AGC 043 解説

yosupo, maroonrk, sigma425

2020 年 3 月 21 日

For International Readers: English editorial starts on page 8.

A: Range Flip Find Route

経路を一つ決めたときにその経路を通れるようにするには何回操作する必要があるかを考えます。

経路において、白いマスから黒いマスに移動する回数に、s_{11} が黒の時のみ 1 を加えた値を Z とします。実はこの経路を通れるようにするための最小の操作回数は Z です。

操作を行ったときに、経路のマスの色がどのように変化するかを考えます。これは、全く変化しないか、もしくは経路のある区間の色が反転することがわかります。また、経路の全ての区間について、そこを反転させるような操作が可能であることもわかります。

以上の観察により、経路の色の列の区間を好きに反転させられるとき、全てのマスを白色にするには何回操作を行う必要があるか、を考えればいいことがわかります。そして、どのような操作をしても Z が必ず 1 しか減らせないこと、また、Z を 1 減らすような操作が可能であることに気づくと、最小回数が Z であることがわかります。

後は Z が最小の経路を探せばよく、DP を行うことでこの問題の答えを得ることができます。

B: 123 Triangle

まず入力から 1 を引いて $a_i = 0, 1, 2$ としてもかまいません。$x_{i,j}$ はすべて $0, 1, 2$ のいずれかになります。まず答えの偶奇を考えてみます。すると $x_{i,j}$ は $x_{i-1,j}$ と $x_{i-1,j+1}$ のちょうどどちらか一方が奇数のときに奇数になることがわかります。よって次の問題が解ければ答えの偶奇がわかります。

> 各要素が $0, 1$ である $x_{1,1}, \ldots, x_{1,N}$ が与えられます。$x_{i,j}$ を次のように定義します。
> $x_{i,j} = x_{i-1,j} \hat{} x_{i-1,j+1}$
> $x_{N,1}$ を求めてください。ただし $\hat{}$ は排他的論理和を表します。

これは次のようにして解けます。まずパスカルの三角形のようなものを考えると、a_i が答えへ寄与する回数は $\binom{N-1}{i}$ であることがわかるので、この偶奇が全て求まればよいです。これは Lucas の定理から求めたり、二項係数の式に従って 2 で何回割れるかを求めたりすることで可能です。

これで答えが 1 のケースは特定できました。0 と 2 を区別するために次のことを考えます。

- 入力に ($0, 1, 2$ のうち) 1 が存在するか？

存在する場合、答えは 2 にならないことが示せます。よって、次のような手順で問題を解くことができます。

1. 答えが奇数かどうか判定する。 奇数なら答えは 1。
2. 入力に 1 が含まれるか判定する。含まれるなら答えは 0
3. そうでない場合、入力は 0 か 2 なので、全体を 2 で割ると前のケースに帰着できる。

計算量は $O(N)$ です。

C: Cubic Graph

価値の決め方から、価値の大きい頂点から順番に見ていき、独立点集合に加えられるならば加える、という貪欲が回ることがわかります。つまり、

$$\mathrm{dp0}[i,j,k] := (x_i, y_j, z_k) \text{ を独立点集合に加えられるか？}$$

という DP で $O((N+M)^3)$ で解けます。なお、$M = \max(M_1, M_2, M_3)$ とします。

次に、以下の二人ゲームを考えます

> X, Y, Z のそれぞれどこかの頂点に、コマが 1 つずつ置いてある。2 人が交互に操作行い、操作を行えなくなった方が負けである。最善を尽くしたら先手と後手、どちらが勝つか？
>
> - 3 つのコマから一つ選ぶ。そのコマを隣接している、より頂点番号の大きい頂点に動かす

この問題は

$$\mathrm{dp1}[i,j,k] := \text{コマが } x_i, y_j, z_k \text{にある状態でスタートしたら後手が勝つか？}$$

という DP で $O((N+M)^3)$ で解けます。

ここで、$\mathrm{dp0}, \mathrm{dp1}$ に注目すると、漸化式と初期状態が全く同じであることに気づくと思います。なので当然内容も全く同じです。

