MẢNG - Lý thuyết

Ban chuyên môn Tin học – The Gifted Battlefield

Xuất bản vào Ngày 16 tháng 10 năm 2023

Mục lục

1 Lời nói đầu

Nhu cầu lưu trữ dữ liệu thành nhóm là một nhu cầu rất cơ bản trong lập trình. Trong C++, **Mảng** là một kiểu biến cho phép lưu nhiều giá trị thành nhóm, được biểu diễn dưới dạng 1 dãy các ô nhớ liên tiếp nhau.

Bài viết sẽ giới thiệu các khái niệm cơ bản xoay quanh mảng, các lý thuyết cần nắm và cách cài đặt, sử dụng mảng trong C++.

Một số kiến thức tiền đề: biến, hàm, vòng lặp for, thư viện bits/stdc++.h. Ngoài ra, tác giả khuyến khích người đọc nắm được khái niệm cơ bản về ***con trỏ*** (được trình bày sơ lược bên dưới) để tìm hiểu về mảng một cách thuận lợi hơn.

2 Kiến thức cơ bản về mảng

2.1 Mảng

Mảng là một cấu trúc dữ liệu cơ bản, biểu diễn 1 dãy **hữu hạn** và **có chiều dài cố định** các phần tử cùng kiểu dữ liệu. Mỗi phần tử được truy xuất (bao gồm thao tác đọc và ghi) thông qua **chỉ số** của nó trong mảng.

Cơ bản

Định nghĩa - lý thuyết con trỏ

- Mảng là một dãy bộ nhớ liên tiếp, do đó, phần tử A_i của mảng có địa chỉ trong bộ nhớ máy tính sau A_0 (là phần tử đầu tiên của mảng) i đơn vị. Từ ý tưởng này, máy tính chỉ cần biết được độ lớn và địa chỉ của phần tử đầu tiên trong mảng để quản lý nó. Trong C++, khi khai báo mảng, ta đã cho chương trình độ lớn của mảng, còn về sau thì được quản lý bằng 1 khái niệm mới: **con trỏ**.
- Mọi dữ liệu trong chương trình được quản lý bằng **con trỏ**. Xét riêng mảng, chương trình sẽ lưu lại địa chỉ phần tử đầu của mảng và với mỗi truy vấn đến phần tử A_i, dịch con trỏ đi i ô dữ liệu. Cũng như với con trỏ bình thường, con trỏ đến mảng cũng có thể dùng để biết được điểm đầu và cuối, **VD:** hàm $sort(a, a + len)$. *Tìm hiểu thêm:* **Con trỏ trong C/C++**

Các loại mảng và cách sử dụng

- Ta có thể khai báo mảng với mọi kiểu dữ liệu cơ bản, bao gồm: *int*, *char*, *string*, *bool*,... Các mảng này đều có các đặc tính và ứng dụng như nhau. Mở rộng hơn, những kiểu dữ liệu do người dùng tự định nghĩa bằng *struct* hay *class* đều có thể xây dựng mảng.

```
//Khai bao mang
int int_array[3];
char char_array[3];
string string_array[3];
bool bool_array[3];
...
// kieu_du_lieu ten_mang[do_lon];
```

LƯU Ý: Khi khai báo mảng *global* - nằm bên ngoài các hàm (bao gồm hàm *main*), mảng mặc định có giá trị 0 với mọi phần tử.

Truy cập vào mảng và giới hạn của mảng

– Trong C++, phần tử đầu tiên của mảng là 0 và phần tử cuối cùng là $len - 1$. len là độ dài khi khai báo mảng.

```
// * TRUY CẬP MẢNG KHÔNG HỢP LỆ
int arr[5];
arr[-1] = 0;
arr[5] = 0;
arr[6] = 0;
```

Nhập xuất dữ liệu

– Mỗi phần tử của mảng *arr* đều có thể thao tác như với một biến có cùng kiểu dữ liệu.

```
// * NHẬP DỮ LIỆU VÀO MẢNG
int arr[4] = {0, 10, 3, 7}
arr[0] = 256;                   // arr = {256, 10, 3, 7}
// bằng vòng lặp
for (int i = 0; i <= 3; i++)
    arr[i] = i;                 // arr = {0, 1, 2, 3}

// * XUẤT MẢNG
cout << "in mang arr: ";
cout << arr[0] << ' ';
cout << arr[1] << ' ';
cout << arr[2] << ' ';
cout << arr[3] << '\n';
// bằng vòng lặp
cout << "in mang arr bang for: ";
for (int i = 0; i < 4; i++)
    cout << arr[i] << ' ';
cout << '\n';
```

Output:

```
mang arr: 0 1 2 3
mang arr bang for: 0 1 2 3
```

■ Nâng cao

Sắp xếp tăng, giảm dần mảng

– Thư viện mặc định của C++ cho người dùng hàm *sort* để sắp xếp theo 1 thứ tự nhất định:

```cpp
int arr = {0, 1, 100, 13};

// arr = {0, 1, 100, 13}
sort(arr, arr + 3);
// arr = {0, 1, 13, 100}
// thứ tự mặc định của hàm std::sort là tăng dần

cout << "in mang arr tang dan: ";
for (int i = 0; i < 4; i++)
    cout << arr[i] << ' ';
cout << '\n';

// Sắp xếp giảm dần
// arr = {0, 1, 13, 100}
sort(arr, arr + 3, greater<int>());
// arr = {100, 13, 1, 0}
cout << "in mang arr giam dan: ";
for (int i = 0; i < 4; i++)
    cout << arr[i] << ' ';
```

