MỤC LỤC

Standard Template Library (STL) trong C++

Giáo viên: Nguyễn Như Thắng – THPT Chuyên Lào Cai

0. Mở đầu

Trang bị kiến thức về ngôn ngữ lập trình chưa bao giờ là vấn đề lớn trong tin học. Đương nhiên, khi chọn một ngôn ngữ lập trình nào đó làm công cụ cho học sinh sử dụng vào các cuộc thi lập trình chúng ta cần trang bị cho học sinh các hiểu biết, kĩ năng sau:

- Điểm mạnh, điểm yếu của ngôn ngữ lập trình cũng như hệ thống hỗ trợ.

- Các dịch vụ mà hệ thống lập trình cung cấp

- Tạo thói quen suy nghĩ và hành động phù hợp với ngôn ngữ lập trình và hệ thống lập trình.

- Cần biết càng sâu càng tốt các thư viện chuẩn hỗ trợ lập trình và biết khai thác chúng một cách linh hoạt, hiệu quả.

Trong xu thế mà việc sử dụng ngôn ngữ lập trình C++ đã trở nên phổ biến trong hầu hết các cuộc thi lập trình, thì việc nắm vững, nắm chắc ngôn ngữ lập trình C++ cũng như khai thác các công cụ có sẵn của nó là rất cần thiết. Do đó, ở chuyên đề này, tôi xin chia sẻ một số bài tập sử dụng thư viện STL của ngôn ngữ lập trình C++ mà khi áp dụng nó thì bài toán trở nên đơn giản.

1. Một số kiến thức cơ bản về STL.

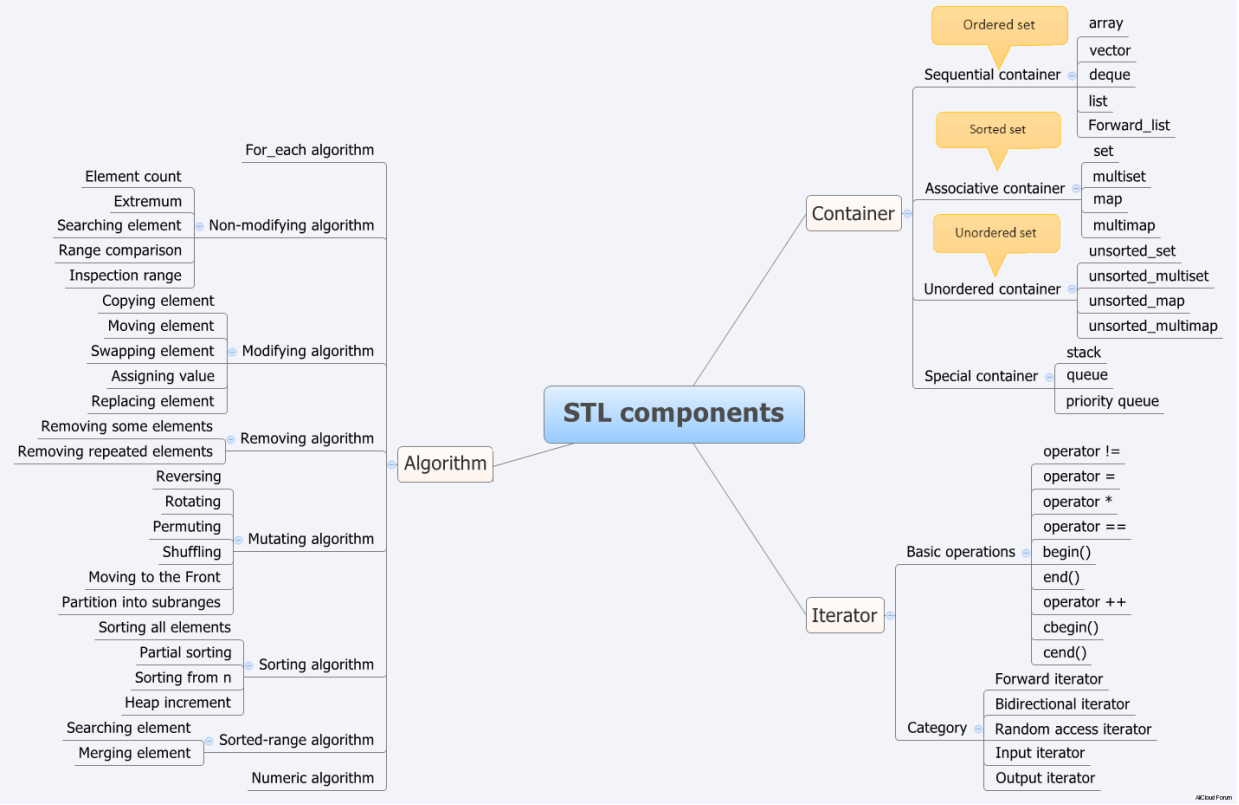
1.1. Khái quát về STL

C++ được đánh giá là ngôn ngữ mạnh vì tính mềm dẻo, gần gũi với ngôn ngữ máy. Ngoài ra, với khả năng lập trình theo mẫu (template), C++ đã khiến ngôn ngữ lập trình trở thành khái quát, không cụ thể và chi tiết như nhiều ngôn ngữ khác. Sức mạnh của C++ đến từ STL, viết tắt của Standard Template Library - một thư viện template cho C++ với những cấu trúc dữ liệu cũng như giải thuật được xây dựng tổng quát mà vẫn tận dụng được hiệu năng và tốc độ của C. Với khái niệm template, những người lập trình đã đề ra khái niệm lập trình khái lược (generic programming), C++ được cung cấp kèm với bộ thư viện chuẩn STL.

Bộ thư viện này thực hiện toàn bộ các công việc vào ra dữ liệu (iostream), quản lý mảng (vector), thực hiện hầu hết các tính năng của các cấu trúc dữ liệu cơ bản (stack, queue, map, set...). Ngoài ra, STL còn bao gồm các thuật toán cơ bản: tìm min, max, tính tổng, sắp xếp (với nhiều thuật toán khác nhau), thay thế các phần tử, tìm kiếm (tìm kiếm thường và tìm kiếm nhị phân), trộn. Toàn bộ các tính năng nêu trên đều được cung cấp dưới dạng template nên việc lập trình luôn thể hiện tính khái quát hóa cao. Nhờ vậy, STL làm cho ngôn ngữ C++ trở nên trong sáng hơn nhiều.

Để sử dụng STL bạn cần khai báo **using namespace std;** ngay sau khai báo các thư viện cần dùng.

Có thể tóm tắt về các thành phần của STL như hình vẽ sau:



Như hình vẽ trên ta có thể thấy STL gồm 3 nhóm thành phần chính: Container, Iterator và Algorithm.

Tôi chỉ đưa ra một số khái niệm cơ bản về chúng:

1.1.2. Containers

Một container là một đối tượng cụ thể lưu trữ một tập các đối tượng khác (các phần tử của nó). Nó được thực hiện như các lớp mẫu (class templates).

Container quản lý không gian lưu trữ cho các phần tử của nó và cung cấp các hàm thành viên (member function) để truy cập tới chúng, hoặc trực tiếp hoặc thông qua các biến lặp (iterator – giống như con trỏ).

Container xây dựng các cấu trúc thuờng sử dụng trong lập trình như: mảng động - dynamic arrays (vector), hàng đợi – queues (queue), hàng đợi ưu tiên – heaps (priority queue), danh sách kiên kết – linked list (list), cây – trees (set), mảng ánh xạ -associative arrays (map),...

