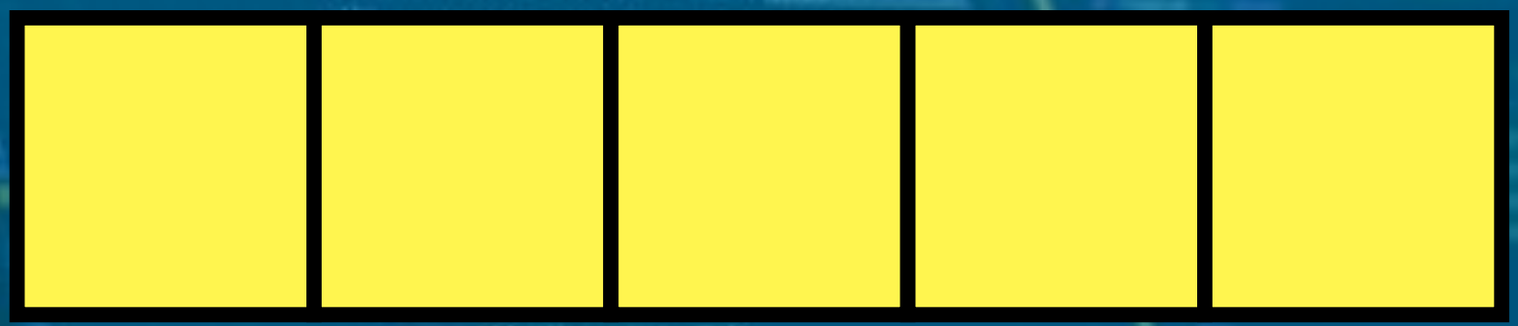
MẢNG MỘT CHIỀU (1D ARRAY)



NỘI DUNG

Bài toán cần giải quyết

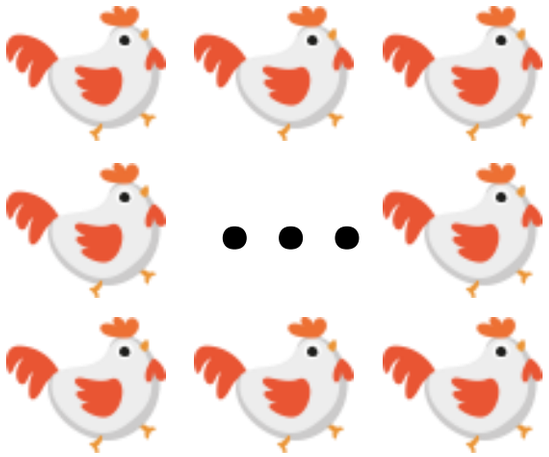
Định nghĩa và tính chất

Các thao tác trên mảng

Một số bài toán cơ bản

1. BÀI TOÁN CẦN GIẢI QUYẾT

Bài toán: Giả sử bạn cần tính trọng lượng của 1 đàn gà lên đến hàng nghìn con. Vậy việc lưu trữ trọng lượng của hàng nghìn con gà này sẽ phải xử lý ra sao?



Giải pháp:

Khai báo 1000 biến cho 1000 con gà

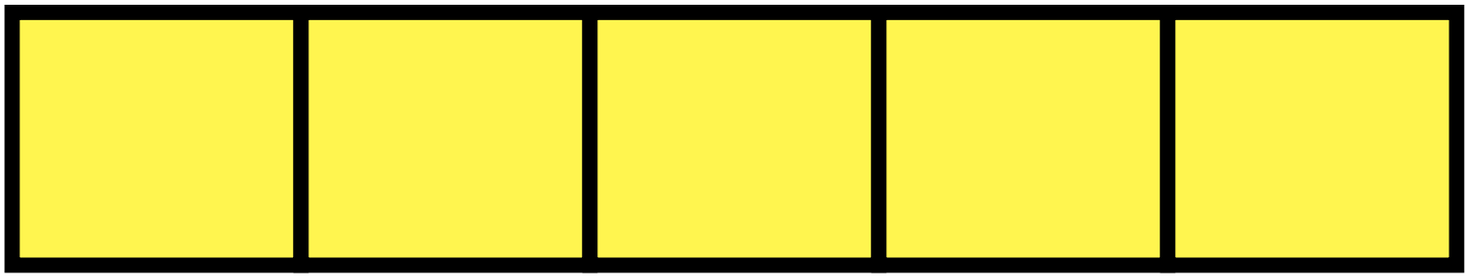
Nhiều hạn chế, không phù hợp với số lượng lớn

Sử dụng **mảng một chiều**

Hiệu quả, dễ quản lí

2. ĐỊNH NGHĨA VÀ TÍNH CHẤT

Mảng 1 chiều là một cấu trúc dữ liệu gồm nhiều phần tử có cùng kiểu dữ liệu, được lưu trữ ở các ô nhớ liên tiếp nhau trong bộ nhớ.



