# ABC 135 解説

drafear, evima, gazelle, sheyasutaka, sigma425

2019 年 7 月 27 日

*For International Readers: English editorial starts on page 7.*

## A: Harmony

$A$ と $B$ の偶奇が異なるとき、どのような整数 $K$ についても $|A-K|$ と $|B-K|$ の偶奇は異なります。したがってこの場合、答えは IMPOSSIBLE です。

次に $A$ と $B$ の偶奇が等しいときを考えます。このとき $K = \frac{A+B}{2}$ とおくと $K$ は整数です。また $|A-K| = |\frac{A-B}{2}|$, $|B-K| = |\frac{B-A}{2}|$ となり、絶対値の定義より両者は等しいことが言えます。よってこれが答えです。

C++ でのコード例を示します。

Listing 1　C++ による実装例

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  long long A, B;
  cin >> A >> B;
  if(A % 2 != B % 2) {
    cout << "IMPOSSIBLE" << endl;
  } else {
    cout << (A + B) / 2 << endl;
  }
  return 0;
}
```

## B: 0 or 1 Swap

$p_i \neq i$ であるような要素 $p_i$ の数を $k$ とします。結論から言うと $k \leq 2$ であるときに限り 1 回以下のスワップで $p$ を昇順にすることが可能です。$k = 0$ のとき、最初から $p$ が昇順であることを意味します。$k = 1$ になることはありません。$k = 2$ のとき、$p_i \neq i$ である 2 つの要素を 1 回スワップすることで $p$ を昇順にできます。1 回のスワップで $k$ の値はたかだか 2 しか減らすことができないため、$k \geq 3$ のとき 1 回以下のスワップで $p$ を昇順にすることはできません。

与えられた順列の $k$ の値は線形時間で求めることができます。よってこの問題を $O(N)$ で解くことができました。ちなみに問題をより一般化した「$M$ $(M \leq N)$ 回以下のスワップで $p$ を昇順にできるか？」という問題も $O(N)$ で解くことができます。

Listing 2　C++ による実装例

```cpp
#include<bits/stdc++.h>
using namespace std;
#define rep(i, n) for(int i = 0; i < (int)n; i++)

