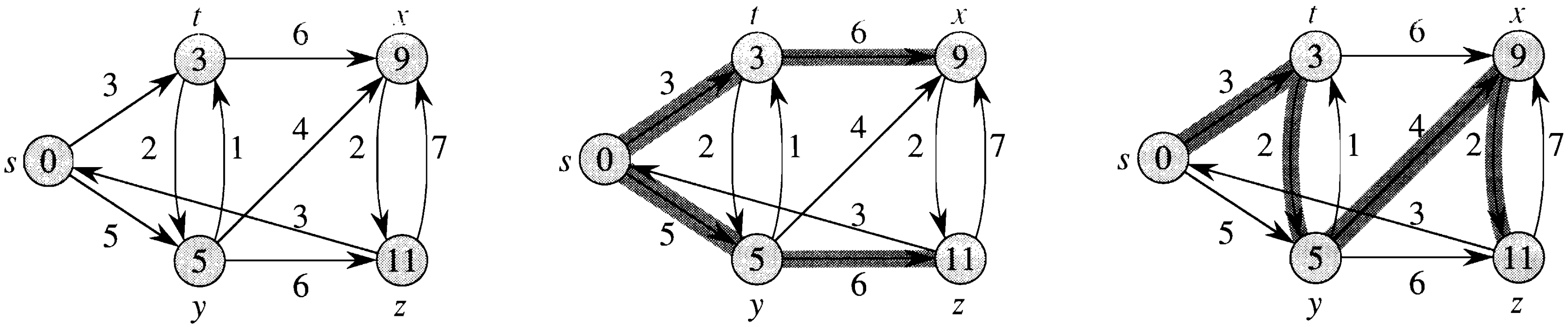
Chapter 24. Shortest path problems

We are given a directed graph $G = (V, E)$ with each directed edge $(u, v) \in E$ having a weight, also called a *length*, $w(u, v)$ that may or may not be negative.

A *shortest path* between two vertices u and v is a directed path from u to v that has **the least total length**. This total length is also referred to as the *distance from u to v*, denoted by $\delta(u, v)$. If there is no such a directed path, define $\delta(u, v) = \infty$.



(a) (b) (c)

We assume that G does not contain negative cycles reachable from the source, since otherwise there are arbitrarily short paths and $\delta(u, v)$ is undefined. Why?

Chapter 24. Shortest path problems

Given a connected, weighted directed graph $G(V,E;w)$, associated with each edge $< u,v > \in E$, there is a weight $w(u,v)$.

The single source shortest paths (SSP) problem is to find a shortest path from a given source s to every other vertex $v \in V - \{s\}$.
The length of a path $p =< v_0, v_1, \ldots, v_k >$ is the sum of the weights of its constituent edges $\langle v_0, v_1\rangle, \langle v_1, v_2\rangle, \ldots, \langle v_{i-1}, v_i\rangle, \ldots, \langle v_{k-1}, v_k\rangle$, i.e.,

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

where $s = v_0$.

The length of a shortest path in G from u to v is defined by
$\delta(u,v) = \min\{\ w(p) \colon\ p$ is a path in G from $u\ to\ v\}$.

24. Various shortest path problems

- **Single-pair shortest path problem**: Given two vertices u and v, find a shortest path from u to v. (It is not known how to solve this faster than the next problem)

- **Single-source shortest path problem**: Given a *source vertex* s, find a shortest path from s to every other vertex.

- **Single-destination shortest path problem**: Given a *destination vertex* t, find a shortest path to t from every other vertex. (Just reverse the edge directions and solve the previous problem.)

- **All-pairs shortest paths problem**: Find a shortest path between each pair of vertices.

Chapter 24. Shortest path problems

For a special case where all the edge lengths in a graph G are the same, the shortest paths are just the paths with the least edges.

In this case the single-pair and single-source problems are solved with Breadth-First Search (BFS), which takes $O(|V| + |E|)$ time, an optimal algorithm.

Applying BFS starting at each vertex solves the all-pairs shortest paths problem in time $O(|V|^2 + |V||E|)$.

