Algorithms & Data Structures

Lecture 11
Single-Source Shortest Paths

Giovanni Bacci
giovbacci@cs.aau.dk

Outline

- Weighted Graphs
- Shortest-Paths problems
- The Bellman-Ford algorithm
- Single-Source shortest paths in DAGs
- Dijkstra’s algorithm

Intended Learning Goals

KNOWLEDGE

- Mathematical reasoning on concepts such as recursion, induction, concrete and abstract computational complexity
- Data structures, algorithm principles e.g., search trees, hash tables, dynamic programming, divide-and-conquer
- Graphs and graph algorithms e.g., graph exploration, shortest path, strongly connected components.

SKILLS

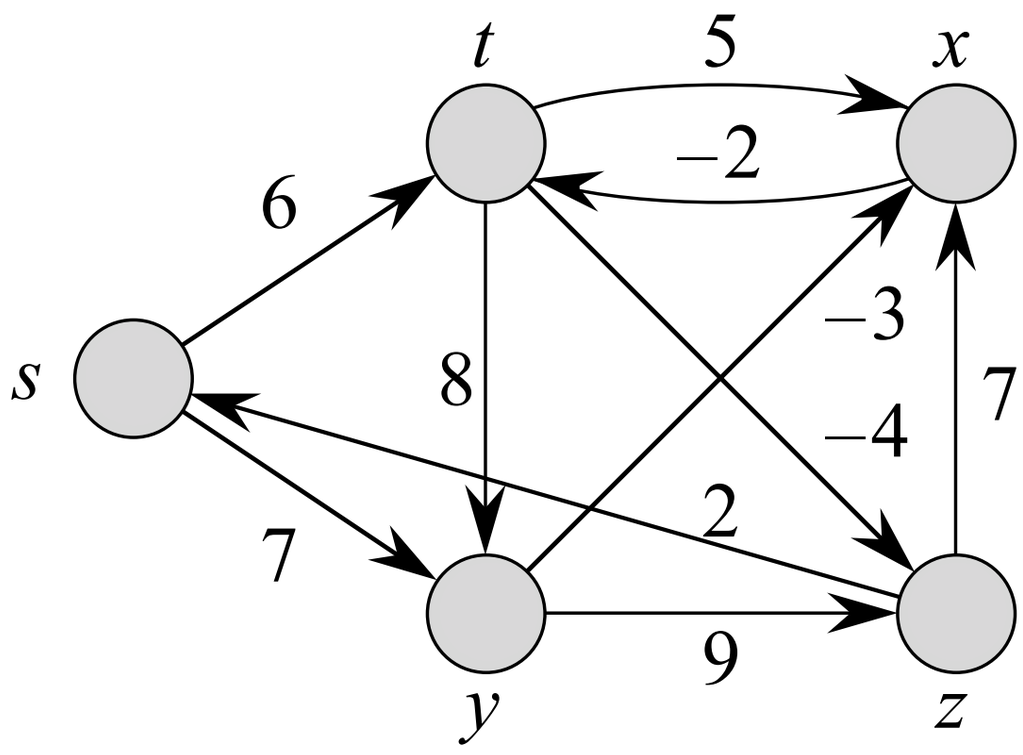
- Determine abstract complexity for specific algorithms
- Perform complexity and correctness analysis for simple algorithms
- Select and apply appropriate algorithms for standard tasks

COMPETENCES

- Ability to face a non-standard programming assignment
- Develop algorithms and data structures for solving specific tasks
- Analyse developed algorithms

Weighted Graphs

- A weighted graph is a graph $G = (V, E)$ with a weight function $w\colon E \to \mathbb{R}$ assigning a weight with each edge.
- The notion extends both to directed and undirected graphs



Representing Weighted Graphs

- **Adjacency-list** representation
  - Add weight satellite information on the elements in the lists
  - $(u, v) \in E$ iff $Adj[u] \in \langle v, w(u, v) \rangle$

- **Adjacency-matrix** representation
  - Boolean entries in the matrix $A = (a_{ij})$ are replaces by weights
  - $a_{i,j} = \begin{cases} w(i,j) & \text{if } (i,j) \in E \\ Nil & \text{otherwise} \end{cases}$
  - Depending on the application it may be convenient to use 0 or ∞ in place of Nil

Shortest Path

- Given a directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$
- The weight $w(p)$ of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weight of its constituent edges

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

Definition: We define the the shortest-path weight $\delta(u, v)$ from u to v by $\delta(u, v) = \min\{w(p) : u \overset{p}{\rightsquigarrow} v\}$. If there is no path connecting u and v then $\delta(u, v) = \infty$.

