TỔ CHỨC THE GIFTED BATTLEFIELD
DỰ ÁN CHICKEN MINDS

KỲ THI THỬ TUYỂN SINH LỚP 10 THPT CHUYÊN
NĂM HỌC 2025 – 2026

HƯỚNG DẪN CHẤM THI

Môn thi chuyên: **TIN HỌC**
Thời gian làm bài: **150 phút** (không kể thời gian phát đề)
Ngày làm bài thi: **23/4/2025 – 05/5/2025**
Hướng dẫn chấm thi gồm 06 trang, 03 bài

Tổng quan đề thi

	Tên bài	File chương trình	File dữ liệu	File kết quả	Hạn chế thời gian	Hạn chế bộ nhớ
Bài 1	Chiếc giày thất lạc	`FINDSHOE.*`	`FINDSHOE.INP`	`FINDSHOE.OUT`	1 giây	1024 MB
Bài 2	Món quà cuối cấp	`FOLDCORD.*`	`FOLDCORD.INP`	`FOLDCORD.OUT`	1 giây	1024 MB
Bài 3	Hoán đổi chữ số	`DGSWAP.*`	`DGSWAP.INP`	`DGSWAP.OUT`	1 giây	1024 MB

Bài 1: Chiếc giày thất lạc (4,0 điểm)

Subtask 1: Vét cạn

Cách tìm ra phần tử lẻ đơn giản nhất là sử dụng hai vòng lặp lồng nhau:

- Vòng lặp ngoài duyệt i từ 1 đến n.
- Vòng lặp trong duyệt j từ 1 đến n, đếm số cặp sao cho $a[i] = a[j]$.

Duyệt cho đến khi số cặp $a[i] = a[j]$ là số lẻ. Khi đó, ta biết phần tử $a[i]$ xuất hiện lẻ lần.

- Độ phức tạp thời gian: $\mathcal{O}(n^2)$

Subtask 2:

Cách 1: Sắp xếp

Sắp xếp lại các giá trị của mảng a. Sau đó duyệt i từ 1 đến n, mỗi lần duyệt tăng i lên 2 đơn vị. Mỗi lần duyệt kiểm tra nếu $a[i] = a[i+1]$. Ngay khi điều kiện này hết thoả thì ta xác định được phần tử $a[i]$ xuất hiện lẻ lần.

- Độ phức tạp thời gian: $\mathcal{O}(n \log n)$

Cách 2: Toán tử XOR

Kí hiểu của toán tử `XOR` là $\oplus$. Dựa vào tính chất của toán tử này:

- $a \oplus 0 = a$
- $0 \oplus a = a$
- $a \oplus a = 0$

Ta khai báo một biến đếm (tạm gọi là `answer`) rồi duyệt qua từng phần từ trong mảng a. Ở mỗi lần duyệt thì ta cập nhật `answer` như sau:

```
answer = answer ^ a[i];
```

Đến cuối cùng thì biến `answer` sẽ có giá trị của phần tử xuất lẻ lần.

- Độ phức tạp thời gian: $\mathcal{O}(n)$

Cài đặt (tham khảo)

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


int main() {
    freopen("FINDSHOE.inp", "r", stdin);
    freopen("FINDSHOE.out", "w", stdout);

    ios::sync_with_stdio(false);
    cin.tie(NULL);

    long long int n;
    cin >> n;
    long long int a;
    long long int result = 0;
    for (long long int i = 0; i < n; i++)
    {
        cin >> a;
        result ^= a;
    }

