DỰ ÁN CHICKEN MINDS
BAN CHUYÊN MÔN TIN HỌC

KỲ THI THỬ TUYỂN SINH LỚP 10 THPT CHUYÊN
NĂM HỌC 2024 – 2025

HƯỚNG DẪN CHẤM THI
Đề thi thử đợt 2

Môn thi: **TIN HỌC**
Ngày thi: **28/4/2024**
Thời gian làm bài: **150 phút** (không kể thời gian phát đề)
Hướng dẫn chấm thi gồm 07 trang

Tổng quan đề thi

	Tên bài	Tập tin dữ liệu	Tập tin kết quả	Hạn chế thời gian	Hạn chế bộ nhớ
Bài 1	Đường tròn	CIRCLE.INP	CIRCLE.OUT	1 giây	512MB
Bài 2	Tổng	SUM.INP	SUM.OUT	1 giây	512MB
Bài 3	Đoạn con chia hết	SDS.INP	SDS.OUT	1 giây	512MB

I. Hướng dẫn chung

1. Bài thi của thí sinh được chấm thi bằng phần mềm chấm thi trực tuyến (online judge) trên website `https://oj.giftedbat.edu.vn/`, sử dụng bộ test của Tổ chức The Gifted Battlefield – Ban Tin học, đúng với đáp án và biểu điểm được nêu trong hướng dẫn chấm thi.
2. Điểm bài thi được xuất từ phần mềm chấm thi; không quy tròn điểm thành phần của từng câu và điểm của bài thi.

II. Lời giải

Bài 1. Đường tròn – CIRCLE (3,5 điểm)

Subtask 1

Giới hạn: $r \leq 2$.
Ta thấy với $r = 1$ thì chỉ có 5 vị trí phân biệt thỏa mãn và với $r = 2$ thì chỉ có 9 vị trí phân biệt thỏa mãn nên ta có thể kiểm tra tay với từng điểm.
Độ phức tạp: $O(n)$.

Subtask 2

Ta thấy khoảng cách giữa một điểm và gốc tọa độ là $d = \sqrt{x^2 + y^2}$. Để điểm đó nằm trong hình tròn bán kính r, cần thỏa điều kiện $d \leq r \Rightarrow \sqrt{x^2 + y^2} \leq r$ hay, $x^2 + y^2 \leq r^2$ và ta chỉ cần đếm số điểm thỏa mãn bất phương trình đó.
Độ phức tạp: $O(n)$.

Code mẫu:

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

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);

    freopen("CIRCLE.INP", "r", stdin);
```

```
    freopen("CIRCLE.OUT", "w", stdout);

    int n;
    long long r;
    cin >> n >> r;

    int cnt = 0;
    for (int i = 1; i <= n; i++) {
        long long x, y;
        cin >> x >> y;
        if (x * x + y * y <= r * r) cnt++;
    }

    cout << cnt;

    return 0;
}
```

Bài 2. Tổng – SUM (3,5 điểm)

Subtask 1

Giới hạn: Dãy số ban đầu của chỉ có đúng một phần tử.
Vì dãy số ban đầu chỉ có **một** số duy nhất, các ký tự chữ cái sẽ chỉ nằm ở hai đầu của xâu s. Đáp án: duyệt qua xâu ban đầu và xuất ra **các kí tự chữ số** theo thứ tự từ trái sang phải, tạo thành số duy nhất trong dãy số ban đầu.
Độ phức tạp: $O(n)$.

Subtask 2

Giới hạn: Đáp án bài toán không vượt quá 10^{18}.
Hướng đi chung: Vì mỗi số trong dãy ban đầu được biểu diễn bởi một xâu con liên tiếp của s mà mọi phần tử trong xâu con đều là chữ số nên ta chỉ cần tách các xâu con này ra thành các số, không quan tâm đến những ký tự là chữ cái. Sau đó, chuyển đổi các số từ `string` sang `long long` rồi cộng như bình thường.
Cài đặt:

- Duyệt qua từng chỉ số trong xâu s và duy trì một biến số hiện tại `cur`
- Khi gặp một chữ số ($s[i]$ là chữ số) thì nhân số hiện tại lên 10 và cộng số đang xét vào hàng đơn vị: `cur = cur * 10 + s[i] - '0'`.
- Khi gặp một chữ cái ($s[i]$ là chữ cái), cộng số hiện tại vào đáp án và đặt lại số về 0: `sum += cur` và sau đó `cur = 0`.

Độ phức tạp: $O(n)$.

Subtask 3

Giới hạn: $n \leq 2000$.
Yêu cầu kiến thức: Kỹ thuật xử lý số lớn bằng string (Bignum).
Với số lượng chữ số đã trở nên quá lớn và không có giới hạn trên giá trị đầu ra, ta không thể lưu các số lấy được dưới định dạng số nguyên trong các ngôn ngữ lập trình và cộng bình thường giống subtask 2 (đáp án cuối cùng có thể là một số có hàng ngàn chữ số). ⇒ Sử dụng kĩ thuật ***Bignum***: biểu diễn các số dưới dạng `string` hoặc `vector` rồi thực hiện cộng tương tự như phép cộng tiểu học.

