ABC 168 解説

sheyasutaka

2020 年 5 月 17 日

For International Readers: English editorial starts on page 8.

A: ∴ (Therefore)

1 の位ごとに場合分けをしても解けますが，より簡潔に書けないでしょうか？

C/C++ や C#, Java などには switch 文というものがあり，これを使うことで問題文通りの記述を直感的に書けます (C での実装例を以下に示します).

余談ですが，Ruby では case 文がこれにあたり，もっと高機能です.

Listing 1　C での実装例

```c
#include <stdio.h>

int main(void) {
        int n;
        scanf("%d", &n);

        switch (n % 10) {
        case 2:
        case 4:
        case 5:
        case 7:
        case 9:
                puts("hon");
                break;
        case 0:
        case 1:
        case 6:
        case 8:
                puts("pon");
                break;
        case 3:
```

```
22                  puts("bon");
23                  break;
24          }
25
26          return 0;
27  }
```

B: ... (Triples Dots)

長さが K 以下かどうか判定し，上回っていれば問題文通りの文字列操作を行ってから出力すればいいです.

PHP には truncate というそのものずばりな機能があり，これを使うのも手です.

Listing 2　C++ での実装例

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

#include <string>
using std::string;

int main(void) {
        int k;
        string s;

        cin >> k;
        cin >> s;

        if (s.size() > k) {
                s = s.substr(0, k) + "...";
        }

        cout << s << '\n';

        return 0;
}
```

Listing 3　Python3 での実装例

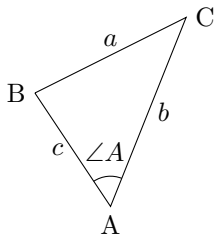
```
k = int(input())
s = input()

if len(s) > k:
        s = s[:k] + "..."

print(s)
```

C: : (Colon)

2 端点の座標を求めてからその距離を計算するのは面倒です．これを使ってみましょう:

余弦定理

三角形 ABC について, $a^2 = b^2 + c^2 - 2bc\cos\angle A$ が成り立つ．ここで, a, b, c は $|BC|, |CA|, |AB|$ のことを指す.



時針・分針の長さを b, c, 時針・分針の間の角度を $\angle A$ として上の式に代入すれば，求める長さ (の 2 乗) が求まります．角度については，「時針が $H + \frac{M}{60}$ 時間で動いた角度」から「分針が M 分で動いた角度」を引けばよいです (C++ 標準の cos 関数の場合，mod をとる必要はありません).

Listing 4　C での実装例

```c
#include <stdio.h>
#include <math.h>

#define PI 3.14159265358979323846264338327950L

int main(void) {
        int a, b, h, m;
        scanf("%d%d%d%d", &a, &b, &h, &m);

        long double rad = PI * 2 * (
                (long double)h / 12.0 + ((long double)m / 60.0) / 12.0 - (long
                     double)m / 60.0);

        long double rsq = (long double)(a * a + b * b) - (long double)(2 * a *
             b) * cosl(rad);

        printf("%20.20Lf\n", sqrtl(rsq));

