AGC 009 解説

DEGwer

2017 年 1 月 22 日

For International Readers: English editorial starts on page 6.

A: Multiple Array

数列の最後の要素の値を変える方法は、最後のボタンを押す以外ありません。つまり、最後のボタンを押すべき回数が $modB_N$ で定まり、すなわち最後のボタンを押すべき最小回数がわかります。最後から 2 番目の要素の値を変えるためには、最後のボタンか最後から 2 番目のボタンを押すしかありません。最後のボタンを押す最小回数が求まっているので、最後のボタンと最後から 2 番目のボタンを押す回数の合計の最小の値も求まります。

数列を後ろから順に見て、以下同じことを繰り返していく貪欲法を行えば、この問題を解くことができます。計算量は $O(N)$ です。

B: Tournament

人 x が y に負けたとき、y の親を x としたような木上での DP を考えます。

$DP[v]$ を、人 v に対応する頂点を根とした部分木だけに人を限定して考えたときのトーナメントの深さの最小値とします。求めるべきものは $DP[1]$ です。

では、更新はどのように行えばいいでしょうか。v に子が k 個あるとします。この k 個の子に対応する人たちはみな v に直接負けています。つまり、v は v の子たちと、1 回ずつ試合をしています。

v がその子たちと、$v_1, ..., v_k$ の順に試合をしたとすると、v_i を根とする部分木で最もトーナメント上で深い位置にいる人は、v を根とする部分木に含まれる人たちの中で優勝をするためには、$DP[v_i] + i$ 回の試合をする必要があります。つまり、$DP[v]$ の値は、すべての子の並べ方に対する、$max_i DP[v_i] + i$ の最小値です。

あとは、v の子に対する DP 値が与えられたときに、上述の値を求められればよいです。これは、もし $i < j$ に対し $DP[v_i] \leq DP[v_j]$ なら $max(DP[v_i]+i, DP[v_j]+j) \geq max(DP[v_i]+j, DP[v_j]+i)$ なので、子の DP 値を降順にソートして順に $v_1, .., v_k$ とする貪欲法で求めることができます。

計算量は全体で $O(NlogN)$ となります。

C: Division into 2

$A \leq B$ として一般性を失いません。$A > B$ なら、 A と B を swap すればよいです。

まず、$S_i + A > S_{i+2}$ なる i が存在する場合、こたえは 0 であることがわかります。

$DP[i][j]$ を、小さいほうから i 番目の要素 S_i までを振り分け、最後に Y に振り分けた要素が小さいほうから j 番目の要素 S_j であるような場合の数とした DP を考えます。

この DP を愚直に計算すると間に合わないので、高速化をする必要があります。i 番目の要素を X に振り分けるとき、つぎの 2 つの場合があります。便宜上、0 番目の要素として負の無限大が入っており、これは Y に振り分けられるとして考えます。

$S_i - S_{i-1} < A$ のとき、もし S_i を X に振り分けるならば、S_{i-1} は Y に振り分けられなければなりません。すなわち、$DP[i][0] = ... = DP[i][i-2] = 0$ です。また、$DP[i][i-1] = DP[i-1][i-1]$ となることも容易にわかります。

そうでないとき、S_i を X に、何の制約もなく振り分けることができます。すなわち、$DP[i][0] = DP[i-1][0], ..., DP[i][i-1] = DP[i-1][i-1]$ です。

また、S_i を Y に振り分けるとき、$DP[i][i] = \sum_{s_i - s_j \geq B} DP[i-1][j]$ として更新を行えばいいです。

さて、$DP[i][x]$ が最初に 0 でなくなる位置のポインタを常に保持しておけば、このポインタの位置は単調に増加するので、この DP は、同じ配列を使いまわすことで全体で $O(N)$ 回の操作で更新を行うことができます。

時間計算量は尺取法のような実装で $O(N)$ 、Y に振り分ける時の和をとる範囲を求めるところに二分探索を用いたり、和をとるのに BIT を用いたりしても $O(NlogN)$ となります。

D: Uninity

まず、問題文のような木の構成法について、ウニ度 k の木を作るときの中心の点として初めて追加された点に値 k を書くことにすると、木のウニ度が k であることは、全ての i に対して、値 i の書かれた任意の 2 つの頂点間のパス上に少なくとも 1 つの、値 $i+1$ 以上の書かれた頂点が存在することと同値です。これは、ウニ度最大の点で再帰的に木を分割していくことを考えると簡単に示すことができます。

