ABC 139 解説

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

2019 年 9 月 1 日

For International Readers: English editorial starts on page 9.

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

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

int main() {
  string s, t;
  cin >> s >> t;
  int ans = 0;
  if(s[0] == t[0]) ans++;
  if(s[1] == t[1]) ans++;
  if(s[2] == t[2]) ans++;
  cout << ans << endl;
  return 0;
}
```

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。

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
  int A, B; cin >> A >> B;
  int ans = 0;
  int outlet = 1;
  while (outlet < B) {
    --outlet;
    outlet += A;
    ++ans;
  }
  cout << ans << endl;
}
```

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、入力の約 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 $\max\{x_1, x_2, ..., x_N\}$ になりますが、愚直に求めようとすると時間計算量が $O(N^2)$ 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

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

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 $1, 2, ..., N$ がそれぞれちょうど 1 回ずつ登場します。$i = 1, 2, ..., N$ 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

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

になります。各項に着目すると、それぞれの項は最大でも $0, 1, 2, ..., N-1$ 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

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

とできるため、目的関数の最大値は $0 + 1 + 2 + ... + N - 1 = \frac{N(N-1)}{2}$ 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 を行った翌日以降にしか行えないときに辺 $x \to y$ 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 $\mathrm{O}(N^2)$ 時間でこの問題を解くことができます。

F: Engines

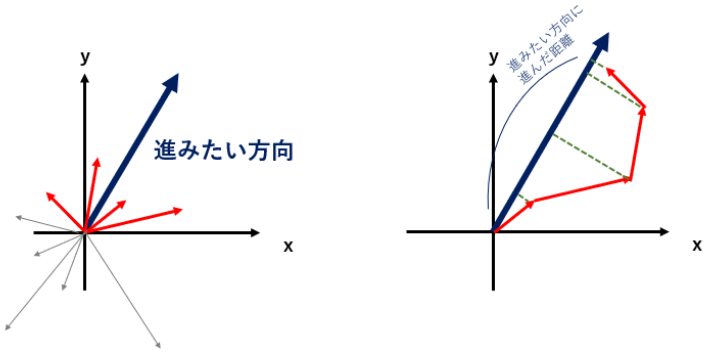
(原案: square1001 + E869120, 準備・解説: square1001 + E869120)

★ 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 軸の正方向から反時計回りに 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。

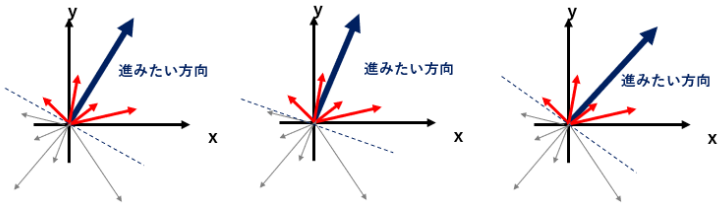
1. $N \leq 100$
2. $-1\ 000\ 000 \leq x_i, y_i \leq 1\ 000\ 000$

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 4×10^{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反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

ABC 139 Editorial

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

September 1, 2019

A: Tenki

You can compare S and T and count the number of corresponding characters.

In this problem the length of string is fixed, so you can get AC by splitting case, without using loop.

Listing 1 Sample Implementation in C++

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

int main() {
  string s, t;
  cin >> s >> t;
  int ans = 0;
  if(s[0] == t[0]) ans++;
  if(s[1] == t[1]) ans++;
  if(s[2] == t[2]) ans++;
  cout << ans << endl;
  return 0;
}
```

B: Power Socket

You can find the answer by the simulation of using power strips one by one, until the number of sockets becomes more than or equal to B. Namely, first assume that there are 1 socket, and while there are less than B empty sockets, repeat using one power strip and one empty socket so as to increase the number of empty socket by A. The following is an implementation example in C++.

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
  int A, B; cin >> A >> B;
  int ans = 0;
  int outlet = 1;
  while (outlet < B) {
    --outlet;
    outlet += A;
    ++ans;
  }
  cout << ans << endl;
}
```

Another solution is like this: every time you use one power strip, the number of empty socket increases by $A-1$, and you ultimately want to increase the number of power socket by $B-1$ compared to the beginning, so the answer will be $\lceil \frac{B-1}{A-1} \rceil$ ($B-1$ divided by $A-1$, rounded up). Since $\lceil \frac{B-1}{A-1} \rceil = \lfloor \frac{B-1+A-2}{A-1} \rfloor$ holds, so it can be calculated rounded-down.

