rangle$ is the **sum** of the weights of its constituent edges:

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

Definition

The **distance** from u to v , denoted $\delta(u, v)$, is the length of the **minimum length path** if there is a path from u to v ; and is ∞ otherwise.

Example



- $\text{length}(\langle a, b, c, e \rangle) = 6$; distance from a to e is 6

Single-Source Shortest-Paths Problem

Definition (Single-source shortest-paths problem)

Given a digraph with **non-negative** edge weights $G = (V, E)$ and a designated **source vertex**, $s \in V$, determine the **distance** and a **shortest path** from the source vertex to **every** vertex in the digraph.

Question

How do you design an efficient algorithm for this problem?

- The single-source shortest paths problem
- Dijkstra's algorithm
- Example
- Correctness
- Running time

The Rough Idea of Dijkstra's Algorithm

- Maintain $d[v]$ and S :
 - $d[v]$ is an upper bound of the length $\delta(s, v)$ of the shortest path for each vertex v .
 - $S \subseteq V$ is a subset of vertices for which we know the true distance, that is $d[v] = \delta(s, v)$.
- Initially
 - $S = \emptyset$
 - $d[s] = 0$ and $d[v] = \infty$ for all others vertices v .
- One by one we select vertices from $V \setminus S$ to add to S .
- Questions to answer at each step:
 - 1 Which vertex do we select?
 - 2 How do we update the distance upper bounds after a vertex is added to S ?

Question

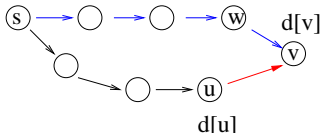
How does the algorithm **select** which vertex among the vertices of $V \setminus S$ to process next?

Answer: We use a **greedy** algorithm.

- For each vertex in $u \in V \setminus S$, we have computed a distance upper bound $d[u]$.
- The next vertex processed is always a vertex $u \in V \setminus S$ for which $d[u]$ is **minimum**
 - that is, we take the unprocessed vertex that is **closest** (by our estimate) to s .

Updating distance estimates

- Current distance upper bound for v : $d[v]$.
- Vertex u just added to S . Edge (u, v) with weight $w(u, v)$.
- How do we update $d[v]$?



- Current shortest path to v : $\langle s, \dots, w, v \rangle$, length $d[v]$.
- New path to v : $\langle s, \dots, u, v \rangle$, length $d[u] + w(u, v)$
- If new path is shorter ($d[u] + w(u, v) < d[v]$), update $d[v]$

$$d[v] = d[u] + w(u, v)$$

Now we have a better (tighter) upper bound for $d[v]$. This is called **relaxing** the edge (u, v)

The Algorithm for Relaxing an Edge

Relax(u, v)

```
if  $d[u] + w(u, v) < d[v]$  then  
  |  $d[v] = d[u] + w(u, v);$   
  |  $pred[v] = u;$   
end
```

Remark 1: The predecessor pointer $pred[]$ is for determining the shortest paths.

Remark 2: After edge (u, v) is relaxed, we have

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

The Selection in Dijkstra's Algorithm

Question

How do we perform this selection efficiently?

Answer: We store the vertices of $V \setminus S$ in a **Priority queue**, where the key value of each vertex v is $d[v]$.

- Note: if we implement the priority queue using a heap, we can perform the operations **Insert()**, **Extract_Min()**, and **Decrease_Key()**, each in $O(\log n)$ time.

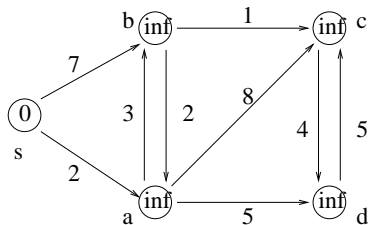
Description of Dijkstra's Algorithm

Dijkstra(G, w, s)

```
foreach  $u \in V$  do
  |  $d[u] = \infty$ ;  $color[u] = \text{white}$ ; // Initialize
end
 $d[s] = 0$ ;
 $pred[s] = \text{NIL}$ ;
 $Q = (\text{queue with all vertices})$ ;
while Non-Empty( $Q$ ) do
  | // Process all vertices
  |  $u = \text{Extract-Min}(Q)$ ; // Find new vertex
  | foreach  $v \in \text{Adj}[u]$  do
  |   | if  $d[u] + w(u, v) < d[v]$  then
  |     | // If estimate improves
  |     |  $d[v] = d[u] + w(u, v)$ ; // relax
  |     | Decrease-Key( $Q, v, d[v]$ );
  |     |  $pred[v] = u$ ;
  |   | end
  | end
  |  $color[u] = \text{black}$ ;
end
```

