AGC 037 解説

yutaka1999

2019 年 08 月 17 日

A: Dividing a String

解法 1

条件をみたす分割のうち分割個数が最大である分割を考えると、各 S_i の長さは高々定数（実際には 2）となります。よって、動的計画法により $O(N)$ で答えを求めることができます。

解法 2

文字列 S に対して、求める答えを $f(S)$ とします。さらに与えられる文字列を $s_1s_2\dots s_N$ として $g(i)=f(s_1\dots s_i)$ とします。$i\geq 3$ のとき実は以下の漸化式が成り立ちます。

$$g(i)=\begin{cases}g(i-1)+1 & (s_i\neq s_{i-1})\\ g(i-3)+2 & (s_i=s_{i-1})\end{cases}$$

この漸化式を用いれば先ほどより簡単に $O(N)$ で答えを求めることができます。以下この漸化式を示します。

帰納法により証明します。帰納法の仮定を用いるとまず g が単調増加であることが分かるので、$s_i\neq s_{i-1}$ の場合は $g(i)=g(i-1)+1$ となります。そうでない場合について考えます。$s_1s_2\dots s_i$ の条件をみたす分割を $S_1S_2...S_I$ とおくと $S_{I-1}=S_I=s_i$ となることはないので、$|S_{I-1}|+|S_I|\geq 3$ となります。これより $S_1S_2...S_{I-2}=s_1s_2\dots s_j$ $(j\leq i-3)$ となるので $I-2\leq g(j)\leq g(i-3)$ が成り立ちます。よって $I\leq g(i-3)+2$ であるから $g(i)\leq g(i-3)+2$ です。また $g(i)\geq g(i-3)+2$ であることは、$s_1s_2\dots s_{i-3}$ の条件をみたす $f(i-3)$ 個への分割を取り、$s_{i-2}s_{i-1}s_i$ をうまく 2 分割して先の分割と連結すると、$s_1s_2\dots s_i$ の $g(i-3)+2$ 個への分割が取れることから分かります。よって、この場合も $g(i)=g(i-3)+2$ となるので、漸化式が示されました。

B: RGB Balls

$\sum_j (c_j - a_j)$ の最小値をまず求めます。R, G, B それぞれの色について、その色を持つボールの番号を小さい順に $r_1 < r_2 < \ldots < r_N, g_1 < g_2 < \ldots < g_N, b_1 < b_2 < \ldots < b_N$ として、$m_i = \min(r_i, g_i, b_i), M_i = \max(r_i, g_i, b_i)$ とします。このとき $\min \sum_j (c_j - a_j) = \sum_i (M_i - m_i)$ となります。これを示します。

人に適当な順序を与えて、$a_1 < a_2 < \ldots < a_N$ とします。$i \leq j$ のとき $a_i \leq a_j < b_j < c_j$ であるので、各 i に対し、番号が a_i より小さいボールの割り当てられている人は人 1 から人 $(i-1)$ のいずれかです。特に番号が a_i より小さいボールには各色のボールは高々 $i-1$ 個しかないため、$a_i \leq r_i, g_i, b_i$ つまり $a_i \leq m_i$ となります。よって $\sum_j -a_j \geq \sum_i -m_i$ が成り立ちます。同様にして、人に適当な順序を再び与え $c_1 < c_2 < \ldots < c_N$ となるようにすれば、$c_i \geq M_i$ となり $\sum_j c_j \geq \sum_i M_i$ が成り立ちます。よって、$\sum_j (c_j - a_j) \geq \sum_i (M_i - m_i)$ となります。この等号が成立するようにボールを分配することができるのは明らかです。（人 i にはボール r_i, g_i, b_i を配ればよいです。）よって $\sum_j (c_j - a_j)$ の最小値が求まりました。