各頂点に grundy 数を割り振ります。つまり各頂点に、その頂点と隣接している、より頂点番号の大きい頂点に書かれていないような最小の正整数を書きます。すると $\mathrm{dp1}[i,j,k] = \mathrm{true}$ と $\mathrm{grundy}(x_i) \oplus \mathrm{grundy}(y_j) \oplus \mathrm{grundy}(z_k) = 0$ が同値となることが知られています。

grundy の性質より、grundy 数は高々 $O(\sqrt{M})$ です。よって、$\mathrm{grundy}(x_i)$, $\mathrm{grundy}(y_i)$, $\mathrm{grundy}(z_k)$ を決め打つことで $O(N+M)$ でこの問題は解けました。

実装例

D: Merge Triplets

まず数列たち $A_1, A_2, \ldots, A_N$ が与えられたとき、最終的な順列 P がどのようになるか考えます。ある一つの数列 $x = (x_1, x_2, x_3)$ に着目したとき、もし $x_i > x_{i+1}$ なら、 x_i が選ばれた直後に x_{i+1} も選ばれることがわかります。したがって、x を前から順番に見ていき、これまでで最も大きい値が出てきたらその手前に区切りを入れる、という操作を行うと、区切られた各ブロックは連続して選ばれます。

全体の中で選ばれるタイミングはブロックの先頭の値によって決まり、ひとつの数列の中のブロックの先頭の値は前から単調に増加するので、最終的にできる数列はブロックの先頭の値でブロックをソートして並べたものになります。

逆に最終的な列 P からも同様にブロックは一意に定まり、それらを $A_1, \ldots, A_N$ にどう割り振るかを考えると、順列 P が作れる条件は次のようになります。

> P を上述の方法でブロックに分割する。これらのブロックを $A_1, \ldots, A_N$ に割り振って、それぞれの数列の長さを 3 にできる。

これは次と同値です。

> P を上述の方法でブロックに分割する。これらのブロックのサイズを考えたとき、
>
> - すべて 3 以下
> - 2 の個数 が 1 の個数以下

ブロックのサイズが前から順番に $a_1, \ldots, a_K$ だったとすると、これから作れる順列の個数は $N!/(a_1 * (a_1 + a_2) * \cdots * (a_1 + \cdots + a_K))$ となるので、うまく遷移することで次のような状態の DP が可能です。

> $dp[s][d]$ = 前から見てサイズの和が s で (1 の個数 - 2 の個数) が d であるようなすべての $a_1, \ldots, a_k$ に対する、$N!/(a_1 * (a_1 + a_2) * \cdots * (a_1 + \cdots + a_k))$ の総和。

s は 0 から $3N$, d は $-3N$ から $3N$ を渡り、遷移は $O(1)$ なので、全体の計算量は $O(N^2)$ です。

E: Topology

説明のため、 $x = \sum_{i \in S} 2^i$ のとき、 A_x を A_S とも書くことにします。まず明らかな必要条件として、$A_S = 1$ かつ $T \subset S$ なら、 $A_T = 1$ である必要があります。逆に、これを満たせば構成可能であることを示します。

まずジャッジ方法について説明します。本質的に同じなので $S = \{0, 1, \ldots, N-1\}$ の場合を説明します。$(i + 0.5, 0.5)$ のことを点 i と呼ぶことにします。

閉曲線 C から、次のように文字列 f_C を生成します。

- 閉曲線をなぞっていって、点 i から上向きに伸ばした半直線を横切ったら文字 u_i を , 下向きに伸ばした半直線を横切ったら文字 d_i を f_C の末尾に追加。

例えばサンプル 2 の出力例に対しては、 $d_1u_1u_0d_0$ という文字列ができます。

すると、この文字列が次の条件を満たすことと、$A_S = 1$ であることが同値であることが示せます。

> 次の操作を繰り返して、文字列を空にできる。
>
> - 隣り合う二つの文字が同じなら、その二つを同時に消す。

以降この文字列の形で閉曲線を表します。

次に、入力が $11\ldots110$ すなわち、すべての点が通れないときのみ閉曲線をうまく $y < 0$ に持っていけないケースを構成します。これは再帰的に構成できます。この文字列を a_N とおきます。

