ABC 131 解説

sheyasutaka, yuma000, DEGwer, gazelle, satashun

2019 年 6 月 22 日

For International Readers: English editorial starts on page 9.

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「入力しずらい」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

```
#include <iostream>
#include <string>
using std::cin;
using std::cout;
using std::endl;
using std::string;

int main(void){
        string s;

        // 入力
        cin >> s;

        // 隣り合う文字を比較して判定
        bool isgood = true;
        if (s[0] == s[1]) isgood = false;
        if (s[1] == s[2]) isgood = false;
        if (s[2] == s[3]) isgood = false;

        // 出力
        if (isgood) {
                cout << "Good" << endl;
```

```
        } else {
                cout << "Bad" << endl;
        }

        return 0;
}
```

Listing 2　Ruby での実装例

```
# 入力
s = gets().chomp()

# 正規表現を使って判定・出力
if s =~ /(.)\1/ then
        puts "Bad"
else
        puts "Good"
end
```

Listing 3　Brainf*ck での実装例

```
>+++>>,<+++[->>,[-<->>+<]<[[-]<<->>]>>[-<<+>>]<<<]+<[[-]>++++++++++[-<++++++>]<.
[--<+++>]<--.+++>]>[++++++++[-<+++++++++>]<-.+++[-->+++<]>..[-<<+>>]<<----------
>>]]<<.
```

B. Bite Eating(writer : yuma000)

N 個のりんごのおいしさは、L,$L+1$,$L+2$,...,$L+N-1$ 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。

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

int main() {

    int N,A;
    cin>>N>>A;

    int L=A;
    int R=A+N-1;

    int eat;
    if(R<=0)eat=R;
    else if(L>=0)eat=L;
    else eat=0;

    int answer=(R+L)*(R-L+1)/2-eat;

    cout<<answer<<endl;

    return 0;

}
```

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。同様にして、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，〆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〆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」$(A_1, B_1), (A_2, B_2), \ldots, (A_N, B_N)$ 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 番目の仕事が〆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〆切の時刻を上回らないことです (つまり $A_1 + A_2 + \cdots, A_k \leq B_k$)．$B_k > B_{k+1}$ 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をできます．実装例を以下に示します．

Listing 4　C++ での実装例

```
// include, using 略

int n;
vector<pair<int, int> > tasks;

int main(void){
        cin >> n;
        for (int i = 0; i < n; i++) {
                int a, b;
                cin >> a >> b;
                tasks.emplace_back(b, a); // 〆切(, 所要時間)
        }

        sort(tasks.begin(), tasks.end());
        int sum = 0;
        bool isyes = true;
        for (auto v : tasks) {
                sum += v.second;

                if (sum > v.first) {
                        isyes = false;
                        break;
                }
        }

```

```
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 $\frac{(N-1)(N-2)}{2} - K$ 回く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 $X_1, X_2, ..., X_{10^5}$ と 頂点 $Y_1, Y_2, ..., Y_{10^5}$ 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。

ABC 131

sheyasutaka, yuma000, DEGwer, gazelle, satashun

06/22/2019

A: Security

Here is a step-by-step instruction how to solve the problem:

1. Receive the string form the standard input.
2. Judge if the string is "hard to enter" by checking the adjacent characters.
3. Output a string corresponding to the judge result ("Good" or "Bad") to the stdout.

The second procedure (judgement) can be written concisely with regular expressions. The following are implementation examples.

Listing 1: An implementation example in C++

```
#include <iostream>
#include <string>
using std::cin;
using std::cout;
using std::endl;
using std::string;

int main(void){
        string s;

        // Input
        cin >> s;

        // Compare the adjacent characters and judge
        bool isgood = true;
```

```
        if (s[0] == s[1]) isgood = false;
        if (s[1] == s[2]) isgood = false;
        if (s[2] == s[3]) isgood = false;

        // Output
        if (isgood) {
                cout << "Good" << endl;
        } else {
                cout << "Bad" << endl;
        }

        return 0;
}
```

Listing 2: An Implementation example in Ruby

```
# Input
s = gets().chomp()

# Compare the adjacent characters and judge
if s =~ /(.)\1/ then
        puts "Bad"
else
        puts "Good"
end
```

Listing 3: An implementation example in Brainf*ck

```
>+++>>,<+++[->>,[-<->>+<]<[[-]<<->>]>>[-<<+>>]<<<]+<[[-]>++++++++++[-<++++++>]<
[--<+++>]<--.+++>]>[++++++++[-<++++++++>]<-.+++[-->+++<]>..[-<<+>>]<<----------
>>]]<<.
```

B. Bite Eating(writer : yuma000)

The flavor of the N apples are L,$L+1$,$L+2$,...,$L+N-1$. It is optimal to eat the apple whose absolute value of taste is minimum, so it appears that you have to print the sum of all the apples except for that apple.

