CÁC CẤU TRÚC DỮ LIỆU NÂNG CAO CHO BÀI TOÁN TRUY VẤN VÙNG

Trần Việt Khoa
Khoa Công nghệ Thông tin, Trường Đại học Khoa học – Đại học Huế
Email:tvkhoa.husc@gmail.com

TÓM TẮT

Lập trình cạnh tranh là một môn thi trí tuệ về lập trình thường được tổ chức trên Internet hay trên mạng nội bộ, thí sinh tham gia cố gắng để viết chương trình giải quyết công việc theo yêu cầu cho trước. Các trường đại học, các hội tin học khác nhau trên thế giới đều chọn phương thức này để tạo sân chơi cho học sinh, sinh viên về kỹ năng lập trình.

Bài toán truy vấn vùng là một bài toán thường xuyên gặp trong các kỳ thi lập trình cạnh tranh. Bài toán này được giải với nhiều phương pháp khác nhau, tuy nhiên lời giải tốt nhất chính là sử dụng các cấu trúc dữ liệu như cây phân đoạn, cây nhị phân chỉ mục. Bài báo này trình bày nội dung chính về cây phân đoạn cũng như cách áp dụng nó để giải một số bài toán cùng dạng trong các kỳ thi Olympic tin học. Hơn nữa, nội dung trên cũng là kiến thức bổ sung cho sinh viên, học viên cao học trong phần phân tích và thiết kế thuật toán.

***Từ khóa**: Cây phân khoảng, Cây phân đoạn, Cây chỉ mục nhị phân.*

1. GIỚI THIỆU

Bài toán truy vấn vùng là một bài toán thường xuyên gặp trong các kỳ thi lập trình cạnh tranh. Bài toán này được giải với nhiều phương pháp khác nhau, tuy nhiên lời giải tốt nhất cho bài toán này chính là sử dụng các cấu trúc dữ liệu như cây phân đoạn, cây nhị phân chỉ mục. Bài toán này trước đây được giải bằng cấu trúc cây phân khoảng (Interval Tree) [3] tuy nhiên việc cài đặt tương đối phức tạp vì đó là một cấu trúc động. Cây phân đoạn (Segment Tree) giải quyết bài toán trên tốt hơn về mặt cài đặt. Nội dung của cấu trúc dữ liệu cây phân đoạn bằng tiếng Việt không nhiều, chưa phổ biến, và cũng không được trình bày trong các giáo trình về cấu trúc dữ liệu và giải thuật, giáo trình về phân tích và thiết kế thuật toán [1, 2, 3]. Các tài liệu nước ngoài đối với phần này chỉ là các bài viết hướng dẫn [4, 5] do đó khó nắm bắt kiến thức.

Bài báo này trình bày về nội dung của bài toán truy vấn vùng và xây dựng một cấu trúc dữ liệu về cây phân đoạn nhằm đưa ra phương án giải cho một lớp các bài toán cùng dạng. Với cấu trúc xây dựng ở trong bài báo này người đọc dễ dàng áp dụng để giải được một lớp các bài toán cùng dạng và dễ dàng được chấp nhận trên các trang thi trực tuyến.

2. BÀI TOÁN TRUY VẤN VÙNG VÀ CÂY PHÂN ĐOẠN

2.1. Bài toán truy vấn vùng

Phát biểu bài toán: Cho một tập *n* đối tượng $A=\{ a_1, a_2,.., a_n\}$, người ta liên tiếp đưa ra các tác động lên tập *A*, mỗi tác động được thực hiện trên một khoảng liên tiếp các đối tượng có dạng $A_{i,j} = \{a_k, i \le k \le j\}$. Các tác động được đề cập đến ở đây là:

- Các phép toán thống kê thông dụng như tìm phần tử lớn nhất, nhỏ nhất, tính tổng các phần tử, tính trung bình cộng.