Single-source Shortest-path

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

Some Variants:

- **Single-destination shortest-paths problem**: find a shortest path to a given destination vertex $t \in V$ from each vertex $s \in V$ s.t. $s \neq t$
- **Single-pair shortest-path problem**: Find a shortest path from u to v for given vertices $u, v \in V$.
- **All-pairs shortest paths problem**: Find a shortest path from u to v for every pair $(u, v) \in V \times V$

Optimal Substructure

Lemma 24.1 (Subpaths of shortest paths are shortest paths)
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ 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) Towards a contradiction, we assume p is optimal and admits a decomposition $p = p_{0i} \cdot p_{ij} \cdot p_{jk}$ such that p_{ij} is not optimal. Then we construct another path $p^* = p_{0i} \cdot p^*_{ij} \cdot p_{jk}$ by replacing p_{ij} with an optimal path from v_i to v_j. This yields the contradiction that $w(p^*) < w(p)$.

Shortest path algorithms typically rely on the above property

Cycles

- Assume G has a cycle c from v to v, that is $v \rightsquigarrow v$.
- Then any path p from v_0 to v_k that contains v, i.e. $p = p' \cdot v \cdot p''$, can be “pumped” by adding the cycle as many times as one wants.
- $p_i = p' \cdot c^i \cdot p''$ for i repetitions of the cycle c
  - (here $c^0 = v$ and $c^i = c^{i-1} \cdot c$)
- We have three cases:
  - **Positive cycle** ($w(c) > 0$): removing one iteration of the cycle decreases the weight of the path
  - **0-weight cycle** ($w(c) = 0$): removing one iteration of the cycle does not alter the weight of the path
  - **Negative cycle** ($w(c) < 0$): every time we **add** one more iteration of the cycle we decrease the weight of the path

Negative Cycles
Assumptions

- ***Dijkstra's algorithm*** assumes that all edge weights are nonnegative

- ***Bellman-Ford algorithm*** allow negative-weight edges but produce a correct answer as long as no negative-cycles are reachable from the source. In particular the algorithm detects and reports the existence of such a problematic cycle.

Representing shortest paths

- Since we assume no negative cycles, we can also assume that the shortest paths that we compute are **simple paths** (i.e., have no cycles)
- Like BFS, we use the attribute π to represent shortest paths form s to any other vertex reachable from it
- We will update a **predecessor subgraph** $G_\pi = (V_\pi, E_\pi)$ where
  - $V_\pi = \{v \in V : v.\pi \neq Nil\} \cup \{s\}$
  - $E_\pi = \{(v_\pi, v) \in E : v \in V_\pi \backslash \{s\}\}$

- At termination we obtain a **shortest-path tree** rooted at s:
  - V_π is the set of vertices reachable from s
  - G_π forms a tree with root s, and
  - For all $v \in V_\pi$ the unique simple path from s to v in G_π is a shortest-path from s to v in G (relative to the weight function w)

Relaxation

- Both algorithms will use a technique called **relaxation**

- For each vertex $v \in V$ we maintain an attribute $v.d$ representing a **shortest-path estimate** from s to v (i.e., a upper bound of the weight of a shortest path)

INITIALIZE-SINGLE-SOURCE(G, s)

