Single-Source Shortest Paths

DEFINITION:

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbf{R}$ mapping edges to real-valued weights. The *weight* of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the following sum:

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

The *shortest-path weight* from u to v is defined as

$$\delta(u,v) = \begin{cases} \min\{\, w(p) \;:\; u \overset{p}{\leadsto} v\}, & \text{if there is a path from } u \text{to } v \\ \infty, & \text{otherwise.} \end{cases}$$

A *shortest path* from a vertex u to a vertex v is defined as any path p with weight $w(p) = \delta(u, v)$.

EXAMPLES: weights can be distances, times, costs, etc.

Note that breadth-first search algorithm finds shortest paths for unweighted graphs (that is, where $w(u, v) = 1$ for any edge $(u, v) \in E$).

DEFINITION:

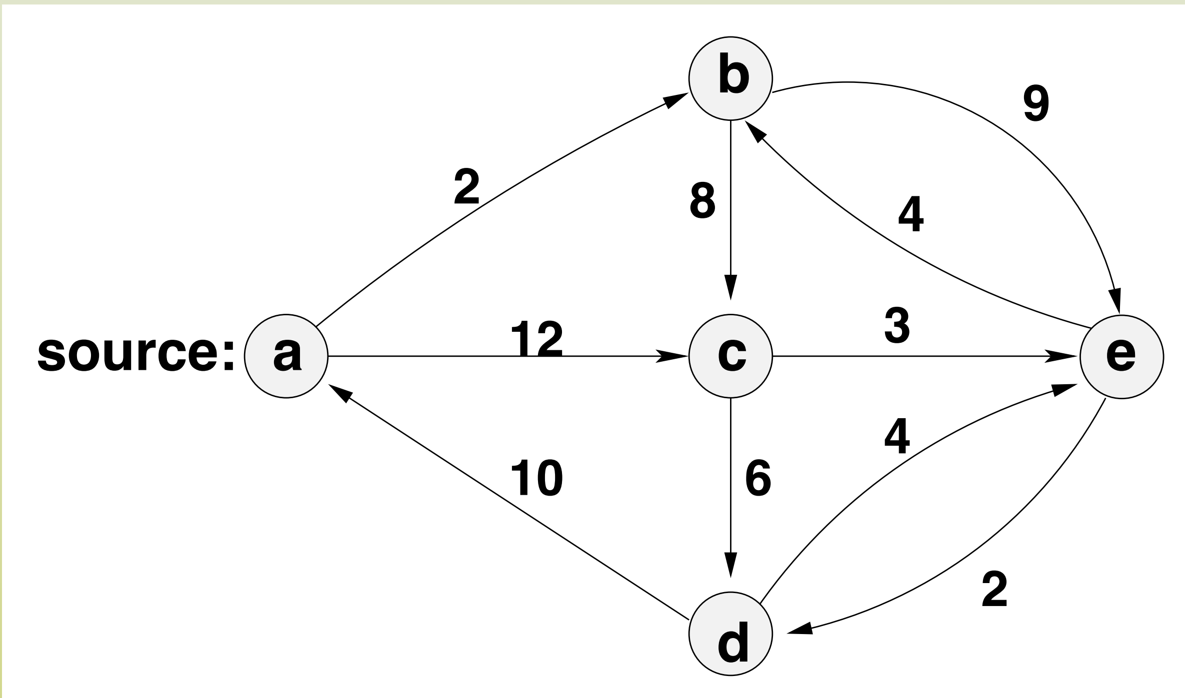
Given a graph $G = (V, E)$. In the *single-source shortest-paths problem* (SSSP) we want to find a shortest path from a given source vertex $s \in V$ to every vertex $v \in V$.

