CODE FESTIVAL 2016 予選 C 解説

sigma425

A: CF

文字列の長さ (N とします) が短いので、$i < j$ であって $s[i] = C, s[j] = F$ なるものがあるか、i と j に関して 1 から N まで二重ループを回して時間計算量 $O(N^2)$ で探せば良いです。また、先頭から順番に,「まだ C が出ていない」,「C が出ていてその後まだ F が出ていない」,「C が出ていてその後 F が出た」のどの状態に今いるかを持って遷移すると $O(N)$ 時間で解くことも出来ます。

B: K 個のケーキ

a_1個の $1, a_2$個の $2, ...a_T$個の T を並び替えて隣り合う同じ数のペアの数を最小化する問題です。a_t が最大となるような t を一つ選びます。まず t を a_t 個一列に並べて、他の数をその間に挟んでいくように列を大きくしていくことを考えます。すると、t 以外の数 x を a_x 個置くときは、$a_t \geq a_x$ より、t の間に挟むことで x が隣り合わないように置くことが可能です。t を置いた段階では隣り合う同じ数のペアは $a_t - 1$ 個で、他の数をひとつ置くたびにこの数は 1 ずつ減っていきます (0 になったらそれ以上減りません)。従って答えは、

$$max(a_t - 1 - (K - a_t), 0)$$

で求まります。

C: 二人のアルピニスト

まず高橋君の記録からどれだけ情報が得られるか考えます。各 $i(1 \leq i \leq N)$ に対して、次のことがわかります。

- $i = 1$ または $T_{i-1} < T_i$ のとき　・・・ 最大値が更新されているので、$h_i = T_i$ がわかります。
- それ以外　・・・ 最大値が更新されていないので、$h_i \leq T_i$ であればなんでもよいです。

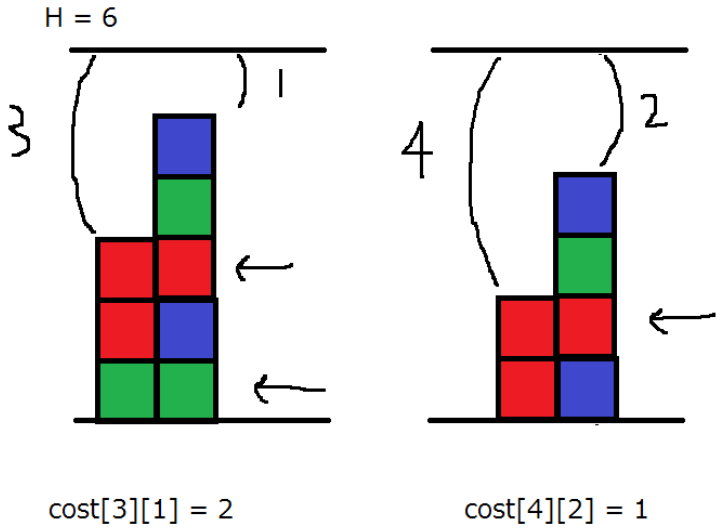
逆に、これを満たす $h_1, h_2, .., h_N$ は高橋君の記録に合致します。青木君の記録についても同様な条件として書けるので、結局、各山の高さ h_i に関する条件は、

- $h_i = T_i$ かつ $h_i = A_i$
- $h_i = T_i$ かつ $h_i \leq A_i$
- $h_i \leq T_i$ かつ $h_i = A_i$
- $h_i \leq T_i$ かつ $h_i \leq A_i$

のどれかの形をしています。等号による条件を含んでいる場合は、それがもう片方の条件を満たしているか確かめる必要があり、満たしていれば h_i は 1 通りで、そうでなければ 0 通りです。二つとも不等号による条件の場合は、$1 \leq h_i \leq min(T_i, A_i)$ となるので、h_i は $min(T_i, A_i)$ 通りあります。よって、これらをかけ合わせると答えが求まります。

