

## Cây chỉ số nhị phân 2 chiều (BIT 2 chiều)

# Cây chỉ số nhị phân 2 chiều (BIT 2 chiều)

### Tác giả:

- ▶ Phạm Công Minh - THPT chuyên Khoa học Tự Nhiên, ĐHQGHN

### Reviewer:

- ▶ Lê Minh Hoàng - Đại học Khoa học Tự nhiên, ĐHQG-HCM
- ▶ Phạm Hoàng Hiệp - University of Georgia
- ▶ Ngô Nhật Quang - The University of Texas at Dallas

## Giới thiệu

BIT 2 chiều là cấu trúc dữ liệu mở rộng của BIT 1 chiều. Công dụng chính của BIT 2 chiều là xử lý các truy vấn lên hình chữ nhật con trên một mảng 2 chiều.

Bài viết yêu cầu người đọc hiểu rõ cách hoạt động của BIT 1 chiều. Các bạn có thể đọc về BIT 1 chiều tại đây: [VNOI - Cây chỉ số nhị phân](#) [🔗](#).

## Bài toán

### Các định nghĩa:

- ▶ Trong mảng hai chiều  $A$ ,  $A[u][v]$  là giá trị của phần tử hàng thứ  $u$ , cột thứ  $v$ .
- ▶  $A[u : x][v : y]$  là hình chữ nhật con có góc trái trên là  $(u, v)$  và góc phải dưới là  $(x, y)$ . Nếu  $u > x$  hoặc  $v > y$  thì hình chữ nhật con rỗng.
- ▶  $\sum A[u : x][v : y]$  là tổng các phần tử trong hình chữ nhật con  $A[u : x][v : y]$ .

### Đề bài:

Cho mảng 2 chiều  $A$  có  $N$  hàng  $M$  cột (đánh số từ 1). Có  $Q$  truy vấn thuộc 2 loại:

- ▶ 1  $u\ v\ x$ : Cộng  $x$  vào  $A[u][v]$ .
- ▶ 2  $u\ v$ : Tính  $\sum A[1 : u][1 : v]$ .

Giới hạn:  $1 \leq N, M \leq 10^3, 1 \leq Q \leq 2 \times 10^5$

## Thuật toán ngây thơ 1

Với truy vấn 1, ta cộng trực tiếp vào mảng. Với truy vấn 2, ta duyệt qua từng phần tử của  $A[1 : u][1 : v]$  và cộng giá trị vào kết quả.

```

1 | int A[N][M];
2 |
3 | void add(int u, int v, int x){
4 |     A[u][v] += x;
5 | }
6 |
7 | int query(int u, int v){
8 |     int sum = 0;
9 |     for(int i = 1; i <= u; i++){
10 |         for(int j = 1; j <= v; j++){
11 |             sum += A[i][j];
12 |         }
13 |     }
14 |     return sum;
15 | }
```

### Phân tích

- Độ phức tạp khi cập nhật:  $O(1)$
- Độ phức tạp khi truy vấn:  $O(u \times v) = O(N \times M)$
- Có  $Q$  truy vấn, nên độ phức tạp là  $O(Q + Q \times N \times M) = O(Q \times N \times M)$

## Thuật toán ngây thơ 2

Ta định nghĩa  $lsb(x)$  là giá trị của bit 1 nhỏ nhất trong biểu diễn nhị phân của  $x$ . Ví dụ:

- $lsb((11)_{10}) = lsb((1011)_2) = 1$
- $lsb((24)_{10}) = lsb((11000)_2) = 8$

Ta sẽ lưu  $n$  BIT 1 chiều, mỗi BIT quản lý một hàng.

Như đã giới thiệu trong bài viết BIT 1 chiều, phần tử thứ  $v$  trong BIT 1 chiều sẽ lưu tổng các phần tử trong đoạn  $[i - lsb(i) + 1, i]$ . Ở đây, phần tử thứ  $j$  của BIT thứ  $i$  sẽ lưu  $\sum A[i : i][j - lsb(j) + 1 : j]$ .

