AtCoder Grand Contest 041 解説

writer: tourist

2019 年 12 月 28 日

For International Readers: English editorial starts on page 9.

A: Table Tennis Training

A, B の偶奇が同じであれば、1 人目はすべての試合で敗北し続け、2 人目は勝利し続けるべきです。この場合、2 人は $\frac{B-A}{2}$ ラウンド後に同じ卓で出会えます。

A, B の偶奇が異なるとしましょう。2 人がただ近づき続けるだけでは、隣り合う卓までは近づけますが、同じ卓で出会えません。

偶奇を調整するには、卓 1 で勝利するか卓 N で敗北するしかありません。次の 2 通りの場合のみ考えれば十分です。

- 1 人目が卓 1 まで移動し、そこでもう 1 勝してから 2 人目がいる方に向かう。2 人目は、1 人目と出会うまで卓 1 の方に移動し続ける。
- 2 人目が卓 N まで移動し、そこでもう 1 敗してから 1 人目がいる方に向かう。1 人目は、2 人目と出会うまで卓 N の方に移動し続ける。

よって、必要な最小のラウンド数は

$$\min(A-1, N-B) + 1 + \frac{B-A-1}{2}.$$

です。ここで、$\min(A-1, N-B)$ は 2 人のどちらかが卓 $1, N$ のいずれかに到達するための最小のラウンド数、1 は偶奇を調整するラウンド、$\frac{B-A-1}{2}$ は偶奇の調整後に 2 人が出会うまでのラウンド数 (2 人の間の距離の半分) を表します。

B: Voting Judges

一般性を失うことなく $A_1 \geq A_2 \geq \ldots \geq A_N$ と仮定します。初期スコアが A_i である問題が採用される可能性がある場合、それより初期スコアが高い問題も採用される可能性があることに注目して、二分探索を用いましょう。X 番目に初期スコアが高い問題に採用される可能性があるかを判定することになります。

$X \leq P$ の場合、すべてのジャッジが問題 $1, 2, \ldots, P$ に投票すれば問題 X は採用されます。

$A_X + M < A_P$ の場合、問題 X に採用される可能性はありません。

上記以外の場合、問題 X が採用されるためには問題 $1, 2, \ldots, P-1, X$ が採用されるように動くのが最適です。なぜでしょうか。「問題 X に $P-1$ 問を残して全問題に勝ってもらう必要がある。どの問題を残そうか」と考えてみましょう。答えが「初期スコアが最も高い $P-1$ 問」であることは明らかです。

当然ながら、全ジャッジが問題 X には投票すると仮定します。問題 $1, 2, \ldots, P-1$ にも全ジャッジが投票して構いません。問題 $X+1, X+2, \ldots, N$ についても、これらの問題の最終スコアは問題 X の最終スコアより高くはならないため、やはり全ジャッジが投票して構いません。$P \leq i < X$ であるような問題 i については、最大で $A_X + M - A_i$ 人が投票しても構いません。

以上で述べたような「入れられても構わない票」の総数が MV 未満であれば、問題 X に採用される可能性がないことがわかります。

そうでない場合、実は採用される可能性があると言い切れます。これは以下のように示せます。問題 i に入れられても構わない票の数を $B_i (\leq M)$ とし、$\sum B_i \geq MV$ と仮定します。$1, 1, \ldots, 1, 2, 2, \ldots, 2, 3, \ldots$ のように、問題 i が B_i 回連続で現れるような問題の列を考え、この列の最初の MV 問にジャッジ $1, 2, \ldots, M, 1, 2, \ldots, M, \ldots$ の票をこの順に割り当てます。このとき、どの問題に入る票数も B_i 以下であり、どのジャッジもちょうど V 票を投じ、どのジャッジも同じ問題に二度投票していないことが容易にわかります。

この解法の時間計算量は $O(N \log N)$ です。

C: Domino Quality

S を、$N \times N$ マスの盤面に牌を何枚か置くことですべての行とすべての列のクオリティを Q にすることが可能であるような組 (N, Q) すべての集合とします。

このとき、$(A, Q) \in S$ かつ $(B, Q) \in S$ であれば $(A + B, Q) \in S$ であることがわかります。実際に、$A \times A$ マスのクオリティ Q の盤面を左上隅に、$B \times B$ マスの盤面を右下隅に配置することで、$(A + B) \times (A + B)$ マスのクオリティ Q の盤面を得ることができます。