Mảng 1 chiều được sử dụng khi bạn cần lưu trữ một số lượng lớn các phần tử có cùng kiểu dữ liệu. Ví dụ: 100 số nguyên, 1000 số thực, 1000 học sinh,...

Mảng 1 chiều đơn giản, dễ hiểu và được sử dụng rất nhiều trong mọi ngôn ngữ lập trình.

3. KHAI BÁO MẢNG

SYNTAX

```
Data_type array_name [Number_of_element]
```

Ví dụ:

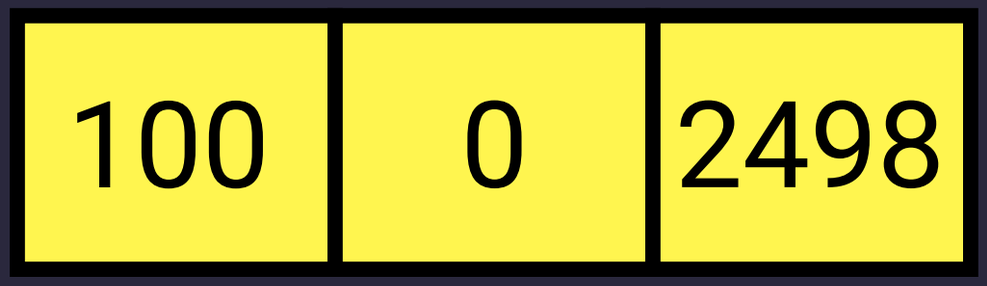
Khai báo	Ý nghĩa
int a[100];	Mảng số nguyên int a có 100 phần tử
float b[1000];	Mảng số thực float b có 1000 phần tử
double diem[10];	Mảng số thực double diem có 10 phần tử
char ten[50];	Mảng kí tự char ten có 50 phần tử

3. KHAI BÁO MẢNG

Ví dụ 1:

Khai báo mảng a có 3 phần tử là số nguyên, các giá trị của a là giá trị rác.

```
int a[3];
```

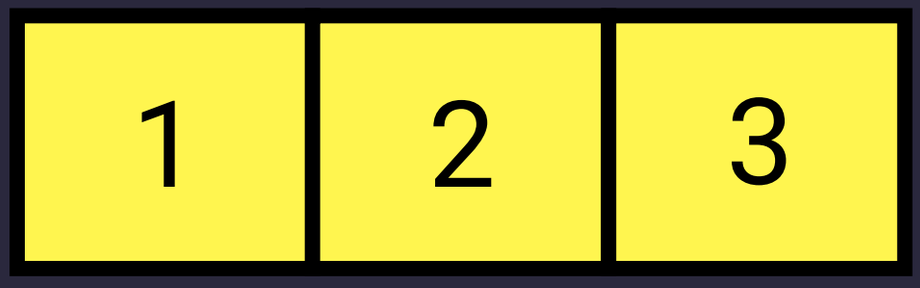


3. KHAI BÁO MẢNG

Ví dụ 2:

Khai báo mảng a có 3 phần tử là số nguyên, gán lần lượt các phần tử trong mảng là 1,2,3

```
int a[3] = {1, 2, 3};
```

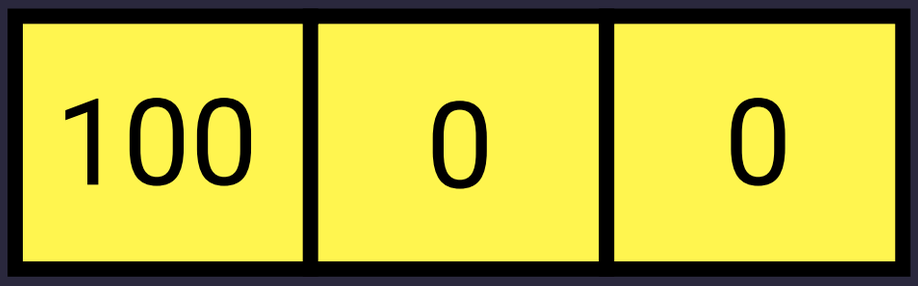


3. KHAI BÁO MẢNG

Ví dụ 3:

Khai báo mảng a có 3 phần tử là số nguyên, chỉ khởi tạo phần tử đầu tiên, khi đó mọi phần tử không được khởi tạo sẽ có giá trị là 0

```
int a[3] = {100};
```



3. KHAI BÁO MẢNG

Ví dụ 3:

Vậy nếu bạn muốn khởi tạo một mảng toàn số 0. Cú pháp:

```
int a[1000] = {0};
```

4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

Các phần tử trong mảng được truy cập thông qua chỉ số.