Output:

```
mang arr tang dan:  0 1 13 100
mang arr giam dan: 100 13 1 0
```

Mảng đa chiều

– Nếu coi chính mảng là một cấu trúc dữ liệu, thì cách khai báo, truy cập, và thay đổi mảng đa chiều cũng tương tự như với mảng 1 chiều.

```cpp
// * KHAI BÁO MẢNG ĐA CHIỀU
// kieu_du_lieu ten_mang[kich_co_1][kich_co_2]...[kich_co_N];
// VD với mảng 2 chiều:

int arr2d[3][4];
// * NHẬP GIÁ TRỊ
for(int i = 0; i < 3; i++)
    for(int j = 0; j < 4; j++)
        cin >> arr2d[i][j];

// * TRUY CẬP MẢNG ĐA CHIỀU
cout << "in mang arr2d: \n";
for(int i = 0; i < 3; i++){
    for(int j = 0; j < 4; j++){
        cout << arr2d[i][j] << ' ';
    }
    cout << '\n';
}
```

```cpp
// thay đổi giá trị tại vị trí (0, 0)
arr2d[0][0] = 100;
cout << "in mang arr2d thay doi: \n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
        cout << arr2d[i][j] << ' ';
    }
    cout << '\n';
}
```

Output:

```
In mang arr2d:
0 1 2 3
4 5 6 7
8 9 10 11
In mang arr2d thay doi:
100 1 2 3
4 5 6 7
8 9 10 11
```

Các cách khai báo mảng hằng

– Ngoài lấy *input* từ người dùng bằng *cin*, ta còn có các cách khai báo cho mảng hằng như sau:

```cpp
// * CÁC CÁCH KHAI BÁO MẢNG 1D

int arr[4] = { 5, 8, 2, 7 };                // arr = {5, 8, 2, 7}
char arr1[4] = { 'a', 'c' };                // arr1 = {'a', 'c', char(0), char(0)}
bool arr2[4] = { };                         // arr2 = {false, false, false, false}
string arr3[] { "I", "love", "you", "<3" }; // arr3 = { "I", "love", "you", "<3" }
int arr4[4];                                // arr4 = {random, random, random, random}
// random là giá trị chương trình gán ngẫu nhiên cho mảng

// * KHAI BÁO MẢNG ĐA CHIỀU
int arr2d[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
// hoặc
int arr2d[3][4] = {
 {0, 1, 2, 3},
 {4, 5, 6, 7},
 {8, 9, 10, 11}
};
```

2.2 Mảng động (vector)

Việc sử dụng mảng cần phải có kích thước nhất định để cấu tạo, điều đó có thể tạo nên một số bất lợi khi lập trình. Qua đó, cấu trúc dữ liệu mảng động ra đời và `vector` là 1 dạng mảng động.

Điều khác biệt ở `vector` so với mảng thông thường là cách hoạt động của số lượng phần tử của mảng. Ta không cần một kích thước nhất định cho trước, mà ta có thể thêm vào từng phần tử một ở phía sau của `vector`, mảng này sẽ được kéo dài ra để phù hợp với số lượng phần tử người dùng thêm vào. Điều này có vẻ tiện lợi, nhưng `vector` và các cấu trúc dữ liệu mảng động nói chung luôn có lượng thời gian xử lý lâu hơn một mảng cố định mặc dù không đáng kể.

Cơ bản

Khai báo và kiểu dữ liệu vector

– Trong `vector`, ta có thể tùy chỉnh các kiểu dữ liệu cơ bản như *int*, *char*, *double*,... Và điều chỉnh độ dài ban đầu hay các phần tử ban đầu của mảng bằng thao tác sau:

```
vector<int> int_vector;
vector<bool> bool_vector;
vector<char> char_vector;
vector<int> sized_vector(5);
vector<int> modified_vector = {3, 5, 2, 6};
...
// vector<kieu_du_lieu> ten_vector;
```

Nhập và xóa dữ liệu trong vector

– Vì `vector` là một mảng động, nên việc nhập thêm phần tử cho mảng sẽ được mặc định là đưa về sau cùng của `vector` trước khi nhập, đối với `vector` chưa có phần tử nào, phần tử được nhập vào sẽ là phần đầu tiên của nó. Xóa dữ liệu ở `vector` cũng vậy, hàm cơ bản sẽ xóa phần tử cuối cùng của nó:

```
vector<int> newVector;

newVector.push_back(3); // newVector = {3}
newVector.push_back(5); // newVector = {3, 5}

newVector.pop_back(); // newVector = {3}
newVector.pop_back(); // newVector = {}

// ten_vector.push_back(gia_tri);
// ten_vector.pop_back();
```

