



SAPIENZA
UNIVERSITÀ DI ROMA

Network Optimization

Volume 2 — Trees and paths

Minimum spanning trees (Kruskal, Prim, the cut and cycle properties) and shortest paths (acyclic graphs, Bellman–Ford, Dijkstra), with potentials as a certificate of optimality.

Fabio Furini

fabiofurini.github.io

Department of Computer, Control and Management Engineering — Sapienza University of Rome

Online version and downloadable material:
fabiofurini.github.io/network-optimization/
github.com/fabiofurini/network-optimization

Academic year 2026–2027

Contents

2	Minimum spanning trees	2
2.1	The minimum spanning tree problem	3
2.2	Cuts and cycles	5
2.3	The cut property and the cycle property	6
2.4	The generic greedy algorithm	9
2.5	Kruskal's algorithm	10
2.6	Prim's algorithm	13
2.7	Uniqueness of the minimum spanning tree	15
2.8	Typical mistakes	17
2.9	In summary	17
3	Shortest paths	19
3.1	Shortest path problems	20
3.2	Optimality conditions and potentials	23
3.3	The linear programming formulation	25
3.4	Shortest paths in acyclic graphs	28
3.5	The Bellman–Ford algorithm	30
3.6	Dijkstra's algorithm	35
3.7	Typical mistakes	37
3.8	In summary	38



Minimum spanning trees

Prerequisites

- The chapter [«Graphs»](#): paths and cycles, connected components, the characterizations of a tree and the fundamental cycles.
- The $O(\cdot)$ notation for the running time of an algorithm, and sorting m numbers in $O(m \log m)$ time.

Objectives

At the end of this chapter you will be able to:

- state the minimum spanning tree problem and recognize it in applications;
- prove the cut property and the cycle property, and use them to certify that a spanning tree is optimal;
- run the algorithms of Kruskal and of Prim, prove that they are correct and analyse their running time;
- decide whether the minimum spanning tree is unique, and find the edges that belong to every optimal tree.

2.1 The minimum spanning tree problem

Definition 2.1: minimum spanning tree problem

Let $G = (V, E)$ be a connected undirected graph with $n = |V|$ vertices and $m = |E|$ edges, and let every edge $e \in E$ have a **weight** $w_e \in \mathbb{R}$. The **weight** of a set of edges $F \subseteq E$ is

$$w(F) = \sum_{e \in F} w_e.$$

The **minimum spanning tree problem** (MST) asks for a spanning tree $\tilde{T} = (V, \tilde{F})$ of G of minimum weight:

$$w(\tilde{F}) \leq w(F) \quad \text{for every spanning tree } (V, F) \text{ of } G.$$

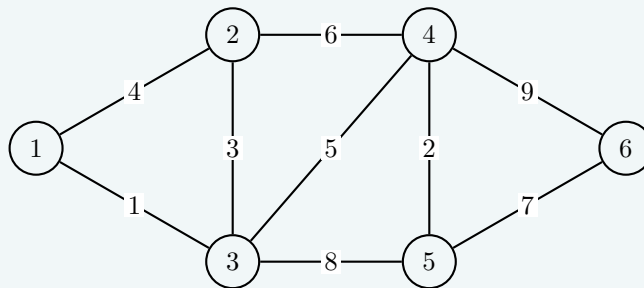
A spanning tree of minimum weight is a **minimum spanning tree**. We write w_{uv} for the weight of the edge $\{u, v\}$, and we identify a spanning tree with its set of edges.

- A connected graph has a spanning tree, and it has finitely many of them: a minimum spanning tree always exists. Every spanning tree has $n - 1$ edges.
- The weights can be negative. Adding the same constant to every weight adds $(n - 1)$ times that constant to the weight of every spanning tree, so the minimum spanning trees do not change.
- A **maximum spanning tree** is a minimum spanning tree for the weights $-w_e$.

Example 2.1: the weighted reference graph

We take the underlying undirected graph of the reference graph of the chapter «Graphs», with $n = 6$ vertices, $m = 9$ edges and the weights

edge	$\{1, 2\}$	$\{1, 3\}$	$\{2, 3\}$	$\{2, 4\}$	$\{3, 4\}$	$\{3, 5\}$	$\{4, 5\}$	$\{4, 6\}$	$\{5, 6\}$
w_e	4	1	3	6	5	8	2	9	7

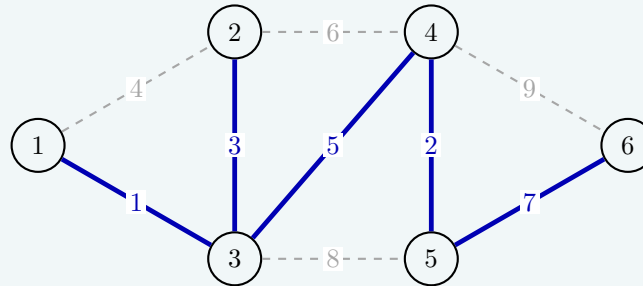


We call this graph the **weighted reference graph**: all the examples of the chapter use it. The weights are the distinct integers $1, 2, \dots, 9$.

Example 2.2: the minimum spanning tree of the reference graph

The weighted reference graph has 55 spanning trees. Enumerating all of them, the minimum weight is 18, attained only by the tree (blue in the figure)

$$\tilde{F} = \{\{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}\}, \quad w(\tilde{F}) = 1 + 3 + 5 + 2 + 7 = 18.$$



The spanning tree $\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 5\}, \{4, 6\}$ found by the breadth-first search in the chapter «Graphs» weighs $4 + 1 + 6 + 8 + 9 = 28$, and a maximum spanning tree weighs 34.

- Enumerating the spanning trees is practical only on tiny graphs: the reference graph, with 6 vertices and 9 edges, already has 55. The algorithms of this chapter find a minimum spanning tree in polynomial time.

2.1.1 Applications

- **Network design.** Vertices are sites (offices, houses, plants) and w_{uv} is the cost of a direct link (a cable, a pipe, a road); the cheapest network connecting all the sites is a minimum spanning tree.
- **Clustering.** Removing the $k - 1$ heaviest edges of a minimum spanning tree splits the vertices into k groups of close points (single-linkage clustering).
- **Subproblem.** Minimum spanning trees give lower bounds for the travelling salesman problem and a starting point for heuristic algorithms for network design.

A minimum spanning tree connects all the vertices of a connected weighted graph with $n - 1$ edges of minimum total weight; on the weighted reference graph it is unique and weighs 18.

2.2 Cuts and cycles

Definition 2.2: cut

Let $G = (V, E)$ be an undirected graph and $S \subseteq V$ with $\emptyset \neq S \neq V$. The **cut** defined by S is the set of the edges with exactly one endpoint in S :

$$\delta(S) = \{ \{u, v\} \in E : u \in S, v \notin S \}.$$

An edge of $\delta(S)$ **crosses** the cut. For a single vertex, $\delta(\{v\}) = \delta(v)$ is the set of the edges incident to v .

- A graph is connected if and only if every cut $\delta(S)$ is nonempty: an empty cut separates S from $V \setminus S$, and a path from $u \in S$ to $v \notin S$ must use an edge of $\delta(S)$.
- In the weighted reference graph, $\delta(\{1, 2, 3\}) = \{\{2, 4\}, \{3, 4\}, \{3, 5\}\}$ and $\delta(\{6\}) = \{\{4, 6\}, \{5, 6\}\}$.

Lemma 2.3: a cycle crosses a cut an even number of times

Let C be a cycle of G and $\delta(S)$ a cut. Then $|C \cap \delta(S)|$ is even. In particular, if a cycle contains an edge of $\delta(S)$, it contains at least one other edge of $\delta(S)$.

Proof. Let the cycle visit $v_0, v_1, \dots, v_k = v_0$. The edge $\{v_{t-1}, v_t\}$ crosses the cut if and only if v_{t-1} and v_t lie on different sides. Walking along the cycle, the side changes at every crossing edge and only there; the walk ends at v_0 , on the side where it started, so the number of changes is even. $\square$

Definition 2.4: fundamental cut

Let T be a spanning tree of a connected graph $G = (V, E)$ and $e = \{u, v\} \in T$. The graph $T \setminus \{e\}$ has two connected components; let S_e be the one that contains u . The **fundamental cut** of e is $\delta(S_e)$.

- $T \setminus \{e\}$ has $n - 2$ edges and no cycle, so it has $n - (n - 2) = 2$ components ([Corollary «edges and components»](#)); v is not in S_e , otherwise the path from u to v in T would not be unique.
- The only edge of T in the fundamental cut of e is e itself: every other edge of T lies inside one of the two components.

Lemma 2.5: exchange

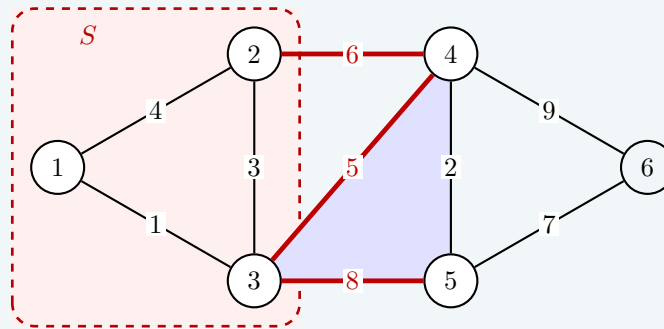
Let T be a spanning tree of a connected graph $G = (V, E)$.

1. If $e \notin T$ and f is an edge of the fundamental cycle $C(T, e)$, then $T \cup \{e\} \setminus \{f\}$ is a spanning tree.
2. If $f \in T$ and $e \notin T$ crosses the fundamental cut of f , then $f \in C(T, e)$, and $T \cup \{e\} \setminus \{f\}$ is a spanning tree.

- Proof.* 1. The rest of $C(T, e)$ joins the endpoints of f , so removing f from the connected graph $T \cup \{e\}$ leaves it connected. The result is connected with $n - 1$ edges: it is a spanning tree by the [characterizations of a tree](#), (b).
2. The endpoints of e lie on different sides of the fundamental cut $\delta(S_f)$. The path between them in T is part of $C(T, e)$, and it must cross $\delta(S_f)$ with an edge of T ; the only one is f . Part 1 concludes. $\square$

Example 2.3: cuts and cycles of the reference graph

The figure shows the cut $\delta(\{1, 2, 3\})$ (red edges) and the cycle 3, 4, 5 (shaded in blue): the cycle uses the edges $\{3, 4\}$ and $\{3, 5\}$ of the cut, an even number.



For the minimum spanning tree $\tilde{F}$ of [Example 2.2](#), the fundamental cut of $\{3, 4\}$ is $\delta(\{1, 2, 3\})$, and the fundamental cycle of $\{3, 5\}$ is $C(\tilde{F}, \{3, 5\}) = 3, 4, 5, 3$.