Chỉ số của mảng được đánh từ 0 và kết thúc bởi n - 1 (với n là số lượng phần tử của mảng)

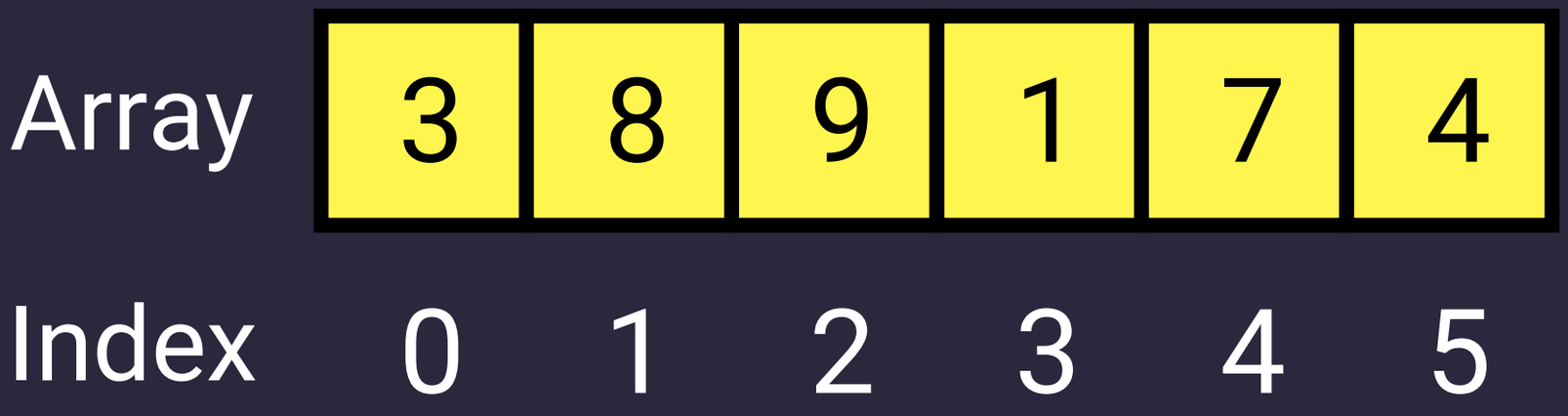
Cú pháp truy cập

```
array_name [index]
```

4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

Ví dụ:

```
int a [6] = {3, 8, 9, 1, 7, 4};
```



4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

Ví dụ:

a[0] =	3
a[1] =	8
a[2] =	9
a[3] =	1
a[4] =	7
a[5] =	4

4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

Ví dụ về nhập và duyệt mảng:

Duyệt thông qua chỉ số:

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

int main(){
    int n; // số lượng phần tử trong
mảng
    cin >> n;
    int a[1000]; // Chú ý nếu n lớn phải
// khai báo mảng a ít nhất n phần tử
     for(int i = 0; i < n; i++){
          cin >> a[i];
     }
     for(int i = 0; i < n; i++){
          cout << a[i] << ' ';
     }
}
```

4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

Ví dụ về nhập và duyệt mảng:

Duyệt bằng for each

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

int main(){
    int a[5] = {1, 2, 3, 4, 5};
    for(int ele : a){
        cout << ele << ' ';
    }
}
Output : 1 2 3 4 5
```

4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

Ví dụ về nhập và duyệt mảng:

