AGC036 解説

writer : maroonrk

2019 年 7 月 21 日

For International Readers: English editorial starts on page 8.

A : Triangle

$X_1 = 0, Y_1 = 0$ と固定して考えてみます。すると三角形の面積は $|X_2 \times Y_3 - Y_2 \times X_3|/2$ になります。よって、$X_2 \times Y_3 - Y_2 \times X_3 = S$ を満たす X_2, Y_2, X_3, Y_3 を見つけられれば十分です。

ここでさらに、$X_2 = 10^9, Y_2 = 1$ と固定してみます。あとは、$10^9 \times Y_3 - X_3 = S$ を満たす X_3, Y_3 を見つければ良いです。これは S を 10^9 で割った商と余りから求めることができます。詳しくは回答例を参照してください。

回答例

B : Do Not Duplicate

$X_0 = X_i$ を満たす最小の $i > 0$ を考えます。(そのような i がないこともありますが、とりあえずあると仮定します) すると、$X_0, X_1, \cdots, X_i$ を処理した段階で、s は空になっています。次に、$X_{i+1} = X_j$ を満たす最小の $j > i+1$ を考えると、X_j を処理した段階でも s は空です。

一般に、各 i $(0 \leq i \leq N \times K - 1)$ について、次に $X_j = X_i$ なる最小の $j > i$ が存在する時、X_i を処理する直前で s が空なら、X_j を処理した直後、つまり X_{j+1} を処理する直前も s が空です。ここで、$y(i) = (X_j = X_i$ なる最小の $j > i($ そのような j が存在しないなら $\infty)) + 1 - i$ と定義します。すると問題は、次のように言い換えられます。

- 最初、整数 $a = 0$ を持っている。
- ステップ 1: $a + y(a) \leq K$ である限り、$a := a + y(a)$ と置き換え続ける。
- ステップ 2: ステップ 1 が終わったら、$i = a, a+1, \cdots, N \times K - 1$ について、問題文中の操作を行う。

ステップ 1 が終了した時点で、$N \times K - 1 \leq a$ が成り立つことは容易に確認できます。なので、ステップ 2 の実行は簡単です。

ステップ 1 を高速に処理します。X の性質から、$y(i) = y(i \mod N)$ であることがわかります。そこで、ダブリングをします。各 $k(0 \leq k \leq log_2(N \times K)), i(0 \leq i \leq N - 1)$ について、$g_k(i) = (i := g(i) + i$ と置き換える操作を 2^k 回行ったあとの $i) - i$ と定義します。あとはこうして得た g_k を使えば、ステップ 1 が終了する段階の a がわかります。

ステップ 1 の実行に $O(Nlog(NK))$ かかります。ステップ 2 は $O(NlogN)$ で実行可能です。よってこの問題は $O(Nlog(NK))$ で解けます。

回答例

C : GP 2

非負整数列 $p_0, p_1, \cdots, p_{N-1}$ が、M 回の操作後の x としてありうることの必要十分条件は、以下の条件を全て満たすことです。

- $p_0 + p_1 + \cdots + p_{N-1} = 3M$
- $\max(p_0, p_1, \cdots, p_{N-1}) \leq 2M$
- p_i が奇数となるような i の個数が M 個以下。

この条件の必要性は明らかです。十分性は、M に関する帰納法で示します。まず、$M = 1$ のとき、明らかに正しいです。次に、$M = k$ でこの命題が正しいとします。$M = k+1$ において条件を満たす p があったとします。このとき、適切に 2 つの添字 i, j を選び、p_i を $p_i - 2$ で置き換え、p_j を $p_j - 1$ で置き換えると、$M = k$ において条件を満たす p が得られます。これは場合分けによって示すことができます。ここでは詳細は省略しますが、$\max(p_0, p_1, \cdots, p_{N-1})$ と、p_i が奇数となるような i の個数に注目して場合分けをすればいいです。

