CÁCH SỬ DỤNG INTERVAL TREE, BINARY INDEXED TREE QUA MỘT SỐ BÀI TOÁN QUI HOẠCH ĐỘNG

Cấu trúc dữ liệu (CTDL) là thành tố quan trọng để đưa ra được một giải thuật có hiệu quả. Trong những năm gần đây, khi giới hạn về bộ nhớ không còn là rào cản cho các bài tập tin học thì các bài toán với kích thước dữ liệu lớn xuất hiện phổ biến trong các kỳ thi. Để có được một chương trình hiệu quả (đo bằng tốc độ tính toán) khi giải các bài tập như vậy, việc sử dụng các CTDL để lưu trữ thông tin là một điều kiện tiên quyết.

Có nhiều loại CTDL khác nhau. Tuy nhiên đối với mức độ khó của các bài thi cấp quốc gia có thể kể đến các CTDL sau:

1. Ngăn xếp (stack)
2. Hàng đợi hai đầu (double queue)
3. Đống (heap)
4. RMQ (Range Minimum Query)
5. IT (Interval Tree)
6. BIT (Binarry Indexed Tree)

Trong chuyên đề này, tôi không có ý định trình bày lại các CTDL nói trên. Trình bày chi tiết về chủ đề này đã được thầy Lê Minh Hoàng trình bày trong các chuyên đề bồi dưỡng giáo viên chuyên cũng như trong sách giáo khoa chuyên tin (Tập 2). Ở đây, tôi chỉ dừng lại ở việc phân tích cách sử dụng hai cấu trúc IT và BIT khi giải một số bài toán quy hoạch động. Qua đó khái quát hóa một số nguyên lý chung (theo đánh giá chủ quan của tôi) trong việc áp dụng các cấu trúc này.

Bài toán 1: Cho dãy số $a_1, a_2, ..., a_n$. Hãy tìm dãy con (không nhất thiết gồm các phần tử liên tiếp) tăng dài nhất.

Đây là bài toán qui hoạch động quen thuộc: Đặt $f[i]$ là độ dài dãy con tăng dài nhất kết thúc tại a_i. Ta có công thức qui hoạch động sau:

$$f[i] = \max\{f[k] : k < i, a_k < a_i\} + 1 \qquad (1)$$

Có nhiều cách để tính toán (1) trong thời gian O(log n). Một trong những cách như vậy là sử dụng tìm kiếm nhị phân. Ở đây, chúng ta tiếp cận theo một cách khác:

Trước tiên giả thiết rằng $a_i \in 1, 2, ..., n$ với i=1,2,...,n. Bất đẳng thức $a_k < a_i$ có thể viết dưới dạng $a_k \in [1...a_i - 1]$. Do đó việc tính (1) có thể qui về việc tính lần lượt f[1], f[2], ... và với mỗi i=1,2,...,n thì f[i] được tính bằng cách lấy giá trị lớn nhất của các giá trị f đã được tính có điểm cuối thuộc [1...a_i-1] (mỗi lần có được giá trị $f[i]$ ta ghi nhận nó vào vị trí $a_i \in [1...n]$) và ta có thể sử dụng BIT hoặc IT để thực hiện các truy vấn tìm max này). Dưới đây là mã chương trình viết bằng IT:

```
void update(int r,int k,int l,int u,int v,int val) {
   if (v<k || u>l) return;
   if (u<=k && l<=v) {dt[r]+=val; return;}
   int g=(k+l)/2;
   dt[2*r]+=dt[r]; dt[2*r+1]+=dt[r]; dt[r]=0;
   update(2*r,k,g,u,v,val);
   update(2*r+1,g+1,l,u,v,val);
   it[r]=max(it[2*r]+dt[2*r],it[2*r+1]+dt[2*r+1]);
}
```

```
int get(int r,int k,int u,int v) {
   if (v<k || u>l) return -INF; // INF là giá trị đủ lớn
   if (u<=k && l<=v) return it[r]+dt[r];
int g=(k+l)/2;
   dt[2*r]+=dt[r]; dt[2*r+1]+=dt[r]; dt[r]=0;
int t1=get(2*r,k,g,u,v);
int t2=get(2*r+1,g+1,l,u,v);
   it[r]=max(it[2*r]+dt[2*r],it[2*r+1]+dt[2*r+1]);
   return max(t1,t2);
}
for(int i=1;i<=n;i++) {
   f[i]=get(1,1,n,1,a[i]-1)+1;
   update(1,1,n,a[i],a[i],f[i]);
}
```

