طراحی الگوریتم

۲۱ آبان ۹۸

ملکی مجد

Topic	Reference
Recursion and Backtracking	Ch.1 and Ch.2 JeffE
Dynamic Programming	Ch.3 JeffE and Ch.15 CLRS
Greedy Algorithms	Ch.4 JeffE and Ch.16 CLRS
Amortized Analysis	Ch.17 CLRS
Elementary Graph algorithms	Ch.6 JeffE and Ch.22 CLRS
Minimum Spanning Trees	Ch.7 JeffE and Ch.23 CLRS
Single-Source Shortest Paths	Ch.8 JeffE and Ch.24 CLRS
All-Pairs Shortest Paths	Ch.9 JeffE and Ch.25 CLRS
Maximum Flow	Ch.10 JeffE and Ch.26 CLRS
String Matching	Ch.32 CLRS
NP-Completeness	Ch.12 JeffE and Ch.34 CLRS

shortest-paths problem

- کوتاهترین مسیر از دانشکده کامپیوتر تا در اصلی دانشگاه؟
  - (از ساختمان یا فضای سبز رد نمی شیم)

- یک نقشه از دانشگاه داریم که فاصله های هر دو تقاطعی بر روی آن مشخص شده است.
- یک راه ممکن:
  - همه مسیرها را در نظر بگیریم و طول مسیرها را با توجه به قسمت های تشکیل دهنده آن حساب کنیم
  - با توجه به طول های بدست آمده، مسیر با کوتاهترین طول را پیدا کنیم

shortest-paths problem

- یک سری مسیرها ارزش بررسی ندارند
  - مثلا از در استخر خارج بشیم و از خارج دانشگاه به در اصلی برسیم
  - چه مسیرهایی را بررسی انتخاب کنیم.

- اگر مساله پیچیده تر شود و تعداد مسیرها زیاد باشد چه کنیم؟
- برنامه های کاربردی مثل waze و google map طول کوتاهترین مسیر را (البته با توجه به داده های برخط) برای ما حساب می کنند؟

shortest-paths problem

- در این جلسه حل مسئله کوتاهترین فاصله از مبدا خاص تا همه مقصدها را در نظر می گیریم.

- الگوریتم کارایی برای محاسبه طول کوتاهترین مسیرها را می بینیم

shortest-paths problem (formal definition)

- we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \rightarrow \boldsymbol{R}$ mapping edges to real-valued weights.

- The ***weight*** of path $p = < v_0, v_1, \ldots, v_k >$ is the sum of the weights of its constituent edges:

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

- We define the ***shortest-path weight*** from u to v by

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

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

می توانیم مسئله مسیر یابی را با یک گراف مدل کنیم

تقاطع ها راس های گراف هستند

مسیر بین هر دو تقاطع یک یال گراف هست که وزن این یال فاصله دو تقاطع است

هدف پیدا کردن کوتاه ترین مسیر بین دو راس این گراف است

- The **breadth-first-search algorithm** from Section 22.2 is a **shortest-paths algorithm** that works on **unweighted** graphs, that is, graphs in which each edge can be considered to have unit weight.

single-source shortest-paths problem

- focus of this session!

- Given a graph $G = (V, E)$, we want to find a shortest path from a given ***source*** vertex $s \in V$ to each vertex $v \in V$.

other shortest paths problems (1)

- **Single-destination shortest-paths problem:** Find a shortest path to a given ***destination*** vertex *t* from each vertex *v*. By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.

other shortest paths problems (2)

- **Single-pair shortest-path problem:** Find a shortest path from u to v for given vertices u and v. If we solve the single-source problem with source vertex u, we solve this problem also.

- Moreover, no algorithms for this problem are known that run asymptotically faster than the best single-source algorithms in the worst case.

other shortest paths problems (3)

- **All-pairs shortest-paths problem:** Find a shortest path from u to v for every pair of vertices u and v.

- Although this problem can be solved by running a single source algorithm once from each vertex, it can usually **be solved faster**. Additionally, its structure is of interest in its own right. Later, we addresses the all-pairs problem in detail.

Optimal substructure of a shortest path

- Shortest-paths algorithms typically rely on the property that a **shortest path between two vertices contains other shortest paths within it**.

- greedy method
  - Dijkstra's algorithm (to solve the single-source shortest-paths problem on a weighted, directed graph in which all edge weights are nonnegative)
- dynamic-programming
  - Floyd-Warshall algorithm (to find shortest paths between all pairs of vertices)

Lemma 24.1
(Subpaths of shortest paths are shortest paths)

- Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \rightarrow \boldsymbol{R}$,

let $p = < v_1, v_2, \ldots, v_k >$ be a shortest path from vertex v_1 to vertex v_k and, for any i and j such that $1 \leq i \leq j \leq k$,

let $p_{ij} = < v_i, v_{i+1}, \ldots, v_j >$ be the subpath of p from vertex v_i to vertex v_j .

