ABC 136 解説

DEGwer, drafear, evima, satashun

2019 年 8 月 4 日

A: Transfer

容器 1 にはあと $A-B$ ミリリットルまで入れることができます。よって、容器 2 から移す水の量 (ミリリットル) は $A-B$ と C の小さい方になります。したがって、容器 2 に残る水の量は $C-\min(A-B, C)$ ミリリットルになります。C++ 言語での実装例を以下に示します。

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

using namespace std;

int main() {
  int A, B, C; cin >> A >> B >> C;
  int cap = A - B;
  int use = min(C, cap);
  int ans = C - use;
  cout << ans;
}
```

B: Uneven Numbers

(原案: DEGwer, 準備・解説: evima)

N は 10 万以下とあまり大きくなく、$1, 2, \ldots, N$ という N 個の数の桁数を直接数えるのが楽です。

そして、それぞれの数の桁数を数える最も素直な方法は、数を文字列に変換してその長さを数えるというものでしょう。最も効率的な方法ではありませんが、今回はこれで十分です。

以下に Python による実装例を二つ掲載します。他の言語でもこれらと似たような実装が可能でしょう。

```python
N = int(input())
ans = 0
for i in range(1, N + 1):
    if len(str(i)) % 2 == 1:
        ans += 1
print(ans)
```