Nhiều container chứa một số hàm thành viên giống nhau. Quyết định sử dụng loại container nào cho nhu cầu cụ thể nói chung không chỉ phụ thuộc vào các hàm được cung cấp mà còn phải dựa vào hiệu quả của các hàm thành viên của nó. Điều này đặc biệt đúng với container dãy (sequence containers), mà trong đó có sự khác nhau về độ phức tạp đối với các thao tác chèn/xóa phần tử hay truy cập vào phần tử

1.1.3. Iterator

Iterator là một đối tượng (giống như con trỏ) được sử dụng để trỏ đến địa chỉ ô nhớ chứa Container. Chúng ta có thể sử dụng Iterator để duyệt qua các phần tử trong Container.

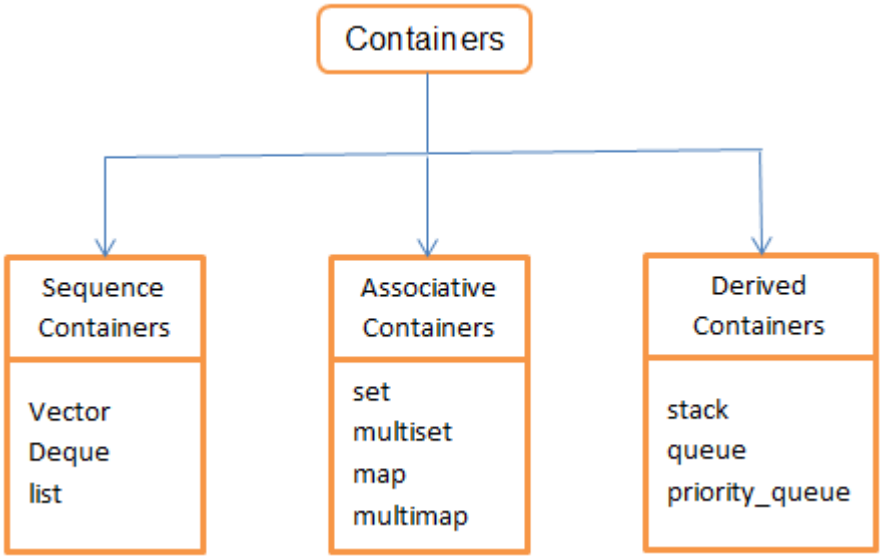
Iterator đóng vai trò cầu nối quan trọng giữa Algorithm với Container. Mỗi loại Container khác nhau chúng ta lại có một loại Iterator tương ứng.

1.1.4. Algorithm

Có rất nhiều Algorithm được xây dựng sẵn trong STL (chẳng hạn các thuật toán về sắp xếp, tìm kiếm, tìm min-max, ...). Các thuật toán được xây dựng sẵn và đã được xây dựng một cách tối ưu nhất có thể, có thể áp dụng đối với nhiều kiểu dữ liệu khác nhau.

1.2. Đối tượng của chuyên đề

Chuyên đề này tôi không giới thiệu chi tiết các thành phần của STL vì điều đó là không thể. Mà tôi chỉ tập trung vào chia sẻ một bài tập có áp dụng **Associative Containers** (là 1 trong 3 loại Containers) mà thôi. Trong trường hợp bài toán cần cấu trúc dữ liệu tương tự thì việc dùng Associative Containers sẽ là thuận lợi hơn rất nhiều so với việc chúng ta tự xây dựng, cài đặt một cấu trúc dữ liệu từ đầu.



1.2.1. Set (Tập hợp)

Set là một loại associative containers để lưu trữ các phần tử không bị trùng lặp (unique elements), và các phần tử này chính là các khóa (keys).

Khi duyệt set theo iterator từ begin đến end, các phần tử của set sẽ tăng dần theo phép toán so sánh.

Mặc định của set là sử dụng phép toán less, bạn cũng có thể viết lại hàm so sánh theo ý mình.

Set được thực hiện giống như cây tìm kiếm nhị phân (Binary search tree).

Khai báo:

```
#include <set>
set <int> s;
set <int, greater<int> > s;
```

Hoặc viết class so sánh theo ý mình:

```
struct cmp{
bool operator() (int a,int b) {return a<b;}
};
set <int,cmp > myset ;
```

Capacity:

- size : trả về kích thước hiện tại của set. ĐPT O(1)

- empty : true nếu set rỗng, và ngược lại. ĐPT O(1).

Modifiers:

- insert : Chèn phần tử vào set. ĐPT O(logN).

- erase : có 2 kiểu xóa: xóa theo iterator, hoặc là xóa theo khóa. ĐPT O(logN).

- clear : xóa tất cả set. ĐPT O(n).

- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:

- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy itarator trỏ về "end" của set. ĐPT O(logN).

- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của set. ĐPT O(logN).

- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của set.. ĐPT O(logN).

- count : trả về số lần xuất hiện của khóa trong container. Nhưng trong set, các phần tử chỉ xuất hiện một lần, nên hàm này có ý nghĩa là sẽ return 1 nếu khóa có trong container, và 0 nếu không có. ĐPT O(logN).

1.2.2. Mutiset (Tập hợp):

- Multiset giống như Set nhưng có thể chứa các khóa có giá trị giống nhau.

- Khai báo : giống như set.

- Các hàm thành viên:

Capacity:

- size : trả về kích thước hiện tại của multiset. ĐPT O(1)

- empty : true nếu multiset rỗng, và ngược lại. ĐPT O(1).

Chỉnh sửa:

- insert : Chèn phần tử vào set. ĐPT O(logN).

- erase :

xóa theo iterator ĐPT O(logN)

xóa theo khóa: xóa tất cả các phần tử bằng khóa trong multiset ĐPT: O(logN) + số phần tử bị xóa.

- clear : xóa tất cả set. ĐPT O(n).

- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:

- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy itarator trỏ về "end" của set. ĐPT O(logN). Dù trong multiset có nhiều phần tử bằng khóa thì nó cũng chỉ iterator đến một phần tử.

- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của set. ĐPT O(logN).

- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của set.. ĐPT O(logN).

- count : trả về số lần xuất hiện của khóa trong multiset. ĐPT O(logN) + số phần tử tìm được.

1.2.3. Map (Ánh xạ):

- Map là một loại associative container. Mỗi phần tử của map là sự kết hợp của khóa (key value) và ánh xạ của nó (mapped value). Cũng giống như set, trong map không chứa các khóa mang giá trị giống nhau.

- Trong map, các khóa được sử dụng để xác định giá trị các phần tử. Kiểu của khóa và ánh xạ có thể khác nhau.

- Và cũng giống như set, các phần tử trong map được sắp xếp theo một trình tự nào đó theo cách so sánh.

- Map được cài đặt bằng red-black tree (cây đỏ đen) – một loại cây tìm kiếm nhị phân tự cân bằng. Mỗi phần tử của map lại được cài đặt theo kiểu pair (xem thêm ở thư viện utility).

Khai báo:

```
#include <map>
...
map <kiểu_dữ_liệu_1,kiểu_dữ_liệu_2>
```

```
// kiểu dữ liệu 1 là khóa, kiểu dữ liệu 2 là giá trị của khóa.
```

Sử dụng class so sánh:

Dạng 1:

```
struct cmp{
bool operator() (char a,char b) {return a<b;}
};
.....
map <char,int,cmp> m;
```

- Truy cập đến giá trị của các phần tử trong map khi sử dụng iterator:

Ví dụ ta đang có một iterator là it khai báo cho map thì:

(*it).first; // Lấy giá trị của khóa, kiểu_dữ_liệu_1

(*it).second; // Lấy giá trị của giá trị của khóa, kiểu_dữ_liệu_2

(*it) // Lấy giá trị của phần tử mà iterator đang trỏ đến, kiểu pair

it->first; // giống như (*it).first

it->second; // giống như (*it).second

Capacity:

- size : trả về kích thước hiện tại của map. ĐPT O(1)

- empty : true nếu map rỗng, và ngược lại. ĐPT O(1).