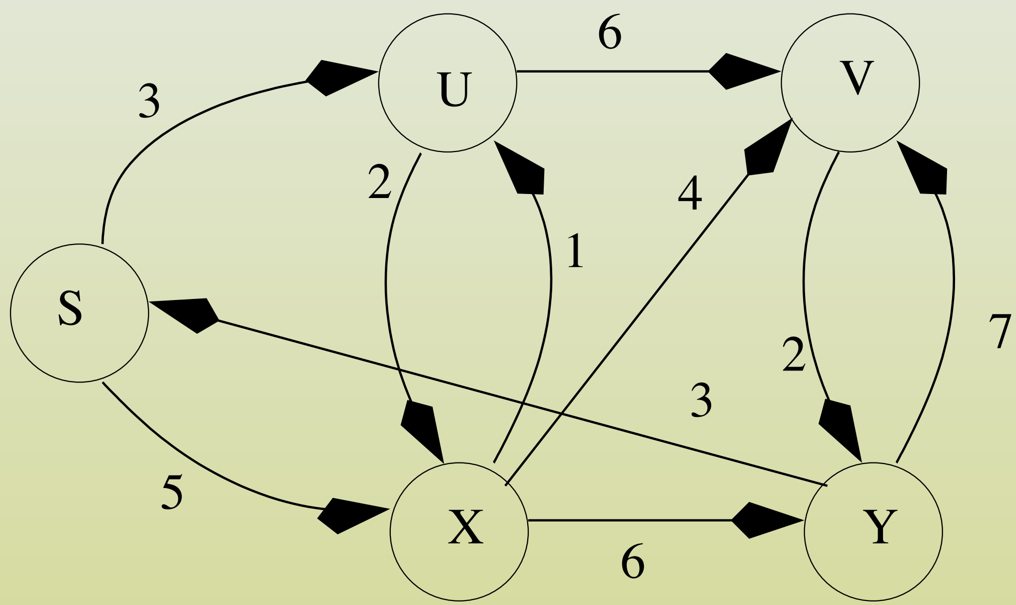
EXAMPLES:

1. Single-destination shortest-paths problem: reverse of the single-source shortest-paths problem. We need to find a shortest path to a given destination vertex f from every vertex v.

2. Single-pair shortest-path problem: subproblem of SSSP. We need to find a shortest path from u to v for given vertices u and v.

   All-pairs shortest-paths problem: collection of SSSP where source vertices are all the vertices from V. We need to find a shortest path from u to v for every pair of vertices u and v.


source:
a
b
c
d
e
2
12
10
8
6
9
4
3
4
2



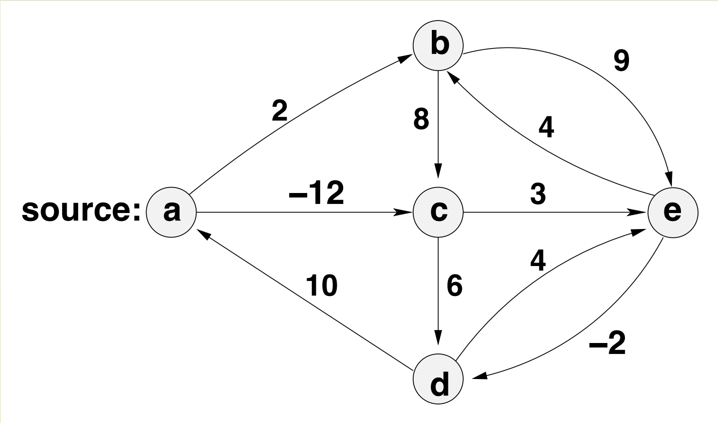
The shortest path from S to Y is not unique.

S → U → X → Y

S → U → V → Y

S → U → X → V → Y

Negative Cycles in SSSP



Find the shortest path from a to d:

- path $\langle a, c, e, d \rangle$: weight $= -12 + 3 - 2 = -11$.
- path $\langle a, c, e, d, a, c, e, d \rangle$: weight $= -12 + 3 - 2 + (10 - 12 + 3 - 2) = -12$.
- path $\langle a, c, e, d, a, c, e, d, a, c, e, d \rangle$: weight $= -12 + 3 - 2 + (10 - 12 + 3 - 2) + (10 - 12 + 3 - 2) = -13$

if we continue doing this we get decreasing sequence of weights of this path.

