

# Range Minimum Query (RMQ) - Sparse Table

## Range Minimum Query (RMQ) - Sparse Table

### Tác giả:

- Lê Minh Hoàng - Đại học Khoa học Tự nhiên, ĐHQG-HCM

### Reviewer:

- Hồ Ngọc Vĩnh Phát - Đại học Khoa học Tự nhiên, ĐHQG-HCM
- Hoàng Xuân Nhật - Đại học Khoa học Tự nhiên, ĐHQG-HCM
- Nguyễn Anh Bảo - Đại học Bách Khoa Hà Nội

## Giới thiệu

Bài toán *RMQ* được phát biểu như sau:

- Cho mảng  $A$  gồm  $N$  phần tử và  $Q$  truy vấn có dạng  $(l, r)$ . Với mỗi truy vấn, in ra giá trị nhỏ nhất trong mảng  $A$  từ  $l$  đến  $r$ .  
Ví dụ:  $A = [4, 6, 1, 5, 7, 3] \rightarrow \min[2 \dots 5] = \min(6, 1, 5, 7) = 1$

Bài toán *RMQ* có nhiều cách giải, nhưng 2 cách phổ biến nhất là:

- **Sparse Table**:  $\mathcal{O}(N \log N)$  tiền xử lý,  $\mathcal{O}(1)$  mỗi truy vấn.
- **Segment Tree**:  $\mathcal{O}(N)$  tiền xử lý,  $\mathcal{O}(\log N)$  mỗi truy vấn.

Sự khác biệt giữa 2 cách giải này nằm ở chỗ, **Segment Tree** có thể xử lý được **hoạt động sửa đổi** xen kẽ với các truy vấn, còn Sparse Table thì không.

Nhưng Sparse Table không "phế", sức mạnh của Sparse Table nằm ở khả năng truy vấn trong  $\mathcal{O}(1)$  khi các phép toán thoả mãn tính chất **Idempotence**  $\square$ : "một giá trị có thể **xuất hiện nhiều lần** nhưng **không làm thay đổi kết quả** phép toán", ví dụ như *min*, *max*, *gcd*, *lcm*, *and*, *or*, ...

Và Sparse Table cũng có khả năng truy vấn trong  $\mathcal{O}(\log N)$  nếu phép toán không thoả mãn tính chất Idempotence (ví dụ như bài toán Range Sum Query ở [bên dưới](#)).

**Lưu ý:** Trong suốt bài viết mình dùng `__lg(x)` để tính  $\log_2$  của  $x$  vì ta cần giá trị nguyên, còn `log2(x)` thì trả về số thực. Nếu không muốn dùng hàm thì có thể tính trước như sau:

```

1 | int lg2[N];
2 | void BuildLog2Array() {
3 |     lg2[1] = 0;
4 |     for (int i = 2; i < N; ++i)
5 |         lg2[i] = lg2[i / 2] + 1;
6 | }

```

## Sparse Table

Đầu tiên, ta sẽ tìm hiểu về ý tưởng để tối ưu thời gian truy vấn từ  $\mathcal{O}(N)$  đến  $\mathcal{O}(1)$  của Sparse Table qua bài toán Range Minimum Query (RMQ), và cách để truy vấn trong  $\mathcal{O}(\log N)$  nếu phép toán không thoả mãn tính chất Idempotence qua bài toán Range Sum Query (RSQ).

### Range Minimum Query (RMQ)

Cho mảng  $A$  gồm  $N$  phần tử và  $Q$  truy vấn có dạng  $(l, r)$ . Với mỗi truy vấn, in ra giá trị nhỏ nhất trong mảng  $A$  từ  $l$  đến  $r$ .

Giới hạn:  $N, Q \leq 10^5$

#### Thuật toán ngây thơ

Ta duyệt qua tất cả phần tử.

