Single Source Shortest Paths (with Positive Weights)

Yufei Tao

ITEE
University of Queensland

In this lecture, we will revisit the single source shortest path (SSSP) problem. Recall that we have already learned that the BFS algorithm solves the problem efficiently when all the edges have the same weight. Today we will see how to solve the problem in the more general situation where the edges can have arbitrary positive weights.

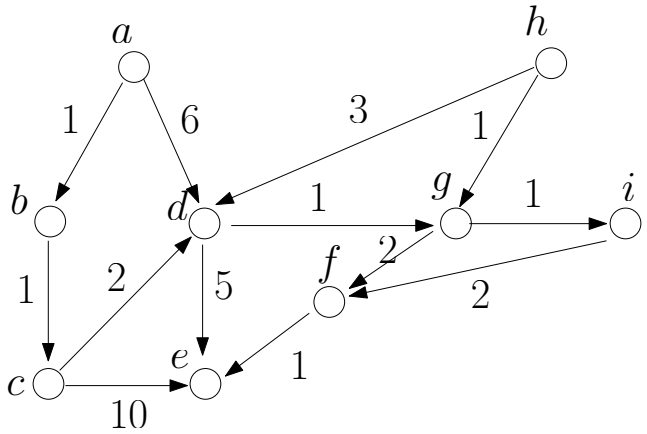
We will discuss two variants of the problem, depending on whether the input graph is a DAG or not. The former is clearly a special case of the latter, and as a result, admits a faster algorithm.

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 a positive integer value, which we call the weight of e.

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

Example



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

Shortest Path

Consider a directed weighted graph defined by a directed graph $G = (V, E)$ and function w.

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

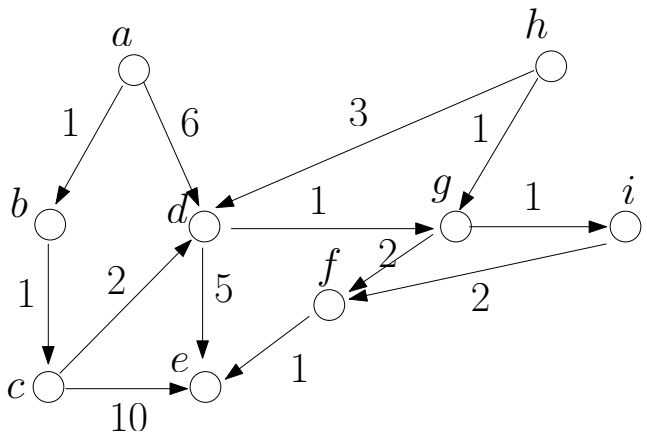
$$\sum_{i=1}^{\ell} w(v_i, v_{i+1}).$$

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

Given two vertices $u, v \in V$, a shortest path from u to v is a path from u to v that has the minimum length among all the paths from u to v.

If v is unreachable from u, then the shortest path distance from u to v is ∞.

Example



- The path $a \to d \to e$ has length 11.
- The path $a \to b \to c \to d \to g \to f \to e$ has length 8.

The second path is a shortest path from a to e.

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.

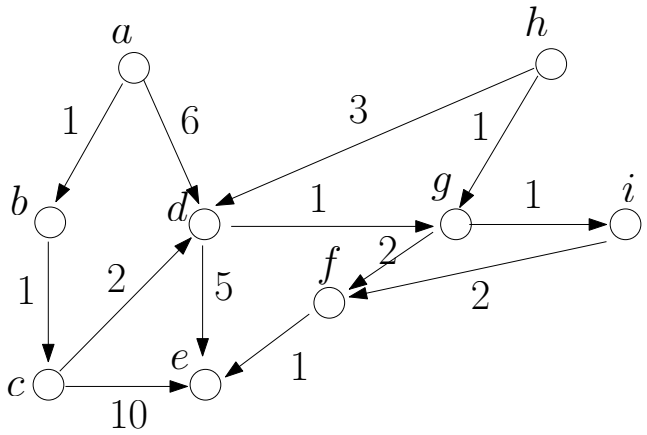
Given a vertex s in V, the goal of the SSSP problem is to find, for every other vertex $t \in V \setminus \{s\}$, a shortest path from s to t, unless t is unreachable from s.