上で示した 3 つの条件を満たす p の個数を数えることにします。まず、2 番目の条件を無視して考えてみます。ここで、p_i が奇数となる i の個数を固定し、a 個だとします。するとまず、p_i が奇数になる i の選び方が ${}_NC_a$ 通り考えられます。次に、総和が $3M - a$ になる長さ N の偶数列が何通りか数える必要がありますが、これは、総和が $(3M - a)/2$ になる長さ N の整数列の個数と同じです。そしてこれは、${}_{(3M-a)/2+N-1}C_{N-1}$ 個あります。コンビネーションの計算は、適切に前計算をしておけば 1 回あたり $O(1)$ で求められます。前計算に必要な計算量は $O(N + M)$　です。a が $O(min(N, M))$ 通りあるので、全体で $O(N + M)$ で答えが求められます。

2 番目の条件を無視した場合が求まりました。あとは、2 番目の条件に違反する p の個数を数えればよいです。p の総和が $3M$ なので、$p_i > 2M$ となる i は高々 1 つです。$p_0 > 2M$ となる p の個数が求まれば、これを N 倍すれば答えが得られます。よって、次のような問題が解ければ良いです。

> 長さ N の非負整数列 q であって、次の条件を全て満たすものの個数を数えよ。
>
> - $q_0 + q_1 + \cdots + q_{N-1} = M$
> - q_i が奇数となるような i の個数が M 個以下。
> - $q_0 > 0$

ここで、3 番目の条件を無視すると、この問題は先程解いた問題と同じ形をしているので、すぐに解けます。3 番目の条件に違反するものを考えると、それは、先程の問題で $N := N - 1$ とした場合に対応しています。

よってこの問題は全体で $O(N + M)$ で解けます。

回答例

D : Negative Cycle

辺をいくつか削除して、負閉路がなくなったという状態を考えます。この時、次の条件を満たすように、各頂点に整数 p_i を設定できます。

- 各辺 $e = (i \to j)$ について、$p_j \leq p_i + weight(e)$ が成り立つ。

これは例えば $p_i =$ 頂点 0 から頂点 i までの最短距離、と定義すればよいです。逆に、この条件を満たすように p_i を設定することができるなら、グラフに負閉路が存在しないことがわかります。(負閉路が存在すると仮定した場合、そこに沿って $p_j - p_i \leq +weight(e)$ を足し合わせると矛盾)

p_i が満たすべき条件を確認します。まず、各 i $(0 \leq i \leq N-2)$ について、$p_i \geq p_{i+1}$ が成り立つ必要があります。これは、もともとある辺 $(i \to i+1)$ (重み 0) に対応しています。ここで、$q_i = p_i - p_{i+1}$ と定義します。$p_i \geq p_{i+1}$ より、q_i は非負整数です。

次に、辺 $(i \to j)$ (重み 1) について考えると、$p_j \leq p_i + 1$ が成り立つ必要があります。この不等式を q を使って表すと、$q_j + q_{j+1} + \cdots + q_{i-1} \leq 1$ となります。

同様に、辺 $(i \to j)$ (重み -1) について考えると、$q_i + q_{i+1} + \cdots + q_{j-1} \geq 1$ が成り立っている必要があります。

使う辺集合を決めた時、条件を満たすように q の値を設定できるか、という問題を考えます。すると、q の要素としては $0, 1$ だけを用いればよいことがわかります。

よって、次のような問題を解けば良いです。

> $0, 1$ からなる数列 q を設定する。この q で条件が満たされないような辺を削除する。このとき、削除する辺のコストの総和を最小化する。

これは、次のような DP で解くことができます。

- $dp[i][j] = q_0$ から q_i までの $0, 1$ を決定していて、そのうち最後の 1 の位置が i、最後から 2 番目の 1 の位置が j になっているときの、既に削除することが決定した辺のコストの総和の最小値。