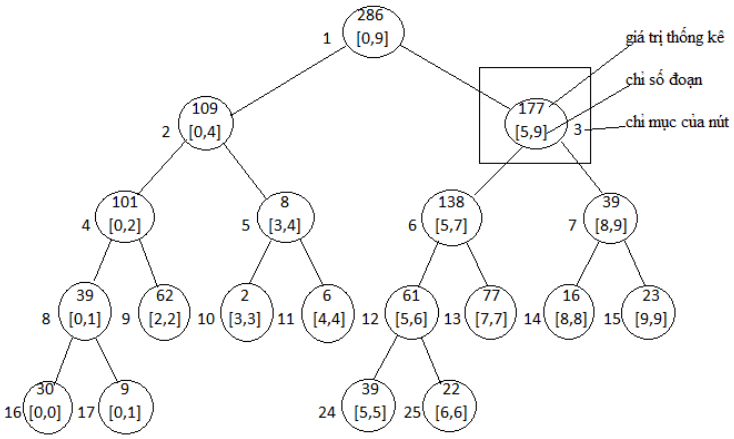
- Phép toán cập nhật, tức là thay đổi giá trị của các phần tử trên khoảng.

Đây là một bài toán tương đối dễ hiểu về giải thuật, có thể dễ dàng giải bài toán bằng thuật toán ngây thơ tức là dùng một vòng lặp xác định từ vị trí *i* đến *j* để xử lý các tác động trên các phần tử. Vì vậy với *n* tác động liên tiếp ta có độ phức tạp thuật toán trong trường hợp này là $O(n^2)$. Việc giảm độ phức tạp cho bài toán trên xuống *O(nlogn)* chính là mục tiêu đặt ra. Bài toán này được giải bằng nhiều phương pháp khác nhau và được phân tích kỹ trong [5], trong phần tiếp theo chúng tôi chỉ đề cập đến phương án giải bằng cây phân đoạn.

2.2. Định nghĩa cây phân đoạn

Cây phân đoạn là một cây nhị phân cân bằng và là cấu trúc tĩnh. Các nút của cây lưu trữ dữ liệu thống kê theo yêu cầu truy vấn của một đoạn xác định của các phần tử trên mảng. Nút lá lưu giá trị là các phần tử của mảng và chỉ số đầu, cuối của đoạn. Nút trung gian (kể cả nút gốc) chứa dữ liệu thống kê truy vấn của đoạn. Chỉ mục đầu tiên của cây (nút gốc) sẽ là *1* và với một nút có chỉ mục là *i* thì nó sẽ có cây con trái với chỉ mục là *2i* và cây con phải có chỉ mục là *2i+1*. Theo tính toán người sử dụng danh sách với kích thước khoảng 4*n+1 phần tử.

Ví dụ 1: xây dựng cây phân đoạn cho dãy số *Arr[]={30, 9, 62, 2, 6, 39, 22, 77, 16, 23}*, với *n = 10* phần tử, trong đó phép thống kê là tính tổng các phần tử trên đoạn. Theo định nghĩa ta có cây như hình 1.



Hình 1. Cây phân đoạn cho bài toán tính tổng

Như vậy nút lá sẽ lưu giá trị của các phần tử trên mảng, nút lá được xác định khi chỉ số đầu bằng chỉ số cuối của đoạn, việc phân chia đoạn thực hiện bằng phép chia để trị, liên tiếp chia đôi đoạn cho đến khi chỉ số đầu bằng chỉ số cuối.

Vấn đề đặt ra là xây dựng các cấu trúc cây sao cho việc nhận dạng được bài toán và áp dụng được thực hiện nhanh nhất. Theo đó ta có các dạng bài toán sau:

- Bài toán 1: truy vấn vùng chỉ có một phép truy vấn thống kê (gọi là query).

- Bài toán 2: truy vấn vùng có cả hai phép toán là truy vấn thống kê (query) và truy vấn cập nhật (update), tức là bài toán tổng quát.

2.3. Cây phân đoạn với bài toán 1

2.3.1. Cấu trúc cây mô tả bằng java

```
public class SegTree{
static class Node{                                          /*1. định nghĩa nút*/
        int start, end;                                     /* địa chỉ đoạn của nút*/
        int sum;                                            /* dữ liệu thống kê*/
        Node(){}                                            /* tạo giá trị ban đầu cho nút*/
        void assignLeaf(..){}                               /* hàm gán giá trị vào nút là*/
        void merge(..){}                                    /* hàm trộn thống kê*/
        int getValue() {}                                   /* hàm lấy giá trị thống kê*/
}
static int Arr[];                                           /*2. Mảng dữ liệu vào */
static Node tree[];                                         /*3.Cây segment dựng lên từ mảng */
static void buildTree(..) {}                                /*4. Hàm tạo cây*/
static Node query(..) {}                                    /*5. Hàm truy vấn – Query*/
static void update(..){}                                    /*6. Hàm cập nhật – Update*/
static void main(){}                                        /*7. Hàm main*/
}
```