Then, p_{ij} is a shortest path from v_i to v_j .

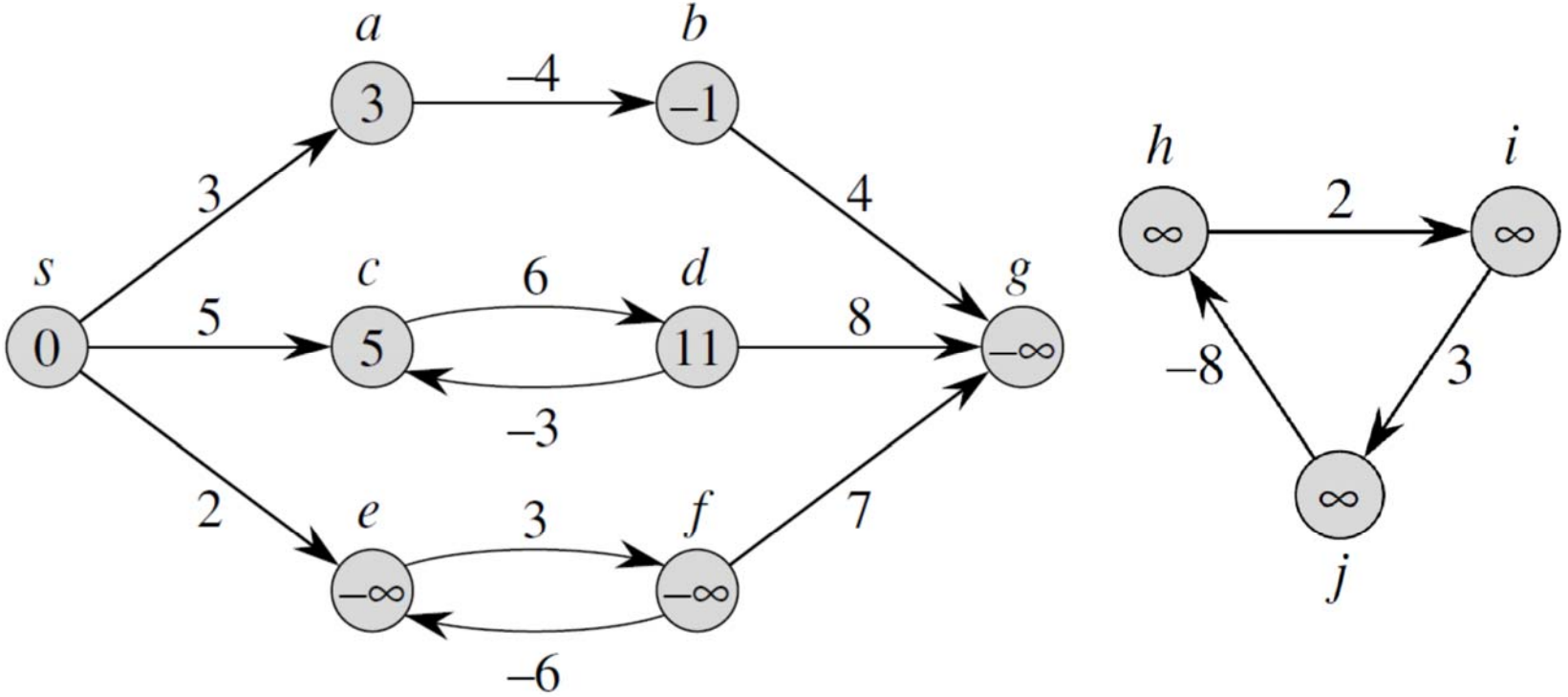
Proof : contradiction

Negative-weight edges

- If there is a **negative-weight cycle reachable from s**, however, shortest-path weights are not well defined.
- If there is a negative-weight cycle on some path from s to v, we define $\delta(s, v) = -\infty$.

- If the graph $G = (V, E)$ contains no negative weight cycles reachable from the source s, then for all $v \in V$, the shortest-path weight $\delta(s, v)$ remains well defined,

Example
to understand the effect of negative weight edges



Cycles

- Can a shortest path contain a cycle?

Cycles

- Can a shortest path contain a cycle?

- As a result:

we can restrict our attention to shortest paths of at most $|V| - 1$ edges.

Representing shortest paths

- wish to compute not only shortest-path weights, but the **vertices on shortest paths** as well
- Given a graph $G = (V, E)$, we maintain for each vertex $v \in V$ a ***predecessor*** $\pi[v]$ that is either another vertex or NIL.