### Truy vấn min trong đoạn [1..7]

	1	2	3	4	5	6	7	8
A	8	5	3	9	7	3	4	2

$mi = +\infty$

```

1 | int a[N];
2 | int queryMin(int l, int r) {
3 |     int mi = INT_MAX;
4 |     for (int i = l; i <= r; ++i) {
5 |         mi = min(mi, a[i]);
6 |     }
7 |     return mi;
8 | }

```

#### Phân tích

- Độ phức tạp truy vấn:  $\mathcal{O}(r - l + 1) = \mathcal{O}(N)$
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp là  $\mathcal{O}(Q \cdot N)$

Câu hỏi đặt ra là ta còn có thể tối ưu thời gian truy vấn được hay không?

- Nhận xét: Thay vì duyệt qua từng phần tử, ta có thể duyệt qua từng nhóm 2 phần tử. Từ đó, ta có thể giảm thời gian truy vấn xuống còn  $\mathcal{O}(\frac{N}{2})$

### Thuật toán tối ưu 1.1

- Ta xây dựng mảng  $a2$  với công thức  $a2_i = \min(a_i, a_{i+1})$ .

	1	2	3	4	5	6	7	8
A	8	5	3	9	7	3	4	2

A2								
	1	2	3	4	5	6	7	

$$A2_i = \min(A_i, A_{i+1})$$

- Khi truy vấn, nếu độ dài đoạn cần truy vấn  $len = 1$  thì ta in ra  $a[l]$ , nếu  $len > 1$  thì ra dùng mảng  $a2$ :

### Truy vấn min trong đoạn [1..7]

	1	2	3	4	5	6	7	8
A	8	5	3	9	7	3	4	2

A2	5	3	3	7	3	3	2	
	1	2	3	4	5	6	7	

$$mi = +\infty$$

```

1 | int a[N], a2[N];
2 | void preprocess() {
3 |     for (int i = 1; i + 1 <= n; ++i) {
4 |         a2[i] = min(a[i], a[i + 1]);
5 |     }
6 | }
7 |

```

```

1  int queryMin(int l, int r) {
2      int len = r - l + 1;
3      if (len == 1) return a[l];
4
5      int mi = INT_MAX;
6      for (int i = l; i + 1 <= r; i += 2) {
7          mi = min(mi, a2[i]);
8      }
9      // khi kết thúc vòng lặp phía trên, có thể còn 1 số phần tử chưa được xét
10     // ví dụ [l, r] = [1, 5], ta chỉ mới xét a[1...4] chứ chưa xét a[5]
11     // nhận xét: lượng phần tử chưa xét nhỏ hơn 2
12     // vậy nên ta chỉ cần xét thêm a[r - 1] và a[r]
13     // (sức mạnh của tính chất Idempotence bắt đầu tại đây)
14     mi = min(mi, a2[r - 1]);
15     return mi;
16 }

```

### Phân tích:

- Độ phức tạp tiền xử lý:  $\mathcal{O}(N)$  (tạo mảng  $a2$ )
- Độ phức tạp truy vấn:  $\mathcal{O}(\frac{N}{2} + 1)$  (1 vòng for và 1 lệnh if cho biến  $len$ )
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(N + Q \cdot (\frac{N}{2} + 1))$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(2N)$  (2 mảng  $a$  và  $a2$ )

### Thuật toán tối ưu 1.2

Tương tự 1.1, ta có nhận xét: Thay vì duyệt qua từng nhóm 2 phần tử, ta có thể duyệt qua từng nhóm 4 phần tử.

	1	2	3	4	5	6	7	8
A	8	5	3	9	7	3	4	2
A2	5	3	3	7	3	3	2	
A4								
	1	2	3	4	5			

$$A4_i = \min(A2_i, A2_{i+2})$$

Từ đó, ta có thể giảm thời gian truy vấn xuống còn  $\mathcal{O}(\frac{N}{4})$

Khi truy vấn:

- Nếu độ dài đoạn cần truy vấn  $len = 1$  thì ta in ra  $a[l]$

- Nếu độ dài đoạn cần truy vấn  $len$  thỏa mãn  $1 < len < 4$  thì ta in ra  $\min(a2[l], a2[r - 1])$
- nếu  $len \geq 4$  thì ra dùng mảng  $a4$ :