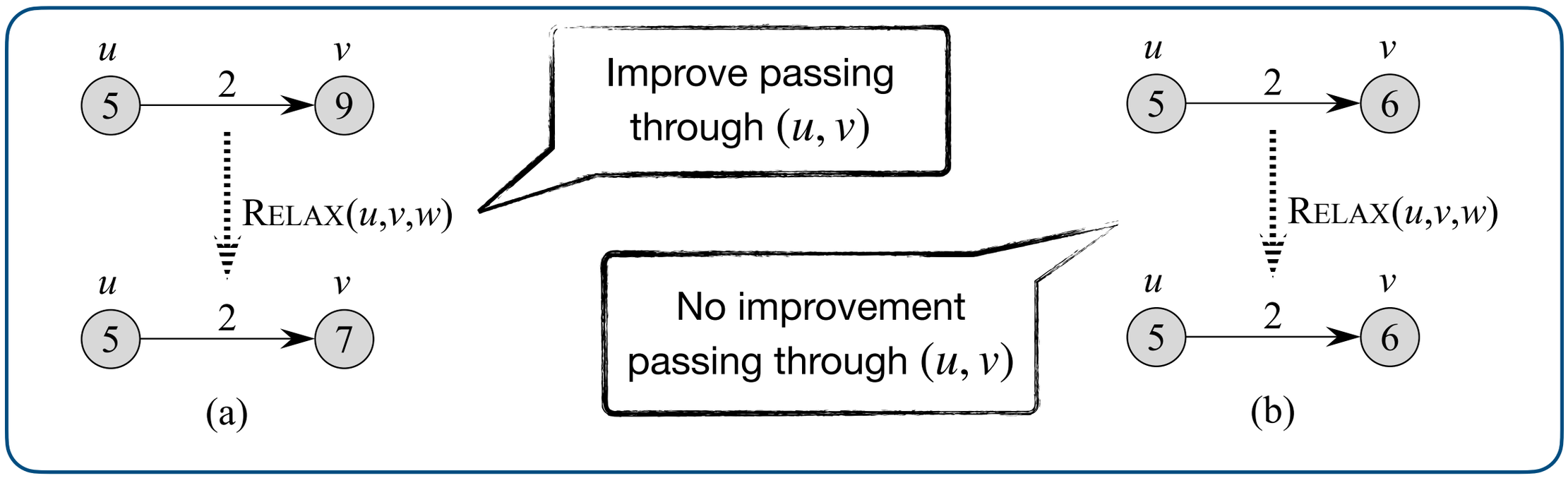
```
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

We initialise the attributes of each vertex as one may expect

Relaxation

- After initialisation we will *improve* the shortest-path estimates by **relaxing edges** $(u, v) \in E$

$\textsc{Relax}(u, v, w)$
1 **if** $v.d > u.d + w(u, v)$
2 $\quad v.d = u.d + w(u, v)$
3 $\quad v.\pi = u$



Relaxation Properties (1/2)

- After any sequence of relaxation steps the following properties hold

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 $v.d \geq \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.

No-path property (Corollary 24.12)

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

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 $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.

Relaxation Properties (2/2)

- After any sequence of relaxation steps the following properties hold

Path-relaxation property (Lemma 24.15)

If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to v_k, and we relax the edges of p in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. 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 $v.d = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at s.

Bellman-Ford Algorithm

Bellman-Ford Algorithm

- Solves the single-source shortest-path problem in the general case in which weights may be negative
- If there is a cycle, the algorithm indicates that no solution exists
- If there is no such a cycle, the algorithm produces a shortest-path tree rooted at $s \in V$.

Main Idea

