Chapter 1. Minimum spanning trees and shortest paths

Section 1.1. Introduction

Discrete optimization or combinatorial optimization involves problems in which we have to choose the best solution from a finite number of relevant solutions.

A typical example of a discrete optimization problem is the following. We are given a set of elements $\{e_1, \dots, e_n\}$, and each element has a weight w_i. The problem is to find a k-element subset S of maximum weight $w(S) = \sum_{e_i \in S} w_i$.

An *instance* of this problem is the following set of eight elements with weights, where the problem is to find a 5-element subset of maximum weight.

i	1	2	3	4	5	6	7	8
w_i	3	6	5	8	4	9	2	6

There are $\binom{8}{5}$ feasible solutions. The *optimal* solution is the set $S = \{2, 3, 4, 6, 8\}$ with weight $w(S) = 6 + 5 + 8 + 9 + 6 = 34$.

The finiteness of the solution set suggests that the brute-force approach of *exhaustive* or *explicit enumeration* will be effective: simply generate all feasible solutions, examine their costs, and select the best one. Such an approach can be very time-consuming; as there is an upper bound on the number of operations that a computer can perform per time period, problems of very limited size only are solvable by explicit enumeration, and we have to search for faster algorithms when we want to start solving discrete optimization problems.

Section 1.2. Graph theory: basic definitions

A (undirected) *graph* G is a pair (V, E), where V is a set of elements called *vertices* or *nodes*. E is a set of 2-element subsets of V called *edges*. A graph $G = (V, E)$ has a graphical representation in which V is drawn as a set of dots and E as a set of line segments each connecting two dots.

Example

$$V = \{1, 2, 3, 4, 5\}$$
$$E = \{\{1,2\}, \{1,4\}, \{1,5\}, \{2,3\}, \{2,4\}, \{3,4\}, \{4,5\}\}$$

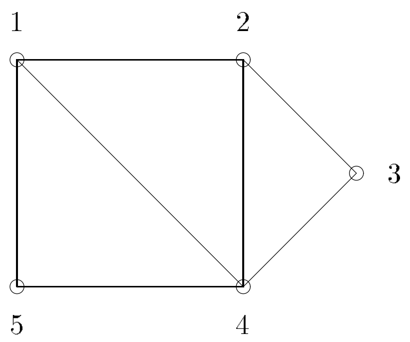


Figure 1.1

A vertex v and an edge e are called *incident* if $v \in e$. Two vertices v and w are called *adjacent* if $\{v, w\} \in E$. The *degree* of a vertex v, denoted by d_v or $d(v)$, is defined as the number of edges to which v is incident. The total number of vertices is denoted by $|V|$ or n, and the total number of edges by $|E|$ or m. The following proposition is elementary:

Proposition 1.1. $\sum_{v \in V} d_v = 2 \cdot |E|$.

A *walk* is a sequence of vertices and edges $(v_1, \{v_1, v_2\}, v_2, \{v_2, v_3\}, \ldots, v_{K-1}, \{v_{K-1}, v_K\}, v_K)$. Usually, the edges are omitted in the notation of a walk, i.e., it is written down as a sequence of vertices $(v_1, v_2, \ldots, v_K)$. The *length* of a walk is equal to the number of edges in it. Hence, the length of the walk $(v_1, \ldots, v_K)$ is $K - 1$. A walk is called a *path* if the vertices $v_1, \ldots, v_K$ are all different. A *cycle* is a closed walk, i.e., a walk with $v_K = v_1$. A *circuit* is a cycle where $v_1, \ldots, v_{K-1}$ are all different.

In the graph of Figure 1.1,

- (1,3,2,5,3,4)	is a walk, but not a path;	- (1,3,2,5)	is a path;
- (1,3,2,5,3,4,1)	is a cycle, but not a circuit;	- (1,3,2,1)	is a circuit.

A graph $G = (V, E)$ is *connected*, if there is a path between each pair of vertices. A graph is *acyclic* if it does not contain any cycles.

Theorem 1.2.

(a) If a graph $G = (V, E)$ is acyclic, then $|E| \leq |V| - 1$,

(b) If a graph $G = (V, E)$ is connected, then $|E| \geq |V| - 1$.