A cycle crosses every cut an even number of times. Adding an edge to a spanning tree and removing any edge of the cycle it closes gives another spanning tree.

2.3 The cut property and the cycle property

Theorem 2.6: cut property

Let $G = (V, E)$ be a connected graph with weights w_e , and let $F \subseteq E$ be contained in some minimum spanning tree. Let $\delta(S)$ be a cut with $F \cap \delta(S) = \emptyset$, and let e be an edge of minimum weight in $\delta(S)$.

- (a) $F \cup \{e\}$ is contained in some minimum spanning tree.
- (b) If $w_e < w_f$ for every $f \in \delta(S) \setminus \{e\}$, then e belongs to every minimum spanning tree.

Proof. Let T^* be a minimum spanning tree with $F \subseteq T^*$; for (b), let T^* be any minimum spanning tree and $F = \emptyset$. If $e \in T^*$ there is nothing to prove; let $e \notin T^*$.

- 1. The fundamental cycle $C(T^*, e)$ contains the edge e of $\delta(S)$, so it contains another

edge $f \in \delta(S)$, $f \neq e$ (Lemma 2.3). All its edges other than e are in T^* , so $f \in T^*$; and $f \notin F$, since $F \cap \delta(S) = \emptyset$.

2. By Lemma 2.5, $T' = T^* \cup \{e\} \setminus \{f\}$ is a spanning tree, with $w(T') = w(T^*) + w_e - w_f$.
3. (a) e has minimum weight in $\delta(S)$, so $w_e \leq w_f$ and $w(T') \leq w(T^*)$: T' is a minimum spanning tree containing $F \cup \{e\}$.
4. (b) Here $w_e < w_f$ and $w(T') < w(T^*)$, against the optimality of T^* : hence $e \in T^*$. $\square$

Theorem 2.7: cycle property

Let $G = (V, E)$ be a connected graph with weights w_e , let C be a cycle of G and e an edge of maximum weight in C .

- (a) Some minimum spanning tree does not contain e .
- (b) If $w_e > w_f$ for every $f \in C \setminus \{e\}$, then no minimum spanning tree contains e .

Proof. Let T^* be a minimum spanning tree with $e \in T^*$ (if there is none, both claims hold), and let $\delta(S_e)$ be the fundamental cut of e in T^* .

1. The cycle C contains the edge e of $\delta(S_e)$, so it contains another edge $f \in \delta(S_e)$ (Lemma 2.3); $f \notin T^*$, because e is the only edge of T^* in its fundamental cut.
2. By Lemma 2.5(2), $T' = T^* \cup \{f\} \setminus \{e\}$ is a spanning tree, with $w(T') = w(T^*) + w_f - w_e$.
3. (a) $w_f \leq w_e$, so T' is a minimum spanning tree without e . (b) $w_f < w_e$ would give $w(T') < w(T^*)$, a contradiction: no minimum spanning tree contains e . $\square$

Example 2.4: the two properties on the reference graph

- *Cut property.* The edge $\{3, 4\}$, of weight 5, is the unique lightest edge of $\delta(\{1, 2, 3\}) = \{\{2, 4\}, \{3, 4\}, \{3, 5\}\}$, with weights 6, 5, 8: it belongs to every minimum spanning tree. In the same way, $\{5, 6\}$ is the unique lightest edge of $\delta(\{6\})$.
- *Cycle property.* The edge $\{1, 2\}$ is the unique heaviest edge of the cycle 1, 2, 3 (weights 4, 3, 1), and $\{3, 5\}$ is the unique heaviest edge of the cycle 3, 4, 5 (weights 5, 2, 8): no minimum spanning tree contains them. The cycles 2, 3, 4 and 4, 5, 6 exclude $\{2, 4\}$ and $\{4, 6\}$.
- The four excluded edges leave exactly the 5 edges of $\tilde{F}$: the two properties alone prove that $\tilde{F}$ is the unique minimum spanning tree.

2.3.1 Optimality conditions

Theorem 2.8: optimality conditions

Let T be a spanning tree of a connected graph $G = (V, E)$ with weights w_e . The following statements are equivalent:

- (i) T is a minimum spanning tree;
- (ii) **cycle condition**: for every $e \notin T$, $w_e \geq w_f$ for every edge f of $C(T, e)$;
- (iii) **cut condition**: for every $e \in T$, $w_e \leq w_f$ for every edge f of the fundamental cut of e .

Proof. We prove (i) $\Rightarrow$ (iii) $\Rightarrow$ (ii) $\Rightarrow$ (i).

- (i) $\Rightarrow$ (iii). If f crosses the fundamental cut of $e \in T$ with $w_f < w_e$, then $T \cup \{f\} \setminus \{e\}$ is a spanning tree (Lemma 2.5(2)) lighter than T .
- (iii) $\Rightarrow$ (ii). Let $e \notin T$ and $f \in C(T, e)$, $f \neq e$. The path of T between the endpoints of e uses f , and it is unique, so removing f separates them: e crosses the fundamental cut of f , and $w_f \leq w_e$ by (iii).

(ii) $\Rightarrow$ (i). Among the minimum spanning trees, let T^* have the largest number of edges in common with T , and assume $T^* \neq T$.

1. Take $e \in T^* \setminus T$, and let $\delta(S_e)$ be its fundamental cut in T^* . The cycle $C(T, e)$ contains e , so it contains another edge $f \in \delta(S_e)$ (Lemma 2.3), with $f \in T$ and $f \notin T^*$.
2. By Lemma 2.5(2), $T' = T^* \cup \{f\} \setminus \{e\}$ is a spanning tree, and $w_f \leq w_e$ by (ii) applied to $e \notin T$. Hence $w(T') \leq w(T^*)$.
3. T' is a minimum spanning tree with one more edge in common with T : a contradiction. So $T = T^*$. $\square$

Example 2.5: certifying the optimal tree

For the tree $\tilde{F}$ of Example 2.2, the four edges outside the tree and their fundamental cycles are:

edge $e \notin \tilde{F}$	$C(\tilde{F}, e)$	weights of the tree edges of the cycle
$\{1, 2\}, w = 4$	1, 3, 2, 1	1, 3
$\{2, 4\}, w = 6$	2, 3, 4, 2	3, 5
$\{3, 5\}, w = 8$	3, 4, 5, 3	5, 2
$\{4, 6\}, w = 9$	4, 5, 6, 4	2, 7

Every edge outside the tree is at least as heavy as the tree edges of its cycle: by Theorem 2.8 the tree is optimal, without enumerating the 55 spanning trees.

Example 2.6: improving a tree that is not optimal

The BFS tree $T = \{\{1, 2\}, \{1, 3\}, \{2, 4\}, \{3, 5\}, \{4, 6\}\}$, of weight 28, violates the cycle condition. For $e = \{4, 5\}$, of weight 2, the fundamental cycle $4, 2, 1, 3, 5, 4$ contains $\{2, 4\}$, $\{1, 2\}$ and $\{3, 5\}$, of weights 6, 4, 8.

- Removing the heaviest of them, $\{3, 5\}$, and adding $\{4, 5\}$ gives a spanning tree of weight $28 + 2 - 8 = 22$.
- The proof of [Theorem 2.8](#) turns into an algorithm: repeat the exchange while an edge violates the condition. The algorithms of the next sections are faster.

The lightest edge of a cut belongs to some minimum spanning tree, and the heaviest edge of a cycle can be left out. A spanning tree is optimal if and only if every edge outside it is the heaviest of its fundamental cycle.

2.4 The generic greedy algorithm

- The cut property suggests building a minimum spanning tree one edge at a time: the current set F is always contained in some minimum spanning tree, and the next edge is the lightest edge of a cut not crossed by F .

Algorithm

Algorithm: Generic greedy algorithm for the MST

Input: a connected undirected graph $G = (V, E)$ with weights w_e

Output: a minimum spanning tree F

- 1 **Step 0 (initialization).** Set $F := \emptyset$.
- 2 **Step 1 (stopping test).** If $|F| = n - 1$, STOP.
- 3 **Step 2 (augmentation).** Choose a cut $\delta(S)$ with $F \cap \delta(S) = \emptyset$, and add to F an edge of minimum weight in $\delta(S)$. Go to Step 1.

Theorem 2.9: correctness of the generic greedy algorithm

At every iteration of the generic greedy algorithm a cut as in Step 2 exists and is nonempty, and F is contained in some minimum spanning tree. At the end, F is a minimum spanning tree.

Proof. By induction on the iterations; $F = \emptyset$ is contained in every minimum spanning tree.

1. If $|F| < n - 1$, the graph (V, F) is a forest (a subset of a tree) with $n - |F| \geq 2$ components. Let S be one of them: no edge of F crosses $\delta(S)$, and $\delta(S) \neq \emptyset$ because G is connected.
2. By the cut property ([Theorem 2.6\(a\)](#)), the new set $F \cup \{e\}$ is still contained in some minimum spanning tree.

3. When $|F| = n - 1$, F is contained in a minimum spanning tree T^* with $n - 1$ edges, so $F = T^*$. $\square$

- The added edge $e = \{u, v\}$ has $u \in S$ and $v \notin S$, and no edge of F crosses $\delta(S)$: u and v are in different components of (V, F) , so e does not close a cycle.
- The algorithms of Kruskal and of Prim are two ways of choosing the cut in Step 2.

Adding, $n - 1$ times, the lightest edge of a cut not crossed by the current edges builds a minimum spanning tree.

2.5 Kruskal's algorithm

Algorithm

Algorithm: Kruskal's algorithm

Input: a connected undirected graph $G = (V, E)$ with weights w_e

Output: a minimum spanning tree F

- 1 **Step 0 (initialization).** Sort the edges by non-decreasing weight, $w_{e_1} \leq w_{e_2} \leq \dots \leq w_{e_m}$, and set $F := \emptyset$.
- 2 **Step 1 (scan).** For $k = 1, 2, \dots, m$: if the endpoints of e_k are in different components of (V, F) , set $F := F \cup \{e_k\}$; otherwise reject e_k .
- 3 If $|F| = n - 1$, STOP.

- An edge is **accepted** if it joins two different components of (V, F) , and **rejected** if it would close a cycle. The components of (V, F) start as the n single vertices and merge at every acceptance.
- In this module, edges of equal weight are scanned in lexicographic order of $\{u, v\}$, with $u < v$.

Theorem 2.10: correctness of Kruskal's algorithm

Kruskal's algorithm stops with $|F| = n - 1$, and F is a minimum spanning tree.