- Since simple paths have at most $|V| - 1$ edges, we discover all shortest paths from s by performing $|V| - 1$ relaxation rounds to all edges $(u, v) \in E$.
- If after this we can still decrease the shortest-path estimate then it must be because of a negative cycle

Bellman-Ford: pseudocode

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

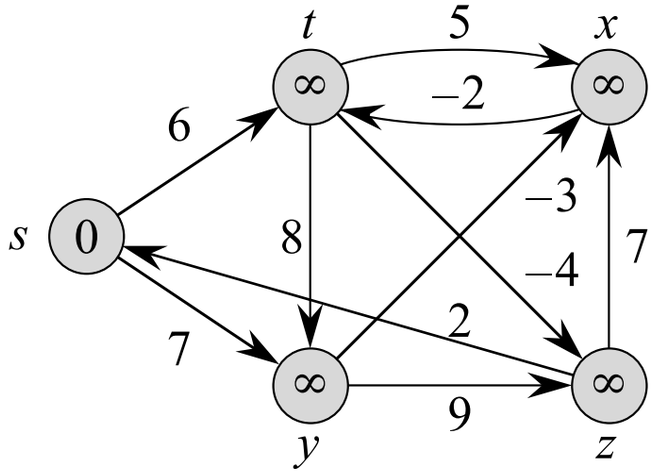
Initialise attributes of each vertex. Time $\Theta(|V|)$

Expand the frontier of shortest paths $|V|-1$ times. Time $\Theta(|V||E|)$

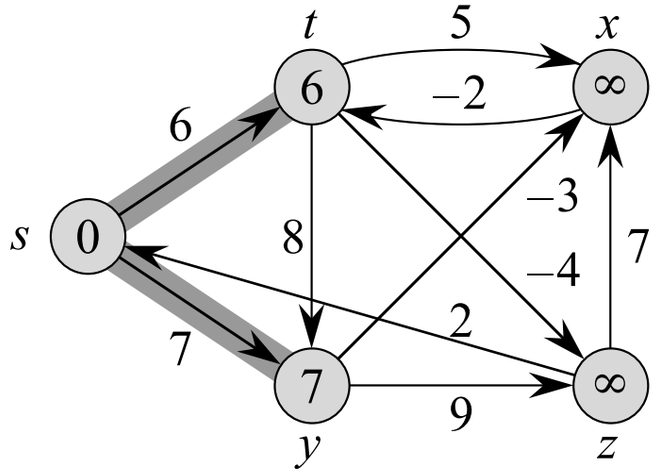
Check if we can further improve some the estimate. Time $\Theta(E)$

- Overall running time $\Theta(|V||E|)$

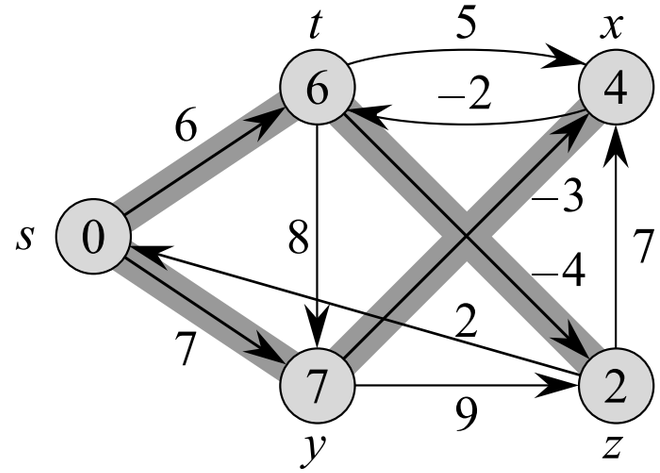
Bellman-Ford: example



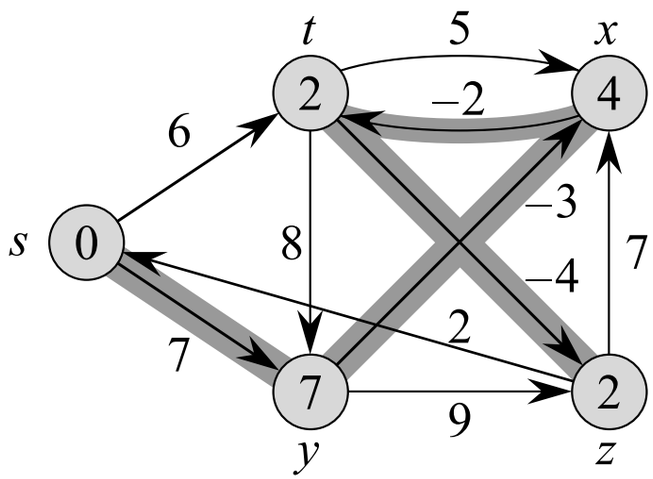
(a)



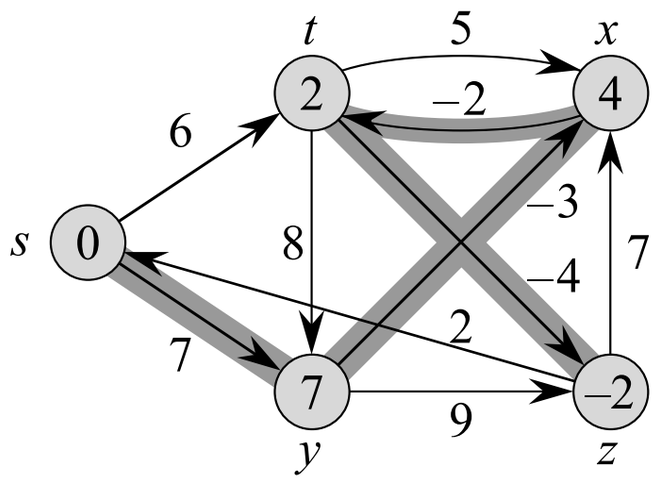
(b)



(c)



(d)



(e)

Bellman-Ford: correctness

Lemma 24.2

Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $w : E \to \mathbb{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the $|V| - 1$ iterations of the **for** loop of lines 2–4 of BELLMAN-FORD, we have $v.d = \delta(s, v)$ for all vertices v that are reachable from s.

Proof.

- Consider $v \in V$ reachable from s and $p = \langle v_0, \ldots, v_k \rangle$ be a simple shortest path from $v_0 = s$ to $v_k = v$. Then $k \leq |V| - 1$.
- Each of the $|V| - 1$ iterations of the outer for loop relaxes all edges. Among these, at iteration i there is also the edge (v_{i-1}, v_i)
- Then by the path relaxation property we have $v \,.\, d = v_k \,.\, d = \delta(s, v_k) = \delta(s, v)$

Bellman-Ford: correctness

Theorem 24.4 (Correctness of the Bellman-Ford algorithm)