2.3.2. Phép dựng cây - thuật toán 1

Dựa trên phần định nghĩa của cây ta có thuật toán dựng cây bằng kỹ thuật đệ quy như sau:

buildTree(int stIndex, int ss, int se) {
Đầu vào
- stIndex: là chỉ mục hiện hành của cây, giá trị khởi đầu là 1
- ss, se: là chỉ mục đầu và cuối của dãy, khởi đầu là 0 và N-1
Đầu ra:Cây phân đoạn
Phương pháp:
Bước 1. Cập nhật địa chỉ đoạn
        tree[stIndex].start=ss; tree[stIndex].end =se;
Bước 2. Cập nhật nút lá, nếu ss = se
        if (ss== se) {tree[stIndex].assignLeaf(Arr[ss]);return;}
Bước 3. Gọi đệ quy xây dựng cây bên trái với địa chỉ stIndex=2*stIndex
        buildTree(2 * stIndex, ss, (ss+se)/2);
Bước 4. Gọi đệ quy xây dựng cây bên phải với địa chỉ stIndex=2*stIndex+1
        buildTree(2 * stIndex+1, (ss+se)/2 + 1, se);
Bước 5. Gọi hàm thống kê trộn nút con trái và con phải, xây dựng nút trung gian và nút gốc
        tree[stIndex].merge(tree[2 * stIndex], tree[2 * stIndex + 1]);
}

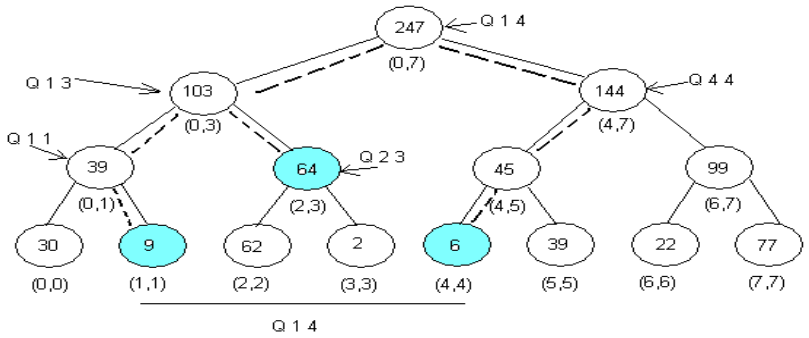
Gọi *T(n)* là hàm đo thời gian thực hiện thuật toán, hàm *T(n)* được gọi hai lần ở bước 3, bước 4 với mỗi lần gọi kích thước *n* giảm đi một nửa ta có được hàm theo công thức truy hồi sau:

$$T(n) = \begin{cases} 1 & nếu\ n = 1 \\ 2T\left(\frac{n}{2}\right) + 1 & nếu\ n > 1 \end{cases} \quad (1)$$

Với công thức 1, dễ dàng xác định được độ phức tạp của thuật toán trên là *O(n)*.

2.3.3. Phép truy vấn - thuật toán 2

Ví dụ 2: xét phép truy vấn Q 2 5 (trên cây lấy địa chỉ là Q 1 4 theo cách lấy địa chỉ của ngôn ngữ lập trình). Do đoạn cần truy vấn [1, 4] nằm ở một phần của đoạn [0, 3] và [4,7] nên bài toán được gọi đệ quy cho nhánh trái và nhánh phải, tương tự trên bài toán được gọi đệ quy liên tiếp, nút có màu sẫm là các nút trả về cho các lời gọi đệ quy và giá trị tổng hợp các nút trên là kết quả của bài toán.



Hình 2. Phép truy vấn Q 2 5 trên cây

Thuật toán được xây dựng bằng kỹ thuật đệ quy như sau:

Node query(int stIndex, int qs, int qe) {
Đầu vào
- stIndex: là chỉ mục hiện hành của cây, giá trị khởi đầu là 1
- qs, qe: là chỉ mục đầu và cuối của vùng truy vấn
Đầu ra: Trả về nút trên cây
Phương pháp:
Bước 1. Nếu [qs start---end qe] thì
if (qs <= tree[stIndex].start && tree[stIndex].end <= qe) return tree[stIndex];
Bước 2. Gọi đệ quy bên phải nếu [mid < qs qe]
int mid = (tree[stIndex].start + tree[stIndex].end) / 2;
if (qs > mid) return **query(2*stIndex+1,qs, qe);**
Bước 3. Gọi đệ quy bên trái nếu [qs qe mid]
if (qe <= mid) return **query(2*stIndex, qs, qe);**
else {
Bước 4. Gọi đệ quy trên cả hai cây là trộn kết quả
- 4.1 Nếu --start---[qs---end qe], trong trường hợp này qe=mid
Node leftResult = **query(2*stIndex, qs, mid);**
- 4.2 Nếu --[qs---qe start]---end, trong trường hợp này qs= mid+1
Node rightResult = **query(2*stIndex+1,mid+1, qe);**
Node result= new Node();
result.merge(leftResult, rightResult);
return result;

```
        }
}
```

Gọi *T(n)* là hàm đo thời gian thực hiện thuật toán, hàm *T(n)* được gọi một lần hoặc ở bước 2 hoặc ở bước 3 với mỗi lần gọi kích thước *n* giảm đi một nửa ta có được hàm theo công thức truy hồi sau:

$$T(n) = \begin{cases} 1 & nếu\ n = 1 \\ T\left(\frac{n}{2}\right) + 1 & nếu\ n > 1 \end{cases} \quad (2)$$

Với công thức 2, dễ dàng xác định được độ phức tạp của thuật toán trên là *O(logn)*.

2.4. Cây phân đoạn với bài toán 2

Do các phép toán cập nhật và truy vấn đan xen nhau, trong lúc cập nhật đoạn, thay vì cập nhật từ nút lá rồi trộn kết quả để xây dựng lại cây như thuật toán 1, người ta chỉ ghi nhận hoặc cập nhật giá trị đó trên nút lá thông qua một trường dữ liệu gọi là pUpdate và sau đó tính được giá trị cập nhật cho toàn đoạn trên nút gốc của đoạn với giá trị là (qs – qe + 1) * pUpdate, quá trình này lặp cho đến khi gặp lệnh query, ngoài việc trộn để lấy giá trị thống kê câu lệnh này gán lại cho trường pUpdate bằng không trên đoạn truy vấn. Như vậy trong trường hợp này hai phép toán query và update có tính năng tương ứng nhau và hỗ trợ cho nhau, phép query cài đặt tương tự thuật toán 2 và có độ phức tạp *O(logn)*. Kết luận với phép cải tiến này bài toán được giải quyết với hai phép toán đều có độ phức tạp *O(logn)*.

2.4.1. Định nghĩa nút trên cây với phép upate đoạn.

Ngoài các trường dữ liệu như trên, ta bổ sung thêm một trường dữ liệu nữa đó là pUpdate tức là chờ đợi cập nhật, giá trị của trường này ghi nhận việc cập nhật dữ liệu ở nút lá, và trong lúc gặp lệnh update nó liên tục được cập nhật cho đến khi gặp lệnh query lập tức nó được trả về giá trị 0. Hàm trộn (merge) ngoài việc thống kê còn tính luôn giá trị cập nhật nếu trường pUpdate lớn hơn 0.

Cấu trúc nút mô tả bằng ngôn ngữ Java

```
static class Node{
        int start, end;
        long sum, pUpdate;
        void assignLeaf(long value) {sum = value;}
        void merge(Node left, Node right) { sum = left.sum + right.sum;
                if (left.pUpdate >0) sum = sum + left.pUpdate * (left.end - left.start +1 );
                if (right.pUpdate >0) sum = sum + right.pUpdate * (right.end - right.start + 1);
        }
        long getValue() { return sum;}
        Boolean hasPUpdate() { return pendingUpdate != 0;}
        void applyPUpdate() {sum += (end - start + 1)*pUpdate; pUpdate = 0;  }
        void addPUpdate(long value) { pUpdate += value;}
        long getPUpdate() {return pUpdate; }

}
```