また、$\{(3, 1), (4, 3), (5, 3), (6, 3), (7, 3)\} \subset S$ であることもわかります。これらは、全探索か乱択アルゴリズムを実装して見つけるのが最も簡単ですが、以下のような整った配置も存在します。

```
(3, 1)      (4, 3)      (5, 3)      (6, 3)      (7, 3)

 aa.         aabc        aabba       aabc..      aabbcc.
 ..a         ddbc        bcc.a       ddbc..      dd.dd.a
 ..a         bcaa        b..cb       ..aabc      ..d..da
             bcdd        a..cb       ..ddbc      ..d..db
                         abbaa       bc..aa      dd.dd.b
                                     bc..dd      ..d..dc
                                                 ..d..dc
```

最後に、$N = 2$ の場合には構成が不可能であること、$N = 3$ の場合には $(3, 1)$ の盤面を出力すればよいこと、そして $N \geq 4$ の任意の場合には $x \geq 0, 4 \leq y \leq 7$ なる整数 x, y を用いて $N = 4x + y$ と表せ、4×4 マスの盤面 x 個と $y \times y$ マスの盤面 1 個を使って $N \times N$ マスのクオリティ 3 の盤面を得られることがわかります。

D: Problem Scores

各 k に対して以下の条件があり、合計 $N-1$ 個の条件を満たす必要があります。

$$\sum_{i=1}^{k+1} A_i > \sum_{i=N-k+1}^{N} A_i.$$

実際にはこのうち 1 個のみ、すなわち $k = \left\lfloor \frac{n}{2} \right\rfloor$ に対する条件のみ満たせばよいことがわかります。$k > \left\lfloor \frac{n}{2} \right\rfloor$ のときは、両辺の同じ変数が打ち消し合い、何らかの $k' < \left\lfloor \frac{n}{2} \right\rfloor$ に対する条件と同じ条件になります。$k < \left\lfloor \frac{n}{2} \right\rfloor$ のときは、k を 1 増やすと左辺に A_{k+2} が、右辺に A_{N-k} が足され、そのような不等式は無視しても差し支えありません。

$A_k = 1 + x_1 + x_2 + \ldots + x_k$ とします (任意の i に対し $x_i \geq 0$)。これを $\left\lfloor \frac{n}{2} \right\rfloor$-th 番目の条件式に代入すると、問題は次のようになります。

以下の条件を満たす非負整数の組 $(x_1, x_2, \ldots, x_N)$ を数えよ。

- $x_1 + x_2 + \ldots + x_N \leq N - 1$
- $x_1 \geq [x_2, x_3, \ldots, x_N] \cdot [0, 1, 2, 3, \ldots, 3, 2, 1]$ (“$\cdot$” は内積を表す)

$x_2, x_3, \ldots, x_N$ を固定して以下が成り立っていると仮定します。

- $x_2 + x_3 + \ldots + x_N = a$
- $[x_2, x_3, \ldots, x_N] \cdot [0, 1, 2, 3, \ldots, 3, 2, 1] = b$

このとき、第一の制約から $x_1 \leq N - 1 - a$ であり、第二の制約から $x_1 \geq b$ です。よって、x_1 の値の選択肢はちょうど $\max(N - a - b, 0)$ 通りです。

ここから、$a + b = [x_2, x_3, \ldots, x_N] \cdot [1, 2, 3, 4, \ldots, 4, 3, 2]$ のみが「本質的」であることがわかり、問題は次のようになります。

- 非負整数の組 $(x_2, x_3, \ldots, x_N)$ すべてに対する $\max(N - [x_2, x_3, \ldots, x_N] \cdot [1, 2, 3, 4, \ldots, 4, 3, 2], 0)$ の和を求めよ。

これは単純な $O(N^2)$ の DP により解けます。

E: Balancing Network

均一的な配置の発見

何らかのケーブル w を選びましょう。どのケーブルから出発しても w に行き着くような均一的な配置があるか判定するにはどうすればよいでしょうか。

長さ N のブール値の列 A を作り、ケーブル i からケーブル w に行き着くことが可能である場合に A_i を真とします。はじめネットワークは空で A_w のみが真であり、右から左へと平衡器を加えていくとします。

現状のネットワークの左端に平衡器 (x, y) を加えたとき、A はどのように変化するでしょうか？起こるのは、A_x, A_y がともに $(A_x|A_y)$ へと変わる、という変化のみです。実際に、A_x, A_y がともに真であるかともに偽である場合、この平衡器は何の影響ももたらず、一方のみが真である場合、平衡器を適切に設定することで双方を真にすることができます (例えば A_x が真である場合、平衡器を y から x へと向けるとケーブル y からでもケーブル w に行き着けます)。