```
string add(string a, string b)  // tra ve a + b {
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());

    if (a.size() > b.size())
        swap(a, b);  // string a represents the number with fewer digits

```

```
    string res = "";
    int carry = 0;
    for (int i = 0; i < a.size(); i++) {
        int sum = a[i] - '0' + b[i] - '0' + carry;
        int digit = sum % 10;
        res += digit + '0';
        carry = sum / 10;
    }

    for (int i = a.size(); i < b.size(); i++) {
        int sum = b[i] - '0' + carry;
        int digit = sum % 10;
        res += digit + '0';
        carry = sum / 10;
    }

    if (carry > 0) res += carry + '0';

    reverse(res.begin(), res.end());
    return res;
}
```

Hàm `add` trên có độ phức tạp $O(\max(|a|, |b|))$.
Khi duyệt tuần tự xâu s, trích các xâu con của s thành các số dưới dạng `string` rồi lần lượt cộng chúng với nhau, thực tế ta sẽ thu được một thuật toán xấp xỉ $O(n^2)$, đủ để vượt qua subtask này.
Độ phức tạp: $O(n^2)$.

Subtask 4

Với thuật toán như ở subtask 3, nếu ta gặp phải một xâu s có dạng:
"78341238467...[5×10^5 chữ số]1a1a1a1a1a1a1a1a1...[5×10^5]"
thì thuật toán sẽ cho ra độ phức tạp $O(n^2)$. Ta cần thêm một vài bước xử lý để cải tiến thuật toán chạy được trong giới hạn cho phép.
Nhận xét là ta có thể cộng lại tất cả số sau khi đã trích xuất đủ các số từ xâu s, mà không cần cộng Bignum tuần tự mỗi khi trích xuất được số mới. Như vậy, sau khi trích xuất được một số, ta lưu lại từng số ở hàng đơn vị, hàng chục, hàng trăm,... Đến cuối ta có thể duyệt qua từng hàng và cộng dồn đáp án lên, với cách cài đặt tương tự với nguyên lý cộng của subtask 3.

Code mẫu

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

using namespace std;

int n;
string s;

vector<int> ans;

signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    freopen("SUM.INP", "r", stdin);
    freopen("SUM.OUT", "w", stdout);
    cin >> n >> s;
    vector<int> num;
    for (int i = n - 1; i >= 0; i--) {
        if ('0' <= s[i] && s[i] <= '9') {
            num.push_back(s[i] - '0');
        } else {
            int carry = 0;
            if (num.size()) {
                for (int i = 0; i < num.size(); i++) {
                    if (ans.size() <= i) ans.push_back(0);
```

```
                    ans[i] += num[i];
                }
            }
            num.clear();
        }
    }
    int carry = 0;
    for (int i = 0;; i++) {
        if (ans.size() <= i && carry == 0) break;
        if (ans.size() <= i) ans.push_back(0);
        ans[i] = (ans[i] + carry);
        carry = ans[i] / 10;
        ans[i] = ans[i] % 10;
    }

    reverse(ans.begin(), ans.end());
    for (int c : ans) cout << c;
    return 0;
}
```

Độ phức tạp: $O(n)$.

Ghi chú:

- Phương pháp khác #1: Sau khi trích xuất được tất cả số nguyên trong dãy số ban đầu, ta sắp xếp lại các số đó theo số lượng chữ số từ nhỏ đến lớn, số có số lượng chữ số càng nhỏ thì đứng trước. Sau đó, ta sẽ duyệt tuần tự qua các số đã sắp xếp theo thứ tự này và cộng Bignum như subtask 3. Phần chứng minh tính đúng đắn và độ phức tạp của thuật toán này xin nhường lại như một bài tập cho độc giả.
- Phương pháp khác #2: Cải tiến hàm `add(a, b)` để chạy trong $O(\min(|a|, |b|))$ ***trên trung bình***, tức là: `add("9999999", "1")` vẫn sẽ chạy trong $O(\max(|a|, |b|))$ (vì phải cộng dồn biến nhớ `carry` đến cuối dãy) nhưng `add("1273616234", "23")` sẽ chạy trong $O(\min(|a|, |b|))$ vì không cần cộng dồn biến nhớ `carry` nhiều.
- Bài toán này có nhiều phương pháp cải tiến, bạn đọc có thể tự tư duy và tìm hiểu thêm.

