AGC 028 解説

writer : maroonrk

平成 30 年 10 月 13 日

For International Readers: English editorial starts from page 8.

A : Two abbreviations

以下、文字列はすべて 0-indexed で考えます。明らかに L は $lcm(N,M)$ の倍数です。

次に、L が一つ与えられたときに条件を満たすものがあるかどうかを判定します。これは、$a \times L/N = b \times L/M$ なるすべての a,b について、$S_a = T_b$ となっているかどうか調べれば良いです。$S_a = T_b$ を満たさないものがあれば、構成は不可能です。逆にこれらの a,b すべてで $S_a = T_b$ ならば、構成は可能です。

$n = N/gcd(N,M),\ m = M/g(N,M)$ とすれば、$a \times m = b \times n$ なる a,b を調べれば良いです。n,m が互いに素であることから、$(a,b) = (k \times n, k \times m)\ (k = 0,1,...gcd(N,M)-1)$ が解であるとわかります。

L の値によらず、調べる a,b の組は同じです。よって、構成が可能な場合は $L = lcm(N,M)$ とするのが最小です。

以上より、この問題は $O(N+M)$ 時間で解けます。

B : Removing Blocks

以下、すべての演算は mod 10^9+7 で行います。特に、mod の逆元を扱うので、これを知らない方は適宜勉強してください。

ブロック i を取り除くときにブロック i と j が連結である確率を $P(i,j)$ と書くことにします。各 j について、$\sum_{i=1}^{N} P(i,j)$ が求まれば良いです。

ブロック i を取り除くときにブロック i と j が連結であるということは、ブロック i と j の間にあるブロック（ブロック $i,\ j$ 含む）の中で、一番最初に取り除かれるものがブロック i であるということです。よって、$P(i,j) = 1/(abs(i-j)+1)$ とわかります。

$1/1,\ 1/2,\ 1/3,\ \ldots\ 1/N$ の累積和を求めておけば、$\sum_{i=1}^{N} P(i,j)$　は各 j について定数時間で求められます。よってこの問題は $O(N)$ 時間で解けます。

C : Min Cost Cycle

辺のコストを $min(A_x, B_y)$ とする代わりに、A_x と B_y のうち好きな方を選ぶ、としても答えは変わりません。

各頂点を 4 つのタイプに分類します。頂点 v のタイプは、以下の様に分類します。

- タイプ X　サイクルにおいて、v から出る辺の重みは A_v を採用する。v に入る辺の重みは B_v を採用する。
- タイプ Y　サイクルにおいて、v から出る辺の重みは A_v を採用する。v に入る辺の重みは B_v を採用しない。
- タイプ Z　サイクルにおいて、v から出る辺の重みは A_v を採用しない。v に入る辺の重みは B_v を採用する。
- タイプ W　サイクルにおいて、v から出る辺の重みは A_v を採用しない。v に入る辺の重みは B_v を採用しない。

各頂点にタイプを定めると、コストは決定します。サイクルが存在するようなタイプの割り振り方は、以下の 3 通りです。

- 全頂点がタイプ Y。
- 全頂点がタイプ Z。
- タイプ X の頂点の個数とタイプ W の頂点の個数が 1 以上かつ等しい。

1 つめと 2 つめのパターンの計算は簡単です。あとは、3 つめの割り振り方をしたときの最小コストを求められれば良いです。

$A_1, A_2, ...A_N, B_1, B_2, ...B_N$ を昇順にソートします。このとき、最初の N 項の中に A_v と B_v が両方含まれる頂点 v が存在すれば、最初の N 項に含まれる重みを採用するようにタイプ分けを行うことが出来ます。よって、最初の N 項の和が答えになります。

それ以外の場合はタイプ W になる頂点の番号を 1 つ決め打って、残りの $N-1$ 頂点に紐付いた重みを小さい方から貪欲に採用する、というふうにすれば答えが求まります。