(mặc dù không cần *cập nhật lười (lazy update)* trên IT nhưng tôi vẫn viết đầy đủ để các bạn đồng nghiệp tham khảo). Độ phức tạp là O(*n* log *n*)

Tuy nhiên, để có thể sử dụng IT (hoặc BIT) như trên chúng ta đã phải giả thiết rằng $a_i \in [1...n]$. Nếu như không có điều kiện trên thì sao?. Trong trường hợp này chúng ta phải sử dụng một kỹ thuật thường được gọi là *rời rạc hóa*: Cho tương ứng dãy $a_1, a_2, ..., a_n$ với dãy $b_1, b_2, ..., b_n$ sao cho thỏa mãn:

- Nếu $a_i < a_j, a_i > a_j, a_i = a_j$ thì $b_i < b_j, b_i > b_j, b_i = b_j$
- $b_i \in [1...n]$

Dễ thấy việc tìm dãy con tăng dài nhất trên dãy $a_1, a_2, ..., a_n$ hoàn toàn giống như việc tìm dãy con tăng dài nhất trên dãy $b_1, b_2, ..., b_n$. Đoạn mã dưới đây làm công việc này:

```
for(int i=1;i<=n;i++) x[i]=a[i];
sort(x+1,x+n+1);
for(int i=1;i<=n;i++) b[i]=lower_bound(x+1,x+n+1,a[i])-x;
```

Nếu code bằng Pascal, trước tiên ta sắp xếp lại mảng A theo chỉ số:a[id[1]] ≤a[id[2]]≤...a[id[n]]. Sau đó thực hiện đoạn mã sau:

```
m=1; b[id[1]]:=1;
for i:=2 to n do
begin
   if a[id[i]]>a[id[i-1]] then inc(m);
   b[id[i]]:=m;
end;
```

Tất nhiên, cách tiếp cận trên để giải bài toán 1 phức tạp hơn cách tiếp cận sử dụng tìm kiếm nhị phân truyền thống. Tuy vậy qua ví dụ trên có thể tổng kết một số điều khi sử dụng IT, BIT:

- Nói chung, IT hoặc BIT được xây dựng trên *miền giá trị* của mảng. Luôn *co* miền giá trị vào một khoảng hữu hạn đủ nhỏ [1...*m*] bằng kỹ thuật *rời rạc hóa* (nếu cần thiết)
- Viết các biểu thức so sánh thành các biểu thức tập hợp: $a \le x \le b$ được viết thành $x \in [a,b]$ để xây dựng được các truy vấn max, min, sum trên một khoảng. Điều này là quan trọng vì nó thể hiện đặc trưng của IT, BIT

Bài toán 2: Trên một lưới ô vuông kích thước *m* x *n* (*m* hàng, *n* cột) các hàng được đánh số từ trên xuống dưới bắt đầu từ 1, các cột đánh số từ trái sang phải bắt đầu từ 1. Trong *k* ô của lưới có chứa các số nguyên dương (các ô khác không chứa số - có thể coi giá trị bằng 0). Tìm một hành trình từ ô (1,1) đến ô (*m*,*n*) thỏa mãn:

- Từ ô (*i*,*j*) chỉ có thể đi đến (*i*+1,*j*) hoặc (*i*, *j*+1)
- Tổng giá trị các ô đi qua là lớn nhất

Dữ liệu được cho bằng danh sách *k* ô chứa giá trị nguyên dương (u_i, v_i, w_i) thể hiện ô (u_i, v_i) chứa số nguyên dương w_i (các ô khác là giá trị 0).

Lời giải qui hoạch động đơn giản có thể dễ thấy: Đặt *f*[*i*,*j*] là giá trị lớn nhất trên hành trình kết thúc tại ô (*i*, *j*). Khi đó:

$$f[i, j] = \max(f[i-1, j], f[i, j-1]) + a[i, j] \qquad (2)$$

với *a*[*i*,*j*] là giá trị tại ô (*i*,*j*)

Dễ thấy độ phức tạp của thuật toán trên là O(*mn*) do đó không khả thi khi *mn* lớn.