Proof.
We only prove part (a) of the Theorem; part (b) can be proven by a similar reasoning. We use induction on $|V|$. Since the theorem trivially holds for $|V| = 2$, we may state our induction-hypothesis that the theorem is true for each graph on $|V| = n - 1$ vertices. We have to show that the theorem then also holds for each graph on n vertices.

Suppose to the contrary that there exists an acyclic graph $G = (V, E)$ on n vertices with $E \geq |V| = n$. If G contains a vertex with degree equal to zero or one, then we just remove this vertex and the corresponding edge (in case of degree one) and obtain an acyclic graph on $n - 1$ vertices with at least $n - 1$ edges, which contradicts the induction-hypothesis. Hence, we know that each vertex then must have degree at least equal to two. Since the graph is assumed to be acyclic, we can travel from each vertex to another one that we have not visited before. But after having traversed $n - 1$ edges, we have visited each vertex. We are then in a vertex with degree at least two, so we can leave this vertex through an edge that we have not traversed before and that must be incident to a vertex that we have already visited, which implies that G contains a cycle. This contradiction proves part (a) of Theorem 1.2. □

A *tree* $T = (V, E)$ is a connected acyclic graph. It contains a unique path between each pair of vertices. A direct corollary of Theorem 1.2 is that $|E| = |V| - 1$.

One of the earliest applications of graph theory is a characterization of an *Eulerian graph*; an Eulerian graph is a graph that contains a cycle in which each edge is traversed exactly once.

Theorem 1.3. (Euler)
A connected graph $G = (V, E)$ contains an Euler-cycle if and only if the degrees of all vertices are even.

A *subgraph* of a graph $G = (V, E)$ is a graph $G' = (V, E')$ with $E' \subseteq E$. An *induced* subgraph of G is a graph $G'' = (V'', E'')$ in which $V'' \subset V$ and E'' consists of all edges in E between two vertices of V''.

Figure 1.2 shows a subgraph and an induced subgraph (induced by the vertices $\{1, 4, 5\}$) of the graph of Figure 1.1, respectively.

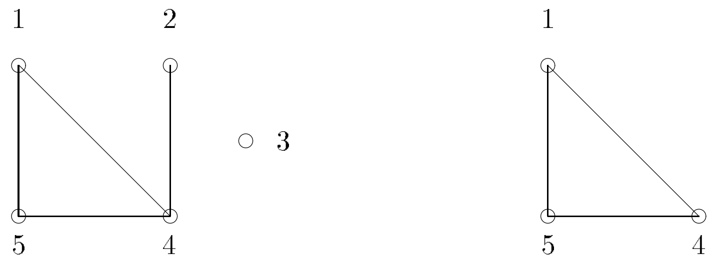


Figure 1.2.

If a graph is not connected, then it can be split into a set of *connected components* $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_K = (V_K, E_K)$, where $V_1, \ldots, V_K$ form a partition of the set of vertices and where each $G_k = (V_k, E_k)$ is the subgraph induced by V_k ($k = 1, \ldots, K$).

A *directed* graph or digraph $G = (V, A)$ consists of a set of vertices V, and a set of *ordered* 2-element subsets. An example of a directed graph is depicted below.

$$\begin{aligned} V &= \{1, 2, 3, 4, 5\} \\ A &= \{(1, 5), (2, 1), (2, 4), (3, 2), \\ &\quad (4, 1), (4, 3), (5, 4)\} \end{aligned}$$

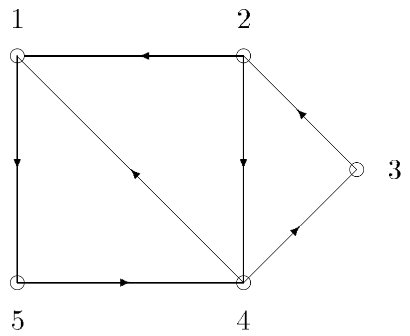


Figure 1.3.

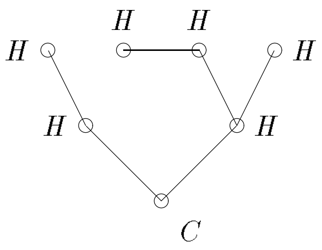
The terminology for digraphs resembles the terminology for (undirected) graphs. For instance, a walk in a digraph is a sequence of vertices and arcs: $(v_1, (v_1, v_2), v_2, (v_2, v_3), \ldots, v_{K-1}, (v_{K-1}, v_K), v_K)$ or shortly $(v_1, v_2, \ldots, v_K)$. Paths, cycles, and circuits are defined analogously. Note that the arcs in a walk have to be traversed in the right directions, i.e., the directions indicated by the arrows.

