Exercise 5.

The algorithm that solves the exercise consists of the following three steps:

1.) find minimal edge weight **m**;

2.) find width **w** of the widest path from the starting point (the capital) to the target point; by the width of the path we mean the minimum width of each of its edges;

3.) return the difference **w - m**, which is equal to the maximum width of the tank.

For the step 2 we use Dijkstra algorithm. At each iteration of the algorithm we choose an unconsidered node **n** with the widest path to it from initial node (denote node by **n** and width of such path by $\mathbf{w}_n$). Then we update all adjacent with **n** nodes $\mathbf{n}_i$, the width of the path to which from the capital is less than $\mathbf{min}(\mathbf{w}_n$, width of the edge from **n** to $\mathbf{n}_i)$, marking **n** as considered after it. We stop this process when we start to consider target node.

Proof of the correctness of the step 2 is literally the same as in the Dijkstra algorithm, so we omit it.

Complexity estimation of the algorithm is the same as in Dijkstra algorithm. Let us denote the number of nodes by N and the number of edges by M. Number of iterations in the step 2 is not greater than N. At each iteration we need to choose an unconsidered node with the highest value. Time cost of this operation depends on how this unconsidered nodes are stored. With vector-like storage this step may cost up to $O(N)$ while updating of the values costs $O(1)$. So totally we have complexity of the algorithm equals to $O(N^2 + M)$ (N times we need to find maximum and not more than $2M$ times we need to update values).

With priority queue storage we have complexity of the algorithm equals to $O(N\log(N) + M\log(N))$, because finding unconsidered node with highest value now costs $O(\log(N))$ as well as updating the node values.

In our C++ code we implement *host* and *policy* idioms (see for details *A. Alexandrescu* "Modern C++ Design: Generic Programming and Design Patterns Applied").

We named the host class *AgileDijkstra*. It requires policy that defines the type of nodes storage. We implement three such policies: vector based, set based and priority queue based.

Performance test:
nodes count: 500000
edges count: 1000000
Set policy: Duration: 450.312ms
Vector policy: Duration: 30692.5ms
Queue policy: Duration: 30776.4ms