TỔ CHỨC THE GIFTED BATTLEFIELD
DỰ ÁN INTELLECTUAL SHOWDOWN

Thi thử HSGTP Lớp 9 - Lần 2
NĂM HỌC 2023 – 2024

Ngày thi: **10/03/2024**
Thời gian làm bài: **180 phút** (không kể thời gian phát đề)
Đề thi gồm 05 trang, 03 bài

Tổng quan đề thi

	Tên bài	Tập tin dữ liệu	Tập tin kết quả	Hạn chế thời gian	Hạn chế bộ nhớ
Bài 1	Đoạn đường liên kết dài nhất	LGS.INP	LGS.OUT	2 giây	512MB
Bài 2	Dãy đẹp	DAYDEP.INP	DAYDEP.OUT	2 giây	512MB
Bài 3	Mã hóa	ENCRYPTION.INP	ENCRYPTION.OUT	2 giây	512MB

Bài 1. Đoạn đường liên kết dài nhất - LGS

Subtask 1

Ý tưởng: Với mỗi trạm giao thông i, tìm vị trí r $(r \geq i)$ lớn nhất sao cho đoạn con liên tiếp $k[i...r]$ là một đoạn đèn giao thông thỏa đề bài.
Cài đặt:

- Với mỗi vị trí i, khởi tạo biến $r = i$. Trong khi $gcd(k_r, k_{r+1}) \neq 1$ và $r + 1 \leq n$ thì ta tịnh tiến `r++`. Giá trị cuối cùng của biến r chính là giá trị thỏa phần "Ý tưởng" trên.
- Hàm $gcd(x, y)$: Duyệt j qua tất cả giá trị từ 1 đến $min(x, y)$, đáp án của hàm chính là giá trị j lớn nhất sao cho x và y đồng thời chia hết cho j.

Độ phức tạp: $O(n^2 \times k_i)$.

Subtask 2

Ý tưởng tương tự như subtask 1. Nhưng hàm $gcd(x, y)$ được cài tối ưu trong $O(\log(\max(x, y)))$ với *thuật toán* ***Euclid***.
Độ phức tạp: $O(n^2 \times \log k_i)$.

Subtask 3

Nhận xét: Ta nén cả dãy lại thành các thành phần sao cho hai trạm giao thông liên tiếp bất kỳ nếu **không** nguyên tố cùng nhau sẽ thuộc chung một thành phần. Có thể thấy, sẽ không có bất kỳ trạm giao thông nào thuộc hai thành phần cùng lúc.
Từ đó, ta có giải pháp như sau:

- Đi qua cả con đường.

- Tuần tự kiểm tra xem trạm đèn giao thông hiện tại có nguyên tố cùng nhau với trạm đèn giao thông trước hay không:
  - Nếu **không**, ta thêm nó vào chung thành phần với trạm trước.
  - Nếu **có**, ta tách ra một thành phần mới.
- Tìm thành phần với độ dài lớn nhất (điều này có thể làm được ngay khi ta đi qua con đường).

Độ phức tạp: $O(n \times \log k_i)$.

Code mẫu:

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

int n;