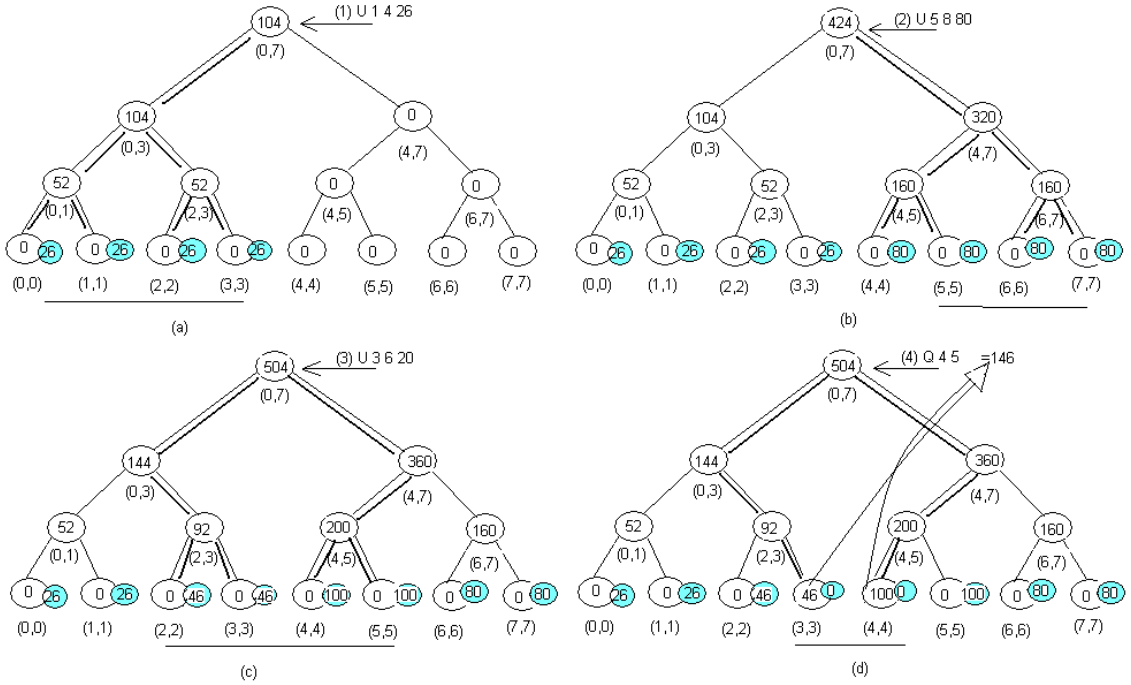
2.4.2. Phép truy vấn- thuật toán 3

Ví dụ 3: xét dãy số *Arr[]={0,0, 0, 0,0, 0, 0, 0}* với các phép toán như hình 3 sau:

- Phép cập nhật U 1 4 26: Phép toán này cập nhật với đoạn chỉ ra tương ứng với cây con trái, sau khi định vị được các nút lá và gán giá trị trường pUpdate với giá trị 26 với nút có màu sẫm, sau đó bằng phép trộn tính ra được giá trị của các nút gốc hình 3.a.

- Phép cập nhật U 5 8 80: tương tự trên cho cây con phải ta có hình 3.b.

- Phép cập nhật U 3 6 20: Phép toán này cập nhật với đoạn thuộc một phần trên cây con trái, một phần cây con phải, các nút được định vị trong khoảng cập nhật sẽ tăng giá trị trường pUdate lên bằng với giá trị đưa vào, sau đó trộn lại các nút gốc hình 3.c.



Hình 3. Cây với phép cập nhật trên đoạn

- Phép truy vấn Q 4 5: Phép toán này với đoạn truy vấn cũng nằm trên hai phần của cây, sau khi định vị được nút lá: cập nhật lại dữ liệu cho nút lá (phép toán này không xảy ra đối với phép cập nhật), gán giá trị pUpdate bằng 0 và thống kê dữ liệu truy vấn hình 3.d.

Như vậy, theo định nghĩa cây ở trên phép query và phép update trong cấu trúc này là tương tự nhau, một hàm kiểm tra tình trạng đã cập nhật hay chưa thông qua dữ liệu pUpdate, một hàm liên tục cập nhật giá trị trường pUpdate và chỉ cập nhật lại nút gốc của đoạn cho giá trị thống kê.