次に $\sum_j (c_j - a_j)$ を最小にする配り方の個数を求めます。$\sum_j (c_j - a_j) \geq \sum_i (M_i - m_i)$ を示す過程で、$a_i \leq m_i$ や $c_i \leq M_i$（この 2 つの式における人の順序は異なることに注意して下さい）といった不等式が成り立ちましたが、これらの等号成立が $\sum_j (c_j - a_j) \geq \sum_i (M_i - m_i)$ の等号成立条件です。つまり、$A = \{m_i | 1 \leq i \leq N\}, C = \{M_i | 1 \leq i \leq N\}, B = \{1, 2, \ldots, 3N\} \backslash (A \cup C)$ としたときに、以下の条件をみたすボールの配り方を数えればよいです。

- j 番目の人が受け取ったボールの番号を小さい順に $a_j < b_j < c_j$ とする。
- このとき $a_j \in A, b_j \in B, c_j \in C$

B に属する各ボールに対して、それより番号の小さい A に属するボールとそれより番号の大きい C に属するボールを割り当てる方法を求めればよいので、簡単に数えることができます。以上により、この問題を $O(N)$ の計算量で解くことができます。ただし、人の区別をするので、答えに $N!$ をかけるのを忘れないよう気を付けてください。

C: Numbers on a Circle

操作を逆方向にして考えると、以下のようになります。

言い換え

i 番目の数は B_i です。以下の操作を繰り返して i 番目の数を A_i にします。必要な操作の最小回数を求めてください。

- $1 \leqq i \leqq N$ なる整数 i を一つ選び、$i-1, i, i+1$ 番目の数をそれぞれ a, b, c とする。
- i 番目の数を $b-a-c$ に置き換える。

上と同様、$1 \leqq i \leqq N$ なる整数 i を一つ固定し、$i-1, i, i+1$ 番目の数をそれぞれ a, b, c とします。与えられる数がすべて正なので、常に各数が正である必要があります。このとき次のことが分かります。

- i 番目の数を変更するためには $b > a+c$ が必要。
- その条件が成立しているとき、i 番目以外の数に対してどのように操作を行ったとしても a, c の値は変化しない。
- よって、i 番目の数を変更する必要があり、かつ $b > a+c$ をみたしているならば、他の操作を行う前に i 番目の数を変更する操作を行ってよい。
- つまり、$b > a+c$ かつ変更する必要のある i を探し、その i に対して操作を行うことを繰り返してよい。

特に条件をみたすように操作を繰り返したとき、かかる操作回数は常に一定であることが分かります。ただし、操作回数自体はとても大きくなる可能性があるので、愚直にシミュレーションするのでは間に合いません。そこで、以下のような工夫が必要です。

- $b > a+c$ かつ変更する必要のある i を探し、その i に対して b が A_i と一致するまでもしくは $b \leq a+c$ となるまで i **に対して繰り返し**操作を行う。

このように行うことにするとシミュレーションにかかるステップ数が十分少なくなります。というのも、b が A_i と一致するように操作を行うのは合計で高々 N 回であり、$b \leq a+c$ となるまで操作を行うのは、行うごとに b の値が半分以下になるため、合計で高々 $N \log M_B$ 回（M_B は B_i の最大値）であるからです。（ただし、$x \geq y$ に対して x を y で割ったあまりが $\frac{x}{2}$ 以下であることを用いました。）よって、後は先程記した操作をシミュレーションする方法を考えればよいです。一例をあげると、以下のように解くことができます。

- 優先度付きキュー等を用いて、変更する必要がある数の内、最大の数を取る。
- その数に対して先程の操作を行う。

計算量は $O(N \log N \log M_B)$ です。

D: Sorting a Grid

可能な手順が3段階あるので、このような問題ではまず条件を整理することが重要です。手順3を終えた時点でみたしているべき条件が与えられているので、手順$k(k \geq 1)$を終えた時点でみたしているべき条件というのをkが大きい方から順に整理すると以下のようになります。

- 手順2を終えた時点で、上からi行目は$M \times (i-1)+1$〜$M \times (i-1)+M$からなる。
- 手順1を終えた時点で、各列は以下のように構成されている。
  - 1以上M以下の数が1つ
  - $M+1$以上$2M$以下の数が1つ
  - ...
  - $MN-M+1$以上MN以下の数が1つ