Let BELLMAN-FORD be run on a weighted, directed graph $G = (V, E)$ with source s and weight function $w : E \to \mathbb{R}$. If G contains no negative-weight cycles that are reachable from s, then the algorithm returns TRUE, we have $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-paths tree rooted at s. If G does contain a negative-weight cycle reachable from s, then the algorithm returns FALSE.

Proof. (only assuming no negative cycles).

- If $v \in V$ is reachable from s then Lemma 24.2 proves that $v\,.\,d = \delta(s, v)$
- If $v \in V$ is not reachable from s, then $\delta(s, v) = \infty$. The fact that also $v\,.\,d = \infty$ follows from the no-path property.

Single-Source shortest paths for DAGs

The case of DAGs

- For DAGs we do not need to worry about negative cycles because there are none
- We can simplify the algorithm by exploiting topological ordering of the vertices.

Main Idea

- In a DAG all paths follow any topological ordering of the vertices. Thus also shortest paths from s.
- We can reuse Bellman-Ford idea reducing the number of iterations of its main for loop

DAG-Shortest-Paths

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

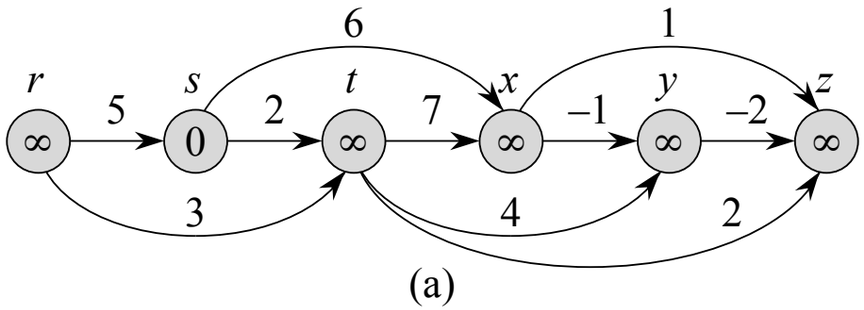
1 topologically sort the vertices of G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **for** each vertex u, taken in topologically sorted order
4 **for** each vertex $v \in G.Adj[u]$
5 RELAX(u, v, w)

Compute a topological ordering.
Time $\Theta(|V| + |E|)$

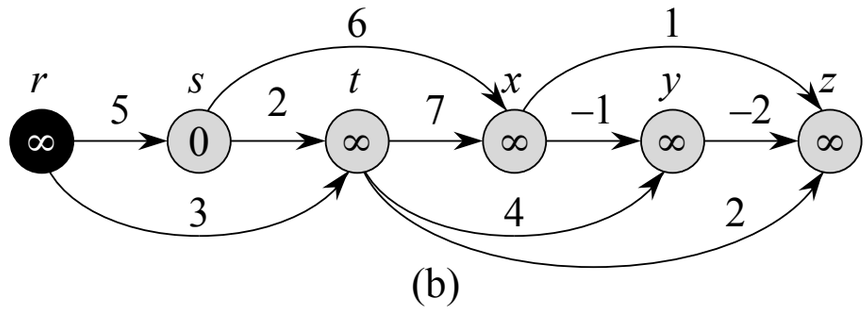
Expand the frontier of shortest paths following topological ordering. Time $\Theta(|V| + |E|)$