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(0);
    freopen("LGS.inp", "r", stdin);
    freopen("LGS.out", "w", stdout);
    cin >> n;
    int maxi = 0, lastnum = 0, currlen = 0;
    for (int i = 0; i < n; i++) {
        int k;
        cin >> k;
        if (i > 0 && __gcd(k, lastnum) == 1) { //__gcd() la ham co san cua C++ giup tinh UCLN cua 2 so trong O(log)
            maxi = max(maxi, currlen);
            currlen = 0;
        }
        lastnum = k;
        currlen++;
    }
    maxi = max(maxi, currlen);
    cout << maxi;
}
```

Bài 2. Dãy đẹp - DAYDEP

Ý tưởng

Ta nhận xét rằng, với mỗi giá trị x, ta có thể tham lam bằng cách sẽ sử dụng số operation xóa hoặc thêm ít nhất sao cho trong dãy cuối cùng giá trị x đó còn đúng x hoặc 0 lần xuất hiện.
Gọi cnt là số lần xuất hiện của một giá trị x.
Ta có ba trường hợp sau:

- $cnt > 0$ và ta muốn đưa cnt về 0: cnt operations.
- $cnt < x$ và ta muốn đưa cnt về x: $x - cnt$ operations.
- $cnt > x$ và ta muốn đưa cnt về x : $cnt - x$ operations.

Từ ba công thức trên, ta có thể rút các trường hợp lại thành công thức đơn giản sau: $min(cnt, abs(x - cnt))$.

Subtask 1

Ta đặt biến đếm $cnt_0, cnt_1, cnt_2, cnt_3, cnt_4, cnt_5$ cho từng giá trị a_i có thể xuất hiện.
Đối với mỗi giá trị x, số operation ít nhất để đưa số lần xuất hiện từ cnt về 0 hoặc x lần xuất hiện là $min(cnt, abs(cnt - x))$. Đáp án sẽ là $\sum_{i=0}^{5} min(cnt_i, abs(cnt_i - i))$.
Độ phức tạp: $O(1)$.

Subtask 2

Cách làm của ta tương tự như subtask 1, nhưng ta sẽ sử dụng một mảng thống kê có size $MAX_a + 1$ để đếm số lần xuất hiện của từng giá trị x.
Trong subtask này, $MAX_a = 10^6$ nên ta sẽ for qua từng giá trị của a để tính kết quả của chúng.
Ta có công thức tính đáp án như sau: $\sum_{i=0}^{MAX_a} min(cnt_i, abs(cnt_i - i))$.
Độ phức tạp : $O(MAX_a)$

Subtask 3

Ta xét giá trị x có số lần xuất hiện là cnt, nhận xét được khi giá trị $x \geq 2 \cdot n$, số operations để đưa cnt của chúng về x luôn lớn hơn số operations để đưa cnt về 0 ($cnt \leq x - cnt$ do $x \geq 2 \cdot n$ và $cnt \leq n$).
Theo tính chất trên, với mọi $a_i > 2 \cdot n$, ta xóa chúng ra khỏi dãy để đưa số lần xuất hiện về 0. Đối với các phần tử $x \leq 2 \cdot n$ còn lại, ta làm tương tự subtask 2 với độ lớn của mảng thống kê là $2 \cdot n$.
Độ phức tạp: $O(n)$

Code mẫu:

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

using namespace std;

const int N = 1e6 + 5;
int n, cnt[2 * N];
signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    freopen("DAYDEP.inp", "r", stdin);
    freopen("DAYDEP.out", "w", stdout);
    cin >> n;
    int ans = 0;
    for(int i = 1; i <= n; i++) {
        int x; cin >> x;
        if(x > 2 * n) {
            ans++;
        } else {
            cnt[x]++;
        }
    }
    for(int i = 0; i <= 2 * n; i++) {
        ans += min(cnt[i], abs(i - cnt[i]));
    }
    cout << ans << "\n";
    return 0;
}
```

Bài 3. Mã hóa - Encryption

Subtask 1 và 2

Ý tưởng và cài đặt: Ta trâu qua mọi S mà $L \leq S \leq R$ và đếm số lượng thỏa yêu cầu đề bài.
Độ phức tạp: $O(R - L)$.

Code mẫu:

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

using namespace std;

// Function to count numbers between l and r such that a * s % k == 0
long long countMultiples(long long a, long long k, long long l, long long r) {
    int dem = 0;
    for(int i = l; i <= r; ++i) {
        if (a * i % k == 0) ++dem;
    }
    return dem;
}

int main() {
    freopen("ENCRYPTION.inp", "r", stdin);
    freopen("ENCRYPTION.out", "w", stdout);
    long long a, k, l, r;
    cin >> l >> r >> a >> k;

    long long result = countMultiples(a, k, l, r);
    cout << result;

    return 0;
}
```

Subtask 3

Ý tưởng:

- Đặt $P = LCM(A, K)$.
- $A \times S \; mod \; k = 0$. Vậy $A \times S$ là bội của P.
- Tìm Min là bội nhỏ nhất của P thỏa yêu cầu đề bài.
- Tìm Max là bội lớn nhất của P thỏa yêu cầu đề bài.
- Kết quả bài toán sẽ là $(Max - Min)/P + 1$.

Độ phức tạp: $O(log(min(a, k)))$.

Code mẫu:

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

using namespace std;

// Function to count numbers between l and r such that a * s % k == 0
long long countMultiples(long long a, long long k, long long l, long long r) {
    long long gcd_ak = __gcd(a, k);
    long long lcm_ak = (a / gcd_ak) * k; // Calculate the LCM of a and k

```

```
    // Adjust l to the next multiple of lcm_ak if l is not a multiple of lcm_ak
    l *= a;
    if (l % lcm_ak != 0)
        l = ((l / lcm_ak) + 1) * lcm_ak;

    // Adjust r to the previous multiple of lcm_ak if r is not a multiple of lcm_ak
    r *= a;
    if (r % lcm_ak != 0)
        r = (r / lcm_ak) * lcm_ak;

    // Count the numbers between l and r (inclusive) that are multiples of lcm_ak
    return (r < l) ? 0 : ((r - l) / lcm_ak + 1);
}

int main() {
    freopen("ENCRYPTION.inp", "r", stdin);
    freopen("ENCRYPTION.out", "w", stdout);
    long long a, k, l, r;
    cin >> l >> r >> a >> k;

    long long result = countMultiples(a, k, l, r);
    cout << result << endl;

    return 0;
}
```