

Atcoder Grand Contest 023 解説

writer : maroonrk

平成 30 年 4 月 28 日

For International Readers: English editorial starts from page 7.

A : Zero-Sum Ranges

S を長さ $N + 1$ の数列とし、 $S_0 = 0$, $S_i = S_{i-1} + A_i$ とします。ある連続する部分列の総和は、 S の 2 要素の差です。よって、和が 0 になる連続する部分列の数は、 S の中で値の等しい 2 つの要素を選ぶ方法の数の数に等しいです。これは、 S の要素を sort するなどして、同じ値の個数を数えることで計算出来ます。sort に $O(N \log N)$ がかかるため、この問題は $O(N \log N)$ で解けました。

B : Find Symmetries

ある (A, B) が条件を満たすことと、 $(A + 1, B + 1)$ が条件を満たすことは同値です。($A + 1$ や $B + 1$ が $N + k$ と表される場合は k と同一視してください) よって、 (A, B) の組として、 $(0, 0)$, $(0, 1)$, ..., $(0, N - 1)$ をチェックし、条件を満たすものの個数を N 倍したものが答えです。一回のチェックは $O(N^2)$ で行えます。チェックは N 回行うので、合計で $O(N^3)$ でこの問題は解けました。

C : Painting Machines

期待値の線形性から、各 K について、 K 回マシンを動かして、すべてのマスが黒く塗られているような順列の個数、が求まれば良いです。

結論から言えば、このような順列の個数は $C(K-1, N-1-k) \times K! \times (N-1-K)!$ です。ここで、 C は二項係数です。

このコンビネーションは、次のように考えると導出されます。まず、 $N-1$ 個のマシンについて、動かした or 動かしていないを決めると、その状態になる順列は $K! \times (N-1-K)!$ 個あります。これは、動かした K 個のマシンを動かす順番と、動かしていない $N-1-K$ 個のマシンを動かす順番を考えることに対応しています。

動かした or 動かしていないの決め方について考えます。 $N \leq 3$ のケースは手で処理してしまうとして、 $N \geq 4$ と仮定します。まず、すべてのマスが塗られるためには、マシン 1 及びマシン $N-1$ は必ず動かしている必要があります。そして、動かしていないマシンは必ず、2 つの動かしたマシンに挟まれている必要があります。動かしたマシンは K 個ですから、その隙間は $K-1$ 個です。 $K-1$ 個の隙間から、動かしていないマシンを置く $N-1-K$ 個の場所を選ぶ方法は、 $C(K-1, N-1-K)$ 通りです。

これで、コンビネーションの導出が出来ました。コンビネーションの計算は、階乗とその逆元を前計算しておけば 1 回あたり定数時間で行なえます。前計算に $O(N)$ かかり、コンビネーションも $O(N)$ 回計算するので、この問題は $O(N)$ で解けました。

D : Go Home

マンション 1 に住む社員が A 人、マンション N に住む社員が B 人だとしましょう。

$A \geq B$ のとき、バスは、マンション N よりもマンション 1 を先に訪れます。これは、仮にマンション N の直前までバスが来たとしても、そこからマンション 1 側の全員が負の方向へ投票し続ければ良いことからわかります。また、バスがマンション 1 についてからマンション N に向かう手順は明らかで、直進するだけです。このことから、バスがマンション N に到着する時刻はマンション 1 に到着する時刻に定数を足したものであるとわかります。

投票の際、各社員は、自分の帰宅時間が短くなるような方へ投票します。よって、投票を行う際、ほとんどのケースでは、マンション 1 の住人とマンション N の住人の動きは一致します。唯一の例外は、バスがマンション N の直前にいる場合です。しかしこの場合は、先程考えたように、マンション 1 側にバスが移動してしまいます。

よって、マンション N の住人はマンション 1 の住人と同じ投票をしても良いとわかります。これは、マンション N がなくなって、代わりに $P[1] = P[1] + P[N]$ としたのとほとんど変わりありません。唯一の違いは、最後にマンション N を訪れるかどうかです。

