Task proposal from Poland: Sequence Construction

Idea authors: Łukasz Kalinowski[*], Marcin Kubica[†]
Scribes: Wojciech Nadara[‡], Marek Sokołowski[§]

February 1, 2023

Statement

Let us call a sequence of positive integers $x_1, \ldots, x_m$ *good* if $x_1 = 1$ and for each $2 \le i \le m$ we either have $x_i = x_{i-1} + 1$ or $x_i = x_j \cdot x_k$ for some $1 \le j, k \le i - 1$.

You are given an integer n $(1 \le n \le 30\,000)$ and a sequence of positive integers $w_1, \ldots, w_n$ $(1 \le w_i \le 10^6)$. We define a *weight* of a sequence $x_1, \ldots, x_k$ as a $w_{x_1} + \ldots + w_{x_k}$. For each $1 \le i \le n$, determine the smallest weight of a good sequence containing i.

Solution

It looks like a textbook exponential backtracking problem, but it surprisingly has a polynomial solution! In fact, the solution will still be backtracking, but its state space will be of almost linear size.

We will design a recursive function that will solve this problem not only for sequences containing some specific integer, but a set of them. Let $F(\{a_1, \ldots, a_k\})$ be the smallest weight of a sequence containing all of $a_1, \ldots, a_k$ and assume that this sequence is increasing. We are interested in computing $F(\{1\}), F(\{2\}), \ldots, F(\{n\})$.

We have a trivial relation: $F(\{1\}) = w_1$. If we denote $A = \{a_1, \ldots, a_k\}$ and $A \neq \{1\}$, then we have

$$F(A) = w_{a_k} + \min\left(F(A \cup \{a_k - 1\} \setminus \{a_k\}), \min_{i \cdot j = a_k,\, 1 < i,j < a_k} F(A \cup \{i, j\} \setminus \{a_k\})\right).$$

As a_k is the maximum element of A and all elements we added are strictly smaller, this recursion terminates.

Now, consider a measure of a set of positive integers equal to the product of the integers in the set. Crucially, in the recursive relation above, the measure of A is greater than or equal to the measures of all sets on the right-hand side of the equation. Since each of initial sets $\{1\}, \{2\}, \ldots, \{n\}$ has its product trivially bounded by n, this recursive algorithm will only ever consider sets A such that $a_1 \cdot a_2 \cdot \ldots a_k \le n$ and $1 < a_1 < a_2 < \ldots < a_k$. Intuitively, that is a very restrictive condition and there will not be many states to visit.

We can leverage that in a clear way by memoizing results for all reachable states. Let us denote the number of such sequences as $C(n)$, the transitions that are required to be considered as $T(n)$ and the upper bound on the time complexity of performing one transition as $p(n)$. Then the total complexity of this solution will be $O(T(n) \cdot p(n))$. Bounding the $C(n)$ and $T(n)$ is done best empirically. A very simple program (that is actually kind of a simple version of the model solution) shows that for $n = 30\,000$ we have $C(n) = 394\,099$ and $T(n) = 1\,428\,839$. For the memoization we can use a simple map of sets of integers (mapping each visited set A to the value $F(A)$). The lookup in this map takes $O(\log C(n))$ operations, where each operation compares two sets of integers. As shown empirically, $C(n)$ is not much bigger than n for the considered limitations, hence we can crudely bound that as $O(\log n)$ (precise proofs will appear later). These comparisons are expected to be fairly quick as they will not be longer than sizes of these sets. As $n \ge a_1 \cdot \ldots \cdot a_k$ and $2 \le a_1 < \ldots < a_k$ we have $n \ge a_1 \cdot \ldots \cdot a_k \ge (k+1)!$, hence $k = O(\frac{\log n}{\log \log n})$, or simply $k \le 7$ for $n \le 30000$. In the result, we can express the complexity as $T(n) \cdot \log n \cdot k$, which for presented bounds for $T(n)$ and k is quite reasonable.

[*]University of Warsaw, Poland.
[†]University of Warsaw, Poland. Polish Olympiad in Informatics main committee member.
[‡]`w.nadara@mimuw.edu.pl`, University of Warsaw, Poland. Polish Olympiad in Informatics task committee member.
[§]`marek.sokolowski@mimuw.edu.pl`, University of Warsaw, Poland. Polish Olympiad in Informatics task committee member.