Truy cập phần tử của vector

– Truy cập phần tử là mục đích chính cho một `vector` tồn tại, chúng ta tạo ra `vector` để chứa dữ liệu và truy cập vào những dữ liệu đó lúc sau. Có hai cách truy cập dữ liệu của `vector`, một cách sử dụng kí hiệu và cách còn lại dùng hàm. Tuy nhiên, việc truy cập phần tử có vị trí lớn hơn kích thước của `vector` hoặc bé hơn 0 (là vị trí phần tử đầu tiên) sẽ luôn trả về giá trị rỗng của kiểu dữ liệu cấu tạo, và với điều kiện là mảng đó phải được gán ít nhất một phần tử trước nếu không sẽ trả về lỗi và làm cho chương trình ngưng hoạt động hoàn toàn.

```cpp
vector<int> int_vector = {3};
vector<char> char_vector = {'a', 'b'};

cout << "Hai cach truy cap du lieu cua vector la: ";
cout << int_vector[0] << ' ' << int_vector.at(0) << '\n';

cout << "Truy cap du lieu o vector char: ";
cout << char_vector[0] << ' ' << char_vector.at(1) << '\n';

cout << "Du lieu ngoai kich thuoc cua vector loai int: ";
cout << int_vector[-1] << '\n';

cout << "Du lieu ngoai kich thuoc cua vector loai char: ";
cout << char_vector[-1] << '\n';

// * ĐOẠN CODE DƯỚI ĐÂY BIỂU DIỄN MỘT LỖI CHƯƠNG TRÌNH, ĐƯỢC ĐƯA RA ĐỂ TRÁNH LỖI NÀY.
/*
vector<int> empty_vector;
cout << empty_vector[-1];
*/
/* Chương trình sau khi nhập những đoạn code trên (không phải comment) sẽ hoàn toàn
không thể chạy.*/

// ten_vector[vi_tri];
// ten_vector.at(vi_tri)
```

Output:

```
Hai cach truy cap du lieu cua vector la: 3 3
Truy cap du lieu o vector char: a b
Du lieu ngoai kich thuoc cua vector loai int: 0
Du lieu ngoai kich thuoc cua vector loai char:
```

Kích thước của vector và làm rỗng vector

– Ở đa số trường hợp sử dụng `vector`, chúng ta sẽ cần biết về số lượng phần tử chứa trong đó, kích thước của `vector` cũng thay đổi theo số lượng phần tử được nhập vào hay xóa đi. Làm rỗng `vector` cũng là một trong những kiến thức cơ bản khi sử dụng, việc làm rỗng dường như không được sử dụng nhiều, nhưng ta có thể tận dụng nó để dùng lại `vector` đã được khai báo như một `vector` mới, điều này giúp tiết kiệm dữ liệu cho chương trình.

```cpp
vector<int> newVector = {1, 2, 3, 4, 5};

cout << newVector.size() << '\n';
// newVector.size() sẽ trả về 5

newVector.clear();
// newVector = {};
```

```
cout << newVector.size();
// newVector.size() sẽ trả về 0

// ten_vector.size();
// ten_vector.clear();
```

– **Output:**

```
5
0
```

■ Một số hàm cơ bản ở vector

– Nhập giá trị phần tử:

Hàm	**Công dụng**
push_back()	Tạo phần tử ở sau cùng của vector và nhập vào giá trị được đưa vào ở đó
pop_back()	Loại bỏ phần tử cuối cùng của vector
assign()	Thay thế vector cũ thành một vector mới với số lượng giá trị đã cho
insert()	Nhập vào giá trị cho phần tử ở vị trí được cho và đẩy các phần tử khác ra sau

– Truy cập giá trị phần tử:

Hàm	**Công dụng**
operator [ ]	Truy cập vào phần tử tại vị trí được cho của vector "operator"
at()	Truy cập vào phần tử tại vị trí được cho của vector
front()	Đưa ra giá trị phần tử đầu tiên của vector
back()	Đưa ra giá trị phần tử cuối cùng của vector

– Thay đổi các phần tử của vector:

Hàm	Công dụng
erase()	Xóa một phần tử ở vị trí nhất định hoặc xóa một khoảng các phần tử được chỉ định trong vector
empty()	Kiểm tra nếu có phần tử trong vector, trả về kiểu dữ liệu bool
size()	Trả về số lượng phần tử hiện tại của vector
clear()	Xóa tất cả dữ liệu của vector

– Vòng lặp trong vector:

Hàm	Công dụng
begin()	Trả về con trỏ cho các phần tử của vector cho vòng lặp chạy từ đầu đến cuối
end()	Trả về con trỏ cho các phần tử của vector cho vòng lặp chạy từ cuối lên đầu
rbegin()	Trả về con trỏ cho các phần tử của vector cho vòng lặp chạy ngược từ cuối lên đầu
rend()	Trả về con trỏ cho các phần tử của vector cho vòng lặp chạy ngược từ đầu đến cuối

