Monotonic approximation

25.07.2009 - James Radoszewski

Probably my favorite task of the competition is the job for 500 in the fourth, the last remote round TopCoder Open 2006. Probably because as a result of several interesting coincidences that task turned out to have an unexpected impact on the history of my competing in programming contests, but not only. But let's start from the beginning.

Mileage round

Here comes the final round of the competition elimination remote TopCoder Open 2006, yet only this round, and we're going to Las Vegas. And all would be well (even relatively large players qualify, because 48), were it not that the round is about three in the morning Polish time. Diks professor says that the great masters win in all conditions. I, unfortunately, means to this group not one, because in the middle of the night thinking it's definitely the last thing that I want to. Because it was the third round in a row with such a horrible time, all Polish members intensely mobilized to start.

Round started without any major surprises, just unpleasant task for the 250th koderskie Stairs started on. Open job for 500, has nice short text, which can be paraphrased as follows:

Given a sequence of numbers dummy . You should find a real-decreasing sequence for which the expression:$a_1, a_2, \ldots, a_n \ (n \leq 2500)$ $b_1, b_2, \ldots, b_n$

$$w_{a,b} = (a_1 - b_1)^2 + (a_2 - b_2)^2 + \ldots + (a_n - b_n)^2$$

the smallest value and return $w_{a,b}$.

Some simple observations suggest themselves. For over *a* form: however many zeros, however many zeros, the search string $b$is of course the same string. But for the form: $j$ones, followed by $z$zeros, the string $b$must be constant. Indeed, increasing the value of the first $j$of the sequence b, and reducing the other (of course within the limits $[0, 1]$) will certainly reduce the value $w_{a,b}$, and in this way, certainly at the end of any sequence will get $b$to a constant. The value of its elements $x$should minimize the expression:

$$w_{a,b} = j * (x - 1)^2 + z * x^2$$

Using the fact that the above quadratic polynomial takes place at least in a zero derivative thereof:

$$2j * (x - 1) + 2z * x = 0,$$

get the correct value $x$is $\frac{j}{j+z}$, the arithmetic mean of all the words a. And then what?

Unfortunately, at this point ideas on observations I completely ran out: not really see how to deal with any string a. A look at the ranking of occupations suggests that not only do I have this problem: just a few players coped with the task pretty quickly (the fastest took about 12 minutes), but the rest does not go so smoothly: evenly for most of the contest every minute or two someone goes to send the solution (I might add that the average time for a correct application of this task was anything over 36 minutes) - clearly a task requires a clever idea. Open in desperation task for 1000 does not look like one that would have a chance to make it solved relatively slow solution to the problem written for 250 most likely not be promoted, and there's been an hour event passes (for the uninitiated: round lasts 75 minutes), and ideas for 500 -tuple not - and I mention that it's the middle of the night?

At that point, we can only try to guess. The only good idea the way this seems to be the arithmetic mean. So let's start searching for $b$the same string a. If it is non-decreasing, then we can stop. If not, opt for a fragment consisting of the following ones, followed by a string of zeros, and replace these two parts of one whose expression is the arithmetic mean of all those ones and zeros. And now we can repeat the procedure over and over again: if the adjacent two permanent parts of the string, the first of which is greater than the second term, then replace the two parts of one whose word is being replaced by the arithmetic mean of all the words. It is not clear whether using such operations always get the optimal sequence b, but there is no longer time to charge. A few minutes fast encoding and solution set. Checking looking for solid sample tests - works. I'm sending a few minutes before the end, and pending the results of spending some of the more nervous 30 minutes of life.

Why does it work?

Luckily for me, it turned out that the described solution was correct and thus, like the record number of Poles, I managed to qualify for the first time in the life of the semi-finals of the competition TCO. However, to complete the description of the guessing, I will present in brief justification for the correctness of the above solutions.

Introduce basic terminology. For the string a, search-decreasing sequence $b$called *the best approximation* (ie, the best approximation), while the value of the $w_{a,b}$call *error of approximation* . Let's start with pseudo-solutions described above. We use it in the list `L` , which represents the current form of the transformed string (representing the start of a list of string a). List `L` contains pairs of the form: (`x` - an element that is the best approximation substring a, `number` - the number of elements in this passage). Since this is only a pseudo, we can afford it some fraud: we assume that we can list more items index numbers from `1` to `L.size ()` , just as in the case of the array (that always use the following indexes only).

```
1   for and =  1 to n to
2     L. insert ( and ( and ) , 1 ) ,
3   and =  1 ,
4   while  ( and < L size ( ) )  {
5     if  ( L [ and ] . x  > L [ and +  1 ] . x )  {
6       L [ and ] . x  = mean ( L [ and ] . x , L [ and ] . amount of Sc, L [ and +  1 ] . x , L [ and +  1 ] . amounts OF ) ,
7       L [ and ] . amount TY + = L [ and +  1 ] . amount FIELD ,
8       L. erase ( L [ and +  1 ] ) ,
9       if  ( and >  1 )  - and ;
10    }  else  + + and ;
11  }
12  return L ;
```

If the condition `L [i]. x> L [i + 1]. x` occurs, according to an informal description of the earlier we make the additional parts of the merge, the resultant fragment assigning a value that is the arithmetic mean of all his words. Responsible for this function `mean` that returns the result of the following formula:

$$\frac{L[i].x * L[i].ilość + L[i+1].x * L[i+1].ilość}{L[i].ilość + L[i+1].ilość}$$

Justification correctness of this formula is immediate, if noted that the box `x` represents the mean of the elements in $i$the passage-and, therefore, expression of the form `L [i]. x * L [i]. amount` is the sum of the elements of $i$the ith substring. What's more, you can see that in the resulting list `L` of `L [i]. x` are sorted in non-decreasing (take care of include the line " `if (i> 1) - i;` "), which represent a valid approximation decreasing within a.