Proof. We show that every accepted edge $e_k = \{u, v\}$ is a choice of Step 2 of the generic greedy algorithm. Let S be the component of (V, F) that contains u , just before e_k is accepted: $v \notin S$, and no edge of F crosses $\delta(S)$.

1. Let $e_h \in \delta(S)$ with $h < k$. When e_h was scanned it was not accepted, since $F \cap \delta(S) = \emptyset$; so its endpoints were in the same component then, and they still are, because components only merge. This contradicts $e_h \in \delta(S)$.
2. Hence every edge of $\delta(S)$ is some e_h with $h \geq k$, and $w_{e_h} \geq w_{e_k}$: e_k has minimum weight in $\delta(S)$.

3. If the scan ended with $|F| < n - 1$, the forest (V, F) would have two components joined by an edge of G , and that edge would have been accepted when scanned, by the argument of point 1.

By [Theorem 2.9](#), F is a minimum spanning tree. □

2.5.1 Implementation and running time

- The components of (V, F) are kept in a **union–find** structure: every vertex v stores the label $comp(v)$ of its component, and every component stores the list of its vertices and its size.
- Testing whether u and v are in different components is the comparison $comp(u) \neq comp(v)$, in $O(1)$ time. Merging two components relabels the vertices of the *smaller* one (**union by size**).

Theorem 2.11: running time of Kruskal’s algorithm

With the union–find structure by labels and union by size, Kruskal’s algorithm runs in $O(m \log n)$ time.

Proof. 1. Sorting the edges takes $O(m \log m)$ time; since $m \leq n^2$, $\log m \leq 2 \log n$ and this is $O(m \log n)$.

2. The scan performs m tests, each in $O(1)$ time.

3. When a vertex is relabelled, its component merges with one at least as large, so the size of its component at least doubles. The size never exceeds n , so each vertex is relabelled at most $\log_2 n$ times: all the merges take $O(n \log n)$ time.

G is connected, so $n - 1 \leq m$, and the total is $O(m \log n + n \log n) = O(m \log n)$. □

- The running time is dominated by the sorting. With the union–find structure by trees, with union by rank and path compression, the scan takes almost linear time; the bound stays $O(m \log n)$ because of the sorting.
- The algorithm uses only comparisons between weights: any increasing transformation of the weights gives the same tree.

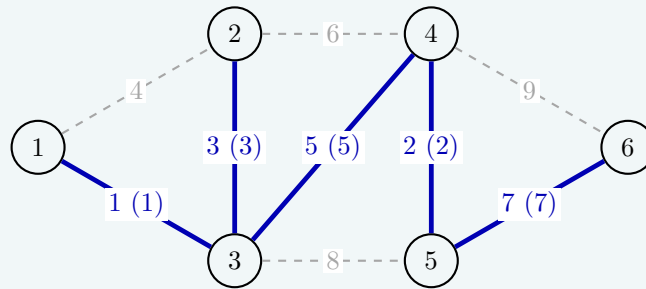
2.5.2 Kruskal's algorithm on the reference graph

Example 2.7: Kruskal's algorithm

We run Kruskal's algorithm on the weighted reference graph. The edges in order of weight are $\{1, 3\}, \{4, 5\}, \{2, 3\}, \{1, 2\}, \{3, 4\}, \{2, 4\}, \{5, 6\}, \{3, 5\}, \{4, 6\}$. Each row is one scanned edge, with the components of (V, F) after the step:

k	edge e_k	w_{e_k}	decision and components of (V, F)
1	$\{1, 3\}$	1	accept: $\{1, 3\}, \{2\}, \{4\}, \{5\}, \{6\}$
2	$\{4, 5\}$	2	accept: $\{1, 3\}, \{2\}, \{4, 5\}, \{6\}$
3	$\{2, 3\}$	3	accept: $\{1, 2, 3\}, \{4, 5\}, \{6\}$
4	$\{1, 2\}$	4	reject (1 and 2 are both in $\{1, 2, 3\}$)
5	$\{3, 4\}$	5	accept: $\{1, 2, 3, 4, 5\}, \{6\}$
6	$\{2, 4\}$	6	reject (2 and 4 are both in $\{1, 2, 3, 4, 5\}$)
7	$\{5, 6\}$	7	accept: $\{1, 2, 3, 4, 5, 6\}; F = 5$, STOP

The algorithm stops after 7 of the 9 edges, with $F = \{\{1, 3\}, \{4, 5\}, \{2, 3\}, \{3, 4\}, \{5, 6\}\}$ of weight 18: the tree $\tilde{F}$ of [Example 2.2](#). The figure shows, on each tree edge, its weight and in brackets the step k of its acceptance.



- The rejected edges $\{1, 2\}$ and $\{2, 4\}$ are the heaviest edges of the cycles 1, 2, 3 and 2, 3, 4, as the cycle property predicts; the edges $\{3, 5\}$ and $\{4, 6\}$ are never scanned.

Kruskal's algorithm scans the edges by non-decreasing weight and accepts those that join two components; with union-find it runs in $O(m \log n)$ time, dominated by the sorting.

2.6 Prim's algorithm

- Prim's algorithm grows a single tree from a start vertex s : the set S of the vertices of the tree defines the cut $\delta(S)$ of Step 2, and the lightest edge of the cut is found through a label $d(v)$ for every vertex $v \notin S$.

Algorithm

Algorithm: Prim's algorithm

Input: a connected undirected graph $G = (V, E)$ with weights w_e , given by its adjacency lists; a start vertex $s \in V$

Output: a minimum spanning tree F

- 1 **Step 0 (initialization).** Set $S := \{s\}$ and $F := \emptyset$. For every $v \neq s$: if $\{s, v\} \in E$, set $d(v) := w_{sv}$ and $pred(v) := s$; otherwise set $d(v) := +\infty$.
- 2 **Step 1 (selection).** If $S = V$, STOP. Otherwise select $v \notin S$ with minimum $d(v)$, and set $S := S \cup \{v\}$, $F := F \cup \{\{pred(v), v\}\}$.
- 3 **Step 2 (update).** For every $u \notin S$ adjacent to v with $w_{vu} < d(u)$, set $d(u) := w_{vu}$ and $pred(u) := v$. Go to Step 1.

- In this module the start vertex is $s = 1$, and ties in Step 1 are broken by the smallest index.

Lemma 2.12: meaning of the labels

At every execution of Step 1 of Prim's algorithm, for every $v \notin S$,

$$d(v) = \min \{ w_{uv} : u \in S, \{u, v\} \in E \}$$

($+\infty$ if v has no neighbour in S), and when $d(v) < +\infty$ the minimum is attained by $u = pred(v)$.

Proof. By induction. At the first execution $S = \{s\}$, and Step 0 sets exactly $d(v) = w_{sv}$, or $+\infty$. Assume the claim before the insertion of a vertex v into S . For $u \notin S \cup \{v\}$, the new set $S \cup \{v\}$ adds the candidate w_{vu} when $\{v, u\} \in E$, and Step 2 replaces $d(u)$ and $pred(u)$ exactly when this candidate is smaller. The other labels and predecessors are unchanged. $\square$

Theorem 2.13: correctness of Prim's algorithm

Prim's algorithm stops with $|F| = n - 1$, and F is a minimum spanning tree.

Proof. All the edges of F have both endpoints in S , so $F \cap \delta(S) = \emptyset$.

1. By Lemma 2.12, $\min_{v \notin S} d(v)$ is the minimum weight of an edge of $\delta(S)$, attained by $\{pred(v), v\}$ for the selected v ; it is finite because G is connected.
2. Hence every iteration is a choice of Step 2 of the generic greedy algorithm, with

the cut $\delta(S)$.

3. Each iteration adds one vertex to S and one edge to F , so the algorithm stops after $n - 1$ iterations with $|F| = n - 1$; [Theorem 2.9](#) concludes. $\square$

Theorem 2.14: running time of Prim's algorithm

With the labels stored in an array, Prim's algorithm runs in $O(n^2)$ time.

Proof. There are $n - 1$ selections, and each scans the labels of at most n vertices: $O(n^2)$ in total. The update after the insertion of v scans the adjacency list of v in $O(\deg(v))$ time, and $\sum_{v \in V} \deg(v) = 2m$. The total is $O(n^2 + m) = O(n^2)$, since $m \leq n^2$. $\square$

- On dense graphs, with m of the order of n^2 , the bound $O(n^2)$ is linear in the size of the graph.
- With the labels in a binary heap, each selection and each update take $O(\log n)$ time, for $O(m \log n)$ in total: better on sparse graphs.

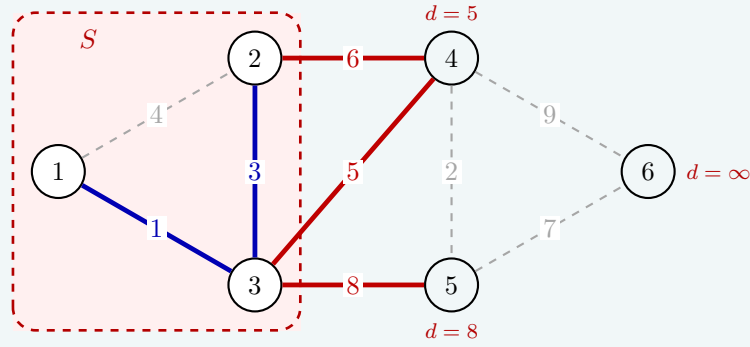
2.6.1 Prim's algorithm on the reference graph

Example 2.8: Prim's algorithm from vertex 1

We run Prim's algorithm on the weighted reference graph from $s = 1$. Each row shows the selected vertex, the edge added to F and the labels $d(v)$ of the vertices outside S after the update, with $\text{pred}(v)$ in brackets; a dash marks the vertices already in S .

iteration	selected	added edge	$d(2)$	$d(3)$	$d(4)$	$d(5)$	$d(6)$
0	1	–	4 (1)	1 (1)	∞	∞	∞
1	3	$\{1, 3\}$	3 (3)	–	5 (3)	8 (3)	∞
2	2	$\{2, 3\}$	–	–	5 (3)	8 (3)	∞
3	4	$\{3, 4\}$	–	–	–	2 (4)	9 (4)
4	5	$\{4, 5\}$	–	–	–	–	7 (5)
5	6	$\{5, 6\}$	–	–	–	–	–

At iteration 1 the label of 2 drops from 4 to 3, because $w_{32} = 3 < 4$; at iteration 3 the label of 5 drops from 8 to 2. The figure shows the state after iteration 2: the cut $\delta(\{1, 2, 3\})$ is crossed by $\{2, 4\}$, $\{3, 4\}$, $\{3, 5\}$, and the next edge is $\{3, 4\}$, with $d(4) = 5$.



The final tree is $\{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}$, of weight 18: the same tree as Kruskal's algorithm, with the edges added in a different order.