Ví dụ sử dụng hàm vector

```cpp
#include <bits/stdc++.h>

using namespace std;

int main(){
    vector<int> exVector; // Tạo vector tên exVector với các phần tử có giá trị int

    exVector.push_back(727); // exVector = {727}
    for (int i = 1; i <= 4; i++)
        exVector.push_back(i); // exVector = {727, 1, 2, 3, 4}
    // Một ví dụ nhập dữ liệu cho vector trong vòng lặp

    exVector.pop_back(); // exVector = {727, 1, 2, 3}

    exVector.assign(3, 15); // exVector = {15, 15, 15}

    exVector.insert(exVector.begin(), 5); // exVector = {5, 15, 15, 15}
    // In ra phần tử đầu và cuối của exVector ở stdout
    cout << "Phan tu dau va cuoi co gia tri: "
         << exVector.front() << ' ' << exVector.back();
```

```
    exVector.erase(exVector.begin()); // exVector = {15, 15, 15}

    // In ra giá trị phần tử thứ 1 trong vector
    cout << "\nPhan tu thu 1 co gia tri: "
         << exVector[1] << ' ' << exVector.at(1);

    cout << "\nSo luong phan tu cua exVector: "
         << exVector.size();

    cout << "\nKiem tra exVector rong: "
         << exVector.empty();

    // Xóa hết dữ liệu của exVector
    exVector.clear();

    cout << "\nKiem tra exVector rong: "
         << exVector.empty();

    cout << "\n1 la vector rong 0 la vector chua gia tri";

    for (int i = 1; i <= 5; i++)
        exVector.push_back(i);

    cout << "\nCac vong lap qua vector: ";
    cout << "\nVong lap thuong: ";
    for (auto i = exVector.begin(); i != exVector.end(); i++)
        cout << *i << ' ';
    cout << "\nVong lap nguoc: ";
    for (auto ir = exVector.rbegin(); ir != exVector.rend(); ir++)
        cout << *ir << ' ';
}
```

– **Output:**

```
Phan tu dau va cuoi co gia tri: 5 15
Phan tu thu 1 co gia tri: 15 15
So luong phan tu cua exVector: 3
Kiem tra exVector rong: 0
Kiem tra exVector rong: 1
1 la vector rong 0 la vector chua gia tri
Cac vong lap qua vector:
Vong lap thuong: 1 2 3 4 5
Vong lap nguoc: 5 4 3 2 1
```

■ Nâng cao

Các cấu trúc dữ liệu trên vector

– Kiểu dữ liệu của `vector` không chỉ giới hạn ở những dữ liệu cơ bản, mà ta còn có thể sử dụng các cấu

trúc dữ liệu khác có sẵn hoặc thậm chí chính mình tạo ra để tạo nên một mảng động của cấu trúc dữ liệu đó. Ví dụ như mảng đa chiều động, mảng cặp động,... Các thao tác trên mảng động mỗi loại cấu trúc dữ liệu là khác nhau, nên code ở dưới chỉ là cách khai báo cấu trúc dữ liệu động đó, chi tiết sẽ tùy thuộc vào cấu trúc dữ liệu được dùng.

```
vector<vector<int>> new2D_vector;
vector<vector<vector<int>>> new3D_vector;
vector<pair<int,int>> pair_vector;
vector<set> set_vector;
...
```

Sắp xếp vector

– Sắp xếp `vector` là một trong những thao tác được áp dụng nhiều nhất trong các bài toán. Sắp xếp `vector` khác một mảng cố định vì `vector` là mảng động, sau đây là cách sắp xếp tăng dần và giảm dần một `vector` sử dụng hàm *std* :: *sort*:

```
vector<int> sorting_vector = {5, 3, 6, 2, 4, 1};
cout << "Truoc khi sap xep: ";
for (auto i = sorting_vector.begin(); i != sorting_vector.end(); i++)
    cout << *i << ' ';
cout << '\n';

sort(sorting_vector.begin(), sorting_vector.end());
cout << "Sau khi sap xep tang dan: ";
for (auto i = sorting_vector.begin(); i != sorting_vector.end(); i++)
    cout << *i << ' ';
cout << '\n';

sort(sorting_vector.begin(), sorting_vector.end(), greater<int>());
cout << "Sau khi sap xep giam dan: ";
for (auto i = sorting_vector.begin(); i != sorting_vector.end(); i++)
    cout << *i << ' ';
```

– **Output:**

```
Truoc khi sap xep: 5 3 6 2 4 1
Sau khi sap xep tang dan: 1 2 3 4 5 6
Sau khi sap xep giam dan: 6 5 4 3 2 1
```

3 Những ứng dụng cơ bản của mảng

3.1 Mảng đếm (Mảng thống kê)

Mảng đếm (hoặc mảng thống kê) là một mảng được tạo ra để thống kê tần số xuất hiện của các phần tử trong một mảng. Nghe có vẻ đơn giản nhưng mảng đếm thường được sử dụng như một bài toán con trong

một bài lập trình lớn.

Ta phát biểu bài toán sau:

Ví dụ 3.1

Cho mảng a với độ dài n $(1 \leq n \leq 10^5)$ chứa các phần tử $a_1, a_2, a_3, \cdots, a_n$ $(0 \leq a_i \leq 10^5; i \leq n)$. Hãy cho biết số loại phần tử trong mảng và tần suất xuất hiện của phần tử đó.

Cài đặt mảng đánh dấu:

- Ta tạo một mảng cnt có $10^5 + 1$ phần tử, mỗi phần tử có giá trị ban đầu là 0 (giá trị $cnt[i]$ có ý nghĩa là số lần xuất hiện của i), sau đó đọc từng giá trị của mảng được cho vào và tăng 1 giá trị tại vị trí của phần tử đó. Sau khi đánh dấu tất cả giá trị, ta in ra những giá trị tần suất lớn hơn 0 là những giá trị tồn tại ở đầu vào.
- Code:

```
void solve(){
    int n;
    cin >> n;
    int cnt[100001]; // Khởi tạo mảng đánh dấu
    int max = 0;
    for (int i = 0; i < n; i++){
        int a;
        cin >> a;
        if (a > max) max = a;
        cnt[a]++; // Tăng tần suất cho loại giá trị a;
    }
    for (int i = 0; i <= max; i++)
        if (cnt[i] > 0) // Chỉ xét các phần tử có xuất hiện trong mảng a[]
            cout << "Phan tu " << i << " xuat hien " << cnt[i] << " lan\n";
}
```

- Độ phức tạp : $O(n + max(a_i))$.

