Petrozavodsk SU Contest

Problem analysis

Artem Vasilev   Pavel Krotkov

Brazilian ICPC Summer School, 2016

A. Gruz Cable

Problem statement

- You are given a string with N ($N \leq 1500$) Latin characters.
- You can connect two equal characters as long as your connections do not intersect. What is the maximum number of connections?

A. Gruz Cable

Problem solution

- Standard Dynamic Programming problem: dp[l][r] is the maximum number of connections only using wires from l to r.
- Consider the wire at position l: it's either connected to another wire, or it's not.
  1. If it's not connected to any other wire, dp[l][r] will be equal to dp[l + 1][r].
  2. In the case it's connected to some other wire, bruteforce over all wires of the same color as wire l from $[l, r]$. If we choose to connect wires l and m, the answer would be 1 + dp[l + 1][m - 1] + dp[m + 1][r - l].
  3. Choose the best answer and store this choice somewhere to restore the wires afterwards.
- Total runtime: $O(N^3)$ with a small constant; will possibly require non-asymptotic optimizations to fit in TL.

B. Genealogy

Problem statement

- You are given a rooted tree with N ($N \leq 100000$) vertices and some queries of total size M ($M \leq 3000000$).
- A query is a group of vertices, and we are required to calculate the number of vertices which are ancestors of at least one vertex in given group.

B. Genealogy

Solution

- Run the DFS on this tree. Write all the vertices from the query in the order they were first visited during DFS. Also save the depth of each vertex $depth(i)$.
- The first vertex v_1 in our query contributes $depth(v_1)$ to the answer. For $i > 1$, vertex i adds $depth(v_i) - depth(LCA(v_i, v_{i-1}))$.
- We can find all LCAs in $O(M \log N)$ time with any online algorithm or in $O((N + M)\alpha(N)$ with Tarjan's offline algorithm using Union-Find data structure.
- Total runtime: $O(M \log N)$

C. LCA Online

Problem statement

- Once again, a rooted tree with N ($N \leq 100000$) vertices and M ($M \leq 100000$) queries.
- Queries can change the parent of any vertex and ask for LCA of two vertices.

C. LCA Online

Solution

- Construct the **Euler tour** of the input tree. For this problem I'll use the version of Euler tour containing directed edges. All the subtrees in this tour will be a contiguous array. We will also store depths along with vertices themselves.
- For this problem we will store the Euler tour in a implicit treap. Changing the parent of a vertex can be implemented as follows:
  1. Cutting the subarray of Euler tour correspon
  2. Adding a constant to all depths to make up for a depth change
  3. Inserting the subarray back
- For the LCA query, take the subarray between the two ingoing edges for u and v, and find the vertex with minimum depth in it.
- All the queries are made online, each one takes $O(\log N)$ time, so the total runtime is $O(M \log N)$.

D. YAPT

Problem statement

- You are given a long string S and some more queries: find k-th non-palindrome substring starting from position i.

D. YAPT

Solution

- First, for all positions find the largest palindrome with the center in this position for both odd and even palindromes. This can be accomplished with a modification of Z-function algorithm called *Manacher's algorithm*.
- The neat trick is to replace string S with string S' which is string S interspersed with some new symbol \$. After that, we don't need a separate code for even palindromes, only for odd.
- We'll solve all the queries in order of decreasing i. We will also keep track of all centers of palindromes to the right of i whose left border is to the left of i.
- Now the problem of finding k'th non-palindrome prefix of $S[i..N]$ is the same as finding k'th number in the set which is **not** a valid center. This can be done with any segment tree-like data structure in $O(\log N)$ or $O(\log^2 N)$ time.
- Every palindrome center get added and removed only once, so the total runtime is $O((N + M) \log N)$.