Ta có thể chỉ ra cách tiếp cận qui hoạch động khác có độ phức tạp O(k^2) nếu viết không sử dụng CTDL như sau:

Chú ý rằng các giá trị *f*[*i*,*j*] chỉ nhận *giá trị mới* khi đi qua một ô chứa số nguyên dương thực sự. Do vậy ta có thể sắp xếp lại các ô có giá trị dương sao cho chỉ số hàng tăng dần, nếu chỉ số hàng bằng nhau thì chỉ số cột tăng dần. Đặt *f*[*i*] là giá trị lớn nhất nhận được khi kết thúc ở ô thứ *i*. Khi đó:

$$f[i] = \max\{f[j] : j < i, v_j \le v_i\} + w_i \qquad (3)$$

Có thể thấy (3) tương tự như (1) và lời giải bài toán 1 ở trên hoàn toàn áp dụng được trong trường hợp này. Độ phức tạp thuật toán là O(*k* log *k*) không phụ thuộc vào *m*, *n*. Cũng như bài toán 1, trước khi sử dụng IT hoặc BIT ta phải *rời rạc hóa* các chỉ số cột

Cách thức giải bài tập trên cũng cho một điều khá thú vị khi dạy học sinh cách tiếp cận bài toán. Tùy theo kích thước dữ liệu cho mà thuật toán hiệu quả có thể khác nhau. Chẳng hạn nếu chỉ cho $mn \le 10^6$ thì cách tiếp cận như (2) là hợp lý, còn nếu cho $k \le 10^5$ thì cách tiếp cận như (3) lại là hợp lý.

Bài toán 3 (VOI 2014)

Dãy $C = \{c_1, c_2, ..c_k\}$ được gọi là dãy con của dãy $A = \{a_1, a_2, .., a_n\}$ nếu *C* có thể nhận được bằng cách xóa bớt một số phần tử của dãy *A* và giữ nguyên thứ tự của các phần tử còn lại, nghĩa là tìm được dãy các chỉ số $1 \le l_1 < l_2 < \ldots < l_k \le n$ sao cho $c_1 = a_{l_1}, c_2 = a_{l_2}, ..., c_k = a_{l_k}$. Ta gọi độ dài của dãy là số phần tử của dãy.Cho hai dãy $A = \{a_1, a_2, ..., a_m\}$ và $B = b_1, b_2, ..., b_n\}$ Dãy $C = \{c_1, c_2, ..., c_k\}$ được gọi là dãy con chung bội hai của dãy *A* và *B* nếu *C* vừa là dãy con của dãy *A*, vừa là dãy con của dãy *B* và thỏa mãn điều kiện $2 \times c_i \le c_{i+1}$ ($i = 1, 2, \ldots, k-1$).
Hãy tìm dãy con chung bội hai độ dài lớn nhất của *A* và *B* cho trước (*m*,*n*≤1500)

Ta giải bài toán này bằng phương pháp qui hoạch động. Nếu như chỉ dừng lại *C* là dãy con chung dài nhất của *A* và *B* thì công thức qui hoạch động truyền thống được đặt bằng cách gọi *f*[*i*,*j*] là dãy con chung dài nhất của *i* phần tử đầu dãy *A* và *j* phần tử đầu dãy *B*. Tuy nhiên ở đây ta còn phải quan tâm đến giá trị phần từ của dãy con chung (điều kiện bội 2). Nhận xét thêm phần tử cuối của *C* luôn phải là phần tử của *B*. Do đó ta đặt *f*[*i*,*j*] là dãy con chung bội hai dài nhất của *i* phần tử đầu tiên dãy *A*, *j* phần tử đầu tiên dãy *B* và giá trị phần tử cuối là b_j. Khi đó:

- Nếu $a_i \neq b_j$ thì $f[i,j]=f[i\text{-}1,j]$
- Nếu $a_i = b_j$ thì $f[i,j] = max(f[i-1,j], \max(f[i-1,v] : v < j, b_v \leq b_j/2)+1)$ (4)

Công thức qui hoạch động trên cho chúng ta giải thuật O(max(m,n)^3).