$A < B$ のときも同様にして、マンションの数を減らして考えることが出来ます。

以上の操作を再帰的に繰り返せば、最終的に問題は自明な形（すべてのマンションがバスの初期位置から見えて一方にある状態）になります。一回の再帰に必要なのは定数回の演算なので、この問題は $O(N)$ で解けました。

E : Inversions

$Cnt[k] = A_i \geq k$ なる A_i の個数 $-(N - k)$ とすると、ありうる P の個数は $\prod_{k=1}^N Cnt[k]$ となります。この数を S とおきます。ここで、 $Cnt[k] = 0$ なる k がある場合は、ありうる順列は存在しないため、答えとして 0 を出力します。以下、 $Cnt[k] \geq 1$ とします。

各 i, j について、 $P_i > P_j$ なる P の個数がわかれば良いです。ここではまず、 $A_i \leq A_j$ となる i, j について数えることにします。

この問題の本質部分は、 $P_i > P_j$ なる P の個数は、 $A_j := A_i$ としたときに条件を満たす P の個数のちょうど半分になるということです。 P_i と P_j を swap してできる P が一意に定まり、またこれも条件を満たすことから明らかです。

よって、各 i, j について、 $A_j := A_i$ としたときに条件を満たす P の個数を求めれば良いです。

$A_j := A_i$ とすることで、 Cnt は範囲 $[A_i + 1, A_j]$ の値が 1 減ることになります。

そこで予め、 $D[k] = \prod_{i=1}^k (Cnt[i] - 1) / Cnt[i]$ と定義しておきます。

すると、 $A_j := A_i$ としたときの、 P の個数は、 $D[A_i] / D[A_j] \times S$ となります。ここで、 $D[A_j]$ が 0 の場合は式が壊れてしまいますが、その対策は後ほど解説することにして、とりあえず $D[k] \geq 1$ と思っておきます。

以上より、結局、各 j について、 $i < j, A_i \leq A_j$ なるすべての i について、 $1/D[A_i]$ の総和が求めれば良いとわかります。これは、 j を増やしながら、 $1/D[A_i]$ の和を BIT を管理すれば高速にできます。

ところで、 $D[A_j]$ が 0 の場合は式が壊れていたのです。そこで、 $D[k]$ を形式的に、 $D[k] \times 0^{x[k]}$ という形だと思ってみます。すると、 $D[A_i] / D[A_j]$ が意味を持つ、つまり 0 でないのは、 $x[A_j] = x[A_i]$ のときです。また明らかに $x[k]$ は広義単調増加です。よって、更新は先程とほぼ同様に行い、BIT で和を求めるときは、 $x[A_j] = x[k]$ なる範囲の和だけを取れば良いです。

$A_i > A_j$ となるのも大体同様にできますが、こちらの場合は転倒していないペアの個数を数えることになるため、 A の転倒数を求めてこれから引き算する操作が必要になります。

BIT で $O(N)$ 回のクエリ処理を行うため、 $O(N \log N)$ でこのアルゴリズムは動作します。

F : 01 on Tree

より一般的な問題を考えます。各頂点には 0 と 1 からなる数列が書かれていて、頂点 i にかかっている数列には 0 が $C0_i$ 個、1 が $C1_i$ 個含まれているとします。そして、これらの頂点を、先祖が子孫の頂点の右側に置かれないよう並び替え、頂点にかかっている数列を並び替えた順に取り出して連結し、出来た数列の転倒数を最小化したいとします。なおここで、もともと各頂点に書かれている数列内での転倒数は考えないことにします。(これは並び替えによらず一定なため)