Bài 3. Đoạn con chia hết – SDS (3,0 điểm)

Subtask 1

Giới hạn: x là số nguyên tố.
Khi x là số nguyên tố, đáp án (nếu có) chỉ có độ dài tối đa là 1.
Duyệt i từ 1 đến n, nếu A_i chia hết cho x, ngay lập tức thoát vòng lặp và xuất ra đáp án là (i, i), ngược lại, nếu không tồn tại A_i chia hết cho x, đáp án sẽ là $(-1, -1)$. Cách làm này vẫn đảm bảo nếu có nhiều A_i khác nhau chia hết cho x thì sẽ xuất ra đáp án có chỉ số nhỏ nhất.

Độ phức tạp: $O(n)$.

Subtask 2

Giới hạn: x có đúng ba ước.
Nhận xét 1: Một số có số lượng ước là lẻ khi và chỉ khi số đó là số chính phương
Nhận xét 2: Một số x có đúng 3 ước sẽ là số chính phương có dạng $x = p^2$ với p là số nguyên tố.
Như vậy, trong subtask 2, x sẽ là số có dạng p^2 (p là số nguyên tố).
Khi, ta chỉ quan tâm các A_i có p là ước ($A_i \bmod p = 0$):

- Nếu $A_i \bmod x = 0$, tức phần tử A_i chia hết cho x, thì (i, i) chính là đáp án đề bài (nếu có nhiều i thì chọn i nhỏ nhất).
- Nếu $A_i \bmod p = 0$, gắn `flag[i] = 1`.
- Các phần tử còn lại ta không cần quan tâm.

Sau khi hoàn tất gắn `flag`, bài toán sẽ trở thành tìm đoạn con liên tiếp ngắn nhất của mảng `flag` chứa ít nhất hai phần tử có giá trị 1 (bạn đọc hãy thử tự chứng minh nhận xét này).
Nhận xét: Đáp án (nếu có) sẽ có dạng `[1, 0, 0, 0,..., 1]`. Nói cách khác, nếu (l, r) là đáp án, ta sẽ có `flag[l] = 1`, `flag[r] = 1` và tất cả phần tử ở giữa sẽ có giá trị `0`.
Ta có thể thực hiện bài toán trên bằng cách duyệt i qua mảng `flag`. Trong suốt quá trình đó, lưu biến l sao cho l là vị trí phải nhất ($l < i$) có `flag[l] = 1` (khởi tạo `l = -100000000`). Nếu `flag[i] = 1`, (l, i) sẽ là một đoạn thỏa đề bài; khi đó, cập nhật (l, i) với kết quả tối ưu đang xét hiện tại, sau đó gán `l = i` và tiếp tục duyệt i đến những phần tử tiếp theo.

Độ phức tạp: $O(n)$.

Subtask 3

Giới hạn: x là tích của hai số nguyên tố.
Như vậy, x có dạng $p \times q$ với p, q đều là số nguyên tố.
Ta sẽ thực hiện tương tự subtask 2, nhưng mảng `flag` lúc này sẽ có thể chứa ba giá trị khác nhau:

- Nếu $A_i \bmod x = 0$, tức phần tử A_i chia hết cho x, thì (i, i) chính là đáp án đề bài (nếu có nhiều i thì chọn i nhỏ nhất).
- Nếu $A_i \bmod p = 0$, gắn `flag[i] = 1`.
- Nếu $A_i \bmod q = 0$, gắn `flag[i] = 2`.
- Các phần tử còn lại ta không cần quan tâm.

Bài toán biến đổi thành: Cho mảng `flag`, tìm đoạn con liên tiếp ngắn nhất sau cho đoạn con đó chứa ít nhất một phần tử 1 và ít nhất một phần tử 2. Cách giải tương tự như subtask trên.

Độ phức tạp: $O(n)$.

Subtask 4

Giới hạn: $n \leq 2000, x \leq 10^9$**.**

- Hướng giải quyết: Duyệt qua tất cả đoạn con liên tiếp (l, r), tính tích của từng đoạn con (l, r) dưới modulo x, nếu tích đó bằng 0 thì (l, r) là một đoạn con thỏa $\Rightarrow$ cập nhật với đáp án tối ưu hiện tại.
- Cách làm:
  - Duyệt i từ $1 \to n$, với mỗi i, duyệt j từ $i \to n$. $\Rightarrow$ Ta đang xét đoạn con (i, j).
  - Với mỗi i, ta khởi tạo biến `product = 1`.
  - Với mỗi j, ta thực hiện: `product = a[j] % x * product % x`.
  - Sau đó, xét điều kiện nếu `product == 0` thì (i, j) là một đoạn con thỏa.