## Truy vấn min trong đoạn [1..7]

	1	2	3	4	5	6	7	8
A	8	5	3	9	7	3	4	2
A4	3	3	3	3	2			
	1	2	3	4	5			

$mi = +\infty$

```

1  int a[N], a2[N], a4[N];
2  void preprocess() {
3      for (int i = 1; i + 1 <= n; ++i) {
4          a2[i] = min(a[i], a[i + 1]);
5      }
6      for (int i = 1; i + 3 <= n; ++i) {
7          a4[i] = min(a2[i], a2[i + 2]);
8      }
9  }
10 int queryMin(int l, int r) {
11     int len = r - l + 1;
12     if (len == 1) return a[l];
13     if (len < 4) {
14         return min(a2[l], a2[r - 1]);
15         // dòng này hợp lý bởi vì chắc chắn 2 đoạn [l, l + 1] và [r - 1, r] sẽ
16     }
17
18     int mi = INT_MAX;
19     for (int i = l; i + 3 <= r; i += 4) {
20         mi = min(mi, a4[i]);
21     }
22     mi = min(mi, a4[r - 3]);
23     return mi;
24 }
```

### Phân tích:

- Độ phức tạp tiền xử lý:  $\mathcal{O}(2N)$  (tạo mảng  $a2$  và  $a4$ )
- Độ phức tạp truy vấn:  $\mathcal{O}(\frac{N}{4} + 2)$  (1 vòng for và 2 lệnh if cho biến  $len$ )

- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(2N + Q \cdot (\frac{N}{4} + 2))$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(3N)$  (3 mảng  $a$ ,  $a2$  và  $a4$ )

### Thuật toán tối ưu 1.3

Ta vẫn có thể tối ưu thời gian truy vấn bằng cách duyệt qua các nhóm lớn hơn (nhóm độ lớn 8 phần tử).

	1	2	3	4	5	6	7	8
A	8	5	3	9	7	3	4	2
A2	5	3	3	7	3	3	2	
A4	3	3	3	3	2			
A8								

1

$$A8_i = \min(A4_i, A4_{i+4})$$

```

1  int a[N], a2[N], a4[N], a8[N];
2  void preprocess() {
3      for (int i = 1; i + 1 <= n; ++i) {
4          a2[i] = min(a[i], a[i + 1]);
5      }
6      for (int i = 1; i + 3 <= n; ++i) {
7          a4[i] = min(a2[i], a2[i + 2]);
8      }
9      for (int i = 1; i + 7 <= n; ++i) {
10         a8[i] = min(a4[i], a4[i + 4]);
11     }
12 }
13 int queryMin(int l, int r) {
14     int len = r - l + 1;
15     if (len == 1) return a[l];
16     if (len < 4) return min(a2[l], a2[r - 1]);
17     if (len < 8) return min(a4[l], a4[r - 3]);
18
19     int mi = INT_MAX;
20     for (int i = l; i + 7 <= r; i += 8) {
21         mi = min(mi, a8[i]);
22     }
23     mi = min(mi, a8[r - 7]);
24
25 
```

```
    return mi;
}
```

### Phân tích:

- Độ phức tạp tiền xử lý:  $\mathcal{O}(3N)$  (tạo mảng  $a2$ ,  $a4$  và  $a8$ )
- Độ phức tạp truy vấn:  $\mathcal{O}(\frac{N}{8} + 3)$  (1 vòng for và 3 lệnh if cho biến  $len$ )
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(3N + Q \cdot (\frac{N}{8} + 3))$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(4N)$  (4 mảng  $a$ ,  $a2$ ,  $a4$  và  $a8$ )

### Thuật toán tối ưu 1.n

Nếu ta làm tiếp như thuật toán tối ưu 1.3 (tiếp tục tạo các mảng  $a16, a32, \dots, a65536$ ) ta sẽ có  $\log_2(N)$  mảng  $a$ , độ phức tạp bài toán lúc này như sau:

- Độ phức tạp tiền xử lý:  $\mathcal{O}(N \log N)$  ( $\log_2$  mảng  $a$ )
- Độ phức tạp truy vấn:  $\mathcal{O}\left(\frac{N}{2^{\log N}} + \log N\right) = \mathcal{O}(\log N)$  (1 vòng for và  $\log_2$  lệnh if cho biến  $len$ )
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(N \log N + Q \log N)$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(N \log N)$  (mảng  $a$  ban đầu và  $\log_2$  mảng  $a$  tiền xử lý)

### Thuật toán tối ưu 2

Nhưng nếu dùng  $\log_2$  mảng  $a$  sẽ mang đến cho ta nhiều bất tiện (code dài, dễ sai, ...). Do đó, ta có thể đặt:

- $st[j][i]$  là giá trị nhỏ nhất của  $2^j$  phần tử tính từ  $i$  (min của  $a[i \dots i + 2^j - 1]$ ), tương ứng với  $a(2^j)[i]$  ( $st$  ở đây là viết tắt của  $S(\text{parse})T(\text{able})$ ).
- Ta có công thức truy hồi sau:

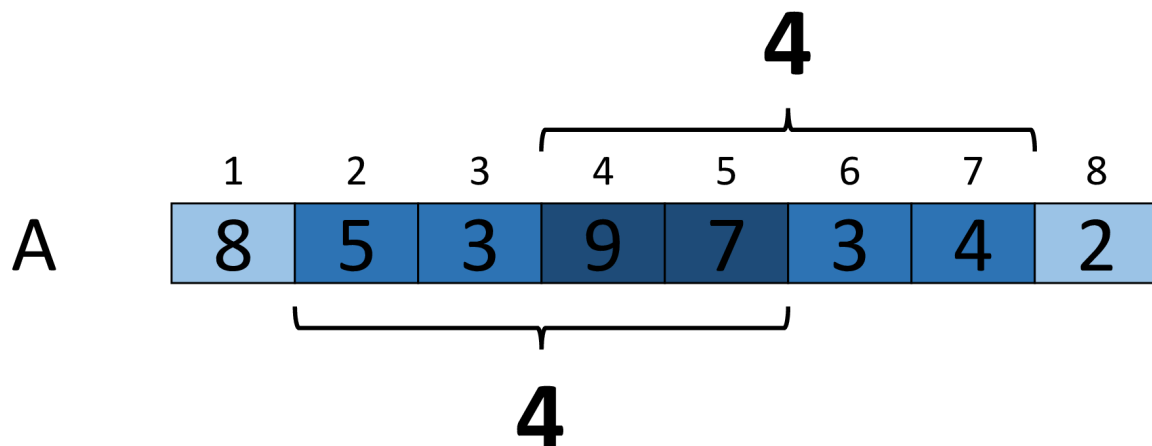
$$st[j][i] = \begin{cases} a[i] & \text{với } j = 0 \\ \min(st[j-1][i], st[j-1][i + 2^{j-1}]) & \text{với } j > 0 \end{cases}$$

Nhận xét thêm:

- log lệnh if trong *queryMin* lúc này thật ra chỉ thực hiện nhiệm vụ: tìm  $k$  nhỏ nhất thoả mãn  $len < 2^{k+1} = 2^k + 2^k$  (hay nói cách khác là để chắc chắn 2 đoạn  $[l \dots l + 2^k - 1]$  và  $[r - 2^k + 1, r]$ )

giao nhau nhưng vẫn nằm trong đoạn  $[l, r]$ .