Nếu so sánh công thức (4) với các công thức (1) và (3) ta thấy chúng hoàn toàn tương tự nhau. Do vậy nếu i cố định thì bằng cách sử dụng IT hoặc BIT ta có thể tính $f[i,...]$ trong O(n log n) và do đó thuật toán tính toàn bộ công thức qui hoạch động là O(mn log n).

Dưới đây là chương trình giải bài toán trên dựa theo tư tưởng trên với một sửa đổi nhỏ là ta sử dụng mảng một chiều thay vì mảng hai chiều (dòng sau tính chồng lên dòng trước). Chương trình sử dụng cấu trúc BIT:

```
// Program: LCS2X - VOI 2014

#include <bits/stdc++.h>

#define MAXN 1501
#define tr(i,c) for(typeof((c).begin()) i=(c).begin();i!=(c).end();i++)

using namespace std;

int m, n, a[MAXN], b[MAXN];
set<int> s;
int slx, x[MAXN], pos[MAXN], pos2[MAXN];
int f[MAXN], bit[MAXN];

int get(int u) {
    int kq=bit[u];
    while (u>0) {
kq=max(kq,bit[u]);
        u=u&(u-1);
}
    return kq;
}

void update(int u,int val) {
    while (u<=slx) {
        bit[u]=max(bit[u],val);
        u+=(u & -u);
    }
}

void doc() {
    scanf("%d%d",&m,&n);
    for(int i=1;i<=m;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++) scanf("%d",&b[i]);
    // roi rac hoa mang b -> mang pos
    // mang pos2[i] cho gia tri roi rac cua so lon nhat <=b[i]/2
for(int i=1;i<=n;i++) x[i]=b[i];
    sort(x+1,x+n+1);
    for(int i=1;i<=n;i++) {
    pos[i]=lower_bound(x+1,x+n+1,b[i])-x;
        pos2[i]=upper_bound(x+1,x+n+1,b[i]/2)-x-1;
```

```
    }
}

void tinh() {
    memset(f,0,sizeof(f));
    for(int i=1;i<=m;i++) {
        memset(bit,0,sizeof(bit));
        for(int j=1;j<=n;j++) {
            int t=(a[i]==b[j]) ? get(pos2[j])+1:0;
update(pos[j],f[j]);
            f[j]=max(f[j],t);
}
    }
    int ret=f[1];
    for(int i=1;i<=n;i++) ret=max(ret,f[i]);
    printf("%d\n",ret);
}

int main() {
freopen("LCS2X.INP","r",stdin);
freopen("LCS2X.OUT","w",stdout);
    int T; scanf("%d",&T);
    for(int i=1;i<=T;i++) {
        doc();
        tinh();
    }
}
```

Ta xét một bài toán phức tạp hơn:

Bài tập 4: Cho dãy số $A=(a_1,a_2,a_3,...,a_n)$. Hãy đếm số lượng dãy con tăng dài nhất của dãy số trên.Một dãy con độ dài *k* của dãy *A* được xác định bởi một bộ chỉ số $(u_1<u_2<...<u_k)$ ($1\le u_i\le n$). Hai dãy con $(u_1,u_2,...,u_k)$ và $(v_1,v_2,...,v_t)$ được gọi là khác nhau nếu $k\ne t$ hoặc tồn tại một vị trí *i* sao cho $u_i\ne v_i$. Kết quả lấy theo phần dư của 10^9+7

Ta giải bài toán trên bằng phương pháp qui hoạch động. Trước tiên đặt *f*[*i*] là độ dài của dãy con tăng dài nhất kết thúc tại a_i. Theo như bài toán 1 ta có thể tính mảng *f* trong thời gian O(*n* log *n*). Ta cũng giả thiết luôn $a_i\in[1...n]$