D: Friction

まず $W = 2$ とした問題を考えます。
この問題は次のような dp によって解けます。$dp[x][y]$:=「1 列目を x 段、2 列目を y 段沈めるまでにかかるコストの総和の最小値」とします。答えは $dp[H][H]$ です。すると、$dp[x][y]$ を計算するのに、直前に 1 列目と 2 列目のどちらを沈めたかの二通りの遷移を考えればよく、$dp[x][y] = min(dp[x-1][y] + cost[x-1][y], dp[x][y-1] + cost[x][y-1])$ と計算できます。ただし、$cost[x][y]$:=「1 列目を x 段、2 列目を y 段沈めた状態で次に動かそうとした時のコスト (つまり、横に隣り合っている同じ色のブロックのペアの個数)」とします。$cost[x][y]$ は実際に隣り合っているペアに対し色が同じかどうかそれぞれ判定することで $O(H)$ 時間で求まります。dp の状態は $O(H^2)$ 個あり、更新に $O(H)$ かかるので全体で計算量は $O(H^3)$ になります。
次に部分点である $W = 3$ の場合を考えます。
$W = 2$ のときと同様に $dp[x][y][z]$:=「1 列目を x 段、2 列目を y 段、3 列目を z 段沈めるまでにかかるコストの総和の最小値」とすると、dp の状態は $O(H^3)$ 個になるので、更新に $O(H)$ かけると全体で $O(H^4)$ になり間に合いません。なので、cost を素早く計算することにします。これは、$cost[i][j]$ と $cost[i+1][j+1]$ が下の図のようにほとんど変わらず、1 列目を i 段、2 列目を j 段沈めた状態での一番低いブロックのペアによる寄与の分だけ変わることを使うと、これも dp によって全体で $O(H^2)$ で計算できます。



これを事前に計算しておくことで元の dp の更新は $O(1)$ に出来るので、全体で計算量は $O(H^3)$ となり部分点を得ることが出来ます。
$W \leq 300$ の場合は、すべての列の状態を同時に持つと状態が $O(H^W)$ 個に膨れ上がってしまうためこの方針では不可能です。まず操作列を、各操作で何列目を選んだか、の数を並べた数列として考えます。操作により発生するコストを、どの二つの列の間で発生したかによって分類します。すると、1 列目と 2 列目の間, 2 列

目と 3 列目の間, ... ,$W-1$ 列目と W 列目の間 、の $W-1$ 通りに分けることが出来ます。すると、 x 列目と $x+1$ 列目の間で発生するコストは、操作列の x と $x+1$ の相対的順番のみに依存し、他の値が表れる順番には関係ありません。
各 x に対して、 x 列目と $x+1$ 列目の間で発生するコストを最小化するような x と $x+1$ の並べ方を考えます。すると、全ての x に対してこの相対的な順番を満たすような元の操作列が実際に作れます。これは、まず 1 と 2 を並べて、3 を並べるときには 1 を無視して 2 との相対位置がうまくいくように挿入して、4 を並べるときには 1,2 を無視して 3 との相対位置がうまくいくように挿入して、.... と繰り返せば良いです。
従って各 x に対して、 x 列目と $x+1$ 列目の間で発生するコストの最小を求めて、それらを足し合わせると答えになります。それぞれのコストは $W=2$ のときと同様に求められるので計算量 $O(H^2)$ で求まり、これを $W-1$ 回繰り返すので全体で計算量は $O(H^2W)$ となります。

E: 順列辞書

忘れてしまった数の個数 (すなわち入力に含まれる 0 の個数) を K 個とおきます。すると $K!$ 個の順列の載っているページ番号の総和を求めることになります。まず解説のわかりやすさのため、順列に含まれる値を 1〜N ではなく 0〜$N-1$ とします。それに伴い、列 $p_0, p_1, ..., p_{N-1}$ を、$P_{i+1} > 0$ なら $p_i = P_{i+1} - 1$, そうでないなら $p_i =?$ とおきます。また、ページ番号も 0〜$N! - 1$ にずらします。こちらのバージョンで答えを求めてから、それに $K!$ を足すと元のバージョンの答えを得ることが出来ます。

まずある順列 $s_0, s_1, ..., s_{N-1}$ が与えられたときにそれが書かれたページ番号を求める方法を考えます。s より小さい順列は、ある $i(0 \le i \le N-1)$ に対し、「任意の $j < i$ に対し $a_j = s_j$ かつ、$a_i < s_i$」となる a たちの集合になります。これらは排反なので、各 i に対し条件を満たすものを数えて足し合わせるとページ番号が求まります。i を固定すると、$a_j (j < i)$ の選択は s_j 一択で、a_i の選択が、「s_i 未満で s_0 から s_{i-1} までに表れない数」で、a_{i+1} から a_{N-1} は $N-1-i$ 個からなる任意の順列が選べるので、結局ページ番号は次の式のように表せます。

$$\sum_{i=0}^{N-1} (s_0\text{から } s_{i-1}\text{までに表れない } s_i\text{未満の数の個数}) \cdot (N-1-i)!$$