## Đoạn cần truy vấn: $[2, 7]$



► Ví dụ:

- $len = 6 \Rightarrow k = 2$ , vì  $6 < 2^{k+1} = 8$
- $len = 8 \Rightarrow k = 3$ , vì  $8 < 2^{k+1} = 16$
- Vậy nên,  $k$  còn có một cách tính khác là  $k = \lfloor \lg(len) \rfloor$  (phần nguyên của phép  $\log_2(len)$ )
- Từ đây, ta có thể giảm độ phức tạp truy vấn xuống còn  $\mathcal{O}(1)$ !!!!!!

```

1 // LG là số lớn nhất thoả  $2^{LG} < N$ 
2 // ví dụ:  $N = 10^5$  thì  $LG = 16$  vì  $2^{16} = 65536$ 
3 int a[N], st[LG + 1][N];
4 void preprocess() {
5     for (int i = 1; i <= n; ++i) st[0][i] = a[i];
6     for (int j = 1; j <= LG; ++j)
7         for (int i = 1; i + (1 << j) - 1 <= n; ++i)
8             st[j][i] = min(st[j - 1][i], st[j - 1][i + (1 << (j - 1))]);
9 }
10 int queryMin(int l, int r) {
11     int k = __lg(r - l + 1);
12     return min(st[k][l], st[k][r - (1 << k) + 1]);
13 }

```

- Độ phức tạp tiền xử lý:  $\mathcal{O}(N \log N)$
- Độ phức tạp truy vấn:  $\mathcal{O}(1)$
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(N \log N + Q)$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(N \log N)$

## Range Sum Queries (RSQ)

### Bài toán



Cho mảng  $A$  gồm  $N$  phần tử và  $Q$  truy vấn có dạng  $(l, r)$ . Với mỗi truy vấn, in ra **tổng** các phần tử trong mảng  $A$  từ  $l$  đến  $r$ .

Giới hạn:  $N, Q \leq 10^5$

## Ý tưởng

Giống như RMQ, ta vẫn sẽ dựng mảng  $st[LG + 1][N]$ .

Nhưng lúc này, ta không thể lấy  $sum[l \dots r] = sum[l \dots l + 2^k - 1] + sum[r - 2^k + 1 \dots r]$  như RMQ được nữa (vì 2 đoạn chắc chắn giao nhau).

Nhận xét: Ta luôn có thể tách một số nguyên dương thành tổng các lũy thừa phân biệt của 2 (hệ nhị phân). Ví dụ:  $25 = 2^4 + 2^3 + 2^0 = 11001_2$ .

Từ nhận xét trên, ta có thể tách  $[l \dots r]$  thành  $\log_2$  đoạn có độ dài  $2^x$  như sau:

- Đặt  $len = r - l + 1$
- Duyệt  $j$  từ 0 đến  $\lfloor \lg(len) \rfloor$ , nếu bit thứ  $j$  của  $len$  là 1 thì:
  - Ta tách  $[l \dots r]$  thành  $[l \dots l + 2^j - 1]$  và  $[l + 2^j \dots r]$
  - $l = l + 2^j$  (tiếp tục tách  $[l + 2^j \dots r]$  như  $[l \dots r]$ )

## Cài đặt

```

1  // LG là số lớn nhất thoả  $2^{LG} < N$ 
2  // ví dụ:  $N = 10^5$  thì  $LG = 16$  vì  $2^{16} = 65536$ 
3  int a[N], st[LG + 1][N];
4  void preprocess() {
5      for (int i = 1; i <= n; ++i) st[0][i] = a[i];
6      for (int j = 1; j <= LG; ++j)
7          for (int i = 1; i + (1 << j) - 1 <= n; ++i)
8              st[j][i] = st[j - 1][i] + st[j - 1][i + (1 << (j - 1))];
9  }
10 int querySum(int l, int r) {
11     int len = r - l + 1;
12     int sum = 0;
13     for (int j = 0; (1 << j) <= len; ++j)
14         if (len >> j & 1) {
15             sum = sum + st[j][l];
16             l = l + (1 << j);
17         }
18     return sum;
19 }
```

- Độ phức tạp tiền xử lý:  $\mathcal{O}(N \log N)$
- Độ phức tạp truy vấn:  $\mathcal{O}(\log N)$
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(N \log N + Q \log N)$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(N \log N)$

## Lưu ý








Mình biết rằng bài này có cách dễ hơn là dùng [Mảng Cộng Dồn](#), nhưng lý do mảng cộng dồn sử dụng được là vì phép cộng có "phép đảo ngược" là phép trừ.