ここで、根以外の頂点の中で、 $C0_i/C1_i$ ($C1_i = 0$ のときは ∞ と考える) が最大の頂点を選び、この頂点を v とします。 v の親の頂点を p とします。ここで、最終的な並び替えにおいて、 p のすぐ右隣に v があるような最適な並び替えが存在することがわかります。もし p と v の間に他の頂点が挟まる最適解があるならば、 v の選び方から、 v を左にずらしても決して損しないことからこれはわかります。

これで、多項式解法が得られました。つまり、 $C0_i/C1_i$ が最大の頂点を選び、それと親の頂点をマージし、一つの頂点にしてしまう、という操作を $N - 1$ 回繰り返す解法です。転倒数は、頂点をマージする際に簡単に計算できます。

この解法はそのままでは $O(N^2)$ なので、改善が必要です。そのために、Union-Find と Priority-Queue を使用します。Union-Find は、頂点のマージを管理します。Priority-Queue は、各頂点の $C0_i/C1_i$ を管理します。この 2 つのデータ構造を使えば、 $C0_i/C1_i$ 最大の頂点を選び、親の頂点とマージする、という一連の動作が $O(\log N)$ 時間で行えるようになり、全体で $O(N \log N)$ でこの問題が解けます。

Atcoder Grand Contest 023 Editorial

writer : maroonrk

April 28, 2018

A : Zero-Sum Ranges

Let S be a sequence of length $N + 1$ defined as follows: $S_0 = 0$, $S_i = S_{i-1} + A_i$. (This is a sequence of prefix sums). Then, the sum of all integers in a contiguous subsequence is the difference of two elements in S .

Thus, the answer is the number of ways to choose two elements from distinct positions in S such that their values are the same. This can be done by sorting the sequence S , and it works in $O(N \log N)$ time.

B : Find Symmetries

A pair (A, B) satisfies the condition if and only if a pair $((A+1)\%N, (B+1)\%N)$ satisfies the condition. This is because if two cells are at symmetric positions in the first pair, they are also at symmetric positions in the second pair.

Thus, we only need to check the following pairs for (A, B) : $(0, 0), (0, 1), \dots, (0, N-1)$. The answer is the number of good pairs among them, multiplied by N .

This solution works in $O(N^3)$ time.

C : Painting Machines

For each K , we want to count the number of permutations such that after activating the first K machines, all cells become black. Then, the answer can be represented as additions/subtractions of these numbers.

Only the set of first K machines matter. For simplicity, assume that $N \geq 3$. First, in order for all cells to be black, machines 1 and $N - 1$ must be in the set of first machines. If the set of first K machines are $1 = a_1 < a_2 < \dots < a_K = N - 1$, for each i , $a_{i+1} - a_i$ must be either one or two. Thus, there are $C(K-1, N-1-K)$ ways to choose them (here C denotes binomial coefficients).

Since we can freely permute machines among the first K machines or remaining $N - 1 - K$ machines, the number we want to compute is $C(K-1, N-1-k) \times K! \times (N-1-K)!$.

If we pre-compute factorials and their inverses, this works in $O(N)$ time.

D : Go Home

Let A, B be the number of employees in apartments 1 and N , respectively.

If $A \geq B$, the bus visits apartment 1 before apartment N . This is because, after apartment $N - 1$ is visited for the first time, everyone except for residents of apartment N will vote for negative direction, and the bus will visit apartment 1 before apartment N . After it visits apartment 1, it directly goes to apartment N because this is the only remaining apartment. Thus, the time when the bus reaches apartment N is, the time when the bus reaches apartment 1 plus constant (constant is the distance between these two apartments).

When a voting takes place, each employee vote for the direction that makes his/her arrival time earlier. Thus, for employees of apartment N , the vote is almost always the same as the vote of residents of apartment 1. The only exception happens after apartment $N - 1$ is visited. However, in this case, the bus will go to negative direction anyway, regardless of the votes from residents from apartment N . Thus, even if they always make the same votes as the residents in apartment 1, it won't change the movement of the bus (except that in the end the bus needs to travel from X_1 to X_N).

