24.1 The Bellman-Ford Algorithm

The Bellman-Ford algorithm solves the single-source shortest paths problem in more general settings.

- Unlike Dijkstra's algorithm, it allows edges of negative length. However, it takes much longer time.

- Unlike Dijkstra's algorithm that adopts the greedy policy, the Bellman-Ford algorithm adopts Dynamic Programming technique, progressively decreasing the estimate of $v.d$ the distance from s to node v until the estimate is precise.

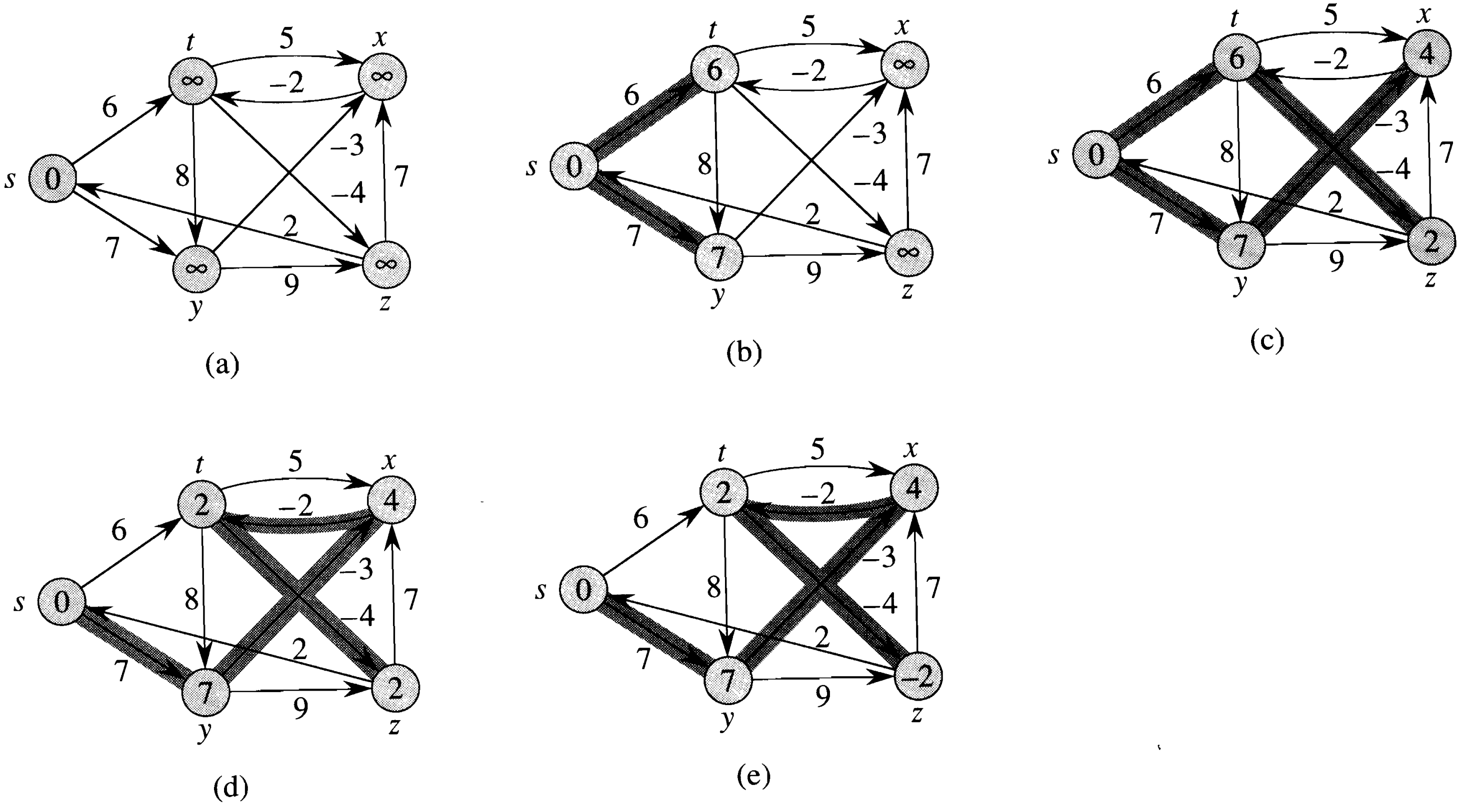
The algorithm returns **true** if and only if the graph does not contain any negative cycles that are reachable from the source.

24.1 The Bellman-Ford Algorithm (continued)

```
Bellman_Ford(G, w, s)
    s.d ← 0;
    s.π ← NIL;
    for all v ∈ V \ {s} do
        v.d ← ∞;
        v.π ← NIL;
    for i ← 1 to |V| − 1 do
        for each edge (u, v) ∈ E do
            Relax(u, v, w);
    for each edge (u, v) ∈ E do
        if v.d > u.d + w(u, v) then      /* i.e., (u, v) can still be relaxed */
            return false
    return true.
```

The running time of algorithm `Bellman_Ford` is $O(|V||E|)$.