これは次のように言い換えられます。

$$\sum_{i=0}^{N-1} \left(s_i - \sum_{j=0}^{i-1} (s_j < s_i) \right) \cdot (N-1-i)!$$

なので、問題の答えは、

$$\sum_{s:\text{ありうる順列}} \sum_{i=0}^{N-1} \left(s_i - \sum_{j=0}^{i-1} (s_j < s_i) \right) \cdot (N-1-i)!$$

となります。ただし、$(s_j < s_i)$ は成り立つなら 1 を, そうでないなら 0 を返すものとします。
$\sum$ の順番を交換して、

$$\sum_{i=0}^{N-1} \left(\sum_{s:\text{ありうる順列}} s_i - \sum_{j=0}^{i-1} (s_j < s_i) \right) \cdot (N-1-i)!$$

を求めることにします。以下この i 項目を求めます。

1.$p_i \neq ?$

s_i による方は O(1) で簡単に計算できるので、

$$\sum_{s:\text{ありうる順列}} \sum_{j=0}^{i-1} (s_j < s_i)$$

が求まれば良いです。さらに $p_j = ?$ かどうかで分類します。

a) $p_j \neq ?$

s によらず一定な部分なので全ての j を見ることで $O(N)$ で計算できます。

b) $p_j = ?$

s_j に s_i より小さい値が入る s の数は、j によらず「(p に現れない数のうち s_i 未満のものの個数) · $(K-1)!$ 」になります。よってこれは $O(logN)$ で計算できます。

2.$p_i = ?$

こちらも s_i による方は p に出てこない値たちの総和を考えると $O(1)$ で計算できるので、

$$\sum_{s:\text{ありうる順列}} \sum_{j=0}^{i-1} (s_j < s_i)$$

が求まれば良いです。$p_j = ?$ かどうかで分類します。

a) $p_j \neq ?$

1b) と同様に計算できます。

b) $p_j = ?$

s_i を固定すると 1b) と同様になります。なので s_i を全て試せば $O(NlogN)$ で計算できます。

これで全体で計算量 $O(N^2logN)$ になり部分点を得ることが出来ます。

満点を得るには、BIT などのデータ構造を使い 1a) を $O(logN)$ で計算することと、2b) で s_i をすべて試すのではなく、これが簡単な $\sum$ の形で表せるのでそれを $O(1)$ で計算することで全体で計算量 $O(NlogN)$ で答えを求めることが必要です。

CODE FESTIVAL 2016 Qual C Editorial (English)

sigma425

A: CF

Since the given string is short, it is enough to search for a pair i, j such that $i < j$ and $s[i] =$`'C'`$, s[j] =$`'F'` in $\mathrm{O}(N^2)$ time, where N is the length of the string. It is also possible to solve this problem in $\mathrm{O}(N)$ time by examining the string from the beginning and executing transitions among the three states: "`'C'` has not occurred", "`'C'` occurred, but since then `'F'` has not occurred" and "`'C'` occurred, and afterward `'F'` occurred".

B: K Cakes

In this problem, you are asked to permute a_1 copies of 1, a_2 copies of 2, ..., a_T copies of T to minimize the number of adjacent pairs of equal numbers.

Let t be such that a_t is maximum. First, let us arrange a_t copies of t in a row, then place the other numbers between them. When placing a_x copies of $x(\neq t)$, we can place each copy between two copies of t so that no two copies of x are adjacent, since $a_t \geq a_x$. At first when only a_t copies of t are placed, the number of adjacent pairs of equal numbers is $a_t - 1$, and this number decreases by 1 each time another number is placed (but does not go below 0). Thus, the answer is:

$$\max(a_t - 1 - (K - a_t), 0)$$

C: Two Alpinists

First, let us examine Takahashi's records. For each $i(1 \leq i \leq N)$, the following can be seen:

- When $i = 1$ or $T_{i-1} < T_i$: since the maximum height so far is updated, $h_i = T_i$.
- Otherwise: since the maximum height so far is not updated, h_i can be any value as long as $h_i \leq T_i$.