ここで以下のような二部グラフを考えます。

- $2N$頂点NM辺からなり、各頂点は$S_1, S_2, \ldots, S_N, T_1, T_2, \ldots, T_N$とラベル付けされている。
- 1からNMの各数jについて、jが属する行をi行目とする。
- このとき、jに対応する辺e_jが頂点S_iと頂点$T_{[\frac{j+M-1}{M}]}$を繋いでいる。

つまり、手順1を終えた時に、各列に属する数に対応する辺が完全マッチングをなすことが条件であるので、先の二部グラフをM個の完全マッチングに分割することができればよいです。ここで、先の二部グラフの次数がすべて等しくMであることに注目します。実は、ホールの定理から各頂点の次数が等しい二部グラフは常に完全マッチングを持つことが分かるので、適当な完全マッチングを見つけ、それを除いた二部グラフで再び完全マッチングを見つける、ということを繰り返すことで、条件をみたすように手順1を行うことができます。計算量は一つのマッチングを見つけるのに$O(NM\sqrt{N})$ですので$O(NM^2\sqrt{N})$となります。

E: Reversing and Concatenating

S に現れる文字の中で辞書順最小のものを a とします。最終的に S の先頭に a を何文字連続させることができるかを考えます。最初の操作での U において a が最大 L 文字連続しているとすると、実は S の先頭に並べることができる a の個数は最大で $2^{K-1}L$ 個であり、実際に $2^{K-1}L$ 個並ぶように操作をする方法はとても少ないです。

まず高々 $2^{K-1}L$ 個しか a を並べられないことを示します。これは操作を一回行った後の S において a が連続する個数は高々 L であり、操作をもう一度行っても S において a が連続する個数は高々2倍にしかならないことから分かります。次に実際に $\min(N, 2^{K-1}L)$ 個 a を連続させられることを示します。i 回目の操作において、以下のように操作を行うことで a をその個数分先頭に並べることができまｓｙ。

- $i < K$ のとき U において a が $2^{i-1}L$ 個連続する部分が末尾となるように次の S を選ぶ。
- $i = K$ のとき U において a が $2^{i-1}L$ 個連続する部分が先頭となるように次の S を選ぶ。

よって、$2^{K-1}L \geq N$ の場合は a を N 個並べた文字列が答えとなります。そうでない場合は、最終的に a が $2^{K-1}L$ 個先頭に並ぶ必要がありますが、その条件を満たすように操作をする方法は、上に述べた方法しかないです。よって、最初の操作でどの S を選ぶかを決めると、上の条件を満たすように操作をする方法が一意に定まるので、最終的な解の候補が定まります。高々 N 個の解の候補の中で辞書順最小のものを求めるのは $O(N^2)$ で可能であるので、全体として $O(N^2)$ の計算量でこの問題を解くことができます。

F: Counting of Subarrays

ある正整数 K に対してレベル (K, L) に属する数列を**良い数列**と呼ぶことにします。まず数列 $A_1, A_2, \ldots, A_N$ が良い数列であるかどうかを判定する方法を考えましょう。まず $N = 1$ の場合は明らかに良い数列です。$N \geq 2$ の場合は以下が必要十分であることが分かります。

- 数列が一つの値のみからなる場合 $N \geq L$ が必要十分。
- そうでない場合、最小の値 m をとるインデックスの集合を S とする。
- S が $[l_1, r_1], \ldots, [l_k, r_k] (l_i + 1 < r_{i+1})$ の和集合となるように k 個の区間を選ぶ。
- $[l_i, r_i]$ に含まれる要素数が全て L 以上であることが必要。
- そのうえで $[l_i, r_i]$ に含まれる要素を A から削除して、対応する部分に値 $m + 1$ の項を $[\frac{r_i - l_i + 1}{L}]$ 個加えた数列が良い数列であることが必要十分。

では良い部分列の個数はどのようにして数えればよいでしょうか。まず問題を少し拡張して、以下のような問題を解くことにします。

- 数列 A に加えて、数列 $L_1, \ldots, L_N, R_1, \ldots, R_N$ が与えられる。
- このとき長さ 2 以上の良い部分列 $A_i, \ldots, A_j$ に対して $L_i \times R_j$ を足し合わせた値を求める。