Đối với truy vấn 1 ta update BIT của hàng  $u$ . Còn đối với truy vấn 2 ta duyệt qua và truy vấn trên từng BIT của các hàng từ 1 đến  $u$ .

```

1 | int A[N][M], BIT[N][M];
2 |
3 | void add(int u, int v, int x){
4 |     for(v; v <= m; v += v & (-v)) BIT[u][v] += x;
5 | }
6 |
7 |
```

```

1  int query(int u, int v){
2      int sum = 0;
3      for(int i = 1; i <= u; i++){
4          for(int j = v; j > 0; j -= j&(-j)) sum += BIT[i][j];
5      }
6      return sum;
7  }
8
9  void preprocess(){
10     for(int i = 1; i <= n; i++){
11         for(int j = 1; j <= m; j++){
12             add(i, j, A[i][j]);
13         }
14     }
15 }

```

## Phân tích

- Độ phức tạp tiền xử lý:  $O(N \times M \times \log M)$
- Độ phức tạp khi cập nhật:  $O(\log M)$
- Độ phức tạp khi truy vấn:  $O(u \times \log M) = O(N \times \log M)$
- Có  $Q$  truy vấn, nên độ phức tạp là  $O(Q \times \log M + Q \times N \times \log M) = O(Q \times N \times \log M)$

## BIT 2 chiều

Ta gọi BIT trong phần nhị phân 2 là  $BIT_{nt}$ . Như vậy  $BIT_{nt}[i][j] = \sum A[i : i][j - lsb(j) + 1 : j]$

Từ thuật toán *ngây thơ 2*, thay vì sử dụng  $n$  BIT 1 chiều độc lập, ta có thể sử dụng một BIT 1 chiều lớn để quản lý toàn bộ  $n$  BIT 1 chiều. Như vậy, mỗi phần tử của BIT lớn là một BIT nhỏ gồm  $m$  phần tử, BIT nhỏ thứ  $i$  quản lý thông tin về các  $BIT_{nt}$  trong đoạn  $[i - lsb(i) + 1, i]$ .

Trong BIT 2 chiều, phần tử thứ  $j$  của BIT nhỏ thứ  $i$  sẽ lưu:

$$\sum_{k=i-lsb(i)+1}^i BIT_{nt}[k][j]$$

Vì  $BIT_{nt}[k][j] = A[k : k][j - lsb(j) + 1 : j]$  nên tổng này tương đương với:

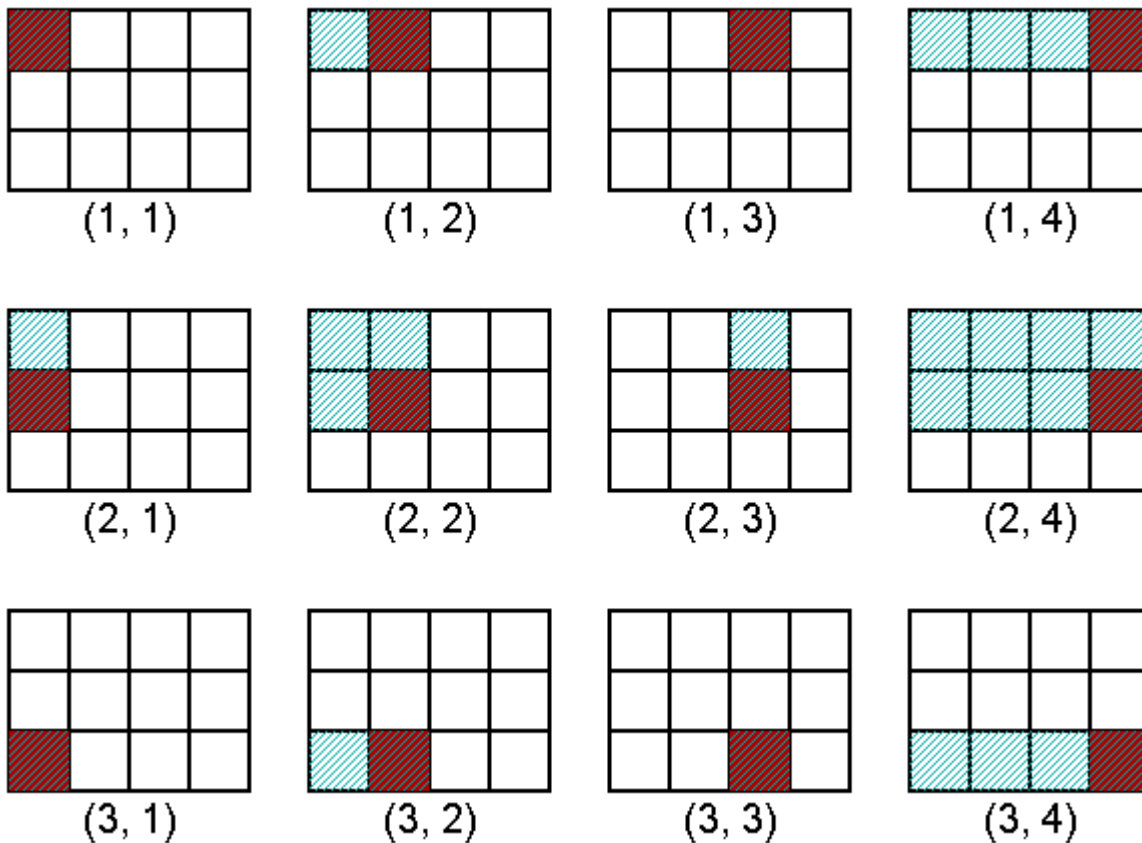
$$\sum_{k=i-lsb(i)+1}^i A[k : k][j - lsb(j) + 1 : j]$$