- π attributes:

the chain of predecessors originating at a vertex v runs backwards along a shortest path from s to v

Remember PRINT-PATH(G, s, v)?

predecessor subgraph G_π

- ***predecessor subgraph*** $G_\pi = (V_\pi, E_\pi)$ induced by the π values.
- $V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$.
- $E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$

- the algorithms in this chapter have the property that at termination *Gπ* is a "shortest-paths tree"
  - During the execution of a shortest-paths algorithm, however, the π values need not indicate shortest paths

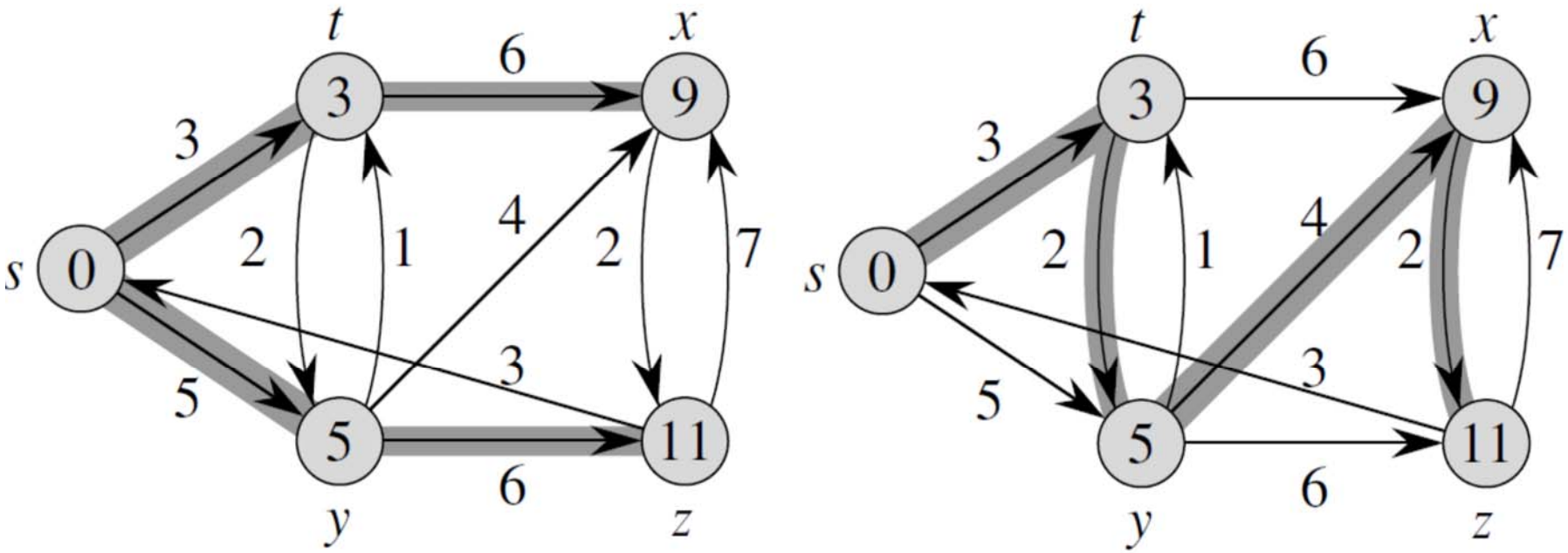
shortest-paths tree

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

1. V' is the set of vertices reachable from s in G,
2. G' forms a rooted tree with root s, and
3. 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 are not necessarily unique, and neither are shortest-paths trees.

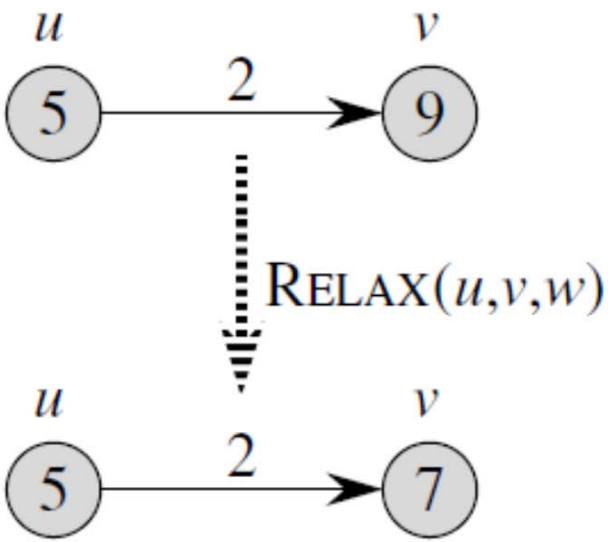
Example
two shortest paths trees with the same root



