Solution to task F/2006

Author: Wojciech Kazana

DESCRIPTION AND SOLUTION

Our solution consists of checking all the possible placements of shafts and gears amongst which the solution must exist.

From the text of the task one gets that the optimal solution has to consist of two sequences of gears which share a common prefix (it's length can be equal to zero) and then they have no more common gears. So our sequences can be described as AX and AY, where X and Y are different and every element of A, X and Y is a name of one gear or a concatenation of two such names. First step of our algorithm is to compute every possible way of describing N (the number of gears - up to 6) as a sum of numbers 1 and 2. From now on let's name such a sum a pattern (for example if N is 6 then 1+2+2+1 is a pattern). We can write it as A'X'Y'Z' where A', X', Y', Z' describe the patterns of A,X,Y,Z (Z are the gears that weren't used and A,X and Y are as shown in the beginning). Now for every permutation π of gears and for every pattern A'X'Y'Z' we claim that π is described as AXYZ. We consider all patterns and every single permutation, so the optimal solution (of course if it exists) must be somewhere amongst them.

For every case we first check whether AX actually describes the minute hand (that the combination of gears in AX allows us to obtain 24 turns a day in a clockwise direction) and AY the hour hand (2 turns a day). If not - we move to another case. If it does, we check if this particular solution is better than the best one so far and if it is, we swap the previously best solution with this one.

In the end we just print the solution onto the output in a way described in the task. In particular, there can be three gears on the last common shaft and there is no mistake in the procedure shown above - it just means that the last element of A, together with the first element of X and the first element of Y, are all situated on the same shaft.

COMPLEXITY ANALISYS

The limitations in the task are very small. Actually ($N \leq 6$) so the number of cases that are being checked is also very small, even though we check not only every permutation and pattern, but also every splitting of a particular pattern into A,X,Y and Z described above. Counting it very roughly we get: at most 6! permutations, only 13 patterns in the worst case of 6 gears, at most $\binom{6}{3} = 20$ splits of a pattern (of length 6), so we have only 187200 cases to check. Each of them is being checked linearly in time of N (we have to count the speed of every shaft in a pattern). Moreover, we use a heuristic (described below), that sometimes allows us to check just a few splits for a certain pattern and permutation.

SOME TECHNICAL INFORMATION

In our implementation we use three different functions to find every proper solution.
All the splits where $Y = \epsilon$ are being checked by function `probuj_liniowo(vector<string> &)`.
And now we use a very important heuristic: if the permutation and a pattern that we are checking does not provide a shaft moving with the speed of neither minute hand nor hour hand, we skip further computations concerning this certain permutation and pattern (it is

pretty obvoius that splits with $Y \neq \epsilon$ cannot give us anything more). That gives us a possibility to decrease the time the programme works a few times.
Next for each pattern (as long as the heuristic does not say us to stop) we use:
`probuj_luzno(vector<string> &, int, int, int)` and
`probuj_wcisnac(vector<string> &, int, int, int)`.
First function checks the possibility of finding a solution when a single gears can cause up to one other gear to move. The other is responsible for situations when there actually is a gear accelerating exactly two other gears (placed on two separate shafts).

To generate every permutation in C++ we use library function next_permutation. In Java we had to implement it ourselves.