$dp[i][j] \to dp[x][i]$ の遷移を考えます。まず、新たに削除する必要のある重み 1 の辺は、$j < a \leq i,\ x < b$ を満たす辺 $b \to a$ です。新たに削除する必要のある重み 1 の辺は、$i < a < b \leq x$ を満たす辺 $a \to b$ です。この遷移は累積和を用いると高速化できます。するとこの DP は $O(N^3)$ 時間で計算でき、この問題を解くのに十分高速です。

回答例

E : ABC string

もとの文字列で同じ文字が隣接している場合は、まとめて 1 文字と考えて良いです。よって、S では隣接する 2 文字が全て異なると仮定します。

S の中で'A','B','C' の個数をそれぞれ a, b, c とします。ここで、$a \leq b, a \leq c$ と仮定しても一般性を失いません。

今、使う 'A' の集合が決まっているとしましょう。このとき、残りの 'B','C' を上手く消すことで、条件を満たす部分列を得られるかどうかを判定することを考えます。

まず、使わない 'A' を削除しておきます。これによって、同じ文字が隣合うことがあるので、それを 1 つにまとめる処理をします。こうして得られた文字列を t とします。また、t に含まれる 'A','B','C' の個数をそれぞれ、p, q, r とします。'A' を 1 文字消すごとに、'B' または 'C' は高々 1 文字しか減らないので、$p \leq q, p \leq r$ が成り立っています。

もしここで $q = r$ なら、'B','C' を上手く消すことで、条件を満たす部分列が得られます。具体的には、$p < q$ を満たす限り、'BC' または 'CB' と 2 文字が隣り合っている部分を選んで消す（このとき 'A' が新たに隣合うことがない）という操作が可能です。これは、操作が不可能な状態（2 つの 'A' に挟まれている部分の長さは 2 以下であり、先頭、末尾にある 'B','C' の個数はそれぞれ 1 以下）を考えると、その段階で既に $2p \geq q + r$ が満たされていることからわかります。

$q \neq r$ の場合について考えます。$q < r$ と仮定しても一般性を失いません。t は、'A' によって $p + 1$ 個の（空かもかもしれない）部分文字列に分割されます。ここで、分割された文字列のうち、長さが 1 であって、2 つの 'A' に挟まれていたもの（つまり、t の先頭や末尾にあったものではないもの）を、単体文字列と呼ぶことにします。この $p + 1$ 個の文字列のうち、'B' を含む文字列の個数を x、'C' からなる単体文字列の個数を y とおきます。ここで、$x < y$ の場合、t からどう 'B','C' を削除しても、'B' と 'C' の個数は同じになりません。逆に、$x \geq y$ の場合、適切に 'C' を削除し続けることで、'B' と 'C' の個数を同じにすることができます。これは、'B' と 'C' の個数の差について注目したとき、'B' を含む文字列なら 'B' が 1 つ多い状態にすることができ、'C' からなる単体文字列の場合は 'C' が 1 つ多い状態にしかできないということからわかります。

もとの問題に戻ります。以上の考察から、この問題は、使う 'A' を上手く選んで、'A' によって分離された部分文字列が次の条件を満たすようにする問題だとわかります。

- 'B' からなる単体文字列の個数が 'C' を含む文字列の個数以下である。
- 'C' からなる単体文字列の個数が 'B' を含む文字列の個数以下である。

まず、'A' をすべて使うと考えてみます。そして、$b1, b2, c1, c2$ を以下のように定義します。

- $b1$ = 'B' を含む文字列の個数
- $b2$ = 'B' からなる単体文字列の個数
- $c1$ = 'C' を含む文字列の個数
- $c2$ = 'C' からなる単体文字列の個数