$N = 1$ の場合は $a_1 = u_0d_0$ でよいです。

$N \geq 2$ の場合は、まず a_{N-1} のすべての文字のインデックスを 1 増やした文字列を s として、次のように構成できます。

- $a_N = u_0 s u_0 d_0 s^{-1} d_0$

ただし s^{-1} は s を文字列として反転させたものとします。

例えばはじめのいくつかの a_N は次のようになります。

- $a_1 = u_0d_0$
- $a_2 = u_0(u_1d_1)u_0d_0(d_1u_1)d_0$
- $a_3 = u_0(u_1u_2d_2u_1d_1d_2u_2d_1)u_0d_0(d_1u_2d_2d_1u_1d_2u_2u_1)d_0$

a_N が条件を満たすことを証明します。

> 帰納法で示す。 $N = 1$ は明らか。$N \geq 2$ とする。まず $S = \{0, 1, \ldots, N-1\}$ のときは、$A_S = 0$ となる。なぜなら、a_N に同じ文字が連続することがないため。次に S がそれ以外の場合に $A_S = 1$ となることを示す。$0 \notin S$ の場合、文字 u_0, d_0 を無視することになるので、残った s と s^{-1} が打ち消し合って消えるのでよい。そうでない場合、ある $i (1 \leq i < N)$ が存在して $i \notin S$ となるので、帰納法の仮定から s (と s^{-1}) は空文字列にできる。従って残る文字列は $u_0u_0d_0d_0$ となり、これは明らかに空文字列にできる。よって示された。

最後に一般のケースについて話します。$A_S = 0$ となる S のうち極小なものをすべて列挙します。そしてそれぞれの S について上述のように閉曲線を作り、それらをつなげばよいです。(例えば、常に $(0,0)$ からスタートする閉曲線にしておけば簡単につなげます)

正当性は次のように示せます。

まず $A_S = 1$ の場合はつないだすべての閉曲線が空文字列にできるので、全体も空文字列にできます。$A_S = 0$ の場合は、まずこのうち極小な S について空文字列にできないことを示せば十分です。実際、S が極小なら S に対応して作った閉曲線以外は空文字列になって、S に対応して作った閉曲線のみが残るので、これは空文字列にはなりません。

サイクルの長さの制限はかなり緩めに設定しています。構成方法にもよりますが、かなり雑に見積もっても $S \subseteq \{0, 1, \ldots, N-1\}$ に対応する閉曲線の長さが $4N \cdot 2^{|S|}$ で抑えられて、これをすべての S について足し合わせても $4N \cdot 3^N$ となり、これは十分小さいです。

F: Jewelry Box

クエリが 1 つで，ちょうど A 個の宝石箱を準備する問題だとします．一般性を失わず，$S_{i,1} \leq S_{i,2} \leq \cdots \leq S_{i,K_i}$ と仮定します．

宝石 $(i,1),(i,2),\cdots,(i,j-1)$ を買った個数の合計を $x(i,j)$ $(1 \leq i \leq N,\ 1 \leq j \leq K_i+1)$ と表すことを考えます．ここで，$x(i,j)$ は，以下の条件を満たす必要があります．

- $x(i,j)$ は整数
- $0 = x(i,1) \leq x(i,2) \leq \cdots \leq x(i,K_i+1) = A$
- $x(i,j+1) - x(i,j) \leq C_{i,j}$
- 全ての制約 U_i, V_i, W_i と，任意の宝石 (V_i, j) について考える．ここで，$S_{U_i,k} \geq S_{V_i,j} - W_i$ を満たす最小の k を考える．そのような k が存在しない場合は $k = K_{U_i}+1$ とする．このとき，宝石 $(V_i,j),(V_i,j+1),\cdots,(V_i,K_{V_i})$ と同時に使うことができる宝石店 U_i の宝石は，宝石 $(U_i,k),(U_i,k+1),\cdots,(U_i,K_{U_i})$ である．よって，$x(U_i,k) \leq x(V_i,j)$.