24.1 The Bellman-Ford Algorithm (continued)



This example is from page 652 of the textbook. Here, each iteration relaxes the edges in the order (t,x), (t,y), (t,z), (x,t), (y,x), (y,z), (z,x), (z,s), (s,t), (s,y). Figures (b)-(e) show the result after each iteration.

24.1 The Bellman-Ford Algorithm (continued)

Proof of the correctness.

As usual with relaxation, $v.d$ can only decrease, and if $\delta(s, v)$ is defined (there are no negative cycles reachable from the source), we always have $v.d \geq \delta(s, v)$.
A phase or a pass is one iteration of the **for** loop of lines 6–8, where each edge is relaxed once.

Case (1): Suppose there is no negative cycle reachable from s.

Consider some shortest path $s \rightarrow v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_{k-1} \rightarrow v_k$.
We can assume $k \leq |V| - 1$.

* When (s, v_1) is relaxed in the 1st phase, $v_1.d$ is set to $\delta(s, v_1)$ if it isn't already.
* When (v_1, v_2) is relaxed in the 2nd phase, $v_2.d$ is set to $\delta(s, v_2)$ if it isn't already.
* $\cdots$
* When (v_{k-1}, v_k) is relaxed in the kth phase, $v_k.d$ is set to $\delta(s, v_k)$ if it isn't already.

So, $v.d = \delta(s, v)$ for all v after $|V| - 1$ phases, and no edges are still relaxable.

24.1 The Bellman-Ford Algorithm (continued)

Case (2): Suppose there is a negative cycle reachable from s.

Say the cycle is $v_0 \to v_1 \to v_2 \to \cdots \to v_k = v_0$.

Consider the situation after $|V|-1$ phases. Note that at this point, all the $v.d$ values of vertices in the cycle are **finite**.

By contradiction: suppose that none of the edges on the cycle are relaxable. That is,

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i) \quad \text{for } i = 1, \ldots, k.$$

Summing this inequality over $i = 1, \ldots, k$, we find a contradiction since the sum of $w(v_{i-1}, v_i)$ is negative by the assumption.

Therefore, some edge of the negative cycle is still relaxable.

24.8 Special Shortest Paths Problems

- SSP in a DAG
- Special linear programming

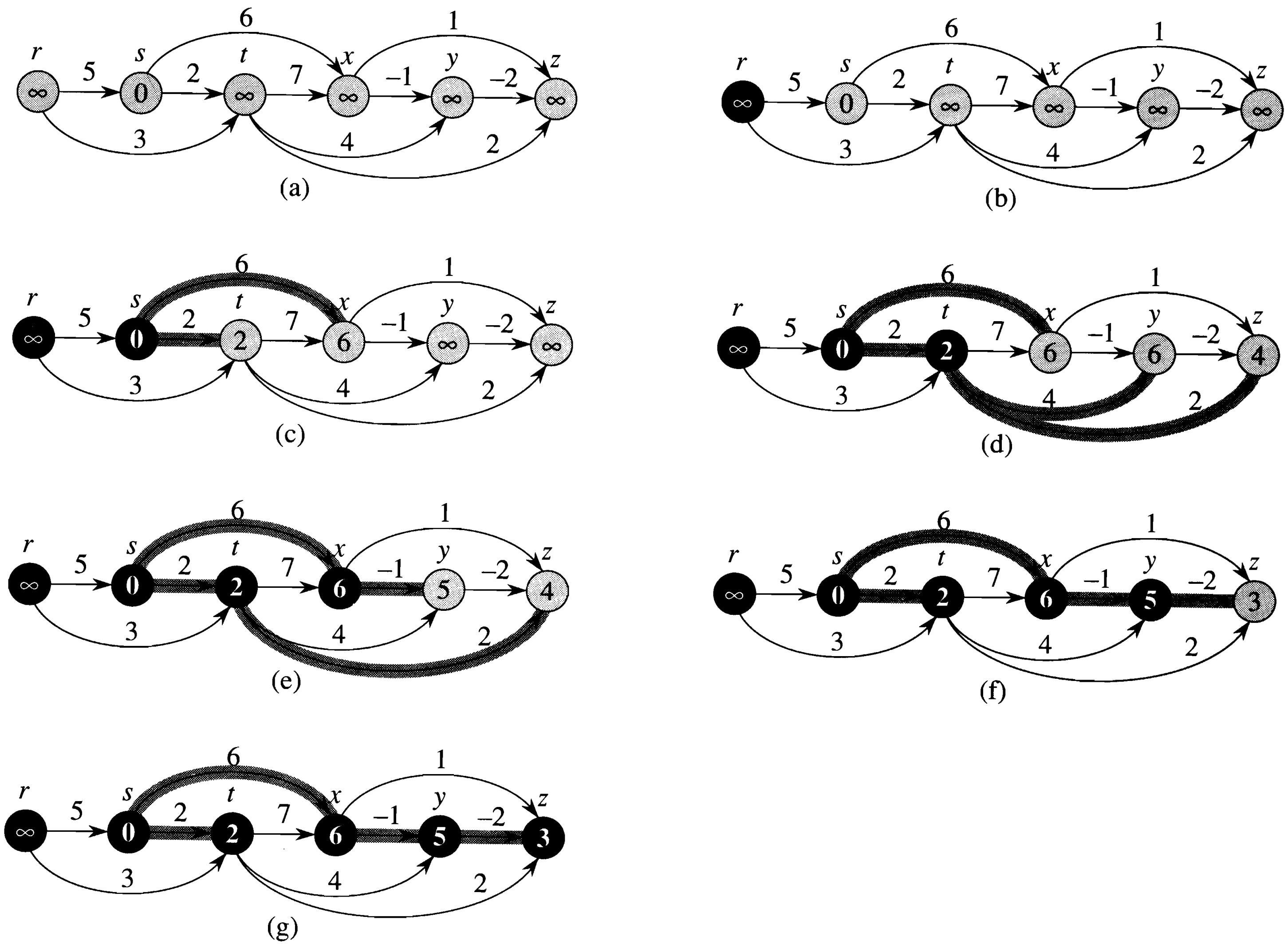
24.2 Single-Source Shortest Paths in DAGs