もし、$b2 \leq c1$ かつ $c2 \leq b1$ ならば、すべての 'A' を使うとしてよいです。以下、一般性を失わずに $b2 > c1$ と仮定します。ある 'A' を使わないことにすると、その両隣の部分文字列を連結することになります。このとき、$b1, b2, c1, c2$ の値はそれぞれ高々 1 減ることになります。また、'B' からなる単体文字列に隣接する 'A'

を消した場合は、$b2$ は 1 減少し、$c1$ は変化しません。よって、'A' を削除する必要のある最小回数は $b2 - c1$ です。'A' の削除が終わった段階では、$b2 = c1$ なので、明らかに $c2 \leq b1$ は成立しています。よって、$b2 - c1$ 回 'A' を削除すれば条件を満たすことができ、またこれが条件を満たす最長の部分列を生成します。

あとは、以上の構築をそのまま実装すれば良いです。全体で $O(N)$ の時間でこの問題は解けます。

回答例

F : Square Constraints

包除原理を使います。各 k $(0 \leq k \leq N)$ について、次の問題が解ければいいです。

> k 個の添字 $0 \leq x_0 < x_1 < \cdots < x_{k-1} \leq N-1$ を選ぶ。それぞれの選び方について、$x_i^2 + P_{x_i}^2 < N^2$ (for all $0 \leq i \leq k-1$), $i^2 + P_i^2 \leq (2N)^2$ (for all $0 \leq i \leq 2N-1$) を満たす順列を数え、その総和を求める。

各 i $(0 \leq i \leq N-1)$ について、$f(i) = (a^2 + i^2 < N^2$を満たす最大の a $(a \leq 2N-1))+1$ と定義します。また、各 i $(0 \leq i \leq 2N-1)$ について、$g(i) = (a^2 + i^2 \leq (2N)^2$を満たす最大の a $(a \leq 2N-1))+1$ と定義します。

k 個の添字 x の選び方が決まっているとき、この問題は簡単です。各 i について、制約が $P_i < h(i) = f(i)$ or $g(i)$ とかけるとします。$h(i)$ を昇順にソートしたものを $h'_0, h'_1, \cdots, h'_{2N-1}$ とすると、条件を満たす順列の個数は $(h'_0 - 0) \times (h'_1 - 1) \times \cdots \times (h'_{2N-1} - (2N-1))$ です。

x が固定されていない場合について考えます。まず、$f(i)$ $(0 \leq i \leq N-1)$ と $g(i)$ $(0 \leq i \leq 2N-1)$ を全てまとめて昇順ソートした列を $A_0, A_1, \cdots, A_{3N-1}$ とします。A のそれぞれの要素に対し、それが $f(i)$ $(0 \leq i \leq N-1)$ 由来なら 'a'、$g(i)$ $(0 \leq i \leq N-1)$ 由来なら 'b'、$g(i)$ $(N \leq i \leq 2N-1)$ 由来なら 'c' というラベルをつけておきます。これらのラベルをならべてみると、'aaccacac...bbbb...' という風になっています（※この文字列は適当です)。正確に言えば、全ての 'b' の前に、'a' と 'c' のラベルが並ぶことになります。

各 i について、$f(i), g(i)$ が A の中で登場する位置を $fpos(i), gpos(i)$ とします。ここで、$f(i), g(i)$ がともに広義単調減少なので、$fpos(0) > fpos(1) > \cdots > fpos(N-1)$, $gpos(0) > gpos(1) > \cdots > gpos(N-1)$ となります。正確に言えば、A をソートする際にタイブレークを適切に行うとこのようになります。

よって、あとは以下のような問題が解ければよいです。

> 各 i $(0 \leq i \leq N-1)$ について、$A_{fpos(i)}$ もしくは $A_{gpos(i)}$ のどちらか一方のみを残し、長さ $2N$ の数列 $h'_0, h'_1, \cdots, h'_{2N-1}$ を作る。$A_{fpos(i)}$ を残した i の個数が k のもの全てについて、$(h'_0 - 0) \times (h'_1 - 1) \times \cdots \times (h'_{2N-1} - (2N-1))$ の値を計算し、その総和を求める。