Node query(int stIndex, int qs, int qe) {
Đầu vào
- stIndex: là chỉ mục hiện hành của cây, giá trị khởi đầu là 1
- qs, qe: là chỉ mục đầu và cuối của vùng truy vấn
Đầu ra: trả về nút trên cây
Phương pháp:

```
Bước 1: Nếu [qs start---end  qe] thì
        if (tree[stIndex].start == qs && tree[stIndex].end == qe) {
                if (tree[stIndex].hasPUpdate()) tree[stIndex].applyPUpdate();
                return tree[stIndex];
        }
Bước 2. Gọi đệ quy bên phải nếu [mid < qs qe]
        int mid = (tree[stIndex].start + tree[stIndex].end)/2;
        Node result=new Node();
        if (qs > mid)        result = query(2* stIndex + 1, qs, qe);
Bước 2. Gọi đệ quy bên phải nếu [mid < qs qe]

        else if (qe <= mid) result = query(2* stIndex, qs, qe);
Bước 4. Gọi đệ quy trên cả hai cây là trộn kết quả

        else {
                Node leftResult = query(2* stIndex, qs, mid);
                Node rightResult = query(2* stIndex +1 , mid+1, qe);
                result.start = leftResult.start;
                result.end = rightResult.end;
                result.merge(leftResult, rightResult);
        }
Bước 4. Kiểm tra xem nút hiện hành đã cập nhật hay chưa
        if (tree[stIndex].hasPUpdate()) {
                result.addPUpdate(tree[stIndex].getPUpdate());
                result.applyPUpdate();
        }
        return result;
}
```

2.4.3. Phép cập nhật- thuật toán 4

```
 void update(int stIndex, int qs, int qe, long value) {
                Đầu vào
- stIndex: là chỉ mục hiện hành của cây, giá trị khởi đầu là 1
- qs, qe: là chỉ mục đầu và cuối của vùng truy vấn
Đầu ra: trả về nút trên cây
Phương pháp:
Bước 1. Nếu ss= se thì cập nhật giá trị tại vị trí index với giá trị là value
        if (tree[stIndex].start == tree[stIndex].end) { tree[stIndex].addPUpdate(value);return;
        }
        int mid = (tree[stIndex].start + tree[stIndex].end)/2;
Bước 2. Gọi đệ quy cây phải
        if (qs > mid)
                update(2* stIndex+1, qs, qe, value);
        else
Bước 3. Gọi đệ quy cây trái
        if (qe <= mid)
                update(2* stIndex, qs, qe, value);
Bước 4. Gọi đệ quy trên cả hai cây
else {
                update(2* stIndex, qs, mid, value);
                update(2* stIndex+1, mid+1, qe, value);
        }
Bước 5. Trộn cây
        tree[stIndex].merge(tree[2* stIndex], tree[2* stIndex+1]);

}
```

3. ÁP DỤNG GIẢI CÁC BÀI TOÁN OLYMPIC TIN HỌC

Với cấu trúc cây phân đoạn như trên, các bài toán [6, 7] sau hoàn toàn được giải một cách dễ dàng bằng việc định nghĩa lại nút trên cây.

Bài toán 1 (NKLINEUP). Cho một dãy n số nguyên $a_1, a_2, .., a_n$. Viết chương trình thực hiện q câu hỏi thuộc các dạng sau: Tìm i và j mà $x \le i, j \le y$ và $i! = j$, sao cho a_i là phần tử lớn nhất và a_j là phần tử nhỏ nhất. In ra $a_i - a_j$, cú pháp: *q x y*

Ràng buộc: $0 \le x, a_i \le 10^6, 1 \le n \le 50000, 1 \le q \le 200000$

Bài toán này thuộc dạng cây phân đoạn ở mục 2.3, ta chỉ cần định nghĩa cấu trúc nút như sau:

```
static class Node{
        int start, end;
        int max, min;
        Node(){ max= min=0;       }
        void assignLeaf( int num) { max = num; min = num;  }
        void merge(Node left, Node right) {
                        max = Math.max(left.max, right.max);
                        min = Math.min(left.min, right.min);
        }
        int getValue() { return max - min; }
}
```

Bài toán 2 (FLIPCOIN). Cho một dãy n giá trị logic $a_1, a_2, .., a_n$. Viết chương trình thực hiện q câu hỏi thuộc các dạng sau:

Truy vấn 1: phủ định giá trị các phần tử trong khoảng [a, b].

Truy vấn 2: đếm xem có bao nhiêu phần tử là *true* trong khoảng [a, b]. In ra giá trị đó.

Ràng buộc: $1 \le a, b \le n, 1 \le n, q \le 10^5$.

