Chapter 8: Optimal Trees and Paths

(cp. Cook, Cunningham, Pulleyblank & Schrijver, Chapter 2)

Trees and Forests (Reminder)

Definition 8.1.

a. An undirected graph having no circuit is called a forest.
b. A connected forest is called a tree.

Theorem 8.2.

Let $G = (V, E)$ be an undirected graph on $n = |V|$ nodes. Then, the following statements are equivalent:

i. G is a tree.
ii. G has $n - 1$ edges and no circuit.
iii. G has $n - 1$ edges and is connected.
iv. G is connected, but $(V, E \setminus \{e\})$ is disconnected for any $e \in E$.
v. G has no circuit. Adding an arbitrary edge to G creates a circuit.
vi. G contains a unique path between any pair of nodes.

Proof: See, e.g., CoMa I. □

Kruskal's Algorithm (Reminder)

Minimum Spanning Tree (MST) Problem

Given: connected graph $G = (V, E)$, cost function $c : E \to \mathbb{R}$.

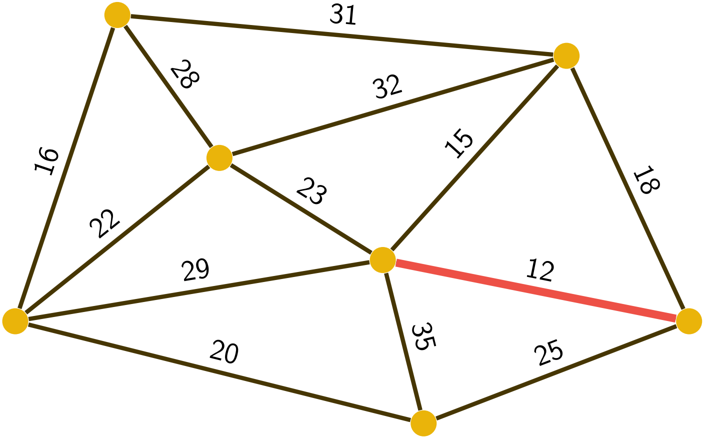
Task: find spanning tree $T = (V, F)$ of G with minimum cost $\sum_{e \in F} c(e)$.

Kruskal's Algorithm for MST

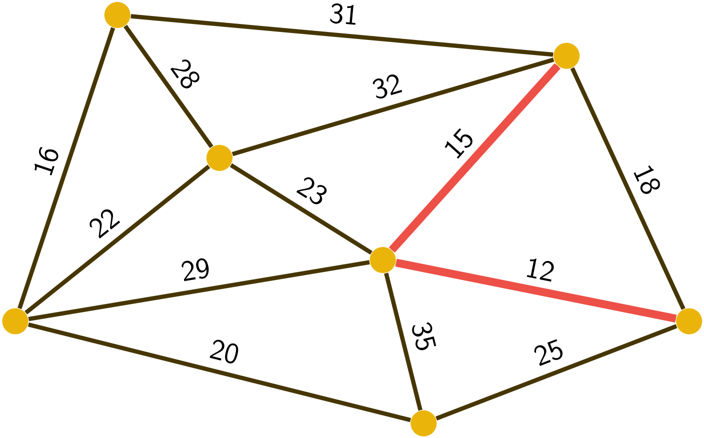
1. sort the edges in E such that $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$;
2. set $T := (V, \emptyset)$;
3. for $i := 1$ to m do:

   if adding e_i to T does not create a circuit, then add e_i to T;