The time complexity of this solution is $\mathcal{O}(n)$. To justify this, just look at the parameter $W = 2 * L.size() - i$. Before the first rotation loop **while** $W = 2n - 1$, and each time around the loop, this parameter is reduced by at least one and is always positive. This shows that the restriction $n \leq 2500$of competition is really very mild.

We already know that our algorithm is very effective and it returns a result string-decreasing. It remains the most difficult question: to justify that it is *the best approximation* over a. To do this, you need to show three things (from the first two shows that the resulting string in the algorithm is the best approximation over the pieces a, and the third, that is the best approximation to the string):

1. If the best approximation of one portion of the string $a$is a constant string of equal parts x, and the best approximation of the second - a sequence of elements equal to a constant $y$and $x \geq y$is the best approximation that is gluing the two parts of the string is fixed.
2. If the best approximation to a portion of the string $a$is a string constant, its expression is the arithmetic mean of the words of this passage.
3. If a string $b$is the best approximation of each of the fragments a(the number of fragments can be anything, it's important to cover the entire string a), it is the best approximation over a.

This latter fact can be explained, as in particular, zero-one, by means of the derivative of the argument expression $w_{a,b}$. Justification of the third fact is broken pattern on $w_{a,b}$the pieces corresponding to the aforementioned fragments a. Unfortunately, the exact proof of the first fact is more complicated and quite technical. W (telegraph) is briefly on the fact that we start from any best-decreasing approximation $u$gluing two pieces and show how to modify the sequence of the following steps $u$in order to become the first permanent glued on each of the fragments, and then all became. Each of the described transformation is a continuous podciąganiu or leaving certain parts within u, and use of the fact that the approximation error function defining the constant string is square and convex.

The stated justification for the correctness of the algorithm is very incomplete. At the end of the article, you can find a link to the full proof of correctness.

Further history of the task

One of the tasks of the Baltic Olympiad in Informatics BOI 2004 was a task Sequence. It turned out to be very difficult and it has been solved correctly only by the winner of the contest - Filip Wolski. This task also consisted of finding niemalejącego (in the original: the growing and integer, but it did not until this matter) the $b$one that best approximates the given string a, except that it was a measure of the approximation error:

$$w_{a,b} = |a_1 - b_1| + |a_2 - b_2| + \ldots + |a_n - b_n|$$

Original, very ingenious solution of Philip (time complexity $\mathcal{O}(n \log n)$) is described in the Delta 11/2007 . What is interesting, however, solving the problem of TopCoder can be used also in this task! Simply replace the function of the `the average` of the above pseudo-code function, which determines the best approximation to a constant in accordance with the new form of the formula for *the* $_{a, b}$, and namely it is the median over. (Justification correctness of this statement, and check that all described in the previous section, there are also facts in this case, skipping). There remains the question of how effectively the median set.

Now, with each element of the list `L` is sufficient to bind a balanced tree S(eg AVL or red-black) to store all the elements corresponding to the item list. Using such a tree, when you $\mathcal{O}(\log n)$set the median of the elements contained in it. There remains only the problem of how to update the tree when merging elements of the list, or simply as the merge tree. It turns out that it is enough to move all items from the *smaller* to the larger tree, that is successively removed from the smaller and inserted into a larger one. Then it is true single merge may take a long time, but the total cost of all will merge $\mathcal{O}(n \log^2 n)$. This is because each element $a_i$is moved across the algorithm, at most $\log n$times, such as during the transfer size of a tree, to which it belongs (in the sense of the number of elements contained in the tree), increases at least twice. As the cost of a single transfer time to $\mathcal{O}(\log n)$and including the elements we have n, so in fact the algorithm time complexity is $\mathcal{O}(n \log^2 n)$.

Going forward, the task can be solved similar to the two already presented, which minimizes the value of the expression:

$$w_{a,b} = \max(|a_1 - b_1|, |a_2 - b_2|, \ldots, |a_n - b_n|)$$

This can also be applied to the general scheme in which the operation this time instead of `the standard` insert mean of the minimum and maximum element substring. (Check that this is what is the best approximation to a constant, in this case, it is immediate). We obtain the solution of time complexity $\mathcal{O}(n)$. There is also a conceptually much simpler solution to this version of the same task complexity, which, however, did not cite here. All three described the task can be solved in Staszicowym on KI: Approximation , Approximation Strikes Back and Return of approximation .

But this is not the end. It turns out that all three of these tasks occurring patterns $w_{a,b}$are special cases of what in mathematics is called p-the standard n -dimensional vector $a - b$. It is expressed by the formula:

$$||a - b||_p = ((a_1 - b_1)^p + (a_2 - b_2)^p + \ldots + (a_n + b_n)^p)^{1/p}$$

for $p$complete positive and:

$$||a - b||_p = max(|a_1 - b_1|, |a_2 - b_2|, \ldots, |a_n + b_n|)$$

for $p = \infty$. As you can guess, appropriately modified solution to the problem of TopCoder works also in this general case. This allows the operator to develop a general algorithm for all p-of these standards, but also efficient algorithms for the other parameter values p, even for $p = 3, 4, 5$. With poczynionym generalizations, night solving a task of the contest not only led me to Las Vegas, but also to write a thesis in mathematics on monotonic sequences of approximate string. And finally determined the fact that the output of the task occupies the first place among my favorites.

Rating:

Your Rating: No Rating: 5 (3 ratings)

Print this page