Truy cập tới phần tử:

- operator [khóa]: Nếu khóa đã có trong map, thì hàm này sẽ trả về giá trị mà khóa ánh xạ đến. Ngược lại, nếu khóa chưa có trong map, thì khi gọi [] nó sẽ thêm vào map khóa đó. ĐPT O(logN)

Chỉnh sửa

- insert : Chèn phần tử vào map. Chú ý: phần tử chèn vào phải ở kiểu "pair". ĐPT O(logN).

- erase :

    - xóa theo iterator ĐPT O(logN)

    - xóa theo khóa: xóa khóa trong map. ĐPT: O(logN).

- clear : xóa tất cả set. ĐPT O(n).

- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:

- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy iterator trỏ về "end" của map. ĐPT O(logN).

- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà lớn hơn hoặc bằng khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của map. ĐPT O(logN).

- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của map. ĐPT O(logN).

- count : trả về số lần xuất hiện của khóa trong multiset. ĐPT O(logN).

2. Bài tập áp dụng

Các bài tập trong Chuyên đề này đã được tác giả cho học sinh làm và test trong quá trình dạy cho học sinh đội tuyển HSG. Áp dụng tốt đối với học sinh lớp 10, học sinh bắt đầu có tư duy căn bản về lập trình.

Nhiều bài toán khi áp dụng STL việc code trở nên đơn giản hơn rất nhiều, chương trình ngắn gọn, dễ hiểu hơn. Trong một số bài chúng ta có thể dùng SET thay thế cấu trúc dữ liệu Segment tree hoặc Binary Index tree.

Test của mỗi bài đều được sinh tự động bằng trình sinh test riêng, sinh ngẫu nhiên theo điều kiện được nêu ra trong đề bài.

2.1. Bài 1. Xếp hàng ưu tiên

Nguồn bài tập: https://vn.spoj.com/problems/MSE07B/

2.1.1. Đề bài: **MSE07B**

Ngân hàng BIG-Bank mở một chi nhánh ở Bucharest và được trang bị một máy tính hiện đại với các công nghệ mới nhập, C2#,VC3+... chỉ chuối mỗi cái là không ai biết lập trình. Họ cần một phần mềm mô tả hoạt động của ngân hàng như sau: mỗi khách hàng có một mã số là số nguyên K, và khi đến ngân hàng giao dịch, họ sẽ nhận được 1 số P là thứ tự ưu tiên của họ. Các thao tác chính như sau:

(0) Kết thúc phục vụ.

(1 K P) Thêm khách hàng K vào hàng đợi với độ ưu tiên P.

(2) Phục vụ người có độ ưu tiên cao nhất và xóa khỏi danh sách hàng đợi.

(3) Phục vụ người có độ ưu tiên thấp nhất và xóa khỏi danh sách hàng đợi.

Tất nhiên là họ cần bạn giúp rồi.

Input: Mỗi dòng của input là 1 yêu cầu, và chỉ yêu cầu cuối cùng mới có giá trị là 0. Giả thiết là khi có yêu cầu 1 thì không có khách hàng nào khác có độ ưu tiên là P ($K \leq 10^6; P \leq 10^7$, tổng số yêu cầu mỗi loại không quá 10^5). Một khách hàng có thể yêu cầu phục vụ nhiều lần và với các độ ưu tiên khác nhau.

Output: Với mỗi yêu cầu 2 hoặc 3, in ra trên 1 dòng mã số của khách hàng được phục vụ tương ứng. Nếu có yêu cầu mà hàng đợi rỗng, in ra số 0.

Example:

MSE07B.INP	MSE07B.OUT
2 1 20 14 1 30 3 2 1 10 99 3 2 2 0	0 20 30 10 0

2.1.2. Hướng dẫn giải thuật

Nhận xét: Nếu danh sách khách hàng được đưa vào mảng không được sắp xếp theo độ ưu tiên P, thì mỗi khi có thao tác loại 2 hoặc 3 thì ta lại phải đi tìm min, max.

Mỗi thao tác loại 2 hoặc 3 đều có độ phức tạp thuật toán phụ thuộc tuyến tính vào số lượng khách hàng. Như vậy chương trình không chạy được trong thời gian cho phép.

Do vậy, danh sách khách hàng cần được sắp xếp theo độ ưu tiên tăng (hoặc giảm). Các thao loại 1, 2, 3 là xen kẽ nhau nên danh sách khách hàng liên tục biến động, việc quản lý cũng tương đối phức tạp.

Tuy nhiên, nếu sử dụng kiểu dữ liệu SET thì vấn đề trở nên đơn giản.

2.1.3. Chương trình minh họa

```
#include<bits/stdc++.h>
#define ii pair<int,int>
using namespace std;
set<ii> s;
set<ii>::iterator it; //iterator de tro den set s
int x,k,p;
int main() {
    freopen("MSE07B.inp","r",stdin);
    freopen("MSE07B.out","w",stdout);
    cin>>x;
    while (x!=0) {
        if (x==1) { //thao tac loai 1
            cin>>k>>p;
            s.insert({p,k});
        }
```

```
            if (x==2) { //thao tac loai 2
                if (!s.empty()) {
                    it=s.end();
                    --it;
                    cout<<it->second<<endl;
                    s.erase(it);
                } else
                    cout<<0<<endl;
            }
            if (x==3) { //thao tac loai 3
                if (!s.empty()) {
                    it=s.begin();
                    cout<<it->second<<endl;
                    s.erase(it);
                } else
                    cout<<0<<endl;
            }
            cin>>x;
        }
    }
```

2.1.4. Cảm nhận

Việc sử dụng SET khiến mọi việc trở lên đơn giản và độ phức tạp thuật toán là O(NlogN) với N là max{thao tác loại 1,2,3}.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.2. Bài 2. KRYP6

2.2.1. Đề bài **KRYP6**

Thầy giáo đưa cho Quang một mảng A gồm N phần tử và yêu cầu Quang với mỗi A[i] hãy tìm A[j] lớn nhất sao cho $j <$ i và $A[j] < A[i]$. Bạn hãy giúp bạn ấy nhé!

Input:

Dòng đầu tiên chứa N – số phần tử của mảng A.

Dòng thứ 2 chứa N số nguyên là các phần tử của mảng A. Giữa 2 số cách nhau 1 dấu cách.

Output: In ra N số là đáp án cần tìm. Trường hợp không có $A[j]$ thỏa mãn in ra -1.

Điều kiện: $N \leq 200000$; $1 \leq A[i] \leq 10^{15}$.

Example:

KRYP6.INP	KRYP6.OUT
5 1 2 3 5 4	-1 1 2 3 3

2.2.2. Hướng dẫn giải thuật

Ta nhận thấy không tồn tại số cần tìm ứng với A[1] nên ta đầu tiên ta sẽ in ra -1.

Cho lần lượt A[i] vào trong set (từ 1 đến N - 1). Sau mỗi lần cho A[i] và trong set, ta chặt nhị phân vị trí đầu tiên có giá trị lớn hơn hoặc bằng A[i+1] trên set.