The following is a sample code in C++.

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

int main() {

    int N,A;
    cin>>N>>A;

    int L=A;
    int R=A+N-1;

    int eat;
    if(R<=0)eat=R;
    else if(L>=0)eat=L;
    else eat=0;

    int answer=(R+L)*(R-L+1)/2-eat;

    cout<<answer<<endl;

    return 0;

}
```

C: Anti-Division

The answer is the number of integers less than or equal to B that satisfies the condition, subtracted by the number of integers less than or equal to $A-1$ that satisfies the condition. Let's first think about how to find the number of integers less than or equal to B that satisfies the number; you can similarly get the number of integers less than or equal to $A-1$ that satisfies the condition.

The number is equal to the number of the integers no less than B ($= B$), subtracted by the number of multiples of C no less than B and the number of multiples of C no less than C, then added by the duplicate of them. The number of multiples of C, D is equal to B/C and B/D, rounded down, respectively. The number of duplicates are B/l rounded down, where l is the least common multiplier of C and D. l is equal to CD/g where g is the greatest common divisor of C, D. You can get g with Euclidean Algorithm, so here the problem could be solved.

D: Megalomania

Intuitively, it seems to be good to finish the job with earlier deadline formerly. Actually this intuition is right; it is optimal to finish the job in the increasing order of B_i. In other words,

Let "good permutation" be the permutation of N jobs, where you can complete all the jobs within the deadlines.

The condition where the answer is Yes If you do the work in the ascending order of B_i, that will be the "good permutation."

Proof The sufficiency is obvious. Now let's show the necessity: that is, if there exists a "good permutation" $(A_1, B_1), (A_2, B_2), \ldots, (A_N, B_N)$, after the permutation is sorted by the order of B_i, it still stays a "good order." You can finish the k-th job by the deadline if the time to take the k jobs so far is no less than the deadline of the job (that is, $A_1 + A_2 + \cdots, A_k \leq B_k$). If $B_k > B_{k+1}$, the permutation stays "good" if you swapped $(A_k, B_k), (A_{k+1}, B_{k+1})$. Therefore, you can sort the permutation in the manner of bubble sort so that the B_i will be a ascending order.

In most programming language like C++ and Python3, you can make use of standard library function for sorting. The implementation is below.

Listing 4: Implementation Example in C++

```
1 // include, using omitted
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); // (deadline, time
                      required)
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

A N-vertices connected graph has at least $N-1$ edges. The shortest distance of these each pair is equal to 1, so obviously, it is clear that there doesn't exist a solution if $K > \frac{N(N-1)}{2} - (N-1) = \frac{(N-1)(N-2)}{2}$.

Conversely, now we're showing that there exist a solution if $K \leq \frac{(N-1)(N-2)}{2}$ by constructing a graph that satisfies the condition.

First, let's consider the graph where vertex 1 and the all other vertices are connected, respectively, with $N-1$ edges in total (such graph is called "star", or in Japanese, "sea urchin"). In this graph, the shortest distance of an arbitrary pair of vertices without vertex 1 is equal to 2, so this can be the answer for $K = \frac{(N-1)(N-2)}{2}$. Here, if you add an edge to the pair of vertices where the shortest distance is 2, it will change to 1, while the shortest distances of other pairs remain unchanged.

Therefore, you can obtain the satisfying graph by repeating this operation for $\frac{(N-1)(N-2)}{2} - K$ times.

F: Must Be Rectangular!

First, for simplification, let's rephrase the problem with graph theory terms.

There are vertices $X_1, X_2, ..., X_{10^5}$ and vertices $Y_1, Y_2, ..., Y_{10^5}$. If there exist a point (x, y), add an edge between vertex X_x and Y_y. How many times can you perform a operation to select a, b, c, d where there exist 3 edges out of 4 edges $a - b, a - d, c - b, c - d$ and add the rest to the graph?

Considering the graph is constructed from the initial points, you can never add edge between different connected components, so you can find the answer by calculating the answer for each connected component and then summing them up.

It can be proved that for each connected component, the edges will be added between all the pairs of vertices between two sides of the bipartite graph (between vertices X and vertices Y), except for the edges that originally existed. In a nutshell, if the number of vertices of both sides are more than or equals to 2, you can find a spanning tree, select an edge $v - u$ where v is a leaf, add all the edges for each vertex of the side containing u, then add edges to v using those edges; if the number of vertices of either side is 1, it is connected already.

Therefore, the problem could be solved by counting how many vertices X and Y there are for each connected component by using DFS or something.