In the graph of Figure 1.3,

(1,5,4,3,2,4)	is a walk;	(1,5,4,3)	is a path;
(1,5,4,3,2,4,1)	is a cycle;	(1,5,4,1)	is a circuit.

Section 1.3. The minimum spanning tree problem

In small local area telephone networks, houses are connected with a central unit by a cable (coax or fiber). To realize such a network, sleeves must be dug in which these cables are laid. Sleeves may be dug between a pair of houses, and between a house and the central unit. There may be more cables in one sleeve. Since digging sleeves is an expensive process, the total length of the sleeves has to be minimized. A typical solution is drawn below.



This problem can be modeled as a minimization problem in graphs as follows. The central unit and the houses are represented by nodes. Between each pair of nodes v and w, there is an edge $e = \{v, w\}$ with a cost c_e that is equal to the length of a sleeve between v and w. We are looking for a subgraph $G' = (V, E')$ in which there is a path between each vertex and the central unit; we wish to find the subgraph for which the total cost of the edges in E' is minimal.

Formally, this is the problem of finding a minimum spanning tree $T = (V, E')$ in a graph $G = (V, E)$ with respect to a cost function $c : E \to \mathbb{Z}^+$, such that $c(E') = \Sigma_{e \in E'} c_e$ is minimal.

The following algorithm constructs a minimum spanning tree edge by edge in a greedy fashion.

Algorithm MST-1 (Kruskal)

Input: A connected graph $G = (V, E)$, and a cost function on the edges $c : E \to \mathbb{Z}^+$. Let $m = |E|$ be the number of edges.

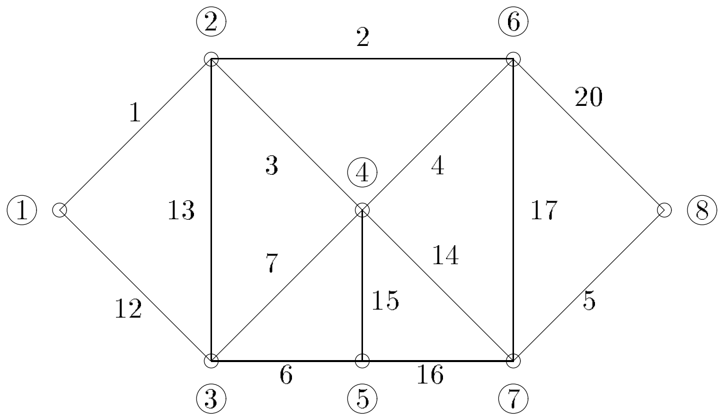
Output: A minimum spanning tree $T = (V, E'), E' \subseteq E$.

Step 1. Number the edges according to non-increasing cost, that is, $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$. Set $E' \leftarrow \emptyset$. Set $j \leftarrow 1$.

Step 2. Suppose that the edges $e_1, \ldots, e_{j-1}$ have been considered, and that an acyclic graph (V, E') has been formed. If $E' \cup \{e_j\}$ is acyclic, then e_j is added to E'; otherwise, E' remains unchanged. Set $j \leftarrow j + 1$.