Gọi vị trí tìm được là j. Nếu $j = 1$, in ra -1. Nếu j khác 1, in ra giá trị của phần tử ở vị trí $j - 1$ trong set.

2.2.3. Chương trình tham khảo

```
#include <bits/stdc++.h>
#define maxn 100005
using namespace std;
set<long long> s;
set<long long> ::iterator it;
int n;
long long a[200005];
main() {
    freopen("KRYP6.inp", "r", stdin);
    freopen("KRYP6.out", "w", stdout);
    cin >> n;
    for(int i=0; i<n; ++i)
        cin >> a[i];
    cout<< -1 << endl;
    s.insert(a[0]);
    for(int i=1; i<n; ++i) {
        long long ans=-1;
        it=s.lower_bound(a[i]);
        if(it != s.begin()) {
            it--;
            ans=*it;
        }
        cout<<ans<<endl;
```

```
            s.insert(a[i]);
        }
    }
```

2.2.4. Cảm nhận

Việc sử dụng SET khiến mọi việc trở lên đơn giản, khó bị sai và độ phức tạp thuật toán là O(NlogN).

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.3. Bài 3. Khôi phục lại mảng

Nguồn tham khảo: https://www.codechef.com/problems/ANUMLA

2.3.1. Đề bài: **ANUMLA**

Thầy giáo dạy Toán ra bài tập cuối tuần cho BT là: có N số nguyên dương và yêu cầu BT liệt kê tất cả các tập con của tập các số nguyên này (dễ thấy sẽ có 2^N tập con như vậy). Với mỗi tập con BT cần phải tính tổng các phần tử của nó và liệt kê tất cả các kết quả nộp cho thầy.

BT đã hoàn thành nhiệm vụ một cách nhanh chóng. Nhưng thật không may, cậu ta đánh mất tờ giấy ghi đề bài của thầy và nếu không có tờ đề này thì không lấy gì chứng minh được rằng BT đã làm đúng (!!!).

Bạn hãy viết một chương trình giúp BT, dựa trên 2 tổng mà BT đã lập được khôi phục lại N số nguyên dương trong đề bài của thầy.

Input: Dòng đầu tiên chứa số nguyên dương T ($T \leq 50$) là số bộ dữ liệu. Tiếp theo là T nhóm dòng, mỗi nhóm dòng mô tả một bộ dữ liệu với cấu trúc:

- Dòng đầu tiên chứa số nguyên dương ($1 \leq N \leq 15$)

- Dòng thứ hai chứa 2^N số nguyên là tổng của các tập con mà BT ghi được. Các số nguyên này có giá trị không vượt quá 10^9.

Ouptut:

Với mỗi bộ dữ liệu in ra trên một dòng số tìm được theo giá trị không giảm. Hai số trên một dòng ghi cách nhau một dấu trống.

Example:

ANUMLA.INP	ANUMLA.OUT
2 1 0 10 2 0 1 1 2	10 1 1

2.3.2. Hướng dẫn giải thuật

Đây là bài tập về việc sử dụng mutiset trong C++ (do có thể có nhiều tập con có tổng bằng nhau). Tư tưởng là với mỗi testcase ta sắp xếp lại tổng theo thứ tự tăng dần, rồi lặp N lần, mỗi lần lấy phần tử nhỏ nhất trong các tổng còn khả dụng làm phần tử tiếp theo. Xóa tất cả các tổng liên quan đến phần tử này... *Cụ thể:* giả sử tại bước thứ i ta đã chọn được $a[i]$ thì ta cần xóa tất cả các tổng được tạo thành từ tổ hợp của $a[i]$ với các tổ hợp khác rỗng của $\{a[1], a[2], \dots a[i-1]\}$. Phần tử bé nhất còn lại của mảng tổng nhập vào ban đầu sẽ là phần tử $a[i+1]$ của bước thứ $i+1$.

2.3.3. Chương trình tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define N 16
multiset <int> s; // s[] chua tong da xuat hien
int a[N];// a[] chua ket qua la mang can khoi phuc
int b[1<<N];// b chua tong cac tap con, input cua bai toan
int c[1<<N];// c chua tong co the cua cac to hop den trang thai dang xet
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL),cout.tie(NULL);
    freopen("ANUMLA.inp","r",stdin);
    freopen("ANUMLA.out","w",stdout);
    int t;  //t la so luong testcase
    cin >> t;
    while(t--) {
        int n;
        cin >> n;
        int m = 1 << n;
        for(int i=0; i<m; i++) {
            cin >> b[i];
        }
        sort(b, b+m);

        int ptr = 0, fptr = 0;
        for(int i=1; i<m; i++) {
            int expected = -1;
            if(!s.empty()) {
                expected = *s.begin(); //gia tri nho nhat hien thoi cua s
            }
            if(b[i] == expected) { // b[i] da xuat hien la to hop cua 1 tap con.
                s.erase(s.begin()); //xoa phan tu nho nhat
            } else {
                a[fptr] = b[i];
```

```
                    //tim thay gia tri b[i], dua vao mang ket qua a[]
                int tptr = ptr;
                for(int j=0; j<tptr; j++) {
                    c[ptr] = c[j] + a[fptr];
                    s.insert(c[ptr]);
                    //cap nhat cac to hop co the cua trang thai i voi
                    ptr++;   // voi cac to hop truoc do
                }
                c[ptr++] = a[fptr];
                fptr++;
            }
        }
        for(int i=0; i<fptr; i++)  //in ket qua
            cout<<a[i]<<" ";
        cout<<endl;
    }
}
```

2.3.4. Cảm nhận

Nếu không sử dụng MULTISET thì có thể sử dụng HEAP để thay thế. Hoặc có thể dụng mảng đánh dấu các tổng của tổ hợp đã xuất hiện tuy nhiên bộ nhớ sử dụng sẽ khá lớn.

Độ phức tạp: $O(T.2^N.\log(2^N))$.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.4. Bài 4. Ronaldo chuyển sang Juventus

2.4.1. Đề bài - **CR7JUVE**

CR7 là cầu thủ bóng đá, nhân dịp kì nghỉ đông đầu tiên khi chuyển sang CLB bóng đá Juventus, CR7 đã mời rất nhiều cầu thủ bóng đá khác đến dự tiệc đánh dấu sự thay đổi lớn về môi trường làm làm việc của mình. Bạn bè đến rất đông nhưng được chia làm 2 tốp lớn, tốp đầu tiên có N bạn đã đến trước, tốp thứ 2 có M bạn đến sau. Vì mức lương các cầu thủ bóng đá luôn được giấu kín tuy nhiên giá trị chuyển nhượng cầu thủ thứ i là A_i thì lại được công khai. Các bạn đến sau muốn giao lưu với các bạn đến trước, tuy nhiên họ lại hay mất tự tin nếu bạn đến trước có giá trị chuyển nhượng khác với mình. Hỏi trong M bạn đến sau, bao nhiêu bạn có thể tìm được bạn phù hợp với mình.

Input:

- Dòng thứ 1 chứa 2 số N, M (số bạn đến trước, số bạn đến sau).

- Dòng tiếp theo chứa N+M số là giá trị chuyển nhượng của lần lượt N bạn đến trước sau đó là M bạn đến sau.

Output: Một số duy nhất là số cầu thủ đến sau tìm được bạn phù hợp với mình, biết $0 < N, M \leq 10^5; 0 \leq A_i \leq 10^{12}$.

Example:

CR7JUVE.INP	CR7JUVE.OUT	Giải thích
3 5 3 2 9 11 2 5 3 8	2	Có 3 bạn đến trước. Trong 5 bạn đến sau thì có 2 bạn có thể tìm được bạn phù hợp với mình.

2.4.2. Hướng dẫn giải thuật

Với bài toán này có nhiều cách để giải quyết

Cách 1: Với mỗi bạn đến sau, tìm trong N bạn đến trước, nếu ai bằng mình thì tăng biến đếm lên 1. Độ phức tạp: O(M.N) – không đảm bảo.

Cách 2: Sắp xếp tăng dần của 2 đoạn. Sau đó với mỗi bạn đến sau có thể tìm kiếm nhị phân xem có bạn nào đến trước bằng mình không, hoặc có thể ngược lại. Độ phức tạp $O(KlogK)$ với $K = \max(M, N)$.

Cách 3: Dùng kĩ thuật Two pointers sau khi sắp xếp 2 mảng. Độ phức tạp của đoạn đếm chỉ là $O(N)$, tuy vậy sắp xếp thì vẫn $O(NlogN)$.

Cách 4: Có thể dùng mảng đánh dấu, dùng mảng trước đánh dấu mảng sau hoặc ngược lại. Độ phức tạp O(M+N), tuy nhiên chỉ thực hiện với A_i cỡ 10^{11} và dùng mảng bool để đánh dấu.

Cách 5: Giống cách 4, nhưng dùng MAP để đánh dấu. Không phụ thuộc vào giá trị của A_i nữa. Độ phức tạp $O(KlogK)$ với $K = \max(M, N)$.

2.4.3. Chương trình tham khảo