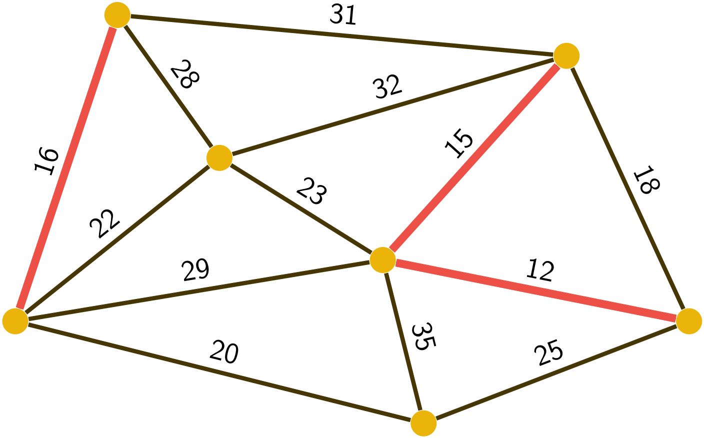
Example for Kruskal's Algorithm



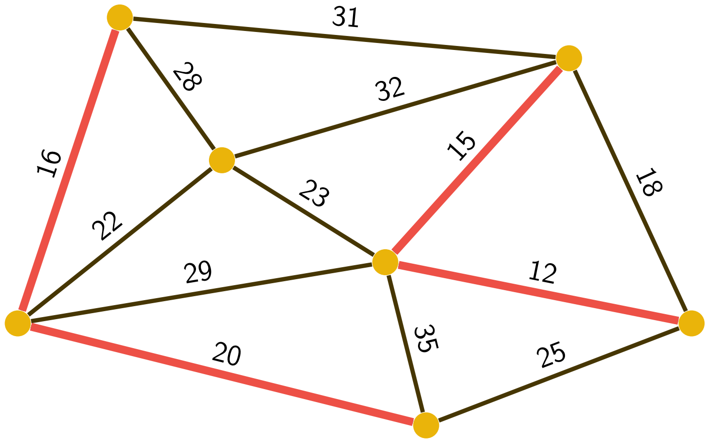
Example for Kruskal's Algorithm



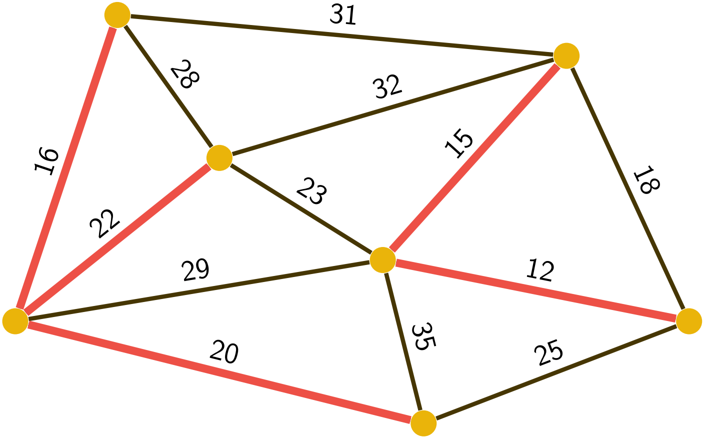
Example for Kruskal's Algorithm



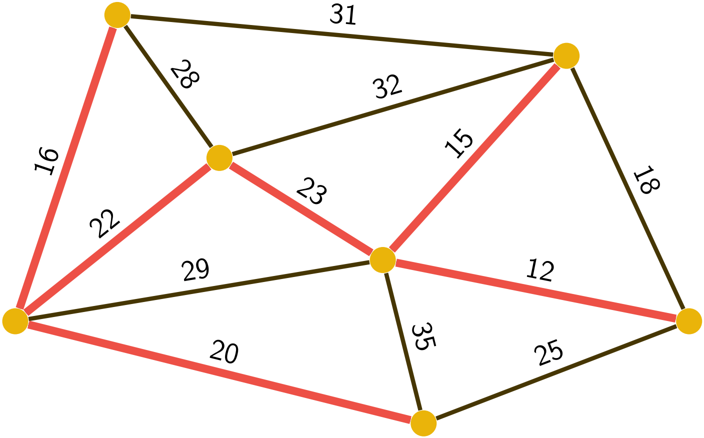
Example for Kruskal's Algorithm



Example for Kruskal's Algorithm



Example for Kruskal's Algorithm



Prim's Algorithm (Reminder)

Notation: For a graph $G = (V, E)$ and $A \subseteq V$ let

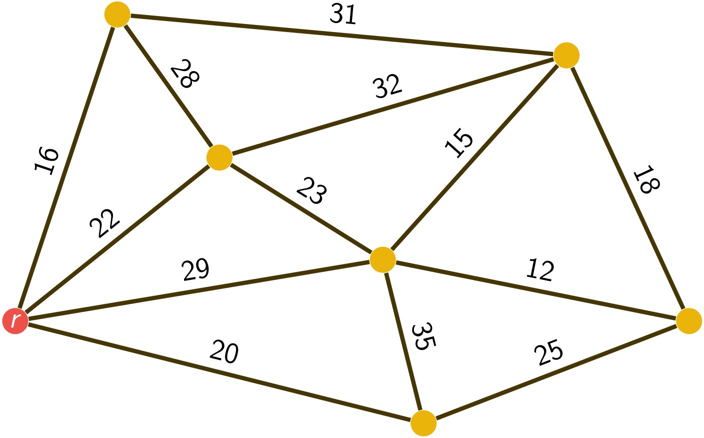
$$\delta(A) := \big\{e = \{v, w\} \in E \mid v \in A \text{ and } w \in V \setminus A\big\}.$$

We call $\delta(A)$ the cut induced by A.

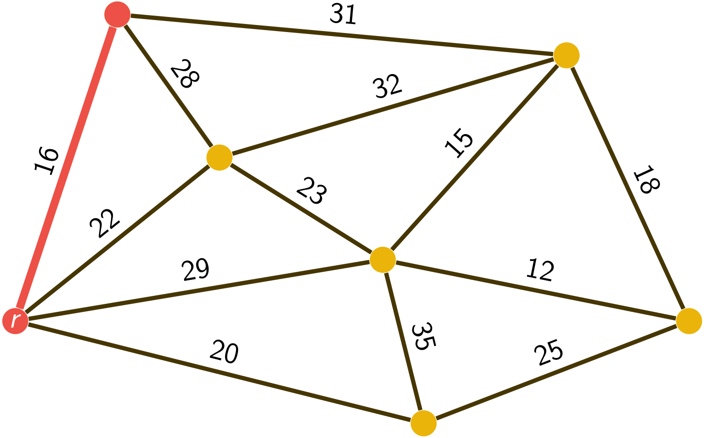
Prim's Algorithm for MST

1. set $U := \{r\}$ for some node $r \in V$ and $F := \emptyset$; set $T := (U, F)$;
2. while $U \neq V$, determine a minimum cost edge $e \in \delta(U)$;
3. set $F := F \cup \{e\}$ and $U := U \cup \{w\}$ with $e = \{v, w\}$, $w \in V \setminus U$;

