ABC 140 解説

beet, drafear, gazelle, kort0n

2019 年 9 月 7 日

For International Readers: English editorial will be published in a few days.

A: Password

各桁ごとに独立して N 通りの数字を設定出来ますから、答えは N^3 です。
N を入力として受け取り、 N^3 を計算して、その値を出力することにより、AC となります。
以下は C++ による実装例です。

```cpp
#include <iostream>
using namespace std;
int main(){
  int N;
  cin >> N;
  cout << N * N * N << endl;
  return 0;
}
```

B: Buffet

次のようにシミュレーションを行うことで答えを求めることができます。まず、A_1 種類目の料理を食べるので、B_{A_1} の満足度を得ます。次に、A_2 種類目の料理を食べるので、B_{A_2} の満足度と、$A_2 = A_1 + 1$ ならば C_{A_1} の満足度を得ます。i $(2 \leq i \leq N)$ 番目には A_i 種類目の料理を食べるので、B_{A_i} の満足度と $A_i = A_{i-1} + 1$ ならば $C_{A_{i-1}}$ の満足度を得ます。これを C++ 言語で実装した例を以下に挙げます。

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
  int N; cin >> N;
  // input
  vector<int> A(N);
  for (int i = 0; i < N; ++i) {
    cin >> A[i];
    --A[i];
  }
  vector<int> B(N);
  for (int i = 0; i < N; ++i) {
    cin >> B[i];
  }
  vector<int> C(N-1);
  for (int i = 0; i < N; ++i) {
    cin >> C[i];
  }
  // calc
  int ans = 0;
  for (int i = 0; i < N; ++i) {
    ans += B[A[i]];
    if (i > 0 && A[i] == A[i-1]+1) {
      ans += C[A[i-1]];
    }
  }
  // output
  cout << ans << endl;
}
```

C: Maximal Value

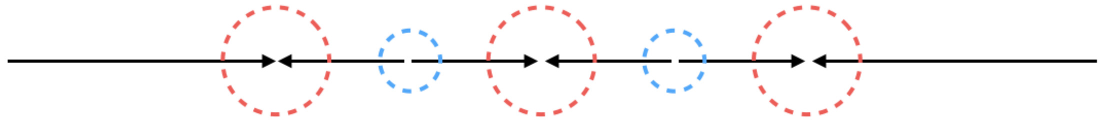
長さ N の数列 C を、

$$\begin{aligned}C_1 &= B_1\\ C_i &= \min(B_{i-1}, B_i) \ \ (i = 2, 3, \cdots, N-1)\\ C_N &= B_N\end{aligned}$$

で定めます。
このとき、任意の i について $A_i \leq C_i$ が成立しなければなりません。実際、 $i = 2, 3, \cdots, N-1$ について $A_i > C_i$ であったと仮定すると、$A_i > B_{i-1}$ または $A_i > B_i$ が成立しますが、 $A_i > B_{i-1}$ であれば $B_{i-1} \geq \max(A_{i-1}, A_i)$ に矛盾し、 $A_i > B_i$ であれば $B_i \geq \max(A_i, A_{i+1})$ に矛盾します。$i = 1, N$ のときも、矛盾することが分かります。
一方、 $A = C$ とすると、これは問の条件を満たすことが分かります。以上より、 問の条件を満たす数列のうち要素の総和が最大となる数列は、 C です。
実装上は、$B_0 = B_N = \infty$ (十分に大きな値) とすると、場合分けが不要で楽です。

D: Face Produces Unhappiness

人の向きを以下の図のように矢印で表すことにします。この図では、同じ向きの人が連続している部分の矢印を繋げて 1 つの矢印で表しています。また、簡単のため、N 人の左側には右向きの人が無限に並んでいて、右側には左向きの人が無限に並んでいることとします。



幸福である人数を数える代わりに、幸福でない人を数えましょう。すなわち、上図の赤い点線で囲った部分の数をできるだけ減らしたいです。赤い点線の部分 1 箇所につき幸福でない人は基本的には 2 人ですが、この部分が (N 人の) 左端や右端にある場合には 1 人です。

(l, r) を選んで反転する操作を行うと、これらの赤・青の点線の部分はどのように変化するでしょうか。反転した区間の端以外では、増えも減りもせず、赤、青の点線の部分はそれぞれ赤、青の点線のままです。したがって、1 回の操作では選ぶ区間の端として赤い点線部分と青い点線部分を選び、赤い点線部分を 1 箇所を消すことができますが、それ以外の選び方では消すことができません。