Ta có thể viết lại biểu thức thành:

$$\sum A[i - lsb(i) + 1 : i][j - lsb(j) + 1 : j]$$

Như vậy phần tử thứ  $j$  của BIT thứ  $i$  trong BIT 2 chiều lưu tổng các phần tử trong hình chữ nhật con có góc trái trên là  $(i - lsb(i) + 1, j - lsb(j) + 1)$  và góc phải dưới là  $(i, j)$ .

Dưới đây là hình minh họa cho trường hợp  $N = 3, M = 4$ .



## Cài đặt

Ta khai báo BIT 2 chiều dưới dạng một mảng  $N \times M$ , trong đó `BIT[u][v]` lưu giá trị của phần tử thứ  $v$  trong BIT thứ  $u$ ;

```
1 | int BIT[N][M];
```

Hàm để update:

```
1 | void add(int u, int v, int x){
2 |     for(int i = u; i <= n; i += i & (-i)){
3 |         for(int j = v; j <= m; j += j & (-j)) BIT[i][j] += x;
4 |     }
5 | }
```

Hàm để truy vấn:

```
1 | int query(int u, int v){
2 |     int sum = 0;
3 |     for(int i = u; i > 0; i -= i & (-i)){
4 |         for(int j = v; j > 0; j -= j & (-j)) sum += BIT[i][j];
5 |     }
```

```

5 |         for(int j = v; j > 0; j -= j&(-j)) sum += BIT[i][j];
6 |     }
7 |     return sum;
  | }

```

Để tính tổng các phần tử trong một hình chữ nhật nhất định, ta có thể sử dụng tổng tiền tố 2 chiều, có công thức như sau:

$$\sum A[a : u][b : v] = \sum A[1 : u][1 : v] - \sum A[1 : a - 1][1 : v] - \sum A[1 : u][1 : b - 1] + \sum A[1 : a - 1][1 : b - 1]$$

## Phân tích

- Độ phức tạp khi cập nhật:  $O(\log N \times \log M)$
- Độ phức tạp khi truy vấn:  $O(\log N \times \log M)$
- Có  $Q$  truy vấn, nên độ phức tạp là  $O(Q \times \log N \times \log M)$

## Cập nhật hình chữ nhật con, truy vấn phần tử

Ta thay đổi bài toán ban đầu như sau:

- 1  $a \ b \ u \ v \ x$ : Cộng  $x$  vào các phần tử thuộc  $A[a : u][b : v]$
- 2  $u \ v$ : Tính  $A[u][v]$

Tương tự với BIT 1 chiều, ta sẽ sử dụng mảng hiệu để cập nhật.