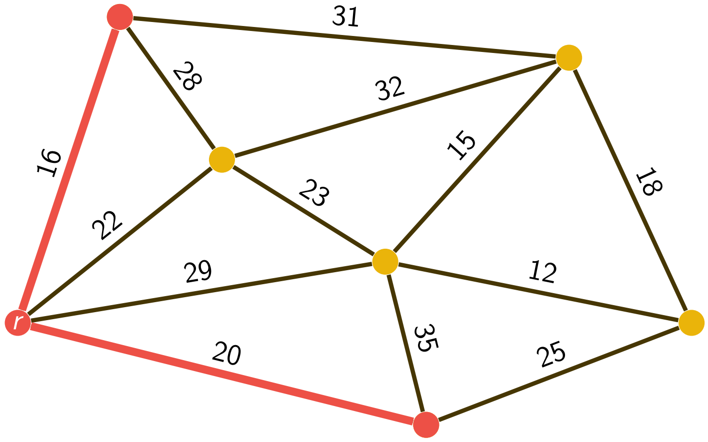
Example for Prim's Algorithm



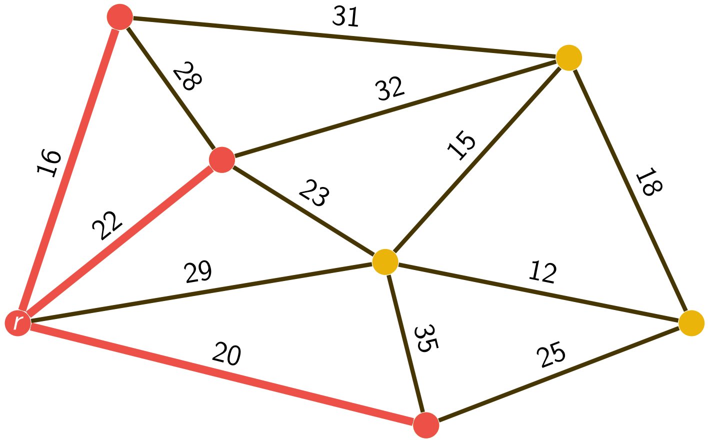
Example for Prim's Algorithm



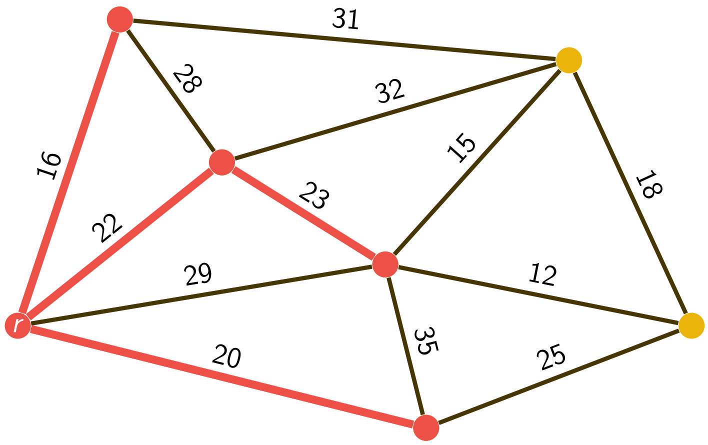
Example for Prim's Algorithm



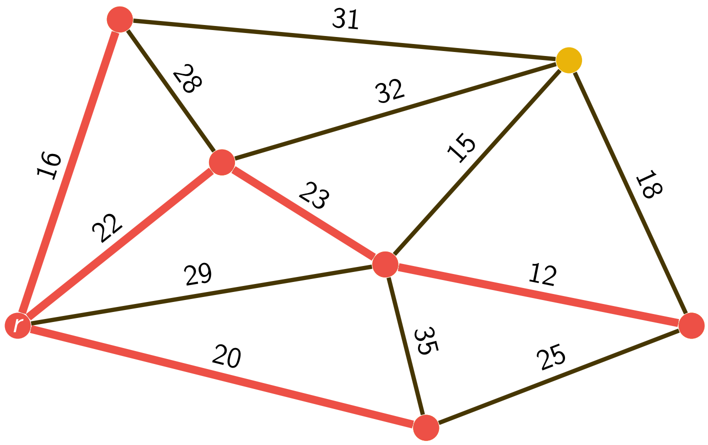
Example for Prim's Algorithm



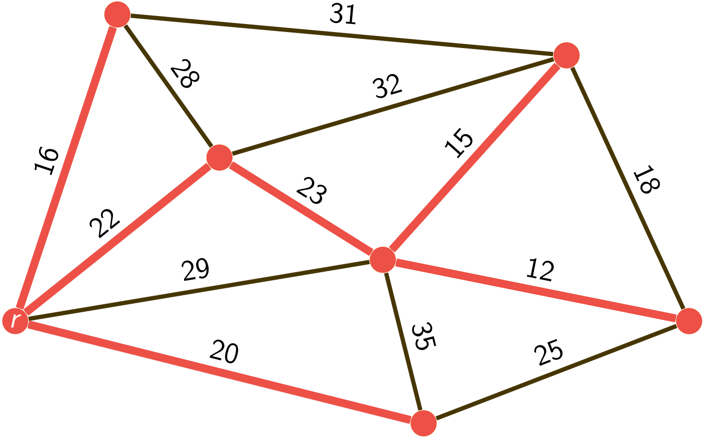
Example for Prim's Algorithm



Example for Prim's Algorithm



Example for Prim's Algorithm



Correctness of the MST Algorithms

Lemma 8.3.

A graph $G = (V, E)$ is connected if and only if there is no set $A \subseteq V$, $\emptyset \neq A \neq V$, with $\delta(A) = \emptyset$.

Proof: See exercise. $\square$

Notation: We say that $B \subseteq E$ is extendible to an MST if B is contained in the edge-set of some MST of G.

Theorem 8.4.

Let $B \subseteq E$ be extendible to an MST and $\emptyset \neq A \subsetneq V$ with $B \cap \delta(A) = \emptyset$. If e is a min-cost edge in $\delta(A)$, then $B \cup \{e\}$ is extendible to an MST.

Proof: See exercise. $\square$

- Correctness of Prim's Algorithm immediately follows.
- Kruskal: Whenever an edge $e = \{v, w\}$ is added, it is cheapest edge in cut induced by subset of nodes currently reachable from v.

Efficiency of Prim's Algorithm (Reminder)

Prim's Algorithm for MST

1. set $U := \{r\}$ for some node $r \in V$ and $F := \emptyset$; set $T := (U, F)$;
2. while $U \neq V$, determine a minimum cost edge $e \in \delta(U)$;
3. set $F := F \cup \{e\}$ and $U := U \cup \{w\}$ with $e = \{v, w\}$, $w \in V \setminus U$;

- Straightforward implementation achieves running time $O(nm)$ where, as usual, $n := |V|$ and $m := |E|$:
  - the while-loop has $n - 1$ iterations;
  - a min-cost edge $e \in \delta(U)$ can be found in $O(m)$ time.
- Idea for improved running time $O(n^2)$:
  - For each $v \in V \setminus U$, always keep a minimum cost edge $h(v)$ connecting v to some node in U.
  - In each iteration, information about all $h(v)$, $v \in V \setminus U$, can be updated in $O(n)$ time.
  - Find min-cost edge $e \in \delta(U)$ in $O(n)$ time by only considering the edges $h(v)$, $v \in V \setminus U$.
- Best running time: $O(m + n \log n)$ (Fibonacci heaps, e.g., CoMa II).

Efficiency of Kruskal's Algorithm (Reminder)

Kruskal's Algorithm for MST

1. sort the edges in E such that $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$;
2. set $T := (V, \emptyset)$;
3. for $i := 1$ to m do:

   If adding e_i to T does not create a circuit, then add e_i to T;

Theorem 8.5.

Step 3 of Kruskal's Algorithm can be implemented to run in $O(m \log^* m)$ time.

Proof: Use Union-Find datastructure; see, e.g., CoMa II. $\square$

Minimum Spanning Trees and Linear Programming

Notation:

- For $S \subseteq V$ let $\gamma(S) := \big\{e = \{v, w\} \in E \mid v, w \in S\big\}$.
- For a vector $x \in \mathbb{R}^E$ and a subset $B \subseteq E$ let $x(B) := \sum_{e \in B} x(e)$.

Consider the following integer linear program:

$$
\begin{aligned}
\min \quad & c^T \cdot x \\
\text{s.t.} \quad & x\big(\gamma(S)\big) \leq |S| - 1 && \text{for all } \emptyset \neq S \subset V && (8.1) \\
& x(E) = |V| - 1 && && (8.2) \\
& x(e) \in \{0, 1\} && \text{for all } e \in E
\end{aligned}
$$

Observations:

- Feasible solution $x \in \{0, 1\}^E$ is characteristic vector of subset $F \subseteq E$.
- F does not contain circuit due to (8.1) and $n - 1$ edges due to (8.2).
- Thus, F forms a spanning tree of G.
- Moreover, the edge set of an arbitrary spanning tree of G yields a feasible solution $x \in \{0, 1\}^E$.

Minimum Spanning Trees and Linear Programming (Cont.)

Consider LP relaxation of the integer programming formulation:

$$\begin{aligned}
\min \quad & c^T \cdot x \\
\text{s.t.} \quad & x\big(\gamma(S)\big) \leq |S| - 1 && \text{for all } \emptyset \neq S \subset V \\
& x(E) = |V| - 1 \\
& x(e) \geq 0 && \text{for all } e \in E
\end{aligned}$$

Theorem 8.6.

Let $x^* \in \{0,1\}^E$ be the characteristic vector of an MST. Then x^* is an optimal solution to the LP above.

Proof: ... $\square$

Corollary 8.7.

