Greedy algorithms

Rossano Venturini[1]

1 **Computer Science Department, University of Pisa, Italy**
rossano.venturini@unipi.it

Notes for the course *"Competitive Programming and Contests"* at Department of Computer Science, University of Pisa.
Web page: `https://github.com/rossanoventurini/CompetitiveProgramming`

These notes sketch the content of the 13th lecture.

1 Problems

▶ **Problem 1** (Lexicographically Maximum Subsequence)**.** *You've got string $s[1, n]$, consisting of only lowercase English letters. Find its lexicographically maximum subsequence.*

As an example, consider the string $s =$ `abbcbccacbbcbaaba`. Its lexicographically maximum subsequence is `cccccbba`.

Scan the string from right to left. Every time you find the current maximum, emit it.

▶ **Problem 2** (Woodcutters)**.** *There are n trees located along the road at points with coordinates $x_1, x_2, \ldots, x_n$. Each tree has its height h_i. Woodcutters can cut down a tree and fell it to the left or to the right. After that it occupies one of the segments $[x_i - h_i, x_i]$ or $[x_i, x_i + h_i]$. The tree that is not cut down occupies a single point with coordinate x_i.*

Woodcutters can fell a tree if the segment to be occupied by the fallen tree doesn't contain any occupied point. The woodcutters want to process as many trees as possible, so Susie wonders, what is the maximum number of trees to fell.

As an example, consider $x = 1, 2, 5, 10, 19$ and $h = 2, 1, 10, 9, 1$. We can cut up to 3 trees.

The problem can be solved with the following greedy algorithm. Let's fell leftmost tree to the left (it always doesn't make an answer worse). After that, try to fell the next tree. If we can fell it to the left, let's do it (because it also always doesn't make an answer worse). If we can't, then try to fell it to the right. If it is possible, let's do it. Last step is correct because felling some tree to the right may only prevent the next tree's fallen. So we may "exchange" one tree to another without worsing an answer.

▶ **Problem 3** (Queue)**.** *In the Main Berland Bank n people stand in a queue at the cashier, everyone knows his/her height h_i, and the heights of the other people in the queue. Each of them keeps in mind number a_i — how many people who are taller than him/her and stand in queue in front of him.*

After a while the cashier has a lunch break and the people in the queue seat on the chairs in the waiting room in a random order.

When the lunch break was over, it turned out that nobody can remember the exact order of the people in the queue, but everyone remembers his number a_i.

Your task is to restore the order in which the people stood in the queue if it is possible. There may be several acceptable orders, but you need to find any of them. Also, you need to print a possible set of numbers h_i — the heights of people in the queue, so that the numbers a_i are correct.

As an example consider the following sequence of names and as; $a0, b2, c0, d0$. A possible solution is the following: $a150, c170, d180, b160$
Let's sort the names by ascending of a_i. If there is an index i, $1 \leq i \leq n$, such that $a_i > i$, then answer is -1. Otherwise the answer exists.

Now the problem is that of generating valid heights. This is done by obtaining persons ordered by height. Person n goes in position a_n. Person a_{n-1} goes in position a_{n-1}, if $a_{n-1} < a_n$, or $a_{n-1} + 1$ otherwise. In general, person i goes in position $a_i + k$, where k is the number of values in $a_{i+1}, \ldots, a_n$ which are smaller than or equal to a_i.

Indeed, sequence $a_1, \ldots, a_n$ can be see as the inversion vector of the sorted permutation.

Values k can be computed with a BIT. Thus, the algorithm runs in $\Theta(n \log n)$ time.