- Overall running time $\Theta(|V| + |E|)$

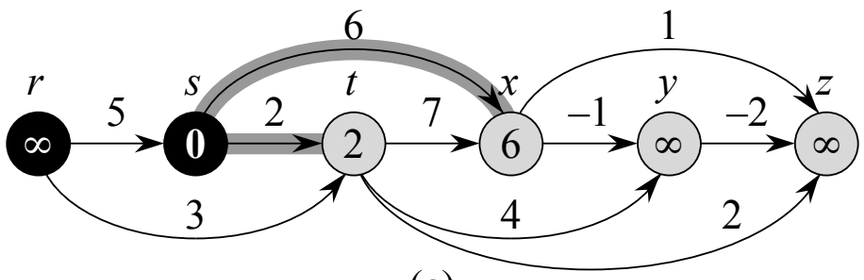
DAG-Shortest-Paths: example



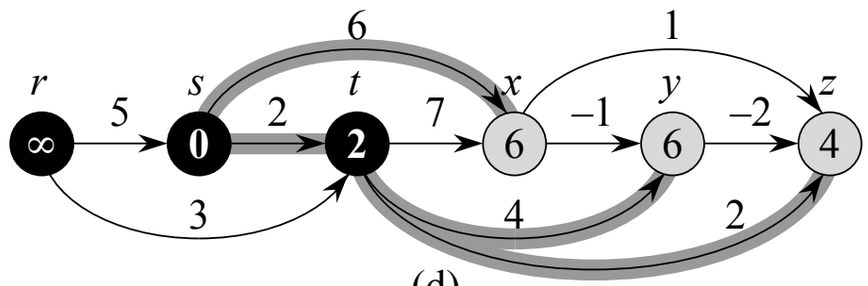
(a)



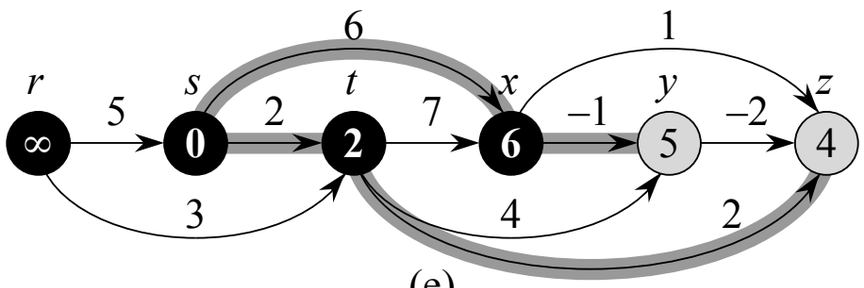
(b)



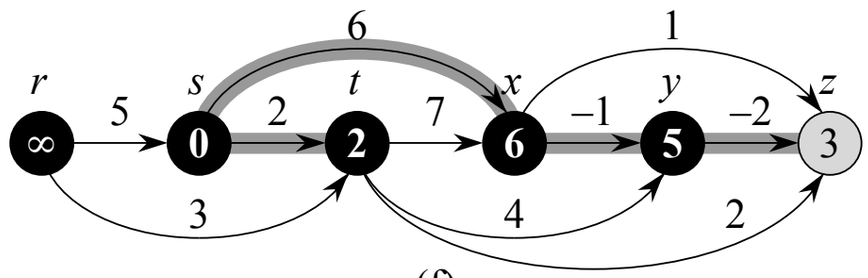
(c)



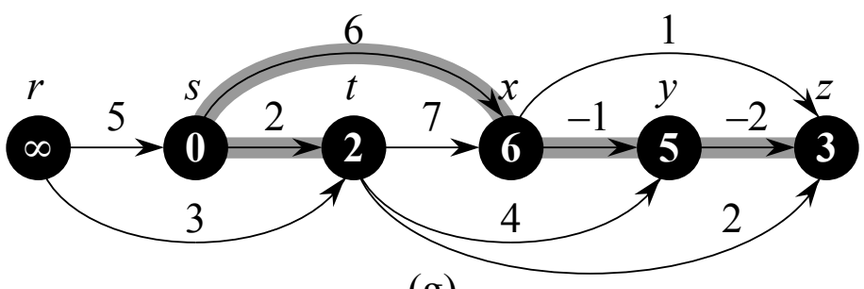
(d)



(e)



(f)



(g)

DAG-Shortest-Paths: correctness

Theorem 24.5
If a weighted, directed graph $G = (V, E)$ has source vertex s and no cycles, then at the termination of the DAG-SHORTEST-PATHS procedure, $v.d = \delta(s, v)$ for all vertices $v \in V$, and the predecessor subgraph G_π is a shortest-paths tree.

Proof.

- If $v \in V$ is reachable from s then by the fact that edges are relaxed according to a topological order we relax edges following the order of a shortest path. Hence, by the path-relaxation property we have $v\,.\,d = \delta(s, v)$
- If $v \in V$ is not reachable from s, then $\delta(s, v) = \infty$. The fact that also $v\,.\,d = \infty$ follows from the no-path property.
- Finally, by the predecessor subgraph property we have that G_π is a shortest-paths tree rooted at s

Dijkstra’s Algorithm

Dijkstra's Algorithm

- Solves the single-source shortest-path problem assuming edge weights are nonnegative
- Hence, no negative cycles
- It generalises BFS over weighted graphs

Main Idea

- Like BSF expands the frontiers in order of increasing distance from s.
- Each node is visited once and its adjacent vertices are explored
- Use a min-priority queue organised according to the actual value of the shortest-path estimates of vertices not encountered yet

