MẢNG CỘNG DỒN

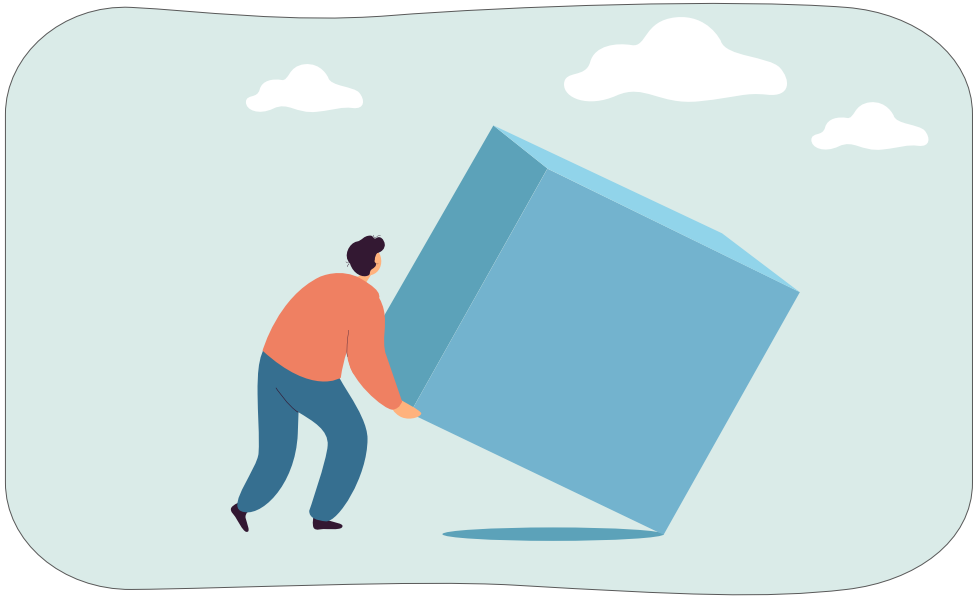
Mảng cộng dồn (tiền tố) là một mảng giúp các bạn có thể nhanh chóng tính toán tổng các phần tử trong các đoạn liên tiếp từ chỉ số left tới chỉ số right.

1. Mảng cộng dồn trên mảng một chiều:

Đặt vấn đề: Cho mảng A[ ] có N phần tử, có Q truy vấn, mỗi truy vấn yêu cầu bạn tính tổng các phần tử từ chỉ số left tới chỉ số right.

Cách giải thông thường với mỗi truy vấn (O(n))

```
int left, right;
cin >> left >> right;
int sum = 0;
for(int i=left; i<=right; i++){
        sum += A[i];
}
```



1. Mảng cộng dồn trên mảng một chiều:

Dùng mảng cộng dồn với mỗi truy vấn (O(1))

Gọi mảng pre[ ] là mảng cộng dồn của mảng A[ ].

Khi đó pre[i] = A[0] + A[1] + A[2] + … + A[i] sẽ lưu tổng các phần tử từ chỉ số 0 tới chỉ số i của mảng A[ ]

Ta có thể pre[i] thông qua pre[i - 1] : pre[i] = pre[i - 1] + A[i]

Xây dựng mảng cộng dồn (O(n))

```
int pre[n];
for(int i = 0; i < n; i++){
   if(i == 0)
        pre[i] = A[i];
   else
        pre[i] = pre[i - 1] + A[i];
}
```

1. Mảng cộng dồn trên mảng một chiều:

Để tính tổng các phần tử từ chỉ số left tới chỉ số right ta lấy pre[right] - pre[left -1], chú ý trường hợp left = 0.

A[]	4	2	3	1	5	6
Pre[]	4	6	9	10	15	21

Ví dụ áp dụng


```
int left, right;
cin >> left >> right;
if(left == 0){
   cout << pre[right] << endl;
}
else{
   cout << pre[right] - pre[left - 1] << endl;
}
```

2. Mảng cộng dồn trên mảng hai chiều:

Đối với mảng 2 chiều, khi muốn tính tổng các phần tử trong phạm vi của 1 hình chữ nhật có N hàng và M cột bạn cần lặp qua N hàng mỗi hàng duyệt qua M cột để tính tổng, độ phức tạp sẽ là O(N*M).

Giả sử mảng 2 chiều ban đầu như sau, ở đây để tiện mình dùng chỉ số hàng, cột của mảng bắt đầu từ 1. Các hàng 0, cột 0 của mảng 2 chiều bằng 0.

0	0	0	0	0	0
0	3	1	2	3	4
0	5	2	1	7	6
0	2	1	2	2	3
0	4	6	5	9	2

Ví dụ: Tính tổng các phần tử màu xanh

Code ngây thơ O(N*M)

EXAMPLE

```
int h1, h2; // hàng 1, hàng 2
int c1, c2; // cột 1, cột 2
int sum = 0;
for(int i = h1; i <= h2; i++){
   for(int j = c1; j <= c2; j++){
      sum += a[i][j];
   }
}
```

2. Mảng cộng dồn trên mảng hai chiều:

Cải tiến: Nhận thấy để tính tổng các phần tử trên hàng 1 từ cột 1 tới cột 2 bạn có thể dùng mảng cộng dồn cho từng hàng, khi đó bạn chỉ cần duyệt qua từng hàng và dùng mảng cộng dồng cho từng hàng của ma trận để tính nhanh tổng các phần tử trên hàng đó.

