BẢNG THƯA (Sparse Table)

1) Ứng dụng tìm Max, Min, GCD, Sum trên đoạn tịnh tiến trong thời gian *O*(1)

Kiến thức:

Ý tưởng chính **Sparse Table** là tính toán trước tất cả các câu trả lời cho các truy vấn trong phạm vi lũy thừa của **2**. Sau đó, một truy vấn phạm vi khác có thể được tính bằng cách chia phạm vi thành các phạm vi nhỏ hơn với độ dài là lũy thừa của **2**, tìm kiếm các câu trả lời được tính toán trước và kết hợp chúng lại để nhận được câu trả lời hoàn chỉnh.

Tính toán trước (Precomputation)

Sử dụng một mảng 2 chiều để lưu trữ các câu trả lời cho các truy vấn được tính toán trước. st[i][j] sẽ lưu trữ câu trả lời cho đoạn [**i, i + 2^j – 1**] có độ dài **2^j**. Kích thước của mảng 2 chiều sẽ là **MAXN x (K+1)**, trong đó **MAXN** là độ dài của mảng lớn nhất có thể và **K** nhỏ nhất thỏa mãn **K** ≥ ⌊log_2**MAXN**⌋. Ví dụ mảng có độ dài tối đa là 10^7 phần tử thì **K** = 25 là giá trị tốt nhất.

```
int st[MAXN][K + 1];
```

Bởi vì đoạn [***i, i + 2^j – 1***] độ dài 2^j chia đẹp nhất thành hai đoạn [***i, i + 2^{j-1} – 1***] và [***i +2^{j-1} , i + 2^j – 1***] có độ dài ***2^{j-1}*** nên cách tính toán hiệu quả nhất là dùng quy hoạch động:

```
for (int i = 0; i < N; i++)
    st[i][0] = f(array[i]);

for (int j = 1; j <= K; j++)
    for (int i = 0; i + (1 << j) <= N; i++)
        st[i][j] = f(st[i][j-1], st[i + (1 << (j - 1))][j - 1]);
```

Hàm **f** phụ thuộc vào loại truy vấn. Nếu truy vấn **tổng** thì hàm **f** tính **tổng**, nếu truy vấn **min** thì hàm **f** là tính **min**.

- ### Truy vấn Sum:

Truy vấn này có dạng cần tính tổng tất cả các phần tử liên tiếp trong một đoạn. Hàm ***f*** được định nghĩa ***f*(x, y) = x + y.** Chúng ta có thể xây dựng Bảng thưa như sau:

```
long long st[MAXN][K + 1];

for (int i = 0; i < N; i++)
    st[i][0] = f(array[i]);

for (int j = 1; j <= K; j++)
    for (int i = 0; i + (1 << j) <= N; i++)
        st[i][j] = st[i][j-1] + st[i + (1 << (j - 1))][j - 1];
```

Khi đó tổng các phần tử trong đoạn **[L, R]** bằng tổng các phần tử trong đoạn có độ dài lũy thừa 2 giảm dần từ (**R – L + 1**), bắt đầu từ đoạn [**L, L + 2^j - 1]** và tiếp tục với các đoạn con còn lại [**L + 2^j, R]:**

```
long long sum = 0;
for (int j = K; j >= 0; j--) {
    if ((1 << j) <= R - L + 1) {
        sum += st[L][j];
        L += 1 << j;
    }
}
```

Độ phức tạp của truy vấn loại này là **O(K)** = ***O***(log **MAXN**)

- **Truy vấn Min/Max:**

Khi tính toán giá trị nhỏ nhất của một đoạn phần tử, kết quả không thay đổi nếu chúng ta xử lý một giá trị trong đoạn đó một lần hoặc hai lần. Do đó, thay vì chia một đoạn thành nhiều đoạn con, chúng ta cũng có thể chia đoạn thành hai đoạn con giao nhau có độ dài lũy thừa **2**. Ví dụ chúng ta có thể chia đoạn **[1, 6]** thành hai đoạn **con [1, 4]** và **[3, 6].** Giá trị nhỏ nhất trong đoạn **[1, 6]** cũng giống giá trị nhỏ nhất của giá trị nhỏ nhất đoạn **[1, 4]** và giá trị nhỏ nhất đoạn **[3, 6].** Vì vậy, chúng ta có thể tính toán giá trị nhỏ nhất của đoạn **[L, R]** như sau:

```
min(st[L][j], st[R - 2^j + 1][j])    với j = log2(R − L + 1)
```

Hàm ***f*** được định nghĩa ***f*(x, y) =min(x,y).** Chúng ta có thể xây dựng Bảng thưa như sau:

```
int st[MAXN][K + 1];
for (int i = 0; i < N; i++)
    st[i][0] = array[i];
```

```
for (int j = 1; j <= K; j++)
    for (int i = 0; i + (1 << j) <= N; i++)
        st[i][j] = min(st[i][j-1], st[i + (1 << (j - 1))][j-
1]);
```

Khi đó giá trị nhỏ nhất của đoạn **[L, R]** được tính toán như sau:

```
int j = lg[R - L + 1];
int minimum = min(st[L][j], st[R - (1 << j) + 1][j]);
```

Độ phức tạp của truy vấn này là ***O*(1)**

- **Truy vấn GCD:** thực hiện tương tự như truy vấn Min/Max.