実は，これらの条件は十分条件でもあります．この事実をまず証明します．

まず，使う宝石を決めた場合，それらを箱に分ける際には，各宝石店の宝石を昇順に並べ，最もサイズの小さい宝石を入れた箱，2 番めに小さい宝石を入れた箱…と作っていくのが最適です．これは次のように証明できます．もしこの方法で構成できなかったとすると，ある制約 (U_i, V_i, W_i) があって，これがどこかで満たされていないということです．しかし，宝石店 U_i, V_i の宝石とこの制約だけが存在するとした場合の最適戦略は，宝石をサイズでソートすることなので，結局，この制約は絶対に満たすことができないことがわかります．

以上の手順で z 番目に作られた宝石箱について考えます．各 i について，$x(i,j(i)) < z$ になる最大の $j(i)$ を考えると，この宝石箱には，宝石 $(1,j(1)),(2,j(2)),\cdots,(N,j(N))$ が入ることになります．ここで，ある制約 (U_i, V_i, W_i) が満たされていなかったと仮定します．すると，$S_{U_i,j(U_i)} < S_{V_i,j(V_i)} - W_i$ となります．ここで，上記のうち 4 番目の条件を考えると，ある k が存在し，$j(U_i) < k$, $x(U_i,k) \leq x(V_i,j(V_i))$ となります．$x(V_i,j(V_i)) < z$ より，$x(U_i,k) < z$ となります．しかしこれは，$j(U_i)$ の取り方に矛盾します．よって，満たされない制約は存在しないことがわかります．

$x(i,j)$ を定めてしまえば，その宝石を買うためのコストは，$\sum P_{i,j} \times (x(i,j+1) - x(i,j))$ と計算できます．

ここで，今まで出てきた $x(i,j)$ の制約と目的関数が全て一次式であることに注目すると，この問題を整数計画問題として定式化できます．そしてこの問題は，緩和して LP にしても，最適解が変わりません．詳細は省略しますが，このことはいくつかの方法で証明できます．例えば，後述するアルゴリズムを拡張して実際に LP の解を復元すると，整数になっていることが確認できます．また，LP の行列が totally unimodular であることを直接示すこともできます．

LP を解きます．まず，$foo = bar$ の形の制約は，$foo \leq bar$, $-foo \leq -bar$ として不等式に分解します．また，$x(i,j) \leq baz$ という形の制約は，ダミー変数 $D = 0$ を用い，$x(i,j) - D \leq baz$ と変形します．すると，今までの制約は全て，$p - q \leq r$ という形の不等式になります．

この LP の双対問題を考えます．するとこれは，**最大**費用流になっていることがわかります．目的関数の符号を反転させて，最小費用流として考えます．

この双対問題では，主問題の変数がグラフの頂点に対応しています．そこで，変数 $x(i,j), D$ に対応する頂点をそれぞれ，頂点 $x(i,j), D$ と呼ぶことにします．この最小費用流では，各 $x(i,j)$ に対して，そこから流

れ出る流量が $P_{i,j-1} - P_{i,j}$ として定まっています．(なお，$P_{i,0} = P_{i,K_i+1} = 0$ です) ここで，今考えているグラフに対し，頂点 $x(i,j)$ から $x(i,j+1)$ へ向かう容量 $P_{i,j}$，コスト 0 の辺を追加したとします．追加後のグラフにおいて，各頂点から流れ出る流量が 0 の最小費用循環流を考えると，これは元の問題と同じ答えになることがわかります．以降はこのグラフに対して，各点から流れ出る流量が 0 の問題を解いていきます．

各点から流れ出る流量が全て 0 なので，最終的な循環流は，負閉路の集まりになっているはずです．ここで，グラフの辺のコストに注目すると，そもそもコストが負の辺は，頂点 $x(i,K_i+1)$ から D へ向かうコスト $-A$ の辺しかありません．D から $x(i,K_i+1)$ へ向かうコスト A の辺は絶対に使う必要がないことに注意すると，この最小費用循環流は，次のようなアルゴリズムで解けることになります．

> 各 $f = 0,1,2,\cdots$ について，D からいずれかの $x(i,K_i+1)$ へ向かう容量 f の最小費用流を求める．このコストを $mcf(f)$ としたとき，$mcf(f) - Af$ の最小値が答えになる．

$mcf(f)$ の値は，primal-dual 法で求めることができます．またここで，$mcf(f+1) - mcf(f) \leq mcf(f+2) - mcf(f+1)$ が成り立っています．よって，$mcf(f+1) - mcf(f) \geq A$ が成立する最小の f が求まればよいです．なお，$f \to \infty$ でも $mcf(f+1) - mcf(f) < A$ となるときは，いくらでも解が小さくなるということになり，これは主問題における解が存在しないことを意味します．

