

## ABC 139 解説

drafear, E869120, evima, gazelle, square1001, wo01, yuma000

2019 年 9 月 1 日

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

### A: Tenki

$S$  と  $T$  を比較して、一致する文字がいくつあるかを数えればよいです。

この問題では文字列が固定長なので、ループを回さなくても場合分けで正答することができます。

Listing 1 C++ による実装例

---

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main() {
5      string s, t;
6      cin >> s >> t;
7      int ans = 0;
8      if(s[0] == t[0]) ans++;
9      if(s[1] == t[1]) ans++;
10     if(s[2] == t[2]) ans++;
11     cout << ans << endl;
12     return 0;
13 }
```

---

## B: Power Socket

$B$  口以上になるまで、電源タップを 1 つずつ使うシミュレーションを行うことで答えを求めることができます。すなわち、最初差込口を 1 口として、差込口が  $B$  口未満である間、電源タップ 1 つと差込口 1 口を使って差込口を  $A$  口増やすことを続けます。これを C++ 言語で実装した例を以下に示します。

---

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main() {
6     int A, B; cin >> A >> B;
7     int ans = 0;
8     int outlet = 1;
9     while (outlet < B) {
10         --outlet;
11         outlet += A;
12         ++ans;
13     }
14     cout << ans << endl;
15 }
```

---

他の解法としては、電源タップを 1 つ使うごとに差込口が  $A - 1$  口増え、最終的に初期状態から  $B - 1$  口増やしたいと考えれば、答えは  $\lceil \frac{B-1}{A-1} \rceil$  ( $B - 1$  を  $A - 1$  で割った切り上げ) になります。 $\lceil \frac{B-1}{A-1} \rceil = \lfloor \frac{B-1+A-2}{A-1} \rfloor$  なので、切り捨てで計算することもできます。

さらなる別解 (苦肉の策) として、入力約 400 通りを手計算して埋め込んでも正答することができますが、おすすめはしません。

## C: Lower

$i$  番目のマスに降り立つと  $x_i$  マス進めるとします。すると、答えは  $\max\{x_1, x_2, \dots, x_N\}$  になりますが、愚直に求めようとする時間計算量が  $O(N^2)$  となり間に合いません。この数列  $\{x_i\}$  を眺めると、 $\{3, 2, 1, 0, 2, 1, 0, 0, 5, 4, 3, 2, 1, 0\}$  のようになるため、まず  $x_1$  を愚直に求めて、次に  $x_{x_1+2}$  を求めて、... としていけば  $O(N)$  で求めることができます。数列を眺めなくとも、左端から  $x_1$  マス進めるなら、その途中のマスからスタートしてもより良くはならないため、左端からスタートし、進めるまで進み、今度は次のマスからスタートした場合を考え、... と考えることができます。

他の考え方としては、 $x_i \geq x_{i+1}$  なら  $i+1$  番目のマスに降り立つよりも  $i$  番目のマスに降り立ったほうがより良いため、そのようなマスから始めた場合を求めない、といった考え方もあります。

## D: ModSum

$A$  を  $B$  で割った余りを  $A \bmod B$  と表すことにすると

$$(1 \bmod 2) + (2 \bmod 3) + \dots + ((N-1) \bmod N) + (N \bmod 1) = 1 + 2 + \dots + N - 1 = \frac{N(N-1)}{2}$$

と順列を選ぶのが最適です。以下、これを説明します。

選んだ順列には  $1, 2, \dots, N$  がそれぞれちょうど 1 回ずつ登場します。 $i = 1, 2, \dots, N$  について、 $i$  が順列の  $x_i$  番目に登場するとします。すると、目的関数 (最大化したいもの) は

$$(x_1 \bmod 1) + (x_2 \bmod 2) + \dots + (x_N \bmod N)$$

になります。各項に着目すると、それぞれの項は最大でも  $0, 1, 2, \dots, N-1$  ですが、 $i \geq 2$  について  $x_i = i-1$  とし、余った  $N$  を  $x_1 = N$  とすると実際に

$$(N \bmod 1) + (1 \bmod 2) + \dots + ((N-1) \bmod N) = 0 + 1 + 2 + \dots + N-1$$