int main() {
    int n;
    cin >> n;
    vector<int> p(n);
    rep(i, n) cin >> p[i];
    int k = 0;
    rep(i, n) if(p[i] != i + 1) k++;
    if(k <= 2) cout << "YES" << endl;
    else cout << "NO" << endl;
    return 0;
}
```

## C: City Savers

1 番目の街は 1 番目の勇者に救ってもらうしかありません。よって、1 番目の勇者は 1 番目の街のモンスターを全力で $\min(A_1, B_1)$ 体倒し、残った力で 2 番目の街のモンスターを全力で $\min(A_2, B_1 - \min(A_1, B_1))$ 体倒すのが最適です。すると、2 番目の街に残ったモンスターは 2 番目の勇者でしか倒すことができません。これを繰り返すことで答えを求めることができます。

正確に述べると、$f_{ij}$ を勇者 $i$ が街 $j$ のモンスターを倒す数とします。すると、合計で

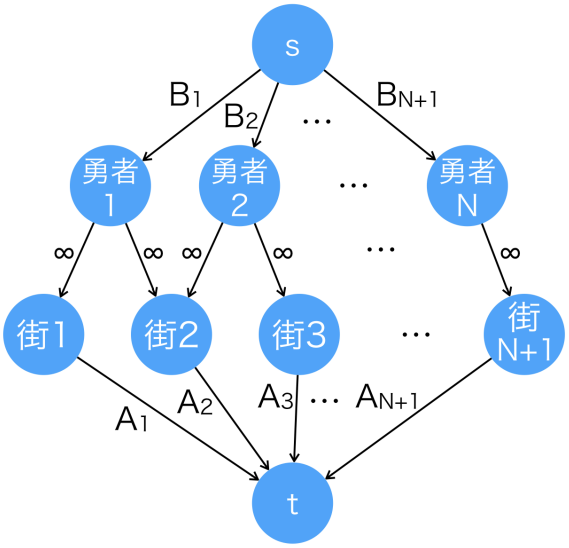
$$\sum_{i=1}^{N}(f_{ii} + f_{i(i+1)})$$

体のモンスターを倒すことになります。これを最大化する方法として上記の方法があり、この方法に従って計算すると、

$$\begin{aligned} f_{11} &= \min(A_1, B_1) \\ f_{i(i+1)} &= \min(A_{i+1}, B_i - f_{ii}) \quad (1 \leq i \leq N) \\ f_{ii} &= \min(A_i - f_{(i-1)i}, B_i) \quad (2 \leq i \leq N) \end{aligned}$$

などによって時間計算量 $O(N)$ で計算できます。

正当性、すなわちこれが最大になることの証明は次のとおりです。まず、この問題は次のような最大流問題に帰着させることができます (s が source、t が sink、辺の重みが容量です)。




勇者 $i$ の頂点から街 $j$ の頂点に流れる流量を上で計算できる $f_{ij}$ とすると、確かに条件を満たしています。さらに、残余グラフにおいて増大道が存在するとすれば作り方に矛盾するため、増大道は存在しません。よって、これが最大です。

ところで、この問題を最大流問題に帰着させて解くことができそうに見えますが、Dinic のアルゴリズムなどでは流す優先順位をうまく設定しないと最悪時間計算量が $O(N^2)$ となり、間に合いません。

## D: Digits Parade

次の漸化式を立てます.

$$dp[i][j] := \text{先頭 } i \text{ 文字として考えられるもののうち, 13 で割ったあまりが } j \text{ であるものの数}$$

このとき $i$ の昇順に dp テーブルを見ると, $i-1$ 文字目までを 13 で割ったあまり ($dp[i-1][j]$ の $j$) と $s[i]$ としてあり得る数字を全て試すことで $dp[i][0] \sim dp[i][12]$ の値がわかります.

以下に解答例を示します.

Listing 3　C++ での実装例

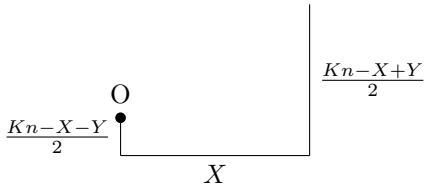
```
static const ull MOD = 1000000007LL;

int n;
char s[100005];
uint64_t dp[100005][13];

uint64_t solve () {
  int i, j, ki;
  uint64_t res = 0;

  scanf("%s", s);
  n = strlen(s);

  dp[0][0] = 1;
  for (i = 0; i < n; i++) {
    int c;
    if (s[i] == '?') c = -1;
    else c = s[i] - '0';

    for (j = 0; j < 10; j++) {
      if (c != -1 && c != j) continue;
      for (ki = 0; ki < 13; ki++) {
        dp[i + 1][(ki * 10 + j) % 13] += dp[i][ki];
      }
    }
    for (j = 0; j < 13; j++) dp[i + 1][j] %= MOD;
  }
  res = dp[n][5];