Conversely, any sequence $h_1, h_2, \ldots, h_N$ that satisfies above is consistent with Takahashi's records. Similar observations can be made on Aoki's records, thus the height of each mountain, h_i, is under one of the following combinations of conditions:

- $h_i = T_i$ and $h_i = A_i$
- $h_i = T_i$ and $h_i \leq A_i$
- $h_i \leq T_i$ and $h_i = A_i$
- $h_i \leq T_i$ and $h_i \leq A_i$

When one of the two conditions on h_i is an equation, verify if the specified value satisfies the other condition. If the other condition is satisfied, the number of the possible values for h_i is 1, and 0 otherwise. When both conditions on h_i are inequations, $1 \leq h_i \leq \min(T_i, A_i)$, thus there are $\min(T_i, A_i)$ possible values for h_i. The answer can be found by multiplying those numbers for all h_i.

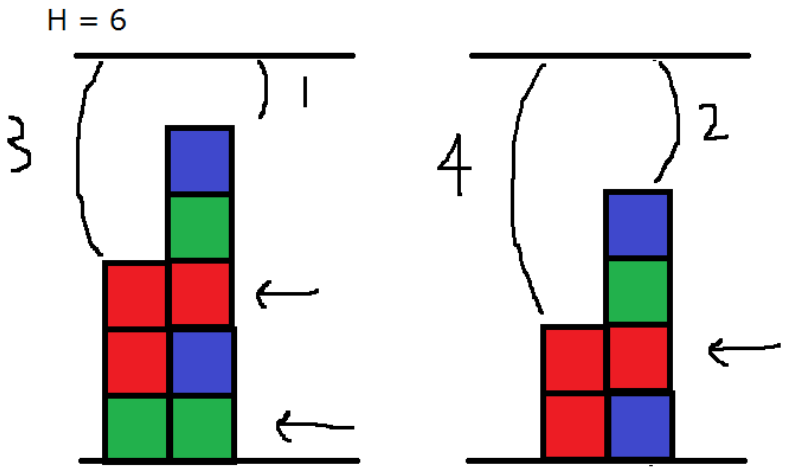
D: Friction

First, consider the case where $W = 2$.

This case can be solved by Dynamic Programming as follows: let $dp[x][y]$:= "the minimum cost for pushing down the first column by x rows and the second column by y rows". The answer will be found as $dp[H][H]$. The value of each $dp[x][y]$ can be computed by considering the two transitions: "which column is pushed down for the last time?" Specifically, $dp[x][y] = min(dp[x-1][y] + cost[x-1][y], dp[x][y-1] + cost[x][y-1])$, where $cost[x][y]$:= "the cost for pushing down a column when the first column is pushed down by x rows and the second column is pushed down by y rows (that is, the number of the horizontally adjacent pairs of blocks of the same color)". Each $cost[x][y]$ can be found in $\mathrm{O}(H)$ time by scanning the adjacent pairs of blocks after pushing down the columns. Since there are $\mathrm{O}(H^2)$ states, and computing the value for each state takes $\mathrm{O}(H)$ time, the time complexity of this solution is $\mathrm{O}(H^3)$.

Next, consider the case where $W = 3$, which is allocated a partial score.

Similarly to the previous case, let $dp[x][y][z]$:= "the minimum cost for pushing down the first column by x rows, the second column by y rows, and the third column by z rows". This time there are $\mathrm{O}(H^3)$ states, thus computing each value in $\mathrm{O}(H)$ time will make the total time complexity $\mathrm{O}(H^4)$, which is not enough. Let us find a faster way to calculate the values of $cost[x][y]$. As seen in the figure below, the difference of two values $cost[i][j]$ and $cost[i+1][j+1]$ only comes from the bottom pair of blocks after pushing down the first column by x rows and the second column by y rows. Using this property, the values of all $cost[i][j]$ can be computed in $\mathrm{O}(H^2)$ time by executing another DP.



After finding the values of all $cost[i][j]$, the value of each $dp[x][y][z]$ can be computed in O(1) time, making the total time complexity $O(H^3)$, which is enough.

Now, we will deal with the original problem, where W can be up to 300. We cannot have the states of all columns as the state of DP, or there will be $O(H^W)$ states.

Consider a sequence of operations as an integer sequence, where each element represents which column to push down. We will classify the costs generated from operations according to which two columns they are generated between: the first and second columns, the second and third columns, ..., the $(W-1)$-th and W-th columns. We can see that the costs generated between the x-th and $(x+1)$-th columns only depend on the relative order in which x and $x+1$ occur in the sequence of operation, and not affected by occurrences of other values.