- Kruskal's algorithm adds $\{4, 5\}$ second, while Prim's algorithm adds it fourth: Kruskal grows a forest, Prim a single tree containing s .

Prim's algorithm grows one tree from s and adds the lightest edge leaving it; the labels $d(v)$ give that edge in $O(n)$ time, for $O(n^2)$ in total.

2.7 Uniqueness of the minimum spanning tree

Theorem 2.15: distinct weights

If the weights w_e , $e \in E$, are pairwise distinct, the connected graph $G = (V, E)$ has exactly one minimum spanning tree.

Proof. Assume that $T_1 \neq T_2$ are minimum spanning trees. They have the same number of edges, so the symmetric difference $(T_1 \setminus T_2) \cup (T_2 \setminus T_1)$ is nonempty; let e be its lightest edge, and say $e \in T_1 \setminus T_2$.

1. The cycle $C(T_2, e)$ is not contained in the acyclic T_1 , so it has an edge $f \notin T_1$; $f \neq e$, and $f \in T_2$, so $f \in T_2 \setminus T_1$.
2. f is in the symmetric difference and $f \neq e$, so $w_f > w_e$, the weights being distinct.
3. $T_2 \cup \{e\} \setminus \{f\}$ is a spanning tree (Lemma 2.5) of weight $w(T_2) + w_e - w_f < w(T_2)$: a contradiction. $\square$

- With distinct weights, Kruskal's algorithm, Prim's algorithm from any start vertex and every execution of the generic greedy algorithm return the same tree.
- On the reference graph, Prim's algorithm from $s = 6$ adds $\{5, 6\}, \{4, 5\}, \{3, 4\}, \{1, 3\}, \{2, 3\}$: once more the tree $\tilde{F}$.

2.7.1 Weights with ties

Proposition 2.16: edges in every or in no minimum spanning tree

Let T be a minimum spanning tree of $G = (V, E)$.

- (a) An edge $e \in T$ belongs to every minimum spanning tree if and only if $w_e < w_f$ for every $f \neq e$ in the fundamental cut of e .
- (b) An edge $e \notin T$ belongs to no minimum spanning tree if and only if $w_e > w_f$ for every $f \neq e$ in $C(T, e)$.

In particular, T is the unique minimum spanning tree if and only if every edge outside T is strictly heavier than the other edges of its fundamental cycle.

Proof. (a) If the inequalities hold, e is in every minimum spanning tree by the cut property ([Theorem 2.6\(b\)](#)). Otherwise some $f \neq e$ in the fundamental cut has $w_f \leq w_e$, and $w_f \geq w_e$ by the cut condition ([Theorem 2.8](#)): so $w_f = w_e$, and $T \cup \{f\} \setminus \{e\}$ is a spanning tree ([Lemma 2.5\(2\)](#)) of the same weight, without e .

- (b) If the inequalities hold, e is in no minimum spanning tree by the cycle property ([Theorem 2.7\(b\)](#)). Otherwise some $f \neq e$ in $C(T, e)$ has $w_f = w_e$, by the cycle condition, and $T \cup \{e\} \setminus \{f\}$ is a minimum spanning tree containing e .

For the last claim: T is unique if and only if no edge outside T belongs to a minimum spanning tree, that is, (b) holds for every $e \notin T$. $\square$

Example 2.9: ties on the reference graph

- Let $w_{12} = 3$ instead of 4, so that $w_{12} = w_{23} = 3$. The fundamental cycle of $\{1, 2\}$ in $\tilde{F}$ is 1, 3, 2, 1, with $w_{23} = w_{12}$: by [Proposition 2.16\(b\)](#), the tree $\{1, 2\}, \{1, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}$ is also optimal, with weight 18.
- Enumeration confirms that these are the only two minimum spanning trees. Kruskal's algorithm, scanning $\{1, 2\}$ before $\{2, 3\}$, returns the one with $\{1, 2\}$.
- Ties do not always create several optimal trees: with $w_{35} = w_{46} = 9$ the tree $\tilde{F}$ is still the unique minimum spanning tree, because each of the four edges outside it remains strictly the heaviest of its cycle.

With distinct weights the minimum spanning tree is unique. With ties, a tree is the unique optimum if and only if every edge outside it is strictly the heaviest of its fundamental cycle.

2.8 Typical mistakes

Warning

- **Using the cut property with a cut crossed by F .** The lightest edge of $\delta(S)$ is safe only if no edge already chosen crosses $\delta(S)$.
- **Accepting an edge in Kruskal's algorithm because it touches a new vertex.** The test is whether the two endpoints are in different components: $\{3, 4\}$ is accepted although both 3 and 4 are already covered.
- **Updating the labels in Prim's algorithm with path lengths.** The new label is the weight w_{vu} of a single edge, not $d(v) + w_{vu}$: that is Dijkstra's algorithm for shortest paths.
- **Forgetting that the labels can only decrease.** In Prim's algorithm $d(u)$ changes only when $w_{vu} < d(u)$.
- **Claiming uniqueness from the algorithm.** Kruskal and Prim return one tree; with ties other optimal trees may exist, and they are found through the equalities in the fundamental cycles.
- **Applying the algorithms to directed graphs.** A minimum spanning tree is defined on undirected graphs; in a directed graph the orientation of the arcs is ignored.

2.9 In summary

Summary

- A minimum spanning tree is a spanning tree of minimum total weight of a connected weighted graph; it has $n - 1$ edges.
- Cut property: the lightest edge of a cut not crossed by F extends F within some minimum spanning tree. Cycle property: the heaviest edge of a cycle can be left out, and if it is strictly the heaviest it is in no optimum.
- A spanning tree is optimal if and only if every edge outside it is the heaviest of its fundamental cycle, or equivalently every tree edge is the lightest of its fundamental cut.
- Kruskal's algorithm sorts the edges and accepts those joining two components, in $O(m \log n)$ time; Prim's algorithm grows one tree with the labels $d(v)$, in $O(n^2)$ time.
- With distinct weights the minimum spanning tree is unique.

The weighted reference graph at a glance.

vertices, edges, spanning trees	$n = 6, m = 9, 55$
minimum spanning tree	$\{1, 3\}, \{2, 3\}, \{3, 4\}, \{4, 5\}, \{5, 6\}$, weight 18 (unique)
Kruskal	accepts $\{1, 3\}, \{4, 5\}, \{2, 3\}, \{3, 4\}, \{5, 6\}$; rejects $\{1, 2\}, \{2, 4\}$
Prim from 1	selects 3, 2, 4, 5, 6

Shortest paths

Prerequisites

- From [Chapter 1 «Graphs»](#): directed walks, paths and cycles, the node–arc incidence matrix, trees and the reference graph.
- From the module Linear Programming, [Chapter 9 «Duality theory»](#): the dual of an LP model and the weak duality theorem.

Objectives

At the end of this chapter you will be able to:

- state the s – t and the single-source shortest path problems, and recognize when a negative cycle makes them ill-posed;
- use the optimality of subpaths, Bellman’s equations and the shortest path tree;
- write the LP model of a shortest path as a unit flow, write its dual with the potentials π_i , and use feasible potentials as a certificate of optimality;
- run the algorithm for acyclic graphs, the Bellman–Ford algorithm (with the detection of a negative cycle) and Dijkstra’s algorithm, prove their correctness and give their running time.

3.1 Shortest path problems

Definition 3.1: shortest path problems

Let $G = (N, A)$ be a directed graph with a **length** $d_{ij} \in \mathbb{R}$ for every arc $(i, j) \in A$, of any sign.

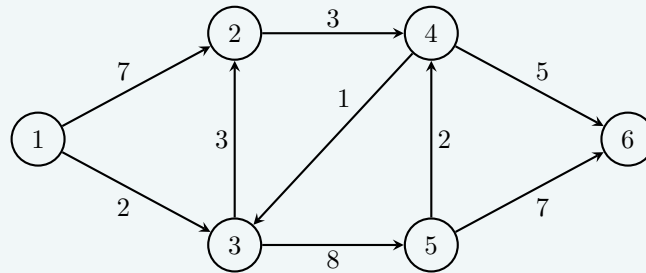
- The length of a directed walk $W = i_0, i_1, \dots, i_k$ is the sum of the lengths of its arcs, $d(W) = \sum_{t=1}^k d_{i_{t-1}i_t}$.
- Given two nodes s and t , the s - t **shortest path problem** asks for a directed path from s to t of minimum length.
- Given a node s , the **source**, the **single-source shortest path problem** asks for a shortest directed path from s to every node $j \in N$.

The minimum length of a directed path from s to j is the **distance** $\tilde{d}(j)$ of j ; we set $\tilde{d}(j) = +\infty$ if j is not reachable from s .

- In this chapter we write $n = |N|$, $m = |A|$, and “path” and “cycle” always mean directed path and directed cycle.
- An algorithm for the single-source problem also solves the s - t problem, for every t at once.

Example 3.1: the reference graph with lengths

We give the reference graph of [Chapter 1 «Graphs»](#) the lengths d_{ij} written on the arcs, all non-negative; the source is $s = 1$.



Three of the paths from 1 to 6 and their lengths are:

$$d(1, 2, 4, 6) = 7 + 3 + 5 = 15, \quad d(1, 3, 2, 4, 6) = 2 + 3 + 3 + 5 = 13, \quad d(1, 3, 5, 6) = 2 + 8 + 7 = 17.$$

The distances from node 1, computed in [Section 3.6](#), are

$$\tilde{d} = (0, 5, 2, 8, 10, 13),$$

so 1, 3, 2, 4, 6 is a shortest path from 1 to 6.

3.1.1 Negative cycles

Definition 3.2: negative cycle

A **negative cycle** is a directed cycle C with $d(C) < 0$. In the rest of the chapter we refer to two assumptions:

- (A1) every node of G is reachable from s ;
- (A2) no negative cycle of G is reachable from s .

Lemma 3.3: walks and paths

Under (A2), for every directed walk W from s to j there is a directed path P from s to j , with arcs of W , such that $d(P) \leq d(W)$.

Proof. By induction on the number of arcs of W . If W has distinct nodes, $P = W$.

1. Otherwise W repeats a node: $W = s, \dots, i_a, \dots, i_b, \dots, j$ with $i_a = i_b$, $a < b$, and $b - a$ minimum. The piece $C = i_a, \dots, i_b$ is a cycle, reachable from s along W .
2. By (A2), $d(C) \geq 0$. Deleting C from W gives a walk W' from s to j with fewer arcs and $d(W') = d(W) - d(C) \leq d(W)$.
3. By induction W' contains a path P with $d(P) \leq d(W') \leq d(W)$. □

- Under (A2) the minimum length of a walk from s to j equals the distance $\tilde{d}(j)$: the algorithms of this chapter compute lengths of walks, and the lemma turns them into paths.
- With a negative cycle C reachable from s and from which j is reachable, going around C again and again gives walks from s to j of arbitrarily negative length. The algorithms of this chapter then detect C instead of computing the distances.

