

ABC 131 解説

sheyasutaka, yuma000, DEGwer, gazelle, satashun

2019 年 6 月 22 日

A: Security

正解までの道のりを細かく分割すると次の通りです。

1. 標準入力から文字列を受け取る。
2. 文字列の隣り合う文字同士を比較することで、文字列が「入力しづらい」かを判定する。
3. 判定結果に対応する文字列 (“Good” か “Bad”) を標準出力に出力する。

手順 2 (判定) については、正規表現を用いて簡潔に書くことも可能です。実装例を以下に示します。

Listing 1 C++ での実装例

```
1 #include <iostream>
2 #include <string>
3 using std::cin;
4 using std::cout;
5 using std::endl;
6 using std::string;
7
8 int main(void){
9     string s;
10
11     // 入力
12     cin >> s;
13
14     // 隣り合う文字を比較して判定
15     bool isgood = true;
16     if (s[0] == s[1]) isgood = false;
17     if (s[1] == s[2]) isgood = false;
18     if (s[2] == s[3]) isgood = false;
19
20     // 出力
21     if (isgood) {
22         cout << "Good" << endl;
23     } else {
24         cout << "Bad" << endl;
```

```

25     }
26
27     return 0;
28 }

```

Listing 2 Ruby での実装例

```

1 # 入力
2 s = gets().chomp()
3
4 # 正規表現を使って判定・出力
5 if s =~ /(.)\1/ then
6     puts "Bad"
7 else
8     puts "Good"
9 end

```

Listing 3 Brainf*ck での実装例

```

1 >++++>>,<++++[->>,-<->>+<][[-]<<->>]>>[-<<+>>]<<<+<[[-]>+++++++++[-<+++++>]<.<
2 [--<++++>]<-- .+++>]>[+++++++[-<+++++++>]<- .+++[->++++<]>..[-<+>>]<<-----
3 >>]<<.<

```

B. Bite Eating(writer : yuma000)

N 個のりんごのおいしさは、 $L, L+1, L+2, \dots, L+N-1$ です。また、この中から、最もおいしさの絶対値の小さいりんごを食べるのが、最適なので、そのりんごを除いた全てのりんごのおいしさの和を出力すればいいことが分かります。

以下が、C++ のサンプルコードです。

```
1    #include<iostream>
2    using namespace std;
3
4    int main() {
5
6        int N,A;
7        cin>>N>>A;
8
9        int L=A;
10       int R=A+N-1;
11
12       int eat;
13       if(R<=0)eat=R;
14       else if(L>=0)eat=L;
15       else eat=0;
16
17       int answer=(R+L)*(R-L+1)/2-eat;
18
19       cout<<answer<<endl;
20
21       return 0;
22
23    }
```

C: Anti-Division

条件を満たす B 以下の整数の個数から条件を満たす $A - 1$ 以下の整数の個数を引いたものが答えです。以下、条件を満たす B 以下の整数の個数を求めることを考えましょう。同様に、条件を満たす $A - 1$ 以下の整数の個数も求めることができます。

B 以下の整数の個数 (B 個) から、 B 以下の C の倍数の個数と B 以下の D の倍数の個数を引き、重複して引いたものを足し戻せばよいです。 B 以下の C, D の倍数の個数はそれぞれ $B/C, B/D$ の小数点以下を切り捨てた数です。重複して引いた数は、 C, D の最小公倍数を l として、 B/l の小数点以下を切り捨てた数です。 l は、 C, D の最大公約数を g として CD/g と求められます。 g は Euclid の互除法などを用いて求められるため、この問題を解くことができました。

D: Megalomania

直感的には、 A 切の差し迫っている仕事から片付けるのがよさそうです。実際にこの直感は正しく、仕事を B_i の小さい順に順番に行うのが最適です。つまり、以下のことが言えます。