予め $A_1, A_2, ...A_N, B_1, B_2, ...B_N$ がソートされており、かつその最初の N 項の中に A_v と B_v が両方含まれる頂点 v が存在しないことから、タイプ W の頂点 w を一つを決め打ったあとで選ばれる辺の重みは、最初の N 項から A_w または B_w を除いたものと、$N+1$ 項目または $N+2$ 項目（$N+1$ 項目が A_w または B_w ならば $N+2$ 項目を採用）であるとわかります。

以上でこの問題を解くことが出来ます。ソートがボトルネックになり、計算量は $O(NlogN)$ です。

D： Chords

円環を切り開いて直線として考えます。

$dp[i][j]$ = 点 i が左端、点 j が右端となるような連結成分が存在する様に $[i, j]$ 内の点を結ぶ方法の数、という DP をします。

$[i, j]$ 内の点が、すでに $[i, j]$ 外の点と結ばれることが確定している場合、$dp[i][j] = 0$ です。

そうでない場合、$[i, j]$ 内で、まだどの点とも結ばれていない点の個数を $f(i, j)$ とします。$f(i, j)$ が奇数なら $dp[i][j] = 0$ です。

そうでない場合、$[i, j]$ 内の点を結ぶ方法は、$g(i, j) = (f(i, j) - 1) \times (f(i, j) - 3) \times ... \times 1$ 通りあります。このうち、点 i を左端とする連結成分の右端が j でない場合の数を引けば、$dp[i][j]$ を求められます。点 i を左端とする連結成分の右端が点 k である場合の数は、$dp[i][k] \times g(k + 1, j)$ となります。

以上の DP でこの問題は解くことが出来ます。計算量は $O(N^3)$ です。

E : High Elements

S の最初の数文字が決定しており、現段階で X, Y の高い項の個数が C_X, C_Y、要素の最大値が H_X, H_Y であるとします。このときに、S の残りの文字を適当に割り振って、よい文字列にできるか、という問題が解ければ良いです。

残りを割り振ってSがよい文字列となったとして、そのときのXの高い項が $..., H_X, a_1, a_2, ..a_{|a|}$、Yの高い項が $..., H_Y, b_1, b_2, ...b_{|b|}$ であったとします。このとき、a, b のどちらか一方は、P の高い項のみを含む、としても問題ありません。もし a, b が両方とも P の中で高くない項を含んでいれば、それらを交換することで、P の中で高くない項の個数を減らせるからです。

一般化して、a が P の高い項のみを含むとして考えます（実際のプログラムでは b の場合についても同様に試します）。a, b の中に登場する P の高い個数の合計は固定です。（P の中で高い項はどのような割り振りでも高い項であり続けるため）この個数を Q とします。b が P の中で高い項を k 個含み、P の中で高くない項を m 個含むとします。すると、a の項数は $Q-k$ 個です。S がよい文字列であるためには、$C_X + |a| = C_Y + |b|$ である必要があります。これを変形すると、$2 \times k + m = C_X - C_Y + Q$ となります。

この式の右辺は定数です。これを以下では T と書きます。あとは、左辺の値をうまく調節することを考えます。今解きたい問題は、P でまだ割り振りが決定していない項の部分列 $b = (b_1, b_2, ..., b_{k+m})$ であって、

- $H_Y < b_1 < b_2 \ ... \ < b_{k+m}$
- b 内にある P の高い項の個数を k、それ以外の項数を m とした時、$2 \times k + m = T$

を満たすものを見つけることです。

ところで、$2 \times k + m \geq T$ かつ $2 \times k + m \equiv T \bmod 2$ なる b があれば、$2 \times k + m = T$ なる b は必ず存在します。$2 \times k + m$ を2減らすことができるためです。よって、mod2 ごとに、$2 \times k + m$ の最大値を求められれば判定が出来ます。

これは、予め P の後ろの方から LIS を求めるのと同じ要領で DP を行っておけば良いです。ここで必要な操作は、配列のある値を変更する、配列のある範囲の最大値を求める、です。これはSegmentTreeを用いれば高速に行えます。