E. Paths

Problem statement

- A tree of n vertices
- Every edge has a guard with two parameters: *rank* b_i and *greediness* c_i
- For every vertice of the tree a person walks from the root to this vertice
- Person pays every guard on his path c_i in case he haven't bribe anyone with lesser rank so far
- Otherwise, person pays median value of the bribes he gave to guards with lesser rank
- Calculate total amount of the bribes for every person

- On every edge all persons coming through it pay same amount of money (we'll call it C_i)
- If we calculate this amount for every edge we can easily solve the problem
- C_i depends only on the amounts paid on previous edges on the walk from root

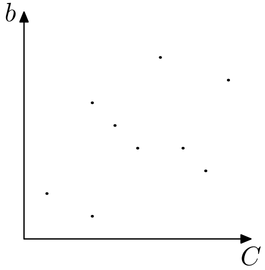
E. Paths

General idea

- We'll do a DFS search in our tree starting from the root
- During DFS we'll store some data about payments during the walk from root to current vertice
- We'll derive C_i for every edge based on this data during DFS

E. Paths
Calculating C_i (1)

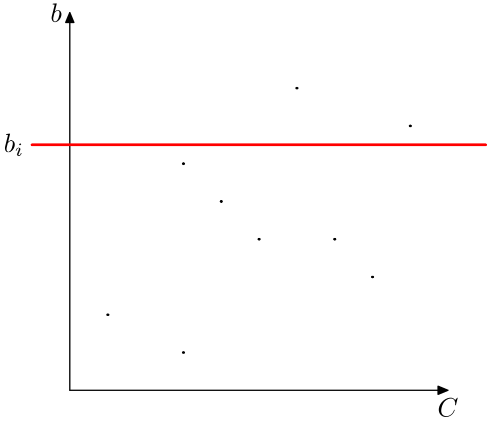
- Let's consider all the data we have when standing in some vertice
- We have a set of guards we already paid to
- For every guard we have his rank b_i and the amount of money we paid to him C_i
- Greediness of passed guards doesn't matter anymore
- We can think of this data as of set of points on a plane



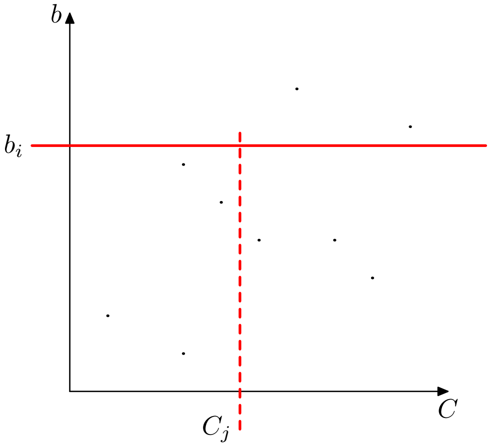
E. Paths

Calculating C_i (2)

- We met a guard with rank b_i
- To decide a size of his bribe we need to consider only guards with lower rank



- There are t considerable points
- $\frac{t}{2}$ of them lie to the left from C_i



- Segment tree
- Every leaf corresponds to some possible size of bribe
- There can be $\approx 10^9$ leaves, we'll discuss it later
- Every vertex of segment tree stores all points with bribe size from it's range
- Points are stored in Cartesian tree (or any other BST) by rank

E. Paths

Calculating C_i (4)

- We traverse down the segmnet tree to find C_i
- In every vertex we calculate amount of points with sufficient rank using BST operations
- Based on that information we decide whether we are going to the left child of segment vertex or to the right one
- Total complexity: $O(n \times \log n \times \log maxC)$

E. Paths

Segment tree modification

- There will be no more then n non-empty leaves in our segment tree
- Let's create a leaf and all it's missing ancestors only when we need to add a point to it
- This way tree height is $\log maxC$, but total amount of vertices is $n \times \log maxC$

F. Scarf

Problem statement

- We have a scarf which consists from 2^X segments
- We fold it m times following some rules
- Need to find position of some segment after all the folding

F. Scarf

Problem solution

- We need to simulate folding process
- 4 variables are enough: *length*, *height*, *segPosition*, *segHeight*
- On each iteration $height \leftarrow height \times 2$
- On each iteration $length \leftarrow \frac{length}{2}$
- We have 3 different cases on each iteration of folding