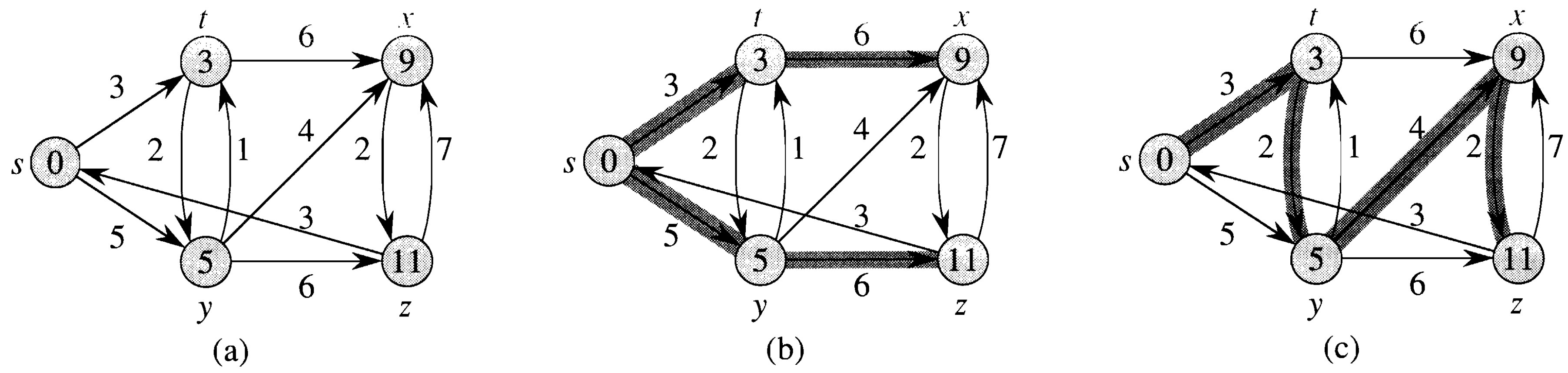
24. The shortest path tree

Consider a source vertex s. A shortest path 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;
- G' forms a rooted tree with root s and edges directed away from the root;
- 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.

Exercise: Show that directed graphs with no negative-length cycles have a shortest path tree. **Hint:** Subpaths of shortest paths are shortest paths.

24.2 Shortest path tree example



(a) An example of a directed graph with edge lengths.

(b) A shortest path tree for root s.

(c) Another shortest path tree for root s.

A single-source shortest path tree in G is not unique!

24. Algorithms for the shortest path problem

We will introduce 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. If there is such a negative cycle, the algorithm can detect and report its existence.

24. Relaxation

An important technique used by most shortest path algorithms is relaxation.

If the distance (also the length of the shortest path) from s to u is at most L_1, and the length of the edge from u to v is L_2, then the distance from s to v is at most $L_1 + L_2$.

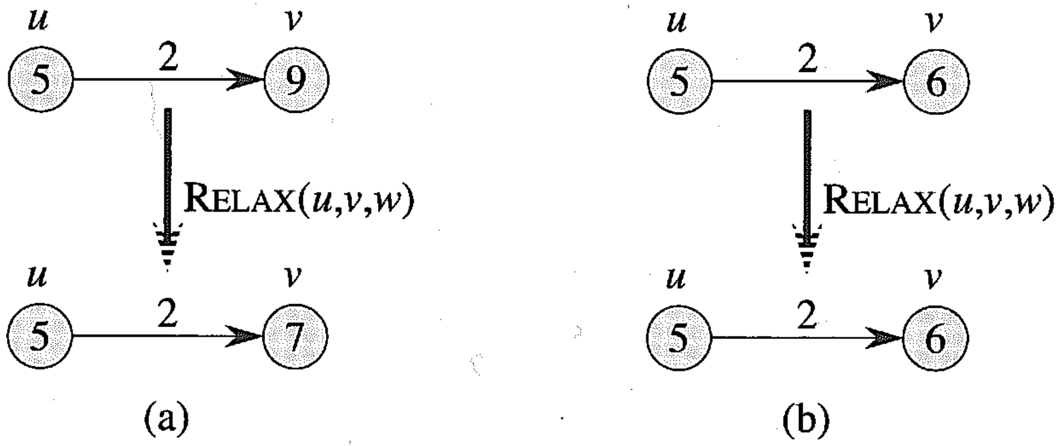
For each vertex v, we maintain an attribute $v.d$, which is the length of some known path from s to v (or ∞ if no such path is known). This is an upper bound on the length of the shortest path.

We also maintain a pointer $v.\pi$ which will point back towards s along this known path.

Relax(u, v, w) /* (u, v) is an edge and $w(u, v)$ is its weight */

1 **if** $v.d > u.d + w(u, v)$ **then**
2 $v.d \leftarrow u.d + w(u, v)$;
3 $v.\pi \leftarrow u$.

24. Relaxation (continued)



The current estimates (upper bounds) of the distance from the source vertex are shown within the vertices.

In Fig. (a), since the distance from the source to vertex u is at most 5 and $w(u,v) = 2$, the distance from the source to v is at most 7. This improves the previous estimate for v, so we also set $v.\pi = u$.

In Fig.(b), the same logic does not improve the estimate for v. (We say that (u,v) "cannot be relaxed" at the moment.)