Ta có:

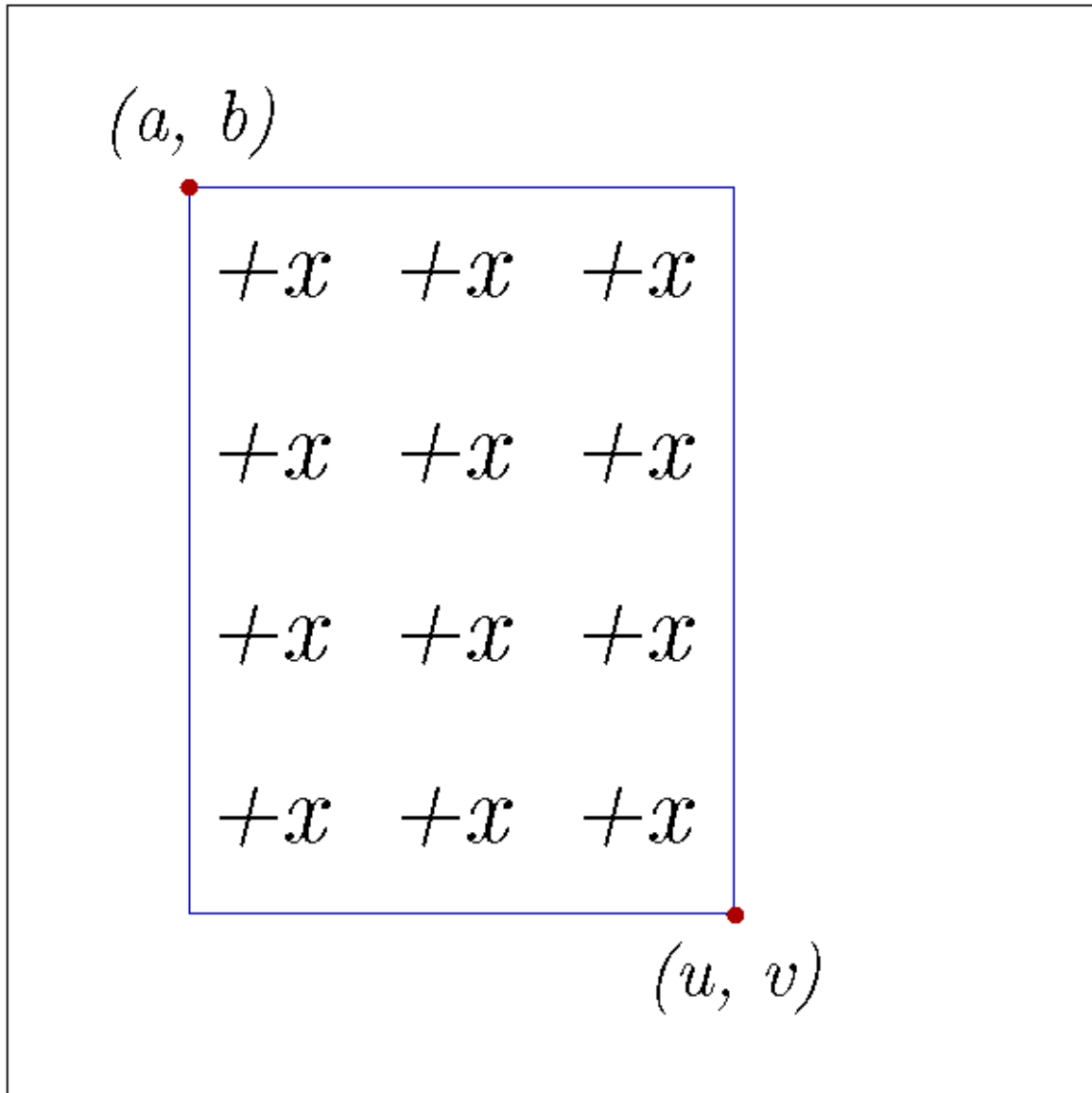
$$\sum A[1 : i][1 : j] = \sum A[1 : i - 1][1 : j] + \sum A[1 : i][1 : j - 1] - \sum A[1 : i - 1][1 : j - 1] + \sum A[i][j]$$