The vertices of the polytope given by the set of feasible LP solutions are exactly the characteristic vectors of spanning trees of G. The polytope is thus the convex hull of the characteristic vectors of all spanning trees.

Ingredients for the Proof of Theorem 8.6

primal LP:

$$\min \; c^T \cdot x$$

$$\text{s.t.} \quad x(\gamma(S)) \leq |S| - 1 \;\; \forall\, \emptyset \neq S \subsetneq V$$

$$x(E) = |V| - 1$$

$$x(e) \geq 0 \qquad \forall\, e \in E$$

dual LP:

$$\max \sum_{S: \emptyset \neq S \subseteq V} (|S| - 1) \cdot z_S$$

$$\text{s.t.} \sum_{S \subseteq V : e \in \gamma(S)} z_S \leq c(e) \quad \forall\, e \in E$$

$$z_S \leq 0 \quad \forall\, \emptyset \neq S \subsetneq V$$

$$z_V \quad \text{free}$$

Show that

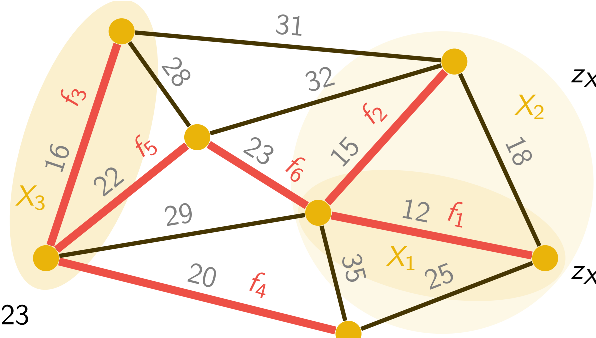
- characteristic vector x of spanning tree T found by Kruskal's Alg. is optimal solution to LP relaxation;
- to this end, construct also dual solution from Kruskal's Alg. such that complementary slackness conditions are fulfilled.

Ingredients for the Proof of Theorem 8.6 (Cont.)

Construction of dual solution:

- $E(T) = \{f_1, \dots, f_{n-1}\}$ with $c(f_1) \leq \cdots \leq c(f_{n-1})$;
- $X_k \subseteq V$ new connected component formed by f_k in Kruskal's Alg.;
- in particular, $X_{n-1} = V$;
- for $k = 1, \dots, n-2$, let $z_{X_k} := c(f_k) - c(f_\ell) \leq 0$,
  where f_ℓ is first edge after f_k (i.e., $\ell > k$) with $f_\ell \cap X_k \neq \emptyset$;
- $z_V := c(f_{n-1})$ and $z_X := 0$ for all $X \subseteq V$, $X \neq X_k$, $k = 1, \dots, n-1$.

Example:



$z_{X_2} = -8$

$z_{X_3} = -4$

$z_{X_1} = -3$

$z_V = z_{X_6} = 23$

Notation: Diwalks, Dipaths, Dicircuits etc.

Let $D = (V, A)$ be a digraph with arc costs c_a, $a \in A$.

- A diwalk P is a sequence

  $$v_0,\ a_1,\ v_1,\ a_2, \ldots, v_{k-1},\ a_k,\ v_k$$

  with $k \in \mathbb{N}$ und $a_i = (v_{i-1}, v_i) \in A$, for $i = 1, \ldots, k$.

  To emphasize start- and end-node of a diwalk we say v_0-v_k-diwalk.

  The length of a diwalk is $c(P) := \sum_{i=1}^{k} c_{a_i}$.

- If $v_0 = v_k$, the diwalk is closed.

- A diwalk is a dipath if $v_i \neq v_j$ for $0 \leq i < j \leq k$.

- A dicircuit is a closed diwalk with $k \geq 1$ and $v_i \neq v_j$ for $0 \leq i < j < k$.

For simplicity, we sometimes also use the terms walk, path, and circuit instead of diwalk, dipath, and dicircuit.

Shortest Path Problem (Reminder)

Given: digraph $D = (V, A)$, node $r \in V$, arc costs (lengths) c_a, $a \in A$;

Task: for each $v \in V$, find dipath from r to v of least cost (if one exists)

Remarks:

- Existence of r-v-dipath can be checked, e.g., by breadth-first search.
- Ensure existence of r-v-dipaths: add arcs (r, v) of suffic. large cost.

Basic idea behind all algorithms for solving shortest path problem:

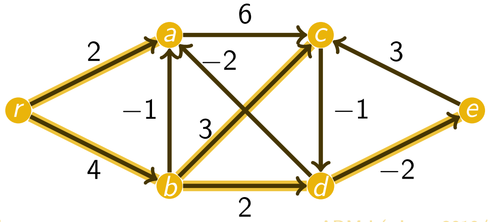
If y_v, $v \in V$, is the least cost of a diwalk from r to v, then

$$y_v + c_{(v,w)} \geq y_w \quad \text{for all } (v, w) \in A.$$

Elementary Facts for Shortest Paths (Reminder)

- Subwalks of shortest walks are shortest walks!
- If a shortest r-v-walk contains a closed subwalk (e.g., circuit), the closed subwalk has cost 0.
- A shortest r-v-walk always contains a shortest r-v-path of equal length.
- If there is a shortest r-v-walk for all $v \in V$, then there is a shortest path tree, i.e., an arborescence T rooted at r such that the unique r-v-path in T is a least-cost r-v-walk in D.

Example: A shortest path tree.



$p(r) = \text{None}$

$p(a) = r$

$p(b) = r$

$p(c) = b$

$p(d) = b$

$p(e) = d$

Feasible Potentials (Reminder)

Definition 8.8.

A vector $y \in \mathbb{R}^V$ is a feasible potential if

$$y_v + c_{(v,w)} \geq y_w \quad \text{for all } (v, w) \in A.$$

Lemma 8.9.

If y is feasible potential with $y_r = 0$ and P an r-v-walk, then $y_v \leq c(P)$.

Proof: Suppose that P is $v_0, a_1, v_1, \ldots, a_k, v_k$, where $v_0 = r$ and $v_k = v$. Then,

$$c(P) = \sum_{i=1}^{k} c_{a_i} \geq \sum_{i=1}^{k} (y_{v_i} - y_{v_{i-1}}) = y_{v_k} - y_{v_0} = y_v.$$

□

Corollary 8.10.