さて、これ以降、このような値の書き込み方をするときに、書かれている値が最大のものを最小化すること考えます。

まず、木を適当な頂点を根として DFS して根つき木とします。値の書き込み方をひとつ固定します。

頂点 v と値 x に対し、v から v を根とする部分木に含まれる頂点 u へのパス (端点含む) であって、そのパス上の頂点に書かれた値が u 以外では x 未満であり、u では x であるような頂点 u が存在するとき、頂点 v から値 x が見えると呼ぶことにします。さらに、各頂点 v について、全ての値 x に対し、頂点 v から値 x が見えるときには 2^x を、見えないときには 0 を加算してできる値を $p[v]$ と書くことにします。

定義より、最大の値が書かれた頂点は根から必ず見えるので、p[根] の値の全ての値の書き込み方に対する最小値がわかれば、求める最小値もわかります。

では、p[根] の最小値はどのように計算すればいいでしょうか。実は、葉から順に p[頂点] としてありうる値の最小値を順に定めていく貪欲法で、この値を求めることができます。

以下これを証明します。頂点 v の子が順に $c_1, .., c_k$ であり、$c_1, .., c_k$ を根とする部分木に含まれる頂点に対する値の割り当てが決まっているとします。

このとき、v に書き込める値の最小値は、$p[c_1], ..., p[c_k]$ の二進表記のうち 2 つ以上で立っているビットの中で一番上位のものより上のビットのうち、$p[c_1], ..., p[c_k]$ の二進表記のいずれでも立っていないビットの中で一番下位であるようなビットに対応する値です。この値を t として、 t を頂点 v に書き込むことにすれば、$p[v]$ の値は、$p[c_1] or ... or p[c_k]$ の下位 $t-1$ ビットを 0 で埋めた後、t ビット目を立てたような値です。ただし、or は bitwise or を表します。

さて、このように $p[c_1], ..., p[c_k]$ の値を受け取って、v に書き込める最小の値を v に書き込んだときの $p[v]$ の値を返す関数を f とします。

まず、上述のビットの立て方より、v には v に書き込める最小の値を書き込んだときが一番 $p[v]$ の値が小さくなることがわかります。

あとは、$f(p[c_1], ..., p[c_k]) \leq f(p[c_1]+1, ..., p[c_k])$ を示せば、この不等式を繰り返し用いることによって、このアルゴリズムがすべての頂点 v で、特に根で、$p[v]$ としてありうる最小の値を求めてくることがわかります。これは、値の二進表記に 1 を足す操作が、下位に連続する立ったビットを 0 にして、そのすぐ上の立っていないビットを 1 にする操作であるということに気を付けて場合分けを行うと、証明することができます。

以上より、この貪欲法が最小の p[根] の値を求めてくることがわかりました。「木の重心に最大の値を書き、木をいくつかに分割して再帰的に書き込む」というアルゴリズムで、ウニ度が $logN$ 以下の解が得られるので、各頂点 v での $p[v]$ の値は高々 $logN$ ビットであり、全体で $O(NlogN)$ でこの問題を解くことができます。

E: Eternal Average

x が K 進表記で $0.x_1x_2x_3...x_t(x_t \neq 0)$ と書けるとします。最終的に値 x が残るような操作列は、どのようなときに存在するでしょうか。

0 の書かれた葉を N 個と 1 の書かれた葉を M 個用意し、$a_1, ..., a_K$ を消してその平均をとる操作を行ったとき、新しくその平均の書かれた頂点をつくり、$a_1, ..., a_k$ の書かれた頂点 K 個の親に設定してできるような K 分木を考えます。最終的に残る値は、各 1 の書かれた葉 v に対し、この木上でのその葉の深さを d_v としたときの、K^{-d_v} の総和です。

つまり、最終的に x が残るとき、正整数 $a_1, ..., a_M$ を用いて $x = K^{-a_1} + ... + K^{-a_M}$ と書き表すことができます。

よって、最終的に x が残るためには、$x_1+x_2+...+x_t \leq M$ が必要です。0 と 1 を入れ替えて、値 $1-x$ について同様の議論を行うことで、$(K-1-x_1)+(K-1-x_2)+...+(K-1-x_{t-1})+(K-x_t) \leq N$ も必要なことがわかります。