これは、次のような DP で解けます。

- $dp[i][j] = A$ の先頭 i 項について残すかどうかを決定していて、j 項について、$A_{fpos(i)}$ を選ぶ選択をした場合の、$(h'_0 - 0) \times (h'_1 - 1) \times \cdots \times (h'_{2N-1} - (2N-1))$ の既に決定している部分の値の総和

$A_{fpos(i)}$ を残す個数が k であることから、$A_{gpos(i)}$ を選んだ場合に、その値が h' で先頭から何番目かを決定することができます。よって、この DP の遷移は $O(1)$ で行えることとなり、DP は全体で $O(N^2)$ でできます。

すべての k について同じ DP を行うので、この問題は全体で $O(N^3)$ で解けます。

回答例

AGC036 Editorial

writer : maroonrk

July 21, 2019

A : Triangle

Let us fix (X_1, Y_1) to $(0, 0)$. Then, the area of the triangle will be $|X_2 \times Y_3 - Y_2 \times X_3|/2$. Thus, we can solve the problem if we find X_2, Y_2, X_3, Y_3 such that $X_2 \times Y_3 - Y_2 \times X_3 = S$.

Here, let us also fix (X_2, Y_2) to $(10^9, 1)$. Now we just need to find X_3, Y_3 such that $10^9 \times Y_3 - X_3 = S$, which can be found from the quotient and remainder when dividing S by 10^9. For more detail, see the sample code below:

Link to sample code

B : Do Not Duplicate

Let us consider the minimum $i > 0$ such that $X_0 = X_i$. (There may be no such i, but for now we assume there is.) When we have processed $X_0, X_1, \cdots, X_i$, s will be empty. Then, let us consider the minimum $j > i + 1$ such that $X_{i+1} = X_j$. Again, when we have processed X_j, s will be empty.

Generally, for each i ($0 \leq i \leq N \times K - 1$), when the minimum $j > i$ such that $X_j = X_i$ exists, if s is empty just before X_i is processed, s will also be empty just after X_j is processed, or, just before X_{j+1} is processed. Let us define $y(i) =$ (the minimum $j > i$ such that $X_j = X_i$, or ∞ if such j does not exist)$+ 1 - i$. Then, the problem can be rephrased as follows:

- Initially, we have an integer $a = 0$.
- Step 1: As long as $a + y(a) \leq K$, we repeatedly assign $a := a + y(a)$.
- Step 2: Then, for each $i = a, a + 1, \cdots, N \times K - 1$, perform the operation in the statement.

We can easily verify that, when Step 1 is done, $N \times (K - 1) \leq a$ holds, so it is easy to execute Step 2.

Now, let us think of a way to run Step 1 fast. We can see that $y(i) = y(i \mod N)$ from the properties of X. Let us use the binary lifting method. For each pair of $k(0 \leq k \leq \log_2(N \times K))$ and $i(0 \leq i \leq N-1)$, let us define $g_k(i) =$ (The value of i when the operation of assigning $i := g(i) + i$ is performed 2^k times)$-i$, which will allow us to know the value of a when Step 1 is finished.

Step 1 takes $\mathrm{O}(N \log(NK))$ time to execute, and Step 2 takes $\mathrm{O}(N \log N))$ time, so we have solved the problem in a total of $\mathrm{O}(N \log(NK))$ time.

Link to sample code

C : GP 2

A sequence of non-negative integers $p_0, p_1, \cdots, p_{N-1}$ can occur as the result of M operations if and only if all the following conditions are satisfied:

- $p_0 + p_1 + \cdots + p_{N-1} = 3M$
- $\max(p_0, p_1, \cdots, p_{N-1}) \leq 2M$
- There are at most M values of i such that p_i is odd.

