Implementation of Dijkstra's Algorithm

Yufei Tao's Teaching Team

In the lecture, we have proved that Dijkstra's algorithm computes the shortest distances correctly. Today, we will see how the algorithm can be slightly augmented to output a shortest path tree. Furthermore, we will also discuss how to implement the algorithm in $O((|V| + |E|) \log |V|)$ time.

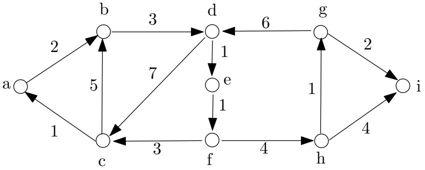
Single Source Shortest Path (SSSP) with Positive Weights

Let (G, w) with $G = (V, E)$ be a directed weighted graph, where w maps every edge of E to a positive value.

Given a vertex s in V, the goal of the **SSSP problem** is to find, for every other vertex $t \in V \setminus \{s\}$, a shortest path from s to t, unless t is unreachable from s.

Example

Suppose that the source vertex is a.

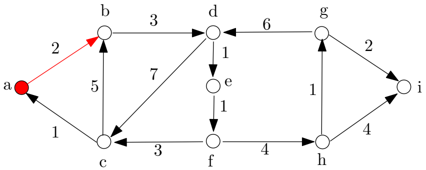


vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	∞	nil
c	∞	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$F = \emptyset$ and
$S = \{a, b, c, d, e, f, g, h, i\}$.
Since $dist(a)$ is the smallest among those of vertices in P, pick a.

Example

Relax the out-going edges of a. **Red** edges correspond to the parent column.

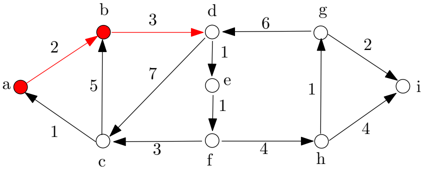


$F = \{a\}$ (vertices finalized) and $S = \{b, c, d, e, f, g, h, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	$\infty \to 2$	nil $\to a$
c	∞	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

Example

Relax the out-going edges of b. **Red** edges correspond to the parent column.

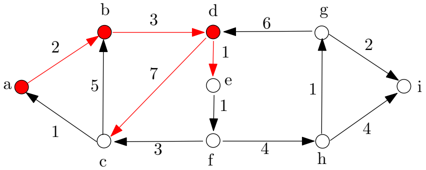


$F = \{a, b\}$ and
$S = \{c, d, e, f, g, h, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	∞	nil
d	$\infty \to 5$	nil $\to b$
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

Example

Relax the out-going edges of d. **Red** edges correspond to the parent column.

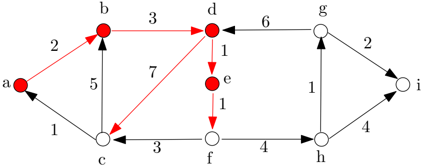


$F = \{a, b, d\}$ and
$S = \{c, e, f, g, h, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	$\infty \rightarrow 12$	nil $\rightarrow d$
d	5	b
e	$\infty \rightarrow 6$	nil $\rightarrow d$
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

Example

Relax the out-going edges of e. **Red** edges correspond to the parent column.

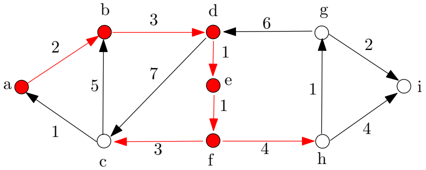


$F = \{a, b, d, e\}$ and
$S = \{c, f, g, h, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	12	d
d	5	b
e	6	d
f	$\infty \to 7$	nil $\to e$
g	∞	nil
h	∞	nil
i	∞	nil

Example

Relax the out-going edges of f. **Red** edges correspond to the parent column.

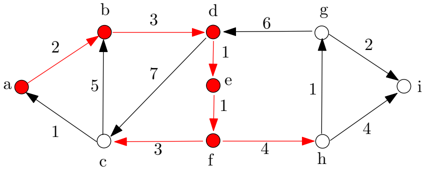


$F = \{a, b, d, e, f\}$ and
$S = \{c, g, h, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	$12 \rightarrow 10$	$d \rightarrow f$
d	5	b
e	6	d
f	7	e
g	∞	nil
h	$\infty \rightarrow 11$	nil $\rightarrow f$
i	∞	nil

Example

Relax the out-going edges of c. **Red** edges correspond to the parent column.