さらに、$x = K^{-a_1} + ... + K^{-a_M}$ として x を K 進表記で計算するとき、繰り上がりが発生することがありますが、繰り上がりによってその各桁の和を $K-1$ で割ったあまりは変わらないので、最終的に x が残るためには $x_1 + x_2 + ... + x_t \equiv M(mod(K-1))$ も必要です。

逆に、これらの 3 つの条件が満たされれば、0 が $(K-1-x_1)+(K-1-x_2)+...+(K-1-x_{t-1})+(K-x_t)$ 個と 1 が $x_1 + x_2 + ... + x_t$ 個になるまで 0,1 の個数を (0 だけ、もしくは 1 だけの平均をとることで) 減らした後、x_i たちの値に応じて適切に平均をとることで、x を残すことができます。

上の 3 条件を満たすような x の個数は、$DP[i][j]$ を、x の i 桁目までを決定し、i 桁目までの各桁の和が j となるような場合の数として更新していくことで得られる情報から簡単に求めることができます。計算量は $O((N+M)^2)$ となります。

AGC 009 Editorial

DEGwer

A: Multiple Array

Let x_i be the number of times the i-th element is incremented. It is easy to see that $x_1 \geq \cdots \geq x_N$ must be hold. On the other hand, when this inequality holds, it is always possible to find such a way to press buttons (specifically, press the i-th button $x_i - x_{i+1}$ times). The total number of button pressings is x_1.

Thus, we want to minimize x_1 under the constraints that:

- For each i, $A_i + x_i$ is a multiple of B_i.
- $x_1 \geq \cdots \geq x_N \geq 0$

In the optimal solution, we can assume that $x_N < B_N$: otherwise we can replace x_N by $x_N - B_N$ and the condition still holds. Thus, x_N should be the minimum value that satisfies the condition $x_N \geq 0$ and $A_N + x_N$ is a multiple of B_N. Similarly, we can determine the values of $x_N - 1, \cdots, x_1$ greedily in this order.

The time complexity is $O(N)$.

B: Tournament

Construct a rooted tree with N vertices: the vertex 1 is the root, and the vertex a_i is the root of the vertex i. (The constraints guarantee that this is a valid rooted tree). Let's call it G.

When can it be a rooted tree that corresponds to a tournament of depth k? Consider a full tournament of depth k (that is, a tournament with 2^k people) and let G_k be the tree that corresponds to this tournament. The answer of the solution is the minimum k such that G_k contains G.

The tree G_k has the following structure:

- G_0 is a single vertex.
- The root of G_k has k children: the children corresponds to the roots of $G_0, \ldots, G_{k-1}$.

We solve this task by the following DP. For each vertex v in G, define $dp[v]$ as the minimum t such that the subtree of G rooted at v is contained in G_t.

How to compute the value of $dp[v]$? Let $v_1, \ldots, v_c$ be the children of v. From the observation above, $dp[v] \leq k$ if and only if $c \leq k$ and $dp[v_i] \leq k - i$ possibly after permutating the children. It's easy to show that the children should be sorted in the decreasing order of the dp value. Thus, after the children are sorted in this order, $dp[v] = \max\{dp[v_i] + i + 1\}$.

The complexity is $O(NlogN)$.

C: Division into 2

Define $DP_X[i][j]$ as the number of ways to divide the first i integers into two sets such that:

- The last (i-th) element is in X.
- The last element that is in Y is the j-th integer.

Similarly, define $DP_Y[i][j]$.

What are transitions from $DP_X[i][j]$? First, when $S_{i+1} - S_i \geq A$, we can try to include S_{i+1} into the set X. In this case, for each j, we should add $DP_X[i][j]$ to $DP_X[i+1][j]$. Also, when $S_{i+1} - S_j \geq B$, we can try to include S_{i+1} into the set Y. For each j that satisfies the inequality above, we shuold add $DP_X[i][j]$ to $DP_Y[i+1][i]$.

Now, instead of creating a 2D array, we keep an array $DP_X[j]$ that represents $DP_X[i][j]$ in the definition above for fixed i. What happens when we increment i? First, compute the sum of $DP_X[j]$ for all j that satisfies the condition $S_{i+1} - S_j \geq B$, and remember this value. Then, if $S_{i+1} - S_i < A$, "clear" the DP_X array. Then, add the remembered value to $DP_Y[i]$. Do a similar thing for DP_Y array.