3.2 Mảng tiền tố, mảng hậu tố

Mảng tiền tố và mảng hậu tố là các mảng sử dụng những giá trị mảng đã được tính trước đó để tính phần tử đang xét hiện tại thuộc mảng.

Mảng tiền tố

Ví dụ, ta đang tính hàm f là phép biến đổi từ mảng a thông qua phép $\circ$ (phép $\circ$ có thể là $max, min, ...$).

$$f_0 = c \ (c \text{ là hằng số})$$
$$f_i = f_{i-1} \circ a_i (i > 0)$$

Ví dụ 3.2

Cho mảng a gồm n số nguyên ($n \leq 10^6$) được đánh số từ $0, 1, ...n-1$. Cho q truy vấn, mỗi truy vấn là 1 số nguyên không âm id. Yêu cầu với mỗi truy vấn, xuất ra phần tử lớn nhất trong đoạn $(0, id)$ của mảng a.

– Ta có thể áp dụng định nghĩa hàm f ở trên để tìm kết quả của bài toán. Lúc này f_i là phần tử lớn nhất từ 0 tới i.

$$f_0 = a_0$$
$$f_i = max(f_{i-1}, a_i)(i > 0)$$

```
// Hàm tạo mảng f
void build() {
    int f[n] = {0}; // Mảng f gồm n phần tử mang giá trị 0
    f[0] = a[0];
    for (int i = 1; i < n; ++i) {
        f[i] = max (f[i - 1], a[i]);
    }
}
void read() {
    // Gọi hàm tạo mảng f
    build();
    cin >> n >> q;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    for (int j = 1; j <= q; ++j) {
        cin >> id;
        cout << f[id] << '\n';
    }
}
```

– Ta cũng có thể áp dụng công thức trên cả với các bài tìm $min, gcd, lcm, ...$

Mảng hậu tố

Mảng hậu tố là dạng cấu trúc mảng có ý tưởng tương tự mảng tiền tố. Ví dụ, ta đang tính hàm f là phép biến đổi từ mảng a thông qua phép $\circ$ (phép $\circ$ có thể là $max, min, ...$).

$$f_{n-1} = c \ (c \text{ là hằng số})$$
$$f_i = f_{i+1} \circ a_i (i < n-1)$$

Ví dụ 3.3

Cho mảng a gồm n số nguyên ($n \leq 10^6$) được đánh số $0, 1, ..., n-1$. Cho q truy vấn, mỗi truy vấn là 1 số nguyên không âm id. Yêu cầu với mỗi truy vấn, xuất ra phần tử bé nhất trong đoạn $(id, n-1)$ của mảng a.

– Ta có thể áp dụng định nghĩa hàm f ở trên để tìm kết quả của bài toán. Lúc này f_i là phần tử bé nhất từ i tới $n-1$.

$$f_{n-1} = a_{n-1}$$
$$f_i = min(f_{i+1}, a_i) \quad (i > 0)$$

```
// Hàm tạo mảng f
void build() {
    int f[n] = {0}; // Mảng f gồm n phần tử mang giá trị 0
    f[n - 1] = a[n - 1];
    for (int i = n - 2; i >= 0; --i) {
        f[i] = min (f[i + 1], a[i]);
    }
}
void read() {
    // Gọi hàm tạo mảng f
    build();
    cin >> n >> q;
    for (int i = 0; i < n; ++i) {
        cin >> a[i];
    }
    for (int j = 1; j <= q; ++j) {
        cin >> id;
        cout << f[id] << '\n';
    }
}
```

– Ta cũng có thể áp dụng công thức trên cả với các bài tìm $max, gcd, lcm, ...$

3.3 Mảng cộng dồn 1 chiều

Mảng cộng dồn (hay còn gọi là Prefix sum) là một dạng cấu trúc mảng được ứng dụng rất nhiều trong các bài tập lập trình. Mảng cộng dồn cũng là một dạng mảng tiền tố.

Ý tưởng

– Giả sử, ta có bài toán:

Ví dụ 3.4

Cho mảng a gồm n số nguyên dương ($n \leq 10^6$). Cho q truy vấn ($q \leq 10^6$), mỗi truy vấn gồm 2 số nguyên dương ($l \leq r \leq n$). Với mỗi truy vấn, trả về kết quả của $a_l + a_{l+1}, \cdots, a_r$.