Duyệt bằng for each

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

int main(){
    int a[10] = {1, 2, 3, 4, 5};
    for(int ele : a){
        cout << ele << ' ';
    }
}
```

Output : 

4. TRUY CẬP PHẦN TỬ VÀ DUYỆT MẢNG

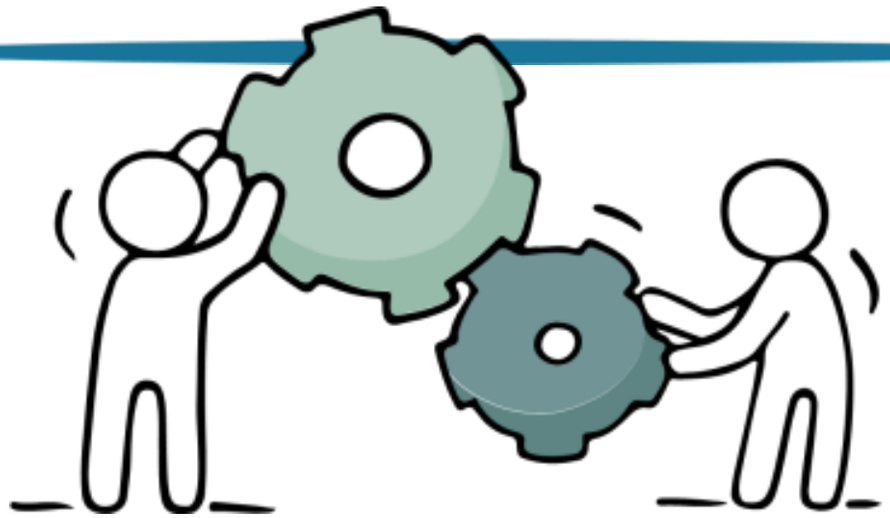
Chú ý : Khi khai báo kích thước của mảng hãy chú ý tới giới hạn số lượng phần tử tối đa của đầu bài.

Ví dụ mảng của đầu bài có thể lên đến 1000 thì mảng của bạn phải khai báo tối thiểu là 1000 phần tử.

5. MẢNG LÀM THAM SỐ CỦA HÀM

Khi mảng làm tham số của hàm, những thay đổi trong hàm sẽ làm thay đổi tới mảng mảng được truyền vào.

Chú ý : Khi xây dựng hàm có tham số là mảng, cần phải kèm theo 1 tham số nữa là số lượng phần tử trong mảng



| VD1: Gấp đôi mọi phần tử trong mảng

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

void nhanDoi(int a[], int n){
        for(int i = 0; i < n; i++){
                a[i] *= 2;
        }
}

int main(){
      int a[5] = {1, 2, 3, 4, 5};
      nhanDoi(a, 5);
      for(int i = 0; i < 5; i++){
              cout << a[i] << ' ';
      }
}
Output : 2 4 6 8 10
```

| VD2: Tính tổng các phần tử trong mảng

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

int tong(int a[], int n){
        int sum = 0;
        for(int i = 0; i < n;
i++){
                sum += a[i];
        }
        return sum;
}

int main(){
      int a[5] = {1, 2, 3, 4,
5};
      cout << tong(a, 5) <<
endl;
}
Output : 15
```

6. ƯU VÀ NHƯỢC ĐIỂM CỦA MẢNG

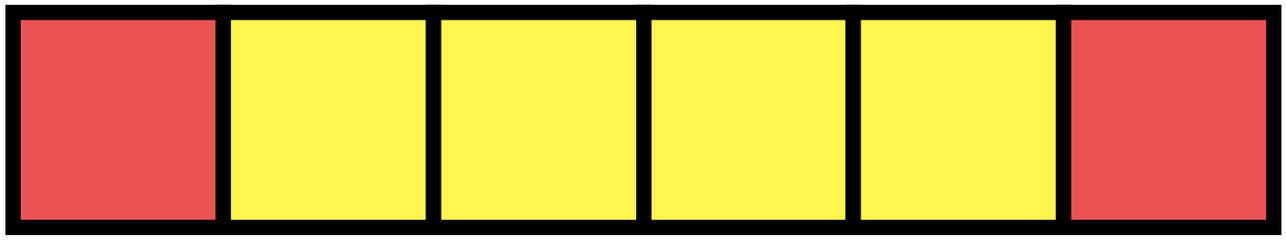
Ưu điểm

- Đơn giản, dễ hiểu, dễ sử dụng
- Thuận tiện truy cập phần tử qua chỉ số,
- Khai báo một loạt các phần tử có cùng kiểu dữ liệu
- Duyệt mọi phần tử trong mảng bằng 1 vòng for

Nhược điểm

- Kích thước của mảng là cố định, bạn không thể mở rộng, thu hẹp mảng một khi nó đã được khai báo
- Việc chèn và xóa phần tử trong mảng là khó khăn

7. NO INDEX OUT OF BOUND CHECKING



Mảng trong ngôn ngữ lập trình C++ không kiểm tra việc bạn có truy cập vào một chỉ số hợp lệ hay không.

Khi mảng của bạn có N phần tử thì chỉ số hợp lệ sẽ là từ 0 tới N - 1. Tuy nhiên bạn hoàn toàn có thể truy cập vào các chỉ số không hợp lệ như -1, N, N+1, ...

Các giá trị này có thể là giá trị rác, các ô nhớ tương ứng với các phần tử này có thể đang thuộc quản lý của một tiến trình khác.

7. NO INDEX OUT OF BOUND CHECKING

Các ngôn ngữ bậc cao sẽ hạn chế điều này còn trong C++ các bạn lập trình viên phải tự đảm bảo code của mình không truy cập vào chỉ số không hợp lệ.

Việc truy cập vào các chỉ số không hợp lệ sẽ gây lỗi Segmentation Fault trên Hackerrank hoặc các lỗi Runtime Error trên các Online judge khác.

8. MỘT SỐ BÀI TOÁN CƠ BẢN CẦN LƯU Ý KHI LÀM VIỆC TRÊN MẢNG

1. Tìm phần tử lớn nhất và nhỏ nhất trong mảng.
2. Liệt kê hoặc đếm các phần tử trong mảng thỏa mãn yêu cầu (số nguyên tố, số chính phương, số fibonacci,...
3. Các bài toán liên quan tới cặp số trong mảng (cặp số nguyên tố cùng nhau, có tổng bằng K,...
4. Sắp xếp các phần tử trong mảng.
5. Tìm kiếm hoặc đếm số lần xuất hiện của một phần tử nào đó trong mảng.

Tìm phần tử lớn nhất, nhỏ nhất trong mảng

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

int main(){
      int n; cin >> n;
      int a[n];
      for(int i = 0; i < n; i++) cin >> a[i];
      int max_ele = a[0], min_ele = a[0];
      for(int i = 1; i < n; i++){
          if (a[i] < min_ele){
               min_ele = a[i];
          }
           if (a[i] > max_ele){
               max_ele = a[i];
           }
      }
      cout << min_ele << ' ' << max_ele << endl;
}
```