If y is a feasible potential with $y_r = 0$ and P an r-v-walk of cost y_v, then P is a least-cost r-v-walk. □

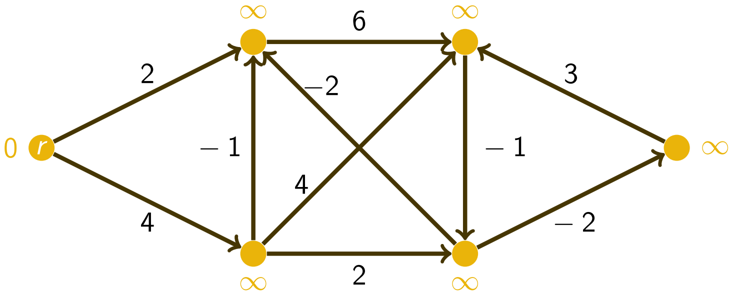
Ford's Algorithm

Ford's Algorithm (Reminder)

1. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
2. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



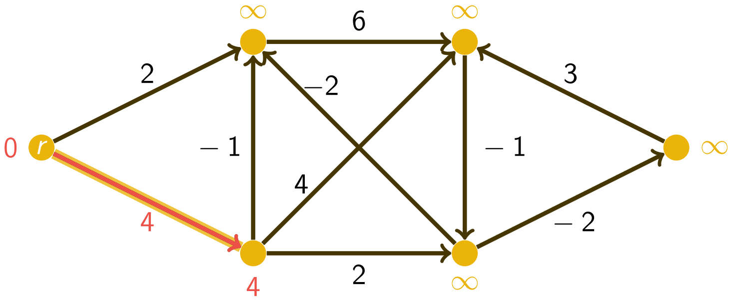
Ford's Algorithm

Ford's Algorithm (Reminder)

1. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) := \text{null}$, for all $v \in V \setminus \{r\}$.
2. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



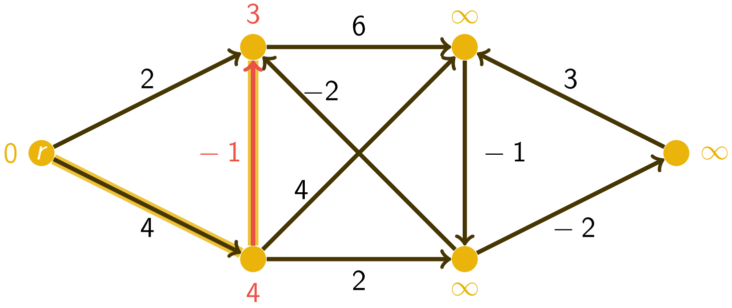
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



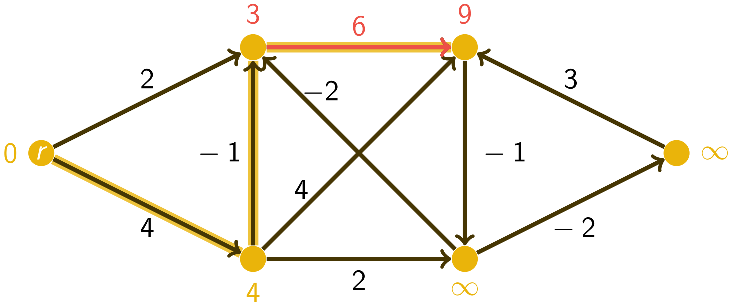
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



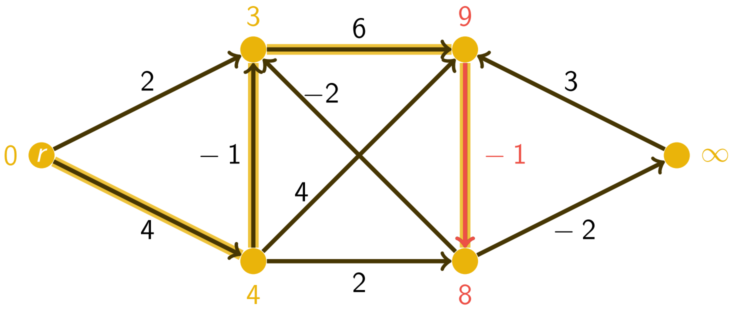
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



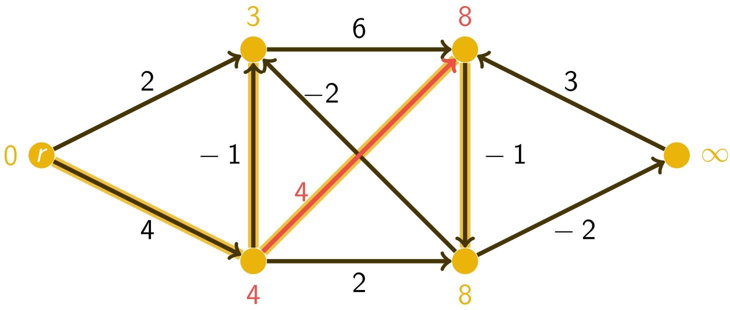
Ford's Algorithm

Ford's Algorithm (Reminder)

- i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
- ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



Ford's Algorithm

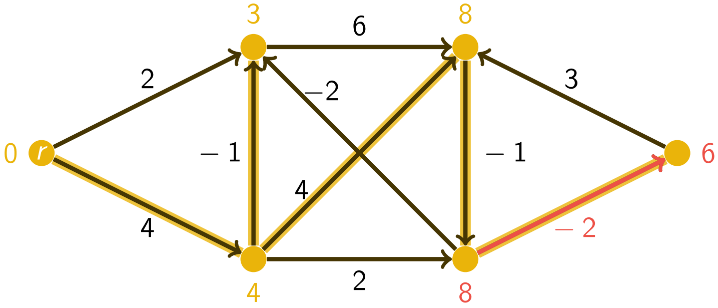
Ford's Algorithm (Reminder)

i Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.

ii While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



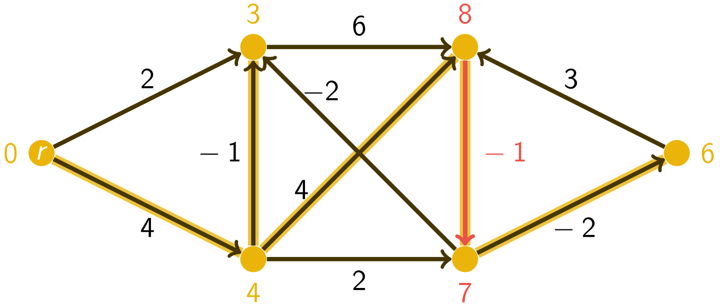
Ford's Algorithm

Ford's Algorithm (Reminder)

- i Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
- ii While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



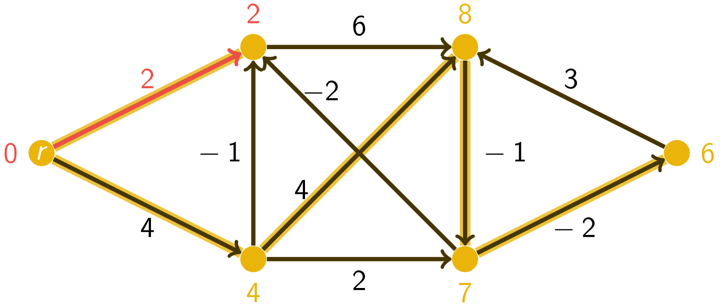
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) := \text{null}$, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