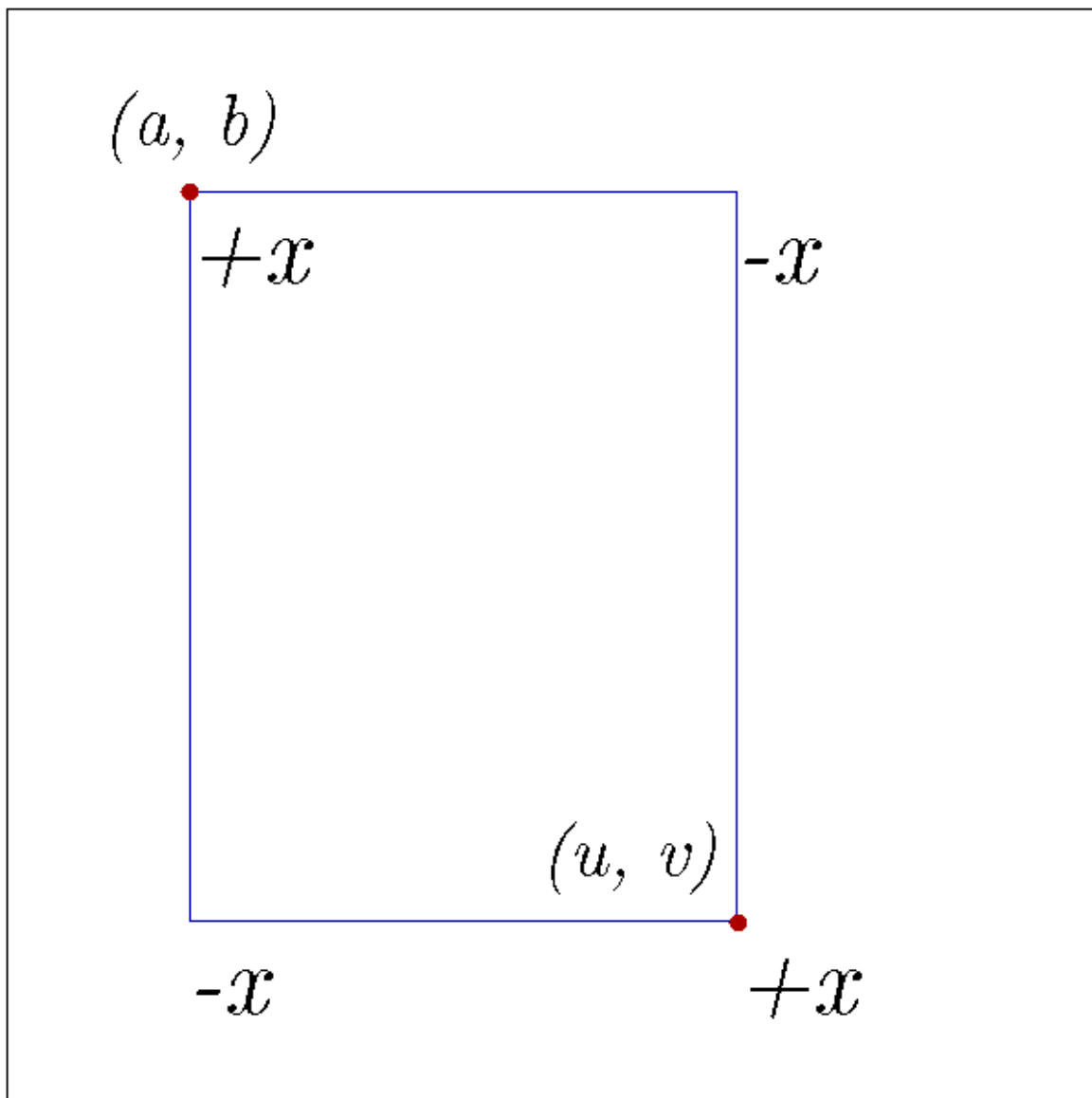
Do đó, ta có thể lưu  $D[i][j] = A[i][j] - A[i - 1][j] - A[i][j - 1] + A[i - 1][j - 1]$ . Khi đó,  $A[i][j] = \sum D[1 : i][1 : j]$