Example 3.2: a negative cycle

The reference graph has two cycles, 2, 4, 3, 2 and 3, 5, 4, 3. With the lengths of [Example 3.1](#) their lengths are 7 and 11. If the length of the arc (5, 4) drops from 2 to $d''_{54} = -10$, the cycle 3, 5, 4, 3 has length

$$8 - 10 + 1 = -1,$$

and the walks 1, 3, 5, 6, 1, 3, 5, 4, 3, 5, 6, 1, 3, 5, 4, 3, 5, 4, 3, 5, 6, ... from 1 to 6 have lengths 17, 16, 15, ...: no walk is shortest. We use these lengths d'' again in [Section 3.5](#).

3.1.2 Optimality of subpaths

Lemma 3.4: optimality of subpaths

Under (A2), let P be a shortest path from s to j , and let i be a node of P . Then the subpath P_{si} of P from s to i is a shortest path from s to i .

Proof. Write P as P_{si} followed by the subpath P_{ij} from i to j , so $d(P) = d(P_{si}) + d(P_{ij})$. Assume by contradiction that a path Q from s to i has $d(Q) < d(P_{si})$.

1. Q followed by P_{ij} is a walk from s to j of length $d(Q) + d(P_{ij}) < d(P)$.
2. By Lemma 3.3 it contains a path from s to j shorter than P : a contradiction. $\square$

Theorem 3.5: Bellman's equations

Under (A1) and (A2), the distances satisfy

$$\tilde{d}(s) = 0, \quad \tilde{d}(j) = \min_{(i,j) \in \delta^-(j)} \{ \tilde{d}(i) + d_{ij} \}, \quad \forall j \in N \setminus \{s\},$$

and $\tilde{d}(j) \leq \tilde{d}(i) + d_{ij}$ for every arc $(i, j) \in A$.

Proof. All distances are finite by (A1), and $\tilde{d}(s) = 0$: the path s has length 0, and a path from s to s with arcs would be a cycle, of length ≥ 0 by (A2).

- *Inequality.* A shortest path from s to i followed by the arc (i, j) is a walk of length $\tilde{d}(i) + d_{ij}$; by Lemma 3.3, $\tilde{d}(j) \leq \tilde{d}(i) + d_{ij}$.
- *Equation.* For $j \neq s$, let P be a shortest path from s to j and (i, j) its last arc. By Lemma 3.4, $\tilde{d}(j) = d(P) = \tilde{d}(i) + d_{ij}$, so the minimum is attained. $\square$

Proposition 3.6: shortest path tree

Under (A1) and (A2), there is a spanning tree T of G , the **shortest path tree**, such that for every $j \in N$ the path from s to j in T is directed and is a shortest path.

Proof. For every j , let $k(j)$ be the minimum number of arcs of a shortest path from s to j ; $k(s) = 0$.

1. For $j \neq s$, choose a shortest path P from s to j with $k(j)$ arcs, and let $\text{pred}(j)$ be the node before j on P .
2. The subpath of P from s to $\text{pred}(j)$ is shortest (Lemma 3.4) with $k(j) - 1$ arcs, so $k(\text{pred}(j)) \leq k(j) - 1$, and $\tilde{d}(j) = \tilde{d}(\text{pred}(j)) + d_{\text{pred}(j)j}$.
3. Moving from j to $\text{pred}(j)$, the number k strictly decreases, so the sequence ends in s . The arcs $(\text{pred}(j), j)$ are $n - 1$ and connect every node to s : they form a spanning tree (Chapter 1, Theorem «characterizations of a tree»).

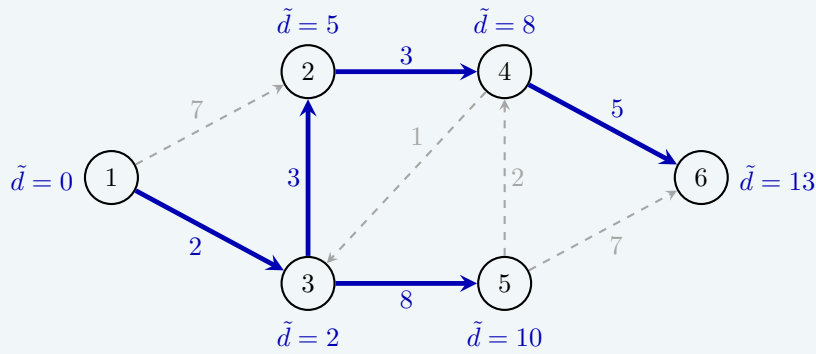
4. Along the tree path $s = i_0, i_1, \dots, i_r = j$ every arc satisfies $\tilde{d}(i_t) = \tilde{d}(i_{t-1}) + d_{i_{t-1}i_t}$; adding up, its length is $\tilde{d}(j)$. $\square$

Example 3.3: Bellman's equations on the reference graph

With the distances $\tilde{d} = (0, 5, 2, 8, 10, 13)$ of Example 3.1:

- node 4 has the entering arcs (2, 4) and (5, 4), and $\tilde{d}(4) = \min\{5 + 3, 10 + 2\} = 8$;
- node 6 has the entering arcs (4, 6) and (5, 6), and $\tilde{d}(6) = \min\{8 + 5, 10 + 7\} = 13$.

The arcs attaining the minimum form the shortest path tree (1, 3), (3, 2), (2, 4), (3, 5), (4, 6), in blue; next to each node its distance.



Without negative cycles reachable from the source, every subpath of a shortest path is shortest, the distances satisfy Bellman's equations, and one spanning tree contains a shortest path to every node.

3.2 Optimality conditions and potentials

Definition 3.7: potentials and reduced costs

A vector $\pi \in \mathbb{R}^n$ gives a **potential** π_i to every node $i \in N$. The **reduced cost** of the arc (i, j) with respect to π is

$$\bar{c}_{ij} = d_{ij} - \pi_i + \pi_j.$$

The potentials are **feasible** if $\bar{c}_{ij} \geq 0$ for every arc $(i, j) \in A$, and an arc with $\bar{c}_{ij} = 0$ is **tight**.

- Given **labels** $d(j)$, lengths of walks from s to j , we use the potentials $\pi_i = -d(i)$. Then $\bar{c}_{ij} = d_{ij} + d(i) - d(j)$, and the potentials are feasible if and only if $d(j) \leq d(i) + d_{ij}$ for every arc.
- Along a walk $W = i_0, i_1, \dots, i_k$ the potentials telescope:

$$d(W) = \sum_{t=1}^k \bar{c}_{i_{t-1}i_t} + \pi_{i_0} - \pi_{i_k}.$$

Theorem 3.8: potentials as optimality certificate

Let π be feasible potentials, and let P be a path from s to t whose arcs are all tight. Then P is a shortest path from s to t , and $d(P) = \pi_s - \pi_t$. Moreover, G has no negative cycle.

Proof. We use the telescoping formula above.

1. The arcs of P have $\bar{c}_{ij} = 0$, so $d(P) = \pi_s - \pi_t$.
2. For every path Q from s to t , $d(Q) = \sum_{(i,j) \in Q} \bar{c}_{ij} + \pi_s - \pi_t \geq \pi_s - \pi_t = d(P)$.
3. For every cycle C the first and the last node coincide, so $d(C) = \sum_{(i,j) \in C} \bar{c}_{ij} \geq 0$. $\square$

Corollary 3.9: optimality conditions for labels

Let the labels $d(j)$, $j \in N$, be finite with $d(s) = 0$, and assume that

- (i) $d(j) \leq d(i) + d_{ij}$ for every arc $(i, j) \in A$;
- (ii) for every $j \neq s$ there is a path from s to j of length $d(j)$.

Then $d(j) = \tilde{d}(j)$ for every $j \in N$.

Proof. By (i) the potentials $\pi_i = -d(i)$ are feasible, so every path Q from s to j has $d(Q) \geq \pi_s - \pi_j = d(j)$ by the argument of [Theorem 3.8](#). By (ii) the value $d(j)$ is attained by a path, so $d(j) = \tilde{d}(j)$. $\square$

Example 3.4: a certificate on the reference graph

With the distances of [Example 3.1](#) the potentials are $\pi = -\tilde{d} = (0, -5, -2, -8, -10, -13)$, and the reduced costs $\bar{c}_{ij} = d_{ij} - \pi_i + \pi_j$ are:

arc	(1, 2)	(1, 3)	(2, 4)	(3, 2)	(3, 5)	(4, 3)	(4, 6)	(5, 4)	(5, 6)
d_{ij}	7	2	3	3	8	1	5	2	7
$\bar{c}_{ij}$	2	0	0	0	0	7	0	4	4

- All reduced costs are non-negative, and the tight arcs are the arcs of the shortest path tree. The path 1, 3, 2, 4, 6 is tight, so it is shortest, with length $\pi_1 - \pi_6 = 13$.
- The tree of the breadth-first search, (1, 2), (1, 3), (2, 4), (3, 5), (4, 6), gives the labels (0, 7, 2, 10, 10, 15). With their potentials, $\bar{c}_{32} = 3 - (-2) + (-7) = -2 < 0$: the potentials are not feasible, and entering node 2 through (3, 2) shortens its path by 2.

Feasible potentials ($\bar{c}_{ij} = d_{ij} - \pi_i + \pi_j \geq 0$ on every arc) and a path made of tight arcs prove that the path is shortest. With labels, $\pi_i = -d(i)$, and feasibility reads $d(j) \leq d(i) + d_{ij}$.

3.3 The linear programming formulation

- A path from s to t sends one unit of flow from s to t . We use a variable $x_{ij} \geq 0$ for every arc, the flow on the arc, and the **balances**

$$b_s = 1, \quad b_t = -1, \quad b_i = 0, \quad \forall i \in N \setminus \{s, t\}.$$

Using these variables, an LP model for the s - t shortest path problem reads as follows:

$$\min \sum_{(i,j) \in A} d_{ij} x_{ij} \quad (3.1a)$$

$$\text{subject to} \quad \sum_{(i,j) \in \delta^+(i)} x_{ij} - \sum_{(j,i) \in \delta^-(i)} x_{ji} = b_i, \quad \forall i \in N, \quad (3.1b)$$

$$x_{ij} \geq 0, \quad \forall (i,j) \in A. \quad (3.1c)$$

This LP model is referred to as the $\text{SP}(s, t)$ model.