The necessity of these conditions is obvious, and we can show the sufficiency by induction. When $M = 1$, the conditions are obviously sufficient. Now, let us assume that the conditions are sufficient when $M = k$, and there is a sequence p satisfying the condition for the case $M = k + 1$. Then, if we choose two indices i, j properly and replace p_i with $p_i - 2$ and p_j with $p_j - 1$, we can obtain a sequence p satisfying the condition for the case $M = k$, which can be proved by cases. We will omit the detail, but we can prove it by paying attention to $\max(p_0, p_1, \cdots, p_{N-1})$ and the number of i such that p_i is odd.

Let us count the number of p satisfying the three conditions. For now, let us ignore the second condition and count it. Assume that there are a values of i such that p_i is odd. There are ${}_NC_a$ choices for such i. Then, we need to count the number of sequences of N even numbers totaling to $3M - a$, which is equal to the number of sequences of N integers totaling to $(3M - a)/2$. There are ${}_{(3M-a)/2+N-1}C_{N-1}$ such sequences. We can compute each binomial coefficient in O(1) time with proper pre-calculation. The pre-calculation takes $O(N + M)$ time, and there are $O(min(N, M))$ values of a, so we can find the answer in a total of $O(N + M)$ time.

We have solved the problem without the second condition, so what remains is to count the number of p violating the second condition. p totals to $3M$, so there is at most one value of i such that $p_i > 2M$. If we find the number of p such that $p_0 > 2M$, we can multiply it by N and find the answer. Thus, we want to solve the following problem:

> Count the number of sequences of N non-negative integers, q, that satisfy all of the following conditions:
>
> - $q_0 + q_1 + \cdots + q_{N-1} = M$
> - There are at most M values of i such that q_i is odd.
> - $q_0 > 0$

If we ignore the third condition here, the problem will be the same as the one we solved above and we can immediately solve it. Now, let us count the number of q violating the third condition. This is also equivalent to the problem above after replacing N with $N - 1$.

Therefore, the problem can be solved in a total of $O(N + M)$ time.

Link to sample code

D : Negative Cycle

Let us consider the situation where we have deleted some edges and have no negative cycle. Now, we can assign an integer p_i to each vertex so that the following condition is satisfied:

- For each edge $e = (i \to j)$, $p_j \leq p_i + weight(e)$ holds.

This can be achieved by, for example, letting $p_i =$ (the shortest distance from Vertex 0 to Vertex i). On the other hand, we can see that, if we can assign p_i to the vertices so that this condition is satisfied, the graph contains no negative cycle. (If we assume there is a negative cycle, adding $p_j - p_i \leq +weight(e)$ along it would lead to contradiction.)

Let us confirm what conditions p_i must satisfy. First, for each i $(0 \leq i \leq N-2)$, $p_i \geq p_{i+1}$ must hold, which corresponds to the originally present edge $(i \to i+1)$ of weight 0. Here, let us define $q_i = p_i - p_{i+1}$, which is a non-negative integer since $p_i \geq p_{i+1}$.

Then, considering the edge $(i \to j)$ of weight 1, $p_j \leq p_i + 1$ must hold, which can be represented as $q_j + q_{j+1} + \cdots + q_{i-1} \leq 1$ with q.

Similarly, considering the edge $(i \to j)$ of weight -1, $q_i + q_{i+1} + \cdots + q_{j-1} \geq 1$ must hold.

Let us consider the problem of whether we can set the values of q so that the condition is satisfied. Then, we can see that we only need to consider 0 and 1 as the elements of q.

Thus, what we have to do is to solve the following problem:

> We will set up a sequence q consisting of 0 and 1, and delete the edges whose conditions are not satisfied for this q. Minimize the total cost of deleting these edges.

We can solve it with the DP as follows:

- $dp[i][j]$ = the minimum total cost of the edges that are confirmed to be deleted when we have chosen the values from q_0 through q_i and the positions of the last 1 and the second last 1 are i and j, respectively.