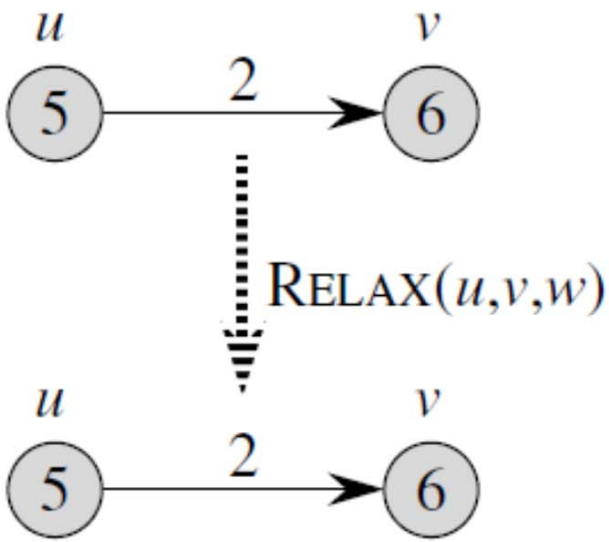
the technique of *relaxation*

- $d[v]$ a ***shortest-path estimate***
  - is an upper bound on the weight of a shortest path from source s to v

- The process of ***relaxing***[1] an edge *(u, v)* consists of testing whether we can improve the shortest path to *v* found so far by going through *u* and, if so, updating *d*[*v*] and π[*v*].



(a)



(b)

Each algorithm in this chapter calls INITIALIZE-SINGLE-SOURCE and then repeatedly relaxes edges.

- INITIALIZE-SINGLE-SOURCE(G, s)

1 **for** each vertex $v \in V[G]$
2 **do** $d[v] \leftarrow \infty$
3 $\pi[v] \leftarrow NIL$
4 $d[s] \leftarrow 0$

- RELAX(u, v, w)

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

- relaxation is the only means by which shortestpath estimates and predecessors change.
- The algorithms in this chapter differ in how many times they relax each edge and the order in which they relax edges.
- In Dijkstra's algorithm and the shortest-paths algorithm for directed acyclic graphs, each edge is relaxed exactly once.
- In the Bellman-Ford algorithm, each edge is relaxed many times.

Properties of shortest paths and relaxation

- These properties are used to prove that the algorithms in this chapter are correct

Properties of shortest paths and relaxation

- **Triangle inequality** (Lemma 24.10)

For any edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

- **Upper-bound property** (Lemma 24.11)

We always have $d[v] \geq \delta(s, v)$ for all vertices $v \in V$, and once $d[v]$ achieves the value *δ(s, v)*, it never changes.

- **No-path property** (Corollary 24.12)

If there is no path from *s* to *v*, then we always have $d[v] = \delta(s, v) = \infty$.

- **Convergence property** (Lemma 24.14)

If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $u, v \in V$, and if $d[u] = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $d[v] = \delta(s, v)$ at all times afterward.

Properties of shortest paths and relaxation

- **Path-relaxation property** (Lemma 24.15)

If $p = < V_0, V_1, \ldots, V_k >$ is a shortest path from $s = V_0$ to V_k , and the edges of p are relaxed in the order $(V_0, V_1), (V_1, V_2), \ldots, (V_{k-1}, V_k)$, then $d[V_k] = \delta(s, V_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of p.

- **Predecessor-subgraph property** (Lemma 24.17)

Once $d[v] = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

(implicitly assume that the graph is initialized with a call to INITIALIZESINGLE- SOURCE(G, s) and that the only way that shortest-path estimates and the predecessor subgraph change are by some sequence of relaxation steps.)

outline

- Bellman-Ford algorithm
  - solves the single-source shortest-paths problem in the general case in which edges can have negative weight.
  - is remarkable in its simplicity
  - detecting whether a negative-weight cycle is reachable from the source
- a linear-time algorithm
  - for computing shortest paths from a single source in a **directed acyclic graph**
- Dijkstra's algorithm
  - a lower running time than the Bellman-Ford algorithm
  - requires the edge weights to be nonnegative
- All algorithms in this chapter assume
  - the directed graph *G* is stored in the adjacency-list representation.
  - stored with each edge is its weight
  - traverse each adjacency list, we can determine the edge weights in *O(1)* time per edge.

The Bellman-Ford algorithm

- Given a weighted, directed graph $G = (V, E)$ with source *s* and weight function $w : E \rightarrow \boldsymbol{R}$, (allow negative-weight edges)

- Returns a **boolean value** indicating whether or not there is a **negative-weight cycle** that is reachable from the source

- If there is no such cycle, the algorithm produces the shortest paths and their weights.

The Bellman-Ford algorithm

- After initializing the d and π values of all vertices, the algorithm makes $|V| - 1$ passes over the edges of the graph

- The algorithm uses relaxation, progressively decreasing an estimate $d[v]$ on the weight of a shortest path from the source s to each vertex $v \in V$ until it achieves the actual shortest-path weight $\delta(s, v)$.