Thus we need a data structure that supports the following:

- Compute the sum of numbers in the given range.
- Update the value of one element.
- Clear the data structure.

This can be done by Binary Indexed Tree and the set of positions that are updated. When you want to clear the data structure, you write zero to all updated positions one by one. This way, when you are given $O(Q)$ update queries, a single clear query may be slow, but the amortized complexity will be $O(Q)$.

The total complexity of this solution is $O(NlogN)$. Also, if you use two-pointer method, you can get an $O(N)$ solution (but a bit complicated).

D: Uninity

Assign a label to each vertex. When you construct a tree of uninity k as described in the statement, you label the center vertex with the label k. This way, you get the labelling with the following property:

- When two distinct vertices s, t have the same label ($label(s) = label(t) = k$), there exists a vertex u on the path between s and t such that $label(u) > k$.

In this task, you are asked to label the given tree with the property above (and minimize the maximum label).

Assume that the given tree is a rooted tree. Let v be an arbitrary vertex of the given tree, and suppose that you decide the labels for all vertices in the subtree rooted at v. We say that a label k is *visible* from v if

- $label(v) = k$, or
- There exists a vertex u such that $label(u) = k$ and no label on the path from u to v is greater than k.

In other words, when k is visible from v, you can't attach a vertex with label k to v.

Now, for each vertex, along with its label, let's write the bitmask of visible labels. For example, when $2, 3, 5$ are visible from v, we write $2^2 + 2^3 + 2^5 = 44$ on v. If we can compute the minimum possible bitmask for the root, we get the answer of the problem.

What is the minimum possible bitmask for the vertex v when it has two children p, q and their bitmasks are known? (For simplicity we assume that the number of children is two, but this is not essential). Let M be a very big constant, and define $weight(mask)$ as the number we get when we read $mask$ as a M-ary number. For example, $weight(44) = M^2 + M^3 + M^5$ (basically we compare the masks lexicographically). It is easy to see that the minimum possible bitmask for the vertex v is the minimum x such that $weight(x) > weight(mask[p]) + weight(mask[q])$. This shows that we can determine the labels greedily: we determine the labels from the leaves to the root, and we always choose a label that minimizes the bitmask.

If we do the centroid decomposition, we see that the answer is always $O(logN)$. Thus, the time complexity is $O(NlogN)$.

E: Eternal Average

Let's describe the process as a rooted tree. The tree has $N+M$ leaves: N of them are labelled with 0, and the others are labelled with 1. Each non-leaf node has exactly K children and it is labelled with the average of labels of its children. The label of the root corresponds to the last number.

Let $x_1, \ldots, x_N$ be the depths of the leaves labelled with 0, and $y_1, \ldots, y_M$ be the depths of the leaves labelled with 1. In this case, the last number will be $\sum K^{-y_i}$. When can we construct such a tree?

It turns out that the condition is $\sum K^{-x_i} + \sum K^{-y_i} = 1$. In general, when $\sum K^{-d_i} = 1$ and d is sorted in decreasing order, we can prove that $d_1 = \ldots = d_K$. We can merge these K leaves of depth d_1 into a single node of depth $d_1 - 1$, and if we repeat this process, we finally get a root.

Thus, in this task, we are asked to count the number of rational numbers $z (0 < z < 1)$ such that:

- z can be written as the sum of exactly N powers of $1/K$.
- $1 - z$ can be written as the sum of exactly M powers of $1/K$.

Let's write z as a d-ary number. z must be able to written with finite number of digits. Let $z = 0.z_1 z_2 \cdots z_l (z_l \neq 0)$. We can prove that z can be written as the sum of exactly N powers of $1/K$ if and only if $\sum z_i \leq N$ and $\sum z_i \equiv N \pmod{K-1}$. (The proof is similar to the observation above).

Thus, we want to count the number of sequences $z_1, \ldots, z_l$ such that

- $0 \leq z_i \leq K - 1$
- $z_l > 0$
- $\sum z_i \leq N$ and $\sum z_i \equiv N \pmod{K-1}$
- $\sum (K - 1 - z_i) \leq M - 1$ and $\sum (K - 1 - z_i) \equiv M - 1 \pmod{K-1}$

This can be solved by a straightforward DP in $O(NM)$ time.