```
#include<bits/stdc++.h>
using namespace std;
int n,m,res;
map<long long,bool> map1;
int main() {
    int ai;
    freopen("CR7JUVE.inp","r",stdin);
   freopen("CR7JUVE.out","w",stdout);
    cin>>n>>m;
    for(int i=1; i<=n; i++) {
        cin>>ai;
        map1[ai]=true;
    }
    for(int i=1; i<=m; i++) {
        cin>>ai;
        if (map1[ai]==true)
```

```
            res++;
        }
        cout<<res;
    }
```

2.4.4. Cảm nhận

Sử dụng MAP khiến chương trình khá ngắn gọn, cách code giống như kĩ thuật đánh dấu, tuy nhiên độ phức tạp thì lớn hơn. Đảm bảo yêu cầu. Tuy nhiên đã khắc phục nhược điểm của đánh dấu về sử dụng bộ nhớ.

Có thể mở rộng bài này cho thú vị hơn ta có yêu cầu là tìm vị trí trong đoạn M+N để có chọn được nhiều bạn phù hợp nhất.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.5. Bài 5 – Công cụ sắp xếp kì lạ

2.5.1. Đề bài: **SORTTOOL**

Những bài toán về sắp xếp tăng dần hay giảm dần theo giá trị của khóa cho trước đã trở nên quá đỗi quen thuộc với các bạn học sinh, để đỡ nhàm chán, thầy giáo giao cho học sinh bài tập xây dựng công cụ sắp xếp theo yêu cầu:

Cho dãy số có $N(1 \leq N \leq 10^5)$ số nguyên $a_1, a_2, \dots a_N$ $(|a_i| \leq 10^9)$, hãy sắp xếp các số trên theo thứ tự giảm dần theo tần số xuất hiện, nếu có những số có cùng tần số xuất hiện thì số nào được xuất hiện trước thì sẽ xếp trước.

Input: Dòng đầu là số N; dòng tiếp theo chứa N số $a_1, a_2, \dots a_N$.

Output: Dãy được sắp xếp theo yêu cầu đã đưa ra.

Example:

SORTTOOL.INP	**SORTTOOL.OUT**
7 2 3 3 3 2 1 2	2 2 2 3 3 3 1
4 2 1 2 2	2 2 2 1

2.5.2. Hướng dẫn giải thuật

Bài toán trên có nhiều cách để xử lý

Cách 1: Không sử dụng STL. Sử dụng mảng để đánh dấu vị trí xuất hiện đầu tiên của một số, mảng đếm tần số xuất hiện, mảng lưu giá trị. Sắp xếp giảm dần theo tần số xuất hiện, nếu có cùng tần số xuất hiện thì số nào có lần xuất hiện đầu tiên bé hơn thì xếp trước. Độ phức tạp: $O(NlogN)$.

Tuy nhiên với điều kiện $0 \leq |a_i| \leq 10^9$ thì không thể thực hiện trực tiếp mà lại cần phải rời rạc hóa để đánh dấu. Nhìn chung là thao tác khá phức tạp.

Cách 2: Ý tưởng tương tự như trên, nhưng sử dụng STL.

Độ phức tạp: $O(NlogN)$.

2.5.3. Chương trình tham khảo

```
#include<bits/stdc++.h>
using namespace std;
struct num {
    int val, count, index;
    num(int v, int c, int i) {
        val=v; //luu gia tri
        count=c; //luu tan so
        index=i; //luu chi so
    }
};
bool operator < (num a, num b) {
    //dinh nghia lai phep so sanh <, phuc vu sort()
    if(a.count==b.count) {
        return a.index<b.index;
    } else
        return a.count>b.count;
}
int main () {
    int n;
    freopen("SORTTOOL.inp","r",stdin);
    freopen("SORTTOOL.out","w",stdout);
    map<int, pair<int,int>> m;
    scanf("%d",&n);
    int temp;
    for(int i=0; i<n; i++) {
        scanf("%d",&temp);
        map <int,pair<int,int>>::iterator it = m.find(temp);
        if(it==m.end()) {
            m[temp]=make_pair(i,1);
        } else {
            it->second.second++;
        }
    }
    vector <num> v;
   for(map<int,pair<int,int>>::iterator it=m.begin(); it!=m.end(); it++){
        v.push_back(num(it->first,it->second.second,it->second.first));
    }
```