また、左には右向きの人が無限に並んでいて、右には左向きの人が無限に並んでいることから、全体として赤または青の点線の部分の数は必ず奇数で、赤、青、赤、青、…、赤と並んでいます。これから分かることとして、赤い点線部分が M 箇所あるとすると、青い点線部分は $M-1$ 箇所であるため、何回操作ができたとしても赤い点線部分は必ず 1 箇所残り、また、$M-1$ 回の操作で $M-1$ 箇所の赤い点線部分を消すことができます。したがって、可能な限り内側の (端でない) 赤い点線部分を消し (1 回あたり幸福な人は 2 人増える)、最後に残った赤い点線部分が内側にあるならそれを端に移動させる (幸福な人は 1 人増える) のが最適です。なぜなら、1 回の操作で幸福な人は高々 2 人しか増えませんが、実際に 2 人ずつ増やすことができるからです。

すなわち、結局は幸福な人 $+2$ 人を X 回でき、$+1$ 人を Y 回でき、K 回までの操作で何人幸福な人を増やせるか、といった問題になります。X, Y は赤い点線部分の数、すなわち S 中に登場する 'RL' の数と、S_1, S_N により定まります。この解法の時間計算量は $O(N)$ です。

別解として、文字列 S を LL...L, RR...R, ..., LL...L のように分解し、奇数番目の RR...R または LL...L の塊について前から順にそれぞれ 1 回の操作で向きを反転させていき、最後に幸福な人数を数える方法もあり、こちらも $O(N)$ で動作します。

E: Second Sum

P の各要素について、$P_i = X_{L,R}$ となるような組 (L, R) の個数を C_i とすると、

$$\sum_{L=1}^{N-1} \sum_{R=L+1}^{N} X_{L,R} = \sum_{i=1}^{N} P_i \times C_i$$

となります。また、$P_i = X_{L,R}$ となる組 (L, R) について、P が順列であることから、$L \leq i \leq R$ を満たします。以降、C_i を求めます。

集合 $S_i = \{j | j < i, P_i < P_j\}, T_i = \{j | j > i, P_i < P_j\}$ を考えます。S_i の中で二番目に大きい要素を w_i、最大の要素を x_i とし、T_i の中で最小の要素を y_i、二番目に小さい要素を z_i とします。S_i, T_i はインデックスの集合であることに注意してください。このとき、$w_i < x_i < i < y_i < z_i$ です。$P_i = X_{L,R}$ となるのは、$w_i < L \leq x_i < i \leq R < y_i < z_i$ のときと、$w_i < x_i \leq L < i < y_i \leq R < z_i$ のときに限られることがわかります。したがって、$C_i = (x_i - w_i) \times (y_i - i) + (i - x_i) \times (z_i - y_i)$ となります。

P の要素を大きい順番に見ていくことにすると、 ある時点までに見た全ての要素は、その時点で見ている要素より大きい要素になっています。したがって、順序集合を扱うデータ構造 (C++ における set, multiset など) を用いてインデックスの集合を管理することで、w_i, x_i, y_i, z_i は 各 i に対し $O(\log N)$ で求めることができます。また番兵として S_i に 0 を二つ、T_i に $N + 1$ を二ついれておくと、境界条件を簡潔に実装することができます。全体の計算量は $O(N \log N)$ です。

F: Many Slimes

この問題は以下の問題と等価です。

深さ N の完全二分木を考える。
葉に集合 S の要素を 1 つずつ書く。また、葉以外の頂点について、子に書かれた数の最大値をボトムアップに書いていく。
葉以外の任意の頂点について、子に書かれる数が相違なるように葉への要素の割当を決めることができるか。

S の大きい要素から (同じ数はまとめて) 葉への割当を決めていくことを考えます。

また、要素を割り当てた葉から根までのパス上の頂点を毎回黒く塗るものとします。

ある同じ数を 2 つの葉に割り当てるとき、2 つの葉を結ぶパス上に黒く塗られた頂点がまだなければ、2 つの葉の LCA で条件が満たされなくなるので不適です。見方を変えると、これは黒い頂点を取り除いて木を分割したとき、同じ連結成分の葉に割り当ててしまうことと等価です。逆にこのような葉のペアがなければ、その割当は条件を満たしています。

この事実を考えると、連結成分をできるだけ多くしておきたいので、頂点数の多い連結成分の葉から貪欲に養素を割り当てていくアプローチが有効です。このとき最後まで上のような事態が起こらなければ答えは可能、起これば不可能です。