Speed-ups

In the presented solution, we used a map of sets of integers, e.g., `std::map<std::set<int>, int>` in C++. This comes with a lot of runtime overhead, so we may consider speeding this solution up by optimizing the memoization part. A good idea is to implement a hash function mapping sets A to "random" integers (say of type `long long`); this way, we can use a more straightforward map of integers (e.g., `std::map<long long, int>`). This optimization straight away eliminates the need to compare the sets of integers within the container. Polynomial hashing (i.e. $h(A) = \sum_{a \in A} c^a \mod M$ for some c and M) seems like a good idea as we can quickly compute hashes of sets to which we are recursing. Note that M needs to be sufficiently large in order to avoid the hash collisions coming from the birthday paradox; say, one should expect some collisions for $M \approx 10^9$. However, for instance, $M = 2^{64}$ should be more than enough for our needs.

Subtasks and scoring

Ideally, we would like to distinguish the optimized versions (the one with hashes) of the solution from the solutions without them. In order to do this, the upper limit on n may need to be increased or the time limit might need to be adjusted carefully as the difference between them is not too big. Though, even when distinguished, the difference in points should be rather small – optimizations like these are not the crux of the problem.

The natural idea for partial solutions are exponential backtracks that try to generate this sequence from the smallest to biggest integers. However in order for them to be reasonable, the lengths of optimal sequences would need to be short. That is the case when all weights are unit (or maybe a bit more generally, when the ratio of the biggest to the smallest weight is very small, e.g. at most 1.05). Then, when properly optimized, such algorithms are able to compute solutions to values of n that are up to a few hundred. Assuming that the weights are unitary ($w_i = 1$), the contestants can preprocess all results on their own computer and paste the results into the source code sent to the judge. We found experimentally that the smallest n with unreasonable running time is $n = 1439$ – it is the smallest number that requires a sequence of length 15. A decently optimized small-to-large backtracking can find the results for smaller values of n within about 10 minutes.

Here are our suggestions regarding the subtasks:

1. $n \leq 16$ – 10 points (here, we intend brute-force $2^n \cdot \mathrm{poly}(n)$ solutions: iterate all possible increasing sequences of integers $\leq n$ and check if they can be constructed);
2. $w_i = 1, n \leq 300$ – 20 points;
3. $w_i = 1, n \leq 1400$ – 10 points;
4. $n \leq 10\,000$ – 45 points;
5. $n \leq 30\,000$ – 15 points.

The subtasks may be slightly changed, e.g., to increase the ratio between bounds in the last two subtasks to distinguish optimizing versions better, or some unit weight constraints may be replaced with $100 \leq w_i \leq 105$ if we are less welcoming of preprocessing solutions (but then it might be the case that the third subtask cannot be solved with suboptimal solutions).

Bonus: Complexity analysis

As a bonus, we present a more rigorous complexity analysis of the presented algorithm. Note that this analysis **is not required** to solve the problem; it is enough to verify the upper bound on the running time experimentally. Also, the proof below requires a heavy use of real analysis, so it might not be suitable for the presentation to the contestants. However, at the end we also provide a much simpler proof that shows why this solution is polynomial, which should be accessible to contestants, but gives a worse bound.

Our main goal is, given an integer $n \geq 1$, to produce an upper bound on the number $C(n)$, which was the number of sets of integers such that:

- each integer is larger than or equal to 2,
- the product of all integers in the set is at most n.

We will show that the number of such sets is bounded from above by $n \cdot e^{2\sqrt{\log n}}$. Note that this gives us a justification to write $O(\log C(n)) = O(\log n)$. Moreover, the indegree of each state in the transition graph can be easily bounded by k^2, where k is the length of the sequence associated with its state, hence

$T(n) \leq C(n) \cdot (\frac{\log n}{\log \log n})^2$. And, as already argued $p(n)$ can be bounded by $\log n \cdot k$, so the total time complexity can be bounded by $n \cdot e^{2\sqrt{\log n}} \cdot \frac{\log^4 n}{\log^3 \log n}$. If we use hashing, then one k factor may be shaved off at the cost of introducing randomness. Moreover, we expect that if we bound outdegree instead of indegree of each state, then $T(n) \leq C(n)k^2$ can be improved to $T(n) \leq C(n) \log n$ as the average number of divisors of a number that is at most n is $\Theta(\log n)$, but showing the full argument based on this idea would require a bit more care in the details that we did not delve into. The resulting bound is quite remarkable: it grows faster than any almost linear function of the form $n \log^c n$ for $c > 0$, yet it grows slower than any function of the form $n^{1+\varepsilon}$ for $\varepsilon > 0$. In literature, such complexities are usually denoted as $n^{1+o(1)}$.

To achieve the promised upper bound, let us produce a slightly different upper bound: given integers $n \geq 1$ and $k \geq 0$, let us upper bound the number of **sequences** (not necessarily increasing) of integers such that:

- each integer is larger than or equal to 2,
- the length of the sequence is exactly k,
- the product of all integers in the sequence is at most n.