For each x, consider an optimal relative order of x and $x+1$ that minimizes the cost generated between the x-th and $(x+1)$-th columns. We can construct a sequence of operations that is consistent with all of these optimal relative order, as follows. First, we will arrange 1s and 2s according to the optimal order. Then, we will place 3s according to the optimal relative order of 2s and 3s, ignoring 1s. Then, we will place 4s according to the optimal relative order of 3s and 4s, ignoring 2s, and so forth.

Therefore, the answer can be found by finding the minimum cost generated between the x-th and $(x+1)$-th columns for each x, and summing up the results. Each of these $W-1$ costs can be computed in $O(H^2)$ time by the methods discussed previously, making the total time complexity $O(H^2W)$.

E: Encyclopedia of Permutations

Let K be the number of the forgotten numbers (that is, the number of 0s in the input). What we must find is the sum of the page number for the $K!$ possible permutations. For simplicity, we will consider permutations of the numbers 0 through $N-1$, instead of 1 through N. According to this change, let the sequence $p_0, p_1, ..., p_{N-1}$ be as follows: $p_i = P_{i+1} - 1$ if $P_{i+1} > 0$, and $p_i =?$ otherwise. We will also shift the page numbers to 0 through $N! - 1$. After finding the answer for this version of problem, we will add $K!$ to obtain the answer for the original problem.

First, let us consider how to find the page number for a given permutation $s_0, s_1, ..., s_{N-1}$. For any permutation $a_0, a_1, ..., a_{N-1}$ that is smaller than s, there exists exactly one $i (0 \leq i \leq N-1)$ such that "$a_j = s_j$ for any $j < i$, and $a_i < s_i$". We can find the page number for s by counting the number of permutations that satisfy the condition for each value of i, and summing up the results. Let us fix i and count the number of these permutations. For each value of $a_j (j < i)$, there is exactly one choice. For the value of a_i, we can choose any integer that is smaller than s_i and does not occur in $s_0, s_1, \ldots, s_{i-1}$. For the values of $a_{i+1}, a_{i+2}, \ldots, a_{N-1}$, any permutations of the remaining $N-1-i$ integers can be used. Thus, the page number of s is represented by:

$$\sum_{i=0}^{N-1} (\text{the number of the integers smaller than } s_i \text{ and not occurring in } s_0, \ldots, s_{i-1}) \cdot (N-1-i)!$$

which can be rewritten as:

$$\sum_{i=0}^{N-1} \left(s_i - \sum_{j=0}^{i-1} (s_j < s_i) \right) \cdot (N-1-i)!$$

thus, the answer for the problem is:

$$\sum_{s \in S} \sum_{i=0}^{N-1} \left(s_i - \sum_{j=0}^{i-1} (s_j < s_i) \right) \cdot (N-1-i)!$$

where S is the set of the possible permutations, and $(s_j < s_i)$ is evaluated to 1 if the inequation holds, and 0 otherwise. Let us swap the order of $\sum$ and find:

$$\sum_{i=0}^{N-1} \left(\sum_{s \in S} s_i - \sum_{j=0}^{i-1} (s_j < s_i) \right) \cdot (N-1-i)!$$

We will calculate each of these i terms below.

1.$p_i \neq ?$

We must find:

$$\sum_{s \in S} \sum_{j=0}^{i-1} (s_j < s_i)$$

(The remaining parts can be easily calculated in O(1) time.) We will further classify the inner terms into two categories:

a) $p_j \neq ?$

These terms are not affected by s and constant, thus the sum can be calculated in $O(N)$ time by scanning through all j.

b) $p_j = ?$

The number of s such that $s_j < s_i$ is (the number of the integers smaller than s_i and not occurring in p) $\cdot (K-1)!$, and thus the sum can be calculated in $O(\log N)$ time.

2.$p_i = ?$

Again, we must find:

$$\sum_{s \in S} \sum_{j=0}^{i-1} (s_j < s_i)$$

(The remaining parts can be calculated in O(1) time, by finding the sum of the integers not occurring in p beforehand.) We will further classify the inner terms into two categories:

a) $p_j \neq ?$

Apply a similar method to the one used in 1b).

b) $p_j = ?$

When s_i is fixed, this case becomes similar to 1b). Thus, the sum can be calculated in $O(N \log N)$ time by going through all s_i.

The total time complexity of this solution is $O(N^2 \log N)$, which can earn the partial score.

To obtain the full score, it is necessary to utilize a data structure such as BIT to calculate the sum in 1a) in $O(\log N)$ time. Also, in 2b), instead of going through all s_i, mathematically calculate the sum in O(1) time. These two techniques will achieve the total time complexity of $O(N \log N)$.