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. □

Example



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

Next, we will first explain how to solve the SSSP problem when the input graph G is a DAG.

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 is guaranteed to be an upper bound of the shortest path distance from s to v.

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

A core operation in our algorithm is called edge relaxation:

- Given an edge (u, v), we relax it as follows:
  - If $dist(v) < dist(u) + w(u, v)$, do nothing;
  - Otherwise, reduce $dist(v)$ to $dist(u) + w(u, v)$.

The Relaxation Lemma

Lemma: $dist(v)$ is still an upper bound of the shortest path distance form s to v after the relaxation.

Proof: It suffices to prove that the shortest path from s to v cannot have a length greater than $dist(u) + w(u, v)$.

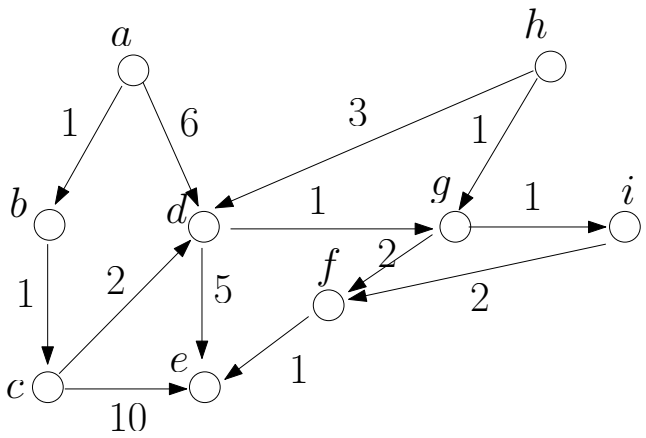
The value of $dist(u)$ indicates the existence of a path from s to u whose distance is at most $dist(u)$. We can therefore go from s to v by first taking that path to go to u, and then reach v by crossing the edge (u, v). This gives us a path from s to v whose distance is at most $dist(u) + w(u, v)$. □

SSSP Algorithm on a DAG

1. Obtain a topological order L on G
2. Set $dist(s) = 0$, and $dist(v) = \infty$ for all other vertices $v \in V$
3. Set $parent(v) =$ nil for all vertices $v \in V$
4. Remove all the vertices in L that are before s
5. Repeat the following until L is empty:
   - 5.1 remove the first vertex u of L
     /* next we relax all the outgoing edges of u */
   - 5.2 for every outgoing edge (u, v) of u
     - 5.2.1 if $dist(v) > dist(u) + w(u, v)$ then
       set $dist(v) = dist(u) + w(u, v)$, and $parent(v) = u$

Example

Suppose that the source vertex is c.

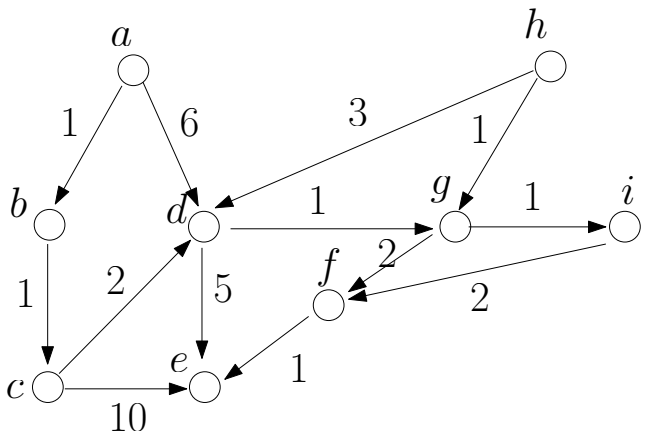


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

First, obtain an arbitrary topological order, e.g.,
$L = (a, b, c, h, d, g, i, f, e)$.
Initialize $dist(v)$ and $parent(v)$.
Remove all vertices before c in L, after which $L = (c, h, d, g, i, f, e)$.

Example

Relax all the outgoing edges of c.