Let us consider the transition $dp[i][j] \to dp[x][i]$. The additional edges of weight 1 we additionally need to delete are the edges $b \to a$ such that $j < a \leq i,\ x < b$, and the additional edges of weight -1 we additionally need to delete are the edges $a \to b$ such that $i < a < b \leq x$. With prefix sums, this transition can be calculated faster in $O(N^3)$ time, which is fast enough.

Link to sample code

E : ABC string

If S has adjacent equal characters, we can treat them as just one character. Thus, below we assume that any two adjacent characters in S are different.

Let a, b, c be the number of occurrences of `A`, `B`, `C` in S, respectively. Without loss of generality, we assume $a \leq b, a \leq c$.

Now, assuming that the set of `A`s to use is already fixed, let us determine if we can choose `B`s and `C`s to delete so that we can obtain a subsequence satisfying the condition.

First, let us delete all the unused `A`s. After this, the string may have adjacent equal characters, but we will combine each series as one character. Let t be the string obtained this way, and p, q, r be the number of occurrences of `A`, `B`, `C` in t, respectively. Each deletion of `A` reduces the number of `B`s or `C`s by at most one, so $p \leq q$ and $p \leq r$ hold.

If $q = r$ at this point, we can obtain a subsequence satisfying the condition by properly choosing `B`s and `C`s to delete. Specifically, we can repeatedly delete a substring `BC` or `CB` from the string as long as $p < q$ without making a pair of adjacent `A`s. This can be seen from the fact that: Consider the situation where the deletion of `BC` or `CB` is no longer possible. That is, the situation where any substring sandwiched between two `A`s has a length of at most 2, and there is at most one `B` or `C` at the beginning and the end of the string. In this situation, $2p \geq q + r$ already holds.

Let us consider the case $q \neq r$. We assume $q < r$ without loss of generality. t is divided into $p + 1$ substrings (some of them may be empty) by `A`s in it. Let us call such a substring of length 1 sandwiched between two `A`s (that is, not at the beginning or the end of t) a *unit string*. Among those $p+1$ substrings, let x be the number of strings containing `B`, and y be the number of unit strings consisting of `C`. If $x < y$, the number of `B`s and `C`s will never be equal after deleting `B`s and `C`s any number of times. On the other hand, if $x \geq y$, we can make the number of `B`s and `C`s equal by properly choosing the `C`s to delete. This can be seen from the fact that, for a string containing 'B' we can make the number of `B`s greater than the number of `C`s by 1, and for a unit string consisting of `C` we can only make the number of `C`s greater than the number of `B`s by 1.

Now, let us get back to the original problem. From the observations above, we can see that the problem is to choose which `A`s to use so that the substrings separated by `A`s satisfy the following conditions:

- the number of unit strings consisting of `B` is at most the number of strings containing `C`s.
- the number of unit strings consisting of `C` is at most the number of strings containing `B`s.

First, assume that we use all the `A`s, and define $b1, b2, c1, c2$ as follows:

- $b1$ = the number of strings containing `B`s
- $b2$ = the number of unit strings consisting of `B`
- $c1$ = the number of strings containing `C`s
- $c2$ = the number of unit strings consisting of `C`

If $b2 \leq c1$ and $c2 \leq b1$, we can go with all the `A`s used. Below, we assume $b2 > c1$ without loss of

generality. If we decide not to use a certain `A`, the two substrings around it will be concatenated into one string, reduing each of $b1, b2, c1, c2$ by at most 1. Also, when we delete an `A` adjacent to a unit string consisting of `B`, $b2$ decreases by 1 and $c1$ does not change. Thus, we have to delete an `A` at least $b2 - c1$ times, and after deleting an `A` as above $b2 - c1$ times, we have $b2 = c1$, from which $c2 \leq b1$ obviously holds. Therefore, we can satisfy the conditions by deleting an `A` $b2 - c1$ times, and this generates one longest subsequence that satisfies the condition.