    cout << result;
    return 0;
}
```

Bài 2: Món quà cuối cấp (3,0 điểm)

Subtask 1: Vét cạn

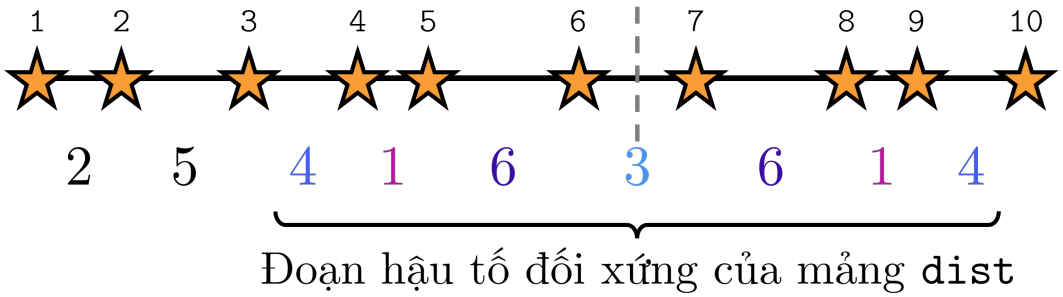
Ở subtask này, vì giới hạn $n, r \leq 100$ nên ta có thể sử dụng phương pháp vét cạn để thực hiện. Bằng cách duyệt qua từng tọa độ và chạy con trỏ hướng về hai đầu dây.

- Độ phức tạp: $\mathcal{O}(N \times R)$.

Subtask 2: Vét cạn + dãy đối xứng

Nếu để ý một chút, ta sẽ nhận thấy khi gấp dây các ngôi sao phải nằm đúng thứ tự trên đoạn dây. Vì vậy ta phải cần sắp xếp lại tọa độ các ngôi sao. Ta nhận thấy nếu điểm gấp dây là một ngôi sao thì khoảng cách của từng cặp ngôi sao từ ngôi sao nhận làm điểm gấp phải bằng nhau. Tương tự với điểm gấp không phải là ngôi sao, ta chỉ cần lấy điểm nằm ở trung điểm giữa hai ngôi sao liền kề. Và cứ như vậy đến khi chạm đến một đầu dây bên kia sẽ là một đáp án hợp lệ.

Ta xây dựng một dãy `dist`[] với `dist`$[i]$ là khoảng cách giữa ngôi sau thứ i và $i+1$ (với $1 \leq i < n$, chỉ số của các ngôi sao được đánh lại sau khi sắp xếp). Lúc này, một điểm gấp là hợp lệ khi nó là **tâm của một tiền tố hoặc hậu tố đối xứng**[†] (palindrome).



Hình 1: Một điểm gập hợp lệ giữa ngôi sao thứ 6 và 7 ứng với hậu tố độ dài 7 của mảng `dist`

Như vậy, ta duyệt mọi tiền tố và hậu tố của dãy `dist` rồi đếm số dãy đối xứng được duyệt.

- Độ phức tạp: $\mathcal{O}(N^2)$.

† Lưu ý, một đoạn con của dãy là tiền tố khi ta có thể tạo ra đoạn con đó bằng cách xóa một số phần tử cuối dãy. Tương tự, một đoạn con là hậu tố khi ta có thể tạo ra đoạn con đó bằng cách xóa một số phần tử đầu dãy. Một dãy số được gọi là đối xứng khi việc đọc dãy số từ trái sang phải hay từ phải sang trái đều như nhau.

Cài đặt (tham khảo)

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

const int N = 1e4 + 4;
int a[N], dist[N];

bool isPalindrome (int l, int r) {
    for (int i = 0; l+i <= r-i; i++)
        if (dist[l+i] != dist[r-i]) return false;
    return true;
}

int main()
{
    freopen("FOLDCORD.INP", "r", stdin);
    freopen("FOLDCORD.OUT", "w", stdout);
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    cin >> n >> r;
    for (int i = 1; i <= n; i++)
        cin >> a[i];
    sort(a+1, a+n+1);

    for (int i = 1; i < n; i++)
        dist[i] = a[i+1] - a[i];
    int ans = 0;
    for (int i = 1; i < n; i++)
        if (isPalindrome(1,i)) ans++;
    for (int i = 2; i < n; i++)
        if (isPalindrome(i,n-1)) ans++;
    cout << ans << endl;

    return 0;
}
```

Bài 3: Hoán đổi chữ số (3,0 điểm)

Trong phần này, ta định nghĩa $|N|$ là số lượng chữ số có nghĩa (tức số chữ số không tính các số 0 tận cùng bên trái) của số nguyên N.