Độ phức tạp: $O(n^2)$.

Subtask 5

Yêu cầu kiến thức: 2 con trỏ.

- Gọi hàm $P(n)$ là số lượng **ước số nguyên tố** tối đa của n. Dễ thấy với các số $n \leq 10^{12}$ thì $P(n)$ tối đa chỉ có giá trị không vượt quá 12. Ta sẽ xét từng **số nguyên tố** p là ước của x.
- Với mỗi số p này, tính số mũ của p trong biểu diễn tích các thừa số nguyên tố của x. Gọi số mũ này là `req[p]`:

$$x = p_1^{req[p_1]} \times p_2^{req[p_2]} \times ...$$

- Sau đó, với mỗi p, ta xây một mảng mới: $B[i] = y$ với $A[i]$ có dạng $p^y \times D$ và y lớn nhất có thể. Ví dụ : ta có mảng `A = {1, 2, 8, 5, 12}` với `p = 2` thì mảng `B = {0, 1, 3, 0, 2}`
- Vì tích của một đoạn cần chia hết cho x nên với $p = 2$, tổng của một đoạn trên mảng B không được bé hơn `req[2]`. Từ đó, ta cần tìm và lưu lại, với mỗi vị trí i trong mảng B, vị trí j lớn nhất ($j \leq i$) sao cho tổng $B[j..i] \geq req[p]$.
- Xét một vị trí i là điểm kết thúc của đoạn, vì bài toán gốc yêu cầu một vị trí j thỏa $B[j..i] \geq req[p]$ $\forall p$ nên ta phải lấy p có giá trị j nhỏ nhất làm chuẩn (vì nếu j của bài toán gốc lớn hơn thì sẽ không thỏa bài toán con p đó).
- Cuối cùng ta duyệt qua từng vị trí i và tìm đoạn con nhỏ nhất thỏa mãn.
- Tại gạch đầu dòng thứ 4, ta có thể xử lý bằng kỹ thuật hai con trỏ với nhận xét: nếu (j, i) là một đoạn thỏa mãn thì $(j-1, i)$ chắc chắn cũng là một đoạn thỏa mãn.
- Lưu ý trường hợp đặc biệt $x = 1$, một số cách cài đặt không cẩn thận sẽ tạo ra lỗi/đáp án sai trong trường hợp này.

Độ phức tạp: $O(P(x)n)$.

Code mẫu

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

const ll maxn = 5 * 1e5 + 5, INF = 1e9 + 9;

void solve() {
    ll n, x;
    cin >> n >> x;
    vector<ll> a(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    if (x == 1) {
        cout << "1 1"
             << "\n";
        return;
    }

    vector<ll> prime, req;
    for (int i = 2; i <= sqrt(x); i++) {
        if (x % i == 0) {
            prime.push_back(i);
            req.push_back(0);
            while (x % i == 0) {
                req[req.size() - 1]++;
                x /= i;
            }
        }
    }

    if (x > 1) {
        prime.push_back(x);
        req.push_back(1);
    }

    vector<int> res(n + 1, INF);
    for (int it = 0; it < prime.size(); it++) {
        ll p = prime[it], k = req[it];
        vector<ll> b(n + 1, 0);
        for (int i = 1; i <= n; i++) {
```

```
            ll t = a[i];
            while (t % p == 0) {
                b[i]++;
                t /= p;
            }
        }
        int j = 0;
        b[0] = INF;
        ll sum = INF;
        for (int i = 1; i <= n; i++) {
            sum += b[i];
            while (j < i && sum - b[j] >= k) {
                sum -= b[j];
                j++;
            }
            if (j == 0) res[i] = 0;
            res[i] = min(res[i], j);
        }
    }

    int ans = INF, ansl = -1, ansr = -1;
    for (int i = 1; i <= n; i++) {
        if (res[i] == 0) continue;
        if (i - res[i] + 1 < ans) {
            ansl = res[i];
            ansr = i;
            ans = i - res[i] + 1;
        }
    }

    if (ans > n) {
        cout << -1 << " " << -1;
        return;
    }

    cout << ansl << " " << ansr;
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    freopen("SDS.inp", "r", stdin);
    freopen("SDS.out", "w", stdout);
    int t;
    cin >> t;
    while (t--) {
        solve();
        cout << "\n";
    }
    return 0;
}
```

— HẾT —