This algorithm is based on the Dynamic Programming idea. It is due to E. W. Dijkstra and it takes $O(n^2)$ time for graphs of n nodes, in the case in which labels form a non-negative ic-semiring.

Dijkstra's algorithm is correct also for any labeled directed graph provided that there are not negative cycles, that is, cycles with negative sum of the labels of its arcs. For simplicity, we will present Dijkstra's algorithm in the case where labels of the arcs are non-negative reals.

5.6.8. Single Source Shortest Path of a Directed Graph.

In this section we present an algorithm for computing the shortest path from a given initial node (also called source) to any other node of a directed graph. We will call this algorithm the *Single Source Shortest Path algorithm*, or SSSP algorithm, for short. Also this algorithm, as the one in Section 5.5 for computing a minimal spanning tree of an undirected graph (see page 121), is due to Prof. E. W. Dijkstra. This algorithm can be viewed as an application of the so called *Primal-Dual algorithm* (see [**11**, Capter 5]) and thus, its correctness can be derived from that of the Primal-Dual algorithm. Now, for reasons of simplicity we asssume that the edges have non-negative lengths.

ALGORITHM 5.6.18.
Computation of a Single Source Shortest Path in a Directed Graph. (SSSP *algorithm*)

Input: A directed, connected graph G with labels on the edges. A node n_0, called *source*, of G. No assumptions are made on the arcs which depart from n_0 or arrive at n_0.
- For each edge $\langle n, m\rangle$, its label $\lambda(n, m)$, also called the *length* of the edge, is a non-negative real number.
- For each node n, we assume that there exists the edge $\langle n, n\rangle$ whose label $\lambda(n, n)$ is 0.

Output: for each node m in G, we compute a *shortest path* from node n_0 to node m, that is, a path from n_0 to m with minimal *length*, that is, minimal sum of the labels of its edges. The minimum is taken over all paths from n_0 to m.
With each node n, we associate a label $\Lambda(n)$ which is the length of a shortest path from n_0 to n.

A. *With each node n we associate a label $\Lambda(n)$ which is the length of a shortest path from n_0 to n.*

1. $\Lambda(n_0) := 0$; *Reach* $:= \{n_0\}$;
2. for each edge from n_0 to n, $\Lambda(n) := \lambda(n_0, n)$;
3. *Repeat* among all nodes not in *Reach*, take a node, say p, with *minimal* label $\Lambda(p)$;
   3.1 *Reach* $:=$ *Reach* $\cup \{p\}$;
   3.2 for each edge $\langle p, m\rangle$ with m not in *Reach,*
   if m has already a label
   then (3.2.1) $\Lambda(m) := min\{\Lambda(p)+\lambda(p, m), \Lambda(m)\}$
   else (3.2.2) $\Lambda(m) := \Lambda(p)+\lambda(p, m)$;

B. *We find by backtracking a shortest path from node n_0 to a node m in G.*

4. Choose *any* node p, different from m, such that $\Lambda(p) = -\lambda(p, m) + \Lambda(m)$, and
5. Recursively, find a shortest path from n_0 to p.

The following remarks illustrate a few important points of the above SSSP algorithm.

(i) The source node n_0 can be any node of the given graph G. In particular, there may be edges going into the source node.

(ii) At Point 4 if there exists more than one node p, different from m, such that $\Lambda(p) = -\lambda(p, m) + \Lambda(m)$, then we may choose any one of them.
If this is the case, then there exists more than one path from n_0 to m with minimal length.

(iii) If $\langle n_0, n_1, \ldots, n_{k-1}, n_k \rangle$ is a shortest path from n_0 to n_k,
then $\Lambda(n_k) = \sum_{i \geq 0}^{k-1} \lambda(n_i, n_{i+1})$.

(iv) This SSSP algorithm is similar to Dijkstra's MST algorithm **[5]** for computing a minimal spanning tree of undirected graphs with labels on the edges (see Algorithm 5.5.1 on page 121). Analogously to Dijkstra's MST algorithm:
- the SSSP algorithm is a *greedy* algorithm, that is, the local optimum is a global optimum (recall what has been said for the MST algorithm at page 118),
- the nodes in *Reach* may be viewed as *red* nodes and the nodes not in *Reach* may be viewed as *blue* nodes, and
- the SSSP algorithm, after transforming a blue node p into a red node (see Point 3.1), readjusts (see Point 3.2.1) or generates (see Point 3.2.2) the label of each blue node m such that there exists an edge $\langle p, m \rangle$.

Now we present the various invariants of the loops of the above Dijkstra's SSSP Algorithm 5.6.18. Those invariants clarifies the way that algorithm works.

The invariant of the *Repeat* statement (see Point 3) is:

> "the label of any node n in the set *Reach* is the length of the shortest path from the source node n_0 to n".

The invariant at Point 3.2 is:

> "each node m *not* in *Reach* such that there is an arc $\langle p, m \rangle$ with $p \in$ *Reach*, has a label which is the length of a shortest path $\langle n_0, \ldots, p, m \rangle$ from the source node n_0 to m such that the nodes $n_0, \ldots, p$ are all in *Reach*".

The invariant at Point 5 is:

> "in a shortest path from the source node n_0 to the node m, the last edge is from node p to node m".

The correctness of Dijkstra's algorithm, which we will not prove here, is based on the following fact. Let us assume that we have computed the shortest path from node n_0 to every node in a subset W of the nodes such that $n_0 \in W$. Let us consider the shortest path, say p, from node n_0 to any node y not in W such that for every node x if there is an arc from x to y in G then $x \in W$. Then the path p has the property that it goes through the nodes of W (except for y itself).