実はこのように問題を拡張すると、先ほどの判定法と同様にして最小値を削除し一つ大きな値を適当な個数加える、という操作を行うことでより短い数列に帰着することができます。具体的には以下のようにすればよいです。

- 数列が一つの値のみからなる場合は、よい部分列は長さ L 以上の部分列すべてなので簡単に計算できる。
- そうでない場合、先ほどと同様に区間 $[l_1, r_1], \ldots [l_k, r_k]$ をとる。
- $[l_i, r_i]$ に含まれる長さ L 以上の部分列すべてに対する $L_i \times R_j$ の値の合計値を求める。
- $[l_i, r_i]$ に含まれる要素を A から削除して、対応する部分に値 $m + 1$ の項を $[\frac{r_i - l_i + 1}{L}]$ 個加えた数列を考える。

ここで、新たに追加する $[\frac{r_i - l_i + 1}{L}]$ 個の要素について L_j, R_j の値をどのように定めればよいか考える必要があります。ここで辻褄を合わせるには、以下のように定めればよいです。

- $[l_i, r_i]$ の区間を削除して、そこに新たな項を加える場合を考える。
- 新たに加える項の中で左から j 個目の項の R_j の値を考える。
- $[l_i, r_i]$ の中で左から jX 項目から $jX + X - 1$ 項目までの R の値の合計を R_j とする。

このように定めればよいことは、良い数列かどうかの判定法から分かります。ただし、少しスコアに余分があるので適当な値を引かなければいけません。具体的には以下の値を引けば求めたい値が求まります。

- $[l_i, r_i]$ の区間を削除して新たに追加してできる区間を $[l'_i, r'_i]$ とする。
- $[l'_i, r'_i]$ の長さ L 以上の部分列に対する $L_i \times R_j$ の合計を求める。

以上により求めたい値が求まることは、判定法のアルゴリズムから従います。計算量は解析すると $O(N \log N)$ であることが分かります。これはアルゴリズムに対応する X 分木のようなものを考えると、その頂点数が $O(N)$ であることから従います。以上により、この問題を $O(N \log N)$ で解くことができました。

AGC 037 Editorial

yutaka1999

A: Dividing a String

We can prove that in the optimal solution, the length of each part is at most 4.

For example, suppose that the length of some part is 5. There are four ways to divide it into two parts: "1+4", "2+3", "3+2", and "4+1". At most two of them may be invalid. For example, "2+3" is invalid if the part immediately to the left have length 2 and the string coincides with the first two letters of the current part, or a similar thing for the part immediately to the right. Since there are four divisions and at most two of them are invalid, we can always find a valid division.

(With more careful analysis we can show that the length is at most 2, but this is not necessary to solve the problem).

Using this fact, a straightforward DP works in $O(N)$.

B: RGB Balls

Let $r_1 < r_2 < \ldots < r_N, g_1 < g_2 < \ldots < g_N, b_1 < b_2 < \ldots < b_N$ be the positions of R, G, B balls, respectively. Let $m_i = \min(r_i, g_i, b_i), M_i = \max(r_i, g_i, b_i)$. Then, we claim that the minimum possible value of

Let k be an arbitrary integer. Since k people get one of the red balls $r_1, ..., r_k$, the value of a_* for them is less than or equal to r_k. The same holds for green and blue balls, so at least k people have the value of a_* less than or equal to m_k. This proves that $\sum_j a_j \leq \sum_i m_i$. By addind a similar inequality for c and M, we get $\sum_j (c_j - a_j) \geq \sum_i (M_i - m_i)$. It's obvious that we can satisfy this equation (for example person i gets r_i, g_i, b_i), so we want to count the number of ways to satisfy the equation.

Let A be the set of balls that appear in m, C be the set of balls that appear in M, and B be the set of all other balls. Then, the equation holds if and only if each person gets one ball each from A, B, C, and the three balls satisfy the relative order "$A < B < C$".

We handle the $3N$ balls from left to right. When we handle a ball, we count the number of people who can receive the ball without breaking the rule above, and multiply it to the answer. It turns out that this way the intermediate state is always (essentially) unique, so the answer is simply the product of those coefficients.