        return 0;
}
```

D: .. (Double Dots)

ある部屋から部屋 1 に移動するために廊下を通る最小の回数のことを，その部屋の「深さ」と呼ぶことにします．特に，部屋 1 の深さだけが 0 です．

すると，以下の重要な事実が成り立ちます．

＜！＞＜！＞＜！＞＜！＞＜！＞＜！＞

深さ $d+1$ の部屋には，深さ d の部屋が少なくとも 1 つ繋がっている．

証明は簡単です (そうでない部屋があると，直感的に嫌な気分になります)．このことから，深さ $d+1$ の部屋の道しるべが深さ d の部屋を指すようにすれば，常に目標を達成できることがわかります．

実装の際には，幅優先探索で各部屋の深さを求めてから，辺を 1 つずつ見ていって随時更新すれば楽です．

E: · (Bullet)

$(A_i, B_i) = (0, 0)$ のイワシは他のどの個体とも仲が悪いので，「$(A_i, B_i) = (0, 0)$ なイワシをどれか 1 匹だけ選ぶ」「$(A_i, B_i) = (0, 0)$ なイワシをどれも選ばない」のいずれかです．勿論，前者の選び方はそのようなイワシの個数に等しいので，以下では $(A_i, B_i) = (0, 0)$ なイワシを除外して考えます．

イワシの「傾き」を $\frac{A_i}{B_i}$ として，これを既約分数で表すことを考えます．具体的には，次のように決めます．

- $B_i = 0$ のとき，傾きは $1/0$.
- $B_i > 0$ のとき，傾きは $\frac{A_i}{\gcd(A_i,B_i)} / \frac{B_i}{\gcd(A_i,B_i)}$.
- $B_i < 0$ のとき，傾きは $-\frac{A_i}{\gcd(A_i,B_i)} / -\frac{B_i}{\gcd(A_i,B_i)}$.

ちょっとした場合分けから，傾きと仲の悪さの関係について次のことが言えます．

- 傾き $1/0$ のイワシと仲が悪いのは，傾き $0/1$ のイワシ全てのみ．逆もまた然り．
- 傾き a/b $(a, b \neq 0)$ のイワシと仲が悪いのは，傾き $-b/a$ (を a の符号で通分したもの) のイワシ全てのみ．逆もまた然り．

たとえば以下の傾きが，互いに ”仲の悪いつがい” です．

$$1/0 \longleftrightarrow 0/1$$
$$5/3 \longleftrightarrow -3/5$$
$$-2/1 \longleftrightarrow 1/2$$

したがって，”仲の悪いつがい” になる傾きのペア (高々 N 通り) 全てについて，連想配列などを使って各傾きになるイワシの個数が求まれば，その後は基礎的な数え上げの範疇です．オーバーフローには気を付けてください．

A_i, B_i の制約が大きいので，傾きを有理数の代わりに実数で表現しようとすると (long double でも) 精度が足りないおそれがあります．

F: . (Single Dot)

x 座標・y 座標それぞれを重複を除いてソートし，十分なサイズの 2 次元グリッド上に各線分を刻み込んでから BFS すれば，$\mathcal{O}(NM)$ 時間となって十分間に合います．

A: ∴ (Therefore)

It can be solved by dividing cases depending on the 1's digit, but aren't there any better ways?

Languages like C/C++, C# and Java have switch switch statements, and by using them the description in the problem statement can be expressed intuitively (sample code in C is shown below).

By the way, Ruby has case statements, which is more functional.

Listing 5 Sample Code in C

```c
#include <stdio.h>

int main(void) {
        int n;
        scanf("%d", &n);

        switch (n % 10) {
        case 2:
        case 4:
        case 5:
        case 7:
        case 9:
                puts("hon");
                break;
        case 0:
        case 1:
        case 6:
        case 8:
                puts("pon");
                break;
        case 3:
                puts("bon");
                break;
        }

        return 0;
}
```

B: ... (Triples Dots)

Check if the length is more than K letters, perform the operations described in the problem statement if needed, and print it.

PHP has a straightforward function called truncate, so that will be an option as well.

Listing 6 Sample Code in C++

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

#include <string>
using std::string;

int main(void) {
        int k;
        string s;

        cin >> k;
        cin >> s;

        if (s.size() > k) {
                s = s.substr(0, k) + "...";
        }

        cout << s << '\n';

        return 0;
}
```

Listing 7 Sample Code in Python3

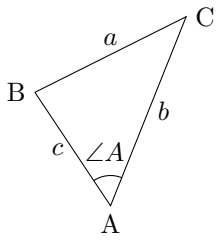
```
k = int(input())
s = input()

if len(s) > k:
        s = s[:k] + "..."

print(s)
```

C: : (Colon)

Computing the coordinates of two endpoints and then calculating the distance between them is troublesome. Let's use the following:

Laws of Cosines

For any triangle ABC, it holds that $a^2 = b^2 + c^2 - 2bc\cos\angle A$. Here, a, b and c denotes $|BC|, |CA|$ and $|AB|$, respectively.