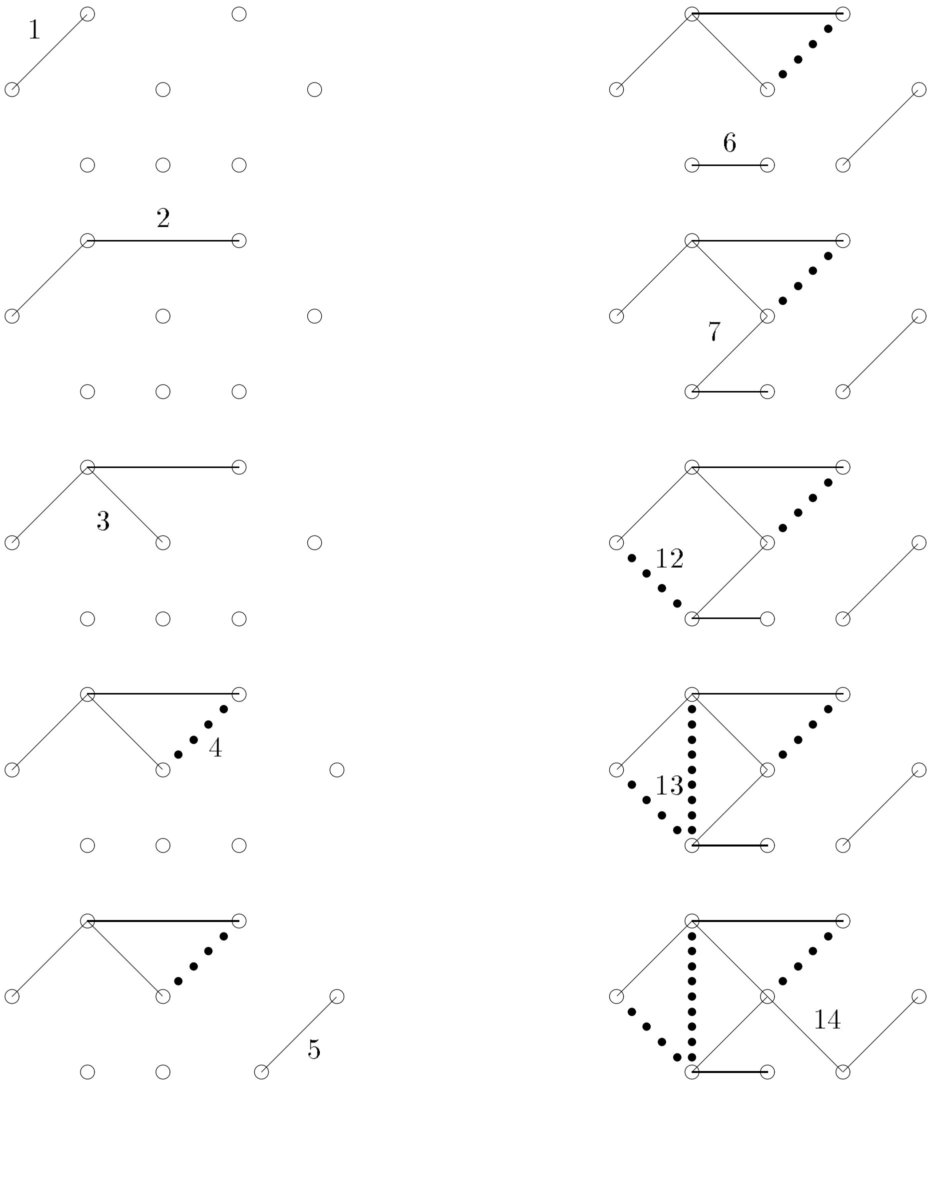
Step 3. If the graph (V, E') is connected, then stop; otherwise, go to Step 2.

Example



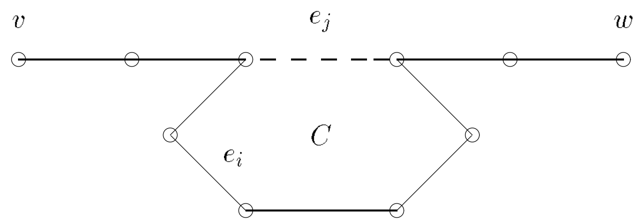
Ordering of the edges

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
e_i	{1,2}	{2,6}	{2,4}	{4,6}	{7,8}	{3,5}	{3,4}	{ 1,3}	{ 2,3}	{4,7}	{4,5}	{ 5,7}	{ 6,7}	{ 6,8}
$c(e_i)$	1	2	3	4	5	6	7	12	13	14	15	16	17	20
$\in E'(Y/N)$	Y	Y	Y	N	Y	Y	Y	N	N	Y				



Theorem 1.4.
Algorithm MST-1 constructs a minimum spanning tree.

Proof.
Let T_1 denote the tree determined by MST-1; renumber the edges such that e_k is the edge that was added to the tree by MST-1 as the k'th edge. Suppose to the contrary that MST-1 does not determine a minimum spanning tree; let T_2 be a minimum spanning tree. Compare T_1 and T_2; let e_i be the first edge from T_1 that is not in T_2. If we add e_i to T_2, then we get a cycle C. As e_i is chosen by MST-1, we know that C contains at least one edge other than $e_1, \ldots, e_{i-1}$; let this be e_j. Since T_2 contains the edges $e_1, \ldots, e_{i-1}$ and e_j, we have that e_j also was available when e_i was added; as e_i was chosen by MST-1, we know that $c(e_i) \leq c(e_j)$. Hence, if we replace e_j with e_i in T_2, then we obtain a spanning tree with cost no more than the cost of T_2. If we proceed our analysis with this new minimum spanning tree T_2, then we end up with the contradiction that the minimum spanning tree T_2, though consisting of the same set of edges as T_1, has length smaller than the length of T_1. This contradiction obviously shows that MST-1 determines a minimum spanning tree.