Khi ta thực hiện truy vấn 1, có 4 giá trị của  $D$  thay đổi:

- $D[a][b]$  tăng lên  $x$
- $D[u + 1][b]$  giảm đi  $x$
- $D[a][v + 1]$  giảm đi  $x$
- $D[u + 1][v + 1]$  tăng lên  $x$

Nếu vẫn chưa rõ, bạn đọc có thể tham khảo hình minh họa sau:

 $A$



$D$

## Cài đặt

Hàm cập nhật:

```

1 | void rectAdd(int a, int b, int u, int v, int x){
2 |     add(a, b, x);
3 |     add(u+1, v+1, x);
4 |     add(u+1, b, -x);
5 |     add(a, v+1, -x);
6 | }
```

Hàm truy vấn tương tự như phần trước.

## Cập nhật hình chữ nhật con, truy vấn hình chữ nhật con

Ta thay đổi bài toán ban đầu:

- 1  $a\ b\ u\ v\ x$ : Cộng  $x$  vào các phần tử thuộc  $A[a : u][b : v]$
- 2  $u\ v$ : Tính  $\sum A[1 : u][1 : v]$

Ta tiếp tục sử dụng ý tưởng mảng hiệu. Do  $A[i][j] = \sum D[1 : i][1 : j]$  nên ta có:

$$\begin{aligned}
 \sum A[1 : i][1 : j] &= \sum_{k=1}^i \sum_{l=1}^j A[k][l] \\
 &= \sum_{k=1}^i \sum_{l=1}^j \sum D[1 : k][1 : l] \\
 &= \sum_{k=1}^i \sum_{l=1}^j \sum_{a=1}^l \sum_{b=1}^k D[a][b] \\
 &= \sum_{k=1}^i \sum_{l=1}^j (i - k + 1)(j - l + 1) D[k][l] \\
 &= \sum_{k=1}^i \sum_{l=1}^j ((i + 1) \times (j + 1) \times D[k][l] - (j + 1) \times k \times D[k][l] \\
 &\quad - (i + 1) \times l \times D[k][l] + k \times l \times D[k][l]) \\
 &= \sum_{k=1}^i \sum_{l=1}^j (i + 1) \times (j + 1) \times D[k][l] - \sum_{k=1}^i \sum_{l=1}^j (j + 1) \times k \times D[k][l] \\
 &\quad - \sum_{k=1}^i \sum_{l=1}^j (i + 1) \times l \times D[k][l] + \sum_{k=1}^i \sum_{l=1}^j k \times l \times D[k][l]
 \end{aligned}$$

Dựa vào công thức biến đổi ở trên, ta cần duy trì  $D[i][j], i \times D[i][j], j \times D[i][j], i \times j \times D[i][j]$  bằng bốn BIT:

```

1  int BIT[4][N][M]; // {D[i][j]; i*D[i][j]; j*D[i][j]; i*j*D[i][j]}
2
3  void add(int u, int v, int x){
4      for(int i = u; i <= n; i += i&(-i)){
5          for(int j = v; j <= m; j += j&(-j)){
6              BIT[0][i][j] += x;
7              BIT[1][i][j] += u * x;
8              BIT[2][i][j] += v * x;
9              BIT[3][i][j] += u * v * x;
10         }
11     }
12 }
13
14 void rectAdd(int a, int b, int u, int v, int x){
15     add(a, b, x);
16     add(a, v + 1, -x);
17     add(u + 1, b, -x);
18
19 
```



```

    add(u + 1, v + 1, x);
}