Tìm phần tử lớn nhất, nhỏ nhất trong mảng

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

int main(){
      int n; cin >> n;
      int a[n];
      for(int i = 0; i < n; i++) cin >> a[i];
      int max_ele = -1e9, min_ele = 1e9;
      for(int i = 1; i < n; i++){
              max_ele = max (max_ele, a[i]);
              min_ele = min (min_ele, a[i])
     }
     cout << min_ele << ' ' << max_ele << endl;
}
```

Liệt kê hoặc đếm các phần tử trong mảng

Chương trình liệt kê số nguyên tố trong mảng

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

bool nt(int n){
      for(int i = 2; i <= sqrt(n); i++){
            if (n % i == 0) return false;
       }
      return n > 1;
}

int main(){
      int n; cin >> n;
      int a[n];
      for(int i = 0; i < n; i++) cin >> a[i];
      for(int i = 0; i < n; i++){
           if (nt(a[i]))
                cout << a[i] << ' ';
      }
}
```

Các bài toán liên quan tới cặp số trong mảng

Chương trình đếm số cặp trong mảng có tổng bằng K

```
#include <bits/stdc++.h>
using namespace std;
int main(){
        int n, k;
        cin >> n >> k;
        int a[n];
        for(int i = 0; i < n; i++) cin >> a[i];
        int ans = 0;
        for(int i = 0; i < n; i++){
             for(int j = i + 1; j < n; j++){
                  if (a[i] + a[j] == k) {
                       ++ans;
                  }
             }
         }
         cout << ans << endl;
}
```

Các bài toán liên quan tới cặp số trong mảng

Chương trình đếm số cặp trong mảng có tổng bằng K

Khi bạn cần xét mọi cặp 2 phần tử trong mảng thì bạn cần 2 vòng for lồng nhau.

Tổng quát nếu bạn cần xét mọi cặp k phần tử thì bạn cần k vòng for lồng nhau

Sắp xếp các phần tử trong mảng

Chương trình sắp xếp mảng tăng dần

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

int main(){
      int n; cin >> n;
      int a[n];
      for(int i = 0; i < n; i++) cin >> a[i];
      sort(a, a + n); // sắp xếp theo thứ tự tăng dần
      for(int i = 0; i < n; i++) cout << a[i] << ' ';
}
```

```
Input
5
1 3 2 5 4
Output
1 2 3 4 5
```

Sắp xếp các phần tử trong mảng

Chương trình sắp xếp mảng giảm dần

```
#include <bits/stdc++.h>
#include <algorithm> // For sort
using namespace std;
int main(){
      int n; cin >> n;
      int a[n];
      for(int i = 0; i < n; i++) cin >> a[i];
      sort(a, a + n, greater<int>()); // sắp xếp theo
      // thứ tự giảm dần
      for(int i = 0; i < n; i++) cout << a[i] << ' ';
}
```

