DỰ ÁN INTELLECTUAL SHOWDOWN
BAN CHUYÊN MÔN TIN HỌC

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

LỜI GIẢI THAM KHẢO

Môn thi chuyên: **TIN HỌC**
Ngày thi: **07/06/2024**
Thời gian làm bài: **150 phút** (không kể thời gian phát đề)
Lời giải tham khảo gồm 05 trang

Bài 1. Tuyệt chiêu – TUYETCHIEU

Subtask 1: $n \leq 10^3$

- Ta sẽ duyệt qua từng cặp phần tử trong dãy số. Với mỗi cặp gồm hai phần tử giống nhau, ta sẽ kiểm tra xem tuyệt chiêu đó có được sử dụng đúng quy định hay không.
- Nếu tuyệt chiêu được sử dụng sai quy định thì ta sẽ cập nhật lại biến min, là mã số của tuyệt chiêu vi phạm có mã nhỏ nhất.
- Vì ta duyệt qua từng cặp phần tử trong mảng nên độ phức tạp thuật toán là $O(n^2)$.

Độ phức tạp: $O(n^2)$.

Subtask 2: $n \leq 10^6$

- Đầu tiên, ta đưa ra nhận xét: với một lần sử dụng tuyệt chiêu có mã là x, ta không cần phải kiểm tra nó với mọi lần sử dụng của x, mà chỉ cần so sánh với lần sử dụng tuyệt chiêu ngay **trước** nó.
- Để ý thấy rằng, giá trị của $a_i \leq 10^6$, vậy nên ta có thể duy trì một mảng $last$ lưu lại lần sử dụng gần nhất của tuyệt chiêu x.
- Thay vì phải duyệt qua từng cặp phần tử như subtask 1, ta chỉ cần duyệt xét phần tử thứ i với phần tử $last$ nên độ phức tạp thuật toán của ta sẽ là $O(n)$.

Độ phức tạp: $O(n)$.

Code mẫu:

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

using namespace std;

const int N = 1e6 + 5, MAX = 1e6, INF = 2e9 + 7;

int n, k, a[N], last[N];
signed main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    freopen("TUYETCHIEU.INP", "r", stdin);
    freopen("TUYETCHIEU.OUT", "w", stdout);

    cin >> n >> k;

    for (int i = 1; i <= MAX; i++) last[i] = -INF;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
```