The Bellman-Ford algorithm

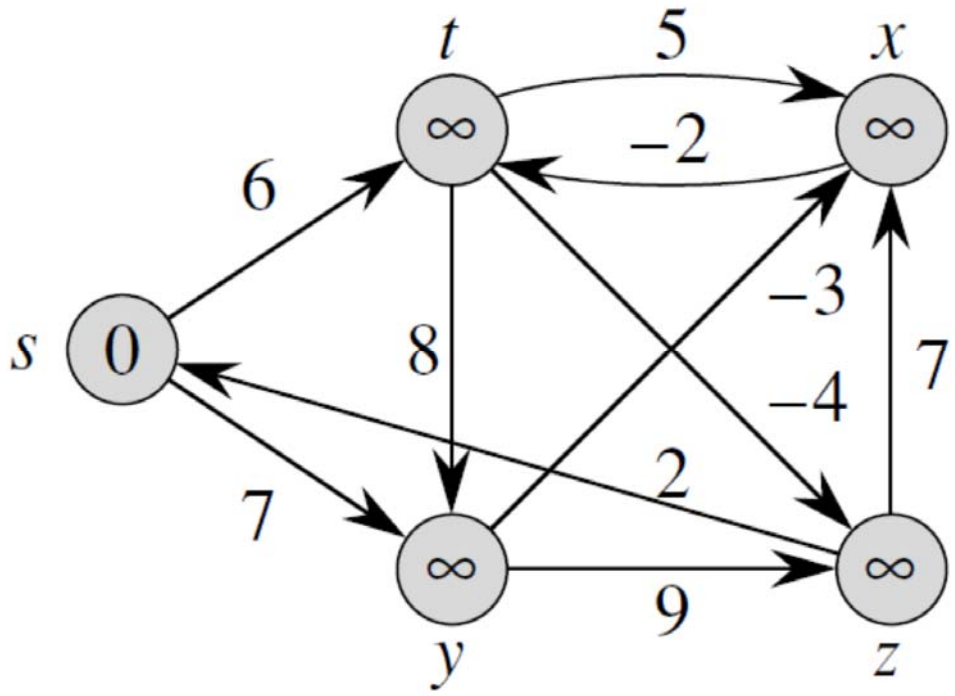
BELLMAN-FORD(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 **for** $i \leftarrow 1$ **to** $|V[G]| - 1$
3 **do for** each edge $(u, v) \in E[G]$
4 **do** RELAX(u, v, w)
5 **for** each edge $(u, v) \in E[G]$
6 **do if** $d[v] > d[u] + w(u, v)$
7 **then return** FALSE
8 **return** TRUE