By directly implementing these methods, we can solve the whole problem in $\mathrm{O}(N)$ time.

Link to sample code

F : Square Constraints

Let us use the inclusion-exclusion principle. For each k ($0 \leq k \leq N$), we want to solve the following problem:

> We will choose k indices $0 \leq x_0 < x_1 < \cdots < x_{k-1} \leq N-1$. For each possible choice, count the number of permutations satisfying $x_i^2 + P_{x_i}^2 < N^2$ for all $0 \leq i \leq k-1$ and $i^2 + P_i^2 \leq (2N)^2$ for all $0 \leq i \leq 2N-1$, and find the sum of these counts for all possible choices.

For each i ($0 \leq i \leq N-1$), let us define $f(i) =$ (the maximum a ($a \leq 2N-1$) such that $a^2+i^2 < N^2$)$+1$. Also, for each i ($0 \leq i \leq 2N-1$), let us define $g(i) =$ (the maximum a ($a \leq 2N-1$) such that $a^2+i^2 < (2N)^2$) $+1$.

This problem would be easily solved if the choice of x, the k indices, is fixed. For each i, assume that the condition can be written as $P_i < h(i) = f(i)$ or $g(i)$. Let $h'_0, h'_1, \cdots, h'_{2N-1}$ be the sorted list of $h(i)$ in ascending order. Then, the number of permutations satisfying the condition is $(h'_0 - 0) \times (h'_1 - 1) \times \cdots \times (h'_{2N-1} - (2N-1))$.

Now, let us consider the problem when x is not fixed. First, let $A_0, A_1, \cdots, A_{3N-1}$ be the sorted list of $f(i)$ ($0 \leq i \leq N-1$) and $g(i)$ ($0 \leq i \leq 2N-1$) all together in ascending order. We assign a label to each element of A as follows: `a` if it derives from $f(i)$ ($0 \leq i \leq N-1$), `b` if it derives from $g(i)$ ($0 \leq i \leq N-1$), and `c` if it derives from $g(i)$ ($N \leq i \leq 2N-1$). As a whole, the labels assigned to the elements will look as this: `aaccacac...bbbb...` (this is just an illustration). More formally, all `a`s and `c`s will come before all `b`s. For each i, let $fpos(i), gpos(i)$ be the positions of $f(i), g(i)$ in A, respectively. Since $f(i), g(i)$ are both non-increasing, we have $fpos(0) > fpos(1) > \cdots > fpos(N-1)$, $gpos(0) > gpos(1) > \cdots > gpos(N-1)$ (more strictly, these will hold if we properly do tie-breaks).

What remains is to solve the following problem:

> For each i ($0 \leq i \leq N-1$), we keep either $A_{fpos(i)}$ or $A_{gpos(i)}$ to make a sequence $h'_0, h'_1, \cdots, h'_{2N-1}$ of length $2N$. For each sequence that can be obtained this way when we choose to keep $A_{fpos(i)}$ for k of the indices i, find the value $(h'_0 - 0) \times (h'_1 - 1) \times \cdots \times (h'_{2N-1} - (2N-1))$, and compute the sum of these values for all such sequences.

We can solve it with the DP as follows:

- $dp[i][j]$ = the sum of the already fixed parts in $(h'_0 - 0) \times (h'_1 - 1) \times \cdots \times (h'_{2N-1} - (2N-1))$ for all possible cases when we have decided whether we keep each of the first i elements in A and we have decided to keep $A_{fpos(i)}$ for j indices.

Since we will keep k of the elements $A_{fpos(i)}$, when we choose to keep $A_{gpos(i)}$, we can determine its position in h'. Thus, we can calculate each transition in this DP in $O(1)$ time, for a total of $O(N^2)$ time.

The whole problem can be solved in $O(N^3)$ time by performing this DP for each value of k.

Link to sample code