Dijkstra: pseudocode

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

- S = vertices completed
- Q = vertices not yet completed

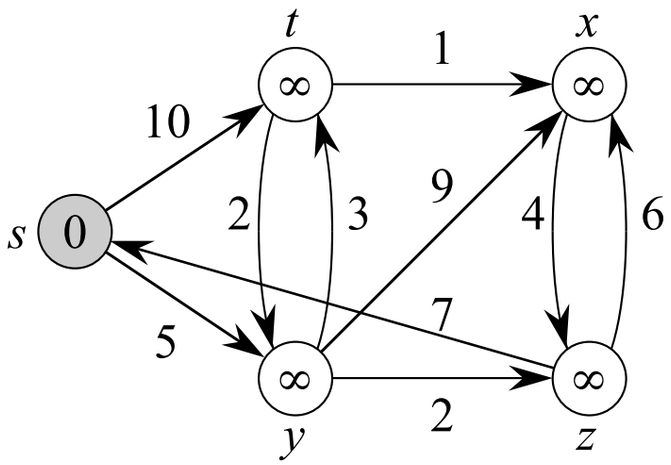
Remark. Q is implicitly populated using Insert operations

Take a vertex at minimal distance from s in Q and expand the frontier from there by relaxing outgoing edges

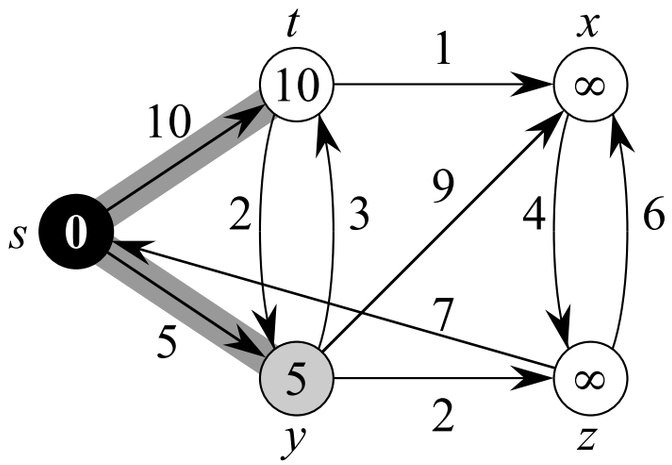
Remark: implicitly use a DecreaseKey operation on Q when relaxing the edge

- The algorithm maintains the loop invariant that $Q = V \backslash S$ at the start of each iteration of the while loop.

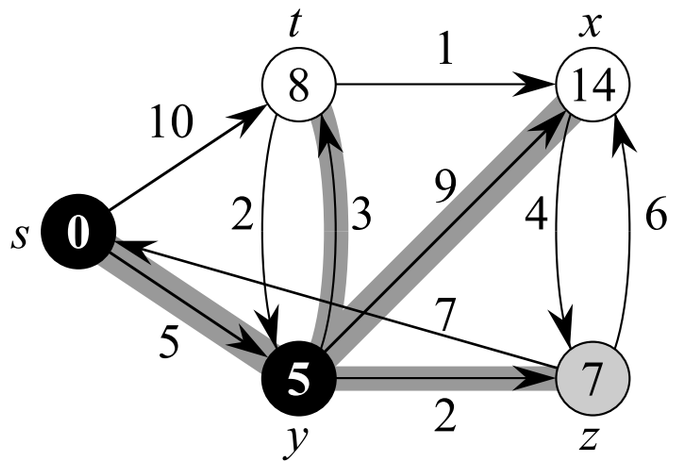
Dijkstra: example



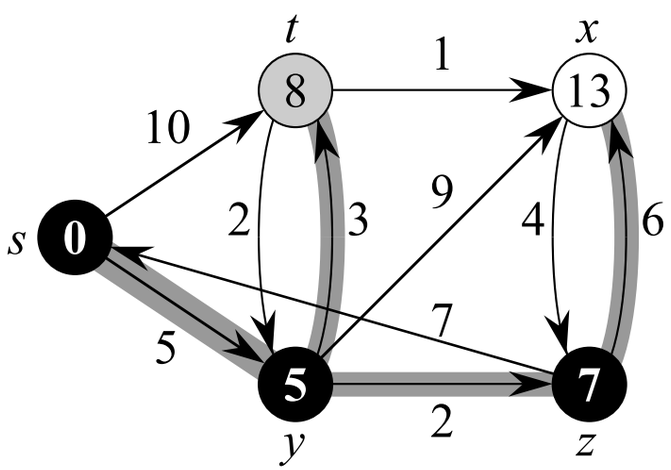
(a)



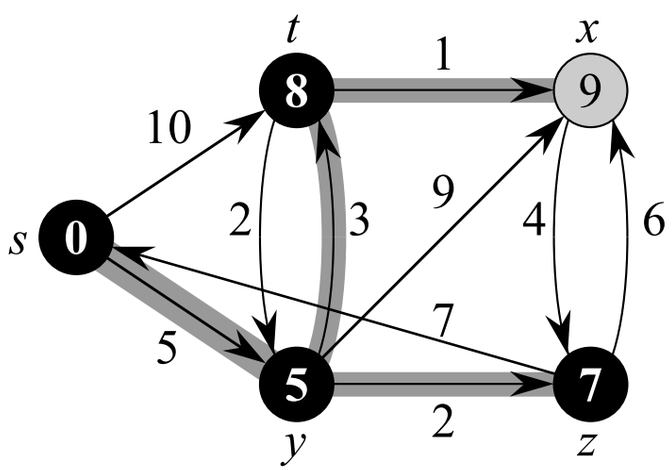
(b)



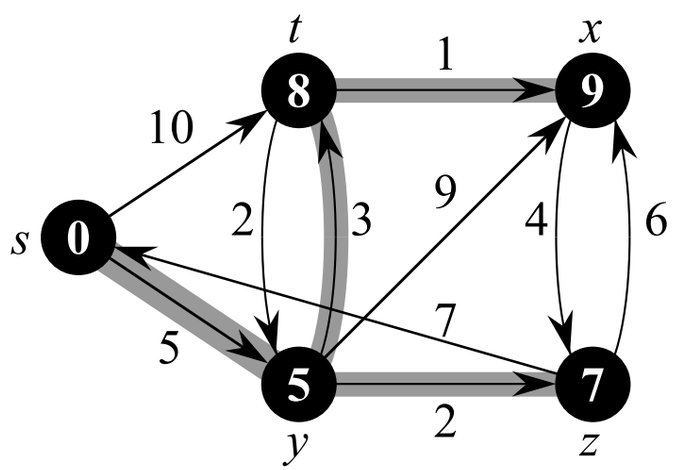
(c)



(d)



(e)



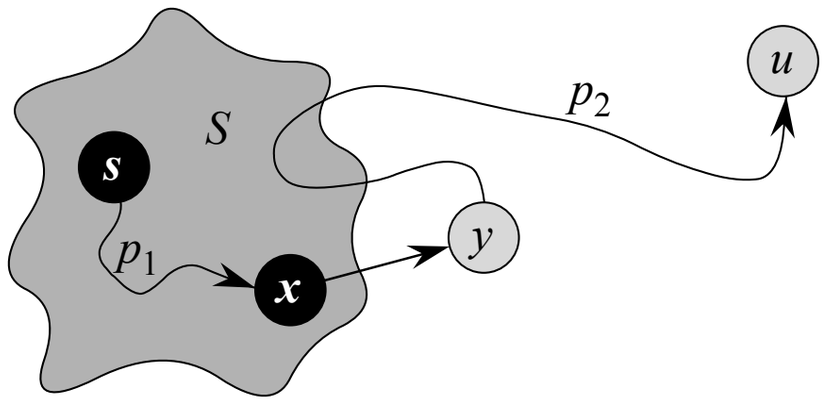
(f)

Dijkstra: correctness

Theorem 24.6 (Correctness of Dijkstra's algorithm)
Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with non-negative weight function w and source s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.

Proof outline.

- We use the following loop invariant: at the start of each iteration $v \,.\, d = \delta(s, v)$ for each vertex $v \in S$
- We show that for each vertex $v \in V$ we have $v \,.\, d = \delta(s, v)$ at the time when v is added to the set S. Then we rely on the upper-bound property to show that the equality holds at all times thereafter



Dijkstra: correctness

Corollary 24.7
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_π is a shortest-paths tree rooted at s.