とできるため、目的関数の最大値は  $0 + 1 + 2 + \dots + N-1 = \frac{N(N-1)}{2}$  です。

## E: League

(原案: wo01, 準備・解説: evima)

二つの条件は次のようにまとめられます。

- 各  $i, j$  に対し、選手  $i$  対 選手  $A_{i,j+1}$  の試合は選手  $i$  対 選手  $A_{i,j}$  の試合を行った日の翌日以降にしか行えない。

すなわち、選手自体にさほど重要性はなく、単に何個かの試合のペアに対してどちらを先に行うべきかが定められていると考えられます。

以下、 $N(N-1)/2$  試合のそれぞれを頂点に見立て、試合  $y$  が試合  $x$  を行った翌日にしか行えないときに辺  $x \rightarrow y$  が存在するような有向グラフを考えます。このグラフに閉路が存在する場合は条件を満たせません。閉路が存在しない場合は、毎日、入次数が 0 であるような頂点 (辺が“刺さって”いない頂点) に対応する試合をすべて行ってそれらの頂点とそれらから出る辺をすべて削除すれば、最小の日数で全試合を行えます。(入次数が 0 であるような頂点に対応する試合を翌日以降に繰り越す意義がないため。) この日数はグラフに存在する最長のパスに含まれる頂点の数に等しくなります。

以上から、閉路検出と最大パス長の算出を深さ優先探索により行うことで、頂点と辺の総数に対して線形時間、すなわち  $O(N^2)$  時間でこの問題を解くことができます。

## F: Engines

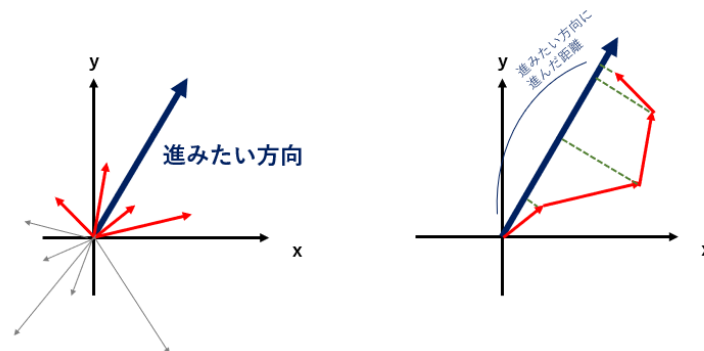
(原案: square1001 + E869120, 準備・解説: square1001 + E869120)

### ★ 謝辞

高校数学を理解していない人には少し分かりにくい解説となっておりますが、最大限分かりやすく説明した限りですので、ご容赦ください。

### ★ 全体的な方針

さて、最終的にベクトル  $(A, B)$  の方向にできるだけ進むことを考えます。この場合、座標  $(X, Y)$  から座標  $(X + x_i, Y + y_i)$  に移動するエンジンについて、「ベクトル  $(x_i, y_i)$  の方向に進むエンジン」とします。その時、ベクトル  $(A, B)$  とベクトル  $(x_i, y_i)$  のなす角が 90 度未満であるエンジンだけを使えば、最大限  $(A, B)$  の方向に進むことができます。



例えば上の図において、「進みたい方向に進んだ距離」を最大化するためには、赤のベクトルを持つエンジンだけを選べばよいです。もう少し分かりやすい例を出しましょう。

- $X$  軸の正の方向に進みたい場合（ベクトル  $(1, 0)$  の方向に進みたい場合）、 $X$  方向に進む距離が正であるエンジン（つまり、 $x_i > 0$  であるエンジン）だけを選べばよいです。
- $Y$  軸の負の方向に進みたい場合は、 $Y$  方向に進む距離が負であるエンジン（つまり、 $y_i < 0$  であるエンジン）だけを選べばよいです。
- $X$  軸の正方向から時計回りに 45 度の方向に進みたい場合（つまりベクトル  $(1, 1)$  の方向に進みたい場合）、 $x_i + y_i \geq 0$  であるエンジンだけを選べばよいです。