Therefore, the answer won't change if we do the following. Move all residents of apartment N to apartment 1 (i.e., $P[1] = P[1] + P[N]$), and decrement N by one. Add $X_N - X_1$ to the answer.

Similarly, when $A < B$, we can also decrease the number of apartments.

We can repeat the operation above recursively while there are at least one apartments for both directions. Since each recursion can be performed in constant time, this solution works in $O(N)$ time.

E : Inversions

For each pair of integers $i, j (i < j)$, we want to count the number of valid permutations such that $P_i > P_j$. The answer is the sum of these numbers for all pairs.

First, only consider pairs i, j such that $A_i \leq A_j$. An essential observation is that the number of valid permutations such that $P_i > P_j$ can be computed as follows: Replace A_j with A_i and count the number of valid permutations. Then the number we want compute is exactly a half of the number of valid permutations (after the replacement). Thus, for each pair i, j such that $i < j$ and $A_i \leq A_j$, we want to compute the number of valid permutations when $A_j := A_i$ is performed.

Let $Cnt[k]$ be (the number of elements such that $A_i \geq k$) minus $(N - k)$. Then the number of valid permutations is $S := \prod_{k=1}^N Cnt[k]$. From now on, we assume that $Cnt[k] \geq 1$ for all k (otherwise there is no valid permutation and the answer is obviously zero).

What happens if we perform $A_j := A_i$? For each k in the interval $[A_i + 1, A_j]$, the value of $Cnt[k]$ will be decremented by one. Define $D[k] = \prod_{k=1}^N (Cnt[k] - 1) / Cnt[k]$. Then, when we perform $A_j := A_i$, the number of valid permutations will be $D[A_j] / D[A_i] \times S$ (unless $D[A_i] = 0$).

Thus, for each j , we want to compute the sum of $1/D[A_i]$ for all i such that $i < j, A_i \leq A_j$. This can be done with *BIT* just like standard inversion number: while we increase j , we keep the sums of $1/D[A_i]$.

Make sure to handle cases with $D[A_j] = 0$ correctly. For example, you can keep $D[k]$ of the form $D[k] \times 0^{x[k]}$. Then, $D[A_i] / D[A_j]$ is nonzero only when $x[A_j] = x[A_i]$. Since $x[k]$ is monotonously non-decreasing, the positions with the same value of $x[k]$ will form an interval. In each interval, you can compute the sum of $D[A_i] / D[A_j]$ using the method mentioned above.

You can handle the cases with $A_i > A_j$ similarly. Note that this time you need compute the product of the inversion number of the original sequence and the original value of S , and subtract what we compute from the product.

We perform $O(N)$ queries with *BIT*, and it works in $O(N \log N)$ time.

F : 01 on Tree

Let's generalize the problem. Each vertex contains multiple 0s and 1s: the vertex i contains $C0_i$ zeroes and $C1_i$ ones. We want to arrange the vertices in a row as in the statement, and minimize the inversion number. Here, we don't count a pair of a zero and a one in the same vertex to the inversion number.

Let v be the non-root vertex with the maximum value of $C0_i/C1_i$ (when $C1_i = 0$, define it as ∞). Let p be its children. Then, we can prove that, in an optimal sequence, v should be immediately after p . (If v is immediately after w and w is not p , we can legally swap w and v , and its inversion number won't be greater.)

Thus, we get the following polynomial-time solution. We start with the given tree, and repeat the following steps $N - 1$ times. In each step, you choose a vertex with the maximum value of $C0_i/C1_i$ (call it v), and merge it with its parent p . When you merge v and p , you should add $C0_v \times C1_p$ to the answer. This solution works in $O(N^2)$ time.

To make it faster, we use Disjoint Set Union and Priority Queue. By Disjoint Set Union, you keep the set of merged vertices. By Priority Queue, you keep the values of $C0_i/C1_i$ of all non-root vertices. This way, each step described above can be performed in $O(\log N)$, and the entire solution works in $O(N \log N)$ time.