

DÃY CON TĂNG DẦN DÀI NHẤT

Bài toán: Viết chương trình chọn ra dãy tăng dần dài nhất từ một dãy số nguyên.

Link bài:

https://puoj.phenikaa-uni.edu.vn/problem/hnue_lis

Hãy lập trình giải bài toán LIS: Tìm dãy con tăng nghiêm ngặt dài nhất của mảng $a[1..n]$ (với $|a[i]| < 10^{18}$). (Chú ý: Dãy con tăng nghiêm ngặt là dãy con có các phần tử theo thứ tự $<$, chứ không phải $\leq$)

- Dữ liệu đầu vào: với dòng đầu là n . n dòng tiếp theo là các phần tử của mảng a
- Dữ liệu đầu ra: với dòng đầu là một số nguyên là độ dài của dãy con tăng dài nhất tìm được. Dòng tiếp theo đưa ra dãy con tăng dài nhất tìm được, mỗi số cách nhau tối thiểu một dấu cách.

Sample Input 0

```
8
-7 10 9 2 3 8 8 6
```

Sample Output 0

```
4
-7 2 3 6
```

HƯỚNG DẪN GIẢI

Tư tưởng tham lam: Bạn theo dõi các đầu mút của dãy con tăng. Để dễ quản lý, bạn tự hỏi: "Nếu dãy con tăng có độ dài là 1, kết thúc nhỏ nhất là mấy? Nếu dãy con tăng độ dài là 2, kết thúc nhỏ nhất là mấy?".

Gọi $tail[i]$ là **giá trị nhỏ nhất** của phần tử kết thúc trong số tất cả các dãy con tăng có độ dài đúng bằng i . Ví dụ, $tail[3] = 7$ nghĩa là dãy tăng dài độ dài 3 có giá trị kết thúc nhỏ nhất là 7.

- Ta xây dựng mảng $tail$ có tính chất tăng nghiêm ngặt như sau:
 $tail[1] < tail[2] < tail[3] < \dots < tail[len]$.
- **Tìm kiếm nhị phân với `lower_bound`:** Khi xét phần tử x , ta dùng tìm kiếm nhị phân trong đoạn từ 1 đến len để tìm vị trí idx đầu tiên sao cho $tail[idx] \geq x$.
 - Nếu $idx > len$: Nghĩa là x lớn hơn mọi đầu mút hiện có, ta mở rộng thêm một độ dài mới: $len++$, $tail[len] = x$.
 - Nếu $idx \leq len$: x tối ưu hóa được vị trí hiện tại, ta cập nhật đề: $tail[idx] = x$.
 - Ghi nhận vị trí trước của phần tử $a[i]$ trong dãy con tăng dài nhất kết thúc bởi $a[i]$ là $vitri[idx - 1]$ bằng cách gán: $Truoc[i] = vitri[idx - 1]$
- Truy vết: Để tìm ra dãy con tăng dài nhất sử dụng đệ quy truy vết sử dụng mảng $Truoc$.

Mô phỏng thuật toán:

<https://gemini.google.com/share/4ecd70fc7200?skid=6a8068ec-27a9-4351-8ef4-e3cf6e6506b5>

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

#define int long long

const int MAXN = 1000005;
int a[MAXN], tail[MAXN], vitri[MAXN], Truoc[MAXN];

void TruyVet(int i) {
    if (Truoc[i] == -1) {
        cout << a[i];
    }
    else {
```

```

        TruyVet(Truoc[i]);
        cout << " " << a[i];
    }
}

signed main() {
    int n;
    cin >> n;
    n = 0;

    ///vector<int> tail;

    vitri[0] = -1;

    cin >> a[1];
    n = 1;
    int i = 1;
    tail[1] = a[1];
    vitri[1] = 1;
    Truoc[1] = -1;
    int sz = 1, idx;

    int x;

    while (cin >> x) {
        i++;
        a[i] = x;
        n = i;

        idx = lower_bound(tail + 1, tail + sz + 1, a[i]) - tail;

        if (idx == sz + 1) {
            sz++;
            tail[sz] = a[i];
        }
        else tail[idx] = a[i];

        vitri[idx] = i;
        Truoc[i] = vitri[idx - 1];
    }

    cout << sz << endl;
    TruyVet(vitri[sz]);

    return 0;
}

```

TỐI ƯU HÓA CODE

Ta có thể loại bỏ mảng `Truoc` trong code ở trên (để giảm kích thước bộ nhớ). Sử dụng mảng `F` thay cho mảng `vitri` (ở đây, ta dùng mảng `F` để lưu độ dài).

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

#define N 1000005
#define int long long

int n, a[N], F[N];
int m, tail[N];

signed main() {
    cin >> n;
    // array a, length n
    for (int i = 1; i <= n; i++) cin >> a[i];

    // array b, length m, in increasing order
    int idx;
    for (int i = 1; i <= n; i++) {
        F[i] = idx = lower_bound(tail + 1, tail + m + 1, a[i]) - tail;
        m = max(m, F[i]);
        tail[idx] = a[i];
    }

    // m is also max(F)
    cout << m << endl;

    int Expected = m;
    vector<int> T;

    // add m elements from a into T
    for (int i = n; i >= 1; i--) {
        if (F[i] == Expected) {
            T.push_back(a[i]);
            Expected--;
        }
    }

    // output T in reversed order

    for (int i = T.size() - 1; i >= 0; i--)
        cout << T[i] << " ";
}
```