Single Source Shortest Paths with Positive Weights

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

In this lecture, we will discuss the **single source shortest path** (SSSP) problem, which is a classic problem on graphs, and also a problem very plenty of applications in practice.

Weighted Graphs

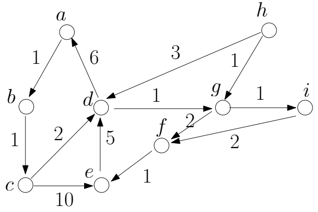
Let $G = (V, E)$ be a directed graph. Let w be a function that maps each edge in E to a **positive** integer value. Specifically, for each $e \in E$, $w(e)$ is an integer at least 0, which we call the **weight** of e.

A **directed weighted graph** is defined as the pair (G, w).

We use the notation (u, v) to denote an edge in G from node u to node v. Here, node u is an **in-neighbor** of v.

Define $IN(v)$ the set of all in-neighbors of v.

Example



The integer on each edge indicates its weight. For example, $w(d, g) = 1$, $w(g, f) = 2$, and $w(c, e) = 10$.

$IN(d) = \{c, e, h\}$.

Shortest Path

Consider a path in G: $(v_1, v_2), (v_2, v_3), ..., (v_\ell, v_{\ell+1})$, for some integer $\ell \geq 1$. We define the **length** of the path as

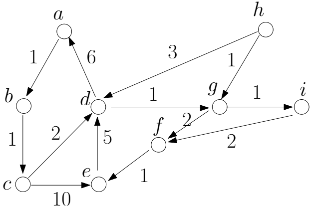
$$\sum_{i=1}^{\ell} w(v_i, v_{i+1}).$$

Recall that we may also denote the path as $v_1 \to v_2 \to ... \to v_{\ell+1}$.

A **shortest path** from u to v is a path that has the minimum length among all the paths from u to v. Denote by $spdist(u, v)$ the length of the shortest path from u to v.

If v is unreachable from u, then $spdist(u, v) = \infty$.

Example



- The path $c \rightarrow e$ has length 10.
- The path $c \rightarrow d \rightarrow g \rightarrow f \rightarrow e$ has length 6.

The first path is a shortest path from c to e; $spdist(c, e) = 6$.

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.

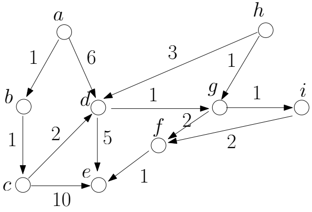
Given a vertex s in V, the goal of the **SSSP problem** is to find, for **every** other vertex $t \in V \setminus \{s\}$, a shortest path from s to t, unless t is unreachable from s.

A Subsequence Property

Lemma: If $v_1 \to v_2 \to ... \to v_{\ell+1}$ is a shortest path from v_1 to $v_{\ell+1}$, then for every i, j satisfying $1 \leq i < j \leq \ell + 1$, $v_i \to v_{i+1} \to ... \to v_j$ is a shortest path from v_i to v_j.

Proof: Suppose that this is not true. Then, we can find a shorter path to go from v_i to v_j. Using this path to replace the original path from v_i to v_j yields a shorter path from v_1 to $v_{\ell+1}$, which contradicts the fact that $v_1 \to v_2 \to ... \to v_{\ell+1}$ is a shortest path. $\square$

Example



Since $c \to d \to g \to f \to e$ is a shortest path, we know that any **subsequence** of of this path is also a shortest path. For example, $c \to d \to g \to f$ must be a shortest path from c to f.

Lemma:

$$spdist(s,u) \quad = \quad \min_{v \in IN(u)} \{spdist(s,v) + w(v,u)\}$$

The proof is simple and left to you.

Implication: This is a dynamic programming problem!
But what is non-trivial is how we should fill in the "matrix"! Namely, what is the order of u by which we should compute $spdist(s,u)$?

Remark: The above lemma holds even if $w(v,u)$ can be negative.

Next, we will first explain **Dijkstra's algorithm** for solving the SSSP problem. As we will see, this algorithm essentially tells us a good order to compute $spdist(s, u)$ when all the edges have positive weights.

Utilizing the subsequence property, our algorithm will output a **shortest path tree** that encodes all the shortest paths from the source vertex s.

The Edge Relaxation Idea

For every vertex $v \in V$, we will – at all times – maintain a value $dist(v)$ that represents the length of the shortest path from s to v **found so far**.

At the end of the algorithm, we will ensure that every $dist(v)$ equals the shortest path distance from s to v.

A core operation in our algorithm is called **edge relaxation**:

- **Relaxing** an edge (u, v) means:
  - If $dist(v) < dist(u) + w(u, v)$, do nothing;
  - Otherwise, reduce $dist(v)$ to $dist(u) + w(u, v)$.

Dijkstra's Algorithm

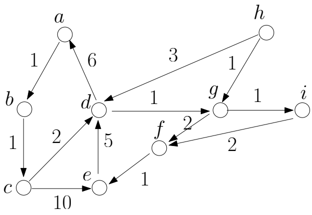
1. Set $parent(v) = \text{nil}$ for all vertices $v \in V$
2. Set $dist(s) = 0$, and $dist(v) = \infty$ for all other vertices $v \in V$
3. Set $S = V$
4. Repeat the following until S is empty:

   5.1 Remove from S the vertex u with the **smallest** $dist(u)$.
   /* next we relax all the outgoing edges of u */

   5.2 Relax every outgoing edge (u, v) of u

Example

Suppose that the source vertex is c.