```
    sort(v.begin(), v.end());
    for(vector<num>::iterator it=v.begin(); it!=v.end(); it++) {
        for(int i=0; i<it->count; i++) {
            printf("%d ",it->val);
        }
    }
    printf("\n");
}
```

2.5.4. Cảm nhận

Việc sử dụng STL rõ ràng tạo ra sự đơn giản hơn trong việc xử lý bài toán này. Nắm chắc các công cụ trong thư viện này sẽ giúp học sinh dễ dàng tổ chức dữ liệu và triển khai kĩ thuật lập trình để giải quyết bài toán hơn.

Độ phức tạp: $O(NlogN)$.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.6. Bài 6- Chỗ ngồi trong nhà hát

2.6.1. Đề bài - **SEATS**

Trong một nhà hát có N chỗ ngồi, chúng được xếp thành một hàng dài đánh số từ 1 đến N và từ trái qua phải. Ghế số 1 gần khán đài nhất và ghế số N là ghế xa nhất. Khi thấy phía trong nhà hát còn ghế trống thì nhân viên bán vé mới bán vé cho khán giả vào. Ban đầu tất cả các ghế đều trống, khách đầu tiên vào chắc chắn sẽ ngồi ghế trên cùng (ghế số 1). Mỗi khi có khán giả vào thêm, họ luôn chọn chỗ sao cho khoảng cách từ họ đến người gần nhất là xa nhất có thể. Nếu có nhiều chỗ như vậy thì họ chọn ghế có số thứ tự nhỏ nhất.

Trong suốt buổi hòa nhạc, nhân viên bán vé thấy có Q người ra và vào. Hỏi số ghế mỗi người vào là sau là số nào theo cách chọn chỗ như trên.

Input:

Dòng 1 là 2 số N, Q (N là số ghế; Q là số người ra, vào).

Q dòng tiếp theo mô tả người ra, người vào:

Nếu là (1) thì có người vào và cần tìm số ghế mà người đó chọn.

Nếu là (2, i) thì là người thứ i đi ra khỏi nhà hát.

Biết rằng $1 \leq N \leq 10^{18}; 1 \leq Q \leq 10^5$.

Output:

Gồm nhiều số tương ứng với số ghế của những người vào sau đã chọn

Example:

SEATS.INP	SEAT.OUT	Giải thích
2 7	1	Có 2 ghế và 7 lượt vào ra.
1	2	*Người 1 vào, chọn ghế 1.*
1	1	*Người 2 vào, chọn ghế 2.*
2 1	1	Người 1 ra, ghế 1 trống.
1		*Người 3 vào, chọn ghế 1.*
2 2		Người 2 ra, ghế 2 trống.
2 3		Người 3 ra, ghế 1 trống
1		*Người 4 vào chọn ghế 1.*

2.6.2. Hướng dẫn giải thuật

Gọi các dãy ghế liên tục còn trống chưa có người ngồi là đoạn. Các đoạn được quản lý bởi set S gồm các thông tin sau: chỉ số đầu, chỉ số cuối, khoảng cách được tạo ra khi có người vào, số ghế được chọn.

Xét đoạn được xác định bởi 2 chỉ số ghế đầu và cuối. Gọi đoạn dài nhất là đoạn $[i, j]$, đoạn này có $j - i + 1$ ghế trống. Khi đó một người mới vào thì họ sẽ chọn ghế $\boldsymbol{k}$ trong đoạn này. Theo cách chọn đã đưa ra, thì vị trí được chọn trong đoạn đó sẽ được xác định như sau:

$$\boldsymbol{k} = A[i,j] = \begin{cases} 1 \; nếu \; i = 1 \\ N \; nếu \; j = N \\ \left\lfloor \frac{i+j}{2} \right\rfloor \; còn \; lại \end{cases}$$

Khi vị trí $\boldsymbol{k}$ được chọn thì khoảng cách $D[i,j]$ mới được tạo ra là:

$$\boldsymbol{des} = D[i,j] = \begin{cases} j & nếu \; i = 1 \\ A[i,j] - i + 1 & nếu \; i \neq 1 \end{cases}$$

Tình huống 1: Khi vị trí $A[i,j] = k$ có người ngồi vào thì đoạn $[i,j]$ cần được loại bỏ ra khỏi S và được thêm tối đa 2 đoạn có độ dài $D[i,j]$ là $[i, k-1]$ và $[k+1, j]$ vào S.

Hình sau mô tả việc thêm một người mới vào đoạn $[i,j]$.

```
              i                                     j
              .-------------------------------------.
   ...#  #  #  i  .  .  .  .  .  k  .  .  .  .  .  .  j  #  #  #...
              '-------------'     '----------------'
              i             k-1   k+1               j
```

Với các vị trí # là vị trí có người ngồi.

Tình huống 2: Khi có người rời đi. Thì cần tìm vị trí $\boldsymbol{k}$ của họ. Từ đó xây dựng lại đoạn $[i,j]$ tương ứng.

2.6.3. Chương trình tham khảo

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long
#define MAX 100005
LL n;
struct data { //thong tin ve doan ghe trong
    LL st, ed, des, seat; //start, end, destination, seat
    bool make(LL a, LL b) {
        st = a, ed = b;
        if (a>b)
            return false;
        if(st == 1)
            seat = 1, des = (ed+1)-seat;
        else if(ed == n)
            seat = n, des = seat - (st-1);
        else {
            seat = (st+ed)/2;
            des = seat - (st-1);
        }
        return true;
    }
};
bool operator < (data a, data b) {
    if(a.des == b.des)
        return a.seat < b.seat;
    return a.des > b.des;
}
set<LL>filled;//tap hop vi tri da co nguoi
set<data>S;//chua thong tin ve cac doan ghe trong
LL pos[MAX];
LL add(int p) {  //tinh huong nguoi vao
    assert(S.size());
    auto it = S.begin();
    filled.insert(it->seat);
    LL a = it->st;
    LL b = it->seat-1;
    data tmp;
    if(tmp.make(a,b))
        S.insert(tmp); //them doan truoc
```

```
    a = it->seat+1;
    b = it->ed;
    if(tmp.make(a,b)) {
        S.insert(tmp); //them doan sau
    }
    pos[p] = it->seat;
    S.erase(*it);
    return pos[p];
}
void del(int p) { //tinh huong nguoi ra
    LL seat = pos[p];
    auto it = filled.find(seat);
    it--;
    LL a = *it+1;
    LL b = seat-1;
    LL myA = a;
    data tmp;
    if(tmp.make(a,b))
        S.erase(tmp);
    it++;
    it++;
    a = seat+1;
    b = *it-1;
    LL myB = b;
    if(tmp.make(a,b))
        S.erase(tmp);
    if(tmp.make(myA, myB))
        S.insert(tmp);
    filled.erase(seat);
}
int q,cnt,type,idx;
int main() {
    freopen("SEATS.INP","r",stdin);
    freopen("SEATS.OUT","w",stdout);
    cin>>n;
    cin>>q;
    filled.insert(0);
    filled.insert(n+1);
    data tmp;
    tmp.make(1, n);
    S.insert(tmp);
```

```
    while(q--) {
        cin>>type;
        if(type == 1) {
            cnt++;
            LL ans = add(cnt);
            cout<<ans<<endl;
        } else {
            cin>>idx;
            del(idx);
        }
    }
}
```

2.6.4. Cảm nhận

Đây là bài toán sử dụng 2 set để lưu trữ dữ liệu phục vụ giải bài toán. Việc sử dụng STL giúp quá trình xử lý đơn giản hơn. Ngoài ra ta có thể kết hợp map, set để giải bài này với cùng ý tưởng về thuật toán như trên.

Độ phức tạp: $O(QlogQ)$.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.7. Bài 7. Đoạn con tổng 0

2.7.1. Đề bài: **SUMSEQ0**

Cho một dãy số nguyên gồm N phần tử: $a_1, a_2, \dots a_n$. Một đoạn con liên tiếp của dãy A có điểm đầu L, điểm cuối R với $(L \leq R)$ là tập hợp tất cả các phần tử a_i với $(L \leq i \leq R)$. Đếm số đoạn con có tổng tất cả các phần tử bằng 0.

Input: Dòng đầu là số tự nhiên N.

Dòng thứ 2 là N số nguyên $a_1, a_2, \dots a_N$ ($|a_i| \leq 10^9$).

Output: Ghi số lượng đoạn con tìm được.

Example:

SUMSEQ0.INP	SUMSEQ0.OUT	Giải thích
5 2 1 -1 -2 0	4	Có 4 đoạn có tổng bằng 0 là: [2,3], [1,4], [1,5], [5,5]

2.7.2. Hướng dẫn giải thuật

Cách 1: Đây là bài toán không còn xa lạ với học sinh lớp 10. Cách tiếp cận bài toán đơn giản nhất là sử dụng kĩ thuật cộng dồn, đánh dấu.

Sử dụng mảng $S[\ ]$ để đánh dấu số lần xuất hiện của tổng cộng dồn.

