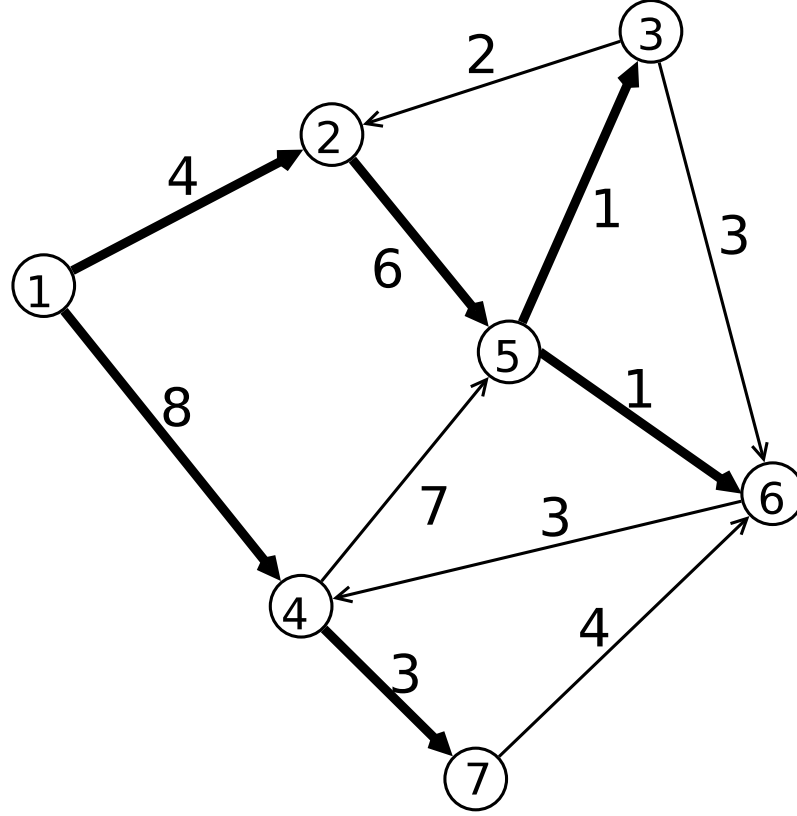


4.1

The Figure below shows the optimal solution. The total length of all paths are 55.



4.3

a) Initially $S = \{s\}$, $P = \emptyset$, $p = [0, -1, -1, -1, -1]$ and $y = [0, \infty, \infty, \infty, \infty]$.

Selecting $v = s \in S$ gives: $S = \{1, 3, 4\}$, $P = \{s\}$, $p = [0, 0, -1, 0, 0]$ and $y = [0, 3, \infty, 2, 4]$.

Selecting $v = 3 \in S$ (since y_3 is least for the elements in S) gives: $S = \{1, 4\}$, $P = \{s, 3\}$ and no modifications of p and y .

Selecting $v = 1 \in S$ gives: $S = \{4, 2\}$, $P = \{s, 3, 1\}$, $p = [0, 0, 1, 0, 0]$ and $y = [0, 3, 2, 2, 4]$.

Selecting $v = 2 \in S$ gives: $S = \{4\}$, $P = \{s, 3, 1, 2\}$ and no modifications of p and y .

Finally $v = 4$ is selected giving: $S = \emptyset$, $P = \{s, 3, 1, 2, 4\}$ and no modifications of p and y . Since $S = \emptyset$ the algorithm terminates, and the result is $p = [0, 0, 1, 0, 0]$ and $y = [0, 3, 2, 2, 4]$. The corresponding shortest path tree is shown in figure 1

The y and p obtained is wrong! The shortest path between s and 3 does not go directly from s to 3 with a length of 2 but goes from s via 1 to 3 with a length of

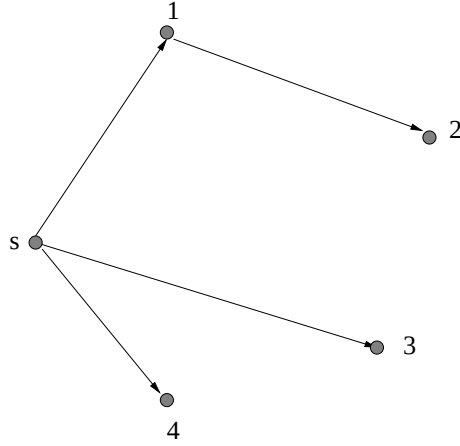


Figure 1: *Shortest path tree for the found solution.*

only 2. This is an example showing that Dijkstra's algorithm cannot handle negative edges.