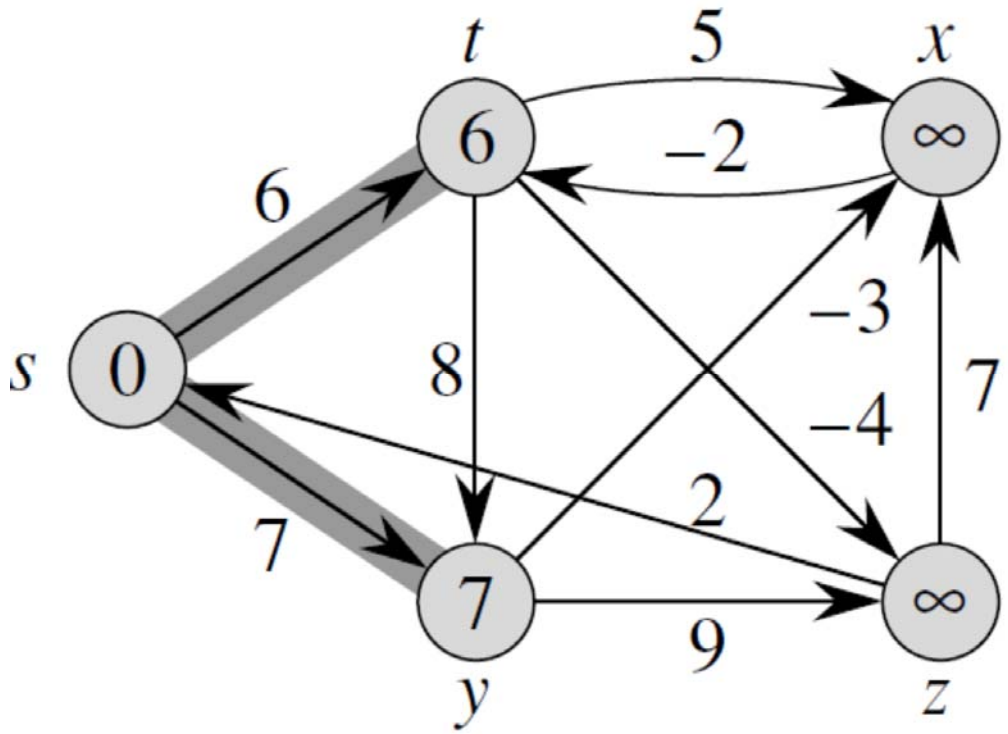
Time complexity

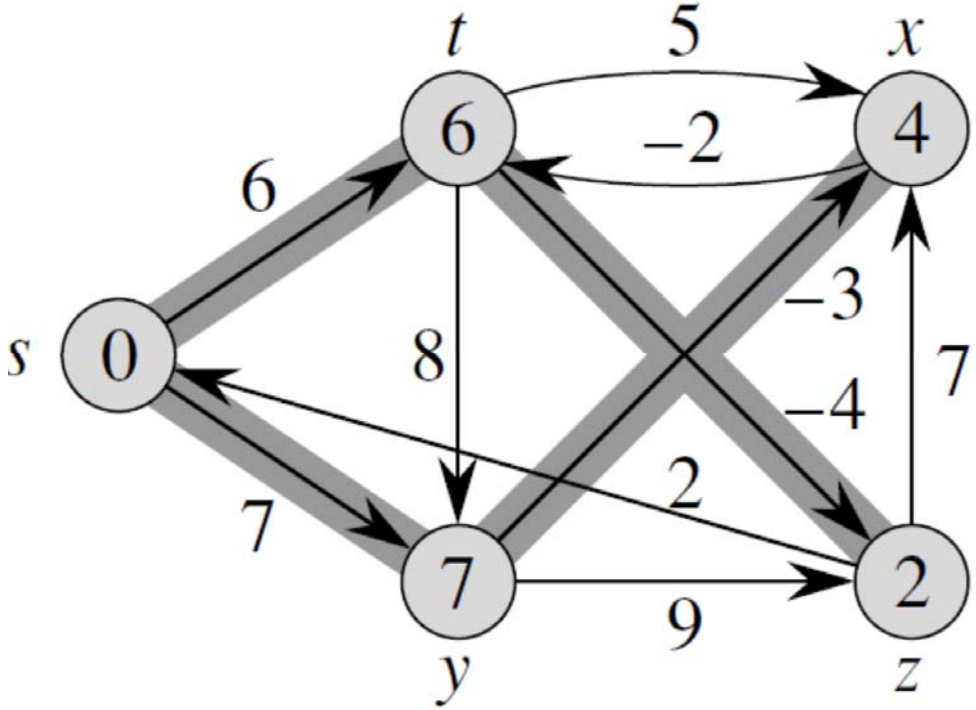
BELLMAN-FORD(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 **for** $i \leftarrow 1$ **to** $|V[G]| - 1$
3 **do for** each edge $(u, v) \in E[G]$
4 **do** RELAX(u, v, w)
5 **for** each edge $(u, v) \in E[G]$
6 **do if** $d[v] > d[u] + w(u, v)$
7 **then return** FALSE
8 **return** TRUE

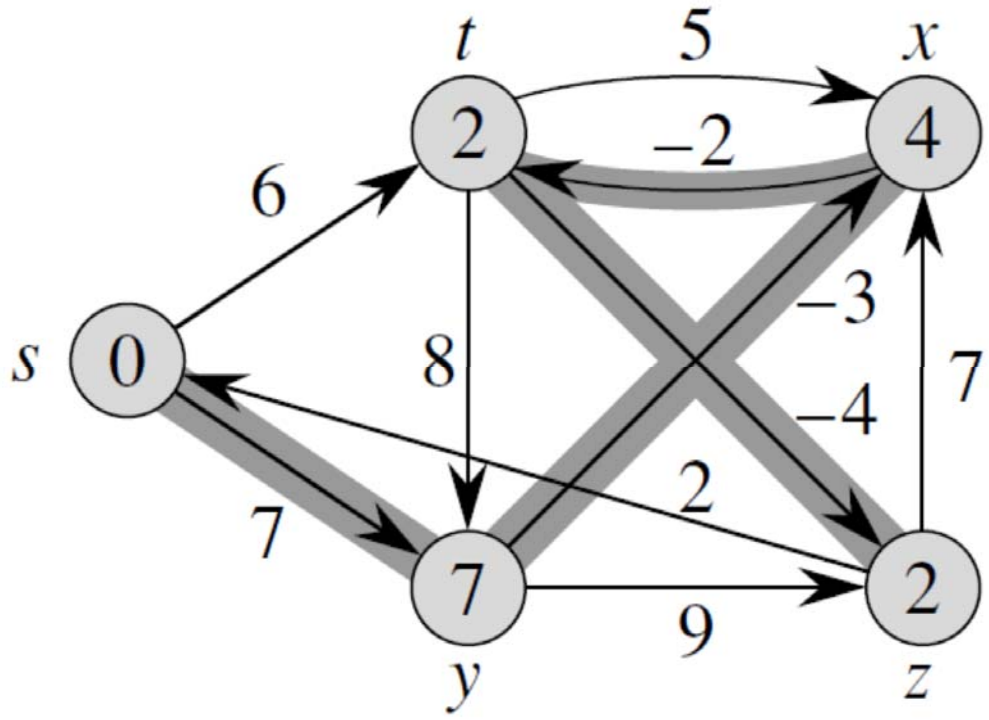
The Bellman-Ford algorithm runs in time *O(V E)*,