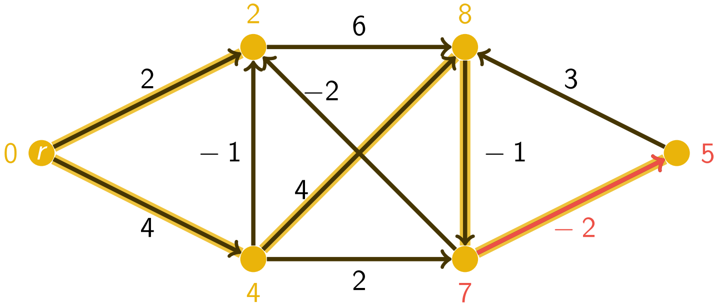
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



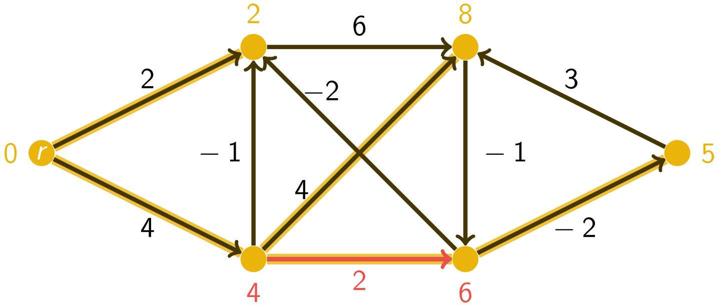
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



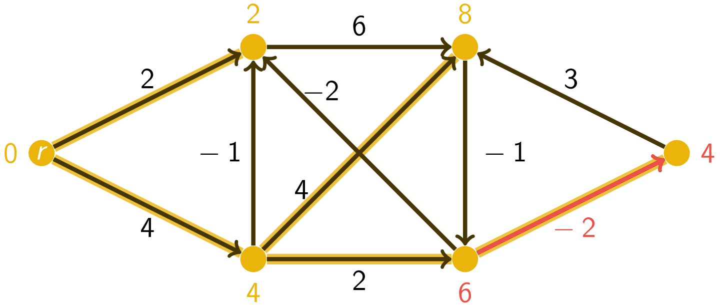
Ford's Algorithm

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

Example:



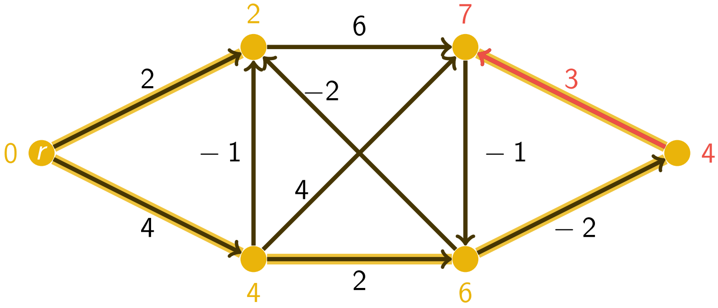
Ford's Algorithm

Ford's Algorithm (Reminder)

1. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) :=$ null, for all $v \in V \setminus \{r\}$.
2. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

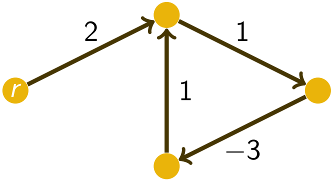
Example:



Termination of Ford's Algorithm (Reminder)

Question: Does the algorithm always terminate?

Example:



Observation:
The algorithm does not terminate because of the negative-cost dicircuit.

Validity of Ford's Algorithm (Reminder)

Lemma 8.11.

If there is no negative-cost dicircuit, then at any stage of the algorithm:

- a) if $y_v \neq \infty$, then y_v is the cost of some r-v-path;
- b) if $p(v) \neq$ null, then p defines a r-v-path of cost at most y_v.

Proof: See CoMa II. $\square$

Theorem 8.12.

If there is no negative-cost dicircuit, then Ford's Algorithm terminates after a finite number of iterations. At termination, y is a feasible potential with $y_r = 0$ and, for each node $v \in V$, p defines a least-cost r-v-dipath.

Proof: See CoMa II. $\square$

Feasible Potentials & Negative-Cost Dicircuits (Reminder)

Theorem 8.13.

A digraph $D = (V, A)$ with arc costs $c \in \mathbb{R}^A$ has a feasible potential if and only if there is no negative-cost dicircuit.

Proof: See CoMa II. $\square$

Remarks:

- If there is a dipath but no least-cost diwalk from r to v, it is because there are arbitrarily cheap r-v-diwalks.
- In this case, finding least-cost dipath from r to v is, however, difficult (i.e., NP-hard; see later).

Lemma 8.14.

If c is integer-valued, $C := 2\max_{a \in A} |c_a| + 1$, and there is no negative-cost dicircuit, then Ford's Algorithm terminates after at most $C\,n^2$ iterations.

Proof: See CoMa II. $\square$

Feasible Potentials and Linear Programming

As a consequence of Ford's Algorithm we get:

Theorem 8.15.

Let $D = (V, A)$ be a digraph, $r, s \in V$, and $c \in \mathbb{R}^A$. If, for every $v \in V$, there exists a least-cost diwalk from r to v, then

$$\min\{c(P) \mid P \text{ an } r\text{-}s\text{-dipath}\} = \max\{y_s - y_r \mid y \text{ a feasible potential}\}.$$

Formulate the right-hand side as a linear program and consider the dual:

$$\begin{array}{ll} \max & y_s - y_r \\ \text{s.t.} & y_w - y_v \le c_{(v,w)} \\ & \quad \text{for all } (v,w) \in A \end{array} \qquad \begin{array}{ll} \min & c^T \cdot x \\ \text{s.t.} & \displaystyle\sum_{a \in \delta^-(v)} x_a - \sum_{a \in \delta^+(v)} x_a = b_v \quad \forall v \in V \\ & x_a \ge 0 \qquad \text{for all } a \in A \end{array}$$

with $b_s = 1$, $b_r = -1$, and $b_v = 0$ for all $v \notin \{r, s\}$.

Notice: The dual is the LP relaxation of an ILP formulation of the shortest r-s-diwalk problem ($x_a \mathrel{\hat{=}}$ number of times a shortest r-s-diwalk uses arc a).

Bases of Shortest Path LP

Consider again the dual LP:

$$\begin{array}{lll} \min & c^T \cdot x & \\ \text{s.t.} & \displaystyle\sum_{a\in\delta^-(v)} x_a - \sum_{a\in\delta^+(v)} x_a = b_v & \text{for all } v \in V \\ & x_a \geq 0 \qquad \text{for all } a \in A & \end{array}$$

The underlying matrix Q is the incidence matrix of D.

Lemma 8.16.