以上で，クエリが 1 つのときの問題は解けました．$mcf(f)$ の値がクエリに寄らないことに注意すると，この問題を複数のクエリについて答えるのも容易です．

計算量について考えます．K_i の最大値を $Kmax$，$P_{i,j}$ の最大値を $Pmax$ とおきます．まず最小費用流の計算量を考えます．グラフの形から，$mcf(f+1) - mcf(f) < mcf(f+2) - mcf(f+1)$ が成立する最大の f は高々 $\sum P_{i,j} = O(NKmaxPmax)$ であることがわかるので，primal-dual 法で最短路を求める回数は，最大でもこの値以下です．グラフの頂点数は $O(NKmax)$，辺数は $O(MKmax)$ なので，最小費用流の計算量は $O(NKmaxPmax \times SP(NKmax, MKmax))$ になります．ただしここで，$SP(V,E)$ は，頂点数 V，辺数 E のグラフの最短路を求めるのにかかる計算量です．普通の dijkstra 法を使えば，$SP(V,E) = O(V + ElogE)$ となり，十分高速です．クエリに関しては，二分探索をするだけなので，1 クエリあたり $O(log(NKmaxPmax))$ で処理できます．

writer は，最小費用流の流量が $O(NKmaxPmax)$ になるケースを用意しましたが，実際に最短路を求める回数が $O(NKmaxPmax)$ になるケースは生成できていません．これについてアイディアのある方がいれば，コメント等で教えてください．

解答例 (グラフの形状が解説とは少し異なります)

A: Range Flip Find Route

Let's fix a certain path from the top-left corner to the bottom-right corner, and compute the minimum number of operations required to make all squares on this path white.

Let's define the value Z as follows: Consider a sequence of 'black' and 'white' obtained by concatenating the colors of all squares on the path (call it S). The value Z is defined as the number of black runs in this sequence. It turns out that the minimum number of required operations is Z.

This is because:

- In each operation, we flip a continuous subsequence of S. Thus, Z decreases by at most 1.
- For any non-empty continuous subsequence of S, we can flip the subsequence by choosing an operation appropriately. Thus, if $Z > 0$, we can always decrease it by 1.

Thus, the answer is the minimum possible value of Z when we choose the path optimally, and this can be computed by a simple DP.

B: 123 Triangle

First, we can assume that the inputs are $0, 1, 2$, not $1, 2, 3$ by subtracting one. The answer will also be $0, 1, 2$. Let's consider the parity of the answer. $x_{i,j}$ is odd if and only if exactly one of $x_{i-1,j}$ and $x_{i-1,j+1}$ is odd, so we consider the following problem.

> Given are $x_{1,1}, \ldots, x_{1,N}$, where each element is either 0 or 1. Let $x_{i,j}$ be defined as follows:
> $x_{i,j} = x_{i-1,j} \hat{} x_{i-1,j+1}$
> Find $x_{N,1}$. Where $\hat{}$ denotes exclusive-or.

This problem can be solved as follows. By analogy with Pascal's triangle, the number of times a_i contributes to the answer is $\binom{N-1}{i}$. So what we have to do is find the parity of $\binom{n-1}{i}$ for all i, this can be done by Lucas's theorem or just computing how many times the binomial coefficients can be divided by 2 according to the definition.

Now we have identified the case with the answer 1. To distinguish between 0 and 2, consider the following:

- Is there 1 (of $0, 1, 2$) in the input?

It can be proved that if 1 exists, then the answer cannot be 2. Therefore, you can solve the problem using the following procedure.

1. Determine if the answer is odd. If so, the answer is 1.
2. Determine if the input contains 1. If so, the answer is 0
3. Otherwise, the input is either 0 or 2, so dividing all a_i by 2 can reduce the problem to the previous case.

The time complexity is $O(N)$.