24. Relaxation "algorithm"

Relaxation$(G = (V, E), s, w)$ /* single-source shortest paths */

for each $v \in V$ **do**
  $v.d \leftarrow \infty$;
  $v.\pi \leftarrow NIL$;
$s.d \leftarrow 0$;
while there is any edge (u, v) that can be relaxed **do**
  Relax(u, v, w)

If this algorithm terminates, $v.d = \delta(s, v)$ for all v (why?), and the edges $\{(v.\pi, v) : v \in V - \{s\}\}$ describe a shortest path tree (Lemma 24.17, Cormen).

The problem is that many iterations may be needed.
(Is it even clear that the algorithm must terminate?)

The idea for better algorithms is to choose the edges to relax in a clever order, so that only a limited number of relaxations are needed.

24.3 Dijkstra's Algorithm

- Dijkstra's algorithm maintains a set S of vertices whose shortest path lengths from the source s have already been determined:
  for every vertex $v \in S$, we have $v.d = \delta(s, v)$ already.
- The algorithm repeatedly selects a vertex $u \in V \setminus S$ with least $u.d$, inserts u into S, and relaxes all the edges leaving u.
- A min-priority queue Q that contains all the vertices in $V \setminus S$ is maintained, keyed by their d attribute, which is used for the selection of vertex u

24.4 Dijkstra's Algorithm (continued)

```
Dijkstra(G, w, s)
    s.d ← 0;
    s.π ← NIL;
    for each v ∈ V \ {s} do
        v.d ← ∞;
        v.π ← NIL;
    S ← ∅;
    Q ← V; /* Q is a MIN-HEAP */
    while Q ≠ ∅ do
        u ← Extract_Min(Q);
        S ← S ∪ {u};
        for each vertex v ∈ G.Adj[u] do
            Relax(u, v, w);
            if the value of v.d has been changed
                then Heapify Q;
            /* due to the Decrease_key operation if v.d does decrease */
```

24.3 Correctness of Dijkstra's algorithm

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

Sketch of the proof:

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

After $u.d = \delta(s,u)$ becomes true, it remains true, since relaxing edges can only decrease $u.d$, and finite $u.d$ is always the length of *some* path from s to u.

Therefore, if $(*)$ is true, all vertices $u \in S$ have $u.d = \delta(s,u)$ and the algorithm must be correct since $S = V$ at the end.

First note that $(*)$ is true for the first vertex put into S, which is s.

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

24.3 Correctness of Dijkstra's algorithm (continued)

Suppose $(*)$ is not true, and suppose u is the first vertex for which $u.d \neq \delta(s,u)$ when u is added to S.

There must be a path from s to u otherwise $u.d = \delta(s,u) = \infty$.

Let p be a shortest path from s to u.

Let y be the first vertex in $V - S$ on p and let x be the previous vertex on p.

Then, p can be decomposed into $s \overset{p_1}{\leadsto} x \to y \overset{p_2}{\leadsto} u$, where everything up to x is in S and both u and y are not in S.

Since subpaths of shortest paths are shortest paths, $s \overset{p_1}{\leadsto} x \to y$ is a shortest path. When x was put into S, $x.d = \delta(s,x)$ and (x,y) was relaxed, so $y.d = \delta(s,y)$.

24.3 Correctness of Dijkstra's algorithm (cont.)

Now, we see

$$
\begin{aligned}
u.d &\geq \delta(s,u) && \text{since } u.d \text{ is always an upper bound} \\
&\geq \delta(s,y) && \text{since } u \text{ is further along a shortest path than } y \\
&= y.d && \text{just proved above} \\
&\geq u.d && \text{or else } y \text{ would have come out } Q \text{ before } u
\end{aligned} \tag{1}
$$

Therefore, all these values must be equal. In particular $u.d = \delta(s,u)$, which contradicts our assumption about u.

Step (1) is where we assumed the edges have nonnegative values.

If this assumption is not true, Dijkstra's algorithm is not applicable.