As one more solution (or last resort), you can get AC by calculating all the 400 patterns by hand and embed into the source code, but it's not recommended.

C: Lower

Assume that if you land on i-th square you can move x_i times. Then the anwser will be $\max\{x_1, x_2, ..., x_N\}$, but if you try to calculate naively, it will need a total of $O(N^2)$ time and it won't finish in time. If you look at the sequence $\{x_i\}$, it appears that it will be like $\{3, 2, 1, 0, 2, 1, 0, 0, 5, 4, 3, 2, 1, 0\}$, so you can compute x_1 naively, then compute x_{x_1+2} naively, ... and so on, you can find the answer in a total of $O(N)$ time. Or otherwise, without looking at the sequence, you can think like the following: if you can move x_1 times from the leftmost squares, you can't move more times if you start halfway, so think of starting from the next square, .. and so on.

Another idea is that, if $x_i \geq x_{i+1}$, then it is better to land on i-th square instead of $i + 1$-th square, so avoid computing the case of starting from such squares.

D: ModSum

Let A mod B be the remainder of A devided by B, then it is optimal to choose the permutation such that

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

The following is the explanation.

In the chosen permutation, each $1, 2, ..., N$ appears exactly once. For $i = 1, 2, ..., N$, assume that i is x_i-the element of the permutation. Then, the objective function (the one we want to maximize) is

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

If you focus on each term, it appears that maximum of each term is $0, 1, 2, ..., N-1$. Let $x_i = i-1$ for $i \geq 2$, and for the remaining N, let $x_1 = N$, then

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

actually holds, so the maximum value of the objective function is $0+1+2+...+N-1 = \frac{N(N-1)}{2}$.

E: League

(Original writer: wo01, preparation and editorial: evima)

The two conditions can be simplified as follows:

- For each i, j, the match of Player i vs Player $A_{i,j+1}$ can only be scheduled one or more days after the day of the match of Player A_i and Player $A_{i,j}$.

Namely, the players themselves are not so important, and we can assume that the only constraint is the order of schedule between some pairs of matches.

Let's assume a directed graph consisting of $N(N-1)/2$ vectors, each of which corresponds to each match, such that if match y can only be scheduled one or more days after the match x, there exists an edge $x \to y$. If this graph contains any cylces, it is impossible to meet the conditions. If there are no cycles in it, you can schedule all the matches in minimum days in the following way: for each day, do all the matches whose corresponding vectors' indegree is0 (the vectors without any incoming edges), and remove the vectors and all the outgoing edges from them. (This is because it is meaningless to postpone the matches whose corresponding vectors' indegree is 0.) The minimum number of days is equal to the number of vectors in the longest path in the graph.

Therefore, if you scan cycles and calculate the length of longest path with DFS, you can solve the problem in a total of linear time of the number of vectors and edges, that is, in a total of $\mathrm{O}(N^2)$ time.

F: Engines

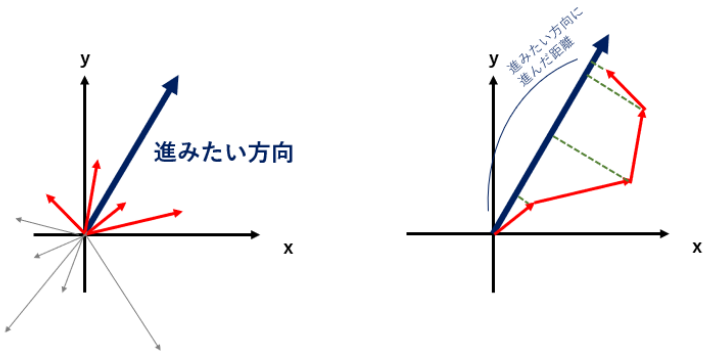
(Original writer: square1001 + E869120, preparation and editorial: square1001 + E869120)

★ Acknowledgment

The editorial may be a little bit difficult because it is explained in mathematical ways, but we tried to explain as easy as possible, so please forgive us.