C: Cubic Graph

By the definition of the weights of vertices, we can get the following greedy algorithm easily: we check vertices one by one in the decreasing order of weights and add it to the independent set if we can add it. In other words, we can solve this problem in $O((N+M)^3)$ time ($M = \max(M_1, M_2, M_3)$) by the following DP:

$$\mathrm{dp0}[i,j,k] := \text{Is } (x_i, y_j, z_k) \text{ contained in the independent set?}$$

Next, we consider the following two-player game.

> There are three graphs X, Y, Z. Initially each graph contains one token (on some vertex). Alice and Bob take turns alternately and performs the following move. The (current) player loses when he cannot move.
>
> - Choose one of the three tokens arbitrary, and move it to an adjacent vertex that has an index higher than that of the current vertex.

This problem can be solved in $O((N+M)^3)$ time by the following DP:

$$\mathrm{dp1}[i,j,k] := \text{Does the current player lose when tokens are on the vertices } x_i, y_j, \text{ and } z_k?$$

We can notice that the recurrence formula and initial values of dp0 and dp1 are completely the same. Therefore we can focus on dp1 instead of dp0.

We write grundy numbers on each vertex. In other words, we write the minimum positive integer that is not written on any adjacent vertices.

It is known that $\mathrm{dp1}[i,j,k] = \text{true}$ iff $\mathrm{grundy}(x_i) \oplus \mathrm{grundy}(y_j) \oplus \mathrm{grundy}(z_k) = 0$.

The upper bound of grundy numbers is $O(\sqrt{M})$, therefore we can solve this problem in $O(N+M)$ time by fixing $\mathrm{grundy}(x_i)$, $\mathrm{grundy}(y_i)$, and $\mathrm{grundy}(z_k)$.

Implementation Example

D: Merge Triplets

First, let's assume the sequences $A_1, A_2, \ldots, A_N$ are given and consider what the final permutation P looks like. Focus on one sequence $x = (x_1, x_2, x_3)$. If $x_i > x_{i+1}$, we know that x_{i+1} is selected immediately after x_i is selected. Therefore, look at x in order from the front, and put a break before x_i if x_i is largest among before it. Then each block is selected all at once.

When a block is chosen is determined by the value of the beginning of the block, and the value of the beginning of the block in one sequence increases monotonously from the front. Thus, the resulting sequence P is a sequence of blocks sorted by the first value.

Conversely, the blocks are uniquely determined from P. Considering how to allocate them to $A_1, \ldots, A_N$, we can show that the condition P can be created is as follows.

> Divide P into blocks as described above. We can allocate these blocks to $A_1, \ldots, A_N$ and make each sequence length to 3.

This is equivalent to the following:

> Divide P into blocks as described above. The size of these blocks satisfy the following conditions.
>
> - Each of them is not more than 3.
> - the number of 2 is less than or equal to the number of 1

Assume that the size of the blocks are $a_1, \ldots, a_K$ from the beginning. Then the number of permutations that can be created from this is $N!/(a_1 * (a_1 + a_2) * \cdots * (a_1 + \cdots + a_K))$. So the following DP is possible.

> $dp[s][d]$: Consider all $a_1, \ldots, a_k$ with $\sum a_i$ is s and (# of 1 - # of 2) is d. $dp[s][d]$ is the sum of $N!/(a_1 * (a_1 + a_2) * \cdots * (a_1 + \cdots + a_k))$ for all such a.

s can be between 0 and $3N$, d can be between $-3N$ and $3N$, and the transition is $O(1)$ time, so the whole complexity is $O(N^2)$.

E: Topology

For simplicity, we will write A_x as A_S under $x = \sum_{i \in S} 2^i$. The first obvious requirement is that if $A_S = 1$ and $T \subset S$, then $A_T = 1$ should hold. Conversely, this is the sufficient condition: we show how to construct the answer from now.

First, we will explain how to judge whether the output is correct. Because it is essentially the same, we explain the case with $S = \{0, 1, \ldots, N-1\}$.

We call $(i + 0.5, 0.5)$ the point i.

From the closed curve C, generate the string f_C as follows:

- Follow the closed curve. When crossing the half line extending upward from the point i, the character u_i is added to the end of f_C. Similarly, the character d_i is added to the end of f_C when crossing the half line extending downward from the point i.

For example, for the output of sample 2, f_C is $d_1u_1u_0d_0$.