□

We now present another algorithm for the minimum spanning tree problem. This algorithm also builds a minimum spanning tree edge by edge in a greedy way, but it differs from MST-1 by maintaining one big acyclic connected component and a set of isolated vertices. In each iteration, an edge connecting an isolated vertex to the big component is added to the component.

The algorithm is implemented by use of a basic technique, called labeling. Each isolated vertex v contains a label expressing the cost of connecting v to the big component by means of the cheapest edge. This label is denoted by $l(v)$.

The technique of labeling returns later in this chapter and in Chapter 2. A label may contain any information, like the cost of an edge, a neighboring vertex or, other information with regard to the vertex.

The algorithm makes use of two additional sets V' and E'. The set V' contains all vertices that are in the big component; the set E' contains all edges that are used to form the big component.

Algorithm MST-2 (Prim-Dijkstra)

Input: A connected graph $G = (V, E)$, and a cost function on the edges $c : E \to \mathbb{Z}^+$.

Output: A minimum spanning tree $T = (V, E')$, $E' \subseteq E$.

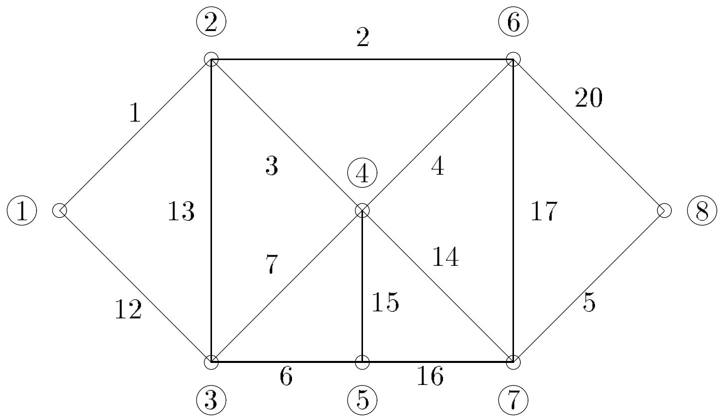
Step 1. Choose any vertex $v \in V$ as the root; set $V' \leftarrow \{v\}$. For any $w \in V \backslash \{v\}$, set $l(w) \leftarrow c(\{v, w\})$; if the edge $\{v, w\}$ does not exist, then $c(\{v, w\}) \leftarrow \infty$. Set $E' \leftarrow \emptyset$.

Step 2. Determine a vertex $w \in V \backslash V'$ with minimum label. Let $e = \{v, w\}$ be an edge that connects w with V' at cost $l(w)$. Add $\{v, w\}$ to E' and add w to V'.

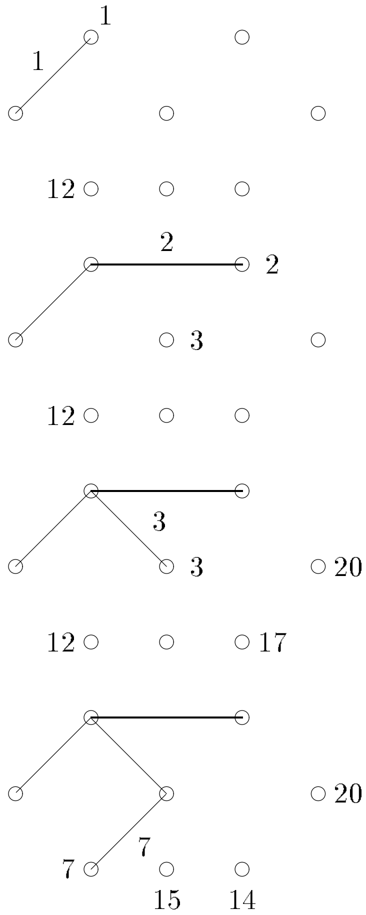
Step 3. Update the labels $l(w')$ for all vertices $w' \notin V'$ for which the edge $\{w, w'\}$ exists: set $l(w') \leftarrow \min\{l(w'), c(\{w, w'\})\}$.