```
Input
5
1 3 2 5 4
Output
5 4 3 2 1
```

Sắp xếp các phần tử trong mảng

Chương trình sắp xếp mảng giảm dần

Khi sắp xếp giảm dần các bạn thêm tham số thứ 3 là greater<>, và chú ý lựa chọn kiểu dữ liệu phù hợp với kiểu dữ liệu trong mảng.

| Ví dụ:

Mảng lưu số float thì dụng greater <float>. Tương tự với các kiểu khác như long long, char, ...

Tìm kiếm hoặc đếm số lần xuất hiện của một phần tử nào đó trong mảng

Tìm kiếm sự xuất hiện của phần tử x trong mảng

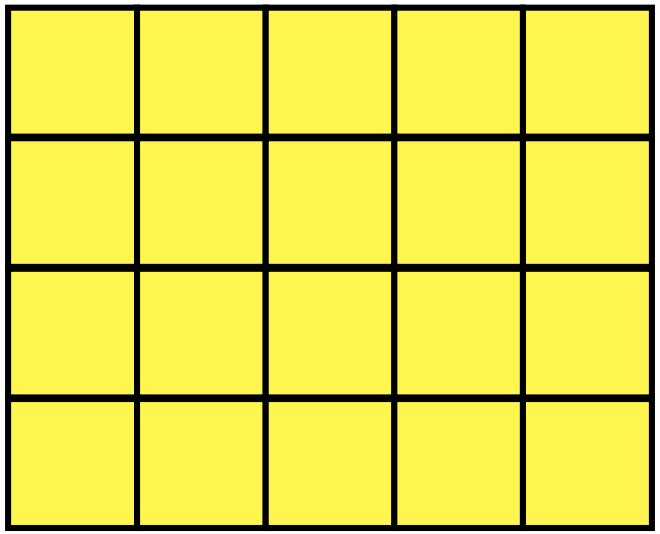
```
#include <bits/stdc++.h>
#include <algorithm> // For sort
using namespace std;

//ham kiem tra xem x co nam trong mang a hay khong?
bool check(int a[], int n, int x){
     for(int i = 0; i < n; i++){
          if(a[i] == x) return true; // found
     }
     return false; // not found
}

int main(){
     int n, x; cin >> n >> x;
     int a[n];
     for(int i = 0; i < n; i++) cin >> a[i];
          if(check(a, n, x)){
               cout << "FOUND\n";
          }
     else cout << "NOT FOUND\n";
}
```

Tìm kiếm hoặc đếm số lần xuất hiện của một phần tử nào đó trong mảng

Đếm số lần xuất hiện của x trong mảng

```
#include <bits/stdc++.h>
#include <algorithm> // For sort
using namespace std;
int count(int a[], int n, int x){
      int res = 0;
      for(int i = 0; i < n; i++){
            if(a[i] == x) ++res;
      }
      return res;
}
int main(){
     int n, x; cin >> n >> x;
     int a[n];
     for(int i = 0; i < n; i++) cin >> a[i];
     cout << count(a, n, x) << endl;
}
```

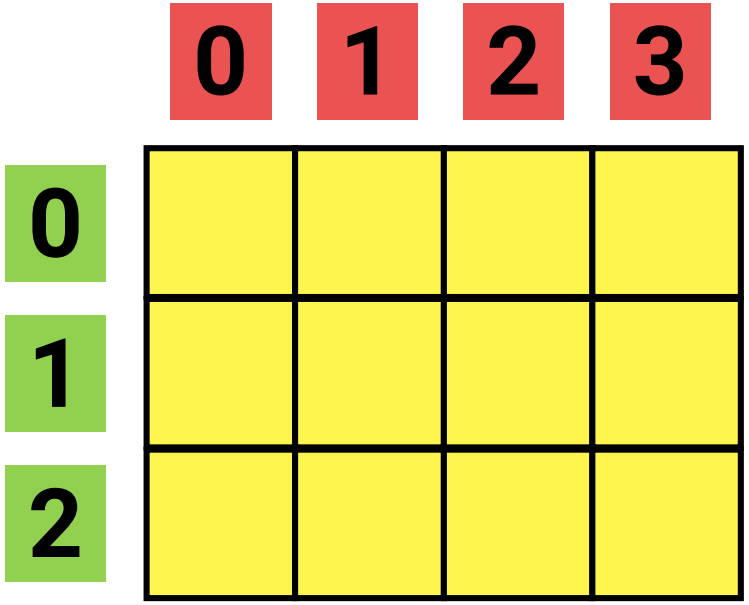
MẢNG HAI CHIỀU (2D ARRAY)



NỘI DUNG

Khái quát về mảng hai chiều:

Mảng 2 chiều được sử dụng trong các bài toán liên quan tới ma trận, bảng số. Bạn có thể coi mảng 2 chiều chính là các mảng một chiều được xếp chồng lên nhau



- Chỉ số hàng
- Chỉ số cột

1. Khai báo mảng hai chiều

Khi khai báo mảng hai chiều, các bạn cần chỉ ra số hàng, số cột của ma trận.

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    //Mảng a gồm 3 hàng, mỗi hàng 3 cột
    int a[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9};
    }
    //Mảng b có 10 hàng, mỗi hàng 10 cột
    int b[10][10];
}
```

a[3][3]

1	2	3
4	5	6
7	8	9

2. Truy cập phần tử trong mảng:

Để truy cập vào phần tử trong mảng, các bạn dùng chỉ số hàng và chỉ số cột.