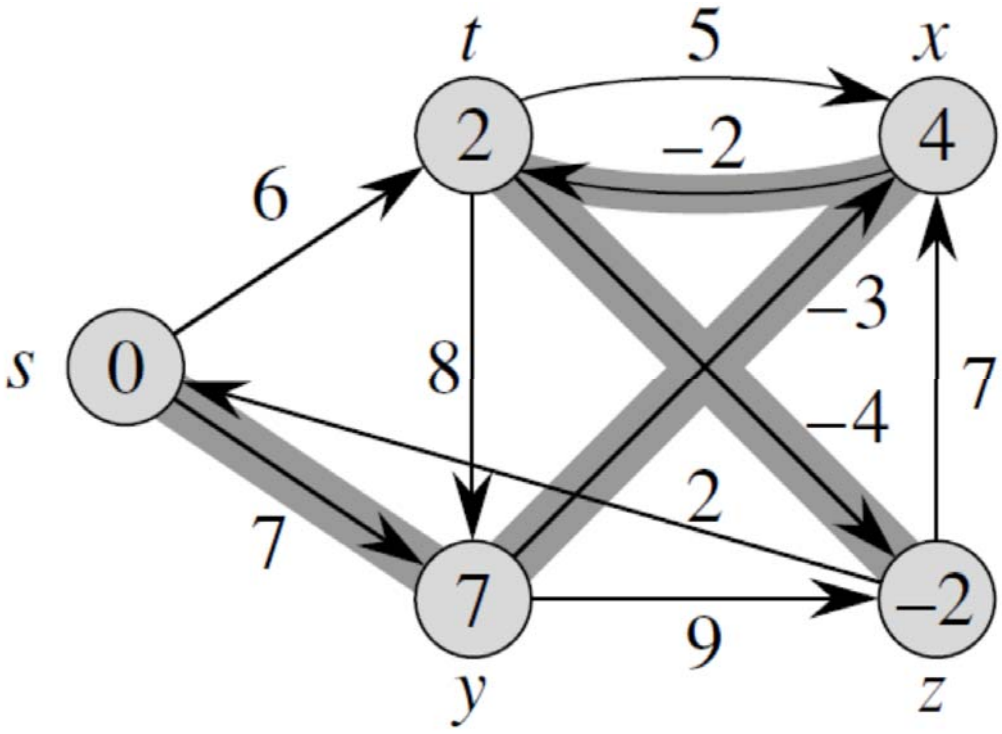
Example of the Bellman-Ford algorithm




t
5
x
6
−2
∞
6
−3
s
0
8
7
−4
2
7
7
∞
9
y
z


t
5
x
6
4
−2
6
−3
s
0
8
7
−4
2
7
7
2
9
y
z


t
5
x
2
4
−2
6
−3
s
0
8
7
−4
2
7
2
9
y
z


t
5
x
2
4
−2
6
s
0
8
−3
7
−4
2
7
7
−2
9
y
z

Correctness of the Bellman-Ford algorithm

Single-source shortest paths in directed acyclic graphs

- By relaxing the edges of a weighted dag (directed acyclic graph) $G = (V, E)$ according to a topological sort of its vertices, we can compute shortest paths from a single source in $\Theta(V + E)$ time.

- Shortest paths are always well defined in a dag, since even if there are negative-weight edges, no negative-weight cycles can exist.
- If there is a path from vertex *u* to vertex v then u precedes v in the topological sort.

DAG-SHORTEST-PATHS(G, w, s)

1 topologically sort the vertices of G

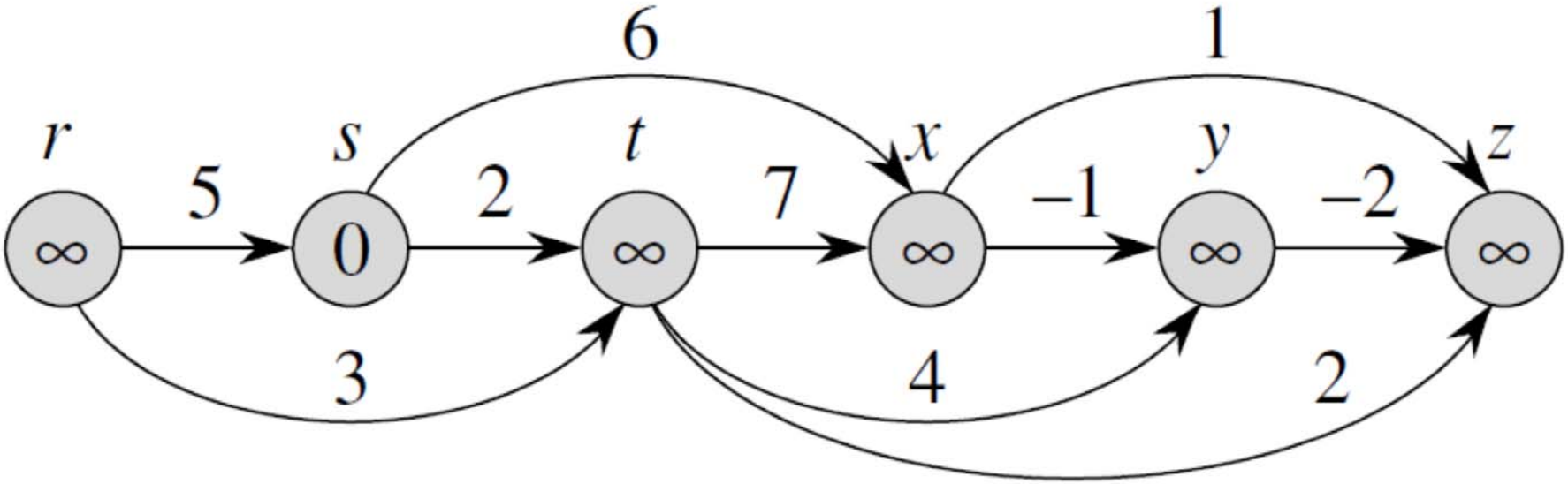
2 INITIALIZE-SINGLE-SOURCE(G, s)