Cài đặt ngây thơ:

+ Với mỗi truy vấn, ta duyệt qua tất cả các phần tử trong $a_l, \cdots, a_r$ và cộng vào biến ans là kết quả của truy vấn.
+ Code:

```cpp
long long getSum(int l, int r) {
    long long ans = 0;
    // Duyệt qua các phần tử từ a[l] -> a[r]
    for (int i = l; i <= r; ++i) {
        ans += a[i];
    }
    return ans;
}

void solve() {
    cin >> n >> q;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    for (int j = 1; j <= q; ++j) {
        cin >> l >> r;
        cout << getSum(l, r) << '\n';
    }
}
```

+ Độ phức tạp: $O(q \times n)$.
+ Cách cài đặt này chỉ được dùng để giải bài toán với n và q nhỏ.

Sử dụng mảng cộng dồn:

+ Ta gọi mảng s là mảng cộng dồn của mảng a. Gọi s_i là kết quả của $a_1 + a_2 + \cdots + a_i$.
+ Dễ dàng thấy kết quả của 1 truy vấn (l, r) là $s_r - s_{l-1}$.

  Chứng minh:

$$s_r - s_{l-1} = (a_1 + \cdots + a_r) - (a_1 + \cdots + a_{l-1}) \tag{1}$$

$$s_r - s_{l-1} = (a_1 - a_1) + \cdots + (a_{l-1} - a_{l-1}) + a_l + \cdots + a_r \tag{2}$$

  Suy ra:

$$s_r - s_{l-1} = a_l + \cdots + a_r \tag{3}$$

+ Code:

```cpp
// Hàm cài đặt mảng cộng dồn
void buildPrefixSum() {
    for (int i = 1; i <= n; ++i) {
        s[i] = s[i - 1] + a[i];
    }
}

long long getSum(int l, int r) {
    long long ans = s[r] - s[l - 1];
    return ans;
}

void solve() {
```

```
    cin >> n >> q;
    for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }

    // Gọi trước hàm cài đặt mảng cộng dồn
    buildPrefixSum();

    for (int j = 1; j <= q; ++j) {
        int l, r;
        cin >> l >> r;
        cout << getSum(l, r) << '\n';
    }
}
```

+ Độ phức tạp: $O(n+q)$.

3.4 Mảng cộng dồn 2 chiều

Ý tưởng

– Ta có bài toán sau:

Ví dụ 3.5

Cho 1 bảng gồm n dòng và m cột $(n, m \leq 1000)$. Cho q truy vấn $(q \leq 10^5)$, mỗi truy vấn gồm 4 số nguyên dương (x_1, y_1, x_2, y_2) $(x_1, x_2 \leq n;\ y_1, y_2 \leq m)$. Với mỗi truy vấn, trả về kết quả của tổng của tất cả các phần tử trong hình chữ nhật có góc trái trên là ô (x_1, y_1) và góc phải dưới là ô (x_2, y_2).

Cài đặt ngây thơ:

+ Với mỗi truy vấn, ta duyệt qua tất cả phần tử thuộc hình chữ nhật có góc trái trên là (x_1, y_1) và góc phải dưới (x_2, y_2).
+ Code:

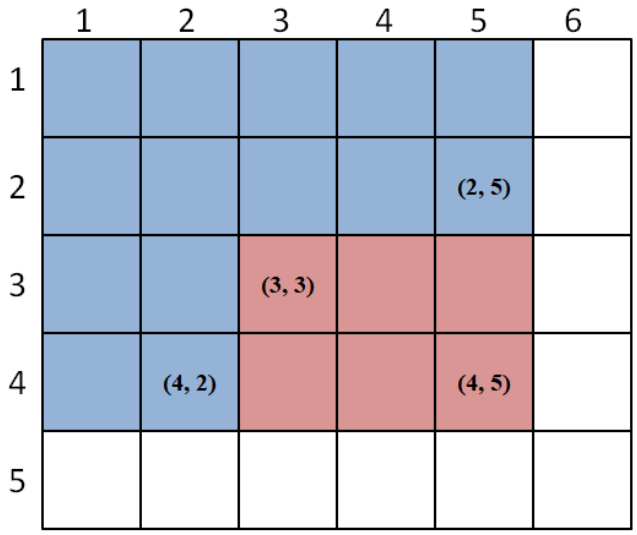
```
long long getSum(int x1, int y1, int x2, int y2){
    long long ans = 0;
    for (int i = x1; i <= x2; ++i){
        for (int j = y1; j <= y2; ++j){
            ans += a[i][j];
        }
    }
    return ans;
}
```

+ Độ phức tạp: $O(q \times n \times m)$.

+ Dễ dàng nhận thấy, cách giải trên sẽ rất tốn thời gian và không hiệu quả. Từ đây, ta có thể áp dụng **Mảng cộng dồn 2 chiều** để giải bài toán.