- Description of the objective function and constraints of the $\text{SP}(s, t)$ model:
 - the objective function (3.1a) minimizes the total length of the flow;
 - constraints (3.1b) are the flow conservation constraints: one unit leaves s , one unit enters t , and every other node sends out what it receives;
 - finally, constraints (3.1c) define the variables of the LP model.
- In matrix form the model is $\min\{\mathbf{d}'\mathbf{x} : \mathbf{B}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$, with the node–arc incidence matrix $\mathbf{B}$ of [Chapter 1](#): +1 in the tail and –1 in the head of every arc.
- The **incidence vector** $\mathbf{x}^P$ of a path P from s to t , with $x_{ij}^P = 1$ on the arcs of P and 0 elsewhere, is feasible, and its value is $d(P)$.

3.3.1 The dual model and the potentials

- The constraint of node i is an equation, so its dual variable π_i is free. The column of x_{ij} in $\mathbf{B}$ has $+1$ in row i and -1 in row j , so the dual constraint of $x_{ij} \geq 0$ is $\pi_i - \pi_j \leq d_{ij}$ (Linear Programming, Chapter 9 «Duality theory»).

Using a free dual variable π_i for each constraint (3.1b), the dual model of the $\text{SP}(s, t)$ model is:

$$\max \quad \pi_s - \pi_t \quad (3.2a)$$

$$\text{subject to} \quad \pi_i - \pi_j \leq d_{ij}, \quad \forall (i, j) \in A, \quad (3.2b)$$

$$\pi_i \in \mathbb{R}, \quad \forall i \in N. \quad (3.2c)$$

This LP model is referred to as the $D(\text{SP}(s, t))$ model.

- Constraint (3.2b) is $\bar{c}_{ij} = d_{ij} - \pi_i + \pi_j \geq 0$: the feasible solutions of the dual are the feasible potentials of Definition 3.7, and $\bar{c}_{ij}$ is the reduced cost of x_{ij} in the LP sense.

Theorem 3.10: LP and shortest paths

Under (A1) and (A2), the $\text{SP}(s, t)$ model has an optimal solution and $z(\text{SP}(s, t)) = \tilde{d}(t)$. The incidence vector of a shortest path from s to t is optimal for $\text{SP}(s, t)$, and the potentials $\pi_i = -\tilde{d}(i)$ are optimal for $D(\text{SP}(s, t))$.

Proof. Let P be a shortest path from s to t , which exists by (A1).

1. $\mathbf{x}^P$ is feasible for $\text{SP}(s, t)$, with value $d(P) = \tilde{d}(t)$.
2. By Theorem 3.5, $\tilde{d}(j) \leq \tilde{d}(i) + d_{ij}$ for every arc, so $\pi_i = -\tilde{d}(i)$ satisfies $\pi_i - \pi_j \leq d_{ij}$; its value is $\pi_s - \pi_t = 0 + \tilde{d}(t)$.
3. A feasible primal and a feasible dual solution with the same value are both optimal, by weak duality. $\square$

Proposition 3.11: a negative cycle makes the model unbounded

If $\text{SP}(s, t)$ is feasible and G has a negative cycle C , reachable from s or not, then $\text{SP}(s, t)$ is unbounded.

Proof. Let $\bar{\mathbf{x}}$ be feasible and let $\mathbf{x}^C$ be the incidence vector of C . The columns of the arcs of C add up to zero (Chapter 1, Lemma «columns of a cycle»), so $\mathbf{B}\mathbf{x}^C = \mathbf{0}$. For every $\lambda \geq 0$ the vector $\bar{\mathbf{x}} + \lambda\mathbf{x}^C$ is feasible, with value $\mathbf{d}'\bar{\mathbf{x}} + \lambda d(C)$, which tends to $-\infty$. $\square$

The sign of the potentials

With the incidence matrix of the course (+1 in the tail) the dual constraint is $\pi_i - \pi_j \leq d_{ij}$, and the optimal potentials are $\pi_i = -\tilde{d}(i)$, up to an additive constant. The form $\pi_j - \pi_i \leq d_{ij}$, with $\pi_i = \tilde{d}(i)$, belongs to the opposite sign convention (-1 in the tail).

3.3.2 The single-source model

- To find a shortest path from s to every node, s sends one unit to each of the other $n - 1$ nodes:

$$b_s = n - 1, \quad b_i = -1, \quad \forall i \in N \setminus \{s\}.$$

With these balances the model (3.1) is called SP(s); its dual maximizes $(n-1)\pi_s - \sum_{i \neq s} \pi_i$ under the same constraints (3.2b).

Proposition 3.12: the single-source model

Under (A1) and (A2), let T be a shortest path tree and, for every $j \neq s$, let T_j be the set of the nodes whose tree path passes through j (j included). The flow with $x_{ij} = |T_j|$ on the arcs (i, j) of T and 0 elsewhere is optimal for SP(s), the potentials $\pi_i = -\tilde{d}(i)$ are optimal for its dual, and

$$z(\text{SP}(s)) = \sum_{j \in N} \tilde{d}(j).$$

- Proof.*
1. *Feasibility.* At a node $j \neq s$ the flow entering on the tree arc is $|T_j|$, and the flow leaving is $\sum |T_h|$ over the tree arcs (j, h) , that is $|T_j| - 1$: the balance is -1 . At s the leaving flow covers all the other nodes: $n - 1$.
 2. *Value.* The flow is the sum of the incidence vectors of the $n - 1$ tree paths, since the arc (i, j) lies on the tree paths of the $|T_j|$ nodes of T_j . Its value is $\sum_j \tilde{d}(j)$.
 3. *Dual.* $\pi_i = -\tilde{d}(i)$ is feasible as in [Theorem 3.10](#), with value $(n-1) \cdot 0 + \sum_{i \neq s} \tilde{d}(i)$. By weak duality both solutions are optimal. $\square$

- In [Chapter 4 «Network flow models»](#) the vertices of these polyhedra are shown to be integral, by the total unimodularity of B .

Example 3.5: the LP models on the reference graph

We solve the models with the lengths of [Example 3.1](#), $s = 1$ and $t = 6$ (values checked with Gurobi).

- SP(1,6): $z(\text{SP}(1,6)) = 13$, with $x_{13} = x_{32} = x_{24} = x_{46} = 1$ and all other variables 0: the path 1, 3, 2, 4, 6. The dual value is $\pi_1 - \pi_6 = 13$.
- SP(1): $z(\text{SP}(1)) = 0 + 5 + 2 + 8 + 10 + 13 = 38$, with the flows on the shortest path tree

$$x_{13} = 5, \quad x_{32} = 3, \quad x_{24} = 2, \quad x_{46} = 1, \quad x_{35} = 1.$$

For example $T_3 = \{2, 3, 4, 5, 6\}$, so $x_{13} = 5$; the cost is $2 \cdot 5 + 3 \cdot 3 + 3 \cdot 2 + 5 + 8 = 38$. The optimal duals, normalized with $\pi_1 = 0$, are $\pi = (0, -5, -2, -8, -10, -13)$.

- With the lengths d'' of [Example 3.2](#) the model $\text{SP}(1, 6)$ is unbounded: each unit of flow around the cycle $3, 5, 4, 3$ lowers the value by 1.

A shortest path is a unit flow from s to t : $\min \mathbf{d}'\mathbf{x}$ with $\mathbf{B}\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$. Its dual has the constraints $\pi_i - \pi_j \leq d_{ij}$, and $\pi_i = -\tilde{d}(i)$ is optimal; a negative cycle anywhere in G makes the model unbounded.

3.4 Shortest paths in acyclic graphs

Definition 3.13: acyclic graph and topological order

A directed graph without directed cycles is a **directed acyclic graph** (DAG). A **topological order** of $G = (N, A)$ is an ordering $v_1, v_2, \dots, v_n$ of its nodes such that every arc goes forward:

$$(v_a, v_b) \in A \implies a < b.$$

Proposition 3.14: topological orders

A directed graph has a topological order if and only if it is acyclic.

Proof. If G has a topological order, a cycle $i_0, i_1, \dots, i_k = i_0$ would go forward on every arc and come back to its first node: impossible. Conversely, let G be acyclic; we argue by induction on n .

1. G has a node with no entering arc: otherwise, walking backwards along entering arcs from any node, after n steps a node repeats, and the walk contains a cycle.
2. Put such a node v_1 first. The graph without v_1 is acyclic, and by induction it has a topological order $v_2, \dots, v_n$; no arc enters v_1 , so $v_1, v_2, \dots, v_n$ is a topological order of G . $\square$

Algorithm

Algorithm: Shortest paths in an acyclic graph

Input: an acyclic graph $G = (N, A)$ with lengths d_{ij} of any sign, a source $s \in N$

Output: the distances $d(j) = \tilde{d}(j)$ and the predecessors $pred(j)$

- 1 **Step 0 (order).** Find a topological order: repeatedly remove a node without entering arcs (the smallest index, if more than one).
- 2 Set $d(s) := 0$ and $d(j) := +\infty$ for $j \neq s$.
- 3 **Step 1 (labels).** For each node i in topological order with $d(i) < +\infty$, and each arc $(i, j) \in \delta^+(i)$:
 - 4 if $d(i) + d_{ij} < d(j)$, set $d(j) := d(i) + d_{ij}$ and $pred(j) := i$.
 - 5 STOP.

Theorem 3.15: correctness and running time on acyclic graphs

On an acyclic graph with lengths of any sign, the algorithm «Shortest paths in an acyclic graph» ends with $d(j) = \tilde{d}(j)$ for every $j \in N$, and it runs in $O(n + m)$ time.

Proof. An acyclic graph has no cycle at all, so (A2) holds. Every finite label is the length of a walk from s , so $d(j) \geq \tilde{d}(j)$ by Lemma 3.3; labels only decrease.

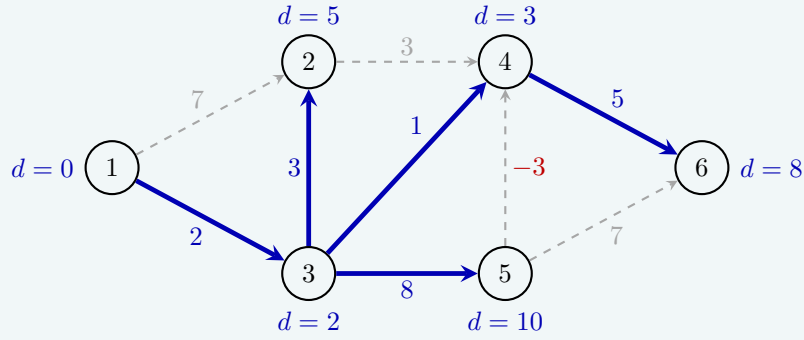
1. Let j be reachable and $s = i_0, i_1, \dots, i_k = j$ a shortest path; its nodes appear in this order in the topological order, since every arc goes forward.
2. By induction on t , after i_t is processed $d(i_t) \leq \tilde{d}(i_t)$: the path to i_{t-1} is shortest (Lemma 3.4), and processing i_{t-1} gives $d(i_t) \leq \tilde{d}(i_{t-1}) + d_{i_{t-1}i_t} = \tilde{d}(i_t)$.
3. Nodes not reachable from s keep $d(j) = +\infty = \tilde{d}(j)$.