```python
N = int(input())
print(len(list(filter(lambda x: len(str(x)) % 2 == 1, range(1, N + 1)))))
```

C: Build Stairs

右のマスから順に操作を選ぶことにすると、できるだけ高くしておいた方が後で選択肢が増えるので、

- 何もしなくて良いなら何もしない
- そうでなくて、右のマスよりもちょうど 1 高いなら 1 低くする
- そうでないならどう頑張ってもダメなので答えは No

を右から順に行い、うまく構成できれば Yes を出力することで $O(N)$ の計算時間で答えを求めることができます。また、同様の考え方で、左から順に行うこともできます。

他の方法としては、i 番目までの高さの最大値を M_i としたときに、全ての $i = 1, 2, ..., N$ について、$H_i \geq M_i - 1$ が成り立てば Yes、そうでなければ No と判定することもできます。これは帰納法によって示すことができます。

実際に「$\forall i. H_i \geq M_i - 1 \Leftrightarrow$ 答えは Yes」を示してみます。

$\Leftarrow$ は簡単です。対偶を取ると、示すべきは「$\exists i. H_i < M_i - 1 \Rightarrow$ 答えは No」なので、これは自明です[*1]。

$\Rightarrow$ は帰納法で示します。

- $N = 1$ のときは必ず答えは Yes です。
- $N = N' + 1$ のとき、仮定より $i = 1, 2, ..., N'$ について条件を満たすので、帰納法の仮定より $H_1, H_2, ..., H_{N'}$ に対する解 (単調非減少にする操作の仕方) が存在します。この解への操作に上書きして、$H_i = M_N$ となる $i \in \{1, 2, ..., N'\}$ について 1 低くする選択肢を選ぶとしても $H_1, H_2, ..., H_{N'}$ に対する解となっていて、これに加えて H_N については何もしない選択をすると $H_1, H_2, ..., H_N$ に対する解が構成でき、Yes です。

[*1] $H_i < M_i - 1$ となる i を x とし、$H_j = M_x$ となる j を y とします。すると、$y < x$ かつ $H_x \leq H_y - 2$ が成り立ちます。つまりあるマスより右側のマスで、高さが 2 以上低いマスが存在するのでどう頑張ってもこの 2 マスに対する条件を満たすことができません。

D: Gathering Childrens

“RR...RLL...L” の部分に分解すると、他の部分と行き来することはないので独立に考えることができます。例えば、“RRLRL” は “RRL” と “RL” について独立に考えることができます。以降、“RR...RLL...L” の形についてのみ考えます。

十分な回数移動を行うので、境界部分、すなわち “RL” の部分に子供が集まり、他の部分には誰も居ない状態になります。さらに、最初左から偶数番目のマスにいた子供と奇数番目のマスにいた子供が同じマスに集まることはなく、逆に左からの位置の偶奇が等しいマスにいた子供たちは最終的には全員同じマスに集まります。移動回数は 10^{100} 回と偶数回なので、“RL” の “R” には “R” から偶数マス離れた子どもたちが集まり、“RL” の “L” には “L” から偶数マス離れた子どもたちが集まります。

このように計算することで時間計算量 $O(N)$ で答えを求めることができます。

実装では、文字列を連続した文字ごとに分解する (例えば “AABCCC” を (‘A’, 2), (‘B’, 1), (‘C’, 3) のように分解する) ランレングス圧縮のライブラリがあると便利です。

別解として、次のような解法もあります。マス数は高々 10^5 個なので、10^5 回移動した後の状態を求められれば十分です。この状態と 10^{100} 回移動した後の状態は一致します。また、左から i 番目のマスをマス i とし、$dp(i,j)$ を「最初マス j に居た子供が i 回の移動の後に居るマス」と定義します。すると、$dp(1,j)$ は S_j が L なら $j-1$、R なら $j+1$ です。さらに、任意の i_1, i_2, j について $dp(i_1+i_2, j) = dp(i_2, dp(i_1, j))$ なので、ダブリングにより $dp(X,j)$ を $O(NlogX)$ で計算できます。$X = 10^5$ について求めたいので、十分間に合います。ダブリングとは、x^n を $x^1, x^2, x^4, x^8, ...$ を組み合わせて計算する (例えば $x^{10} = x^8 \times x^2$ と計算します) 繰り返し二乗法を応用した手法で、今回の場合だと $dp(10,j) = dp(8, dp(2,j))$ と計算します。ここで、$dp(2^{n+1}, j)$ は $dp(2^n, dp(2^n, j))$ と計算します。この手法を用いれば、例えば問題が 10^{100} 回の移動後の状態ではなく K 回の移動後の状態を求めるものであったり、L,R といった単純な移動でなく右に R_i マス (R_i が負なら左に $-R_i$ マス) 移動といった場合にも答えを求めることができます。

E: Max GCD

まず、K 回の操作で A に対してどのような操作ができるか考えてみましょう。$A_1+d_1, A_2+d_2, \cdots A_N+d_N$ になるとします。

このとき、各操作で和が変化しないことに注目すると、$d_1 + d_2 + \cdots + d_N = 0$ である必要があります。また、d のうち正のものの和が K より大きい場合はありえません。負についても同様であり (和が 0 の時は自動的に満たされますが)、これら全てが満たされる場合はそのように操作可能であることが帰納的に示せます。

答え (d とします) の候補としては、$S = A_1 + A_2 + \cdots + A_N$ としたとき、d が S の約数である必要があります。d を S の約数全て試すことにして固定しましょう。このような d の候補は $O(sqrt(N * max(A)))$ 個あります。

まず、$A_1, \cdots A_N$ を d で割った余りを $r_1, r_2, \cdots r_N$ としましょう。0 であるものはなく、これは小さい順にソートされているとします。このとき、d を正にするものについては最低 $(d - r_i)$ の和の回数の操作が必要です。d を負にするものについては最低 r_i の和の回数の操作が必要です。それぞれ、d の倍数回の操作は余分に行うことができます。よって、r の小さい方から途中までを d を負に設定し、そこからは正になるようにする形のみを考えればよく (r の和が d の倍数であることに注意しましょう)、累積和などを用いて全ての区切り方の d が正のものの回数の和と負のものの回数を求めることができます。

よって、d を固定した場合には $O(NlogN)$ で解けるので、この問題は $O(sqrt(N * max(A)) * NlogN)$ で解けました。

F: Enclosed Points

まず、空でない部分集合は $2^N - 1$ 通りあります。これらについて直接 f の値を考えるのではなく、各点が長方形に含まれるような集合が何個あるかを各点について求めることを考えます。

ある点 P に注目しているとしましょう。P を中心に、P との x, y 座標の大小関係は 4 方向ありますが、このそれぞれに点が存在するかどうかという状態を固定すればこの P が長方形に含まれるかどうかが決まることがわかります。ある領域に K 個の点があるとした時、この領域に点が存在する場合は $2^K - 1$ 通りで、点が存在しない場合は 1 通りです。例えばですが、点がそれぞれに存在するかどうかの 2^4 通りについて、長方形に P が含まれる場合にこれらをかけ合わせて足していけばよいです。(P が集合に含まれるかどうかについても場合分けが必要です)

よって、各点 (x_i, y_i) について $x < x_i, y < y_i$、$x < x_i, y > y_i$、$x > x_i, y < y_i$、$x > x_i, y > y_i$ それぞれの領域についてそれぞれ何点存在するかが求まれば、答えを求めることができます。これはまず点の座標を $1 - N$ に圧縮した後、点を x 座標についてソートし、x の小さい方からと大きい方からで 2 回 segment tree / BIT (fenwick tree) を用いて各 y に存在する点の個数を管理していけば $O(NlogN)$ で計算できます。

以上でこの問題は $O(NlogN)$ で解けました。

解答例