すべての平衡器を加えた後で A_i が偽であるような i が存在するなら、条件を満たす状態は存在しません。そうでない場合には存在するのでしょうか。すなわち、どの i に対しても、i から w へと行き着くような状態が個別に存在することはわかりますが、どのケーブルからでも w に行き着くような単一の状態は存在するのでしょうか。

実は存在します。上記と似たような、平衡器を右から左へと加えていく過程を考えます。

- A_x が真、A_y が偽であるなら、平衡器を y から x へと向ける。
- A_x が偽、A_y が真であるなら、平衡器を y から x へと向ける。
- このいずれでもなければ、平衡器の方向を任意に設定する。

任意の i に対して A_i が真であるなら、このような設定で任意の i から w へと行き着くような状態が得られることがわかります。

この解法の時間計算量は $O(nm)$ です。これを高速化するには、すべてのケーブル w に対する処理はほぼ同じであり、A の初期状態のみが異なることに着目します。各 A_i を単一のブール値とするのではなく、各 w に対して 1 ビット、合計 N ビットの bitset としましょう。この他はすべて上記と同一ですが、時間計算量は $O(\frac{nm}{wordsize})$ となり、$wordsize = 64$ のとき演算の回数は 10^8 未満です。

非均一的な配置の発見

実は、$N \geq 3$ に対しては必ず非均一的な配置が存在します。

各 i に対し、ケーブル i から出発したときにどのケーブルに行き着くかを表す整数列 B_i を管理しましょう。はじめ、ネットワークは空で、任意の i に対して $B_i = i$ であるとします。ここに右から左へと平衡器を加えていきましょう。

現状のネットワークの左端に平衡器 (x, y) を加えたとき、B はどのように変化するでしょうか。平

衡器を x から y へと向けた場合、起こるのは B_x が B_y と等しくなるという変化のみです。y から x へと向けた場合は、起こるのは B_y が B_x と等しくなるという変化のみです。目標は、すべての平衡器を加えた後で B に 2 個以上の異なる値が存在するようにすることです。

そして、実はこの条件を常に保つことができます。実際に、$N \geq 3$ の場合は以下のいずれかが成り立ちます。

- B_x が B に 2 回以上現れる: この場合、$B_x \leftarrow B_y$ と設定して問題ありません。
- B_y が B に 2 回以上現れる: この場合、$B_y \leftarrow B_x$ と設定して問題ありません。
- 別の何らかの値 z ($z \neq B_x, B_y$) が B に 1 回以上現れる: この場合、平衡器の向きは任意に設定して問題ありません。

時間計算量は $O(M)$ です。

F: Histogram Rooks

$f(L, R, B)$ を、残された盤面の列 L から列 R までの部分について、$y > B$ のマスのみを考慮して (すなわち、列 i の高さを $h_i - B$ として) 何らかの DP テーブルを返す関数とします。$a_i = B$ であるような $L \leq i \leq R$ が存在する場合、この区間を $a_i > B$ なる列のみからなる何個かの区間に分割し、返された DP テーブルを併合します。そうでない場合、$f(L, R, B + 1)$ を呼んで返された DP テーブルを書き換えます。

下から B 行目より上の行にルークを詰める方法を固定すると、すべての列は次の 3 つのカテゴリーに分類されます。

1. カテゴリー R: ルークを含む。
2. カテゴリー A: ルークを含まないが、この列のどのマスも横方向からのいずれかのルークの支配下にある。
3. カテゴリー U: ルークを含まず、ルークの支配下にないマスが 1 つ以上存在する (すなわち、この列にルークを 1 個以上置く必要がある)。

f から返されるテーブルには 2 つの引数 A, U を持たせます。これらは、対応するカテゴリーの列の数を表します (R は (区間の長さ) $-$ $(A + U)$ です)。

するとテーブルの併合は簡単で、左側と右側の A, U の組を全列挙するのみです。$dp(A, U) = \sum_{i=0}^{A} \sum_{j=0}^{U} dpLeft(i, j) \cdot dpRight(A - i, U - j)$ となります。