Let $D = (V, A)$ be a connected digraph and Q its incidence matrix. A subset of columns of Q indexed by a subset of arcs $F \subseteq A$ forms a basis of the linear subspace of $\mathbb{R}^n$ spanned by the columns of Q if and only if F is the arc-set of a spanning tree of D.

Proof: Exercise. $\square$

Refinement of Ford's Algorithm (Reminder)

Ford's Algorithm (Reminder)

i. Set $y_r := 0$, $p(r) := r$, $y_v := \infty$, and $p(v) := \text{null}$, for all $v \in V \setminus \{r\}$.
ii. While there is an arc $a = (v, w) \in A$ with $y_w > y_v + c_{(v,w)}$, set

$$y_w := y_v + c_{(v,w)} \quad \text{and} \quad p(w) := v.$$

- \# iterations crucially depends on order in which arcs are chosen.
- Suppose that arcs are chosen in order $\mathcal{S} = f_1, f_2, f_3, \ldots, f_\ell$.
- Diwalk P is embedded in $\mathcal{S}$ if P's arc sequence is a subsequence of $\mathcal{S}$.

Lemma 8.17.

If an r-v-diwalk P is embedded in $\mathcal{S}$, then $y_v \leq c(P)$ after Ford's Algorithm has gone through the sequence $\mathcal{S}$.

Proof: See CoMa II. □

Goal: Find short sequence $\mathcal{S}$ such that a least-cost r-v-diwalk is embedded in $\mathcal{S}$ for all $v \in V$.

Ford-Bellman Algorithm (Reminder)

Basic idea:

- Every dipath is embedded in $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n-1}$ where, for all i, $\mathcal{S}_i$ is an ordering of A.
- This yields a shortest path algorithm with running time $O(nm)$.

Ford-Bellman Algorithm

i initialize y, p (see Ford's Algorithm);

ii for $i = 1$ to $n - 1$ do

iii   for all $a = (v, w) \in A$ do

iv   if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Theorem 8.18.

The algorithm runs in $O(nm)$ time. If, at termination, y is a feasible potential, then p yields a least-cost r-v-dipath for each $v \in V$. Otherwise, the given digraph contains a negative-cost dicircuit. □

Acyclic Digraphs and Topological Orderings (Reminder)

Definition 8.19.

Consider a digraph $D = (V, A)$.

- a An ordering $v_1, v_2, \ldots, v_n$ of V so that $i < j$ for each $(v_i, v_j) \in A$ is called a topological ordering of D.
- b If D has a topological ordering, then D is called acyclic.

Observations:

- Digraph D is acyclic if and only if it does not contain a dicircuit.
- Topological ordering of D can be found in time $O(n + m)$ (if it exists).
- Let D be acyclic and $\mathcal{S}$ an ordering of A such that (v_i, v_j) precedes (v_k, v_ℓ) if $i < k$. Then every dipath of D is embedded in $\mathcal{S}$.

Theorem 8.20.

Shortest path problem on acyclic digraphs can be solved in time $O(n + m)$.

Proof: See CoMa II. □

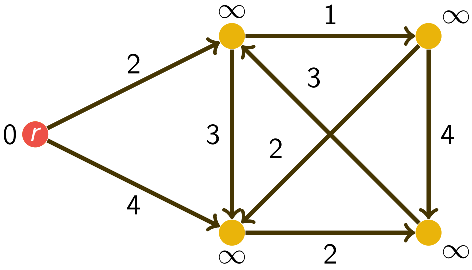
Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

- i initialize y, p (see Ford's Algorithm); set $S := V$;
- ii while $S \neq \emptyset$ do
- iii   choose $v \in S$ with y_v minimum and delete v from S;
- iv   for each $w \in V$ with $(v, w) \in A$ do
- v   if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



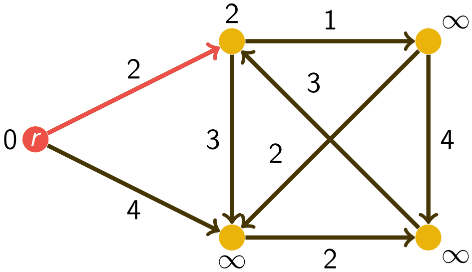
Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

- i initialize y, p (see Ford's Algorithm); set $S := V$;
- ii while $S \neq \emptyset$ do
- iii choose $v \in S$ with y_v minimum and delete v from S;
- iv for each $w \in V$ with $(v, w) \in A$ do
- v if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



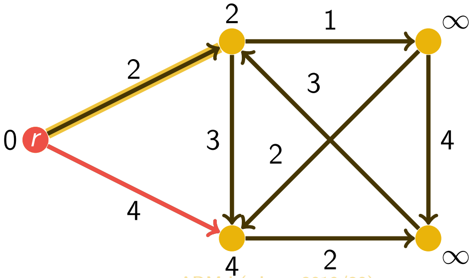
Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

- i initialize y, p (see Ford's Algorithm); set $S := V$;
- ii while $S \neq \emptyset$ do
- iii choose $v \in S$ with y_v minimum and delete v from S;
- iv for each $w \in V$ with $(v, w) \in A$ do
- v   if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



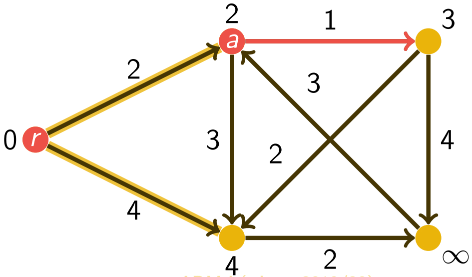
Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

- i initialize y, p (see Ford's Algorithm); set $S := V$;
- ii while $S \neq \emptyset$ do
- iii choose $v \in S$ with y_v minimum and delete v from S;
- iv for each $w \in V$ with $(v, w) \in A$ do
- v if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



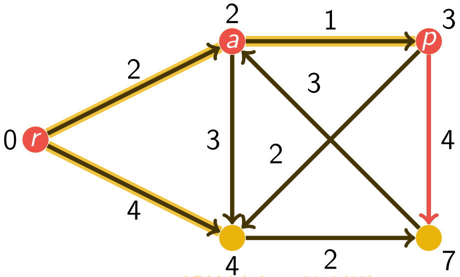
Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

- i initialize y, p (see Ford's Algorithm); set $S := V$;
- ii while $S \neq \emptyset$ do
- iii   choose $v \in S$ with y_v minimum and delete v from S;
- iv   for each $w \in V$ with $(v, w) \in A$ do
- v   if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

i initialize y, p (see Ford's Algorithm); set $S := V$;