Subtask 1: Vét cạn

Với giới hạn $N < 10^9$, ta hoàn toàn có thể đọc dữ liệu vào trong bài toán này dưới dạng số nguyên bình thường. Sau đó, nếu N không chia hết cho M ta duyệt từng cặp chữ số rồi xây dựng lại số nguyên mới sau khi hoán đổi rồi kiểm tra số này có chia hết cho M không.

Việc hoán đổi chữ số có thể được thực hiện qua hàm sau:

```cpp
int swapDigit (int num, int i, int j) {
    vector<int> vec;
    while (num > 0)
        vec.push_back(num % 10), num /= 10;
    swap(vec[i], vec[j]);

```

```
    num = 0;
    for (int u : vec) num = num * 10 + u;
    return num;
}
```

- Độ phức tạp: $\mathcal{O}(|N|^2)$.

Subtask 2: Vét cạn + số học

Có thể thấy, việc kiểm tra một số nguyên N chia hết cho M tương đương với việc kiểm tra $S = N \bmod M$ có bằng 0 hay không.

Trong subtask này, do $N < 10^{1000}$ (tức $|N| \leq 1000$) nên ta chỉ có thể đọc giá trị N dưới dạng chuỗi. Để xử lý số nguyên lớn này, ta đưa ra nhận xét rằng với một số nguyên $N = \overline{n_{|N|-1}n_{|N|-2}\dots n_1 n_0}$ trong hệ cơ số thập phân, ta có thể biểu diễn lại thành tổng:

$$N = n_0 \cdot 10^0 + n_1 \cdot 10^1 + n_2 \cdot 10^2 + \dots n_{|N|-1} \cdot 10^{|N|-1}$$

Tận dụng giới hạn $M \leq 10^9$ và tính chất phân phối của phép modulo, ta có thể tính S mà không bị tràn số. Cụ thể hơn, đoạn code sau tính $S = N \bmod M$ mà đảm bảo không tràn số:

```
long long S = 0;
for (int i = 0; i < N.size(); i++) S = (S * 10 + N[i] - '0') % M;
```

Sau đó, khi hoán đổi chữ số thứ i và j, ta có thể cập nhật lại giá trị của S như sau:

$$S \leftarrow S + (n_i - n_j) \cdot 10^j + (n_j - n_i) \cdot 10^i$$

Có thể hiểu rằng, thao tác hoán đổi cũng tương đương với việc "bỏ" chữ số n_i ở vị trí thứ i và n_j ở vị trí thứ j, sau đó "thêm lại" chữ số n_j ở vị trí thứ i và n_i ở vị trí thứ j. Khi tính toán với phép modulo, cần lưu ý xử lý số âm một cách khéo léo.

- Độ phức tạp: $\mathcal{O}(|N|^2)$.

Subtask 3: Số học + cấu trúc dữ liệu

Để tăng tốc cho thuật toán, ta duyệt qua từng vị trí i trong số nguyên N rồi tìm một vị trí j trong khoảng $[0; i-1]$ thích hợp để hoán đổi. Đặt $\Delta = n_j - n_i$, công thức cập nhật S được biến đổi thành:

$$S \leftarrow S - \Delta \cdot 10^j + \Delta \cdot 10^i$$

Mà ta muốn giá trị S sau khi cập nhật chia hết cho M, tức là:

$$\begin{aligned} S - \Delta \cdot 10^j + \Delta \cdot 10^i &\equiv 0 \pmod{M} \\ \iff S + \Delta \cdot 10^i &\equiv \Delta \cdot 10^j \pmod{M} \end{aligned}$$