  printf("%llu\n", res);
  return 0;
}
```

## E: Golf

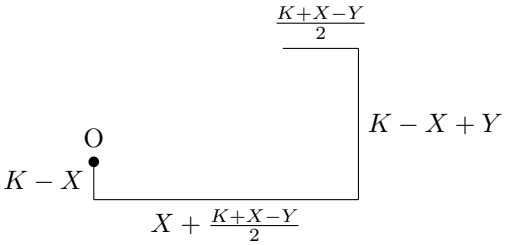
話を簡単にするため，$X \geq Y \geq 0$ としてよいです (適当に変換することで帰着できます)．$n$ をスコアの最小値とします．

$K$ が偶数，$X+Y$ が奇数のときは不可能です．逆に，それ以外のケースでは常に，以下のどちらかの形式で構築できます．

タイプ A:




タイプ B (常に $n=3$):




$c := (K - (X+Y \bmod K)) \bmod K$ とします．$X+Y<K$ かつ $X+Y$ が奇数のときタイプ B が，それ以外のときは次のように $n$ の値を決めることでタイプ A が適用できます．

- $X+Y<K$ かつ $X+Y$ が偶数のとき，$n=2$.
- $X+Y \geq K$ かつ $c$ が偶数のとき，$n = \lceil \frac{X+Y}{K} \rceil$.
- $X+Y \geq K$ かつ $c$ が奇数のとき，$n = \lceil \frac{X+Y}{K} \rceil + 1$.

これらの構築方法が正しいことの証明は容易です．

## F: Strings of Eternity

問題文の注記では文字列の先頭の文字を 1 文字目としていますが、ここでは 0 文字目とします。問題の内容を直感的に述べると、$s$ を無限個連結して得られる文字列 $s'$ に $t$ は最大で何連続で現れるか、となります。

ここで、0 以上 $|s|-1$ 以下のすべての整数 $i$ に対して、$s'$ の $i$ 文字目から $i+|t|-1$ 文字目までの部分文字列が $t$ と一致するか否か (この真偽値を $\mathrm{match}_i$ とします) が判明しているとします。すると、$s'$ の周期性を利用してシミュレーションを行うことで $\mathrm{O}(|s|)$ 時間で答えを求めることができます。グラフの言葉で表現すると、次のような有向グラフの最長パスの長さが答えとなります。

- $0, 1, \ldots, |s|-1$ の番号がついた $|s|$ 個の頂点を持つ。
- $\mathrm{match}_i = \mathrm{true}$ のとき有向辺 $i \to (i+|t|) \bmod |s|$ が存在する。その他に辺はない。

残るは $\mathrm{match}_0, \ldots, \mathrm{match}_{|s|-1}$ を時間内にすべて求めることです。もちろん、ナイーブに $s'$ と $t$ を比較すると最悪の場合に $\mathrm{O}((|s|+|t|)\times|t|)$ 時間かかってしまい間に合いません。[*1] しかしこれは古典的な問題であり、この問題を高速に解くためのアルゴリズムが多く考案されています。線形時間で $\mathrm{match}_0, \ldots, \mathrm{match}_{|s|-1}$ をすべて求めるアルゴリズムを二つ挙げます。

- KMP 法 (Wikipedia の記事へのリンク)
- Z algorithm (GeeksforGeeks の記事へのリンク)

この他に文字列を非十進表記の数値とみなすローリングハッシュを用いる方法もありますが、剰余の計算を省くために数値を $2^{64}$ で割った余りを用いると衝突するような入力を用意してありますのでご注意ください。

[*1] 入力がランダムな文字列であればナイーブに比較しても大抵の場合すぐにマッチに失敗するため計算量の期待値は線形ですが、$s = \texttt{aaa...aa}, t = \texttt{aaa...ab}$ のような繰り返しの多い文字列ではなかなかマッチに失敗しません

# ABC 136 Editorial

drafear, evima, gazelle, sheyasutaka, sigma425

July 27, 2019

## A: Harmony

If the parities of $A$ and $B$ are differemt, For any integer $K$, the parities of $|A-K|$ and $|B-K|$ are different. In such cases, the answer is IMPOSSIBLE.

Next, let's think about the cases when the parities of $A$ and $B$ are the same. In such cases, $K = \frac{A+B}{2}$, then $K$ is an integer. Also, $|A-K| = |\frac{A-B}{2}|$ and $|B-K| = |\frac{B-A}{2}|$ holds, so these are the equals, according to the definition of absolute value. Therefore, this is the answer.

The following is an example code in C++.

Listing 1 An implementation example in C++

```
#include <bits/stdc++.h>
using namespace std;
int main() {
  long long A, B;
  cin >> A >> B;
  if(A % 2 != B % 2) {
    cout << "IMPOSSIBLE" << endl;
  } else {
    cout << (A + B) / 2 << endl;
  }
  return 0;
}
```

## B: 0 or 1 Swap

Let $k$ be the number of elements $p_i$ such that $p_i \neq i$. To come to the point, you can sort $p$ in ascending order by performing at most 1 operation of swap only if $k \leq 2$. If $k = 0$, it means that $p$ is already sorted in ascending order. It is impossible that $k = 1$. If $k = 2$, you can swap the two elements such that $p_i \neq i$ only one time so that $p$ is sorted in ascending order. In one swap, the value of $K$ can be decreased by at most 2, so if $k \leq 3$, you cannot sort $p$ in ascending order by performing at most one swap.

The value of $k$ can be obtained in linear time. Here the problem was solved in $O(N)$. By the way, the problem that "can $p$ be sorted in ascending order by performing at most $M$ ($M \leq N$) swaps?" can also be solved in $O(N)$ time.

Listing 2 An implementation example in C++

