LECTURE 06

2D ARRAY

Big-O Coding

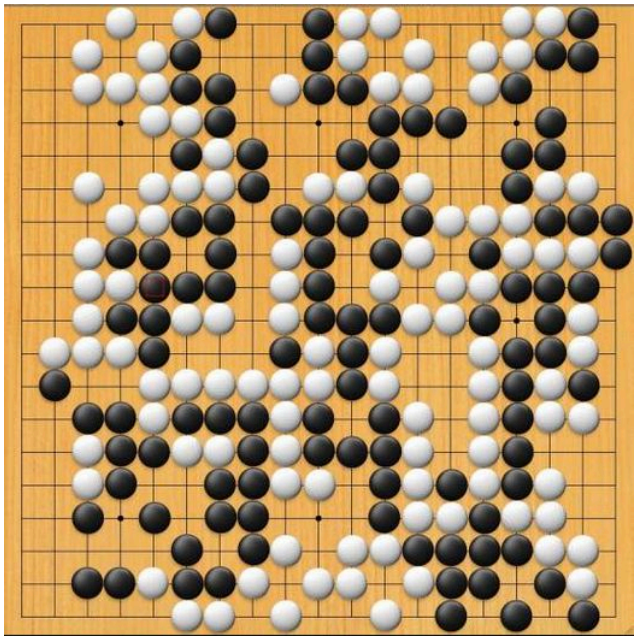
Website: www.bigocoding.com

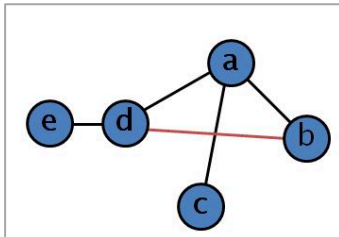
Giới thiệu

Mảng 2 chiều là gì?

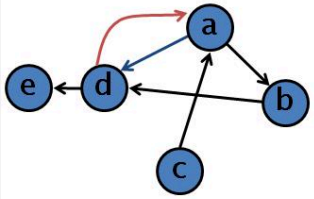
- Mảng 2 chiều, ma trận, 2-dimensional array, matrix.
- Cấu trúc dữ liệu dùng để biểu diễn các đối tượng dữ liệu ở dạng nhiều dãy khác nhau, được chia thành **nhiều dòng và nhiều cột**.

Ứng dụng mảng 2 chiều





	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	0	0
d	1	1	0	0	1
e	0	0	0	1	0



	a	b	c	d	e
a	0	1	0	1	0
b	0	0	0	1	0
c	1	0	0	0	0
d	1	0	0	0	1
e	0	0	0	0	0

Hình ảnh mảng 2 chiều

- Mảng 2 chiều có 3 dòng và 4 cột.

4	18	9	3
12	45	74	15
84	87	75	67

Khai báo và sử dụng

1. Tạo mảng 2 chiều (C++)

```
int a[100][50];
int m = 3, n = 4;
```

```
int a[][4] =
    {{1, 2, 3, 4},
     {5, 6, 7, 8},
     {9, 10, 11, 12}};
int m = 3, n = 4;
```

- Phải xác định số dòng, số cột của mảng lúc biên dịch.
- Phải xác định kiểu dữ liệu mỗi phần tử trong mảng (int).
- Sử dụng 2 biến số nguyên m (số dòng) và n (số cột) đi kèm để biết được kích thước thật sự của mảng.

1	2	3	4
5	6	7	8
9	10	11	12

1. Tạo mảng 2 chiều (Java)

```
int [][]a;
int m = 3;
int n = 4;
a = new int[m][];
for(int i = 0; i < m; i++)
    a[i] = new int[n];
```

```
int [][]a =
        {{1, 2, 3, 4},
         {5, 6, 7, 8},
         {9, 10, 11, 12}};
```

- Có thể qui định số dòng và số cột của mảng lúc chạy.
- Phải xác định kiểu dữ liệu mỗi phần tử trong mảng (int).

1. Tạo mảng 2 chiều (Java)

1	2	3	4

1	2	3	4
5	6	7	8

1	2	3	4
5	6	7	8
9	10	11	12

1. Tạo mảng 2 chiều (Python)

```
a1 = [1, 2, 3, 4]
a2 = [5, 6, 7, 8]
a3 = [9, 10, 11, 12]
```

```
a = [a1, a2, a3]
```

```
a = []
a.append(a1)
a.append(a2)
a.append(a3)
```

- Coi mỗi dòng là 1 mảng một chiều.
- Gộp các mảng này lại, ta được mảng 2 chiều.