よってこの問題は $O(NlogN)$ で解けます。

F : Reachable Cells

分割統治をします。与えられる盤面は必ず縦の長さが横以上のものだとします。そうでない場合は、縦横を反転させてから問題を解きます。盤面を上下半分ずつに分割して、上半分に X、下半分に Y があるような組 X, Y について問題を解くことができればよいです。$H \times W$ の盤面が与えられているときに $O(HW)$ 時間でこれを解ければ、全体で $O(N^2 logN)$ 時間で解けます。

以下では、各マスに書いてある数字を無視し、単に X, Y のペアの個数を数えます（数字がある場合への拡張は容易です)。上半分の盤面を盤面 U と呼び、そのうち i 行目 j 列目のマスを $U(i, j)$ で表します。盤面 U の行数を H_U とします。同様に、下半分の盤面を盤面 D とよび、そのうち i 行目 j 列目のマスを $D(i, j)$ で表します。盤面 D の行数を H_D とします。まず以下のサブルーチンを準備します。

- $MeetingPoint(a, b)$：$D(1, a), D(1, b)$ のどちらからでも到達できるマスの行番号であって最小のもの（存在しないなら ∞）を $O(1)$ 時間で返す。
- $BothReachable(a, b, l)$：$D(1, a), D(1, b)$ のどちらからでも到達できるマスのうち、行番号が l 以下のものの個数を $O(1)$ 時間で返す。

各 $D(i, j)$ に対して、$Left(i, j)$ を、$D(1, x)$ から $D(i, j)$ に到達可能であるような x の最小値（存在しないなら ∞）と定義します。同様に、$Right(i, j)$ を、$D(1, x)$ から $D(i, j)$ に到達可能であるような x の最大値（存在しないなら $-\infty$）と定義します。また、$Top(j)$ を、$U(y, x)$ から $U(H_U, j)$ に到達可能であるような y の最小値と定義します。同様に、$Bottom(j)$ を、$D(1, j)$ から $D(y, x)$ に到達可能であるような y の最大値と定義します。

累積 min の DP をすると、$Left(i, j) \leq a, b \leq Right(i, j)$ なる $D(i, j)$ の中での i の最小値が求められます。この最小値をとるマスを $D(p, q)$ とします。ここで、$p \leq \min(Bottom(a), Bottom(b))$ であれば、$MeetingPoint(a, b) = p$ です。$Left(p, q), Right(p, q), Bottom(a), Bottom(b)$ を実際に達成するパスを考えると、$D(p, q)$ に $D(1, a), D(1, b)$ から到達可能であることがわかるからです。

また、$p > \min(Bottom(a), Bottom(b))$ の場合、明らかに $MeetingPoint(a, b) = \infty$ となります。これで $MeetingPoint(a, b)$ が実装できました。明らかに前処理 $O(HW)$、クエリ $O(1)$ です。

次に $BothReachable(a, b, l)$ を考えます。まず、$MeetingPoint(a, b) > l$ の場合は、0 を返せば良いです。それ以外の場合は、

- $Left(y, x) \leq a$
- $b \leq Right(y, x)$
- $y \leq l$

を満たす (y, x) の個数を数えれば良いです。上 2 つの条件だけならば、次の様に数えることが出来ます。

- まず、すべての $Left(y, x) \leq Right(y, x)$（つまりどちらも ∞ でない）なる (y, x) の個数を数えて、答えに足す。以下でも $Left(y, x) \leq Right(y, x)$ が成り立つものについてのみ数える。
- $Right(y, x) < b$ となる (y, x) の個数を数えて答えから引く。
- $a < Left(y, x)$ となる (y, x) の個数を数えて答えから引く。