Therefore, its weight is $\delta(a,d) = -\infty$ (since in this way we can obtain any negative number ≤ -11).

To avoid such situations we assume that graphs have no negative-weight cycles.

- *A shortest path problem is well defined for a graph without negative cycles.*
- *Sub-paths of the shortest paths are the shortest paths.*

Let $G = (V, E)$ be directed, weighted graph and $p = (v_1, v_2, .., v_k)$ be a shortest path in G, then for every $1 \leq i \leq j \leq k$, $p_{ij} = (v_i, v_{i+1}, .., v_j)$ is the shortest sub-path of p.

p can be decomposed into $v_1 \dashrightarrow v_i \dashrightarrow v_j \dashrightarrow v_k$ and
$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$

Suppose that there exits a sub-path p'_{ij},and $w'(p'_{ij}) < w(p_{ij})$ so the weight of new path is $w(p_{1i}) + w'(p'_{ij}) + w(p_{jk}) < w(p)$ and its contradicts the assumption that p is the shortest path.

- *There are not positive-weight cycles on shortest path from the source to destination.*

Let $p = (v_0, v_1, .., v_k)$ be a shortest path and $c = (v_i, v_{i+1}, .., v_j)$ with $w(c) > 0$ and $p' = (v_0, v_1, .., v_i, v_{j+1}, v_{j+2}, .., v_k)$.
$w(p') = w(p) - w(c) < w(p)$ then p cannot be a shortest path or $w(c) = 0$.
If a shortest path has 0-weight cycles then we can remove them.

- *A shortest path is any acyclic path in a graph $G = (V, E)$ which contains at most V distinct vertices and at most $|V| - 1$ edges.*

Shortest-Path Tree

DEFINITION:

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow \mathbf{R}$, and assume that G contains no negative-weight cycles reachable from the source vertex $s \in V$. A *shortest-paths tree* rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that

V' is the set of vertices reachable from s in G.

G' forms a rooted tree with root s

for all $v \in V'$, the unique simple path from s to v in G' is a shortest path from s to v in G.

Shortest paths and shortest-paths trees are not necessarily unique. There can be two or more shortest-paths trees with the same root.

Relaxation

Further we will use the following property of shortest-path weights.

LEMMA 1:

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbf{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

For each vertex $v \in V$, we assign a field $d[v]$ (shortest-path estimate), which is an upper bound on the weight of a shortest path from source s to v. We also assign a field $\pi[v]$ which is a predecessor of v in an algorithm, or it is `NIL`.

Initialize-Single-Source(G, s):

for (each vertex $v \in V$) {
  $d[v] = \infty$;
  $\pi[v] =$ `NIL`; }
$d[s] = 0$;

Relax(u, v, w):

if ($d[v] > d[u] + w(u, v)$) {
  $d[v] = d[u] + w(u, v)$;
  $\pi[v] = u$; }

Dijkstra's Algorithm

Dijkstra's algorithm solves SSSP problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are **nonnegative**. We assume here that $w(u, v) \geq 0$ for each edge $(u, v) \in E$.

This algorithm uses a set S of vertices for which the shortest-path weights from the source vertex s have been determined. That is, if $v \in S$, then $d[v] = \delta(s, v)$. A priority queue Q contains all the vertices in $V - S$, ordered by d values. In this implementation it is assumed that G is represented by adjacency lists.