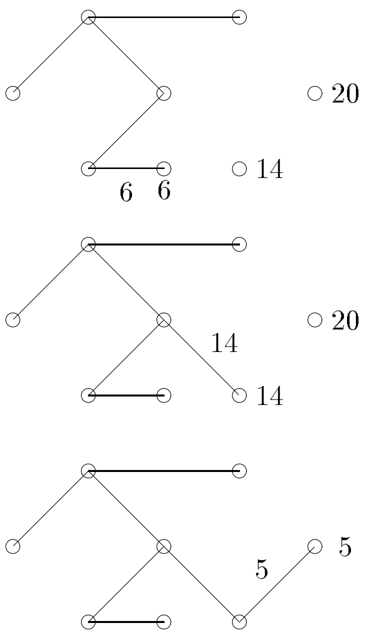
Step 4. If $V' = V$, then stop; otherwise, go to Step 2.

Example



iteration \ vertex	1	2	3	4	5	6	7	8
0		1	12	∞	∞	∞	∞	∞
1			12	3	∞	2	∞	∞
2			12	3	∞		17	20
3			7		15		14	20
4					6		14	20
5							14	20
6								5





Note that the edges are not added according to non-decreasing cost.

Algorithm MST-2 lends itself for an analysis of the number of elementary steps performed by the algorithm. Elementary operations are: addition, multiplication, comparison of two numbers, and addition and deletion of elements (here edges or vertices) to and from a set.

MTS-2 performs $n-1$ iterations. In each iteration, an edge (and a vertex) is added to the tree. We distinguish two elementary operations.

The first operation is that we have to determine the minimum label from among the set of labels of the vertices in $V \backslash V'$. This takes n comparisons in the worst case.

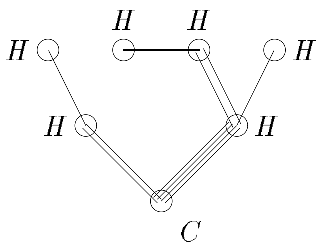
The second one is that, when w moves from $V \backslash V'$ to V', we have to compare the cost of each edge $\{w, w'\}(w' \in V \backslash V')$ with the label of w'. This takes at most $n-1$ comparisons. This operation is also performed at the initialization. Hence, the total number of elementary steps amounts to at most $n + (n-1)(n+n) \leq 2n^2$.

We use the following notation. If there exists a positive integer k such that for two functions $f, g : I\!N \rightarrow I\!R$ we have that $f(n) \leq k|g(n)|$ for all $n \geq n_0$ for some $n_0 \in I\!N$, then we write $f(n) = O(g(n))$.

According to this notation, we have that the number of elementary steps that have to be performed by MST-2 to calculate a minimum spanning tree for a graph $G = (V, E)$ on n vertices amounts to $O(n^2)$; hence, the *running time* $r(n)$ of algorithm MST-2 amounts to is $O(n^2)$.

Section 1.4. The shortest path problem

Reconsider the problem of the previous section, where a telephone company must connect a set of houses to a central unit. If not the cost of digging the sleeves, but the cost of the cables is the most important cost component, then the problem becomes one of minimizing the total length of cable laid in the ground. Therefore, we are then asked to determine the shortest route between the central unit and each house. A typical solution is drawn below.



We can model this problem in terms of graphs in the following way. Consider a graph $G = (V, E)$ and a length function on the edges $c : E \to \mathbb{Z}^+$. Furthermore, there is a special vertex $s \in V$ from which all shortest paths to all other vertices must be determined.

The first algorithm to solve this problem was proposed by Dijkstra. He used the more general setting of the problem in digraphs. This is no serious limitation, however, since a graph can be 'directed' by replacing each edge $\{v, w\}$ with length $c(\{v, w\})$ with two arcs (v, w) and (w, v) that both have length $c(\{v, w\})$.

The idea behind the algorithm is to use labels for all vertices; these labels consist of upper bounds on the lengths of the shortest paths from s to each vertex. Each upper bound is decreased until its value equals the length of a shortest path.

The algorithm makes use of two additional sets V' and A'. The set V' contains all vertices v for which the shortest path from s to v has been determined; the set A' contains all arcs that are used in the shortest paths from s to the vertices v in V'.

