F – Insert Sum Sort 解説

原案：Nerve
改題・解説：olphe

問題概要

長さ N の数列 A に対して以下の手順でソートする。

・空の数列Bを用意する。

・i 回目の操作で $A_i \geq \sum_{k=1}^{j-1} B_k$ を満たす最大の j に対し、A_i を挿入する。

ある順列の一部を空欄にした数列 A が与えられる。

空欄を適切に埋めて作られる順列の候補のうち、
上記の手順で昇順にソートされる順列の個数を求めよ。

・$1 \leq N \leq 2 \times 10^5$

・与えられる数列 A はある順列の一部を空欄にしたもの

操作の実験

昇順にソートされる数列が持つ性質は？

- 降順の数列 (5, 4, 3, 2, 1)

  常に先頭に挿入される ⇒ (1, 2, 3, 4, 5)　※ソート可能

- 適当な数列 (4, 5, 2, 3, 1)

  (4) ⇒ (4, 5) ⇒ (2, 4, 5) ⇒ (2, 3, 4, 5) ⇒ (1, 2, 3, 4, 5)　※ソート可能

- 適当な数列 (1, 3, 4, 5, 2)

  (1) ⇒ (1, 3) ⇒ (1, 3, 4) ⇒ (1, 3, 5, 4) ⇒ (1, 2, 3, 5, 4)　※ソート不可能

操作の実験

昇順にソートされる数列が持つ性質は？

- 降順をベースとして、昇順に反転している数字の個数で決まる？
- 実は自身より左にある小さな数字が1個以下（1を除く）のとき、昇順にソート可能
  - (4, 5, 2, 3, 1) や (1, 2, 5, 4, 3) は条件を満たす
  - 1は任意の位置に存在してよい

数列の条件の証明

自身より小さな数字が2個左に存在するときに昇順可能と仮定

= ある数 $1 < a < b < c$ が数列内で a, b, c $(b, a, c$ でも可$)$ の順に並んでると仮定

- $c = b + 1$ のとき

  $a + b > c$ であり、c は正しい位置に挿入されないため矛盾

- $c > b + 1$ のとき

  $b + 1$ は c より右側に存在する

  $a + b > b + 1$ であり、$b + 1$ は正しい位置に挿入されないため矛盾

同様に小さな数字が3個以上左に存在するときも証明可（省略）

小さな数字が1個以下左に存在するときに必ず昇順可能であることも証明可（省略）

条件を満たす数列の構築

① 1以外の数字を降順に並べる

(7 6 5 4 3 2)

② いくつかの区間に分ける

(7 6 | 5 | 4 3 2)

③ 各区間の右端の数字を左端にシフトする

(6 7 | 5 | 2 4 3)

④ 1を適当な位置に挿入する

(6 1 7 5 2 4 3) ⇒ ソート可能

- $N-1$ 個の数字の区間の分け方は 2^{N-2} 通り、1の挿入位置の決め方は N 通りなので

  長さ N の任意の順列のうち昇順にソートされる個数は $N \times 2^{N-2}$ 通り

考察の続き

- 問題では一部の数字の位置が確定した数列が与えられる

Sample2（3, -1, -1, 4, -1, -1） ⇒ (3, 6, 5, 4, 2, 1)
(3, 6, 5, 4, 1, 2) の 2通りが答え

⇒ DPによって数え上げることを考える

◆ 数えるために考えるべき要素

・今見てる位置より前に1は既に出現したか

・今見てる位置を含む区間は右端（シフトされる数字）が決まっているか

・シフトする数字が決まっていたとして、矛盾しない位置に数字が配置可能か

DP解法（想定解）

dp[i][j] := 数列をi番目まで見て (j : 1を既に使ったか) 区間が確定した順列の個数

shifts[i][j] := 数列をi番目まで見て (j : 1を既に使ったか) i+1番目の数字が未確定のときに
その数字をシフトできる左端候補の個数

state[j] := ある地点(j : 1を既に使ったか) を含む区間でシフトが確定しているか

- 確定していれば左端にシフトされた数字（元は右端にいたはずの数字）
- 未確定であれば0

- stateで数列の状態を管理しながらdp, shiftsを適切に更新することで計算可能
- 時間計算量：$O(N)$

DP解法（想定解）

- 降順で数字を仮置きしておく
- 今の地点の数字を右端とする区間の左端の候補数をshiftsで管理する

6 7 | 5 4 3 2 1
↑今ここ

決め方
⇒

① 左端が4, 右端が3

6 7 | 5 | 3 4 2 1

➢ これ以降左端に選べない5も確定

② 左端が5, 右端が3

6 7 | 3 5 4 | 2 1

➢ 区間が確定した①,②の通り数をdpに加算

③ 区間を決めない

6 7 | 5 4 3 2 1

青：区間確定済み

緑：未確定

実装例

```
vector<int> state(2); // シフトが確定していないとき0，確定していたら左端にシフトした数字
dp[0][0] = 1;
for (int i = 0; i < N; i++) {
    for (int j = 0; j < 2; j++) {
        int desc_num = N - i + j; // シフトが起きていなければ現在の位置にいるはずの数字
        bool valid = true;
        // 入力の整合性を検証
        if (A[i] >= 2) {
            // シフト確定していて、確定している数字ではないならダメ
            if (state[j] != 0 and desc_num + 1 != A[i]) valid = false;
            // シフト確定していなくて、ありえない位置にある数字ならダメ
            if (state[j] == 0 and A[i] > desc_num + 1) valid = false;
        }
        // 1を既に使っていて1が出てきたらダメ
        if (A[i] == 1 and j == 1) valid = false;

        if (!valid) continue; // 有効な状態でないなら無視

        if(A[i] == 1) {
            dp[i + 1][1] += dp[i][0];
            shifts[i + 1][1] += shifts[i][0];
        } else if (A[i] == -1) {
            // 1を使う
            if(j == 0) {
                dp[i + 1][1] += dp[i][0];
                shifts[i + 1][1] += shifts[i][0];
            }

            if(state[j] == 0) {
                // シフト未確定でA[i]未確定
                dp[i + 1][j] += dp[i][j] + shifts[i][j];
                shifts[i + 1][j] += dp[i][j] + shifts[i][j];
            }else{
                // シフト途中または終了
                dp[i + 1][j] += dp[i][j];
                // 今までのシフト候補はなくなる
            }
        } else {
            // 2以上の数字
            dp[i + 1][j] += dp[i][j];
            if (A[i] == desc_num + 1) {
                // 右シフトされた数字
                shifts[i + 1][j] += shifts[i][j]; // 今の位置より左にシフト可能
            }
        }
    }
```

・stateの更新部分

```
// stateを更新
for (int j = 0; j < 2; j++) {
    int desc_num = N - i + j;
    // 左シフト
    if (A[i] >= 2 and A[i] < desc_num) state[j] = A[i];
    // シフト終了
    if (state[j] == desc_num) state[j] = 0;
}
```