Dijkstra(G, w, s):

Initialize-Single-Source(G, s);

$S = \emptyset$;

initialize Q to contain all $v \in V$;

`while` ($Q! = \emptyset$) {

  $u =$ Extract-Min(Q);

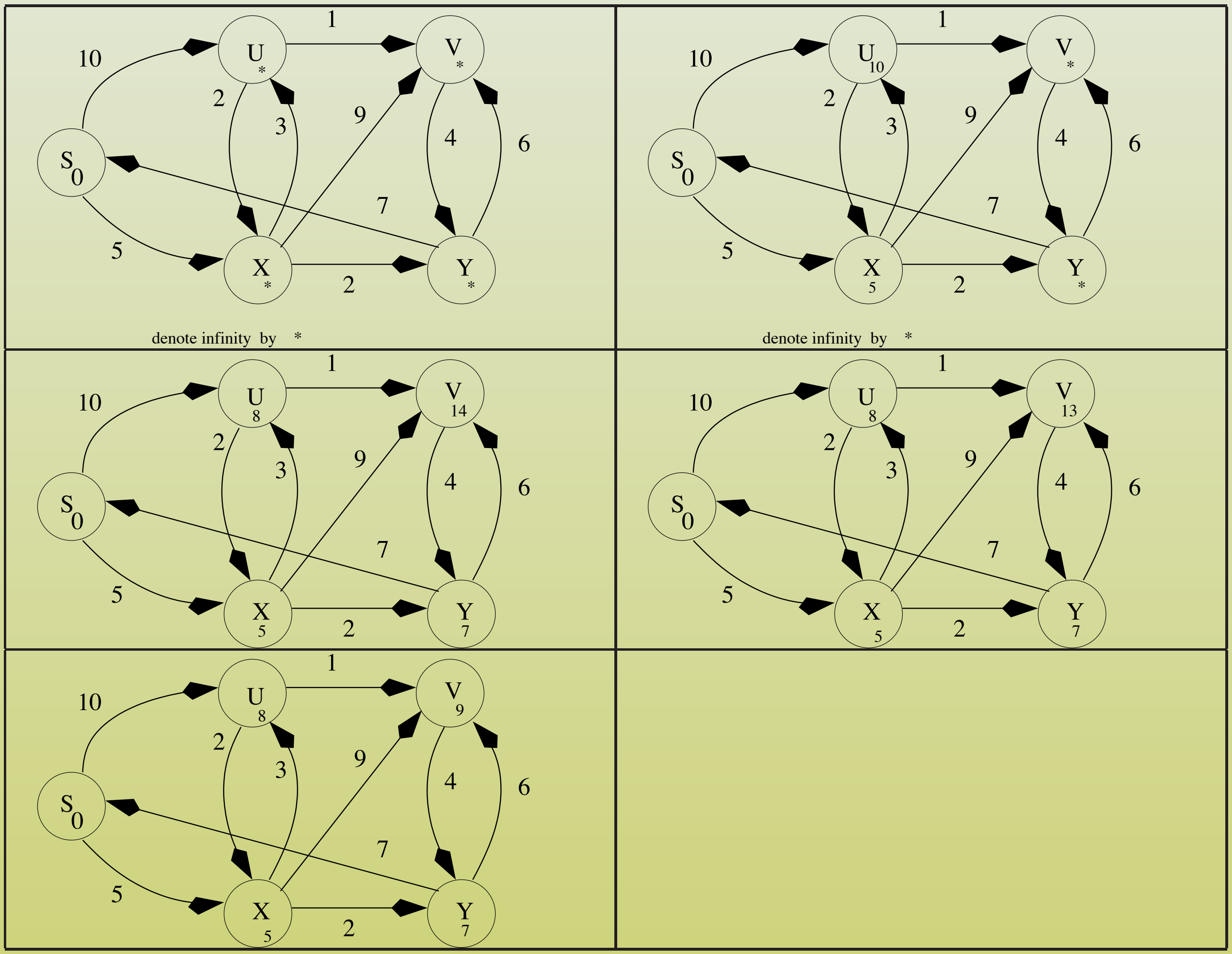
  $S = S \cup \{u\}$;