Proof.

- Immediate from Theorem 24.6 and the predecessor-subgraph property

Dijkstra: run-time (1/2)

- It depends on the implementation of the priority queue Q

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

$\Theta(|V|)$

Q is implicitly populated using |V| INSERT operations

Each vertex enters Q exactly once. Then the DECREASE-KEY operation (implicitly called by RELAX) is called at most $|E|$ times

Dijkstra: run-time (2/2)

- Q implemented using an array
  - Assumption: vertices are numbered 1 to $|V|$
  - Insert and Decrease-Key take $\Theta(1)$
  - Extract-Min takes $\Theta(V)$
  - Therefore Dijkstra's algorithm running time is $\Theta(|V|^2) + \Theta(|E|) = \Theta(|V|^2)$

- If G is sufficiently sparse— in particular $|E| = o(V^2/\lg V)$ one can use
  - **Min-Heap:** achieving $O((V + E) \lg V)$ which becomes $O(E \lg V)$ if all vertices are reachable from s
  - **Fibonacci heap:** achieving $O(V \lg V + E)$

Learned Today

- Weighted Graphs & Representing weighted graphs
- Single-source shortest-Paths problem and other variants
  - Optimal substructure property (*Lemma 24.1 CLRS-3*)
- The Bellman-Ford algorithm: $\Theta(|V||E|)$
  - Admits negative weights
  - Determines if there are negative cycles
- Single-Source shortest-paths in DAGs: $\Theta(|V| + |E|)$
  - Assumes G has no cycles
  - Exploit topological ordering to improve performance
- Dijkstra's algorithm: $\Theta(|V|^2)$
  - Assumes G has no negative weighted edge
  - Generalisation of BFS for non-negative weighted graphs
  - Running time depends on underlying implementation of min-priority queue