Time. The topological order takes $O(n + m)$ time with a counter of the entering arcs of every node, decreased when a tail is removed; Step 1 scans every arc once. $\square$

- The smallest-index rule fixes one order for the examples; any node without entering arcs can be removed.
- With the lengths $-d_{ij}$ the algorithm computes **longest paths**, since an acyclic graph has no cycle to repeat. Longest paths in acyclic graphs are the core of Chapter 9 «Project management».

Example 3.6: an acyclic instance

Reversing the arc $(4, 3)$ of the reference graph into $(3, 4)$, with $d_{34} = 1$, gives an acyclic graph. We give it the lengths of Example 3.1 except $d_{54} = -3$ (red).



Step 0 gives the topological order 1, 3, 2, 5, 4, 6: after 1, only 3 has no entering arc; then 2 and 5 are free, and 2 goes first. Each row of the table processes one node and shows the labels after it:

step	node i	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$	$d(6)$
1	1	0	7	2	∞	∞	∞
2	3	0	5	2	3	10	∞
3	2	0	5	2	3	10	∞
4	5	0	5	2	3	10	17
5	4	0	5	2	3	10	8
6	6	0	5	2	3	10	8

The predecessors give the tree (1, 3), (3, 2), (3, 4), (3, 5), (4, 6) (blue). The negative arc (5, 4) is not used: $d(5) - 3 = 7 > 3 = d(4)$. With the lengths $-d_{ij}$ the longest path lengths from 1 are (0, 7, 2, 10, 10, 17).

A graph is acyclic if and only if it has a topological order. Scanning the nodes in that order, each label is final when its node is processed: shortest paths in $O(n + m)$ time, with lengths of any sign.

3.5 The Bellman–Ford algorithm

Definition 3.16: labels with at most k arcs

For $k \geq 0$ and $j \in N$, $d^k(j)$ is the minimum length of a walk from s to j with at most k arcs, and $d^k(j) = +\infty$ if there is none.

Lemma 3.17: recursion of the labels

$d^0(s) = 0$, $d^0(j) = +\infty$ for $j \neq s$, and for every $k \geq 1$

$$d^k(j) = \min \left\{ d^{k-1}(j), \min_{(i,j) \in \delta^-(j)} (d^{k-1}(i) + d_{ij}) \right\}, \quad \forall j \in N.$$

Proof. The only walk with 0 arcs is the walk s . For $k \geq 1$, a walk to j with at most k arcs either has at most $k - 1$ arcs, with minimum length $d^{k-1}(j)$, or has exactly k arcs and last arc (i, j) : it is a walk to i with $k - 1$ arcs followed by (i, j) , and its minimum length is $\min_{(i,j)} (d^{k-1}(i) + d_{ij})$. $\square$

Algorithm

Algorithm: Bellman–Ford algorithm

Input: a directed graph $G = (N, A)$ with lengths d_{ij} of any sign, a source $s \in N$

Output: the distances $\tilde{d}(j)$ and the predecessors, or a negative cycle reachable from s

- 1 **Step 0 (initialization).** Set $d^0(s) := 0$, $d^0(j) := +\infty$ for $j \neq s$, and $k := 1$.
- 2 **Step 1 (pass k).** Compute $d^k(j)$ for every $j \in N$ with the recursion of Lemma 3.17;
- 3 if $d^k(j) < d^{k-1}(j)$, set $\text{pred}(j)$ to the tail i of the arc that attains the minimum.
- 4 **Step 2 (test).** If $d^k = d^{k-1}$, STOP: $\tilde{d} = d^k$.
- 5 If $k = n$, STOP: a negative cycle is reachable from s . Otherwise set $k := k + 1$ and go to Step 1.

- Each pass depends only on the labels of the previous pass: if $d^k = d^{k-1}$, all the following passes repeat the same labels, and the algorithm can stop.
- In the examples, ties in the minimum are broken by the smallest tail i , and the predecessor changes only when the label strictly decreases.

Theorem 3.18: correctness of Bellman–Ford

If no negative cycle is reachable from s , then $d^{n-1}(j) = \tilde{d}(j)$ for every $j \in N$, and the algorithm stops at a pass $k \leq n$ with $d^k = d^{k-1} = \tilde{d}$.

Proof. A path has at most $n - 1$ arcs, so $d^{n-1}(j) \leq \tilde{d}(j)$. By Lemma 3.3 (which needs only the cycles reachable from s) every walk from s to j contains a path not longer, so $d^k(j) \geq \tilde{d}(j)$ for every k .

- Hence $d^{n-1} = d^n = \tilde{d}$.
- The labels never increase, and the algorithm stops at the first pass k with $d^k = d^{k-1}$, which is at most n ; from that pass on the labels no longer change, so $d^k = d^{n-1} = \tilde{d}$. $\square$

Theorem 3.19: detection of a negative cycle

A negative cycle is reachable from s if and only if $d^n \neq d^{n-1}$. In that case, let $d^n(j) < d^{n-1}(j)$; reading the predecessors backwards from j , one row per pass, starting from row n of the table, gives a walk from s to j with n arcs that contains a negative cycle.

Proof. If no negative cycle is reachable from s , then $d^n = d^{n-1}$ by Theorem 3.18. Conversely, let $d^n(j) < d^{n-1}(j)$.

1. Every walk to j of length $d^n(j)$ with at most n arcs has exactly n arcs, otherwise $d^{n-1}(j) \leq d^n(j)$.
2. Let i be the predecessor of j in row n , so $d^n(j) = d^{n-1}(i) + d_{ij}$. A walk realizing $d^{n-1}(i)$ followed by (i, j) realizes $d^n(j)$, so it has exactly $n - 1$ arcs: $d^{n-1}(i) < d^{n-2}(i)$, and i has a predecessor in row $n - 1$.
3. Repeating the argument down to row 1, whose predecessor is s , gives a walk W from s to j with n arcs and $d(W) = d^n(j)$.
4. W has $n + 1$ nodes, so it repeats a node and contains a cycle C . Deleting C gives a walk to j with fewer than n arcs, of length $d(W) - d(C) \geq d^{n-1}(j) > d^n(j) = d(W)$. Hence $d(C) < 0$, and C is reachable from s along W . $\square$

Proposition 3.20: the predecessor tree of Bellman–Ford

If no negative cycle is reachable from s , at the stop the arcs $(pred(j), j)$, for the nodes $j \neq s$ with finite label, form a shortest path tree of the reachable nodes.

Proof. Let $d = \tilde{d}$ be the final labels, and let $t(j)$ be the last pass in which the label of j decreased; $t(s) = 0$, since $d^k(s) = 0$ for all k . Let $p = pred(j)$.

1. In pass $t(j)$, $d(j) = d^{t(j)-1}(p) + d_{pj} \geq d(p) + d_{pj}$, since labels never increase.
2. By Theorem 3.5, $d(j) \leq d(p) + d_{pj}$. Hence $d(j) = d(p) + d_{pj}$ and $d(p) = d^{t(j)-1}(p)$: the label of p did not change after pass $t(j) - 1$, so $t(p) < t(j)$.
3. Moving to the predecessor t strictly decreases, so the sequence ends in s , along tight arcs: the tree path from s to j has length $d(j)$. $\square$

Theorem 3.21: running time of Bellman–Ford

The Bellman–Ford algorithm runs in $O(nm)$ time.

Proof. A pass computes $d^k(j)$ for every j by scanning the backward star $\delta^-(j)$, so it scans every arc once: $O(n + m)$ time. There are at most n passes, and the total is $O(n(n + m))$, which is $O(nm)$ when $m \geq n - 1$, as under (A1). $\square$

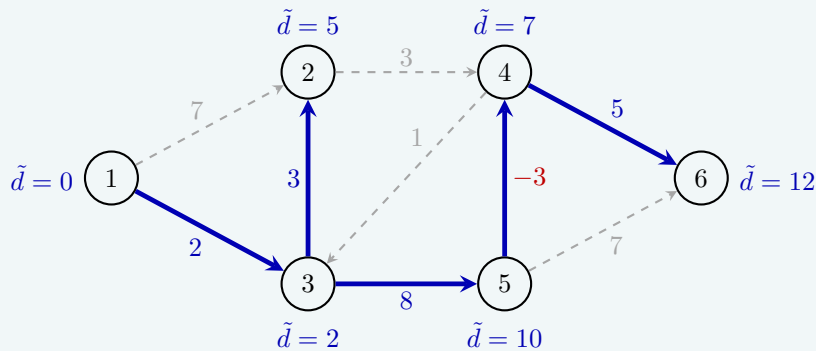
3.5.1 Bellman–Ford on the reference graph

Example 3.7: Bellman–Ford with a negative arc

We use the lengths d' of Example 3.1 with $d'_{54} = -3$ in place of 2. The two cycles have lengths 7 and $8 - 3 + 1 = 6$, so there is no negative cycle. Each row is a pass k ; next to a label that decreased, the new predecessor in brackets:

k	$d^k(1)$	$d^k(2)$	$d^k(3)$	$d^k(4)$	$d^k(5)$	$d^k(6)$
0	0	∞	∞	∞	∞	∞
1	0	7 (1)	2 (1)	∞	∞	∞
2	0	5 (3)	2	10 (2)	10 (3)	∞
3	0	5	2	7 (5)	10	15 (4)
4	0	5	2	7	10	12 (4)
5	0	5	2	7	10	12

- In pass 3, $d^3(4) = \min\{10, d^2(2) + 3, d^2(5) - 3\} = \min\{10, 8, 7\} = 7$, with predecessor 5.
- Pass 5 repeats the labels of pass 4: the algorithm stops with $\tilde{d} = (0, 5, 2, 7, 10, 12)$, after 5 passes instead of the $n = 6$ allowed.
- The shortest path tree has the arcs $(1, 3)$, $(3, 2)$, $(3, 5)$, $(5, 4)$, $(4, 6)$, and the shortest path to 6 is $1, 3, 5, 4, 6$, of length $2 + 8 - 3 + 5 = 12$.



