ABC 134 解説

yuma000, drafear, DEGwer, evima, gazelle, potetisensei

2019 年 7 月 20 日

A. Dodecagon(writer : yuma000)

$3 * r * r$ を出力すればよいです。以下が C++ のサンプルコードです。

```
#include<iostream>
using namespace std;

int main() {
        int R;cin>>R;

        int area=R * R * 3;

        cout<<area<<endl;

        return 0;
}
```

B: Golden Apple

1 人の監視員を配置すると連続した $2D+1$ 本のりんごの木を監視できます。前から順に詰めて配置する (例えば最初の監視員には $1, 2, ..., 2D+1$ 番目の木を監視させる) のが最適なので、答えは N を $2D+1$ で割った切り上げになります。一般的に、整数 A, B に対して $\frac{A}{B}$ の切り上げは $\frac{A+B-1}{B}$ の商と等しいです。したがって、今回の場合は $\frac{N+2D}{2D+1}$ の商が答えになります。C++ での実装例を以下に挙げます。

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

using namespace std;

int main() {
  int N, D; cin >> N >> D;
  int ans = (N + D * 2) / (D * 2 + 1);
  cout << ans << endl;
}
```

C: Exception Handling

N が数千以下 (言語によっては数万以下) なら、各問で A_i 以外の $N-1$ 個の要素すべてに対してループを回して最大値を直接求めても実行制限時間の 2 秒に間に合います。しかし実際には N は最大で 20 万であり、この方針では C++ でも間に合う望みはありません。計算時間を削減する方針を以下に二つ示します。

方針 1: 場合分け

ほとんどの場合、問いの答えは N 個すべての要素のうちの最大の値 $A_{\max}$ です。唯一の例外は問いで取り除かれる値 A_i が $A_{\max}$ と等しい場合で、この場合の答えはすべての要素のうち 2 番目に大きい値 A_{second} (数列中に $A_{\max}$ が複数回現れる場合は $A_{\mathrm{second}} = A_{\max}$ とします) です。問いの処理を始める前に $A_{\max}$ と A_{second} をあらかじめ求めておけば、各問を直ちに処理できます。なお、A_{second} を最も簡単な実装で求める方法は、与えられた数列をコピーして言語の標準ライブラリでソートすることでしょう (やや「余計」な計算をすることになりますが十分高速です)。

方針 2: 両端から攻める

$j = 0, 1, \ldots, N-1$ に対し、$A_1, A_2, \ldots, A_j$ のうちの最大の値を left_j とします (ただし $\mathrm{left}_0 = 0$ とします)。$j \geq 1$ のとき $\mathrm{left}_j = \max(\mathrm{left}_{j-1}, A_j)$ [*1] であることに注意すると、これらの値は一周のループですべて求められます。また、$j = 2, \ldots, N+1$ に対し、$A_j, A_{j+1}, \ldots, A_N$ のうちの最大の値を right_j とします (ただし $\mathrm{right}_{N+1} = 0$ とします)。$j \leq N$ のとき $\mathrm{right}_j = \max(\mathrm{right}_{j+1}, A_j)$ であることに注意すると、これらの値も一周のループですべて求められます。問いの処理を始める前にこれらの値をあらかじめ求めておけば、各問 i の答えを $\max(\mathrm{left}_{i-1}, \mathrm{right}_{i+1})$ として直ちに求められます。

[*1] $\max(a, b)$ は a と b のうち大きい方 (より厳密には小さくない方) を表します

D: Preparing Boxes

大きい数が書かれた箱からボールを入れるかを決めていくことにします。こうすると、整数 i が書かれた箱にボールを入れるか決めるとき、i 以外の i の倍数が書かれた箱については、すでにボールを入れるかが決まっています。それらの箱に入ったボールの総和の偶奇が a_i と異なる場合は箱にボールを入れて、そうでないときはボールを入れないことにします。このようにしてボールを入れるかを決めていくと、与えられた条件をすべて満たすようにボールを入れることができます。このアルゴリズムを愚直に実装すると $O(\frac{N}{1} + \frac{N}{2} + ... + \frac{N}{N})$ 程度の計算量がかかります。これは一見 $O(N^2)$ に見えますが、丁寧に解析すると $O(NlogN)$ で抑えることができます。よって十分高速です。

E: Sequence Decomposing

結論から言うと、答えは「与えられた数列の、**広義**単調減少列の長さの最大値 L」です。これを証明します。

まず、問題の答えが、L 以上であることは明らかです。$\{A_i\}$ から、長さ L を与える広義単調な部分列を適当に取ります。すると、この部分列に属する要素同士は必ず異なる色で塗られている必要があるため、L 色以上用いる必要があります。

逆に、そのような部分列 $\{A_{k_i}\}$ を取った時、任意の A_{k_i} と $A_{k_{i+1}}$ の間の要素は、必ず「A_{k_i} より大 または $A_{k_{i+1}}$ 未満」が成立します。したがって、A_{k_i} または $A_{k_{i+1}}$ と同じ色で塗り分ければ条件を満たすことが出来ます。A_{k_1} より左側の要素、A_{k_L} より右側の要素についても、明らかにそれらと同じ色で塗れば条件を満たすため、答えは L 色以下であることもわかり、結局答えは L に一致することが示せました。

結局、求めるべきものは LIS と同等であるため、$O(N \log N)$ で求める事が出来ます。

F: Permutation Oddness

簡単のため、問題を以下のように言い換えます。

> うさぎ 1, うさぎ 2, ..., うさぎ N と、かめ 1, かめ 2, ..., かめ N がいる。うさぎとかめのペアを N 組作る方法のうち、ペアになったうさぎの添字とかめの添字の差の和 (これを「奇妙さ」と呼ぶ) が K になるものはいくつあるか。

結論から言うと、以下のように状態を定義することで、解を動的計画法で求めることができます。

- $dp[i][j][k][l]$ = うさぎ 1, ..., うさぎ i, かめ 1, ..., かめ i までを見て、ペアにするのを保留にしている (かめ $i+1$ 以降とペアにする) うさぎの数がj、ペアにするのを保留にしている (うさぎi+1以降とペアにする) かめの数が k、確定している奇妙さが l であるときの場合の数。

ここで確定している奇妙さとは、たとえばうさぎ i をかめ $i+1$ 以降とペアにする場合、少なくとも 1 の奇妙さが確定する、といったことを意味します。ところで、ペアを保留にしているうさぎの数とかめの数は一致します。よって DP テーブルの二次元目と三次元目は 1 つにまとめることができます。このとき DP の遷移は以下のように書けます。

- $dp[i][j][k] = (2*j+1)*dp[i-1][j][k-2*j]+(j+1)*(j+1)*dp[i-1][j+1][k-2*j]+dp[i-1][j-1][k-2*j]$

ここで右辺の第 1 項は、うさぎ i とかめ i をペアにする場合および、うさぎ i とかめ i のいずれかを保留にしておいた動物とペアにする場合を表します (ペアを保留にしている動物の数だけ、確定している奇妙さが増えます)。右辺の第 2 項は、うさぎ i とかめ i の両方を保留にしておいた動物とペアにする場合を表します。そして右辺の第 3 項は、うさぎ i とかめ i のどちらもペアにするのを保留する場合を表しています。

この動的計画法は状態数が $O(N^4)$、遷移が $O(1)$ なので、全体で $O(N^4)$ の計算量で解を求めることができます。