```cpp
#include<bits/stdc++.h>
using namespace std;
#define rep(i, n) for(int i = 0; i < (int)n; i++)

int main() {
    int n;
    cin >> n;
    vector<int> p(n);
    rep(i, n) cin >> p[i];
    int k = 0;
    rep(i, n) if(p[i] != i + 1) k++;
    if(k <= 2) cout << "YES" << endl;
    else cout << "NO" << endl;
    return 0;
}
```

## C: City Savers

Only the first hero can save the first town. So, it is optimal that the first hero do the best to defeat $\min(A_1, B_1)$ monsters in the first town, and then do the best to defeat $\min(A_2, B_1 - \min(A_1, B_1))$ monsters in the second town. Then, the monsters remaining in the second town can only be defeated by the second hero. You can find the answer repeating this procedure.

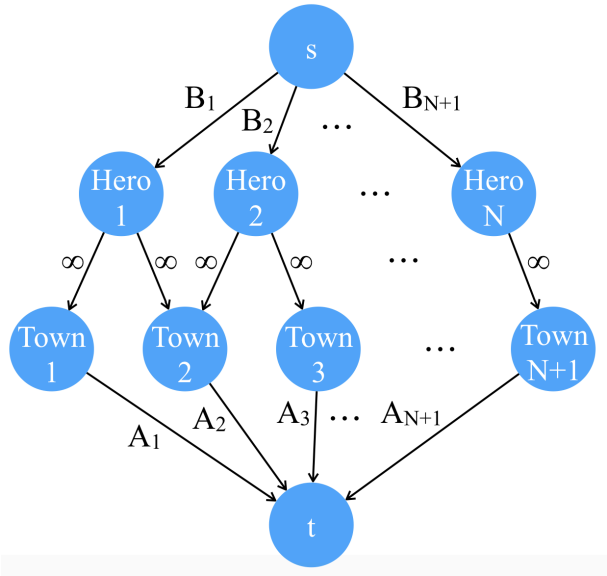
To be precise, let $f_{ij}$ be the number of the monsters in the town $j$ that are defeated by the hero $i$. Then, in total

$$\sum_{i=1}^{N}(f_{ii} + f_{i(i+1)})$$

monsters would be defeated. One of the way to maximize this is the procedures mentioned above, and with the procedures, it can be calculated in time complexity $O(N)$ with the steps like:

$$\begin{aligned} f_{11} &= \min(A_1, B_1) \\ f_{i(i+1)} &= \min(A_{i+1}, B_i - f_{ii}) \quad (1 \leq i \leq N) \\ f_{ii} &= \min(A_i - f_{(i-1)i}, B_i) \quad (2 \leq i \leq N). \end{aligned}$$

The validity that the sum mentioned above is the maximum can be proved like this: First, this problem can be attributed to the following maximum flow problem (here, s denotes source, t denotes sink and the weight of edges are the capacities).




If the flow from the vertex of hero $i$ to the vertex of town $j$ is $f_{ij}$, it meets the conditions. Moreover, if there exists a flow augmenting path in the residual network, it contradicts to the way of construction, for there doesn't exist a flow augmenting path. Therefore, this is the maximum.

By the way, it seems that this problem can be solved by attributing to a maximum flow problem, but

if you use Dinic's algorithm etc., the worst time complexity will be $O(N^2)$ and it will not finish in time, if you don't configure a proper priorities.

## D: Digits Parade

You can build the following relation:

$$dp[i][j] := \text{The number of all the possible first } i \text{ digits such that remainder of 13 is } j$$

If you see the dp table in an increasing order of $i$, by trying all the possible reaminder by 13 of the first $i-1$ digits ($j$ of $dp[i-1][j]$) and $s[i]$, you can determine the values of $dp[i][0] \sim dp[i][12]$.

The following is a sample answer.

Listing 3 An implementation example in C++

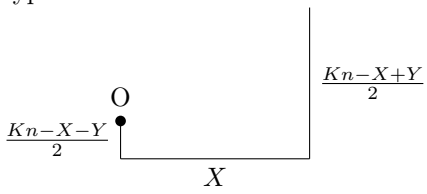
```
static const ull MOD = 1000000007LL;