1. Tạo mảng 2 chiều (Python)

1	2	3	4

1	2	3	4
5	6	7	8

1	2	3	4
5	6	7	8
9	10	11	12

2. Xác định kích thước mảng 2 chiều (C++)

```
int a[][4] =
    {{1, 2, 3, 4},
     {5, 6, 7, 8},
     {9, 10, 11, 12}};
int m = 3, n = 4;
```

```
cout << m << endl;
cout << n << endl;
```

- Sử dụng 2 biến số nguyên m (số dòng) và n (số cột) đi kèm để biết được kích thước thật sự của mảng.

2. Xác định kích thước mảng 2 chiều (Java)

```
int [][]a;
int m = 3;
int n = 4;
a = new int[m][];
for(int i = 0; i < m; i++)
    a[i] = new int[n];
```

```
System.out.println(a.length); // rows
System.out.println(a[0].length); // columns
```

- Sử dụng thuộc tính length để biết được số dòng và số cột của mảng.

2. Xác định kích thước mảng 2 chiều (Python)

```
a1 = [1, 2, 3, 4]
a2 = [5, 6, 7, 8]
a3 = [9, 10, 11, 12]
a = [a1, a2, a3]
```

```
print(len(a))
print(len(a[0]))
```

- Sử dụng hàm len() để biết được số dòng và số cột trong mảng.

3. Truy xuất 1 phần tử trong mảng

- Sử dụng index để truy xuất 1 phần tử trong mảng: a[i][j], index bắt đầu từ 0.
- Trong đó, nếu mảng có m dòng, n cột:
  - i: chỉ số dòng, bắt đầu từ 0, kết thúc là m-1.
  - j: chỉ số cột, bắt đầu từ 0, kết thúc là n-1.
- Mảng có m dòng, n cột sẽ có m*n phần tử.

VD1: Index hợp lệ

```cpp
int a[][4] =
    {{10, 20, 30, 40},
     {50, 60, 70, 80},
     {90, 100, 110, 120}};
int m = 3, n = 4;

cout << a[0][0] << endl;
cout << a[2][1] << endl;
cout << a[2][3] << endl;
```

```python
a1 = [10, 20, 30, 40]
a2 = [50, 60, 70, 80]
a3 = [90, 100, 110, 120]
a = [a1, a2, a3]


print(a[0][0])
print(a[2][1])
print(a[2][3])
```

```java
int [][] a =
        {{10, 20, 30, 40},
         {50, 60, 70, 80},
         {90, 100, 110, 120}};
System.out.println(a[0][0]);
System.out.println(a[2][1]);
System.out.println(a[2][3]);
```

VD2: Index out of range

```cpp
int a[][4] =
    {{10, 20, 30, 40},
     {50, 60, 70, 80},
     {90, 100, 110, 120}};
int m = 3, n = 4;

cout << a[3][0] << endl;
cout << a[2][10] << endl;
cout << a[4][7] << endl;
```

```
// Giá trị rác
```

```
a1 = [10, 20, 30, 40]
a2 = [50, 60, 70, 80]
a3 = [90, 100, 110, 120]
a = [a1, a2, a3]


print(a[3][0])
print(a[2][10])
print(a[4][7])
```

```
IndexError: list index out of
range
```

```java
int [][] a =
        {{10, 20, 30, 40},
         {50, 60, 70, 80},
         {90, 100, 110, 120}};
System.out.println(a[3][0]);
System.out.println(a[2][10]);
System.out.println(a[4][7]);
```

```
java.lang.ArrayIndexOutOfBoundsException
```

VD3: Index là số âm

```cpp
int a[][4] =
    {{10, 20, 30, 40},
     {50, 60, 70, 80},
     {90, 100, 110, 120}};
int m = 3, n = 4;

cout << a[-1][0] << endl;
cout << a[2][-2] << endl;
cout << a[-1][-3] << endl;
```

```
a1 = [10, 20, 30, 40]
a2 = [50, 60, 70, 80]
a3 = [90, 100, 110, 120]
a = [a1, a2, a3]


print(a[-1][0])
print(a[2][-2])
print(a[-1][-3])
```

```
// Giá trị rác
```

```
90
110
100
```

```java
int [][] a =
        {{10, 20, 30, 40},
         {50, 60, 70, 80},
         {90, 100, 110, 120}};
System.out.println(a[-1][0]);
System.out.println(a[2][-2]);
System.out.println(a[-1][-3]);
```

```
java.lang.ArrayIndexOutOfBoundsException
```

4. Duyệt theo index

```cpp
int a[][4] =
    {{10, 20, 30, 40},
     {50, 60, 70, 80},
     {90, 100, 110, 120}};
int m = 3, n = 4;

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

