

Single-Source Shortest Paths

We are given a connected, weighted, directed graph $G(V, E)$ and a weight function $w: E \rightarrow \mathbb{R}$ mapping edges to real-valued weights.

The **single-source shortest paths** (SSP) problem is to find a shortest path from a given source r to every other vertex $v \in V - \{r\}$.

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

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

The weight of a shortest path from u to v is defined by

$$\delta(u, v) = \min\{w(p): p \text{ is a path from } u \text{ to } v\}.$$

Variants of the SSP problem

- Single-destination shortest paths problem
- Single-pair shortest path problem
- All-pairs shortest paths problem

In some applications, there may be edges whose weights are **negative**.

If there is a negative weight cycle which is reachable from the source s , to some vertex v , $\delta(s, v)$ is **undefined**.

See Fig. 24.1 (page 583).

Representation of a Shortest Path Tree

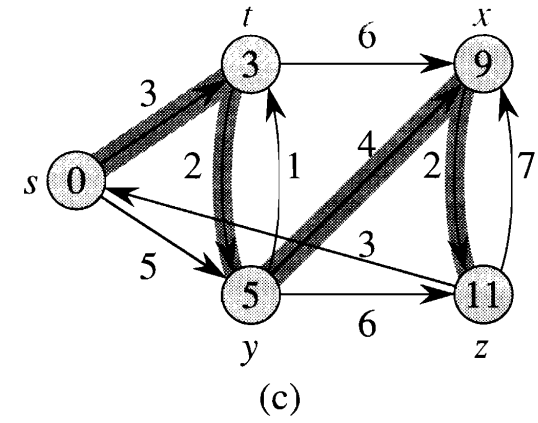
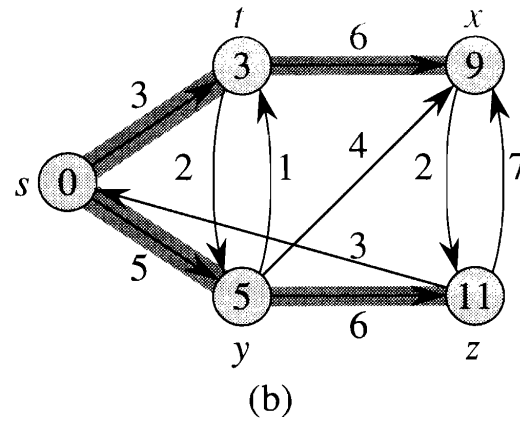
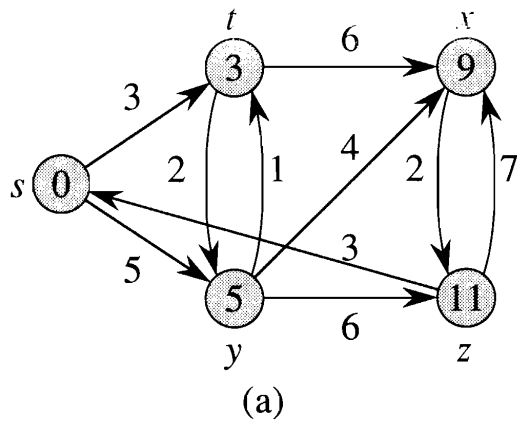
The shortest path tree rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$ such that

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

For each vertex $v \in V$, we maintain a predecessor $\pi(v)$ that is either a vertex or NIL.

Thus, the output for the single-source shortest path problem is **a tree rooted at the source**.

Shortest Path Tree Example



Algorithms for the SSP Problem

We will consider two well-known algorithms.

- Dijkstra's algorithm,
which assumes that all the edges in G are nonnegative.
- The Bellman-Ford algorithm,
which allows negative-weight edges in G and will produce the correct results as long as there are no negative weight cycles reachable from the source. Typically if there is such a negative cycle, the algorithm can detect and report its existence.

Relaxation

The main technique used by all shortest path algorithms is **relaxation** (a method that repeatedly **decreases** an upper bound on the length of an actual shortest path for each vertex until **the upper bound equals the length of the shortest path**).

For each vertex v , we maintain an attribute $d[v]$ which is an upper bound on the length of a shortest path from source s to v ($d[v]$ is also called the **shortest path estimate**).

Relax(u, v, w)

- 1 if $d[v] > d[u] + w(u, v)$
- 2 then $d[v] = d[u] + w(u, v)$
- 3 $\pi[v] = u$

See Fig 24.3 (page 586).

Dijkstra's Algorithm

Dijkstra's algorithm maintains a set S of vertices whose final shortest path weights from the source s have already been determined.

For every vertex $v \in S$, we have $d[v] = \delta(s, v)$ already.

The algorithm repeatedly selects a vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S and relaxes all the edges leaving u .

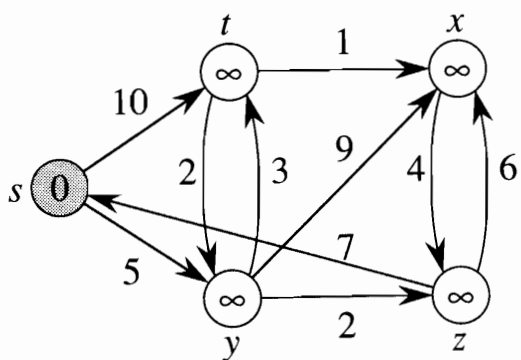
A priority queue Q that contains all the vertices in $V - S$ is maintained, keyed by their $d[]$ values.

