Analysis: MAG
Magical stones

HISTORY:
• v. 2.00: 2008.27.03, JR, translation into English
• v. 1.00: 2006.12.03, AN, task analysis

dokument systemu SINOL 1.6

1 Model solution

Solution of this task consists of finding the i^{th} in the lexicographical order word composed of n digits 0 (`I`) and 1 (`X`) satisfying two conditions:

- (a) numbers 0 and 1 are next to each other at at most k positions,
- (b) the word read the other way around (from right to left) is not lexicographically less than the original word.

From now on we will call each word satisfying the above conditions a `correct` word.

Let us introduce some notations. $\sigma(a_1 \ldots a_m)$ denotes the number of correct words that start with the prefix $a_1 \ldots a_m$ and $\sigma(a_1 \ldots a_m, b_{m'} \ldots b_1)$ — the number of correct words that start with the prefix $a_1 \ldots a_m$ and end with the suffix $b_{m'} \ldots b_1$ (if $m+m' > n$ then the prefix and suffix intersect).
We will be finding the symbols of the i^{th} word one by one, from left to right. We start with a task of finding the i^{th} in the lexicographical order correct word starting with the prefix ε.

In order to find the j^{th} in the lexicographical order correct word starting with the prefix $a_1 \ldots a_m$, we will compute $\sigma(a_1 \ldots a_m 0)$. $\sigma(a_1 \ldots a_m 0) \geq j$ then the requested word is the j^{th} correct word starting with the prefix $a_1 \ldots a_m 0$. In the opposite case the requested word is the $(j - \sigma(a_1 \ldots a_m 0))^{th}$ correct word starting with the prefix $a_1 \ldots a_m 1$.

Now we show how to compute values $\sigma(a_1 \ldots a_m)$. Each correct word that starts with the prefix $a_1 \ldots a_m$ is of exactly one of the following forms:

- (1) l final letters of the word (where $0 \leq l < m$) read from right to left are exactly the same as l starting letters of the word, 0 is the $(l+1)^{st}$ letter from the beginning of the word and 1 is the $(l+1)^{st}$ letter from the end of the word (in other words, the considered word is of the form $a_1 \ldots a_l 0 a_{l+2} \ldots a_m \ldots 1 a_l \ldots a_1$).
- (2) m final letters of the word read from right to left are the same as m starting letters of the word.

This leads to the following formula:

$$\sigma(a_1 \ldots a_m) = \sum_{l=0}^{m-1} ([a_{l+1} = 0] \cdot \sigma(a_1 \ldots a_m, 1 a_l \ldots a_1)) + \sigma(a_1 \ldots a_m, a_m \ldots a_1).$$

Let us notice that we will be only computing values of σ with $a_m = 0$.
Let us assume that in the word $a_1 \ldots a_m$ letters 0 and 1 are next to each other at k_m positions, whereas in the word $1 a_l \ldots a_1$ — at k_l positions.

We now show that if $2m \leq n$ then $\sigma(a_1 \ldots a_m, a_m \ldots a_1) = \frac{1}{2} \cdot (\sum_{i=0}^{\frac{k-2k_m}{2}} \binom{n-2m+1}{2i} + \sum_{i=0}^{\frac{k-2k_m}{2}} \binom{n-2m+1}{i})$: The number of words of the form $a_1 \ldots a_m \ldots a_m \ldots a_1$ that satisfy condition (a) is $\sum_{i=0}^{\frac{k-2k_m}{2}} \binom{n-2m+1}{2i}$ — there are

$n-2m$ free positions in the word, so a change of letter ($0 \to 1$ or $1 \to 0$) could take place at any of $n-2m+1$ positions; we also know that the number of letter changes is even. The number of words of the form $a_1 \ldots a_m \ldots a_m \ldots a_1$ that satisfy condition (a) and are palindromes is $\sum_{i=0}^{\frac{k-2k_m}{2}} \left(\binom{\frac{n-2m+1}{2}}{i}\right)$ — half of the word can be filled arbitrarily, using no more than $\frac{k-2k_m}{2}$ letter changes, and the remaining part of the word is uniquely determined. Only a half of words that are not palindromes satisfies condition (b), so the above formula holds.

Similarly, if $m+l+1 \leq n$ and $a_l+1=0, a_m=0$ then $\sigma(a_1 \ldots a_m, 1 a_l \ldots a_1) = \sum_{i=0}^{\frac{k-k_l-1}{2}} \binom{n-m-l}{2i+1}$ — we can choose the places of letter changes from $n-m-l$ positions, taking care that the total number of letter changes is odd (bacause $a_m=0$) and is not greater than $k-k_l$.

If the prefix and the suffix intersect (condition $2m>n$ or $m+l+1>n$ holds) then the number of correct words with the desired prefix or suffix is equal to 0 or 1 and this number can be computed in a straightforward manner.