```
21
22      int mn = MAX + 1;
23      for (int i = 1; i <= n; i++) {
24          if (last[a[i]] + k > i) mn = min(mn, a[i]);
25          last[a[i]] = i;
26      }
27
28      if (mn == MAX + 1)
29          cout << -1 << "\n";
30      else
31          cout << mn << "\n";
32      return 0;
33  }
```

Bài 2. Đắp núi – DAPNUI

Subtask 1: $(3 \leq n \leq 10^3)$

Ta xét mọi vị trí x $(1 < x < n)$ và giả sử rằng a_x là đỉnh núi, ta xây dựng mảng b sao cho sau khi thực hiện thao tác cộng 1 vào mảng a, b trở thành ngọn núi và đồng thời, b_x là đỉnh núi.
Đầu tiên ta gán $b_1 = a_1$, $b_n = a_n$, duyệt từ $2 \to x$ và gán $b_i = max(b_{i-1} + 1, a_i)$, duyệt từ $n - 1 \to x$ và gán $b_i = max(b_{i+1} + 1, a_i)$. Ta hiểu đơn giản rằng a_i cần phải lớn hơn giá trị trước nó ít nhất một đơn vị ($a_i \geq a_{i-1} + 1$ hoặc $a_i \geq a_{i+1} + 1$, tùy vị trí đang xét là bên trái hay bên phải đỉnh núi).
Sau cùng số, thao tác cần thực hiện để xây ngọn núi với a_x là đỉnh núi là:

$$\sum_{i=1}^{n}(b_i - a_i)$$

Với mỗi giá trị x $(1 < x < n)$, ta cần tính lại đáp án trong $O(n)$.
Độ phức tạp: $O(n^2)$.

Subtask 2: $(3 \leq n \leq 10^5)$

Gọi mảng $pref$ là giá trị mảng a sau khi cộng 1 để $a_{i-1} < a_i$ $(1 < i \leq n)$.
Gọi mảng $suff$ là giá trị mảng a sau khi cộng 1 để $a_{i-1} > a_i$ $(1 < i \leq n)$.
Gọi mảng $prefSum$ được định nghĩa như sau:

$$prefSum_i = \sum_{j=1}^{i}(pref_j - a_j)$$

Gọi mảng $suffSum$ được định nghĩa như sau:

$$suffSum_i = \sum_{j=i}^{n}(suff_j - a_j)$$

Cách tính mảng $pref$ và $suff$ cũng gần như tương tự với cách tính mảng b ở Subtask1.
Còn để tính mảng $prefSum$ và $suffSum$, ta có thể áp dụng kiến thức mảng cộng dồn.
Như vậy ta có thể xét mọi vị trí x $(1 < x < n)$ và tính kết quả cho mỗi giá trị x trong $O(1)$.
Số thao tác cần thực hiện để xây ngọn núi có a_x là đỉnh núi được tính bằng công thức:

$$prefSum_{x-1} + suffSum_{x+1} + max(pref_{x-1} + 1, suff_{x+1} + 1, a_x) - a_x$$

Độc giả được khuyến khích tự chứng minh tính đúng đắn và hiểu rõ bản chất của công thức này trước khi áp dụng vào bài làm.
Độ phức tạp: $O(n)$.

Code mẫu

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

const int N = 1e5 + 10;
const int V = 1e7 + 10;
const long long INF = 1e18;
const long long M = 1e9 + 7;

int a[N + 1], prefE[N + 1], suffE[N + 1], n;
long long pref[N + 1], suff[N + 1];

void readData() {
    cin >> n;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
}

void precompute() {
    prefE[1] = a[1];
    suffE[n] = a[n];

    for (int i = 2; i <= n; ++i) {
        prefE[i] = max(a[i], prefE[i - 1] + 1);
        pref[i] = pref[i - 1] + prefE[i] - a[i];
    }

    for (int i = n - 1; i >= 1; --i) {
        suffE[i] = max(a[i], suffE[i + 1] + 1);
        suff[i] = suff[i + 1] + suffE[i] - a[i];
    }
}

long long solve() {
    long long ans = INF;

    for (int i = 2; i < n; ++i) {
        long long ele = max(a[i], max(prefE[i - 1], suffE[i + 1]) + 1);
        ans = min(ans, pref[i - 1] + suff[i + 1] + ele - a[i]);
    }

    return ans;
}

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

    freopen("DAPNUI.INP", "r", stdin);
    freopen("DAPNUI.OUT", "w", stdout);

    readData();
    precompute();
    cout << solve();

    return 0;
}
```

Bài 3. Lọc nước – LOCNUOC

Subtask 1: $1 \leq n \leq 10^3, 1 \leq m \leq 10^2$

- Ta có thể dựng lại mảng hai chiều kích thước $n \times m$ như trong đề bài, rồi làm như đề yêu cầu với độ phức tạp $O(n \times m)$.