Đặt *g*[*j*] là độ dài dãy con tăng dài nhất kết thúc tại a_i. Ta có công thức:

$$g[i]=\sum\{g[j]: j<i, a[j]<a[i], f[j]=f[i]-1\}$$

Nếu bỏ qua điều kiện *f*[j]=*f*[*i*]-1 thì việc tính *g*[*j*] có thể sử dụng một BIT. Do đó với mỗi giá trị *f*[*j*] ta xây dựng một BIT gồm các giá trị cuối có thể có của các dãy con tăng dài nhất có độ dài *f*[*j*]. Ta có *n* BIT, tuy nhiên tổng số nút của *n* BIT này chỉ bằng *n*. Điều này cho phép chúng ta cài đặt các BIT này tương tự như biểu diễn đồ thị theo kiểu Forward Star - biểu diễn liên tiếp các BIT trên một mảng, vị trí bắt đầu mỗi BIT được đặc trưng bởi một mảng khác (khi code bằng Pascal) hoặc theo kiểu mảng vector - mỗi BIT là một vector (khi code bằng C++). Dưới đây là chương trình viết bằng C++:

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

```
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
#define MAXN 100005
#define oo 1000000005
#define module 1000000007

using namespace std;

int n,m=0,a[MAXN],f[MAXN],h[MAXN],len[MAXN],g[MAXN];
vector<int> b[MAXN], bit[MAXN];

int get(int k,int u) {
    int kq=0;
    while (u>0) {
        kq=(kq+bit[k][u-1])%module;
        u=u&(u-1);
    }
    return kq;
}

void update(int k,int u,int val) {
    while (u<=len[k]) {
        bit[k][u-1]=(bit[k][u-1]+val)%module;
u+=u&(-u);
    }
}

int main() {
    scanf("%d",&n);
    rep(i,1,n) scanf("%d",a+i);
    // Cach tinh mang f[...] su dụng chat nhi phan
    h[0]=-oo;
    rep(i,1,n) {
        int k=lower_bound(h,h+m+1,a[i])-h;
        if (k>m) h[++m]=a[i]; else h[k]=a[i];
        f[i]=k;
    }

    // Xây dựng n BIT bằng mảng vector (nếu sử dụng Pascal thì dùng Forward Star)
    rep(i,1,n) {b[f[i]].push_back(a[i]); bit[f[i]].push_back(0);}
    rep(i,1,m) sort(b[i].begin(),b[i].end());
    rep(i,1,m) len[i]=b[i].size();

    // Đoạn tính toán chính
    rep(i,1,n) {
        int k=f[i], u;
        if (k==1) g[i]=1; else {
            k--;
            // u là chỉ số của a[i] trong cây k=f[i]-1
            u=lower_bound(b[k].begin(),b[k].end(),a[i])-b[k].begin();
            g[i]=get(k,u);
            k++;
        }
        u=lower_bound(b[k].begin(),b[k].end(),a[i])-b[k].begin()+1;
        update(k,u,g[i]);
```

```
}
    int ans=get(m,len[m]);
    printf("%d",ans);
}
```

Với bốn bài toán được trình bày ở trên, một lần nữa nhắc lại các điểm chính cần lưu ý khi sử dụng cấu trúc IT, BIT để giải các bài toán:

- Viết các bất đẳng thức so sánh dưới dạng điều kiện tập hợp (phần tử thuộc một khoảng nào đó). Từ đó tạo ra các sự kiện truy vấn trên một khoảng (max, min, sum)
- Nếu miền giá trị quá lớn thì sử dụng kỹ thuật *rời rạc hóa* để đưa về miền giá trị chấp nhận được.
- Khi có nhiều cấu trúc IT, BIT ta có thể *nén* chúng lại trên một mảng với vị trí đầu mỗi cấu trúc được xác định trên một mảng khác (Forward Star) hoặc sử dụng mảng động nếu ngôn ngữ cho phép.

Dưới đây là một số bài tập áp dụng:

http://vn.spoj.com/problems/MCONVOI/
http://vn.spoj.com/problems/NKREZ/
http://vn.spoj.com/problems/NKTEAM/
http://vn.spoj.com/problems/NKINV/
http://vn.spoj.com/problems/LEM4/
http://vn.spoj.com/problems/PBCSEQ/
http://vn.spoj.com/problems/QBSEGPAR/
http://vn.spoj.com/problems/FOCUS/