- $a < Left(y,x) \leq Right(y,x) < b$ となる (y,x) の個数を数えて答えに足す。

上記 3 つについては、$y \leq l$ の制約を含めても、累積和をとることで求めることが出来ます。ところで、$a < Left(y,x) \leq Right(y,x) < b$ が成り立つならば、必ず $y < MeetingPoint(a,b)$ が成り立つことは、先ほどと同様にパスを書いてみるとわかります。今、$MeetingPoint(a,b) \leq l$ の場合を考えているので、$a < Left(y,x) \leq Right(y,x) < b$ ならば $y \leq l$ は常に成り立ちます。よって、4 番目については、y の条件を無視しし、これも累積和を取ることで求めることが出来ます。前処理は $O(HW + W^2) = O(HW)$ です（$H \leq W$ より）。また、クエリは $O(1)$ です。

また、これを利用することで、「$Reachable(a) = D(1,a)$ から到達できるマスの個数」も $O(1)$ で求めることが出来ます。

サブルーチンが準備できたので、これから各 $1 \leq y \leq H_U$ について、$X = U(y,x)$、$Y = D(i,j)$ であって、X から Y に到達可能な組 X, Y の個数を $O(W)$ で求めます。

$L(x)$ を、$U(y,x)$ から $U(H_U,j)$ に到達できる最小の j（存在しないなら ∞）と定義します。同様に、$R(x)$ を、$U(y,x)$ から $U(H_U,j)$ に到達できる最大の j（存在しないなら $-\infty$）と定義します。

ここで、L, R はともに x に対して単調増加です。よって、次のような問題が解ければよいことになります。

> マスの集合 S を用意する。最初 S は空である。S に対する次のクエリを均し計算量 $O(1)$ で処理せよ。
>
> - S に $D(1,j)$ を追加する。ただしこのときの j は、今まで追加されたどの j よりも大きい。
> - S から $D(1,j)$ を削除する。ただしこのときの j は、S 内のマスの中で j が最小のもの。
> - S 内のいずれかのマスから到達可能であるマスの個数を数える。

$has[j]$ を $D(1,j)$ から到達でき、かつ、S 内にあるすべての $D(1,j')$ $(j < j')$ から到達出来ないマスの個数と定義します。S を変更するクエリのたびに $has[j]$ を更新できれば良いです。削除クエリの際には $has[j]$ は変化しないので追加クエリのことを考えます。

いま S に $D(1,j)$ を追加したいとします。まず、明らかに $has[j] = Reachable(j)$ です。次に、ある $D(1,j') \in S$ に対して、$Bottom(j') \leq Bottom(j'')$ なる $j'' \in S$ が存在すれば、$has[j']$ は変化しません。なぜなら、$D(1,j')$ と $D(1,j)$ からともに到達可能であるようなマスは明らかに $D(1,j'')$ からも到達可能であり、$has[j']$ にカウントされていないからです。よって、$has[j']$ の変化を考えるべき j' は、$Bottom(j')$ の階段凸法に含まれるものだけです。$Bottom(j')$ の階段凸法に含まれる j' を、$J_1 < J_2 < ... < J_k$ とします。$has[J_k]$ は、$BothReachable(J_k, j, min(Bottom(J_k, j)))$ の分だけ減ります。また、$p < k$ なる p について、$has[p]$ の減少分を考えるとそれは、$D(1,J_p)$ と $D(1,j)$ の両方から到達可能であって、$D(1,J_q)(p < q)$ からは到達出来ないマスの個数です。これは $D(1,J_p)$ と $D(1,j)$ の両方から到達可能であって、行番号が $Bottom(J_{p+1})$ 以上のものと一致します。よってその値は、$BothReachable(J_p, j, min(Bottom(J_p), Bottom(j)))$
$-BothReachable(J_p, j, min(Bottom(J_{p+1}), Bottom(j)))$ です。ところで、$Bottom(J_p) \geq Bottom(j)$ となった段階で、$has[J_q]$ $(q < p)$ は変化しないことがわかります。よって、実際に $has[J_p]$ を更新する回数は、$Bottom(J_p) < Bottom(j)$ なる p の個数 $+1$ 回です。$Bottom(J_p) < Bottom(j)$ なる J_p は以降階段凸法に現れないため、W 回のクエリに対して合計で $O(W)$ 時間での計算が行なえます。

以上より、この問題は解けました。

AGC 028 Editorial

writer : maroonrk

平成 30 年 10 月 13 日

A : Two abbreviations

We use 0-based indices.