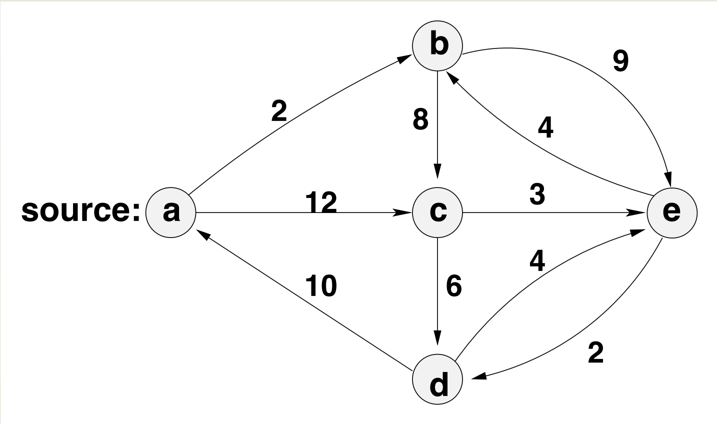
  `for` (each vertex $v \in Adj[u]$)

    Relax(u, v, w);

} `/* end of while */`

Examples





iter	$\Pi[v]$	$d[a]$	$d[b]$	$d[c]$	$d[d]$	$d[e]$
0	–	0	∞	∞	∞	∞
1	a	0	2(a)	12 (a)	∞	∞
2	b	0	2 (a)	10 (b)	∞	11(b)
3	c	0	2 (a)	10(b)	16 (c)	11(b)
4	e	0	2(a)	10(b)	13(e)	11(b)
5	d	0	2(a)	10(b)	13(e)	11(b)

Correctness

THEOREM:

If we run Dijkstra's algorithm on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, then at termination, $d[u] = \delta(s, u)$ for all vertices $u \in V$.

COROLLARY:

If we run Dijkstra's algorithm on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, then at termination, the predecessor subgraph $G_\pi = (V_\pi, E_\pi)$ is a shortest-paths tree rooted at s. Notation:
$V_\pi = \{v \in V \ : \ \pi[v] \neq \texttt{NIL}\} \cup \{s\}$ and
$E_\pi = \{(\pi[v], v) \in E \ : \ v \in V_\pi - \{s\}\}$.

Analysis

Assume that the priority queue Q is implemented as a binary heap.

Initialization (steps 1–3) takes $O(V)$ time. The time to build the binary heap is $O(V)$ (step 3).

The `while` loop has $|V|$ iterations, since after each vertex has been extracted from Q it is inserted in S, and is never inserted back in Q.

The `for` loop is executed $|E|$ times overall, since each edge in the adjacency list $Adj[v]$ is examined exactly once during the course of the algorithm and the total number of edges in all adjacency lists is $|E|$.

Extract-Min takes $O(\lg V)$ time. There are $|V|$ such operations.

After the assignment $d[v] = d[u] + w(u, v)$ (in Relax) we must heapify the priority queue Q which takes $O(\lg V)$ time. There are at most $|E|$ such operations.

Total running time: $O(V \lg V + E \lg V) = O((V + E) \lg V)$.