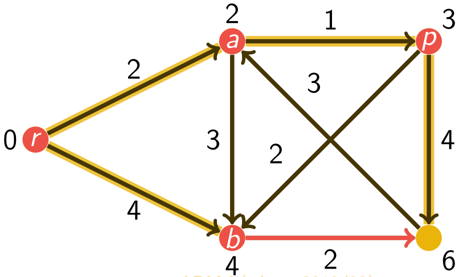
ii while $S \neq \emptyset$ do

iii choose $v \in S$ with y_v minimum and delete v from S;

iv for each $w \in V$ with $(v, w) \in A$ do

v if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



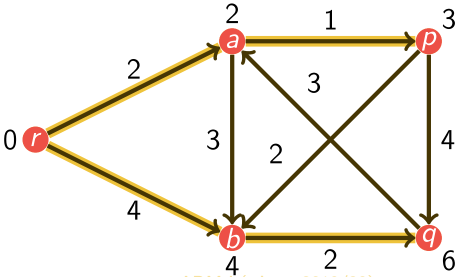
Dijkstra's Algorithm (Reminder)

Consider the special case of nonnegative costs, i.e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

i. initialize y, p (see Ford's Algorithm); set $S := V$;
ii. while $S \neq \emptyset$ do
iii. choose $v \in S$ with y_v minimum and delete v from S;
iv. for each $w \in V$ with $(v, w) \in A$ do
v. if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$;

Example:



Correctness of Dijkstra's Algorithm (Reminder)

Lemma 8.21.

For each $w \in V$, let y'_w be the value of y_w when w is removed from S. If u is deleted from S before v, then $y'_u \leq y'_v$.

Proof: See CoMa II. □

Theorem 8.22.

If $c \geq 0$, then Dijkstra's Algorithm solves the shortest paths problem correctly in time $O(n^2)$. A heap-based implementation yields running time $O(m \log n)$ or even $O(m + n \log n)$ (Fibonacci-Heap).

Proof: See CoMa II. □

Remark: The for-loop in Dijkstra's Algorithm (step iv) can be modified such that only arcs (v, w) with $w \in S$ are considered.

Feasible Potentials and Nonnegative Costs (Reminder)

Observation 8.23.

For given arc costs $c \in \mathbb{R}^A$ and node potential $y \in \mathbb{R}^V$, define arc costs $c' \in \mathbb{R}^A$ by $c'_{(v,w)} := c_{(v,w)} + y_v - y_w$. Then, for all $v, w \in V$, a least-cost v-w-diwalk w.r.t. c is a least-cost v-w-diwalk w.r.t. c', and vice versa.

Proof: Notice that for any v-w-diwalk P it holds that

$$c'(P) = c(P) + y_v - y_w. \qquad \square$$

Corollary 8.24.

For given arc costs $c \in \mathbb{R}^A$ (not necessarily nonnegative) and a given feasible potential $y \in \mathbb{R}^V$, one can use Dijkstra's Algorithm to solve the shortest paths problem. $\square$

Definition 8.25.

For a digraph $D = (V, A)$, arc costs $c \in \mathbb{R}^A$ are called conservative if there is no negative-cost dicircuit in D, i.e., if there is feasible potential $y \in \mathbb{R}^V$.

All Pairs Shortest Paths Problem

Given: digraph $D = (V, A)$, conservative arc costs (lengths) c_a, $a \in A$;

Task: for all $r, v \in V$ with $r \neq v$, find r-v-dipath of least cost (if it exists)

Simple algorithm: Call Ford-Bellman Algorithm for each start node $r \in V$.
$\Longrightarrow$ running time $O(mn^2)$

Better algorithm:

Theorem 8.26.

All Pairs Shortest Paths Problem can be solved in $O(mn + n^2 \log n)$ time.

Proof:
Use Ford-Bellman Algorithm to compute feasible potential in $O(mn)$ time.
Call Dijkstra's Algo. ($O(m + n \log n)$ time) for each start node $r \in V$. $\square$

Alternative approach: Floyd-Warshall Algorithm (see exercise session)

Side Note:
Definition of Efficient Algorithms

Efficient Algorithms

What is an efficient algorithm?

- efficient: consider running time
- algorithm: Turing Machine or other formal model of computation

Simplified Definition

An algorithm consists of

- "elementary steps" like, e.g., variable assignments
- simple arithmetic operations

which only take a constant amount of time. The running time of the algorithm on a given input is the number of such steps and operations.

Bit Model and Arithmetic Model

Two ways of measuring the running time and the size of the input I of A:

Bit Model	Arithmetic Model
Count bit operations; e.g., adding two n-bit numbers takes $n(+1)$ steps; multiplying them takes $O(n^2)$ steps.	Simple arithmetic operations on arbitrary numbers can be performed in constant time.
Size of input I is the total number of bits needed to encode "structure" and numbers.	Size of input I is total number of bits needed to encode "structure" plus # numbers in the input.

Polynomial vs. Strongly Polynomial Running Time

Definition 8.27.

1. An algorithm *runs in polynomial time* if, in the bit model, its (worst-case) running time is polynomially bounded in the input size.
2. An algorithm *runs in strongly polynomial time* if, in the bit model as well as in the arithmetic model, its (worst-case) running time is polynomially bounded in the input size.

Examples:

- Prim's and Kruskal's Algorithm as well as the Ford-Bellman Algorithm and Dijkstra's Algorithm run in strongly polynomial time.
- The Euclidean Algorithm runs in polynomial time but not in strongly polynomial time.

Example: Arithmetic Model vs. Bit Model

Consider the following algorithm:

Input: n numbers $a_1, \dots, a_n \in \mathbb{Z}_{>0}$

1. for $i = 1$ to n:
2. $\qquad a_1 := a_1 \cdot a_1$
3. output a_1

Arithmetic Model:

- Input size is n; running time is $O(n)$ (polynomial, even linear).

Bit Model:

- Input size is $\sum_{i=1}^{n}(\lfloor \log a_i \rfloor + 1)$
- Encoding size of the computed output ${a_1}^{2^n}$ is $\lfloor 2^n \log a_i \rfloor + 1$.
- Output size can thus be exponential in the input size (e.g., if $a_1 \geq a_i$ for all $i = 1, \dots, n$).
- Notice that the output size is a lower bound on the running time.

Pseudopolynomial Running Time

- In the bit model, we assume that numbers are binary encoded, i.e., the encoding of the number $n \in \mathbb{N}$ needs $\lfloor \log n \rfloor + 1$ bits.
- Thus, the running time bound $O(C\, n^2)$ of Ford's Algorithm where $C := 2 \max_{a \in A} |c_a| + 1$ is not polynomial in the input size.
- If we assume, however, that numbers are unary encoded, then $C\, n^2$ is polynomially bounded in the input size.

Definition 8.28.

An algorithm runs in pseudopolynomial time if, in the bit model with unary encoding of numbers, its (worst-case) running time is polynomially bounded in the input size.

Example:

Checking whether a given number $a \in \mathbb{Z}_{\geq 2}$ is prime by testing for all $1 < b < a$ whether b divides a is a pseudopolynomial time algorithm.