- c) The Floyd-Warshall algorithm maintains a matrix giving an estimate on the shortest path distance. Initially this matrix contains the arc distances between node pairs. ∞ means that there are no arcs, and hence the initial estimate on the shortest path is infinity. The matrix is shown here:

from\to	s	1	2	3	4
s		3	∞	2	4
1	∞		-1	-2	∞
2	1	∞		∞	3
3	∞	∞	∞		∞
4	3	∞	∞	∞	

The first iteration modifies the matrix such that the first node (i.e. node s) is included in paths. Updating the matrix gives the following, updates are in bold:

from\to	s	1	2	3	4
s		3	∞	2	4
1	∞		-1	-2	∞
2	1	4		3	3
3	∞	∞	∞		∞
4	3	6	∞	5	

- d) The Bellmann-Ford algorithm should be used, since it can handle negative distances.

Bellmann-Ford is applied to the digraph in figure 2. Initially all distances from K to $[s, 1, 2, 3, 4]$ is $\pi = [\infty, \infty, \infty, \infty, \infty]$. The first iteration modifies this to $\pi = [0, 0, 0, 0, 0]$, since all nodes can be reached directly from K . The second iteration gives distances $\pi = [0, -1, -2, 0, 0]$, since $w_{12} + \pi_1 < 0$ and $w_{13} + \pi_1 < 0$. No more updates are carried out, and since $\pi_j \leq \pi_i + w_{ij}$ holds for arbitrary i and j (you may check this), no negative cycles exist.

- e) The inequality $\pi_j \leq \pi_i + w_{ij}$ can be explained as follows: π_i is the length of a shortest path from K to i and π_j is the length of a shortest path from K to j . Consider the path from K to j consisting of a shortest path from K to i and the arc from i to j . Obviously this path cannot be shorter than a shortest path between K and j , hence the inequality follows. Subtracting π_j on both sides in the inequality, yields $0 \leq \pi_i + w_{ij} - \pi_j$, hence $\pi_i + w_{ij} - \pi_j = w'_{ij} \geq 0$.

Consider any path P between nodes v_0 and v_k and denote nodes on the path by $v_0, v_1, \dots, v_k$. The length of P (in the graph with modified edge costs) is given by the following sum (we abbreviate π_{v_i} by π_i):

$$w'_P = \sum_{i=0}^{k-1} w'_{v_i v_{i+1}} = \sum_{i=0}^{k-1} (w_{v_i v_{i+1}} + \pi_i - \pi_{i+1}) = \sum_{i=0}^{k-1} w_{v_i v_{i+1}} + \sum_{i=0}^{k-1} (\pi_i - \pi_{i+1})$$

The above is obtained by inserting definitions of c' and splitting the sum. The last sum can be modified:

$$\sum_{i=0}^{k-1} (\pi_i - \pi_{i+1}) = (\pi_0 - \pi_1) + (\pi_1 - \pi_2) + \dots + (\pi_{k-1} - \pi_k) = \pi_0 - \pi_k$$

The last equality follows, since 2nd and 3rd term adds to zero and 4th and 5th term adds to zero and so forth. Denoting the cost of P with the original costs by w_P we get:

$$w'_P = w_P + \pi_0 - \pi_k$$

Thus for all paths between nodes v_0 and v_k the same constant $(\pi_0 - \pi_k)$ is added and hence all paths *with the same endpoint nodes* is in- or de-created by the same amount, and hence the shortest path with respect to w_{ij} is the same as with respect to w'_{ij} .

- f) The number of nodes is denoted n and the number of edges is denoted m . The Floyd-Warshall algorithm has complexity $O(n^3)$ since three loops with n iterations each are run in each other.

In Dijkstra's algorithm one node is considered during each iteration. Hence "Select a $i \in S$ such that y_i is minimal" has to be run n times. Each such find-min can obviously be done in $O(n)$ investigating the y array. Alternatively this can be replaced by a binary heap where find minimum require only $O(\log n)$. In total for one run of Dijkstra's algorithm this gives $O(n \log n)$.

In the **for** statement the update is only carried out *once* for each edge in total. If the binary heap is used, the heap has to be updated at a cost of $O(\log n)$ in total a cost of $O(m \log n)$. The complexity for Dijkstra's algorithm is $O(m \log n + n \log n) = O(m \log n)$

Since Dijkstra's algorithm has to be run n times, the total complexity is then $O(mn \log n)$.

In order to be able to handle negative edges, the modification described in question d) and e) may be carried out, including *one* run of the Ford-Bellmann algorithm.

This has complexity $O(mn)$ and hence the total complexity is dominated by the n applications of Dijkstra's algorithm, i.e. $O(mn \log n)$.

The Floyd-Warshall algorithm is simpler to implement than Dijkstra's algorithm and it has the same complexity ($O(n^3)$) regardless of the number of edges. The complexity of n applications of Dijkstra's algorithm depends on the number of edges. If the number of edges is $O(n)$, the total complexity is $O(n^2 \log n)$, which is better than $O(n^3)$. Also if the number of edges is $O(n^2)$, the total complexity is $O(n^3 \log n)$, which is worse than $O(n^3)$.

Thus if the graph is dense (i.e. the graph has many edges), Floyd-Warshall should be used and if the graph is sparse (i.e. the graph has few edges) n applications of Dijkstra's algorithm should be used.