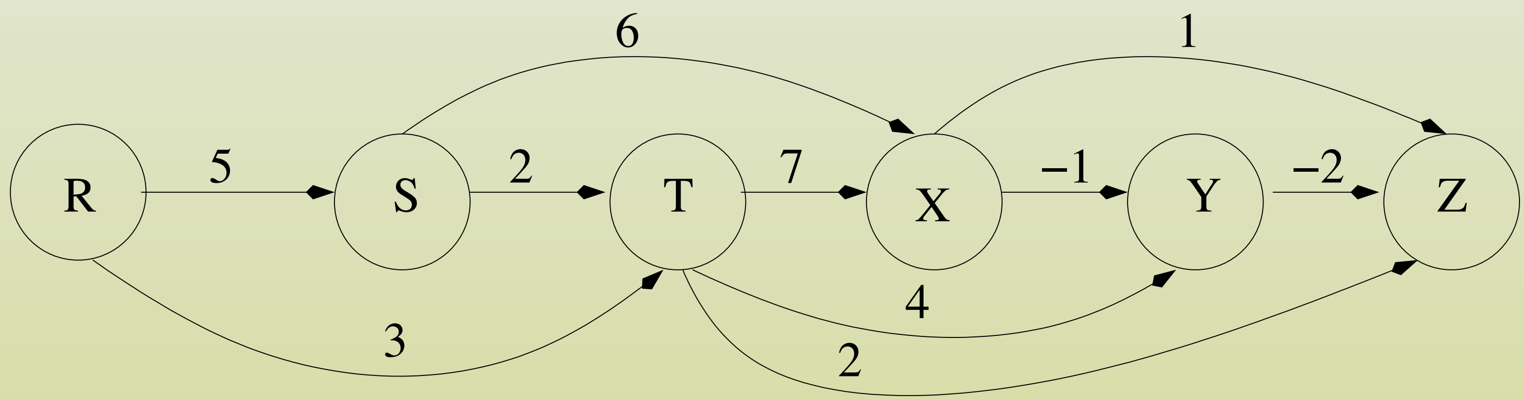
SSSPs in a DAG

Let $G = (V, E)$ be a directed weighted, weighted, topologically sorted graph. The DAG shortest path algorithm computes the shortest path in $O(V + E)$.

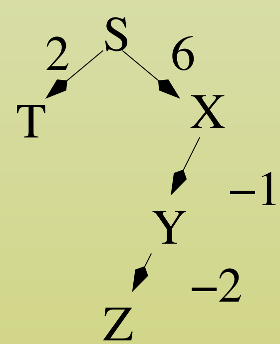
DAG-Shortest-Parth-Algorithm(G,u,s)

```
Topologically sort the vertices of G
Initialize-Single-Source(G,s)
   for each vertex in topologically sorted order
      do for each vertex v ∈ Adj[u]
         do RELAX(u,v,w)
```

Example.



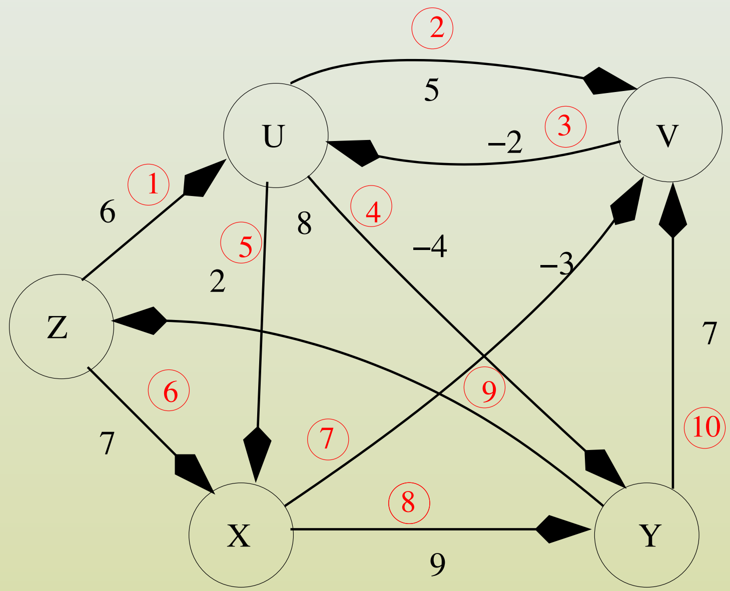
S	R	T	X	Y	Z
0	∞	∞	∞	∞	∞
	∞	2	6	∞	∞
				6	4
				5	4
					3



Bellman-Ford(G,w,s)

```
Initialize-Single-Source(G,s)
for i=1 to |V[G]-1|
   do for each edge (u,v)∈ E[G]
      do RELAX(u,v,w)
for each edge (u,v)∈ E[G]
   do if d[v]>d[u]+w(u,v)
       then return FALSE
return TRUE
```

Running time $O(VE)$ for sparse graph and $O(V^3)$ for dense graphs.



	0	1	2	3	4
z	0	0	0	0	0
x	∞	14/7	7	7	7
u	∞	6	6	2	2
v	∞	11/9	4	4	4
y	∞	2	2	-2	-2