これらは全て、**進みたい方向とエンジンの進む方向の成す角が 90 度以内であるエンジンを選ぶ**という例です。

★ まず、どうやって成す角が 90 度未満かを判定するか

$(ax, ay)$  の方向 (ベクトル) と、 $(bx, by)$  の方向 (ベクトル) のなす角が 90 度未満である条件は、 $axbx + ayby > 0$ 、つまりベクトルの内積が正であるという非常に簡単なものです。競プロだけでなく、高校数学とかでも良く使われるので、覚えておきましょう。

そこで、いくつか方針を考えてみることを考えます。

★ 方針 A: 進む方向を全探索する (TLE 解法)

まず、最も簡単な解法として、進む方向 (あるいはベクトル) を全探索することを考えます。しかし、制約は以下の通りとなっています。

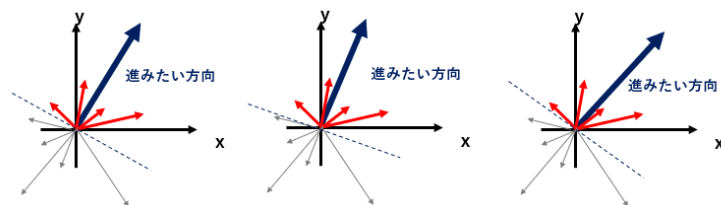
1.  $N \leq 100$
2.  $-1\,000\,000 \leq x_i, y_i \leq 1\,000\,000$

ですので、最大で座標  $(100\,000\,000, 100\,000\,000)$  まで進む可能性があります。進むベクトルの通り数はおおよそ  $4 \times 10^{16}$  通りになるため、TLE します。

★ 方針 B: 進む方向を絞る

一個重要な考察があります。これは、「必ず進む方向と最終的な位置が一致しなくても、選ぶエンジンが最適であれば問題がない」という考え方です。

例えば、以下の図において、3 つの進む方向 (青の矢印) について、全て同じエンジンの集合を選ぶことになります。



ですので、進む方向を絞る事はできないのでしょうか。例えば、進む方向を  $1 \leq i \leq N$  に対して、 $(x_i, y_i)$ 、 $(-y_i, x_i)$ 、 $(-x_i, -y_i)$ 、 $(y_i, -x_i)$  だけに絞ると、高々  $O(N)$  通りだけで済みます。各進む方向について  $N$  回の計算が必要ですので、 $O(N^2)$  が達成できます。普通に上の 4 パターンだけを追加すると WA するので、境界を上手く場合分けすることが必要です。次の方針は、もう少し実装や境界判定が簡単なものです。

★ 方針 C: 「角度でソート」を利用する

ここでは、エンジン  $i$  が  $x$  軸の正の方向から見て時計回りに何度回ったかを示す値を  $r_i$  とします。例えば、 $(1, 2)$  の方向に進むエンジンの場合、 $r_i$  の値はおおよそ 64 度くらいになります。

さて、この解説の 2 つの図に注目していただきたいのですが、全て  $L \leq r_i \leq R$  を満たすエンジンのみ選ばれています。(ただし、360 度と 0 度の境界を通り過ぎる場合、 $L \leq r_i$  または  $r_i \leq L$  となります。)

つまり、エンジンの進む方向の角度でエンジンをソートすることにしましょう。(豆知識ですが、角度でソートすることを「偏角ソート」と言い、 $O(N \log N)$  でできます。) そうすると、選ぶエンジンの集合は一つの区間になります。

最後に選ぶ区間を全探索すると、 $O(N^3)$  或いは  $O(N^2)$  でこの問題を解くことが出来ます。上手く尺取り法を使うと、 $O(N \log N)$  で解くことができますが、境界判定や実装が大変です。

★ サンプルコード (C++)

<https://atcoder.jp/contests/abc139/submissions/7244437>