Let $f_k : \mathbb{R}_{>0} \to \mathbb{R}_{>0}$ be a real function with the following property: f_k is non-decreasing and for each integer $n \geq 1$, the number of sequences of integers of length k with product at most n is bounded from above by $f_k(n)$. Note that we can set $f_1(x) = x$.

Next, consider sequences of length $k \geq 2$. Fix the last integer $a_k \geq 2$ of the sequence. Then, $(a_1, \ldots, a_{k-1})$ is a sequence of length $k-1$ and product at most $\left\lfloor \frac{n}{a_k} \right\rfloor$. The number of such sequences is upper bounded by $f_{k-1}\left(\frac{n}{a_k}\right)$. Iterating over all possible values of a_k, we find that the number of sequences of length k and product at most n is at most

$$\sum_{i=2}^{n} f_{k-1}\left(\frac{n}{i}\right). \tag{1}$$

Since f_{k-1} is non-decreasing, we can upper-bound (1) by

$$\int_1^n f_{k-1}\left(\frac{n}{t}\right) \mathrm{d}t.$$

Therefore, we can set $f_k(x) = \int_1^x f_{k-1}\left(\frac{x}{t}\right) \mathrm{d}t$ as the real-valued upper bound on the number of sequences of length k and product at most n.

We can prove inductively that for each $k \geq 1$, we have

$$f_k(x) = \frac{1}{(k-1)!} x \log^{k-1} x.$$

In other words, the number of sequences of length k and product at most n is bounded from above by $\frac{1}{(k-1)!} n \log^{k-1} n$. Note that here, log is assumed to be the natural logarithm.

Now, observe that if A is an upper bound on the number of *sequences* of length k and product at most n, then $A/k!$ is an upper bound on the number of *sets* of different integers of length k and product at most n. (This is because each set of k different integers has $k!$ different permutations, each counting towards the number of sequences of length k.) Therefore, the number of *sets* of size k and product at most n is bounded from above by

$$\frac{1}{(k-1)!k!} n \log^{k-1} n. \tag{2}$$

Summing (2) over all possible lengths $k \geq 1$, we find that the number of sets of different integers ≥ 2 and product at most n is at most

$$\sum_{k \geq 1} \frac{1}{(k-1)!k!} n \log^{k-1} n. \tag{3}$$

Now,

$$(3) \leq n \sum_{k \geq 1} \frac{\log^{k-1} n}{[(k-1)!]^2} = n \sum_{k \geq 0} \frac{\log^k n}{(k!)^2} \overset{(\star)}{\leq} n \left(\sum_{k \geq 0} \frac{\log^{k/2} n}{k!} \right)^2 = n \left(\sum_{k \geq 0} \frac{(\sqrt{\log n})^k}{k!} \right)^2$$

Note that $(\star)$ follows from the fact that $(a_1 + a_2 + a_3 + \cdots + a_\ell)^2 \geq a_1^2 + a_2^2 + a_3^2 + \cdots + a_\ell^2$ holds for non-negative real values $a_1, a_2, \ldots, a_\ell \geq 0$. Here, we set $a_i = \frac{\log^{i/2} n}{i!}$.

Now, recall the Taylor series for the function e^x:

$$e^x = \sum_{k \geq 0} \frac{x^k}{k!}.$$

By setting $x = \sqrt{\log n}$, we get

$$(3) \leq n \cdot \left(e^{\sqrt{\log n}}\right)^2 = n \cdot e^{2\sqrt{\log n}}.$$

Therefore, the number of different sets of integers, each greater than or equal to 2, such that their product does not exceed n, is upper bounded by $n \cdot e^{2\sqrt{\log n}}$.

Easier, but worse bound

Let $R(n)$ denote the number of sequences of integers that are at least two, whose product is at most n (so, the $\sum_k f_k(n)$ from the previous proof). It is clear that $C(n) \leq R(n)$. As each such sequence has some last element, we can get a bound $R(n) \leq R(\frac{n}{2}) + R(\frac{n}{3}) + \ldots + R(\frac{n}{n})$ (let us ignore needed floors). From that we can claim an inductive hypothesis that $R(n) \leq n^c$ for some c. $c = 2$ can be clearly verified to go through the induction proof (as $\frac{\pi^2}{6} - 1) < 1$, and actually the best c that works is the c such that $\zeta(c) = 2$,[1] so $c = 1.73$ will work as well.

[1]Here, ζ is the Riemann zeta function, defined as $\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{x^n}$ for $x > 1$.