C: Numbers on a Circle

Let's see the operations in the reverse order:

> You want to convert the sequence B into A with minimum number of the following operations:
>
> - Choose i and let a, b, c be the $i-1, i, i+1$-th terms.
> - Replace the i-th term with $b - a - c$.

Let i be an arbitrary indice and a, b, c be the $i-1, i, i+1$-th terms. We get some observations:

- $b > a + c$ must hold to perform an operation at i.
- If $b > a + c$ holds, no matter how we perform operations at other places, the values a, b, c never change.
- Thus, if $B_i > A_i$ and $b > a + c$, we can assume that we perform the next operation at i.

Thus, a (slow) solution is the following: Find i that satisfies $B_i > A_i$ and $b > a + c$, and perform an operation at i. Repeat it until B coincides with A.

To make it faster, do the following:

- With a priority queue, keep the set of indices that satisfy $B_i > A_i$, in the decreasing order of B_i.
- While the queue is non-empty, pop the first element and see the index (call it i). If we can't perform the operation at i, the answer is -1 (because we can never satisfy $B_i = A_i$). Otherwise, replace B_i with $B_i \% (B_{i-1} + B_{i+1})$ with $[B_i / (B_{i-1} + B_{i+1})]$ operations.

It's known that if $x > y$, $x \% y \leq x/2$. Thus, each time we perform the operation above on i, B_i is halved, and the total number of opeartions is $O(N \log MAX_B)$.

This solution works in $O(N \log N \log MAX_B)$ time.

D: Sorting a Grid

In the decreasing order of k, let's determine the desired state of the grid after Phase k.

- After Phase 2, the i-th row must be a permutation of $(i-1)M+1, \cdots, iM$. Let's "color" these integers with Color i. Then, after Phase 2, the colors of all squares in the i-th row must be i.
- In Phase 2, we can shuffle each column, and as a result we want to have Color 1 at the top square, Color 2 at the next square, and so on. Thus, after Phase 1, we want each column to contain exactly one square of each color.
- At the beginning, the grid contains exactly M occurrences of each of Color 1, $\cdots$, Color N.

Thus, the problem can be restated as follows:

> You are given an $N \times M$ grid consisting of exactly M squares of each of Color 1, $\cdots$, Color N. You can freely swap two cells in the same row. Find a way to swap the cells such that each column contains each color exactly once.

We decide the colors column by column. First, let's decide the colors of the leftmost column.

Consider the following bipartite graph:

- There are N vertices corresponding to Row 1, ..., Row N.
- There are N more vertices corresponding to Color 1, ..., Color N.
- If Row i contains a square with Color j, add an edge between two vertices corresponding to Row i and Color j.

Let's find a perfect matching of this bipartite graph (this is always possible as described below). If Row i is matched with Color j in the matching, we decide the color of the leftmost square of Row i to be Color j.

Why can we always find a complete matching? This comes from the fact that the degrees of all vertices are the same (M), and all such bipartite graphs have perfect matcihngs. (We can prove this by, for example, Hall's marriage theorem).

Now we fix the leftmost column and consider the remaining $N \times (M-1)$ grid; in this grid each color appears exactly $M-1$ times, so the same

condition holds and we can find a perfect matching again. Thus, we can repeat the same process M times, and achieve the goal.

This solution works in $O(NM^2\sqrt{N})$ time.

E: Reversing and Concatenating

Let a be the lexicographically smallest letter that appears in S.

We want to mazimize the number of consecutive occurrences of a at the beginning of the resulting string (call it M). Let L be the maximum length of consecutive occurrences of a in U (in the beginning). Then, we can prove that $M = 2^{K-1}L$ (unless $2^{K-1}L > N$, in this case $M = N$). This is because, in each step, we can always double the length of consecutive occurrences of a in U (unless it exceeds N), and this is the optimum.

Thus, in case $2^{K-1}L \geq N$, the answer is the concatenation of N as. Otherwise, in order to achieve $M = 2^{K-1}L$, we have limited options at each step. In the i-th step,

- If $i < K$, we must choose S' in U such that $2^{i-1}L$ consecutive occurrences of a comes at the end of S'.
- If $i = K$, we must choose S' in U such that $2^{i-1}L$ consecutive occurrences of a comes at the beginning of S'.