Suppose that we are given an integer L that is divisible by both N and M, and let's check if we can find a valid string. For each pair (a, b) such that $a \times L/N = b \times L/M$, we want to check if $S_a = T_b$. (The answer is possible if and only if this condition holds for all pairs.)

Let $n = N/gcd(N, M)$, $m = M/g(N, M)$. Then, $a \times m = b \times n$ must holds. Since n and m are coprime, all possible pairs are $(a, b) = (k \times n, k \times m)$ $(k = 0, 1, ...gcd(N, M) - 1)$.

This set of pairs doesn't depend on the value of L. Thus, in case the answer is not impossible, the answer is $L = lcm(N, M)$.

This solution works in $O(N + M)$ time.

B : Removing Blocks

Let's compute the expected value of the total score (when we choose the order of removals uniformly at random).

Let $P(i, j)$ be the probability such that the blocks i and j are connected, when we remove block i. Then, for each j, we compute the value $B_j := \sum_{i=1}^{N} P(i, j)$. This is the expected number of times the value A_j is added to the total score. Thus, the answer is the sum of $A_j B_j$ for all j.

Notice that $P(i, j)$ happens if and only if the first block that is removed among the blocks $i, i+1, \ldots, j$ is the block i. Thus, $P(i, j) == 1/(abs(i-j)+1)$.

If we compute the values $1/1,\ 1/2,\ 1/3,\ \ldots\ 1/N$ (in modulo $10^9 + 7$) and their prefix sums, we can compute each B_j in $O(1)$.

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

C : Min Cost Cycle

Instead of assigning the cost $min(A_x, B_y)$ to the edge from x and y, let's add two edges with costs A_x and B_y (and we can choose whichever we want). This doesn't change the problem.

Let's fix a Hamiltonian Cycle of the graph. There are four types of vertices. The type of a vertex v is classified as follows:

- Type X. The cost of the outgoing edge from v is A_v. The cost of the incoming edge to v is B_v.
- Type Y. The cost of the outgoing edge from v is A_v. The cost of the incoming edge to v is not B_v.
- Type Z. The cost of the outgoing edge from v is not A_v. The cost of the incoming edge to v is B_v.
- Type W. The cost of the outgoing edge from v is not A_v. The cost of the incoming edge to v is not B_v.

Here, for example, "The cost of the outgoing edge from v is not A_v" means that if the edge goes to a vertex w, we use B_w instead.

If we decide the types of all vertices, we can compute the cost. Thus, we want to decide all valid assignments of types to vertices.

Let's color the edges along the cycle. We color it red if its cost correponds to A (of its source), and color it blue if its cost corresponds to B (of its sink). If we follow the cycle, we get N pairs of two consecutive edges. Depending on the colors of the two edges, there are four types of two consecutive edges, and those four types correspond to the four types of the vertex in the middle.

There are three cases:

- All edges along the cycle are red.
- All edges along the cycle are blue.
- The cycle contains both red and blue edges. In this case, the numbers of (red, blue) pairs and (blue, red) pairs must be the same and non-zero, but except for that we can arbitrarily decide the frequencies of the four types.

In word of the vertex types, these are:

- All vertices are of type Y.
- All vertices are of type Z.
- The number of vertices of type X and type W are the same, and they are nonzero.

It's easy to compute the cost of the first two cases, so let's compute the optimal cost of the third case.

Let's sort the values $A_1, A_2, ...A_N, B_1, B_2, ...B_N$ in the increasing order. We want to choose N of these, but we must make sure that for some i, we choose both A_i and B_i. Let's fix such i,

and choose $N-1$ smallest elements except for them. Basically, this is the sum of the first $N-1$ numbers of the sorted array, but in case this part contains A_i or B_i, we must remove them and add next elements instead. However, since there are at most two replacements, we can always do it in constant time.