0	0	0	0	0	0
0	3	4	6	9	13
0	5	7	8	15	21
0	2	3	5	7	10
0	4	10	15	24	26

Mảng cộng dồn của từng dòng

2. Mảng cộng dồn trên mảng hai chiều:

Mảng cộng dồn trên mảng 2 chiều, giả sử mảng 2 chiều của bạn có n hàng và m cột. Bạn cần tính tổng các phần tử trên HCN con bắt đầu từ hàng 1 tới hàng a và từ cột 1 tới cột b:

$$\texttt{prefix}[a][b] = \sum_{i=1}^{a}\sum_{j=1}^{b}\texttt{arr}[i][j].$$

Công thức tính prefix[a][b]

```
prefix[i][j] = prefix[i-1][j] + prefix[i][j-1] - prefix[i-1][j-1] + arr[i][j]
```

Từ đó khi muốn tính tổng các phần tử trong HCN bắt đầu từ hàng a kết thúc ở hàng A, bắt đầu từ cột b và kết thúc ở cột B ta chỉ mất O(1).

$$\sum_{i=a}^{A}\sum_{j=b}^{B}\texttt{arr}[i][j] = \texttt{prefix}[A][B] - \texttt{prefix}[a-1][B] - \texttt{prefix}[A][b-1] + \texttt{prefix}[a-1][b-1]$$

2. Mảng cộng dồn trên mảng hai chiều:

Xây dựng mảng cộng dồn

```
int n, m; // hàng, cột
int prefix[n + 1][m + 1];
memset(prefix, 0, sizeof(prefix));
for(int i = 1; i <= n; i++){
   for(int j = 1; j <= m; j++){
      prefix[i][j] = prefix[i - 1][j] + prefix[i][j - 1] - prefix[i - 1][j - 1] + a[i][j];
   }
}
```

Truy vấn

```
int a, A, b, B;
cin >> a >> A >> b >> B;
cout << prefix[A][B] - prefix[a - 1][B] - prefix[A][b - 1] + prefix[a - 1][b - 1];
```

3. Mảng hiệu (Difference Array):

Đặt vấn đề: Cho mảng A[ ] có N phần tử, có Q thao tác mỗi thao tác sẽ tăng các phần tử trong đoạn từ chỉ số L tới R của mảng A[ ] lên K đơn vị. Hãy xác định mảng A[ ] sau Q thao tác.

Cách tiếp cận đơn giản cho vấn đề đó là đối với mỗi truy vấn bạn sẽ duyệt từ L tới R và thêm K vào các phần tử trong mảng A[ ], như vậy với mỗi truy vấn bạn sẽ mất O(N).

3. Mảng hiệu (Difference Array):

Cách tối ưu hơn

- Có một cấu trúc dữ liệu hiệu quả hơn để giải quyết problem trên, với mỗi truy vấn bạn chỉ mất O(1).
- Gọi mảng D[] là mảng hiệu của mảng A[], trong đó:
  - D[0] = A[0]
  - D[i] = A[i] - A[i - 1] với i ≥1
    Khi đó để khôi phục mảng A[] từ mảng D[] ta chỉ cần tính mảng cộng dồn của mảng D[].

Code xây dựng mảng D[ ]

EXAMPLE

```
int n; cin >> n;
int a[n];
for(int &x : a) cin >> x;
int D[n + 5];
for(int i = 0; i < n; i++){
   if(i == 0)
      D[i] = a[i];
   else
      D[i] = a[i] - a[i - 1];
}
```

3. Mảng hiệu (Difference Array):

MẢNG A:	3	1	8	7	6	2

MẢNG D:	3	-2	7	-1	-1	-4

Từ mảng D bạn có thể tìm ra A[i] bằng cách tính tổng các phần tử từ chỉ số 0 tới chỉ số i của mảng D. Ví dụ : A[3] = D[0] + D[1] + D[2] + D[3] = 3 + (-2) + 7 +(-1) = 7

Code cập nhật mảng D[ ] với truy vấn

```
D[L] += K;
D[R + 1] -= K;
```

3. Mảng hiệu (Difference Array):

Giải thích: D[L] += K sẽ làm tất cả các phần tử từ chỉ số L tới chỉ số N - 1 của mảng A[] tăng lên K đơn vị, vì bạn biết rằng A[i] tính bằng cách cộng các phần tử từ chỉ số 0 tới chỉ số i của mảng D, vì thế D[L] += K sẽ ảnh hưởng tới mọi phần tử tính từ chỉ số L, tuy nhiên bạn chỉ muốn cập nhật cho các phần tử từ chỉ số L tới R, vì thế những phần tử từ R + 1 tới N - 1 cần được trừ đi K đơn vị, khi đó bạn giảm D[R + 1] đi K đơn vị.

Khôi phục mảng D[ ] sau Q truy vấn

```
for(int i = 0; i < n; i++){
   if(i == 0) a[i] = D[i];
   else a[i] = D[i] + a[i - 1];
}
```

3. Mảng hiệu (Difference Array):

Code hoàn thiện cho problem

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

int main(){
    int n; cin >> n;
    int a[n];
    for(int &x : a) cin >> x;
    int D[n + 5];
    for(int i = 0; i < n; i++){
        if(i == 0) D[i] = a[i];
        else D[i] = a[i] - a[i - 1];
    }
```

```
    int q; cin >> q;
    while(q--){
        int l, r, k; cin >> l >> r >> k;
        D[l] += k;
        D[r + 1] -= k;
    }
    for(int i = 0; i < n; i++){
        if(i == 0) a[i] = D[i];
        else a[i] = D[i] + a[i - 1];
        cout << a[i] << ' ';
    }
}
```

INPUT

```
6
3 1 8 7 6 2
2
2 3 4
1 4 3
```

OUTPUT

```
3 4 15 14 9 2
```