★ Main idea

Let's try to move towards the direction of vector (A, B) as much as possible. In this case, we call the engine that enables him to move from coordinates (X, Y) to $(X + x_i, Y + y_i)$ "the engine that moves in the direction of vector (x_i, y_i)." Then, if he use the engine such that the angle between vector (A, B) and vector (x_i, y_i) is no more than 90 degrees, he can move towards the direction of (A, B) as much as possible.



For example, in order to maximize the "progressed distance towards the desired direction" ("the desired direction" is drawn with blue arrow in the diagram above), it is sufficient if you only use the red vectors. The following example may be easier to understand:

- If he wants to move towards the positive X direction (towards the direction of vector $(1, 0)$), it is sufficient to choose such engines that enables him to move positively in the X direction (that is, engines such that $x_i > 0$).
- If he wants to move towards the negative Y direction, it is sufficient to choose such engines that enables him to move negatively in the Y direction (that is, engines such that $y_i < 0$).
- If he wants to move towards the direction of X axis rotated by 45 degrees counterclockwise (towards the direction of vector $(1, 1)$), it is sufficient to choose such engines that $x_i + y_i \geq 0$.

All of them are the examples of **choosing the engines such that the angle between its progression direction and the desired direction is no more than 90 degrees**.

★ First, how to judge if the angle between two vectors are less than 90 degrees?

The condition of the angle between the direction (vector) (ax, ay) and the direction (vector) (bx, by) is less than 90 degrees is that $axbx + ayby > 0$, or that inner product of two vectors is positive, which is pretty easy. It is commonly used not only in competitive programming but also other mathematical problems, so it will be good for you to remember.

Let's think about several solutions.

★ Solution A: Brute force all the "desired directions" (TLE solution)

First, the most naive solution is to brute force all the desired directions (or vectors). However, the constraints are the following:

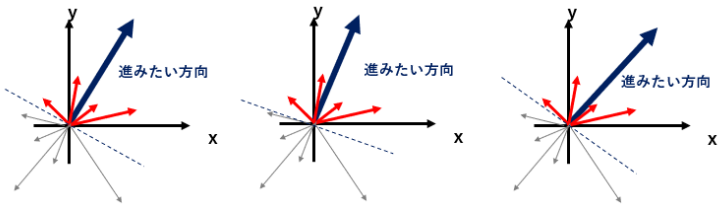
1. $N \leq 100$
2. $-1\ 000\ 000 \leq x_i, y_i \leq 1\ 000\ 000$

It means that there is a possibility of ending up with coordinates $(100\ 000\ 000, 100\ 000\ 000)$ at most. There are about 4×10^{16} possible desired vectors, so it will lead to TLE.

★ Solution B: Limit the number of desired directions

One important point is that "even if the desired direction and the ultimate position does not coincide, it's no problem if the selected engines are optimal."

For example, in the diagram below, no matter which of those three desired directions (the blue arrows) you choose, the selected set of engines are all the same.



Then isn't it possible to limit the number of desired directions? For example, if you limit the desired direction to (x_i, y_i), $(-y_i, x_i)$, $(-x_i, -y_i)$ and $(y_i, -x_i)$ for $1 \leq i \leq N$, the number of desired directions are at most $O(N)$. If you only add those 4 patterns it will end up with WA, so you have to split border cases properly. If you choose the solution described below, the implementation and border judgement will be a little bit easier.

★ Solution C: Use "sort by angles"

Let i be the angle from x-axis to the direction of engine i, rotated counterclockwise. For instance, the r_i value for the engine that enables him to move in the direction of $(1, 2)$ is about 64 degrees.

Pay attention to the two diagrams in the editorial, then you see that only the engines such that $L \leq r_i \leq R$ are selected. (However, if it passes the border of 0 degrees and 360 degrees, it will hold that $L \leq r_i$ or $r_i \leq L$.)

Therefore, let's sort the engines by the angles of directions of them. (Tip: this sort can be achieved in $O(N \log N)$.) Then, the set of engines will be a segment of the sorted sequence.

Finally, if you brute force the segment to choose, you can solve it in a total of $O(N^3)$ or $O(N^2)$ time. If you use sliding window properly, it can be solved in $O(N \log N)$, but border judge and implementation is fairly hard.

★ Sample Code (C++)

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