Example 3.8: Bellman–Ford detects a negative cycle

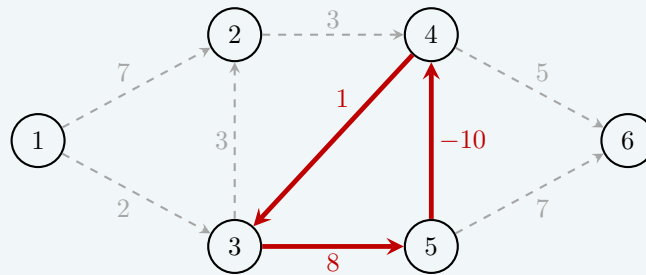
With the lengths d'' of Example 3.2 ($d''_{54} = -10$) the algorithm runs all the $n = 6$ passes:

k	$d^k(1)$	$d^k(2)$	$d^k(3)$	$d^k(4)$	$d^k(5)$	$d^k(6)$
0	0	∞	∞	∞	∞	∞
1	0	7 (1)	2 (1)	∞	∞	∞
2	0	5 (3)	2	10 (2)	10 (3)	∞
3	0	5	2	0 (5)	10	15 (4)
4	0	5	1 (4)	0	10	5 (4)
5	0	4 (3)	1	0	9 (3)	5
6	0	4	1	-1 (5)	9	5

- In pass 6 the label of node 4 decreases from 0 to -1 : $d^6 \neq d^5$, and a negative cycle is reachable from 1.
- Reading backwards from node 4 (bold entries): row 6 gives $pred = 5$, row 5 for node 5 gives 3, row 4 for node 3 gives 4, then 5, 3, 1. The walk is

1, 3, 5, 4, 3, 5, 4, of length $2 + 8 - 10 + 1 + 8 - 10 = -1 = d^6(4)$.

- Node 3 repeats: the walk contains the cycle 3, 5, 4, 3, of length -1 (red).



The Bellman–Ford algorithm computes d^k , the shortest walks with at most k arcs, in $O(nm)$ time. Without negative cycles reachable from s it stops within n passes with the distances; a change in pass n reveals a negative cycle, found by reading the predecessors backwards.

3.6 Dijkstra's algorithm

- In this section all lengths are non-negative: $d_{ij} \geq 0$ for every arc. Then every cycle has length ≥ 0 , and (A2) holds.
- The nodes are **permanent**, when their label is final, or **temporary**. At every iteration the temporary node with the smallest label becomes permanent.

Algorithm

Algorithm: Dijkstra's algorithm

Input: a directed graph $G = (N, A)$ with lengths $d_{ij} \geq 0$, a source $s \in N$

Output: the distances $d(j) = \tilde{d}(j)$ and the predecessors $pred(j)$

- 1 **Step 0 (initialization).** Set $d(s) := 0$, $d(j) := +\infty$ for $j \neq s$; all nodes are temporary.
- 2 **Step 1 (selection).** If no temporary node has a finite label, STOP.
- 3 Otherwise make permanent the temporary node i with the smallest $d(i)$ (the smallest index, if more than one).
- 4 **Step 2 (update).** For each arc $(i, j) \in \delta^+(i)$ with j temporary: if $d(i) + d_{ij} < d(j)$, set $d(j) := d(i) + d_{ij}$ and $pred(j) := i$.
- 5 Go to Step 1.

Theorem 3.22: correctness of Dijkstra's algorithm

If $d_{ij} \geq 0$ for every arc, at every iteration of Dijkstra's algorithm:

- (a) every permanent node i has $d(i) = \tilde{d}(i)$;
- (b) every temporary node j has as $d(j)$ the minimum length of a path from s to j whose nodes other than j are all permanent ($+\infty$ if there is none).

At the stop, $d(j) = \tilde{d}(j)$ for every $j \in N$.

Proof. Both properties hold after Step 0, with no permanent node. Let i be the node selected in Step 1.

1. $d(i) = \tilde{d}(i)$. Let P be any path from s to i , and let h be its first temporary node. The part of P up to h has length $\geq d(h)$ by (b), and the rest has length ≥ 0 , since $d \geq 0$. Hence $d(P) \geq d(h) \geq d(i)$, and by (b) the value $d(i)$ is the length of a path: $d(i) = \tilde{d}(i)$.
2. (b) after Step 2. Let j be temporary and Q a path to j whose other nodes are permanent, with h the node before j . If $h = i$, then $d(Q) \geq \tilde{d}(i) + d_{ij} = d(i) + d_{ij}$, the value tested in Step 2.
3. If $h \neq i$, then h became permanent before i , through a path of permanent nodes of length $\tilde{d}(h)$; when h was selected, Step 2 gave $d(j) \leq \tilde{d}(h) + d_{hj} \leq d(Q)$.

Hence the new minimum is $\min\{d(j), d(i) + d_{ij}\}$, attained by a path, which is the label

left by Step 2. At the stop, the temporary nodes have $d(j) = +\infty$: by (b) no arc leaves the permanent set towards them, so they are not reachable and $\tilde{d}(j) = +\infty$. $\square$

- For the s - t problem the algorithm can stop as soon as t is permanent, by (a).
- The arcs $(pred(j), j)$ form a shortest path tree: $pred(j)$ is permanent before j , and when it is last set $d(j) = d(pred(j)) + d_{pred(j)j}$ with both labels final.

Theorem 3.23: running time of Dijkstra's algorithm

With the labels in an array, Dijkstra's algorithm runs in $O(n^2)$ time.

Proof. There are at most n iterations, and Step 1 scans the temporary labels in $O(n)$ time. Step 2 scans the forward star of the selected node, and each node is selected once, so all the updates take $O(m)$ time. The total is $O(n^2 + m) = O(n^2)$, since $m \leq n(n-1)$. $\square$

- With the temporary labels in a binary heap, Step 1 and each decrease of a label take $O(\log n)$ time: the total is $O((n+m)\log n)$, better than $O(n^2)$ on sparse graphs.

3.6.1 Dijkstra's algorithm on the reference graph

Example 3.9: Dijkstra from node 1

We run the algorithm with the lengths of [Example 3.1](#). Each row is an iteration: the selected node becomes permanent (bold label), and a label that decreased has the new predecessor in brackets.

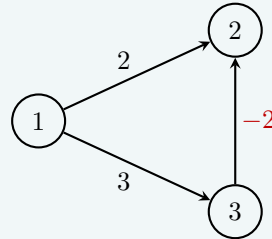
iteration	selected i	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$	$d(6)$
1	1	0	7 (1)	2 (1)	∞	∞	∞
2	3	0	5 (3)	2	∞	10 (3)	∞
3	2	0	5	2	8 (2)	10	∞
4	4	0	5	2	8	10	13 (4)
5	5	0	5	2	8	10	13
6	6	0	5	2	8	10	13

- In iteration 2 the label of node 2 drops from 7 to $2 + 3 = 5$; in iteration 5 the arc $(5, 6)$ offers $10 + 7 = 17 > 13$, and the arc $(5, 4)$ ends in a permanent node.
- The nodes become permanent in the order 1, 3, 2, 4, 5, 6, with $\tilde{d} = (0, 5, 2, 8, 10, 13)$; the predecessors give the shortest path tree of [Example 3.3](#).

3.6.2 Negative lengths

Example 3.10: a negative arc breaks Dijkstra's algorithm

In the graph below, with $s = 1$, the algorithm makes node 2 permanent with $d(2) = 2$, then node 3 with $d(3) = 3$; the arc $(3, 2)$ ends in a permanent node and is ignored. The path 1, 3, 2 has length $3 - 2 = 1 < 2$.



On the reference graph with the lengths d' of Example 3.7 ($d'_{54} = -3$), Dijkstra's algorithm makes the nodes permanent in the order 1, 3, 2, 4, 5, 6 and returns $d(4) = 8$ and $d(6) = 13$: node 4 is permanent before node 5, so the shorter path 1, 3, 5, 4 of length 7 is missed, and with it the path to 6 of length 12.

- The proof of Theorem 3.22 uses $d \geq 0$ in one point: the rest of P after its first temporary node has length ≥ 0 . A negative arc breaks exactly this step.

With non-negative lengths, Dijkstra's algorithm makes permanent, at every iteration, the temporary node with the smallest label, whose label is then final; it runs in $O(n^2)$ time. With a negative arc it can fail, and the Bellman–Ford algorithm is used instead.

3.7 Typical mistakes

Warning

- **Running Dijkstra's algorithm with a negative arc.** Its labels can be wrong, as with $d'_{54} = -3$ on the reference graph; use Bellman–Ford, or the topological order if the graph is acyclic.
- **Adding a constant to every length.** It makes the lengths non-negative but changes the shortest paths: a path with k arcs grows by k times the constant.
- **Swapping the sign of the potentials.** With $+1$ in the tail of $\mathbf{B}$ the dual constraints are $\pi_i - \pi_j \leq d_{ij}$ and the optimal potentials are $\pi_i = -\tilde{d}(i)$.
- **Skipping the test of pass n .** After $n - 1$ passes the labels are the distances only without negative cycles; a change in pass n reveals one.
- **Expecting Bellman–Ford to see every negative cycle.** It detects only the cycles reachable from s , while a negative cycle anywhere in G makes the LP model unbounded.
- **Updating a permanent node in Dijkstra's algorithm.** Its label is final; only temporary nodes are updated.

- **Looking for a topological order in a graph with cycles.** It exists only in acyclic graphs: the reference graph has none.

3.8 In summary

Summary

- The distance $\tilde{d}(j)$ is the length of a shortest path from s to j ; it is well defined when no negative cycle is reachable from s (A2).
- Subpaths of shortest paths are shortest; the distances satisfy Bellman's equations, and a shortest path tree exists.
- Feasible potentials, $\bar{c}_{ij} = d_{ij} - \pi_i + \pi_j \geq 0$, and a path of tight arcs certify optimality.
- The LP model is a unit flow, $\mathbf{B}\mathbf{x} = \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$; its dual is $\max\{\pi_s - \pi_t : \pi_i - \pi_j \leq d_{ij}\}$, optimal at $\pi_i = -\tilde{d}(i)$.
- Acyclic graphs: topological order, $O(n + m)$, any lengths. Bellman–Ford: $O(nm)$, any lengths, detects negative cycles. Dijkstra: $O(n^2)$, non-negative lengths.

The reference graph at a glance.

lengths d (Dijkstra)	$\tilde{d} = (0, 5, 2, 8, 10, 13)$, order 1, 3, 2, 4, 5, 6
LP models	$z(\text{SP}(1, 6)) = 13$, $z(\text{SP}(1)) = 38$
lengths d' , $d'_{54} = -3$ (Bellman–Ford)	$\tilde{d} = (0, 5, 2, 7, 10, 12)$, 5 passes
lengths d'' , $d''_{54} = -10$	negative cycle 3, 5, 4, 3 of length -1
acyclic graph, arc (3, 4)	order 1, 3, 2, 5, 4, 6; $\tilde{d} = (0, 5, 2, 3, 10, 8)$