Nói cách khác, ta cần chọn một vị trí j sao cho $\Delta \cdot 10^j$ đồng dư với $S + \Delta \cdot 10^i$ theo modulo M. Để ý rằng mỗi chữ số chỉ có thể là 1 trong 10 giá trị khác nhau, nên số lượng giá trị Δ phân biệt cũng chỉ là 9 (không tính $\Delta = 0$ vì khi đó thao tác hoán đổi vô nghĩa). Tóm lại, với mọi vị trí i, ta duyệt các giá trị $\texttt{val} \neq n_i$ từ 0 đến 9 rồi tìm một vị trí j thỏa:

$$\begin{cases} n_j = \texttt{val} \\ (\texttt{val} - n_i) \cdot 10^j \equiv S + \Delta \cdot 10^i \pmod{M} \end{cases}$$

Để tìm j một cách nhanh chóng, ta duy trì với mọi cặp chữ số (a, b) một cấu trúc dữ liệu `map` có `key` là kết quả của phép tính $(a - b) \cdot 10^j \bmod M$ và `value` là chỉ số tương ứng, khi xét các vị trí j đã duyệt thỏa $n_j = a$.

Khi đã cố định một vị trí i và giá trị `val`, ta có thể tìm vị trí j phù hợp bằng cách kiểm tra xem `map` của cặp $(\texttt{val}, n_i)$ có chứa `key` là $(S + \Delta \cdot 10^i) \bmod M$ hay không.

Bên cạnh đó, để tính $10^i \bmod M$ một cách nhanh chóng, ta có thể tiền xử lý trước một mảng `tenPow` với $\texttt{tenPow}[i] = 10^i \bmod M = \texttt{tenPow}[i-1] \cdot 10 \bmod M$.

- Độ phức tạp: $\mathcal{O}(10|N| \cdot \log(10|N|))$.

Cài đặt (tham khảo)

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

using ll = long long;
#define all(a) a.begin(), a.end()

map<int,int> cached[10][10];

int main()
{
    freopen("DGSWAP.inp", "r", stdin);
    freopen("DGSWAP.out", "w", stdout);

    ios::sync_with_stdio(0);
    cin.tie(0);

    string bigInt; ll m; cin >> bigInt >> m;

    // tiền xử lý
    vector<ll> tenPow(bigInt.size());
    tenPow[0] = 1;
    for (int i = 1; i < bigInt.size(); i++)
        tenPow[i] = tenPow[i - 1] * 10 % m;

    // tính giá trị S ban đầu
    ll S = 0;
    for (int i = 0; i < bigInt.size(); i++)
        S = (S * 10 + bigInt[i] - '0') % m;
    if (S == 0) { // không cần hoán đổi
        cout << "0";
        return 0;
    }
    reverse(all(bigInt));

    for (int i = 0; i < bigInt.size(); i++) {
        ll cur = bigInt[i] - '0';

        // tìm vị trí j phù hợp để hoán đổi
        for (int jDigit = 0; jDigit < 10; jDigit++) {
            ll delta = ((jDigit - cur) % m + m) % m;
            ll targ = (S + delta * tenPow[i]) % m;
            if (cached[jDigit][cur].count(targ)) { // tìm ra phương án hoán đổi
                int j = cached[jDigit][cur][targ];
                cout << "1 " << j << " " << i;
                return 0;
            }
        }

        // cập nhật map
        for (int iDigit = 0; iDigit < 10; iDigit++) {
            ll delta = ((cur - iDigit) % m + m) % m, val = delta * tenPow[i] % m;
            cached[cur][iDigit][val] = i;
        }
    }

    // không có phương án hoán đổi
    cout << -1;
    return 0;
}
```

— HẾT —

- Thí sinh **KHÔNG** được sử dụng tài liệu.
- Giám thị **KHÔNG** được giải thích gì thêm.