Then we can prove the equivalence between that this string satisfies the following condition and that $A_S = 1$

> You can make the string empty by repeating the following:
>
> - If two adjacent characters are the same, erase the two at the same time.

Hereafter, we represent a closed curve in the form of this string.

Next, we construct a case where the input is $11\ldots110$: Only when all points cannot pass, the curve cannot be moved to $y < 0$ completely. This can be constructed recursively. Let's call this string a_N.

For $N = 1$, $a_1 = u_0d_0$.

For the case with $N \geq 2$, let s be the string a_{N-1} with all characters indexed up by 1. Then,

- $a_N = u_0su_0d_0s^{-1}d_0$

Where s^{-1} is the inverse of s as a string.

For example, the first few a_N are:

- $a_1 = u_0d_0$
- $a_2 = u_0(u_1d_1)u_0d_0(d_1u_1)d_0$
- $a_3 = u_0(u_1u_2d_2u_1d_1d_2u_2d_1)u_0d_0(d_1u_2d_2d_1u_1d_2u_2u_1)d_0$

Let's prove that a_N satisfies the conditions.

Prove by induction. $N = 1$ is obvious. Assume $N \geq 2$.
First, for $S = \{0, 1, \ldots, N-1\}$, $A_S = 0$. This is because a_N does not have the same character in a row. Next we prove that $A_S = 1$ otherwise.
In the case of $0 \notin S$, the characters u_0, d_0 should be ignored, so the remaining parts (s and s^{-1}) are canceled out. Otherwise, there exists some $i (1 \leq i < N)$ such that $i \notin S$. Then by the induction hypothesis s (and s^{-1}) can become an empty string. The remaining string is $u_0 u_0 d_0 d_0$, which can obviously be an empty string. QED.

Finally, let's consider about general cases. Enumerate all minimal S among $A_S = 0$. Then create a closed curve for each S as described above and "connect" them. For example, a closed curve that always starts from $(0, 0)$ can be easily connected as string.

The validity can be shown as follows.

First, for cases with $A_S = 1$, all closed curves can be empty strings, so the whole curve can also be an empty string. Next, for cases with $A_S = 0$, it is sufficient to show that S cannot be an empty string for minimal Ss among them. Indeed, if S is minimal, it will not be an empty string bccause only the closed curve created for S remains and other curves becomes empty strings.

The limit for cycle length is not so tight. It depends on the construction method, but the length of the closed curve corresponding to $S \subseteq \{0, 1, \ldots, N-1\}$ is less than $4N \cdot 2^{|S|}$. Summing up this for all S, we get the upper bound $4N \cdot 3^N$, which is small enough.

F: Jewelry Box

Assume there is only one query, and we want to prepare A jewelry boxes. WLOG $S_{i,1} \leq S_{i,2} \leq \cdots \leq S_{i,K_i}$.

Let $x(i,j)$ $(1 \leq i \leq N, 1 \leq j \leq K_i+1)$ be the total number of purchased jewelries $(i,1),(i,2),\cdots,(i,j-1)$. Here, $x(i,j)$ must satisfy the following conditions:

- $x(i,j)$ is an integer
- $0 = x(i,1) \leq x(i,2) \leq \cdots \leq x(i,K_i+1) = A$
- $x(i,j+1) - x(i,j) \leq C_{i,j}$
- Let's consider a constraint U_i, V_i, W_i and a jewel (V_i, j). Then let k be the smallest number such that $S_{U_i,k} \geq S_{V_i,j} - W_i$. If no such number exists, let k be $K_{U_i}+1$. What jewelry from Shop U_i can be purchased with one of jewelries $(V_i,j),(V_i,j+1),\cdots,(V_i,K_{V_i})$? It's one of jewelries $(U_i,k),(U_i,k+1),\cdots,(U_i,K_{U_i})$. This implies that $x(U_i,k) \leq x(V_i,j)$.

These are necessary conditions, but they are also sufficient! We will prove this fact.

First, if you decide a set of jewels to purchase, the optimal strategy to distribute them into boxes is to combine the smallest jewels from each shop, combine the second smallest, third, and so on. It's easy to prove this.