仕事 N 件の並べ替えであって、この順に行えば全ての仕事が A 切に間に合うようなものを「いい順序付け」と呼ぶことにすると、

答えが Yes となる条件 B_i の小さい順に仕事を行えば、それが「いい順序付け」となる。

証明 十分性は明らかなので、必要性を示します。つまり、「いい順序付け」 $(A_1, B_1), (A_2, B_2), \dots, (A_N, B_N)$ があるとき、 B_i の昇順に並べ替えてもまた「いい順序付け」となることを示します。

k 番目の仕事が A 切に間に合う条件は、それまでの仕事にかかる時間の合計が A 切の時刻を上回らないことです (つまり $A_1 + A_2 + \dots, A_k \leq B_k$)。 $B_k > B_{k+1}$ のとき、 $(A_k, B_k), (A_{k+1}, B_{k+1})$ の順序を入れ替えても「いい順序付け」のままです。したがって、バブルソートの要領で B_i の昇順にソートしてもまた「いい順序付け」となります。

C++ や Python3 などの多くのプログラミング言語では、標準で用意されている関数などを用いることでソートができます。実装例を以下に示します。

Listing 4 C++ での実装例

```
1 // include, using 略
2
3 int n;
4 vector<pair<int, int> > tasks;
5
6 int main(void){
7     cin >> n;
8     for (int i = 0; i < n; i++) {
9         int a, b;
10        cin >> a >> b;
11        tasks.emplace_back(b, a); // A切 (, 所要時間)
12    }
13
14    sort(tasks.begin(), tasks.end());
15    int sum = 0;
16    bool isyes = true;
17    for (auto v : tasks) {
18        sum += v.second;
19
20        if (sum > v.first) {
21            isyes = false;
22            break;
23        }
24    }
25
```

```
26         cout << (isyes ? "Yes" : "No") << endl;
27
28         return 0;
29     }
```

E: Friendships

N 頂点の連結グラフには少なくとも $N - 1$ 本の辺が存在します。これらの辺によって結ばれた頂点間の最短距離は 1 になるため、 $K > \frac{N(N-1)}{2} - (N - 1) = \frac{(N-1)(N-2)}{2}$ のとき解が存在しないのは明らかです。

逆に $K \leq \frac{(N-1)(N-2)}{2}$ のとき解が存在することを、条件を満たすグラフを実際に構成することで示します。

頂点 1 とそれ以外の任意の頂点とを結ぶ $N - 1$ 本の辺だけが存在するグラフを考えます（このようなグラフをスターまたはうにと言います）。このグラフでは頂点 1 を除く任意の頂点間の最短距離は 2 であり、 $K = \frac{(N-1)(N-2)}{2}$ のときの解になっています。ここで、このグラフの最短距離が 2 である頂点同士を結ぶ辺を追加すると、他の頂点間の最短距離に影響を与えずに、その最短距離を 1 にすることができます。

よって、この操作を $\frac{(N-1)(N-2)}{2} - K$ 回繰り返すことで条件を満たすグラフを作ることができます。

F: Must Be Rectangular!

まず、このままでは考えにくいので問題を次のようにグラフの言葉で言い換えます。

頂点 $X_1, X_2, \dots, X_{10^5}$ と 頂点 $Y_1, Y_2, \dots, Y_{10^5}$ がある。点 (x, y) が存在するとき、頂点 X_x と Y_y の間に辺を張る。辺 $a-b, a-d, c-b, c-d$ のうちちょうど 3 辺が存在するような a, b, c, d を選び残りの 1 辺を追加する、という操作を何回行うことができるか。

初めに存在する点に対してグラフを構築すると、異なる連結成分を結ぶような辺は追加しようがないので、連結成分ごとに答えを求めて和を取ることで元の問題の答えが求まります。

ここで、1 つの連結成分については、2 部グラフの両側 (X の頂点と Y の頂点) の全ての組について (元から存在した辺を除き) 辺が追加されることが示せます。概略ですが、両側の頂点数が 2 以上の時は全域木を 1 つ取り、葉 (v) に繋がっているようなある辺 $v-u$ を選び、 u 側の連結成分について全ての辺を張った後それらの辺を利用して v と結んでいき、どちらかの頂点数が 1 ならばその時点で連結なことでにより全て結ばれています。

よって DFS などを用いて各連結成分について X と Y が何個含まれるか計算することでこの問題が解けました。