```


Khi truy vấn, ta lấy từng hệ số nhân lên rồi cộng trừ để ra kết quả

```

1  int query(int u, int v){
2      int a[4] = {0, 0, 0, 0};
3      for(int ty = 0; ty < 4; ty++){
4          for(int i = u; i > 0; i -= i&(-i)){
5              for(int j = v; j > 0; j -= j&(-j)){
6                  a[ty] += BIT[ty][i][j];
7              }
8          }
9      }
10     return a[0]*(u + 1)*(v + 1) - a[1]*(v + 1) - a[2]*(u + 1) + a[3];
11 }

```

## Kĩ thuật nén BIT 2 chiều

Phần này được lấy nhiều cảm hứng từ [blog cá nóc cắn cáp](#) 

**Chú ý kĩ thuật này chỉ dùng được khi ta biết trước tất cả các truy vấn.**

Ta thay đổi giới hạn bài toán ban đầu thành  $1 \leq N, M, Q \leq 10^5$ .

Ta sẽ không thể lưu được toàn bộ BIT 2 chiều bằng một mảng  $N \times M$ , nếu sử dụng `std::map` hay `std::unordered_map` thì code sẽ không đủ nhanh để AC.

Tuy nhiên, ta nhận thấy rằng, với mỗi truy vấn, chỉ có  $\log N$  BIT được duyệt qua, và mỗi BIT chỉ có thao tác là cộng một phần tử và tính tổng một tiền tố. Vì vậy, đối với mỗi BIT, ta có thể lưu lại tất cả các vị trí được cộng và được truy vấn trước, sau đó tiến hành rời rạc hóa các vị trí đó. Do có  $Q$  truy vấn, mỗi truy vấn tạo thêm tối đa 1 vị trí trên  $\log N$  BIT nên có tổng cộng  $Q \times \log N$  vị trí cần lưu.

Để lưu như vậy, ta tạo 2 hàm mới chỉ để lưu các vị trí được truy vấn.

```

1  vector<int> pos[N];
2  vector<int> BIT[N];
3
4  void fakeAdd(int u, int v, int x){
5      for(u; u <= n; u += u&(-u)){
6          pos[u].push_back(v);
7      }
8  }
9
10 void fakeQuery(int u, int v){
11     for(u; u <= n; u += u&(-u)){
12         pos[u].push_back(v);
13     }
14 }

```

```

    }
}

```

Sau khi lưu các vị trí cần thiết, ta tiến hành rời rạc hóa trên từng BIT.

```

1 void compress(){
2     for(int i = 1; i <= n; i++){
3         pos[i].push_back(0);
4         sort(pos[i].begin(), pos[i].end());
5         pos[i].erase(unique(pos[i].begin(), pos[i].end()), pos[i].end());
6         BIT[i].assign(pos[i].size(), 0);
7     }
8 }

```

Khi đã rời rạc hóa xong, ta thực hiện các truy vấn như thường. Lưu ý lúc này mảng `pos` chỉ để ánh xạ lại index trên mảng đã được rời rạc hóa.






```



1 void add(int u, int v, int x){
2     for(int i = u; i <= n; i += i & (-i)){
3         for(int j = lower_bound(pos[i].begin(), pos[i].end(), v) - pos[i].begin(); j < pos[i].size(); j++){
4             BIT[i][j] += x;
5         }
6     }
7 }
8
9 void query(int u, int v){
10    int sum = 0;
11    for(int i = u; i > 0; i -= i & (-i)){
12        for(int j = lower_bound(pos[i].begin(), pos[i].end(), v) - pos[i].begin(); j < pos[i].size(); j++){
13            sum += BIT[i][j];
14        }
15    }
16    return sum;
17 }

```



## Bài tập áp dụng

- ▶ [SPOJ - MATSUM](#) 
- ▶ [Codeforces - The Untended Antiquity](#) 
- ▶ [DHBB 2022 - Thao tác trên bảng](#) 
- ▶ [Thi thử VNOI - Đếm nghịch thế](#) 
- ▶ [LQDOJ - Khu Rừng 5](#) 

- [LQDOJ - Khu Rừng 6](#) 
- [VNOJ - Another Longest Increasing Subsequence Problem](#) 

Được cung cấp bởi [Wiki.js](#)