- The single-source shortest paths problem
- Dijkstra's algorithm
- Example
- Correctness
- Running time

Example



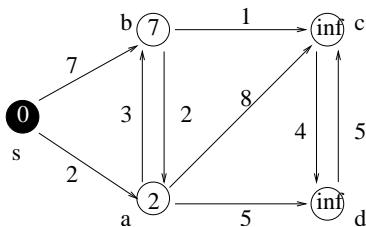
Step 0: Initialization.

v	s	a	b	c	d
$d[v]$	0	∞	∞	∞	∞
$pred[v]$	nil	nil	nil	nil	nil
$color[v]$	W	W	W	W	W

Priority Queue:

v	s	a	b	c	d
$d[v]$	0	∞	∞	∞	∞

Example...



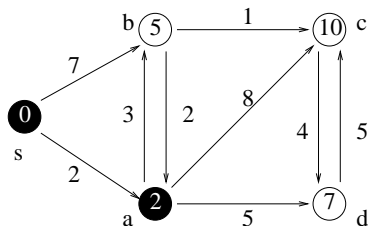
Step 1: As $Adj[s] = \{a, b\}$, work on a and b and update information.

v	s	a	b	c	d
$d[v]$	0	2	7	∞	∞
$pred[v]$	nil	s	s	nil	nil
$color[v]$	B	W	W	W	W

Priority Queue:

v	a	b	c	d
$d[v]$	2	7	∞	∞

Example...



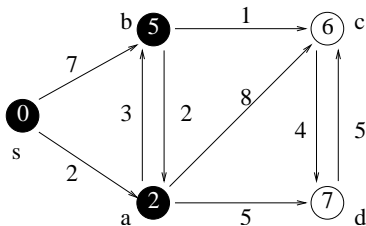
Step 2: After Step 1, a has the minimum key in the priority queue. As $Adj[a] = \{b, c, d\}$, work on b, c, d and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	10	7
$pred[v]$	nil	s	a	a	a
$color[v]$	B	B	W	W	W

Priority Queue:

v	b	c	d
$d[v]$	5	10	7

Example...



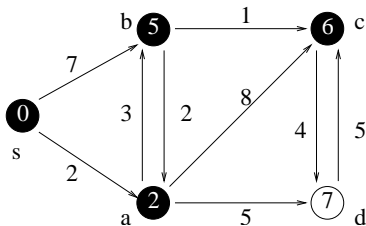
Step 3: After Step 2, b has the minimum key in the priority queue. As $Adj[b] = \{a, c\}$, work on a, c and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	W	W

Priority Queue:

v	c	d
$d[v]$	6	7

Example...



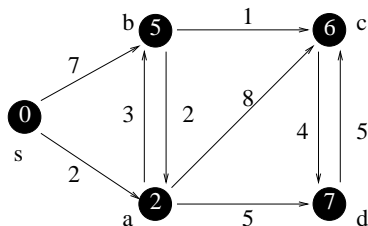
Step 4: After Step 3, c has the minimum key in the priority queue. As $Adj[c] = \{d\}$, work on d and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	B	W

Priority Queue:

v	d
$d[v]$	7

Example...



Step 5: After Step 4, d has the minimum key in the priority queue. As $Adj[d] = \{c\}$, work on c and update information.

v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a
$color[v]$	B	B	B	B	B

Priority Queue: $Q = \emptyset$.

We are done.