Notice that, once we decide the choice of S' when $i = 1$, the choice of S' in the future can be uniquely determined. There are $O(N)$ choices for the case $i = 1$, and we can get the resulting string for a fixed choice in $O(N)$, so this solution works in $O(N^2)$ time.

F: Counting of Subarrays

Let's call a sequence **good** if it belongs to level (K, L) for some K.

First, let's check if the entire sequence $(A_1, A_2, \ldots, A_N)$ is good. In case $N = 1$, it's good. Otherwise, we can check in the following way:

- If all elements of the sequence are the same, the sequence is good iff $N \geq L$.
- Otherwise, let m be the minimum of the sequence, and let $[l_1, r_1] \cup \cdots \cup [l_k, r_k]$ be the set of indices where the elements are m (here, the intervals are not adjacent, i.e., $r_i + 1 < l_{i+1}$)
- If the length of some interval $[l_i, r_i]$ is less than L, the sequence is not good.
- Otherwise, for each i, replace the interval $[l_i, r_i]$ (that is currently $(r_i - l_i + 1)$ repetitions of m) with $[\frac{r_i - l_i + 1}{L}]$ repetitions of $m + 1$; then recursively check if the new sequence is good.

What's the time complexity of this solution? Each time we handle L elements, the length of the sequence decreases by $L - 1$, so we need to handle $O(N)$ elements in total. Thus, if we keep the elements with sets, it's $O(N \log N)$.

How to count the number of good (continuous) subsequnces? Notice that if Y is a subsequence of X, the sequences we get in intermediate steps when we run the algorithm for Y is also a subsequence of the intermediate sequences of X. Thus, basically, we run the same process, but with "coefficients" associated with elements.

Let's extend the problem a bit. You are additionally given parameters $L_1, \ldots, L_N, R_1, \ldots, R_N$, and your task is to compute the sum of $L_i R_j$ for all pairs (i, j) such that the sequence $A_i, \ldots, A_j$ is good. Then, you can solve the problem in a similar way as the observation above.

- If all elements of the sequence are the same, the answer is the sum of $L_i R_j$ for all pairs (i, j) such that $i = j$ or $j = i + 1 \geq L$, and it can be computed in linear time.
- Otherwise, define m and $l_1, r_1, \cdots, l_k, r_k$ in the same way as above.
- For each i, consider the sequence corresponding to the interval $[l_i, r_i]$, and compute the sum of $L_p R_q$ of good intervals $[p, q]$ within this interval.

- For each i, replace the interval $[l_i, r_i]$ with $[\frac{r_i - l_i + 1}{L}]$ repetitions of $m+1$, with proper values of Ls and Rs associated with those new elements.
- Recursively compute the answer for the new sequence.

How should we define Ls and Rs for the new elements?

We describe it with an example (but works for general cases in the same way). Suppose that $m = 1$, $L = 3$, and the sequence starts with ten 1s, followed by some greater terms.

1, 1, 1, 1, 1, 1, 1, 1, 1, 1, a, b, c, d, e

If we "compress" this sequence, we get the following:

2, 2, 2, a, b, c, d, e

Here, the bold part in the former sequence will be compressed to the bold part in the latter sequence:

1, 1, **1, 1, 1, 1, 1, 1, 1, 1, a, b, c**, d, e

2, **2, 2, a, b, c**, d, e

This means that the value L_3 in the former sequence should be added to the L of the second 2 in the latter sequence.

In this manner, we get the values of L of the three 2s of the compressed sequence should be $L_1 + L_2$, $L_3 + L_4 + L_5$, and $L_6 + L_7 + L_8$, respectively.

Don't forget to avoid extra terms to be added to the answer. Intervals that are entirely contained in the newly added part doesn't correspond to anything in the original sequence, so we should subtract the sum of $L_p R_q$ of good intervals $[p, q]$ within the newly added interval from the answer.

This solution works in $O(N \log N)$ time for the same reason as above.