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

```
a[0][2] = 3
a[2][1] = 8
```

Chỉ số hàng và cột của mảng 2 chiều được đánh số từ 0 như mảng 1 chiều.

3. Nhập và duyệt mảng hai chiều:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n, m; cin >> n >> m;
    int a[n][m];
    for(int i = 0; i < n; i++){
       for(int j = 0; j < m; j++){
           cin >> a[i][j];
       }
    }
    for(int i = 0; i < n; i++){
       for(int j = 0; j < m; j++){
           cout << a[i][j] << " ";
       }
       cout << endl;
    }
}
```

4. Các bài toán cơ bản trên mảng hai chiều:

Tìm phần tử lớn nhất, nhỏ nhất

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n, m; cin >> n >> m;
    int a[n][m];
    int max_val = INT_MIN, min_val = INT_MAX;
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> a[i][j];
            max_val = max(max_val, a[i]);
            min_val = min(min_val, a[i]);
        }
     }
     cout << max_val << " " << min_val;
}
```

4. Các bài toán cơ bản trên mảng hai chiều:

Tính tổng từng hàng của mảng 2 chiều

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n, m; cin >> n >> m;
    int a[n][m];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> a[i][j];
        }
    }
    for(int i = 0; i < n; i++){
        int sum = 0;
        for(int j = 0; j < m; j++){
            sum += a[i][j];
        }
        cout << sum << endl;
    }
}
```

4. Các bài toán cơ bản trên mảng hai chiều:

Cộng trừ hai ma trận:

Trong đại số tuyến tính, ma trận tương tự như một mảng 2 chiều gồm n hàng và m cột. Để 2 ma trận có thể cộng hoặc trừ cho nhau thì chúng phải có cùng số hàng và số cột.

1	2	0
0	4	1

+

1	4	8
9	2	3

=

2	6	8
9	6	4

1	2	0
0	4	1

−

1	4	8
9	2	3

=

0	-2	-8
-9	2	-2

4. Các bài toán cơ bản trên mảng hai chiều:

Cộng 2 ma trận cỡ n hàng m cột

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n, m; cin >> n >> m;
    int a[n][m], b[n][m];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> a[i][j];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> b[i][j];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << a[i][j] + b[i][j] << " ";
        }
        cout << endl;
    }
}
```

Trừ 2 ma trận cỡ n hàng m cột

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n, m; cin >> n >> m;
    int a[n][m], b[n][m];
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> a[i][j];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cin >> b[i][j];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << a[i][j] - b[i][j] << " ";
        }
        cout << endl;
    }
}
```

4. Các bài toán cơ bản trên mảng hai chiều:

Nhân hai ma trận:

Giả sử có 2 ma trận a cỡ nxm, ma trận b cỡ pxq, để ma trận a có thể nhân với ma trận b thì số cột của ma trận a, tức là m phải bằng số hàng của ma trận b, tức là p.

```
a[n][m] x b[p][q] = c[n][q]
```

Khi đó m = p thì ma trận tích của a với b sẽ là ma trận c có cỡ nxq. Phần tử ở chỉ số (i, j) của ma trận tích c được tính bằng cách nhân từng cặp phần tử ở hàng i của ma trận a với các phần tử ở cột j của ma trận b.

4. Các bài toán cơ bản trên mảng hai chiều:

Nhân hai ma trận:

Nhập 2 ma trận:

```
int n, m, p;
cin >> n >> m >> p;
int a[n][m], b[m][p], c[n][p];
for(int i = 0; i < n; i++){
    for(int j = 0; j < m; j++){
        cin >> a[i][j];
    }
}
for(int i = 0; i < m; i++){
    for(int j = 0; j < p; j++){
        cin >> b[i][j];
    }
}
```

Tính ma trận tích và in kết quả

```
for(int i = 0; i < n; i++){
   for(int j = 0; j < p; j++){
       c[i][j] = 0;
       for(int k = 0; k < m; k++){
          c[i][j] += a[i][k] * b[k][j];
       }
   }
}
for(int i = 0; i < n; i++){
   for(int j = 0; j < p; j++){
       cout << c[i][j] << " ";
   }
   cout << endl;
}
```

5. Kĩ thuật duyệt các ô liền kề:

i-1, j-1	i-1, j	i-1, j+1
i, j-1	i, j	i, j+1
i+1, j-1	i+1, j	i+1, j+1

5. Kĩ thuật duyệt các ô liền kề:

Duyệt 4 ô chung cạnh với ô [i][j]