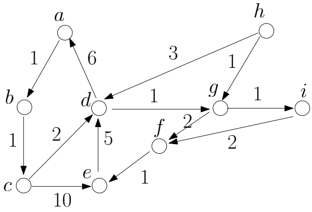


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

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

Example

Relax the out-going edges of c (because $dist(c)$ is the smallest in S):

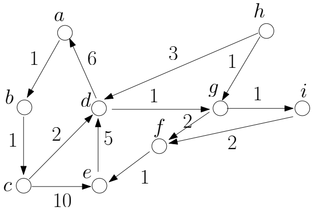


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$S = \{a, b, d, e, f, g, h, i\}$.
Note that c has been removed!

Example

Relax the out-going edges of d (because $dist(d)$ is the smallest in S):

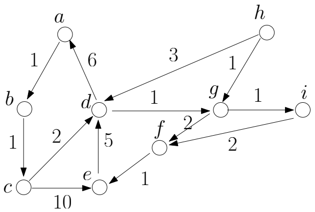


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	3	d
h	∞	nil
i	∞	nil

$S = \{a, b, e, f, g, h, i\}$.

Example

Relax the out-going edges of g:

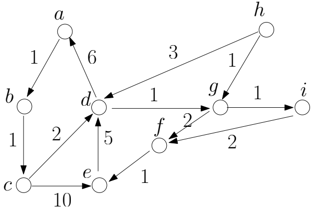


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, e, f, h, i\}$.

Example

Relax the out-going edges of i:

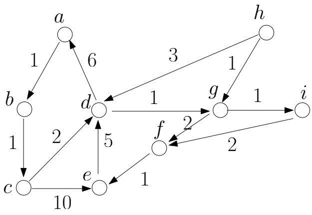


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, e, f, h\}$.

Example

Relax the out-going edges of f:

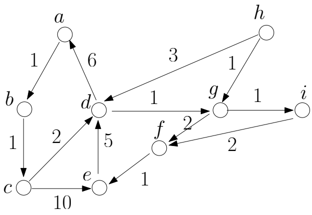


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, e, h\}$.

Example

Relax the out-going edges of e:

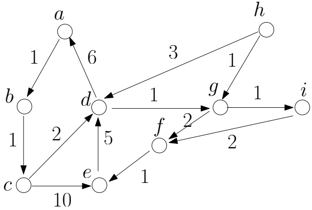


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{a, b, h\}$.

Example

Relax the out-going edges of a:

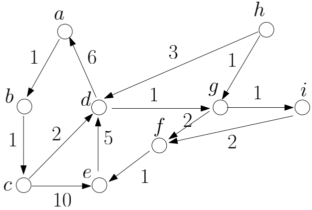


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{b, h\}$.

Example

Relax the out-going edges of b:

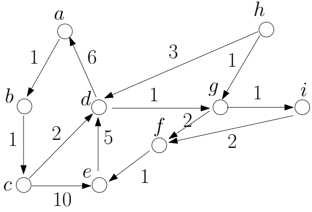


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{h\}$.

Example

Relax the out-going edges of h:



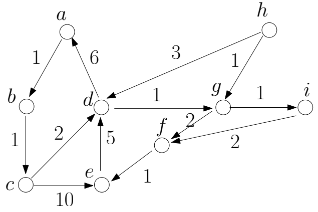
vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{\}$.
All the shortest path distances are now final.

Constructing the Shortest Path Tree

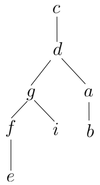
For every vertex v, if $u = parent(v)$ is not nil, then make v a child of u.

Example



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

shortest path tree



Running Time

It will be left as an exercise for you to implement Dijkstra's algorithm in $O((|V| + |E|) \cdot \log |V|)$ time (solutions provided).

Correctness

Theorem: When a node u is removed from S, the value $dist(u)$ equals precisely $spdist(s, u)$.

We will prove the theorem by induction on the order of vertices removed. The first vertex removed is just the source vertex s itself, on which the statement of the theorem obviously holds because
$dist(u) = spdist(s, u) = 0$.

Assuming that the theorem holds for all the vertices removed so far, we will prove its correctness on the **next** vertex u to be removed from S.

Let π be a shortest path from s to u. We will prove the following claim:

Claim: When u is to be removed from S, all the vertices on π has been removed.

The claim implies $dist(u) = spdist(u)$ when u is removed from S. To see why, let p be the node right before u on π. By our inductive assumption, when p was removed from S, we had $dist(p) = spdist(p)$. Recall that after removing p, we needed to relax all the outgoing edges of p, one of which was (p, u). After relaxing the edge, we must have $dist(u) = dist(p) + w(p, u) = spdist(p) + w(p, u) = spdist(u)$.

Proof of the claim: Suppose that the claim is not true. Define v_{bad} be the **first** vertex on π that is still in S, when u is to be removed from S.

Let v_{good} be the vertex right before v_{bad} on π; note that v_{good} definitely exists because v_{bad} cannot be s.

By our inductive assumption, when v_{good} was removed from S, we had $dist(v_{good}) = spdist(v_{good})$. Remember we needed to relax all the the outgoing edges of v_{good}, one of which was (v_{good}, v_{bad}). After relaxing the edge, we must have

$$\begin{aligned} dist(v_{bad}) &= dist(v_{good}) + w(v_{good}, v_{bad}) \\ &= spdist(v_{good}) + w(v_{good}, v_{bad}) = spdist(v_{bad}). \end{aligned}$$

Since $dist(v_{bad})$ never increases during the algorithm, we must have $dist(v_{bad}) < dist(u)$ when u is to be removed from S. But this contradicts the fact that u has the **smallest** *dist*-value among all the vertices in S. $\square$

We have completed the proof on the correctness of Dijkstra. It is important to note that Dijkstra does **not** work if edges can take negative weights — can you spot the place in our earlier argument that depends on positive weights?