3 **for** each vertex u, taken in topologically sorted order

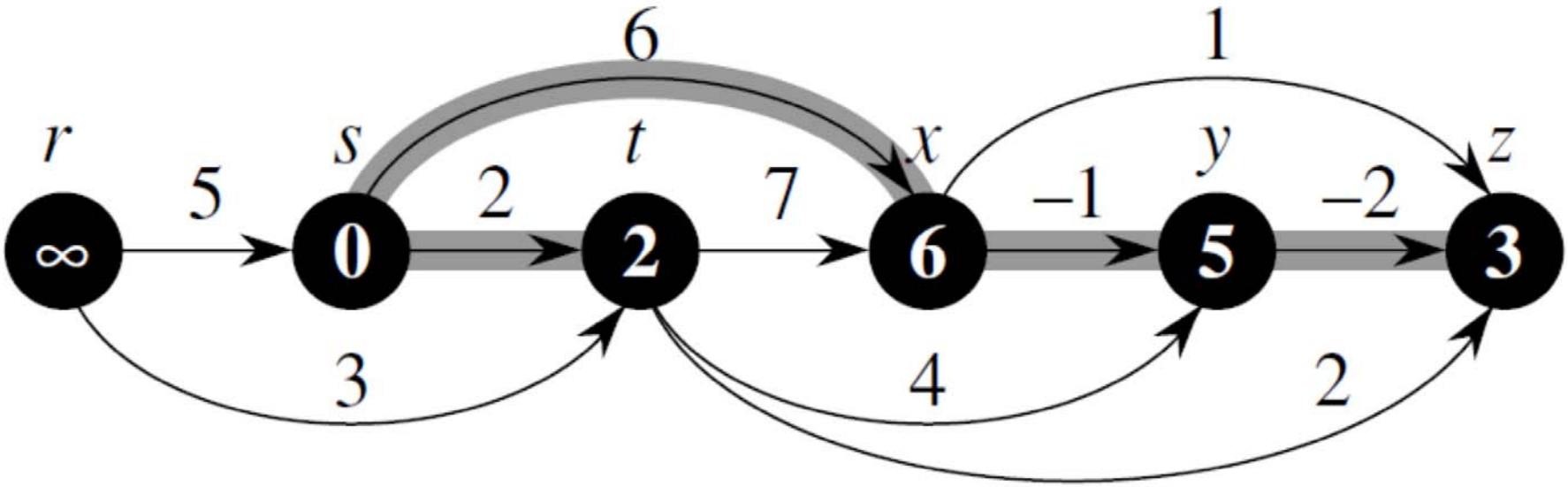
4 **do for** each vertex $v \in Adj[u]$

5 **do** RELAX(u, v, w)

Example of Single-source shortest paths in DAG



Example of Single-source shortest paths in DAG



critical path

- ***PERT chart***
  - Edges represent jobs to be performed, and edge weights represent the times required to perform particular jobs.
  - If edge *(u, v)* enters vertex *v* and edge *(v, x)* leaves *v*, then job *(u, v)* must be performed prior to job *(v, x)*.
  - A path through this dag represents a sequence of jobs that must be performed in a particular order

- A ***critical path*** is a *longest* path through the dag, corresponding to the longest time to perform an ordered sequence of jobs.
- The weight of a critical path is a lower bound on the total time to perform all the jobs.

How to find a critical path

- negating the edge weights and running DAG-SHORTEST-PATHS, or

- running DAG-SHORTEST-PATHS, with the modification that we replace "∞" by "$-\infty$" in line 2 of INITIALIZE-SINGLE-SOURCE and "$>$" by "$<$" in the RELAX procedure.