Chỉ dùng một vòng lặp, xét đến phần tử a_i, tổng cộng dồn đoạn $[1, i]$ là sum_i, khi đó nếu kết quả của bài toán được cộng thêm một lượng $S[sum_i]$ như sau:

```
for(int i=1; i<=n; i++) {
        cin>>a;
        sum=sum+a;
        res=res+s[sum];
        s[sum]++;
    }
```

Độ phức tạp là: $O(N)$.

Tuy nhiên bài toán có giới hạn $1 \leq a_i \leq 10^9$ nên việc sử dụng mảng S[ ] như vậy là bất khả thi.

Để cải tiến cho phù hợp thì ta duy trì một mảng lưu tổng, một mảng lưu tần số tương ứng, mảng tổng được sắp xếp, mỗi khi nhận được 1 tổng sum_i thì tìm kiếm nhị phân để có được tần số đã xuất hiện của nó. Làm tiếp các bước cộng dồn như trên thì ta hoàn toàn thu được kết quả của bài toán. Độ phức tạp khi đó là: $O(N.logN)$.

Cách 2: Thay thế kiểu dữ liệu mảng của cách 1 bằng kiểu map<> ta có thể dễ dàng xử lý bài toán trên.

2.7.3. Chương trình minh họa khá đơn giản như sau:

```
#include<bits/stdc++.h>
using namespace std;
long long n,a,res,sum;
map<long long, int> s;
int main() {
    freopen("SUMSEQ0.inp","r",stdin);
    freopen("SUMSEQ0.out","w",stdout);
    cin>>n;
    s[0]=1;
    for(int i=1; i<=n; i++) {
        cin>>a;
        sum=sum+a;
        res=res+s[sum];
        s[sum]++;
    }
    cout<<res;
}
```

2.7.4. Cảm nhận

Dùng dữ liệu kiểu map<> sẽ hỗ trợ rất nhiều trong giải các bài toán cần đến thao tác tìm kiếm nhị phân trên một mảng thông thường.

Độ phức tạp của thuật toán: $O(N.logN)$.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.8. Bài 8. Không segment tree

2.8.1. Đề bài: **NOST**

Thầy giáo có điểm của N học sinh: $a_1, a_2, \dots a_N$ $(1 \le a_i \le 50)$. Học sinh thứ i có điểm là a_i. Thầy giáo giao cho bạn 2 loại câu hỏi như sau:

Loại 1: dạng $(1\ D\ M)$ là yêu cầu bạn cập nhật lại bạn thứ D với điểm số là M.

Loại 2: dạng $(2\ L\ R)$ là hỏi giữa đoạn $[L, R]$ thì 2 điểm bằng nhau nào có khoảng cách xa nhất, nếu không có 2 điểm bằng nhau thì khoảng cách bằng 0 và in ra điểm nhỏ nhất. Nếu có khoảng cách bằng nhau thì in ra điểm nhỏ nhất.

Input:

Dòng 1 chứa 2 số N, Q là số học sinh và số câu hỏi $(1 \le N, Q \le 10^5)$.

Dòng tiếp theo là điểm ban đầu của N học sinh.

Q dòng tiếp theo cho tương ứng Q câu hỏi theo định dạng như trên.

Output:

Với mỗi câu hỏi loại 2 thì in ra 1 số là kết quả tương ứng.

Example:

NOST.INP	NOST.OUT	Giải thích
5 6 12 13 13 12 1 **2 1 3** **2 1 2** **2 1 5** 1 3 5 1 1 5 **2 1 3**	13 12 12 5	Có 5 bạn học sinh, 6 câu hỏi. Câu (2,1,3) trong đoạn [1,3] in ra 13 vì khoảng cách lớn nhất (=1). Câu (2,1,2) trong đoạn [1,2] không có điểm nào lặp lại. Câu (2,1,5) in ra 12 vì có khoảng cách xa nhất. Sau 2 câu cập nhật thì câu cuối cùng in ra 5.

2.8.2. Hướng dẫn giải thuật

Cách 1: Duyệt trâu

Ngay lập tức khi đọc đề thì học sinh có thể nghĩ đến duyệt trâu. Dùng mảng để lưu điểm của học sinh. Mỗi thao tác loại 1 thì cập nhật mất $O(1)$. Tuy nhiên mỗi thao tác loại 2 thì phải xử lý tìm khoảng cách của từng loại điểm sau đó tìm giá trị lớn nhất trong số 50 loại điểm. Độ phức tạp của thao tác loại 2 là $O(N)$.

Độ phức tạp: $O(N.Q)$, với $1 \le N, Q \le 10^5$ thì không thể giải quyết trọn vẹn bài toán.

Cách 2: Dùng cấu trúc Segment tree.

Xây dựng 50 cây Segment tree quản lý tương ứng với 50 loại điểm. Xây dựng cây mất $O(50.N.logN)$ Mỗi truy vấn loại 1 mất $O(logN)$. Mỗi truy vấn loại 2 mất $O(50.logN)$. Tổng chung độ phức tạp cỡ $O(50.(Q+N).logN)$.

Tuy nhiên việc xây dựng cây và truy vấn sẽ có đôi chút khó khăn với học sinh lớp 10.

Cách 3: Dùng STL (cụ thể là dùng SET) trong C++.

Dùng một mảng 50 set, mỗi set lưu các vị trí xuất hiện của loại điểm tương ứng.

Dùng thêm một mảng $a[\ ]$ để lưu điểm của N học sinh.

Với truy vấn loại 1 cần xóa vị trí xuất hiện của loại điểm đang ở vị trí $a[i]$. Cập nhật lại điểm $a[i]$ sau đó thêm vị trí xuất hiện của loại điểm này trong set.

```
s[a[x]].erase(x);
a[x]=y;
s[y].insert(x);
```

Với truy vấn loại 2 thì cần kết hợp tìm kiếm nhị phận để tìm khoảng cách xa nhất trên 50 set.

2.8.3. Chương trình tham khảo

```
#include <bits/stdc++.h>
using namespace std;
int n,q,a[100005];
set<int> s[55];
void solve() {
    cin>>n>>q;
    for(int i=1; i<=n; i++) {
        cin>>a[i];
        s[a[i]].insert(i);
    }
    for(int i=1; i<=q; i++) {
        int t;
        cin>>t;
        if(t==1) {
            int x,y;
            cin>>x>>y;
            s[a[x]].erase(x);
            a[x]=y;
            s[y].insert(x);
```

```
        } else {
            int l,r,mx=-10,mn=-1;
            cin>>l>>r;
            for(int i=1; i<=50; i++) {
                auto it1=s[i].lower_bound(l);
                auto it2=s[i].upper_bound(r);
                if(it2!=s[i].begin()) {
                    --it2;
                    if(it1!=s[i].end()) {
                        int dif=(*it2)-(*it1);
                        if(dif>mx) {
                            mx=dif;
                            mn=i;
                        }
                    }
                }
            }
            cout<<mn<<'\n';
        }
    }
}
int main() {
    freopen("NOST.INP","r",stdin);
    freopen("NOST.INP","r",stdin);
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    solve();
    return 0;
}
```

2.8.4. Cảm nhận

Việc sử dụng SET đã tránh được việc phải dùng cấu trúc dữ liệu Segment tree. Cài đặt chương trình cũng trở nên dễ tiếp cận hơn cho học sinh lớp 10.

Độ phức tạp: $O(50.(N+Q).logN)$.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

2.9. Bài 9. Không Binary index tree

2.9.1. Đề bài: **NOST2**

Cho một mảng gồm N số nguyên dương $a_1, a_2, \dots a_N$ ($1 \le a_i \le 10^9$) và M truy vấn trên đó. Mỗi truy vấn thuộc một trong hai loại sau:

Loại 1: Có dạng ($1\ L\ R\ P$) với ý nghĩa là truy vấn loại 1, lấy tất cả các số trong đoạn $[L, R]$ mà chia hết cho P đem chia cho P, với $P \in \{2,3,5\}$.