Dijkstra's Algorithm

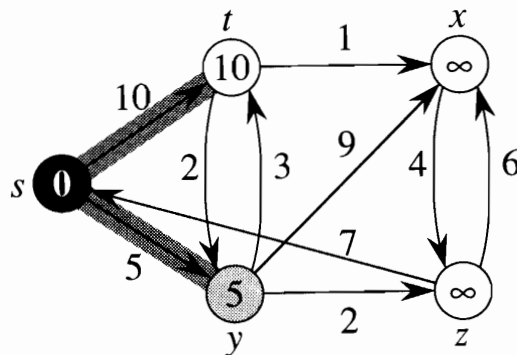
Dijkstra_Short(G, w, s)

```
1    $d[s] \leftarrow 0$ ;  
2    $\pi[s] \leftarrow NIL$   
3   for all  $v \in V - \{s\}$  do  
4        $d[v] \leftarrow \infty$ ;  
5        $\pi[v] \leftarrow NIL$   
6    $S \leftarrow \emptyset$   
7    $Q \leftarrow V$   
8   while  $Q \neq \emptyset$  do  
9        $u \leftarrow \text{Extract\_Min}(Q)$   
10       $S \leftarrow S \cup \{u\}$   
11      for each vertex  $v \in \text{Adj}[u]$  do  
12          Relax( $u, v, w$ )
```

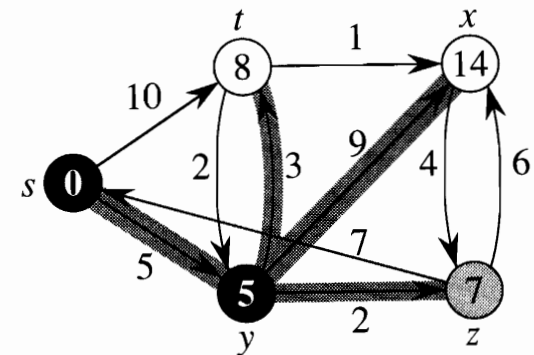
Dijkstra's Algorithm Example



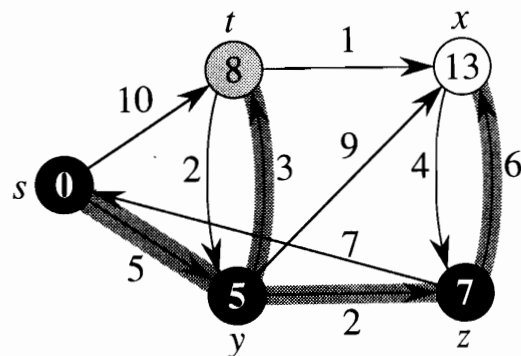
(a)



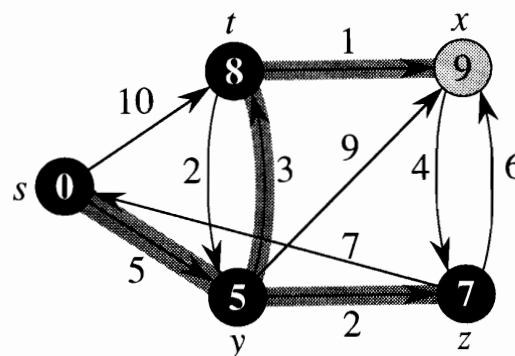
(b)



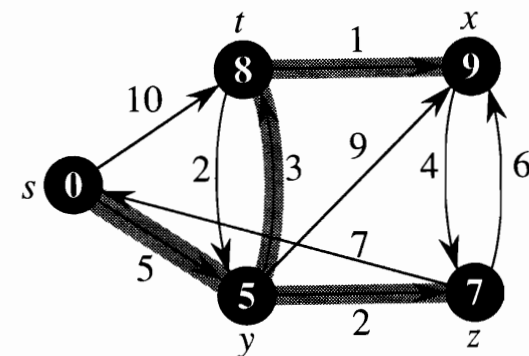
(c)



(d)



(e)



(f)

Correctness of Dijkstra's algorithm

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

Sketch of the proof:

It suffices to show for each vertex $u \in V$, we have $d[u] = \delta(s, u)$ at the time when u is added to S .

Once we show $d[u] = \delta(s, u)$, we rely on the upper bound property to show the equality holds at all times thereafter.

Initially $S = \emptyset$.

We wish to show that in each iteration $d[u] = \delta(s, u)$ for the vertex added to set S .

Correctness of Dijkstra's algorithm

Assume that u is the first vertex that, when it is added to S , $d[u] \neq \delta(s, u)$.

$u \neq s$ and $S \neq \emptyset$ just before u is added to S - there must be a path from s to u .

Otherwise $d[u] = \delta(s, u) = \infty$, which violates our assumption that $d[u] \neq \delta(s, u)$.

Let p be a shortest path from s to u and y be the first vertex in $V - S$ on p while x is the last vertex in S on p .

Then, p can be decomposed into $s \rightsquigarrow^{p_1} x \rightarrow y \rightsquigarrow^{p_2} u$.

Following the assumption that $d[y] = \delta(s, y)$, and $\delta(s, y) \leq \delta(s, u)$.

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

but both y and u are in $V - S$ when u is chosen, so $d[u] \leq d[y]$.

Thus, $d[y] = \delta(s, y) = \delta(s, u) = d[u]$, which gives us a contradiction.

The time complexity of Dijkstra's algorithm

- Use a linear array to implement Q , $\rightarrow O(n^2 + m) = O(n^2)$.
- Use a binary heap to implement Q , $\rightarrow O(n \log n + m \log n)$.