Subtask 2: $1 \leq n \leq 10^3, 1 \leq m \leq 2 \times 10^5$

Lưu ý: *Đội ngũ TGB quyết định sửa giới hạn của m thành 2×10^5, khác với đề gốc (để phù hợp với kích thước dữ liệu nhập vào). Giới hạn này cũng được áp dụng cho bộ test chấm tham khảo trên contest của TGBOJ.*

- Ta nhận thấy rằng $a_i \leq 10^5$, nên chắc chắn rằng nhiều nhất 10^5 bồn chứa sẽ được sử dụng đến. Vì thế, ta chỉ quan tâm nhiều nhất 10^5 bồn chứa. Nên nếu $m > 10^5$ thì ta có thể đặt m là 10^5 luôn.
- Nếu như ta sử dụng thuật toán như Subtask 1, chỉ lưu mảng đáp án của m máy và cập nhật đáp án từ từ thì độ phức tạp sẽ là $O(n \times \min(m, 10^5))$, xấp xỉ $10^8 \rightarrow$ vừa đủ thời gian chạy. Trên thực tế, thời gian chạy sẽ thấp hơn rất nhiều.

Subtask 3: $1 \leq n \leq 2 \times 10^5, 1 \leq m \leq 2 \times 10^5$

Lưu ý: *Đội ngũ TGB quyết định sửa giới hạn của m thành 2×10^5, khác với đề gốc (để phù hợp với kích thước dữ liệu nhập vào). Giới hạn này cũng được áp dụng cho bộ test chấm tham khảo trên contest của TGBOJ.*

- Ta mặc định $m \leq 10^5$ (lý do được giải thích ở subtask 2).
- Ta thấy rằng với giá trị a_i thì nó sẽ lấp đầy hết k_i bể đầu tiên. Vì thế, ta có thể đánh dấu là bể thứ i được lấp đầy bao nhiêu lần và được lặp không đầy một lượng nhiêu. Việc đánh dấu thứ hai có thể thực hiện trong $O(1)$. Nhưng việc đánh dấu đầu tiên thì khó hơn.
- Ta xét bài toán con: Cho q lần truy vấn cập nhật đoạn $[l, r]$ tăng lên giá trị c; và cuối cùng, sau tất cả q truy vấn, ta cần xuất ra giá trị của tất cả các phần tử trong dãy. Bài toán này có thể giải quyết trong độ phức tạp $O(n+q)$ với ứng dụng của **mảng hiệu**. Vì thế, bước đánh dấu đầu ta có thể sử dụng mảng hiệu.
- Vấn đề cuối cùng là phương pháp tìm giá trị k_i với mỗi giá trị a_i. Ta thấy, với a_i càng lớn thì k_i cũng sẽ càng lớn nên ta có thể sử dụng tìm kiếm nhị phân để tìm k_i.
- Như vậy thuật toán của chúng ta sẽ là: với mỗi giá trị a_i, ta sẽ sử dụng mảng hiệu để cập nhật k_i máy đầu tiên là bể được lấp đầy và cập nhật máy $k_i + 1$ được thêm một lượng nước thừa nào đó. Cuối cùng, ta sẽ sử dụng mảng hiệu và số lượng nước dư của mỗi máy để tính lượng nước mỗi máy xử lý. Độ phức tạp của thuật toán là $O(n \times \log(m) + m)$.

Code mẫu

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

using namespace std;

ll diff[200009];
ll a[200001];
ll r[200001];
ll pref[200001];
ll n, m;
ll ans[200001];

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

    freopen("LOCNUOC.INP", "r", stdin);
    freopen("LOCNUOC.OUT", "w", stdout);
```

```
    cin >> n >> m;

    m = min(m, (ll)2e5);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    for (int i = 1; i <= m; i++) {
        cin >> r[i];
    }

    for (int i = 1; i <= m; i++) {
        pref[i] = pref[i - 1] + r[i];
    }

    ll mx = 0;
    for (int i = 1; i <= n; i++) {
        ll pos = lower_bound(pref, pref + m + 1, a[i]) - pref;
        mx = max(mx, pos);
        diff[0]++;
        diff[pos]--;
        ll over = a[i] - pref[pos - 1];
        ans[pos] += over;
    }

    for (int i = 1; i <= m; i++) {
        diff[i] += diff[i - 1];
    }

    for (int i = 1; i <= mx; i++) {
        cout << diff[i] * r[i] + ans[i] << ' ';
    }
}
```

— HẾT —