AtCoder Regular Contest 073 Editorial

Kohei Morita(yosupo)

平成 29 年 4 月 29 日

A: Shiritori

文字列の入力と if 文を書くことが必要になります。
以下に python3 でのコード例を示します。

```
a, b, c = input().split()
if a[len(a)-1] == b[0] and b[len(b)-1] == c[0]:
    print('YES')
else:
    print('NO')
```

B: Choice Integers

A の倍数はいくつ足しても A の倍数です。よって、実は選ぶ数は 1 個だけで良いです (いくつか選んで足さなくても、最終的な総和を直接選べます)。

次に、$A\%B, 2A\%B, 3A\%B, ...$ という数列を考えます。なお $A\%B$ は A を B で割ったあまりを表します。

ここで、$(k+B)A\%B = (kA+BA)\%B = kA\%B$ に注目すると、この数列は周期的で、最初の B 個の要素を繰り返す数列になっていることがわかります。

よって、この問題は A から BA まで、愚直に B で割った余りを求めて調べれば良いです。

C: Sentou

i 人目がスイッチを押した後、$i+1$ 人目が

- T 秒以内に来るならば、ずっとお湯は出続ける
- T 秒よりも経ってからくるならば、お湯は T 秒間出て止まる

ということがわかります。

よってこの問題の答えは、それぞれの人について、次の人が何秒後に来るかを求め、$min(T,$ 次の人が来るまでの時間$)$ の総和を求めれば良いです。

D: Simple Knapsack

この問題はナップザック問題として知られていて、効率良く解くアルゴリズムは存在しないと信じられています。ただし、特殊な制約がある場合はその限りではありません。例えば N や W や v_i のうちどれかが小さい場合は ABC032 D 問題で出題されています。

今回は、すべての $i = 2, 3, \cdots, N$ について、$w_1 \leq w_i \leq w_1 + 3$ という特殊な制約を利用するのだろうと考えられます。

この性質に注目すると、物の重さは高々 4 種類であることがわかります。

よって、各重さについてその重さの物を何個選ぶか決めてしまいます。すると、各重さごとに価値の高い順に使うことにすれば良くなります。

そして選び方は、最大でも $(N/4)^4 = 25^4 = 390625$ 通りありますが、この程度ならば全探索が可能です。

使う個数を決めた後は、各重さごとに価値の高いものから使っていくことになります。この、価値の総和は O(選んだ個数) で計算できます。

C++や Java や D 言語など、高速な言語ならばこれで間に合うと思いますが、Python や Ruby などの場合、累積和などを使い $O(1)$ で計算できるようにしないと厳しいかもしれません。

E: Ball Coloring

ボールは $400,000$ 個もありますが、答えに影響するパラメーターは $R_{max}, R_{min}, B_{max}, B_{min}$ の高々 4 個だけであることに注目します。

更に、

$$MIN = min(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)$$

$$MAX = max(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)$$

とすると、$R_{min} = MIN, B_{min} = MIN$ のどちらかは満たし、$R_{max} = MAX, B_{max} = MAX$ のどちらかは満たすことがわかります。

もっというと、$R_{min} = MIN \& B_{max} = MAX$ と $R_{min} = MIN \& B_{max} = MAX$ のどちらかを仮定して良いです。

よってこれで場合分けをします。

$R_{min} = MIN \& B_{max} = MAX$ の場合

この場合、R_{min} や B_{max} がこれ以上小さくなったり大きくなったりすることはありません。よって、各袋について、小さい方を赤色、大きい方を青色にすれば良いです。

$R_{min} = MIN \& R_{max} = MAX$ の場合

最小のボールと最大のボールが同じ色の場合です。

この場合、赤色の方には何も考えず好きなボールを押し付けられます。

よって青色に塗るボールのみ着目すればよいです。

つまり、各袋からボールを 1 個選び最大-最小を最小化する、という問題を考えれば良いです。

これは、最初全ての袋について小さい方のボールを選んだと仮定し、

それをそのなかで小さい順に並べ、順番に、そのボールを袋のもう一つのボールと交換する、ということを行えば良いです。

F: Many Move

この問題は愚直な DP を考え、それを高速化していく方針を取ります。

まず、$O(N^3)$ の DP として以下の様なものが考えられます。

$dp[i][a][b] := i$ 個のクエリを処理しており、コマは a と b にある。今までかかった時間の最小

ここで、どちらかのコマは必ず直前のクエリの位置にいることを考えると、$O(N^2)$ に高速化が出来ます。

$dp[i][a] := i$ 個のクエリを処理しており、コマは a と x_i にある。今までかかった時間の最小

$x_0 = B$ とすると、

$dp[0][A] = 0, dp[0][x] = \infty (x \neq A)$

を初期値として、

$\min(dp[Q][1], dp[Q][2], ..., dp[Q][N])$

を答えとすれば良いことがわかります。

この DP の漸化式は、

$dp[i][a] = dp[i-1][a] + |x_{i-1} + x_i| (a \neq x_i)$

$dp[i][x_{i-1}] = \min(dp[i-1][x_{i-1}] + |x_{i-1} + x_i|, \min_{j=1,2,...,n}(dp[i-1][j] + |j - x_i|))$