テーブルの書き換えについては、何個かの場合を考えます。

1. この行にルークがない: カテゴリー A の列がすべて U の列に変わる。$dp(0, A + U)$ に $dpOld(A, U)$ を加える。
2. R の列にのみルークがある: $dp(A, U)$ に $dpOld(A, U) \cdot (2^R - 1)$ を加える。
3. ルークが A の列に X 個、U の列に Y 個 (そして R の列に任意個) ある: $dp(A - X, U - Y)$ に $dpOld(A, U) \cdot \binom{A}{X} \binom{U}{Y} 2^R$ を加える。

$f(1, N, 0)$ が返したテーブルの $dp(A, 0)$ の和が答えです。

この解法を高速化するには、f が返すテーブルの状態数を $O(N^2)$ から $O(N)$ にする必要があります。次の 2 つの場合を考えます。

1. 残された盤面のこの部分より下では、どの「行」(正確には、行のうちこの列区間と繋がっている部分) もルークを 1 個以上含む。
2. 残された盤面のこの部分より下に、ルークを含まない「行」がある。

ケース 1 では A の列をすべて R の列とみなしても差し支えず、ケース 2 では A の列を (いずれ U の列になるのでこの時点で) すべて U の列とみなしても差し支えないことに注目します。

よって、C を上記のケース番号 (1 または 2) として、$dp(C,U)$ というテーブルで計算を行えます。遷移はいずれも元のテーブルの場合と類似したものになります。当然ながら、$C=1$ のとき「この行にルークがない」という遷移は生じません。また、$dpOld(2,U)$ から $dp(1,U)$ に新たな遷移が生じます。空の「行」があったときに、これ以降に空行がない可能性を想定しての遷移です。

時間計算量は $O(N^3)$ です。

AtCoder Grand Contest 041 Editorial

writer: tourist

December 28, 2019

A: Table Tennis Training

If A and B have the same parity, the first player should lose all matches and the second player should win all matches. In this case, they will meet after $\frac{B-A}{2}$ rounds.

Suppose that A and B have different parity. If the players just move towards each other, they can get very close (to neighboring tables), but they can't meet.

The only way to change parity is winning at table 1 or losing at table N. We need to consider only two cases:

- The first player moves to table 1, wins an additional game there, and moves towards the second player, who just moves towards table 1 for the whole time, until they meet.
- Similarly, the second player moves to table N, loses an additional game there, and moves towards the first player, who just moves towards table N for the whole time, until they meet.

The smallest number of rounds is then

$$\min(A-1, N-B) + 1 + \tfrac{B-A-1}{2}.$$

Here, $\min(A-1, N-B)$ stands for the number of rounds needed to reach table 1 or table N, 1 stands for a single round to change parity, and $\frac{B-A-1}{2}$ stands for the number of rounds before the friends meet after changing parity (which is half the distance between them).

B: Voting Judges

Without loss of generality, let $A_1 \geq A_2 \geq \ldots \geq A_N$. Note that if a problem with an initial score of A_i can be chosen for the problemset, a problem with a higher initial score can also be chosen. Let's use binary search to find the answer. We need to check if problem with the X-th highest initial score has a chance to be chosen.

If $X \leq P$, problem X can be chosen if all judges vote for problems $1, 2, \ldots, P$.

If $A_X + M < A_P$, problem X can't be chosen.

Otherwise, it's best to try to form a problemset with problems $1, 2, \ldots, P-1, X$. Why? If you ask yourself a question "we need problem X to beat all problems except $P-1$ of them, which problems should be left unbeaten?", it's obvious that the answer is "$P-1$ problems with the highest score".

Clearly, we assume that all judges vote for problem X. It's fine if all judges vote for problems $1, 2, \ldots, P-1$. It's also fine if all judges vote for problems $X+1, X+2, \ldots, N$, as these problems can't have a higher final score than problem X anyway. For problem i such that $P \leq i < X$, at most $A_X + M - A_i$ judges can vote.

If the total number of votes that can be cast (as described above) is less than MV, we can see that problem X can't be chosen.

Otherwise, it can! This can be shown as follows. Let $B_i \leq M$ be the number of votes that can be cast for problem i, and $\sum B_i \geq MV$. Write down a sequence where problem i is repeated B_i times consecutively. Now, assign votes by judges $1, 2, \ldots, M, 1, 2, \ldots, M, \ldots$ to the first MV problems in the sequence, in this order. It's easy to see that every problem gets at most B_i votes, every judge votes exactly V times, and no judge votes for the same problem twice.

Time complexity of this solution is $O(N \log N)$.