F. Scarf

First case

- $segPosition \le \frac{length}{4}$
  - $segPosition \leftarrow \frac{length}{4} - segPosition$
  - $segHeight \leftarrow height \times 2 - segHeight$
- $\frac{length}{4} < segPosition \le 3 \times \frac{length}{4}$
  - $segPosition \leftarrow segPosition - \frac{length}{4}$
- $segPosition > 3 \times \frac{length}{4}$
  - $segPosition \leftarrow 3 \times \frac{length}{2} - segPosition$
  - $segHeight \leftarrow height \times 2 - segHeight$

F. Scarf

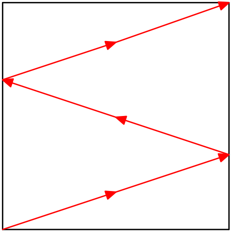
Second case

- $segPosition \leq \frac{length}{4}$
- $segHeight \leftarrow height \times 2 - segHeight$

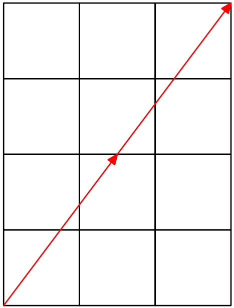
G. Square

Problem statement

- We have square with reflectable walls
- Ray of light should get from the bottom-left corner to the top-right corner
- It should do it without hitting other corners
- It should be reflected K times



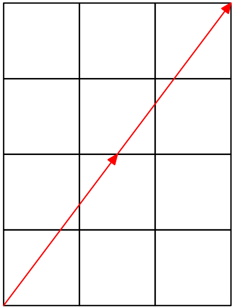
- Let's reflect our square instead of ray
- This way we'll get field consisting from $n \times m$ squares
- Ray should go between corners without hitting any internal knots



G. Square

Constraints on n and m

- $n+m=K+2$ – so that we had exactly K reflections
- $n+m=0 (\mod 2)$ – so that we finished in the top-right corner
- n and m are coprime – so that we won't hit any internal knots



G. Square

Solution

$$answer = \begin{cases} 0, \text{ if } K = 1 \pmod 2 \\ \phi(K+2) \text{ otherwise} \end{cases} \quad (1)$$

H. Strings

Problem statement

- Two strings A and B, $|A|, |B| \leq 10\,000$
- Find all substrings of A that occur at B exactly K times

- Let's divide our problem to $|A|$ simpler problems
- i-th problem will be *Find all substrings* $A_{i..k}, s_i \leq k \leq |A|$ *that occur in* B *exactly* K *times*
- $A_{i..s_i-1}$ is the longest substring which starts at A_i and occured in A before A_i

- Consider the string $C = A_{i..|A|} + \$ + B$ and Z-function for this string
- Substring $A_{i..j}$ occurs in B exactly

$$D_{i,j} = \sum_{k=|C|-|B|+1}^{|C|} \begin{cases} 1, \text{ if } z_k \geq j - i + 1 \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

times

- We can calculate $D_{i,j}$ for all values of j and fixated i in linear time

Calculating s_i

- Now we only need to find s_i values
- We can just update s_i with $i + z_i$ on every iteration of our solution

- Total complexity: $O(n^2)$

I. Triangle

Problem statement

- You are given a regular polygon with vertices colored in two colors.
- Find a number of isosceles (two sides are equal) triangles, where all three vertices are of the same color.

I. Triangle

Problem solution

- Let's fix the color of the triangle and replace all the occurences of that color with 1 and all the other characters with 0. Call that new array a_i.
- Consider position i: how many triangles there are with i as it's top vertex? The answer is $\sum_{j=1} \frac{N}{2} a_{i+j} a_{i-j}$. Notice that the sum of indices is always equal to $2i$ mod N

- Let's consider another array $b_i = \sum_{j=0}^{i} a_j a_{i-j}$. Having this array allows us to get answers for all vertices. How do we get it? Fast Fourier Transform.
- One last detail: we counted equilateral triagles 3 times instead of one. This is only possible if $n = 3k$, and it's sufficient to check all the triangles $(i, i+k, i+2k)$ and subtract 2 each time we encounter a one-colored triangle.
- FFT takes $O(N \log N)$ time and all the other processing can be done in linear time.