Consider the z-th jewelry box made in the above procedure. For each i, let $j(i)$ be the largest number such that $x(i,j(i)) < z$. Then, the z-th jewelry box contains jewelries $(1,j(1)),(2,j(2)),\cdots,(N,j(N))$. Now, suppose a constraint (U_i,V_i,W_i) is violated. This means $S_{U_i,j(U_i)} < S_{V_i,j(V_i)} - W_i$. However, considering the 4-th condition, there exists k such that $j(U_i) < k$ and $x(U_i,k) \leq x(V_i,j(V_i))$. Since $x(V_i,j(V_i)) < z$ we get $x(U_i,k) < z$, but this contradicts the maximality of $j(U_i)$. So there is no violated constraint.

If you decide $x(i,j)$, the total cost can be calculated as $\sum P_{i,j} \times (x(i,j+1) - x(i,j))$.

Now, all the constraints and the cost is linear to $x(i,j)$, so we can formulate the problem as integer programming. And we can prove that the relaxation to LP doesn't change the answer. We will omit the proof, but there are a few approaches. For example, you can indeed restore an integral answer to the LP by modifying the algorithm below. Also, we can directly prove that the constraint matrix is totally unimodular.

Let's solve the LP. First, equality constrains like $foo = bar$ can be reduced to $foo \leq bar$ and $-foo \leq -bar$. Also, we prepare dummy variable D and transform constraints like $x(i,j) \leq baz$ to $x(i,j) - D \leq baz$. Now, all the constraints are of the form $p - q \leq r$.

Take the dual of the LP. One can see that this dual problem is **maximum** cost flow. Let's negate the goal value and solve the minimum cost flow.

In this dual problem, variables in the primal problem correspond to the vertices in the graph. So we will call the vertices $x(i,j)$ or D according to the correspondant variables. In this minimum cost flow problem, each vertex $x(i,j)$ has a fixed flux of $P_{i,j-1} - P_{i,j}$. Consider adding edge from $x(i,j)$ to $x(i,j+1)$, with capacity $P_{i,j}$ and cost 0. After this addition, we need to solve the minimum cost

circulation (with each vertex having the flux of 0). You can check this doesn't change the answer.

What we want to calculate is a set of negative cycles. However, you can note that edges with negative costs are those edges from one of $x(i, K_i + 1)$ to D. Since the edges from D to $x(i, K_i + 1)$ with cost A will never be used in a negative cycle, we can solve the problem by the following algorithm:

> For each $f = 0, 1, 2, \cdots$, let $mcf(f)$ be the minimum cost of flow of capacity A from D to one of $x(i, K_i + 1)$. Then, the answer is the minimum value of $mcf(f) - Af$.

We can calculate $mcf(f)$ by primal-dual method. Because of the convexity of $mcf(f)$, we just need to know the minimum f such that $mcf(f+1)-mcf(f) \geq A$. Note that if $f \to \infty$ and $mcf(f+1)-mcf(f) < A$, this means A can go $-\infty$ and the primal problem has no feasible solution.

Now we solved the problem with one query. Since $mcf(f)$ doesn't change between queries, it's easy to answer many queries.

What is the time complexity? Let $Kmax$ be the maximum of K_i and $Pmax$ be the maximum of $P_{i,j}$. We can notice that maximum f such that $mcf(f + 1) - mcf(f) < mcf(f + 2) - mcf(f + 1)$ is at most $\sum P_{i,j} = O(NKmaxPmax)$, so the number of shortest path algorithms is $O(NKmaxPmax)$. The graph has $O(NKmax)$ vertices and $O(MKmax)$ edges, so the complexity of the (ordinary) mincost flow algiorithm is $O(NKmaxPmax \times MinDist(NKmax, MKmax))$. Here, $SP(V, E)$ is the time complexity of the shortest path algorithm. If you run dijkstra, $SP = O(V + ElogE)$ and it is fast enough. Each query requires a simple binary search and takes only $O(log(NKmaxPmax))$ time.

The author of this tasks developed a test case where the capacity of the min-cost-flow is $O(NKmaxPmax)$, but couldn't generate a test where one needs to run the shortest path algorithm $O(NKmaxPmax)$ times. If you have any ideas, please share!

authors solution(the graph is a bit different from what explained in this editorial)