C: Domino Quality

Let S be the set of all pairs (N, Q) such that it's possible to place some dominoes on an $N \times N$ grid so that the quality of all rows and columns is Q.

Observe that if $(A, Q) \in S$ and $(B, Q) \in S$, then $(A + B, Q) \in S$. Indeed, we can form an $(A + B) \times (A + B)$ matrix of quality Q by putting an $A \times A$ matrix of quality Q into the top-left corner, and a $B \times B$ matrix of quality Q into the bottom-right corner.

Observe that $\{(3, 1), (4, 3), (5, 3), (6, 3), (7, 3)\} \subset S$. The easiest way to find the corresponding matrices is to implement a brute force or a randomized approach, but there are some nice regular pictures as well:

```
(3, 1)      (4, 3)      (5, 3)      (6, 3)      (7, 3)

aa.         aabc        aabba       aabc..      aabbcc.
..a         ddbc        bcc.a       ddbc..      dd.dd.a
..a         bcaa        b..cb       ..aabc      ..d..da
            bcdd        a..cb       ..ddbc      ..d..db
                        abbaa       bc..aa      dd.dd.b
                                    bc..dd      ..d..dc
                                                ..d..dc
```

Finally, observe that the $N = 2$ case is impossible, for $N = 3$ we can output the $(3, 1)$ matrix, and any $N \geq 4$ can be represented as $N = 4x + y$, where x and y are integers such that $x \geq 0$ and $4 \leq y \leq 7$, and thus we can form an $N \times N$ matrix of quality 3 using the 4×4 matrix x times along with a single $y \times y$ matrix.

D: Problem Scores

We have $N-1$ conditions that must be satisfied, one condition for each k:

$$\sum_{i=1}^{k+1} A_i > \sum_{i=N-k+1}^{N} A_i.$$

Note that we only need to satisfy one condition, namely, when $k = \lfloor \frac{n}{2} \rfloor$. When $k > \lfloor \frac{n}{2} \rfloor$, equal variables in both sides cancel each other, and we end up with the same condition for some $k' < \lfloor \frac{n}{2} \rfloor$. When $k < \lfloor \frac{n}{2} \rfloor$, increasing k by 1 adds A_{k+2} to the left side and A_{N-k} to the right side, and such an inequality is never less restricting.

Let $A_k = 1 + x_1 + x_2 + \ldots + x_k$, where $x_i \geq 0$ for any i. If we substitute A_i in the $\lfloor \frac{n}{2} \rfloor$-th condition using this equality, our problem becomes the following:

Count the number of tuples of non-negative integers $(x_1, x_2, \ldots, x_N)$ such that:

- $x_1 + x_2 + \ldots + x_N \leq N - 1$;
- $x_1 \geq [x_2, x_3, \ldots, x_N] \cdot [0, 1, 2, 3, \ldots, 3, 2, 1]$, where "$\cdot$" denotes dot product.

Suppose that we have fixed $x_2, x_3, \ldots, x_N$, and:

- $x_2 + x_3 + \ldots + x_N = a$;
- $[x_2, x_3, \ldots, x_N] \cdot [0, 1, 2, 3, \ldots, 3, 2, 1] = b$.

Then we have $x_1 \leq N - 1 - a$ from the first constraint, and $x_1 \geq b$ from the second constraint. Thus, we have exactly $\max(N - a - b, 0)$ choices for x_1.

It follows that only $a+b = [x_2, x_3, \ldots, x_N] \cdot [1, 2, 3, 4, \ldots, 4, 3, 2]$ is important. The problem becomes:

- Sum up $\max(N - [x_2, x_3, \ldots, x_N] \cdot [1, 2, 3, 4, \ldots, 4, 3, 2], 0)$ over all tuples of non-negative integers $(x_2, x_3, \ldots, x_N)$.

This leads to a simple $O(N^2)$ DP solution.

E: Balancing Network

Finding a uniforming state

Let's choose some wire w. How can we check if a uniforming state exists that leads to w from any starting wire?

Create a boolean array A of length N. Let A_i be true if we can finish at wire w starting from wire i. Initially, the network is empty, and only A_w is true. We'll add balancers from right to left.

How does A change when we add a balancer (x, y) on the left of the network? The only change is that A_x and A_y are changed to $(A_x|A_y)$. Indeed, if A_x and A_y are both true or both false, this balancer doesn't change anything, and if one of them is true (say, A_x), we can direct the balancer correspondingly (say, from y to x), and now we can finish at wire w starting from wire y as well.