Nếu xét phép toán nhân 2 ma trận **không** vuông:

- ▶ Phép [nhân ma trận](#) không thoả tính chất Idempotence
- ▶ Ma trận **không** vuông không có ma trận nghịch đảo nên không có "phép đảo ngược"

Vậy bài toán lúc này không thể dùng mảng cộng dồn, cũng không thể dùng Sparse Table truy vấn  $\mathcal{O}(1)$ . Do đó, Sparse Table truy vấn  $\mathcal{O}(\log N)$  là một giải pháp hợp lý (dù khá tốn bộ nhớ so với Segment Tree).

## Bài tập áp dụng

- ▶ [Library Checker - Static RMQ](#) 
- ▶ [VNOJ - AVLBIT \(Dãy cấp số cộng\)](#) 
- ▶ [Codechef - FRMQ \(Chef and Array\)](#) 
- ▶ [Codeforces - 5C \(Longest Regular Bracket Sequence\)](#) 
- ▶ [Codeforces - 1454F \(Array Partition\)](#) 
- ▶ [Codeforces - 475D \(CGCDSSQ\)](#) 
- ▶ [Codeforces - 487B \(Strip\)](#) 

## Sparse Table 2D

Ta có bài toán như sau:

- ▶ Cho một ma trận 2 chiều độ lớn  $M \times N$  và  $Q$  truy vấn  $(x_1, y_1, x_2, y_2)$ . Với mỗi truy vấn, in ra giá trị nhỏ nhất trong ma trận con có góc trái dưới là  $(x_1, y_1)$  và góc phải trên là  $(x_2, y_2)$ .
- ▶ Giới hạn:
  - ▶  $M, N \leq 10^3$
  - ▶  $Q \leq 10^6$

Hiện tại, ta đang có 2 thuật toán:

- ▶ **Thuật toán ngây thơ:** Duyệt qua tất cả phần tử trong ma trận con.
  - ▶ Độ phức tạp tiền xử lý:  $\mathcal{O}(1)$
  - ▶ Độ phức tạp truy vấn:  $\mathcal{O}((x_2 - x_1 + 1) \times (y_2 - y_1 + 1)) = \mathcal{O}(M \times N)$
  - ▶ Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(Q \times M \times N)$
  - ▶ Độ phức tạp bộ nhớ:  $\mathcal{O}(M \times N)$
- ▶ **Thuật toán thông minh:** Với mỗi hàng, ta dựng một Sparse Table 1D.
  - ▶ Độ phức tạp tiền xử lý:  $\mathcal{O}(M \times N \log N)$
  - ▶ Độ phức tạp truy vấn:  $\mathcal{O}((x_2 - x_1 + 1) \times 1) = \mathcal{O}(M)$

- ▶ Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(Q \times M)$
- ▶ Độ phức tạp bộ nhớ:  $\mathcal{O}(M \times N \log N)$

Với giới hạn như trên, rõ ràng cả 2 thuật toán đều không đủ nhanh, và vấn đề nằm ở thời gian truy vấn còn chậm.

Để truy vấn nhanh hơn, ta có thể ứng dụng ý tưởng của Sparse Table 1D vào **thuật toán thông minh**:

- ▶ Xem mỗi Sparse Table 1D của mỗi hàng như 1 "nhóm" phần tử.
- ▶ Để gộp 2 "nhóm" phần tử, ta thực hiện gộp từng "phần tử" trong "nhóm".