となります。

ここで、DP テーブル $dp[i-1][1], dp[i-1][2], ..., dp[i-1][n]$ を配列として持っているとします。そしてここから一気に $dp[i][1], dp[i][2], ..., dp[i][n]$ を計算することを考えます。

上の漸化式から考えると、

1. 配列全体に $|x_{i-1} + xi|$ を足し
2. $\min_{j=1,2,...,n}(dp[i-1][j] + |j - x_i|)$ を求め、$dp[i][x_{i-1}]$ より小さければ代入

という操作ができれば良いことがわかります。

難しい操作は $\min_{j=1,2,...,n}(dp[i-1][j] + |j - x_i|)$ ですが、これは j と x_i の大小で場合分けをすると

$\min_{j=1,2,...,x_i}(dp[i-1][j] - j + x_i)$

$\min_{j=x_i+1,2,...,n}(dp[i-1][j] + j - x_i)$

が求められれば良くなります。

これは、$dp[i-1][j] - j$ と $dp[i-1][j] + j$ を持った配列に対する区間の min を求める操作になります

整理すると、実は持つべきは $dp[i-1][j]$ の配列ではなく、$dp[i-1][j] - j$ と $dp[i-1][j] + j$ の配列です。

$dp[i-1][j]-j$ と $dp[i-1][j]+j$ の配列を持っておくと、全体に add と区間 min ができれば良くなります。これは、Segment Tree で解くことが出来ます。

AtCoder Regular Contest 073 Editorial

Kohei Morita(yosupo)

平成 29 年 4 月 29 日

A: Shiritori

python3 example:

```
a, b, c = input().split()
if a[len(a)-1] == b[0] and b[len(b)-1] == c[0]:
    print('YES')
else:
    print('NO')
```

B: Choice Integers

The sum of multiples of A is always a multiple of A. Thus, we can assume that we choose only one integer.

Consider the sequence $A\%B, 2A\%B, 3A\%B, ...$ Since $(k+B)A\%B = (kA+BA)\%B = kA\%B$, this sequence has a period of B.

Therefore, we can check the first B elements of this sequence by brute force.

C: Sentou

After the i-th person pushes the switch, the $i + 1$-th person

- If the $i+1$-th person comes within T seconds, the water keeps emitting.
- If the $i + 1$-th person comes after at least T seconds, the water emits for T seconds and stops.

Therefore, the answer is the sum of $min(T, \mathrm{t_{i+1}} - \mathrm{t_i})$.

D: Simple Knapsack

This problem is known as the knapsack problem, and it's hard to solve in general case.

This time we use the constraint $w_1 \leq w_i \leq w_1 + 3$. Because of this, there are at most 4 possible weights for a single item.

For each weight w, let's fix k_w, the number of items of weight w we choose. Obviously, we should choose k_w items greedily (in the descending order of values).

Let A, B, C, D be the number of items of weights w_1, w_1+1, w_1+2, w_1+3, respectively. This algorithm works in $O(ABCD)$. In the worst case, this is $(N/4)^4 = 25^4 = 390625$.

E: Ball Coloring

Let

$$MIN = min(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)$$

$$MAX = max(x_1, x_2, ..., x_n, y_1, y_2, ..., y_n)$$

.

One of $R_{min} = MIN, B_{min} = MIN$ will be satisfied, and one of $R_{max} = MAX, B_{max} = MAX$ will be satisfied. Without loss of generality, we can consider the following two cases:

$R_{min} = MIN \& B_{max} = MAX$

In this case we want to minimize R_{max} and maximize B_{min}. For each bag, we should color the ball with the larger integer blue, and color the other ball blue.

$R_{min} = MIN \& R_{max} = MAX$

In this case we are only interested in blue balls (the value of $B_{max} - B_{min}$. First, for each bag, we color the ball with the smaller integer blue, and sort the blue balls in ascending order. Call them $z_1, \ldots, z_n$ (in ascending order). Then, for each $i = 1, \ldots, n$, color z_i red and color the opponent of z_i blue.

F: Many Move

Let $dp[i][a][b]$ be the minimum cost required to process the first i queries such that the tokens are at a and b after the queries. This DP works in $O(N^3)$. We'll improve this.

Let $dp[i][a]$ be the minimum cost required to process the first i queries such that the tokens are at a and x_i after the queries. This DP works in $O(N^2)$.

Suppose that we have an array $dp[i-1][1], dp[i-1][2], ..., dp[i-1][n]$, and we want to convert it to the array $dp[i][1], dp[i][2], ..., dp[i][n]$.

The following operations are required:

1. Add $|x_{i-1} + xi|$ to the whole array.

2. Compute $\min_{j=1,2,\ldots,n}(dp[i-1][j] + |j - x_i|)$.

This can be done by two RMQs: one of them holds the values of $dp[i-1][j] - j$ and the other one holds the values of $dp[i-1][j] + j$.