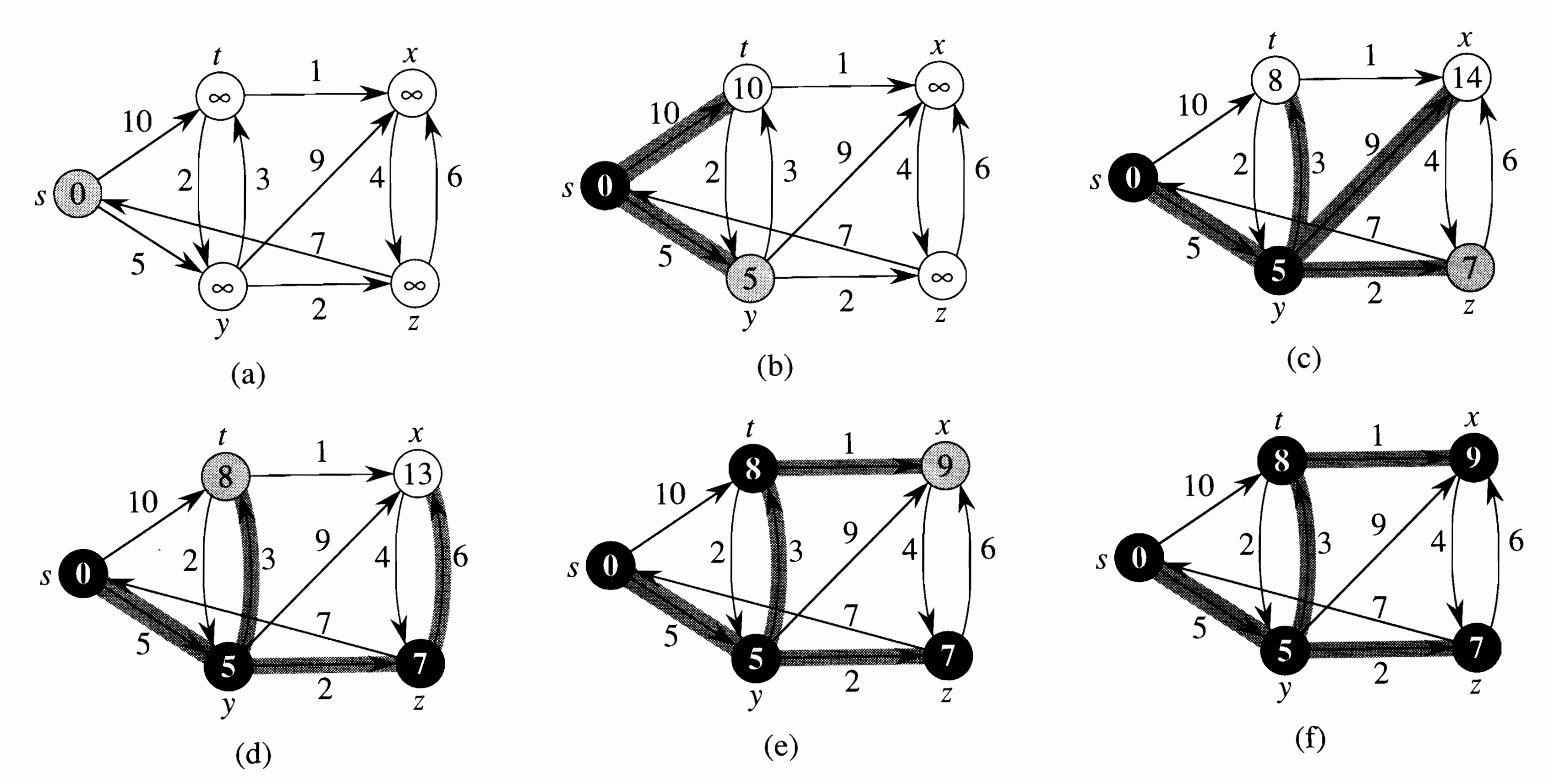
24.3 The time complexity of Dijkstra's algorithm

Each vertex enters the minimum queue at Step 7 and leaves at Step 9.
So there are $|V|$ Extract_Min operations.

Each (u,v) is relaxed exactly once: when u is added to S.
So there are E Relax and resulting Decrease_key operations.

- If we use a linear array to implement Q, Extract_Min takes $O(|V|)$ time and Decrease_key each take $O(1)$ time. So, the total is $O(|V|^2 + |E|) = O(|V|^2)$.

- If we use a minimum heap to implement Q, then Extract_Min and Decrease_key both take $O(\log|V|)$ time. So the total is $O(|V|\log|V| + |E|\log|V|)$.


t
x
1
10
∞
∞
s
0
2
3
9
4
6
5
7
∞
∞
2
y
z
(a)
t
x
1
10
10
∞
s
0
2
3
9
4
6
5
7
5
∞
2
y
z
(b)
t
x
1
10
8
14
s
0
2
3
9
4
6
5
7
5
7
2
y
z
(c)
t
x
1
10
8
13
s
0
2
3
9
4
6
5
7
5
7
2
y
z
(d)
t
x
1
10
8
9
s
0
2
3
9
4
6
5
7
5
7
2
y
z
(e)
t
x
1
10
8
9
s
0
2
3
9
4
6
5
7
5
7
2
y
z
(f)