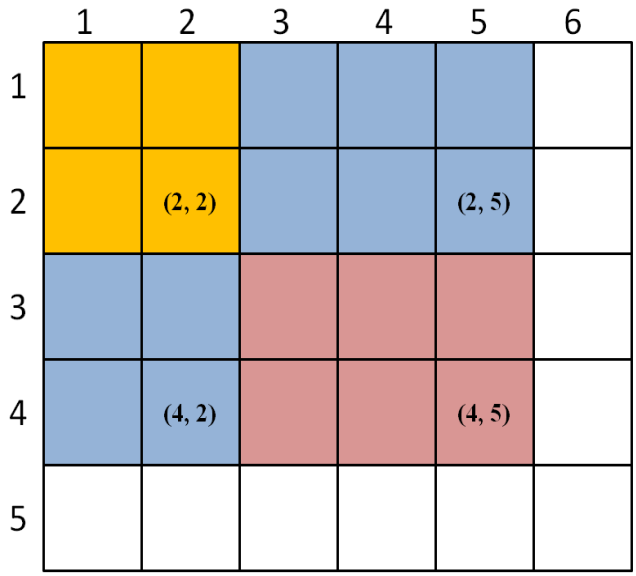
Mảng cộng dồn 2 chiều

+ Ta gọi mảng 2 chiều s có giá trị ô $s(i, j)$ là tổng của các giá trị của hình chữ nhật có góc trái trên là $(1, 1)$ và góc phải dưới là (i, j).
+ Từ đây, ta có thể dễ dàng nhận ra để tính được hình chữ nhật có góc trái trên là (x_1, y_1) và góc phải dưới (x_2, y_2). Khi có đáp án của hình chữ nhật góc trái trên là $(1, 1)$ và góc phải dưới là (x_2, y_2), ta cần giảm đi phần bị dư là $s(x_1 - 1, y_2)$ và $s(x_2, y_1 - 1)$.
+ Dưới đây là hình vẽ minh họa cho ví dụ có $(x_1, y_1, x_2, y_2) = (3, 3, 4, 5)$:



Hình 1. Loại bỏ 2 hình chữ nhật có góc phải trên là (1, 1) và góc phải dưới lần lượt là (2, 5) và (4, 2)

+ Tuy nhiên, ta nhận thấy, hình chữ nhật có góc phải dưới $(2, 2)$ bị trừ 2 lần. Vì vậy ta cần thêm tiếp giá trị của $s(2, 2)$ vào đáp án.



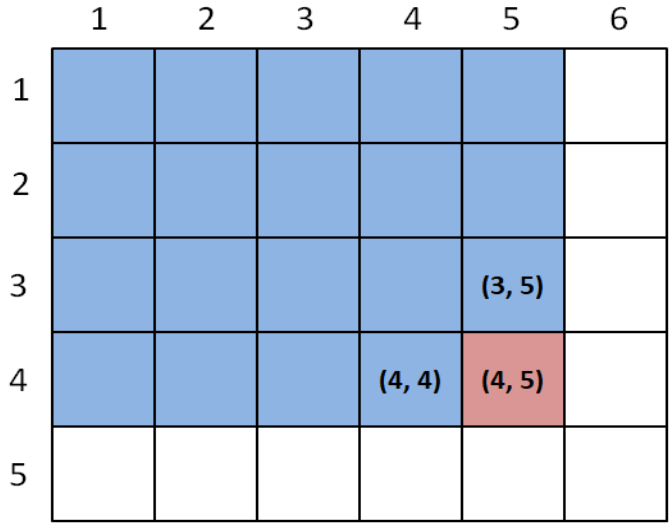
Hình 2. Thêm hình chữ nhật có góc phải trên là (1, 1) và góc phải dưới (2, 2)

+ Như vậy công thức tổng quát để tính giá trị hình chữ nhật có góc trái trên là (x_1, y_1) và (x_2, y_2):

$$ans(x_1, y_1, x_2, y_2) = s(x_2, y_2) - s(x_1 - 1, y_2) - s(x_2, y_1 - 1) + s(x_1 - 1, y_1 - 1)$$

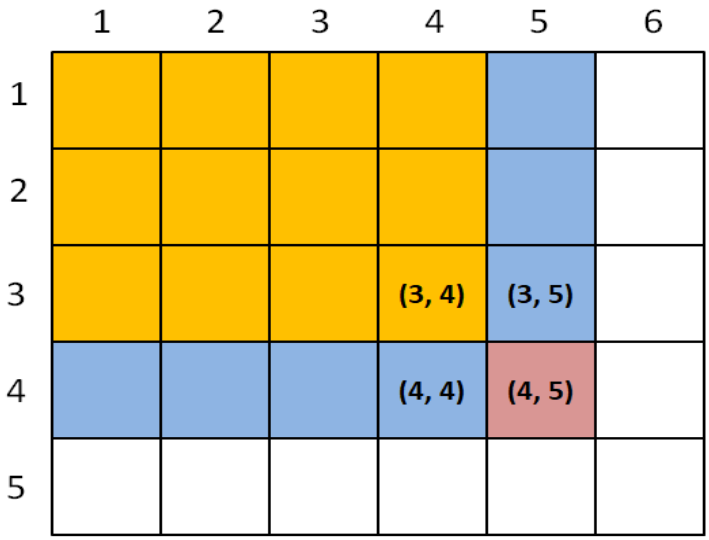
Cách cài đặt

+ Ý tưởng để xây dựng mảng cộng dồn 2 chiều cũng tương tự với cách tư duy của công thức trên.
+ Ta có thể dễ dàng nhận thấy được để tính được hình chữ nhật có góc trái trên là $(1, 1)$ và góc trái dưới là (i, j). Ta cần lấy giá trị $a(i, j)$ cộng với các giá trị ở phần hình chữ nhật có góc trái dưới $(i - 1, j)$ và hình có góc trái dưới là $(i, j - 1)$.
+ Dưới đây là minh họa cho ví dụ cần tính giá trị ô $s(4, 5)$.