Loại 2: Có dạng ($2\ L\ D$) với ý nghĩa là truy vấn loại 2, gán số nguyên dương D cho phần tử ở vị trí L.

Yêu cầu: hãy tin ra mảng cuối cùng nhận được sau M truy vấn.

Input: Dòng đầu tiên là 2 số nguyên dương N, M ($1 \le N, M \le 5.10^4$).

Dòng tiếp theo là N số nguyên dương $a_1, a_2, \dots a_N$.

M dòng tiếp theo là mô tả các truy vấn thuộc loại 1 hoặc 2 nêu ở trên.

Output: Dãy số thu được sau M truy vấn

NOST2.INP	NOST2.OUT	Giải thích
3 5 1 2 3 1 2 2 2 1 2 2 2 2 2 3 1 2 3 3 2 1 5	5 1 1	Dãy ban đầu: {1,2,3}. Sau truy vấn 1 2 2 2 ta được: {1,1,3} Sau truy vấn 1 2 2 2 ta được: {1,1,3} Sau truy vấn 2 2 3 ta được : {1,3,3} Sau truy vấn 1 2 3 3 ta được : {1,1,1} Sau truy vấn 2 1 5 ta được : {5,1,1}

2.9.2. Hướng dẫn giải thuật

Với bài này ta có thể sử dụng cấu trúc dữ liệu Binary Index Tree để giải, tuy nhiên hoàn toàn có thể dùng SET với độ phức tạp thuật toán tương đương, cách viết code thì cũng đơn giản hơn.

Giả sử chúng ta có cấu trúc dữ liệu để có thể tìm các phần tử tiếp kế tiếp chia hết cho P trong đoạn $[L, R]$ trong thời gian $O(t)$ ($O(t)$ bằng bao nhiêu ta sẽ nói sau).

Giả sử chỉ có truy vấn loại 1, khi đó vì các $a_i \le 10^9$, nên trong trường hợp xấu nhất thì ta chỉ chia số a_i cho P được 30 lần (vì $P \ge 2$ và $2^{30} > 10^9$). Do đó độ phức tạp là $30.N.O(t)$.

Bây giờ ta thêm các truy vấn loại 2. Mỗi số được thay thế có thể cần tối đa 30 bước để đạt giá trị bằng 1. Do đó có tối đa M truy vấn 2 thì thời gian thực hiện là $30.O(t).(M+N)$ + thời gian để thay đổi cấu trúc thực hiện truy vấn 2.

Vậy nếu mà $O(t) = O(logN)$ thì bài toán hiển nhiên được giải quyết.

Cách 1: Sử dụng 3 set. Set 1 lưu vị trí các phần tử chia hết cho 2. Set 2 lưu vị trí các phần tử chia hết cho 3. Set 3 lưu vị trí các phần tử chia hết cho 5.

Khởi tạo set như sau:

```
for(int i=1; i<=n; i++) { //tao set
        scanf("%d",&arr[i]);
        if(arr[i]%2==0) myset[2].insert(i);
        if(arr[i]%3==0) myset[3].insert(i);
        if(arr[i]%5==0) myset[5].insert(i);
    }
```

Chương trình tham khảo trong code mẫu.

Cách 2: Dùng cấu trúc dữ liệu BIT xử lý cũng tương tự. Nhưng không nói ở đây, nhưng chắc chắn cách cài đặt sẽ dài dòng hơn rất nhiều.

2.9.3. Chương trình tham khảo

```
#include<bits/stdc++.h>
#define si set<int>::iterator
#define vi vector<int>::iterator
using namespace std;
int n,m,arr[100005];
set<int> myset[6];
int main() {
    freopen("NOST2.INP","r",stdin);
    freopen("NOST2.OUT","w",stdout);
    cin>>n>>m;
    for(int i=1; i<=n; i++) { //tao set
        scanf("%d",&arr[i]);
        if(arr[i]%2==0) myset[2].insert(i);
        if(arr[i]%3==0) myset[3].insert(i);
        if(arr[i]%5==0) myset[5].insert(i);
    }
    while (m--) {
        int type,l,r,p,d;
        scanf("%d",&type);
        if (type==1) {
            scanf("%d%d%d",&l,&r,&p);
            si a=lower_bound(myset[p].begin(),myset[p].end(),l);
            vector<int> del;
            for(si it = a; it!=myset[p].end(); it++) {
                if(*it>r)
                    break;
                arr[*it]/=p;
```

```
                if (arr[*it]%p)
                    del.push_back(*it);
            }
            for(vi it = del.begin(); it!=del.end(); it++)
                myset[p].erase(*it);  //deleting the numbers
        }
        //xoa l khoi set[arr[l]] va them l vao set[d].
        if (type==2) {
            scanf("%d%d",&l,&d);
            if(arr[l]%2==0) myset[2].erase(l);
            if(arr[l]%3==0) myset[3].erase(l);
            if(arr[l]%5==0) myset[5].erase(l);
            //them d vao set
            if (d%2==0) myset[2].insert(l);
            if (d%3==0) myset[3].insert(l);
            if (d%5==0) myset[5].insert(l);
            arr[l] = d;
        }
    }
    for(int i=1; i<=n; i++)
        cout<<arr[i]<<" ";
}
```

2.9.4. Cảm nhận

Việc sử dụng SET trong tình huống này đã thay thế được việc dùng Binary index tree. Tuy nhiên lợi thế về mặt cài đặt là rất rõ ràng.

Độ phức tạp thuật toán $O(30.(M+N).logN)$.

Test kèm theo:

https://drive.google.com/drive/folders/19ZOg8CcfcTpfhvzfE5LDCiYS2l7eXAnc?usp=sharing

3. Một số bài tự giải

https://vn.spoj.com/problems/VMSORT/

https://vn.spoj.com/problems/CPPSET/

https://www.spoj.com/problems/RKS/

https://www.spoj.com/problems/SUBSEQ/

https://www.spoj.com/problems/FACEFRND/

https://www.spoj.com/problems/BOI7SOU/

4. Kết luận

Trong chuyên đề này tôi đã trình bày kiến thức căn bản nhất về cấu trúc dữ liệu được cung cấp sẵn trong thư viện STL của C++ và một số bài toán có ứng dụng nó. Việc ứng dụng giúp quá trình giải quyết một bài toán trở nên đơn giản hơn (đôi khi đơn giản hơn rất nhiều).

Tuy có nhiều cách để giải quyết cùng một vấn đề, nhưng nếu biết khai thác sức mạnh của STL thì mọi việc sẽ được đơn giản hơn đáng kể.

Chuyên đề này tôi đã áp dụng trong quá trình dạy đội tuyển HSG lớp 10.

Do thời gian còn hạn chế và kiến thức, kinh nghiệm còn chưa nhiều nên chắc chắn chuyên đề còn thiếu sót. Tôi rất mong quý thầy cô đồng nghiệp đóng góp ý kiến để chuyên đề được hoàn thiện hơn.

5. TÀI LIỆU THAM KHẢO

- Tài liệu giáo khoa chuyên Tin tập 1, 2, 3.
- http://www.cplusplus.com
- https://www.spoj.com
- http://vn.spoj.com
- https://www.codechef.com
- http://vnoi.info/wiki/algo/trick/matrix-multiplication
- http://codeforces.com