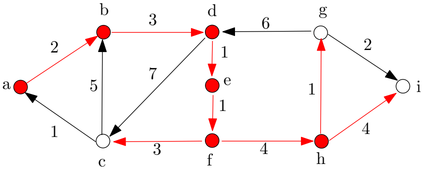


$F = \{a, b, c, d, e, f\}$ and $S = \{g, h, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	∞	nil
h	11	f
i	∞	nil

Example

Relax the out-going edges of h. **Red** edges correspond to the parent column.

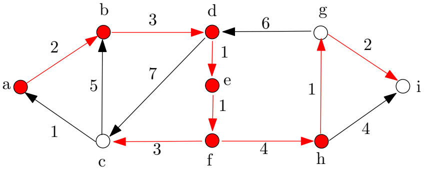


$F = \{a, b, c, d, e, f, h\}$ and $S = \{g, i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	$\infty \to 12$	nil $\to h$
h	11	f
i	$\infty \to 15$	nil $\to h$

Example

Relax the out-going edges of g. **Red** edges correspond to the parent column.

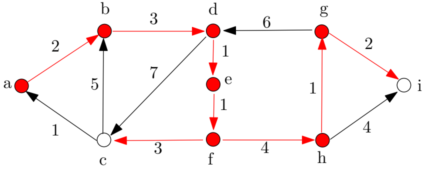


$F = \{a, b, c, d, e, f, g, h\}$ and $S = \{i\}$.

vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	12	h
h	11	f
i	$15 \rightarrow 14$	$h \rightarrow g$

Example

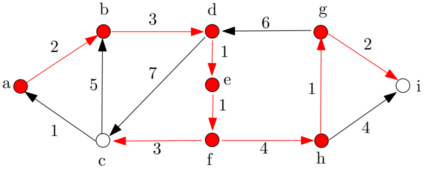
Relax the out-going edges of i. **Red** edges correspond to the parent column.



vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	12	h
h	11	f
i	14	g

$F = \{a, b, c, d, e, f, g, h, i\}$
and
$S = \{\}$.
Done.

Example



vertex v	$dist(v)$	$parent(v)$
a	0	nil
b	2	a
c	10	f
d	5	b
e	6	d
f	7	e
g	12	h
h	11	f
i	14	g

The set of red edges indicates the shortest path tree that we should output. **Think:** Given the table's last column, how can you find this tree in $O(|V|)$ time?

Next, we will discuss how to implement the algorithm in $O((|V| + |E|) \log |V|)$ time. For this purpose, we will assume that each vertex in V carries an ID from 1 to $n = |V|$.

Data Structures

Recall that S is the set of vertices that have not been finalized.

We create an array A of size n where, for each $v \in [1, n]$, the cell $A[v]$ stores $dist(v)$ and $parent(v)$.

Define $D = \{dist(v) \mid v \in S\}$. We store D in an AVL-tree T.

All these data structures can be constructed at the beginning of Dijkstra's algorithm in $O(n \log n)$ time.

To run Dijkstra's, we need to support two operations:

- **DecreaseKey**(v, k_{new}): reduce $dist(v)$ to k_{new} in D if $k_{new} < dist(v)$.
- **DeleteMin**: remove the smallest $dist(v)$ from D.

We perform DecreaseKey at most $|E|$ times and DeleteMin at most $|V|$ times.

Think: Why?

Implementation of DecreaseKey

Goal: Reduce $dist(v)$ to k_{new} in D if $k_{new} < dist(v)$.

1. Find the current $dist(v)$ from array A
2. If $dist(v) \leq k_{new}$, do nothing and return
3. Delete $dist(v)$ from the AVL-tree T
4. Set $A[v] = k_{new}$
5. Insert k_{new} into T

$O(\log |V|)$ time.

Implementation of DeleteMin

Goal: Remove the smallest $dist(v)$ from D

Finding the smallest key in an AVL-tree takes only $O(\log |V|)$ time. After that, we can remove the key.

Two Minor Issues

- What if D has duplicate $dist(v)$ values?
- In DeleteMin, we also need to identify the vertex v whose $dist(v)$ is the smallest in D.

How to solve these issues (easily)?