Hình 3. Thêm giá trị s(4, 4) và s(3, 5)

+ Ta lại nhận thấy hình chữ nhật có góc trái dưới $(i - 1, j - 1)$ được cộng vào $s(i, j)$ 2 lần. Vì vậy, ta cần trừ bớt giá trị của $s(i - 1, j - 1)$ để có được đáp án của $s(i, j)$.



Hình 4. Trừ giá trị s(3, 4)

+ Như vậy để giải được bài toán trên hiệu quả, ta có thể giải như sau:

```
// Hàm cài đặt mảng cộng dồn 2 chiều
void buildPrefixSum(){
    for (int i = 1; i <= n; ++i){
        for (int j = 1; j <= m; ++j){
            s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];
        }
    }
}

long long getSum(int x1, int y1, int x2, int y2){
    long long ans = s[x2][y2] - s[x1 - 1][y2] - s[x2][y1 - 1] + s[x1 - 1][y1 - 1];
    return ans;
}

void solve(){
    // Gọi trước hàm cài đặt mảng cộng dồn 2 chiều
    buildPrefixSum();
    cin >> n >> m >> q;
    for (int i = 1; i <= n; ++i){
        for (int j = 1; j <= m; ++j){
            cin >> a[i][j];
        }
    }
    for (int j = 1; j <= q; ++j){
        int x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        cout << getSum(x1, y1, x2, y2a) << '\n';
    }
}
```

+ Độ phức tạp: $O(n \times m + q)$.

3.5 Mảng hiệu

Mảng hiệu là một trong những kỹ thuật cơ bản để giải các bài tập về mảng. Mảng hiệu thú vị do có liên hệ hữu dụng với mảng cộng dồn cũng như khả năng xử lý truy vấn thay đổi giá trị trên những đoạn con mà mảng cộng dồn không làm được.

Định nghĩa

Mảng hiệu của mảng A trong tiếng anh là Difference Array - kí hiệu là $D(A)$, là mảng được khởi tạo theo quy tắc: $D_i = A_{i+1} - A_i$.

```
// Tạo mảng D từ mảng A cho trước
// Mảng A đánh số từ 1 để có D[0] = A[1] - 0;
for (int i = 0; i <= n - 1; i++)
    D[i] = A[i + 1] - A[i];
```

Tính chất

Từ $D(A)$ có thể biến đổi trở lại thành mảng A bằng cách lấy *mảng cộng dồn* của $D(A)$. Dễ thấy:

$$D_0 + D_1 + ... + D_i = A_1 + (A_2 - A_1) + ... + (A_{i+1} - A_i) = A_i$$

Điều ngược lại cũng đúng: Để nhận lại A ta lấy *mảng hiệu* của *mảng cộng dồn* của A.

Ứng dụng

– Ta có bài toán như sau:

Ví dụ 3.6

Cho mảng a gồm n số nguyên dương ($n \leq 10^6$). Cho q truy vấn ($q \leq 10^6$), mỗi truy vấn gồm 3 số nguyên dương l, r, k ($l \leq r \leq n$; $k \leq 10^e$). Với mỗi truy vấn, thêm k vào các phần tử $a_l,\ a_{l+1}, ...,\ a_r$. Hãy xuất ra mảng a sau q truy vấn.

Ý tưởng ngây thơ

– Mỗi truy vấn ta đi qua $r - l + 1$ phần tử của mảng và lần lượt cộng k. Khi xử lí hết q truy vấn chỉ cần in ra mảng a.

– Code:

```
for(int i = 1; i <= q; i++){
    cin >> l >> r >> k;
for(j = l ; j <= r; j++)
    a[j] += k;
}

// in mảng sau xử lí
for(int i = 1; i <= n; i++){
    cout << a[i] << ' ';
}
```

– Độ phức tạp: $O(q \times n)$.

Cài đặt mảng hiệu

– Xét truy vấn q_i với ($l_i,\ r_i,\ k_i$). Nhận thấy với 2 phần tử liền kề nhau mà $l_i \leq a_i,\ a_{i+1} \leq r_i$ thì truy vấn sẽ không ảnh hưởng đến $D_i = a_{i+1} - a_i$.

– Vì thế ta chỉ cần thay đổi giá trị D_{l_i-1} và D_{r_i} với mỗi truy vấn q_i rồi theo tính chất đầu tiên của *mảng hiệu* mà nhận lại mảng A đã thay đổi.

– Code:

```
for(int i = 1; i <= q; i++){
    cin >> l >> r >> k;
    D[l-1] += k;
```

```
    D[r] += k;
}

for(int i = 1; i <= n; i++){
    A[i] = A[i-1] + D[i-1];
}
```

– Độ phức tạp: $O(n+q)$.

4 Bài tập

1. **Mảng thống kê**
   a. CSES - Missing Number
   b. SUMK - Bài 3 TS10 PTNK 2023
2. **Mảng cộng dồn 1 chiều**
   a. CSES - Subarray Sum II
   b. Codeforces - Good Subarrays
3. **Mảng cộng dồn 2 chiều**
   a. CSES - Forest Queries
   b. VNOJ - Maxcub
4. **Mảng hiệu**
   a. Codeforces - Karen And Coffee (Kết hợp mảng cộng dồn)
   b. Codeforces - Encrypting Messages

5 Tham khảo

1. Mảng cộng dồn và mảng hiệu - VNOI