Note that if at Point 3 we take a node p which has no *minimal* label $\Lambda(p)$, the algorithm is *not* correct.

EXAMPLE 5.6.19. Given the graph of Figure 5.6.4 and the node n_0, after Point 3 of Dijkstra'a algorithm we get the 'boxed labels' for each node. Then, having labeled all nodes with the boxed labels, we may find *by backtracking* every edge whose label is exactly the difference of the labels of its nodes. By doing so (see Points 4 and 5 of Dijkstra's algorithm), we get the following two sequences of values:

(i) $\boxed{12} \xleftarrow{6} \boxed{6} \xleftarrow{1} \boxed{5} \xleftarrow{2} \boxed{3} \xleftarrow{3} \boxed{0}$ and

(ii) $\boxed{12} \xleftarrow{3} \boxed{9} \xleftarrow{4} \boxed{5} \xleftarrow{2} \boxed{3} \xleftarrow{3} \boxed{0}$

which correspond to the following two shortest paths from node n_0 to node n_6 (see the thicker edges of the graph of Figure 5.6.4):

(i) $n_0 \longrightarrow n_2 \longrightarrow n_3 \longrightarrow n_5 \longrightarrow n_6$ and

(ii) $n_0 \longrightarrow n_2 \longrightarrow n_3 \longrightarrow n_4 \longrightarrow n_6$, respectively.

The total length of those shortest paths is 12, that is, the boxed label of the final node n_6. Note that by backtracking, from node n_6 we do *not* go back to node n_3 (because $12-9 \neq 5$). We go back either to node n_4 (because $12-6=6$) or to node n_5 (because $12-3=9$). □

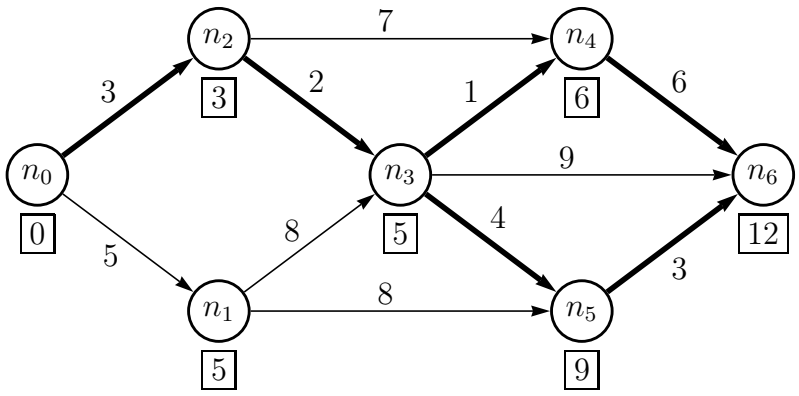

FIGURE 5.6.4. A directed graph with non-negative labels on the edges. The thicker edges are those belonging to the two shortest paths from node n_0 to node n_6. Those paths have both length 12.

For a graph of n nodes the time complexity of Dijkstra's SSSP algorithm is $O(n^2)$, and this is due to the fact that: (i) the body of the *Repeat* statement (see Point 3) is performed at most n times because there are at most n nodes not in *Reach*, and (ii) for each node p not in *Reach* there exist at most n edges of the form $\langle p, m\rangle$ with m not in *Reach* (see Point 3.2).

Dijkstra's algorithm shows that if we fix the initial node and the labels of the given graph form a non-negative ic-semiring, we may solve the shortest path problem in $O(n^2)$ time. This is an improvement over the time complexity for solving the

path:		worst case time complexity:	
from any node	to any node	$O(n^{2.81})$	(the complexity is that matrix multiplication)
from a fixed node	to any node	$O(n^2)$	
from a fixed node	to a fixed node	$O(n^2)$	(it is an open problem whether or not one can do better)

FIGURE 5.6.5. Complexity of the computation of a shortest path in a directed graph with n nodes.

shortest path problem between any two nodes. Indeed, we have shown that this problem requires the same amount of time which is needed for matrix multiplication, say $O(n^{2.81})$ time (see also Figure 5.6.5).

One may wonder whether or not it is possible to derive an even faster algorithm for the shortest path problem when we fix both the initial node and the target node. This is the so called s*ingle source single target shortest path problem*. The answer, to our knowledge, is still open, in the sense that no algorithm has been found for this last problem which has a better time performance, in the worst case, than the best algorithm known for the single source shortest path problem.

A final point is worth noticing. Let us consider an *undirected graph* with weights on its edges as a particular case of directed graph where for each edge from node x to node y with label w, there exists a symmetric edge from node y to node x with the same label w. In an undirected graph any pair of directed edges from node x to node y and vice versa will be denoted by x—y. Now, given an undirected graph G with non-negative weights on its edges, the shortest path from one node to another node may be made out of edges none of which belongs to the minimal spanning tree of G. This fact is illustrated in the graph of Figure 5.6.6. In that figure we use the convention of drawing every pair of directed edges of an undirected graph by one edge only (without arrowhead). In Figure 5.6.6 the shortest path from node b to node c is the edge b—c with weight 3, while the minimal spanning tree of the graph G has total weight 4 and it is made out of the edges a—b and a—c.

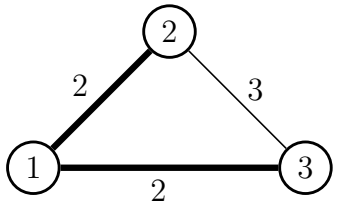

FIGURE 5.6.6. An undirected graph with non-negative labels on the edges. The thicker edges are those of the minimal spanning tree.

In what follows we present a Java program which implements Dijkstra's Single Source Shortest Path algorithm. It is made out of two files: `SSSP.java` and `Graph.java`, located in the same folder.