By assigning the length of the hour and minute hands to b and c, and the angle between the hour and minute hands to $\angle A$ in the formula above, the (squared) desired length can be calculated. The angle can be obtained by subtracting "the amount of angle the minute hand moved in M minutes" from "the amount of angle the hour hand moved in $H + \frac{M}{60}$ hours" (in C++, you don't have to take mod for the cos function).

Listing 8 Sample Code in C

```
#include <stdio.h>
#include <math.h>

#define PI 3.14159265358979323846264338327950L

int main(void) {
        int a, b, h, m;
        scanf("%d%d%d%d", &a, &b, &h, &m);

        long double rad = PI * 2 * (
                (long double)h / 12.0 + ((long double)m / 60.0) / 12.0 - (long
                     double)m / 60.0);

        long double rsq = (long double)(a * a + b * b) - (long double)(2 * a *
             b) * cosl(rad);

        printf("%20.20Lf\n", sqrtl(rsq));

```

```
17          return 0;
18  }
```

D: .. (Double Dots)

We define the "depth" of room by the minimum number of passages from the room to the room 1. In particular, only the room 1 is of depth 0.

Then, the following important fact holds:

<! >< ! >< ! >< ! >< ! >< ! >

For any room of depth $d+1$, at least one room of depth d is connected to it.

The proof is easy (if there is no such room, it is intuitively unpleasant). Therefore, if each the signpost of depth $d+1$ indicates a room of depth d, then the purpose can be always achieved.

When implementing, you can compute the depths of the rooms first, and then look at each edge and update it one by one.

E: · (Bullet)

Since a sardine of $(A_i, B_i) = (0, 0)$ is on bad terms with every other individuals, so either "choose one sardine such that $(A_i, B_i) = (0, 0)$" or "never choose sardines such that $(A_i, B_i) = (0, 0)$" is possible. Of course, the number of the former ways of choosing is equal to the number of such sardines, in the further discussion the sardines such that $(A_i, B_i) = (0, 0)$ will be excluded.

Define the "slope" of a sardine by $\frac{A_i}{B_i}$ and consider representing them by irreducible fractions. Specifically, we will define them by:

- When $B_i = 0$, the slope is $1/0$.
- When $B_i > 0$, the slope is $\frac{A_i}{\gcd(A_i,B_i)} / \frac{B_i}{\gcd(A_i,B_i)}$.
- When $B_i < 0$, the slope is $-\frac{A_i}{\gcd(A_i,B_i)} / -\frac{B_i}{\gcd(A_i,B_i)}$.

By a little case division, the following dependency of the relationships on the slope can be found:

- The sardines of slope $1/0$ is on bad terms with, and only with, all the sardines of slope $0/1$, and vice versa.
- The sardines of slope a/b $(a, b \neq 0)$ is on bad terms with, and only with, all the sardines of slope $-b/a$ (normalized by the sign of a), and vice versa.

For example, the following slopes are "the pairs on bad terms with each other."

$$\begin{aligned} 1/0 &\longleftrightarrow 0/1 \\ 5/3 &\longleftrightarrow -3/5 \\ -2/1 &\longleftrightarrow 1/2 \end{aligned}$$

Therefore, for all pairs of slopes on bad terms with each other, count the number of sardines of such slopes using data structures such as associative arrays, and all the left is a basic combinatorics. Be careful of overflows.

Since the constraints of A_i, B_i is large, representing the slope with real numbers instead of rationals may not hold enough precision.

F: . (Single Dot)

Sort the x and y coordinates without duplicates, inscribe each segment into a two-dimensional grid of sufficient size and perform the BFS, then it will be in a total of $\mathcal{O}(NM)$ time, which is fast enough.