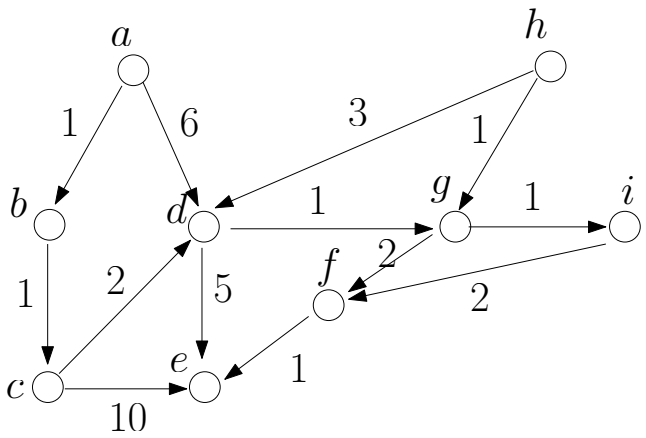


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$L = (h, d, g, i, f, e)$.

Example

Relax all the outgoing edges of h.

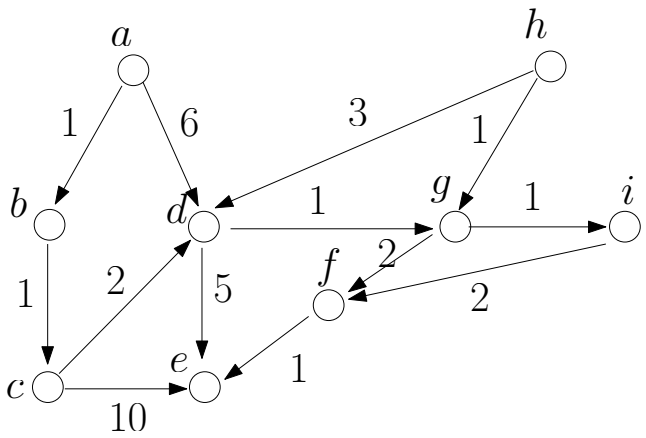


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

$L = (d, g, i, f, e)$.

Example

Relax all the outgoing edges of d.

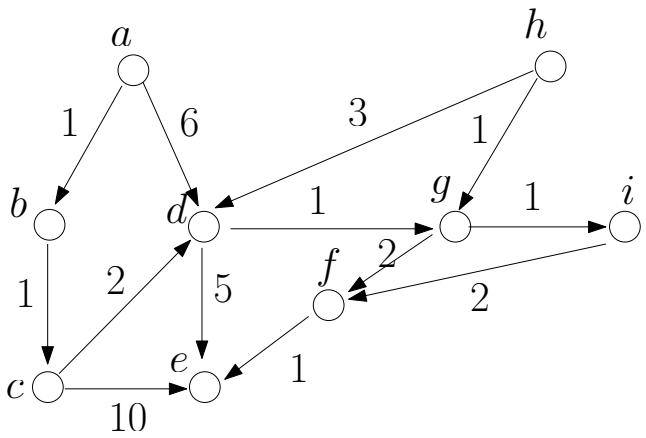


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	7	d
f	∞	nil
g	3	d
h	∞	nil
i	∞	nil

$L = (g, i, f, e)$.

Example

Relax all the outgoing edges of g.

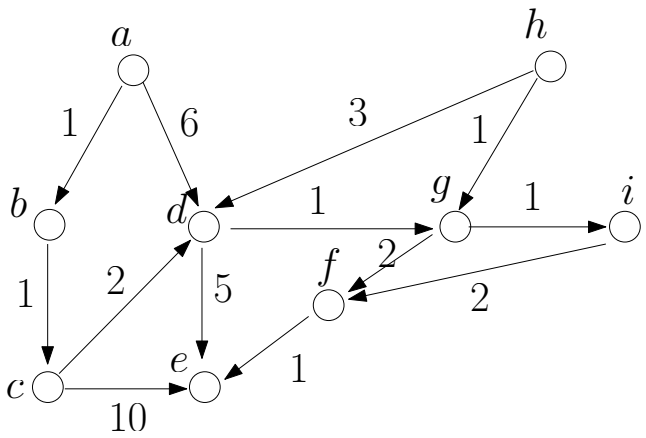


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	7	d
f	5	g
g	3	d
h	∞	nil
i	4	g

$L = (i, f, e)$.