An improvement of complexity of the algorithm can be achieved by keeping partial sums $\sum_{i=0}^{j} \binom{c}{2i}$ and $\sum_{i=0}^{j} \binom{c}{2i+1}$ in an auxiliary array. Computation of the numbers of neighbouring zeroes and ones for changing prefixes and suffixes can be performed in constant time. In conclusion, time complexity of the whole algorithm is $O(n^2)$ (values of the form $\sigma(a_1 \ldots a_m)$ are computed n times and the time complexity of computing one such value is linear in terms of n).

A last remark should be done about the case when $\sigma(\varepsilon) < i$ — in this case the desired word does not exist. Such situation can be easily recognized — after performing n steps of the algorithm a word consisting of n ones would be achieved and the index j of the requested word with that prefix would be greater than 1.

2 Other solutions

Files `prog/mags0.cpp` and `prog/mags2.pas` contain implementations of the simplest and most straightforward solution of this task. This solution consists of generating words of length n consisting only of letters 0 and 1 and checking, for each one of them, if it is a correct word. The i^{th} correct word found is the output of the algorithm. The time complexity of this solution is $O(n \cdot i)$.

Files `prog/mags1.cpp` and `prog/mags3.pas` contain an improved version of the previous solution in which the process of verification of correctness of a single word has amortized $O(1)$ time complexity, so the total time complexity of the algorithm is $O(i)$.

3 Incorrect solutions

File `prog/magb0.cpp` contains an implementation of an algorithm in which the bound on the number of neighbouring zeroes and ones in Xi-*n*-*k* stones is ignored. Time complexity of this solution is $O(n^2)$, which is the same as the complexity of the model solution. This solution gives correct results in case if the value of parameter k is quite large in terms of the length of the words n (this solution is always correct if $k=n-1$) and for small values of parameter i.

File `prog/magb1.cpp` contains an implementation of an algorithm that does not take into consideration the condition that each correct word must not be greater lexicographically than its upside-down equivalent. This solution gives correct results if i is sufficiently small in terms of n.

File `prog/magb2.cpp` contains a programme that simply outputs the i^{th} word of length n in the lexicographical order.

The algorithm implemented in the file `prog/magb3.cpp` is very similar to the model solution, but it uses the formula $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$ to compute the binomial coefficients. For values of n about 20 or greater this causes an overflow of 64-bit integer type.

4 Limits

The limits on the input data (in particular, the limit $n \leq 60$) are such that all computations in the model solutions can be performed on 64-bit integers, so there is no need to implement large integer arithmetical operations.

The choice of limits enables differentiation of solutions with time complexity $O(n^2)$ or $O(n^3)$ from $O(i)$ or $O(n \cdot i)$ solutions. However, differentiating solutions with complexities $O(n^2)$ and $O(n^3)$ is impossible, but this is not a problem because solutions with complexity $O(n^3)$ are not significantly easier than $O(n^2)$ solutions.

5 Tests

4 tests for the contestants and 15 groups of tests for the contest were prepared.

- `mag1ocen.IN` (ε sek.) $n = 6, k = 4$ — easy test for the contestants
- `mag2ocen.IN` (ε sek.) $n = 10, k = 3$ — easy test for the contestants
- `mag3ocen.IN` (ε sek.) $n = 12, k = 11$ — easy test for the contestants with answer `NO SUCH STONE`
- `mag4ocen.IN` (ε sek.) $n = 20, k = 18$ — easy test for the contestants
- `mag1.IN` (ε sek.) $n = 12, k = 7$ — simple test for correctness
- `mag2a.IN` (ε sek.) $k = 0$ — simple test for correctness
- `mag2b.IN` (ε sek.) $n = 14, k = 7$ — simple test for correctness
- `mag3a.IN` (ε sek.) $n = 10, k = 9$ — simple test for correctness with answer `NO SUCH STONE`
- `mag3b.IN` (ε sek.) $n = 18, k = 15$ —simple test for correctness
- `mag4.IN` (ε sek.) $n = 20, k = 8$ — simple test for correctness
- `mag5.IN` (ε sek.) $n = 25, k = 13$ — simple test for correctness
- `mag6.IN` (ε sek.) $n = 30, k = 27$
- `mag7.IN` (ε sek.) $n = 45, k = 30$
- `mag8.IN` (ε sek.) $n = 40, k = 35$
- `mag9.IN` (ε sek.) $n = 47, k = 20$
- `mag10.IN` (ε sek.) $n = 50, k = 40$
- `mag11.IN` (ε sek.) $n = 55, k = 35$
- `mag12.IN` (ε sek.) $n = 55, k = 49$
- `mag13.IN` (ε sek.) $n = 58, k = 47$
- `mag14.IN` (ε sek.) $n = 60, k = 49$
- `mag15a.IN` (ε sek.) $n = 60, k = 57$

- `mag15b.IN` (ε sek.) $n = 60, k = 57$ — a query for the first stone in lexicographical order

The solution with time complexity $O(n \cdot i)$ passes first four tests and the solution with time complexity $O(i)$ — first five tests.

Some groups of tests were created so that programmes that always output `NO SUCH STONE` and programs that ignore one of the conditions of stone's `correctness` do not score any points.

The times were measured on a computer with CPU 1.7 GHz.