Bài toán này thuộc dạng cây phân đoạn với hai phép toán query và update, do đó ta dùng cấu trúc cây phân đoạn ở mục 2.4 và chỉ cần định nghĩa là cấu trúc nút như sau:

```
static class Node{
        int start, end;
        int count;
        Boolean pUpdate;
        Node(){ count=0; pUpdate=false;}
        void assignLeaf(Boolean value) {}
        void merge(Node left, Node right) {
                count = (left.pUpdate ? (left.end - left.start + 1 - left.count) : left.count)
                          + (right.pUpdate ? (right.end - right.start + 1 - right.count) :
                        right.count);
        }
        int getValue() {return count; }
        Boolean hasPUpdate() {return pUpdate;}
        void applyPUpdate() {
                count = (end - start + 1) - count;
                pUpdate = false;
```

```
        }
        void addPUpdate(Boolean value) {  pUpdate = !pUpdate;        }
        Boolean getPUpdate() { return true;}
}
```

Kỹ thuật cây phân đoạn này cũng được dùng để giải quyết được nhiều bài toán cùng dạng ở [8].

4. KẾT LUẬN

Như vậy qua việc khảo sát và cài đặt cho bài toán truy vấn vùng bằng cấu trúc dữ liệu là cây phân đoạn, ta có được một số kết luận sau: đạt được tốc độ thực hiện mà bài toán đặt ra, cấu trúc bộ nhớ hợp lý, cài đặt dễ dàng theo mẫu để giải các bài toán cùng dạng thông qua các thuật toán mẫu. Hạn chế của cấu trúc này là các phép toán thống kê trên đoạn là đơn giản, nếu là phép thống kê phức tạp như tần suất xuất hiện, đếm số phần tử thỏa điều kiện nào đó, thì bài toán này sẽ được giải quyết bằng cấu trúc dữ liệu khác đó là cấu trúc cây chỉ mục nhị phân.

Trong khuôn khổ của bài báo, chúng tôi chỉ đưa ra phần lý thuyết cũng như phần thực hành trong việc nghiên cứu bài toán Truy vấn vùng và cấu trúc dữ liệu Cây phân đoạn. Với các bài toán mẫu kèm lời giải cụ thể nó có thể giúp ích cho các sinh viên tiếp cận được với lớp bài toán này trong các kỳ thi Olympic Tin học, thi lập trình trực tuyến. Ngoài các cấu trúc dữ liệu trên, bài toán này còn có thể giải bằng cấu trúc cây chỉ mục nhị phân là hướng nghiên cứu tiếp theo của bài báo.

TÀI LIỆU THAM KHẢO

[1]. Lê Minh Hoàng (1992-1993). *Bài giảng chuyên đề: Giải thuật và lập trình*, Đại học Sư phạm Hà Nội.

[2]. Nguyễn Xuân Huy (2015). *Sáng tạo trong thuật toán và lập trình*, tập 1-3, Nhà xuất bản thông tin và truyền thông.

[3]. Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest. *Introduction to Algorithms 3rd Edition*, PHI LEARNING PVT. LTD-NEW DELHI.

[4]. Steven Halim, Felix Halim. *Competitive Programming 3*, HandBook for ACM ICPC and IOI Contestants 2013.

[5]. Danield. Range Minimum Query and Lowest Common Ancestor, http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=lowestCommonAncestor.

[6]. www.spoj.com/NKLINEUP

[7]. www.codechef.com/problems/FLIPCOIN

[8]. http://praveendhinwacoding.blogspot.com/2013/06/700-problems-to-understand-you-complete.html

ADVANCED DATA STRUCTURES FOR RANGE QUERY PROBLEM

Tran Viet Khoa
Department of Information Technology, Hue University College of Sciences
Email:tvkhoa.husc@gmail.com

ABSTRACT

Competitive programming is an intellectual sport. Competitive programming contests are usually held over the Internet or local networks in which participants trying to solve *given problems. A lot of universities and Information Associations in the world have used this method to organize "playing field" on programming skills for pupils and students.*

The Range Query Problem is a frequently encountered problem in the Vietnam's Olympiads in Information and Technology for Students and International Collegiate Programming Contest (ACM/ICPC). The problem can be solved by various methods; however the best solution to this problem is to apply data structures such as segment tree and binary index tree. In this paper, we present the primary idea about Segment tree and how to apply it to solve Range Query Problems. The paper also provides additional knowledge for students, graduate students in area of algorithms analysis and design .

Keywords*: Interval Tree, Segment Tree, Binary Index Tree.*