Example

Relax all the outgoing edges of i.

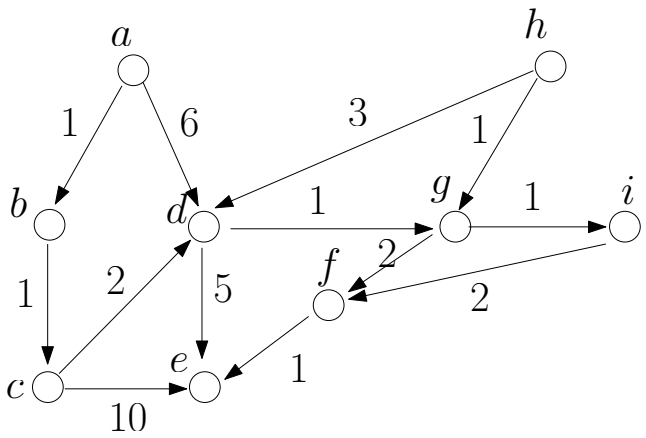


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	7	d
f	5	g
g	3	d
h	∞	nil
i	4	g

$L = (f, e)$.

Example

Relax all the outgoing edges of f.

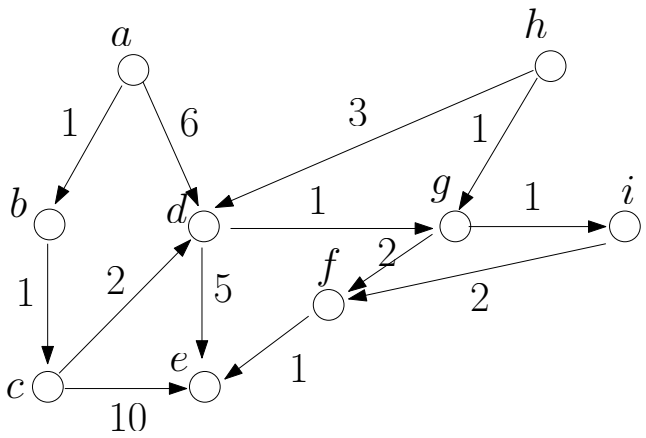


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$L = (e)$.

Example

Relax all the outgoing edges of e.



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

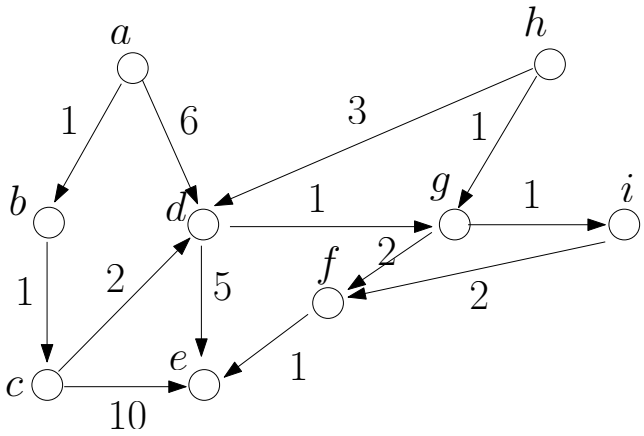
$L = ()$.

At this point, we have computed the shortest path distances from c to all the other vertices.

Constructing the Shortest Path Tree

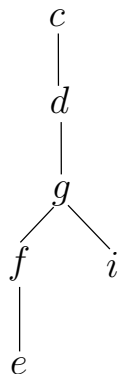
For every vertex v, if $u = parent(v)$ is not nil, then make v a child of u.

Example



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

shortest path tree



Time Analysis

Step 1 (on Slide 30) can be done in $O(|V| + |E|)$ time (DFS).
Steps 2-4 obviously take $O(|V|)$ time.