	1	2	3	4	5	6
A	8	5	3	9	7	3
A2	5	3	3	7	3	
A4	3	3	3			
	1	2	3			
B	9	2	6	5	4	8
B2	2	2	5	4	4	
B4	2	2	4			
	1	2	3			

	1	2	3	4	5	6
A	8	5	3	9	7	3
A2	5	3	3	7	3	
A4	3	3	3			
	1	2	3			
B	9	2	6	5	4	8
B2	2	2	5	4	4	
B4	2	2	4			
	1	2	3			

	1	2	3	4	5	6
C	8	2	3	5	4	3
C2	2	2	3	4	3	
C4	2	2	3			

## Gộp 2 "nhóm" Sparse Table

Từ ý tưởng trên, ta xây dựng công thức như sau:

- Đặt  $st(k, i)(l, j)$  là giá trị nhỏ nhất của hình chữ nhật  $[i \dots i + 2^k - 1][j \dots j + 2^l - 1]$
- Khi  $k = 0$ , ta dựng Sparse Table 1D của hàng  $i$  (vì  $2^k = 1$ ), là  $st(0, i)$ :

$$st(0, i)(l, j) = \begin{cases} A[i][j] & \text{với } l = 0 \\ \min \{ st(0, i)(l-1, j), st(0, i)(l-1, j+2^{l-1}) \} & \text{với } l > 0 \end{cases}$$

- Khi  $k > 0$ , ta dựng  $st(k, i)$  bằng cách gộp  $st(k-1, i)$  và  $st(k-1, i+2^{k-1})$ .

Tóm lại, ta có công thức truy hồi như sau:

$$st(k, i)(l, j) = \begin{cases} A[i][j] & \text{với } k = 0 \text{ và } l = 0 \\ \min \{ st(k, i)(l-1, j), st(k, i)(l-1, j+2^{l-1}) \} & \text{với } k = 0 \text{ và } l > 0 \\ \min \{ st(k-1, i)(l, j), st(k-1, i+2^{k-1})(l, j) \} & \text{với } k > 0 \end{cases}$$

```

1  int a[M][N], st[LGM + 1][M][LGN + 1][N];
2  void preprocess() {
3      for (int k = 0; k <= LGM; ++k) {
4          for (int i = 1; i + (1 << k) - 1 <= m; ++i) {
5              for (int l = 0; l <= LGN; ++l) {
6                  for (int j = 1; j + (1 << l) - 1 <= n; ++j) {
7                      if (k == 0) {
8                          if (l == 0) {
9                              st[0][i][0][j] = a[i][j];
10                         }
11                         else {
12                             st[0][i][l][j] = min(st[0][i][l-1][j], st[0][i][l-1][j+2^{l-1}]);
13                         }
14                     }
15                     else {
16                         st[k][i][l][j] = min(st[k-1][i][l][j], st[k-1][i+2^{k-1}][l][j]);
17                     }
18                 }
19             }
20         }
21     }
22 }
```

Để truy vấn  $\min([x \dots a][y \dots b])$  trong  $\mathcal{O}(1)$  bằng Sparse Table 2D, ta làm tương tự như [Bài Toán RMQ](#), nhưng tách cả 2 chiều  $M$  và  $N$ , nghĩa là:

$$\begin{cases} k = \log_2(a - x + 1) \\ l = \log_2(b - y + 1) \\ \min([x \dots a][y \dots b]) = \min( \min([x \dots x + 2^k][y \dots y + 2^l]), \\ \min([x \dots x + 2^k][b - 2^l + 1 \dots b]), \\ \min([a - 2^k + 1 \dots a][y \dots y + 2^l]), \\ \min([a - 2^k + 1 \dots a][b - 2^l + 1 \dots b]) \end{cases}$$

```

1 | int getMin(int x, int y, int a, int b) {
2 |     int k = __lg(a - x + 1);
3 |     int l = __lg(b - y + 1);
4 |     return min( { st[k][x][l][y],
5 |                  st[k][x][l][b - (1 << l) + 1],
6 |                  st[k][a - (1 << k) + 1][l][y],
7 |                  st[k][a - (1 << k) + 1][l][b - (1 << l) + 1] } );

```

## Bài toán: Codechef - Chef and Rectangle Array

[Link đề bài tiếng Việt](#) 

Cho ma trận 2 chiều  $A$  có kích thước  $M \times N$ .