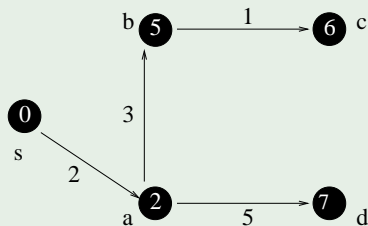
Dijkstra's Algorithm

Shortest Path Tree: $T = (V, A)$, where

$$A = \{(pred[v], v) | v \in V \setminus \{s\}\}.$$

The array $pred[v]$ is used to build the tree.

Example



v	s	a	b	c	d
$d[v]$	0	2	5	6	7
$pred[v]$	nil	s	a	b	a

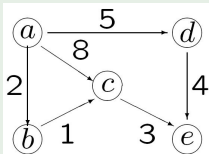
- The single-source shortest paths problem
- Dijkstra's algorithm
- Example
- Correctness
- Running time

Key Observation

Lemma

Any sub-path of a shortest path must also be a shortest path

Example



$\langle a, b, c, e \rangle$ is a shortest path; sub-path $\langle a, b, c \rangle$ is also a shortest path.

Question

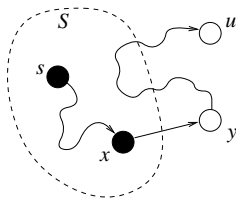
Why?

Correctness of Dijkstra's Algorithm

Theorem

When a vertex u is added to S (i.e., dequeued from the queue), $d[u] = \delta(s, u)$.

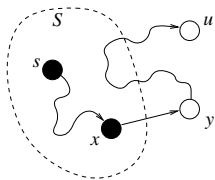
Proof:



- Suppose **to the contrary** that at some point Dijkstra's algorithm **first** attempts to add a vertex u to S for which $d[u] \neq \delta(s, u)$.
- Since $d[u] = \infty$ initially, and remains an **upper bound** of the true shortest distance, we have $d[u] > \delta(s, u)$.
- Consider the situation just prior to the insertion of u .
 - Consider the **true shortest path** from s to u .
 - Because $s \in S$ and $u \in V \setminus S$, at some point this path must first take a jump out of S .
 - Let (x, y) be the edge taken by the path, where $x \in S$ and $y \in V \setminus S$ (it may be that $x = s$ or/and $y = u$).

Correctness of Dijkstra's Algorithm – Continued

We now prove that $d[y] = \delta(s, y)$.



- We have done relaxation when processing x , so
$$d[y] \leq d[x] + w(x, y). \quad (1)$$

- Since x is added to S earlier, by hypothesis,
$$d[x] = \delta(s, x). \quad (2)$$

- Since $\langle s, \dots, x, y \rangle$ is sub-path of a shortest path, by (2)
$$\delta(s, y) = \delta(s, x) + w(x, y) = d[x] + w(x, y). \quad (3)$$

- By (1) and (3), $d[y] \leq \delta(s, y)$. Hence $d[y] = \delta(s, y)$. So $y \neq u$ (because we suppose $d[u] > \delta(s, u)$).

Since y appears midway on the path from s to u , and all subsequent edges have non-negative weights, we have $\delta(s, y) \leq \delta(s, u)$, and thus

$$d[y] = \delta(s, y) \leq \delta(s, u) < d[u].$$

Thus y should have been added to S **before** u , in contradiction to our

- The single-source shortest paths problem
- Dijkstra's algorithm
- Example
- Correctness
- Running time

Analysis of Dijkstra's Algorithm:

(Algorithm)

- The initialization uses only $O(n)$ time.
- Each vertex is processed exactly once, so `Non-Empty()` and `Extract-Min()` are called exactly once, i.e., n times in total.
- The inner loop `for (each $v \in Adj[u]$)` is called once for each edge in the graph. Each call of the inner loop does $O(1)$ work plus, possibly, one `Decrease-Key` operation.
- Recalling that all of the priority queue operations require $O(\log |Q|) = O(\log V)$ time, we have that the algorithm uses

$$V \cdot O(1 + \log V) + O(E) + O(E \log V) = O(E \log V)$$

time.

Minimum spanning tree vs shortest path tree

- Dijkstra's algorithm looks similar to Prim's algorithm.
- To understand the differences clearly, try them both on some example.
- Is a minimum spanning tree a shortest path tree? Vice versa?