Regarding Step 5, notice that every outgoing edge is relaxed only once. Since each relaxation takes only $O(1)$ time, we know that Step 5 can be implemented in $O(|V| + |E|)$ time ($O(1)$ time spent removing each vertex from L).

Constructing the shortest path tree at the end obviously takes $O(|V|)$ time.

Overall the algorithm runs in $O(|V| + |E|)$ time (which is optimal).

We now prove that our algorithm is correct. In particular, we will prove that when the algorithm finishes, $dist(v)$ holds the correct shortest path distance from s to v.

Strictly speaking, we also need to prove that the fields of $parent(v)$ allow us to construct the shortest path tree correctly, but this is essentially a simple corollary of the above claim (and is left for you).

Correctness

Notice that $dist(v)$ will never change after v is removed from L (and has its out-going edges relaxed).

We will prove:

Lemma: When $dist(v)$ is removed from L, $dist(v)$ equals the shortest path distance from s to v.

The lemma is sufficient for establishing the correctness of our algorithm.

Correctness

Proof: Recall that at the beginning of the algorithm, we directly remove all the vertices v that precede s in L, leaving $dist(v) = \infty$. This is correct because none of those vertices are reachable from s (by definition of topological order).

We will prove that the lemma also holds for the other vertices. We will do so by induction on the order that vertices are removed from L, starting from s.

Base Case: When s is removed, $dist(s) = 0$, as required by the lemma.

Correctness

Proof (cont.):

Inductive Case: Suppose that we are removing v from L, and that the lemma holds on all the vertices removed before v.

Let π be a shortest path from s to v, and u the vertex on π immediately before v. It thus follows that u must be before v in L, and hence, must have been removed.

It follows that $dist(u)$ must be the shortest path distance from s to u. Thus, by relaxing edge (u, v), the algorithm ensures that $dist(v)$ cannot be higher than $dist(u) + w(u, v)$, which is exactly the shortest path distance from s to v.

Then the lemma follows from the fact that $dist(v)$ always remains an upper bound of the shortest path distance (the relaxation lemma). □

Next, we will extend the above algorithm to solve the SSSP problem when the input graph G is a general directed graphs (i.e., cycles may exist).

The major difference is the order that vertices are processed (recall that a cylcic graph has no topological orders).

The extended algorithm is called Dijkstra's algorithm.

The Edge Relaxation Idea

Same idea as before!

For every vertex $v \in V$, we will—at all times—maintain a value $dist(v)$ that is guaranteed to be an upper bound of the shortest path distance from s to v. At the end of the algorithm, we will ensure that every $dist(v)$ equals the precise shortest path distance from s to v.

Edge relaxation: Given an edge (u, v), we relax it as follows:

- If $dist(v) < dist(u) + w(u, v)$, do nothing;
- Otherwise, reduce $dist(v)$ to $dist(u) + w(u, v)$.

The relaxation lemma still holds.

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:

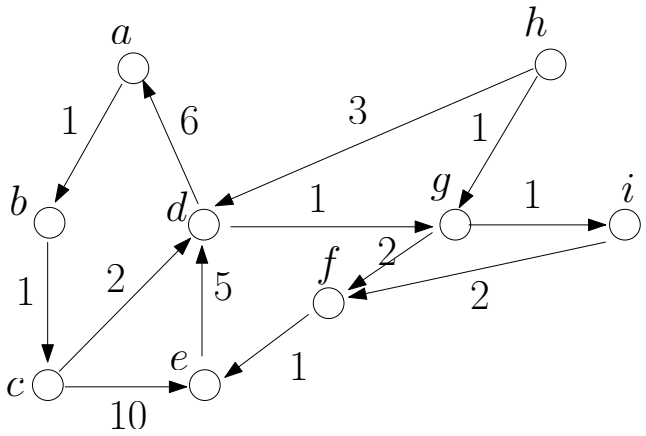
   5.1 Remove from S the vertex u with the smallest $dist(u)$.
   /* next we relax all the outgoing edges of u */

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

      5.2.1 if $dist(v) > dist(u) + w(u, v)$ then
      set $dist(v) = dist(u) + w(u, v)$, and $parent(v) = u$

Example

Suppose that the source vertex is c.

