Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

17rd November 2016

Algorithm & Data Structures — Solutions to Sheet P8 — AS 16

Solution P8.1 *Wind Turbines.*

We will use dynamic programming for this task: for every $i \in \{0, \ldots, n\}$, let m_i be the best solution only using the positions $\{d_1, \ldots, d_i\}$, so $m_0 = 0$ and we are interested in m_n.

Now there are two ways the best solution on positions $\{d_1, \ldots, d_i\}$ may look: either it uses the turbine at d_i or not. If it does use the turbine at d_i, then the value is e_i plus the best solution from turbines before d_i far enough from d_i, which is captured by some m_z. If it does not use the turbine at d_i, then we have just $m_i = m_{i-1}$. Taken together, we have

$$m_i = \max\{e_i + m_z, m_{i-1}\},$$

where z is maximal such that $d_z \leq d_i - D$ (or 0 if already $d_1 > d_i - D$). Now we can compute the values m_i from m_1 to m_n and we just have to find the right value of z in every step.

One way to find z is to look at all values $0, \ldots, i-1$ in every step, comparing d_z to $d_i - D$, but that would give us an $\mathcal{O}(n^2)$ time algorithm, too slow for the tests. We can improve this with a binary search on $d_1, \ldots, i-1$ to find the last value before $d_i - D$ (or 0 if there isn't such), and get an $\mathcal{O}(n \log n)$ time solution.

However, the fastest way to find z happens to be also the simplest one: Observe that the value of z for step i has to be at least the value of z for step $i-1$. So we can remember the old value of z we used in the step computing m_{i-1} and in the step computing m_i we increase this z by 1 in a `while` loop until we hit $d_{z+1} > d_i - D$.

This gives an $\mathcal{O}(n)$ time solution to the problem: It may happen that we need to increase z many times (checking $d_{z+1} > d_i - D$ every time) in a single step from d_{i-1} to d_i but since z starts at 0 and never goes above $n-1$, we can only do this at most n times during the whole algorithm, so the runtime consumed by this `while` loop is $\mathcal{O}(n)$ during the whole program.

Solution programs

On the lecture website, you can find a solution running in time $\mathcal{O}(n)$ that uses the last approach. The solution source contains further comments on the implementation.

There is also a second solution illustrating a different dynamic programming approach: let t_i be the best solution from turbines at $d_1, \ldots, d_i$ that *does* use d_i. When computing t_i, we let $t_i = e_i + \max\{0, t_1, t_2, \ldots, t_z\}$ where z is as above. The presented solution looks at all such t_j every step, obtaining an $\mathcal{O}(n^2)$ time solution. However, even in this case this could be easily sped up to $O(n)$ by remembering not only the last z, but also the previous maximum of $\{0, t_1, t_2, \ldots, t_z\}$ and updating it along with increasing z. We leave the details to the reader.

Data

Every judge test also contained 1-3 small additional special cases, e.g. with only 1 position, with all positions too close (so you can only build one), and with all the positions far enough (so you can build all of them).

`judge1` $n = 100$, $D = 1\,000\,000$, all positions in conflict.

`judge2` $n = 100$, $D = 1$, no positions in conflict.

`judge3` $n = 100$, $D = 200\,000$, every position in conflict with ca 2/5 of others.

`judge4` $n = 1\,000$, $D = 50\,000$, random positions from $0 \ldots 1\,000\,000$.

`judge5` $n = 100\,000$, $D = 5\,000$, random positions from $0 \ldots 1\,000\,000$. A quadratic program should fail this.