ABC 167 解説

gazelle, kort0n, kyopro_friends, potetisensei, sheyasutaka, tempura0224

2020 年 5 月 10 日

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

A: Registration

T の最後の 1 文字を消去して得られる文字列が S と一致していれば答えは Yes であり, そうでなければ No です.
以下は C++ での実装例です.

```
#include <bits/stdc++.h>
using namespace std;

string S, T;
void input() {
    cin >> S >> T;
}

void solve() {
    T.pop_back();
    if(S == T) cout << "Yes" << endl;
    else cout << "No" << endl;
}

int main() {
    input();
    solve();
    return 0;
}
```

B: Easy Linear Programming

大きい数が書かれたカードから貪欲に選んでいくのが最適です。

ナイーブにシミュレートすると時間がかかってしまうので、以下のコードのように場合分けをするとよいです。

Listing 1　C++ による実装例

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
using P = pair<ll, ll>;
#define rep(i, n) for(ll i = 0; i < (ll)n; i++)

int main() {
    int a, b, c, k;
    cin >> a >> b >> c >> k;
    if(k <= a) cout << k << endl;
    else if(k <= a + b) cout << a << endl;
    else cout << a - (k - a - b) << endl;
    return 0;
}
```

C: Skill Up

どの参考書を購入するかを決めると、それらを読むことによってすべての理解度を X 以上にできるかどうかの判定は難しくありません。
そこで、購入する参考書の組み合わせ全てについて上記の判定をし、条件を満たすもののなかでの合計金額の最小値を求めることにしましょう。
購入する参考書の組み合わせは、各参考書について 買う or 買わない を選ぶ場合の数なので全部で 2^N 通りです。1 回あたりの判定は $O(NM)$ でできるので計算量は全体で $O(2^N NM)$ であり、今回の制約においては十分高速です。
購入する参考書の組を全探索する部分の実装は、「深さ優先探索」あるいは 集合を整数にエンコードする (いわゆる「bit 全探索」) などを用いるとよいでしょう。

以下は C++ における bit 全探索での実装例です。https://atcoder.jp/contests/abc167/submissions/13086283

D: Teleporter

町 1 から愚直に移動をシミュレートしていくと，N 回以内に既に来た町にもう一度帰りつきます．そこでその町を「香り高い町」として，香り高い町から移動を続けて何度目に香り高い町に帰りつくかを (これまた愚直に) 求めます．あとは，残りの移動回数をこれで割ったあまりは N 未満なので，愚直にシミュレートすればよいです．

E: Colorful Blocks

「隣り合うブロックの組であって同じ色で塗られている組が、丁度 k 組である」場合の色の塗り方を考えます.
条件を満たす為の必要十分条件は, 左端を除いた $N-1$ 個のブロックのうち, k 個が左隣のブロックと同じ色で塗られており, $N-1-k$ 個が左隣と異なる色で塗られていることです.
このような色の塗り方は, 左端のブロックの色の選び方及び前述の k 個のブロックの選び方も考慮すると,

$$M \times {}_{N-1}\mathrm{C}_k \times 1^k \times (M-1)^{N-1-k}$$

で表されます. この値を $k = 0, 1, \cdots, K$ について足し合わせることで, 答えを求めることが出来ます.

F: Bracket Sequencing

概要

文字列のうち '(' と ')' が連続している箇所があれば、その 2 文字を予め取り除いても結果に影響しません。この操作を繰り返すことで、各 S_i は「')' が 0 個以上連続したあと、'(' が 0 個以上連続するような文字列 (空文字列含む)」となります。このときの')' の個数を A_i, '(' の個数を B_i とします。

S_i たちを次の順にソートして連結したものが括弧列になっていれば Yes、そうでなければ No が答えです。

前半： $B_i - A_i \geq 0$ であるような S_i たちを、A_i が小さい順に並べたもの

後半： $B_i - A_i < 0$ であるような S_i たちを、B_i が大きい順に並べたもの

正当性は、「もしこのような順になっていない隣接 2 項があれば、その 2 項の順序を入れ替えても損をしない」という方針で示すことができます。

気持ち

括弧列は、「全ての括弧がうまくペアになっている文字列」です。文字列を前から順に見て「すでに全ての括弧がペアになっているにも関わらず')' が登場した」とき、そのときに限り、その文字列は括弧列ではありません。なので、'(' が増える文字列を先に使うのが良さそうです。すでにある'(' の貯金分以上の')' をつなげることは出来ないので、A_i の小さい順に使うのが良さそうです。

また、文字列が括弧列であることと、文字列全体を反転し、'(' を')' に、')' を'(' に置き換えた文字列が括弧列であることは同値です。したがって後半も同様の条件を得ることができます。

実装

A_i, B_i は括弧を取り除く操作を実際に行うことなく、次のようにして文字列長に対して線形な時間で求めることができます。

$$A_i = S_i\text{の prefix についての、')' の個数} - \text{'(' の個数 の最大値}$$
$$B_i = S_i\text{の suffix についての、'(' の個数} - \text{')' の個数 の最大値}$$