If after adding all balancers A_i is false for some i, the required state doesn't exist. Does it exist otherwise? In other words, we know that for any i individually a state that leads from i to w exists, but does a single state that leads all wires to w exists?

It turns out that it does. Just consider the same process of adding balancers from right to left:

- whenever A_x is true and A_y is false, direct the balancer from y to x;
- whenever A_x is false and A_y is true, direct the balancer from x to y;
- otherwise, direct the balancer arbitrarily.

Observe that if A_i is true for any i, there exists a state that leads from i to w in this setting.

Time complexity of this solution is $O(nm)$. To optimize it, notice that the process for all wires w is almost the same, the only difference is the initial state of A. Instead of keeping a boolean value in each A_i, let's keep a bitset of N values, one bit per each w. Everything else stays the same, but time complexity becomes $O(\frac{nm}{wordsize})$ (with $wordsize = 64$ it's less than 10^8 operations).

Finding a non-uniforming state

It turns out that a non-uniforming state always exists for $N \geq 3$.

Let's maintain an integer array B_i, denoting the wire we finish at if we start at wire i, for each i. Initially, the network is empty, and $B_i = i$ for any i. We'll add balancers from right to left.

How does B change when we add a balancer (x, y) on the left of the network? If we direct the balancer from x to y, the only change is B_x becoming equal to B_y, and if we direct it from y to x, the only change is B_y becoming equal to B_x. Our goal is to have at least two distinct values in B after adding all balancers.

It turns out we can keep this condition true at any time. Indeed, if $N \geq 3$, at least one of the following is true:

- B_x appears at least two times in B: then we can safely set $B_x \leftarrow B_y$;
- B_y appears at least two times in B: then we can safely set $B_y \leftarrow B_x$;
- some other value z ($z \neq B_x, B_y$) appears at least once in B: then we can direct the balancer arbitrarily.

Time complexity is $O(M)$.

F: Histogram Rooks

Let $f(L, R, B)$ be a function that returns some DP on a part of the histogram from column L to column R, only considering cells at $y > B$ (that is, assuming that the height of column i is $h_i - B$). If some $a_i = B$ for $L \le i \le R$, we can divide this segment into several segments containing columns with $a_i > B$, and merge DPs. Otherwise, we can call $f(L, R, B + 1)$ and modify DP.

If we have fixed the way of filling all rows above B-th with rooks, all columns can be divided into three categories:

1. category R: contains a rook;
2. category A: doesn't contain a rook, but all cells in this column are under horizontal attack of some rook;
3. category U: doesn't contain a rook, and at least one cell is not under attack of any rook (thus, this column needs at least one rook).

Our DP can have two arguments: A and U, the number of columns of the corresponding categories (R is segment width minus $A + U$).

Merging DPs is easy then, just loop over A and U in the left part and in the right part: $dp(A, U) = \sum_{i=0}^{A} \sum_{j=0}^{U} dpLeft(i, j) \cdot dpRight(A - i, U - j)$.

In modifying DP, there are a couple of cases:

1. no rook in this row: all A-columns transform into U-columns, increase $dp(0, A + U)$ by $dpOld(A, U)$;
2. only rooks in R-columns: increase $dp(A, U)$ by $dpOld(A, U) \cdot (2^R - 1)$;
3. X rooks in A-columns and Y rooks in U-columns (and maybe rooks in R-columns): increase $dp(A - X, U - Y)$ by $dpOld(A, U) \cdot \binom{A}{X}\binom{U}{Y}2^R$.

The answer is the sum of $dp(A, 0)$ returned from $f(1, N, 0)$.

To optimize this solution, we need to move from $O(N^2)$ DP states returned from f to $O(N)$ DP states. Consider two cases:

1. below this part of histogram, every "row" (actually, part of the row connected to this column segment) will contain at least one rook;
2. below this part of histogram, at least one "row" will contain no rooks.

In case 1, note that we can assume that all A-columns are just R-columns, and it doesn't change anything.

In case 2, note that we can assume that all A-columns are just U-columns (they will become U-columns in the future, but we can just assume they already are).

Hence, we can have $dp(C, U)$ where C is 1 or 2, denoting case number from above. All transitions work in a similar way. Of course, the "no rook in this row" transition can not happen when $C = 1$. An additional transition happens

from $dpOld(2, U)$ to $dp(1, U)$: any time we encounter an empty "row", we can make this transition as an assumption that we won't have empty rows in the future.

Time complexity is $O(N^3)$.