For a directed acyclic graph (DAG), we can relax the edges according to the topological order of their start vertices, from left to right.

DAG_Shortest_Paths(G, w, s)

$s.d \leftarrow 0$;
$s.\pi \leftarrow NIL$;
for all $v \in V - \{s\}$ **do**
    $v.d \leftarrow \infty$;
    $v.\pi \leftarrow NIL$;
determine the topological order of each vertex $v \in V$, using the DFS technique;
for each vertex u in increasing topological order **do**
    for $v \in G.Adj[u]$ **do**
        Relax(u, v, w).

The time complexity of algorithm `DAG_Shortest_Paths` is $O(|V| + |E|)$.



This example is from page 656 of our textbook.

24.2 Shortest Paths in DAGs (continued)

Proof of the correctness.

Consider any shortest path $s \to v_1 \to v_2 \to \cdots \to v_{k-1} \to v_k$.
The algorithm relaxes the edges from left to right.

- When (s, v_1) is relaxed, $v_1.d$ is set to $\delta(s, v_1)$ (note that it was ∞ before this point).
- When (v_1, v_2) is relaxed, $v_2.d$ is set to $\delta(s, v_2)$ if it isn't already.
- $\ldots$
- When (v_{k-1}, v_k) is relaxed, $v_k.d$ is set to $\delta(s, v_k)$ if it isn't already.

So $v.d = \delta(s, v)$ for all v by the time all edges are relaxed.

24.4 Difference constraints

Suppose we have to schedule n tasks $T_1, T_2, \ldots, T_n$, and we have a set of constraints like these:

T_3 must be done at least 15 minutes after T_7

T_2 must be done before T_9

T_2 must be done at least 5 minutes before T_4

T_5 must be done at most 10 minutes after T_1

$\vdots$ We wish to know if this arrangement is possible, and if so, find a schedule.

If T_i is scheduled at time x_i, then the above constraints can be written as:

$$x_7 - x_3 \leq -15$$

$$x_9 - x_2 \leq 0$$

$$x_2 - x_4 \leq -5$$

$$x_5 - x_1 \leq 10$$

24.4 Difference constraints (continued)

In general, we have real variables $x_1, x_2, \ldots, x_n$, and some numbers of constraints of the form $x_j - x_i \leq b_k$.

We will define a weighted graph $G = (V, E, w)$, called the constraint graph.

There are $n+1$ vertices $V = \{v_0, v_1, v_2, \ldots, v_n\}$.

There is a directed edge (v_0, v_i) of length 0 from v_0 to v_i for all i with $i = 1, 2, \ldots, n$.

For each constraint $x_j - x_i \leq b_k$, there is a directed edge (v_i, v_j) from v_i to v_j with length b_k.

Interesting fact: If the constraint graph has a negative cycle, there is no solution. Otherwise, an example of a solution is

$$x_i = \delta(v_0, v_i) \quad \text{for} \quad i = 1, 2, \ldots, n.$$

Is the solution is unique?

24.4 Example system of difference constraints

$$x_1 - x_2 \leq 0, \tag{1}$$
$$x_1 - x_5 \leq -1, \tag{2}$$
$$x_2 - x_5 \leq 1, \tag{3}$$
$$x_3 - x_1 \leq 5, \tag{4}$$
$$x_4 - x_1 \leq 4, \tag{5}$$
$$x_4 - x_3 \leq -1, \tag{6}$$
$$x_5 - x_3 \leq -3, \tag{7}$$
$$x_5 - x_4 \leq -3. \tag{8}$$

24.4 Example constraint graph and solution