int n;
char s[100005];
uint64_t dp[100005][13];

uint64_t solve () {
  int i, j, ki;
  uint64_t res = 0;

  scanf("%s", s);
  n = strlen(s);

  dp[0][0] = 1;
  for (i = 0; i < n; i++) {
    int c;
    if (s[i] == '?') c = -1;
    else c = s[i] - '0';

    for (j = 0; j < 10; j++) {
      if (c != -1 && c != j) continue;
      for (ki = 0; ki < 13; ki++) {
        dp[i + 1][(ki * 10 + j) % 13] += dp[i][ki];
      }
    }
    for (j = 0; j < 13; j++) dp[i + 1][j] %= MOD;
  }
  res = dp[n][5];

  printf("%llu\n", res);
  return 0;
}
```

## E: Golf

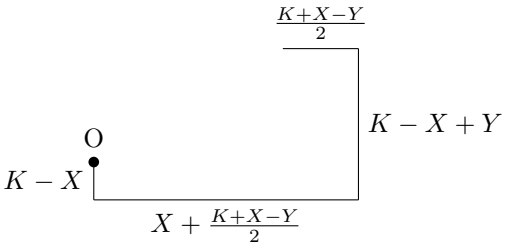
For simplicity, you can assume that $X \geq Y \geq 0$ (you can apply some appropriate transformations to achieve this).

If $K$ is even and $X + Y$ is odd, it's impossible. Otherwise, you can construct the answer by either of the following two ways:

Type A:




Type B (alyaws $n = 3$):




Let $c := (K - (X + Y \bmod K)) \bmod K$. If $X + Y < K$ and $X + Y$ is odd, type B can be applied; otherwise type A can be applied by determining $n$ with the following steps:

- If $X + Y < K$ and $X + Y$ is even, $n = 2$.
- If $X + Y \geq K$ and $c$ is even, $n = \lceil \frac{X+Y}{K} \rceil$.
- If $X + Y \geq K$ and $c$ is odd, $n = \lceil \frac{X+Y}{K} \rceil + 1$.

It is easy to prove that the construction procedures are rightful.

## F: Strings of Eternity

In the notes in the problem statement, the first letter of a string is regarded as the first letter, but here let it 0-th letter. An intuitive restatement of the problem statement is: what is the maximum number of consecutive appearances of $t$ in $s'$, where $s'$ is an infinitely repetition of the string $s$?

Here, for each integer $i$ from 0 through $|s|-1$, suppose that it's already known whether the substring of $s'$ from $i$-th letter through $i+|t|-1$-th letter is equal to $t$ (let this truth values be $\mathrm{match}_i$). Then you can find the answer in $\mathrm{O}(|s|)$ by simulation, because $s'$ is periodic. In graph-theory terms, the answer is the longest path of the following directed graph such that:

- the graph has $|s|$ vertices, each of which is indexed $0, 1, \ldots, |s|-1$.
- the graph has a directed edge $i \to (i+|t|) \bmod |s|$ if $\mathrm{match}_i = \mathrm{true}$. There are no edges otherwise.

The remaining task is to find all the $\mathrm{match}_0, \ldots, \mathrm{match}_{|s|-1}$ in time. Of course, if you compare $s'$ and $t$ naively, it will need a time of $\mathrm{O}((|s|+|t|) \times |t|)$ at worst, so it won't finish in time. [*1] However, this is a very classic problem and many algorithms are invented to solve it fast. The following two algorithms are some of the ways to find all the $\mathrm{match}_0, \ldots, \mathrm{match}_{|s|-1}$ in a linear time.

- KNP algorithm (a link to Wikipedia article)
- Z algorithm (A link to GeeksforGeeks article)

Otherwise you can also use rolling hash algorithms by regarding the string as non-decimal numbers, but we prepared an input case such that collision would happen if you use mod $2^{64}$ in order to avoid the modulo operations.

[*1] If the input is random, the match would fail in most case so the expected time complexity is linear, but if the string has many repetition, such as $s = \texttt{aaa...aa}, t = \texttt{aaa...ab}$, it will take long time until the match fails