```
#include <bits/stdc++.h>
using namespace std;
int dx[4] = {-1, 0, 0, 1};
int dy[4] = {0, -1, 1, 0};

int main(){
    int a[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int i = 1, j = 1;
    for(int k = 0; k < 4; k++){
        int i1 = i + dx[k], j1 = j + dy[k];
        cout << a[i1][j1] << " ";
    }
}
```

OUTPUT: 2 4 6 8

5. Kĩ thuật duyệt các ô liền kề:

Duyệt 8 ô chung đỉnh với ô [i][j]

```
#include <bits/stdc++.h>
using namespace std;
int dx[8] = {-1, -1, -1, 0, 0, 1, 1, 1};
int dy[8] = {-1, 0, 1, -1, 1, -1, 0, 1};

int main(){
    int a[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int i = 1, j = 1;
    for(int k = 0; k < 8; k++){
        int i1 = i + dx[k], j1 = j + dy[k];
        cout << a[i1][j1] << " ";
    }
}
```

OUTPUT: 1 2 3 4 6 7 8 9

5. Kĩ thuật duyệt các ô liền kề:

1	2	3	4	5	6
7	8	9	1	2	5
1	2	1	0	3	5
1	2	1	3	4	9
1	2	1	3	0	4
1	8	7	6	2	9

Duyệt 8 ô xung quanh nước đi của quân mã

```
#include <bits/stdc++.h>
using namespace std;
int dx[8] = {-2, -2, -1, -1, +1, +1, +2, +2};
int dy[8] = {-1, +1, -2, +2, -2, +2, -1, +1};

int main(){
    int a[6][6] = {
        {1, 2, 3, 4, 5, 6},
        {7, 8, 9, 1, 2, 5},
        {1, 2, 1, 0, 3, 5},
        {1, 2, 1, 3, 4, 9},
        {1, 2, 1, 3, 0, 4},
        {1, 8, 7, 6, 2, 9}
    };
    int i = 2, j = 3;
    for(int k = 0; k < 8; k++){
        int i1 = i + dx[k], j1 = j + dy[k];
        cout << a[i1][j1] << " ";
    }
}
```

OUTPUT: 3 5 8 5 2 9 1 0

MẢNG CỘNG DỒN

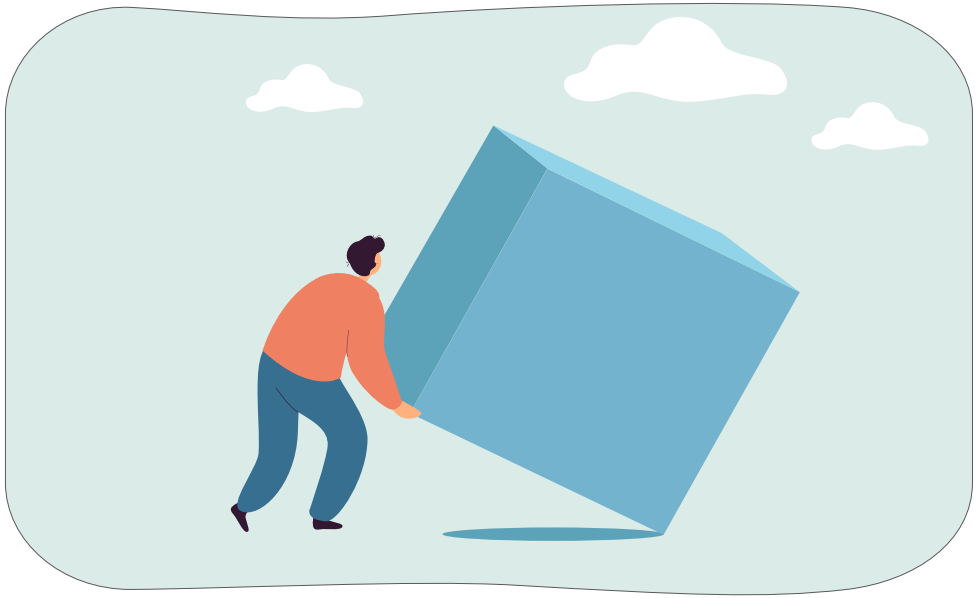
Mảng cộng dồn (tiền tố) là một mảng giúp các bạn có thể nhanh chóng tính toán tổng các phần tử trong các đoạn liên tiếp từ chỉ số left tới chỉ số right.

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
```