Một ma trận 2 chiều được gọi là "đẹp" nếu tồn tại một ma trận con  $a \times b$  sao cho tất cả phần tử đều bằng nhau. Trong một thao tác, ta có thể tăng giá trị của 1 phần tử trong ma trận  $A$  lên 1 đơn vị.

Có  $Q$  truy vấn, mỗi truy vấn gồm hai số  $a$  và  $b$ , ta cần in ra số thao tác tối thiểu để ma trận  $A$  được gọi là "đẹp".

Giới hạn:

- $1 \leq M, N, A_{ij} \leq 1000$
- $1 \leq Q \leq 50$
- $1 \leq a \leq M$
- $1 \leq b \leq N$

## Ý tưởng

- Duyệt qua tất cả các ma trận con  $a \times b$ , số thao tác tối thiểu để các phần tử của ma trận con  $a \times b$  bằng nhau là  $\max Value \times a \times b - \text{sum} Value$ .

## Phân tích độ phức tạp

- Độ phức tạp tiền xử lý:  $\mathcal{O}(MN \log M \log N)$
- Độ phức tạp truy vấn:  $\mathcal{O}(1)$
- Có  $Q$  truy vấn, vì thế tổng độ phức tạp thời gian là  $\mathcal{O}(MN \log M \log N + Q)$
- Độ phức tạp bộ nhớ:  $\mathcal{O}(MN \log M \log N)$

## Cài đặt

```

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

const int N = 1000 + 3, LG = 9;
int m, n;
int a[N][N];

int prf[N][N];
int st[LG + 1][N][LG + 1][N];
void preprocess() {
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            prf[i][j] = prf[i - 1][j] + prf[i][j - 1] - prf[i - 1][j - 1] + a[i][j];
        }
    }

    for (int k = 0; k <= LG; ++k) {
        for (int i = 1; i + (1 << k) - 1 <= m; ++i) {
            for (int l = 0; l <= LG; ++l) {
                for (int j = 1; j + (1 << l) - 1 <= n; ++j) {
                    if (k == 0) {
                        if (l == 0) {
                            st[0][i][0][j] = a[i][j];
                        }
                        else {
                            st[0][i][l][j] = max(st[0][i][l - 1][j], st[0][i][l][j - 1]);
                        }
                    }
                    else {
                        st[k][i][l][j] = max(st[k - 1][i][l][j], st[k - 1][i + (1 << k) - 1][l][j]);
                    }
                }
            }
        }
    }
}

int getSum(int x, int y, int a, int b) {
    return prf[a][b] - prf[a][y - 1] - prf[x - 1][b] + prf[x - 1][y - 1];
}

int getMax(int x, int y, int a, int b) {
    int k = __lg(a - x + 1);
    int l = __lg(b - y + 1);
    return max({ st[k][x][l][y],
                 st[k][x][l][b - (1 << l) + 1],
                 st[k][a - (1 << k) + 1][l][y],
                 st[k][a - (1 << k) + 1][l][b - (1 << l) + 1] });
}

int main() {
    cin >> m >> n;

```

```
51     for (int i = 1; i <= m; ++i) {
52         for (int j = 1; j <= n; ++j) {
53             cin >> a[i][j];
54         }
55     }
56     preprocess();
57
58     int q;
59     cin >> q;
60     while (q--) {
61         int a, b;
62         cin >> a >> b;
63
64         int ans = INT_MAX;
65         for (int i = a; i <= m; ++i) {
66             for (int j = b; j <= n; ++j) {
67                 int tmp = getMax(i - a + 1, j - b + 1, i, j) * a * b - getSum(i, j);
68                 ans = min(ans, tmp);
69             }
70         }
71         cout << ans << '\n';
72     }
73 }
```

## Bài tập áp dụng

- [Codechef - Chef and Rectangle Array](#) 

Được cung cấp bởi [Wiki.js](#)