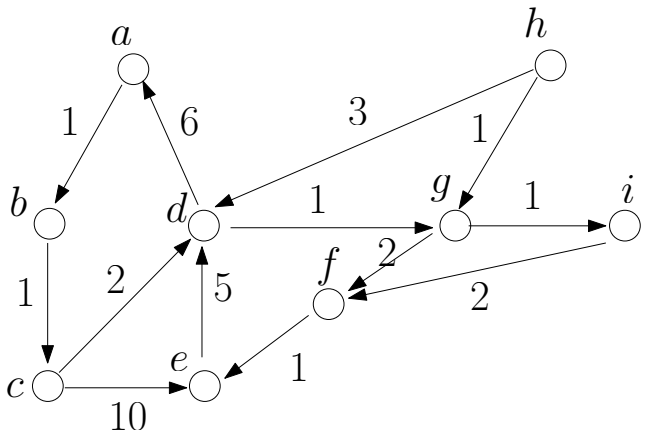


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	∞	nil
e	∞	nil
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

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

Example

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

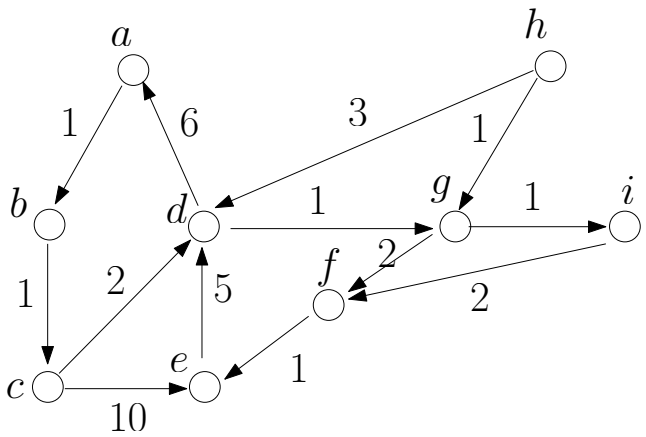


vertex v	$dist(v)$	$parent(v)$
a	∞	nil
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	∞	nil
h	∞	nil
i	∞	nil

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

Example

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

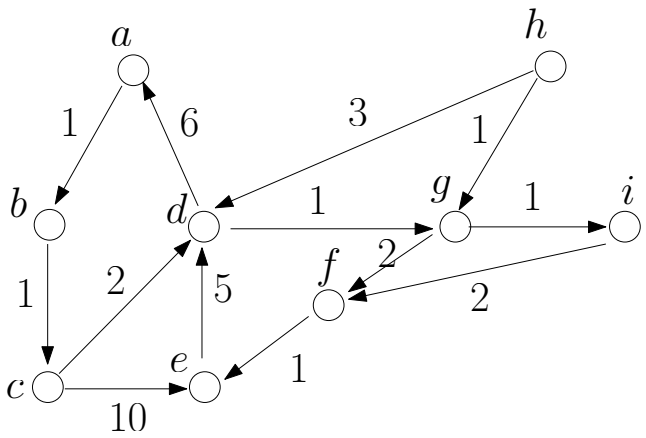


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	∞	nil
g	3	d
h	∞	nil
i	∞	nil

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

Example

Relax the out-going edges of g:

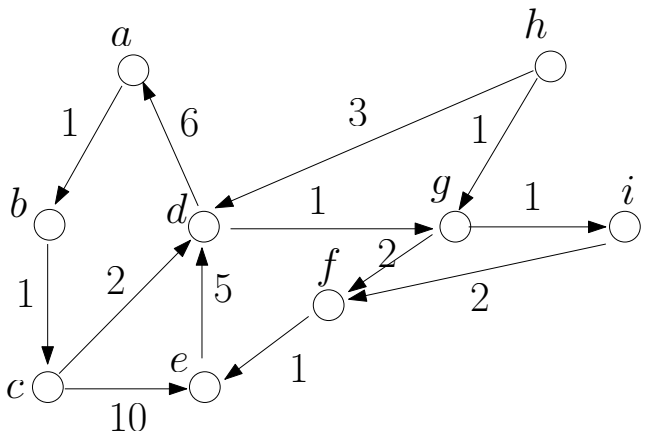


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	5	g
g	3	d
h	∞	nil
i	4	g

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

Example

Relax the out-going edges of i:

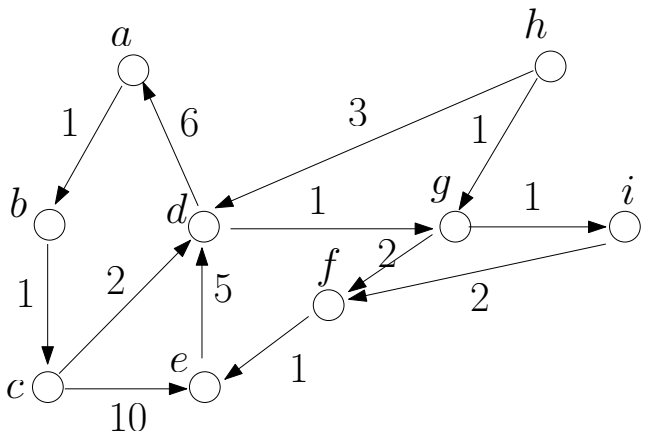


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	10	c
f	5	g
g	3	d
h	∞	nil
i	4	g

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

Example

Relax the out-going edges of f:

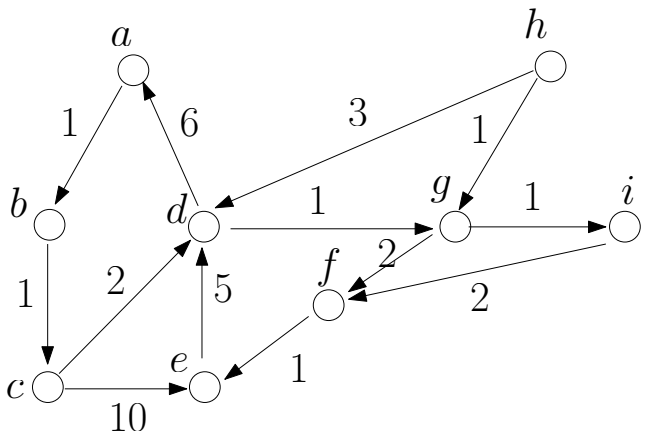


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

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

Example

Relax the out-going edges of e:

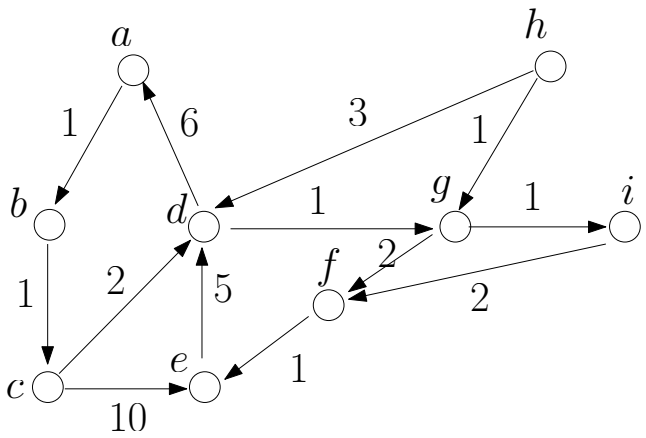


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	∞	nil
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

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

Example

Relax the out-going edges of a:

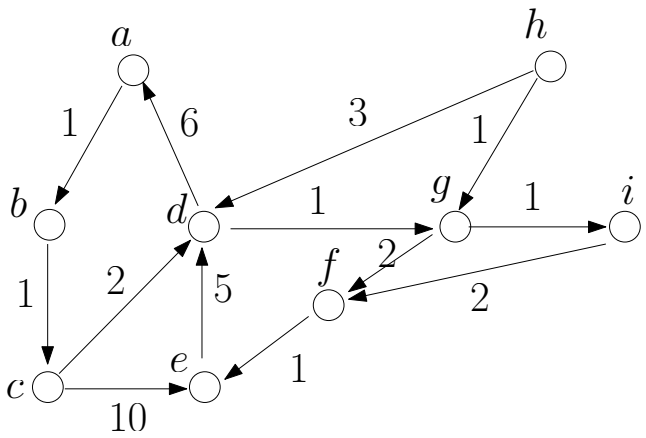


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

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

Example

Relax the out-going edges of b:

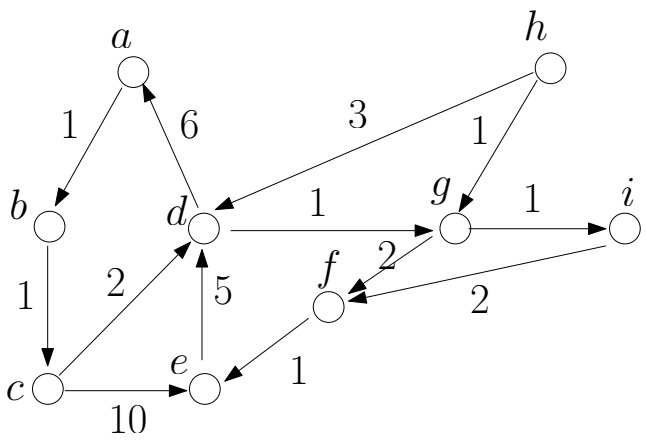


vertex v	$dist(v)$	$parent(v)$
a	8	d
b	9	a
c	0	nil
d	2	c
e	6	f
f	5	g
g	3	d
h	∞	nil
i	4	g

$S = \{h\}$.

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

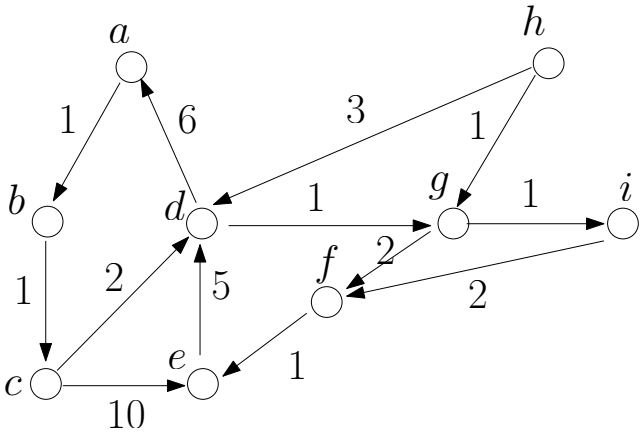
$S = \{\}$.

All the shortest path distances are now final.

Constructing the Shortest Path Tree

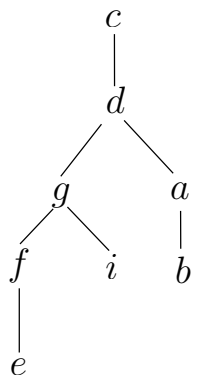
For every vertex v, if $u = parent(v)$ is not nil, then make v a child of u.

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



Correctness and Running Time

It will be left as an exercise for you to prove that Dijkstra's algorithm is correct (by extending the correctness of our DAG algorithm).

Just as equally instructive is an exercise for you to implement Dijkstra's algorithm in $O((|V| + |E|) \cdot \log |V|)$ time. You have already learned all the data structures for this purpose. Now it is the time to practice applying them.

This concludes our discussion on the SSSP problem. We have learned how to solve the problem in 3 settings:

- Unit weight for all edges and arbitrary graphs: BFS, running time $O(|V| + |E|)$.
- Arbitrary weights and DAGs: The first algorithm in this lecture, running time $O(|V| + |E|)$.
- Positive weights and arbitrary graphs: Dijkstra's, running time $O((|V| + |E|) \cdot \log |V|)$.

Remark: Using an advanced data structure (called the Fibonacci Heap) that will not be covered in this course, we can actually improve the running time of Dijkstra's algorithm to $O(|V| \log |V| + |E|)$.