This solution works in $O(NlogN)$ time (sorting is the slowest).

D : Chords

Let $dp[i][j]$ be the number of ways to make pairs, such that there exists a connected component whose minimum is i and maximum is j. (A connected component contains a set of points, and we consider their minimum index and maximum index.) Then, the answer is the sum of these values.

Let X be the set of points with indexes $i, \ldots, j$, and Y be the set of all other points. First, we must not connect a point in X and a point in Y. If there are x unpaired points in X and y unpaired points in Y, the number of such pairings is $g(x)g(y)$, where $g(x) = (x-1)\times(x-3)\times\cdots\times 1$ is the number of ways to make pairs among x things (or $g(x) = 0$ if x is odd).

In all such pairings, we can find a connected component whose minimum is i. However, there may be some $k(i < k < j)$ such that there is a connected component whose minimum is i and maximum is k, and the component doesn't have j. We need to subtract the number of such pairings. The number of such pairings is $dp[i][k] \times g(z) \times g(y)$, where z is the number of unpaired points among points $k+1, \ldots, j$.

If we fill the $dp[i][j]$ table in the increasing order of $j - i$, we can compute all these values. This solution works in $O(N^3)$ time.

E : High Elements

In order to find the lexicographically smallest good string, we want to check if a certain string can be a prefix of a good string. Suppose that we appended some first elements of P to X and Y, and currently, the number of high elements are C_X, C_Y, and the maximum elements are H_X, H_Y, respectively. (In case X is empty, $H_X = -1$.) We want to check if we can make the number of high elements in X and Y same by assiging the remaining elements properly.

Suppose that after assigning the remaining elements, the high elements of X are $..., H_X, a_1, a_2, ..a_{|a|}$ and the high elements of Y are $..., H_Y, b_1, b_2, ...b_{|b|}$. They must satisfy the following conditions:

- $H_X < a_1 < \cdots < a_{|a|}, H_Y < b_1 < \cdots < b_{|b|}$
- $C_X + |a| = C_Y + |b|$
- All high elements in P (in the "remaining" part) are included in a or b. This is because a high element in P will always be a high element in the new sequence.

These conditions are also necessary, that is, if we can find such a and b, we can find a good assignment. We shuold carefully assign remaining elements that are not in a or b such that they don't become high elements in new sequences (and this is always possible). Thus, we want to determine if such a and b exist.

We can also assume that one of the following holds:

- All elements in a are high elements in P.
- All elements in b are high elements in P.

If none of the above two holds, we can remove one non-high element each from a and b, and still keep the conditions above. Thus, assume that the former holds (we should also try the latter case, but it works in the exactly same way.)

Let Q be the number of high elements among "the remaining elements". If we decide b, we can automatically decide a because a must be the set of all high elements that are not chosen in b (and as we described above, we assume that all elements in a are high in P). The conditions can be rewritten only using b:

- $H_Y < b_1 < \cdots < b_{|b|}$
- $C_X + Q - k = C_Y + |b|$. Here, h is the number of high elements in Q that are included in k.

We can even simplify the second condition. Let m be the number of non-high elements in Q that are included in m (i.e., $m = |b| - k$). Then, the second condition is equivalent to $2 \times k + m = C_X - C_Y + Q$.

In words, we are given a sequence and a constant. Some elements of the sequence are marked as "1 point" and others are marked with "2 points". We want to check if the seqeunce contains a increasing subsequence whose total points is exactly the given constant.

Notice that if there is an increasing subsequence whose total points is $c+2$, we can also achieve exactly c points (by removing some points). Thus, to check if we can achieve exactly c points,

we just need the maximum points we can get, with the same parity as c. We can compute it by a DP with a simple segment tree (similar to the computation of LIS). If we do the DP from right to left and keep the DP table, we can answer these questions for all suffixes of P.

This solution works in $O(NlogN)$ time.

F : Reachable Cells

Sorry, please wait for a while. Our intended solution is $O(N^3)$ for F, $O(N^2 logN)$ for F2.