Algorithm SP (Dijkstra)

Input: A digraph $G = (V, A)$ and a length function $c : A \to \mathbb{Z}^+$.
A special vertex $s \in V$.

Output: The lengths of all shortest paths from s to the other vertices: $l : V \backslash \{s\} \to \mathbb{Z}^+$.

STEP 1. For any vertex $v \in V \backslash \{s\}$, determine the initial label as the distance from v to s, that is, set $l(v) \leftarrow c((s, v))$; if the arc (s, v) does not exist, then set $l(v) \leftarrow \infty$. Set $V' \leftarrow \{s\}$ and $A' \leftarrow \emptyset$.

STEP 2. Determine a vertex $w \in V \backslash V'$ with minimum label. Let v be a vertex in V' such that $l(w) = l(v) + c((v, w))$, that is, there exists a path of length $l(w)$ from s to w that goes through v. Add (v, w) to A' and add w to V'.

STEP 3. Update the labels $l(w')$ for all vertices $w' \notin V'$ for which there exists an arc (w, w'): set $l(w') \leftarrow \min\{l(w'), l(w) + c((w, w'))\}$.

STEP 4. If $V' = V$, then stop; otherwise, go to STEP 2.

Theorem 1.5.
Algorithm SP calculates the lengths of the shortest paths from s to the other vertices correctly.

Proof.
We prove with induction that:

(a) For $x \in V'$: $l(x)$ is the length of the shortest path from s to x.

(b) For $y \in V \backslash V'$: $l(y) = \min\limits_{x \in V'} \{l(x) + c((x,y))\}$.

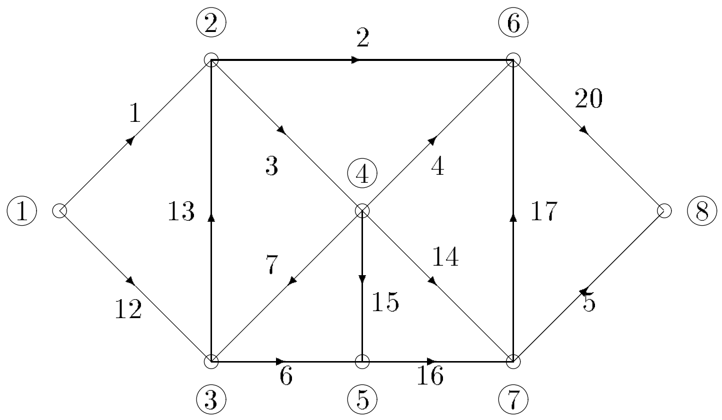
This is certainly true after the initialization. Now consider the iteration in which v is moved from $V \backslash V'$ to V'.
Proof of (a): Consider any path P from s to v. Let y be the first vertex of P in $V \backslash V'$, and let x be its direct predecessor. Hence, $P = (s, \ldots, x, y, \ldots, v)$, where $(s, \ldots, x)$ in V' and $y \notin V'$. Then the length of the path P is:

$$
\begin{aligned}
c(P) &= c((s,\ldots,x)) + c((x,y)) + c((y,\ldots,v)) \\
&\geq c((s,\ldots,x)) + c((x,y)) && \text{(lengths of arcs are positive)} \\
&\geq l(x) + c((x,y)) && \text{(by hypothesis (a))} \\
&\geq l(y) && \text{(by hypothesis (b))} \\
&\geq l(v) && \text{(v is chosen, so its label is minimal).}
\end{aligned}
$$

Hence, $l(v)$ is equal to the length of the shortest path from s to v, which implies that property (a) is restored at the end of the iteration.
In the second part of the iteration, the labels of the vertices in $V \backslash V'$ are updated so that (b) holds again, too. □

This algorithm resembles MST-2. By means of a proof similar to the one used for MST-2, we can show that Algorithm SP performs $O(n^2)$ elementary steps in case of a digraph on n vertices.
Below, an example of Algorithm SP in action is depicted; vertex 1 serves as s.

Example



Labels during the algorithm

iteration \ vertex	1	2	3	4	5	6	7	8
1		1	12	∞	∞	∞	∞	∞
2			12	4	∞	3	∞	∞
3			12	4	∞		20	23
4			11		19		18	23
5					17		18	23
6							18	23
7								23