```python
a1 = [10, 20, 30, 40]
a2 = [50, 60, 70, 80]
a3 = [90, 100, 110, 120]
a = [a1, a2, a3]


for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=" ")

    print()
```

```java
int [][] a =
        {{10, 20, 30, 40},
         {50, 60, 70, 80},
         {90, 100, 110, 120}};
for(int i = 0; i < a.length; i++){
    for(int j = 0; j < a[i].length; j++){
        System.out.printf("%d ", a[i][j]);
    }
    System.out.println();
}
```

5. Duyệt theo phần tử

```
// C++ không hỗ trợ
```

```
int [][] a =
        {{10, 20, 30, 40},
         {50, 60, 70, 80},
         {90, 100, 110, 120}};
for(int []temp: a){
    for(int x: temp){
        System.out.printf("%d ", x);
    }
    System.out.println();
}
```

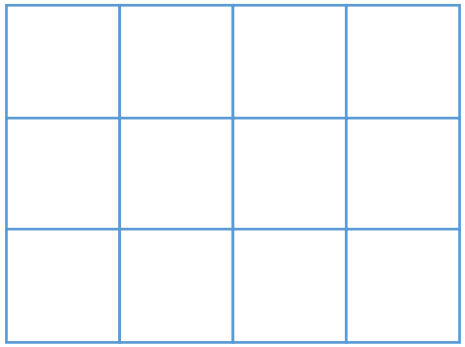
```
a1 = [10, 20, 30, 40]
a2 = [50, 60, 70, 80]
a3 = [90, 100, 110, 120]
a = [a1, a2, a3]

for tempList in a:
    for x in tempList:
        print(x, end=" ")
    print()
```

6. Đọc mảng 2 chiều (C++)

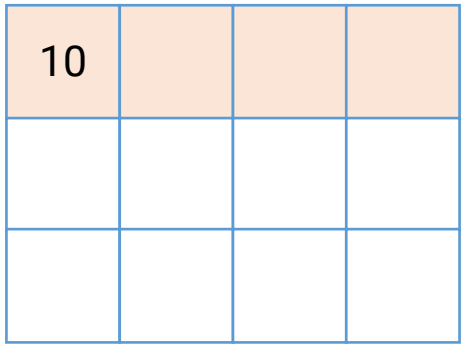
```
3 4
10 20 30 40
50 60 70 80
90 100 110 120
```

```
int a[100][100];
int m, n;
cin >> m >> n;
for(int i = 0; i < m; i++){
    for(int j = 0; j < n; j++){
        cin >> a[i][j];
    }
}
```

Minh họa



```
cin >> a[0][0]
```



```
cin >> a[0][1]
```

10	20		

```
cin >> a[0][2]
```

10	20	30	

```
cin >> a[0][3]
```

10	20	30	40

```
cin >> a[1][0]
```

10	20	30	40
50			

```
cin >> a[1][1]
```

10	20	30	40
50	60		

```
cin >> a[1][2]
```

10	20	30	40
50	60	70	

```
cin >> a[1][3]
```

10	20	30	40
50	60	70	80

```
cin >> a[2][0]
```

10	20	30	40
50	60	70	80
90			

```
cin >> a[2][1]
```

10	20	30	40
50	60	70	80
90	100		

```
cin >> a[2][2]
```

10	20	30	40
50	60	70	80
90	100	110	

```
cin >> a[2][3]
```

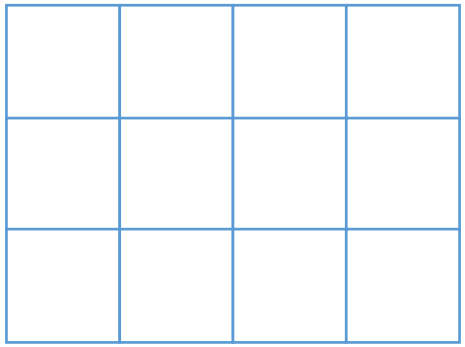
10	20	30	40
50	60	70	80
90	100	110	120

6. Đọc mảng 2 chiều (Java)

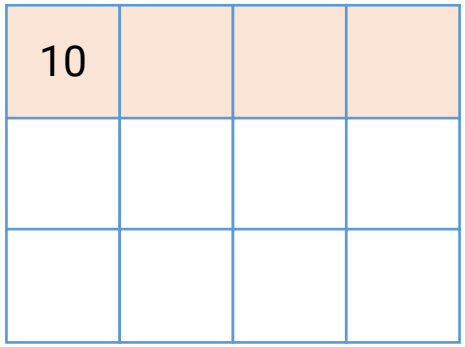
```
3 4
10 20 30 40
50 60 70 80
90 100 110 120
```

```java
int[][]a;
int m, n;
Scanner sc = new Scanner(System.in);
m = sc.nextInt();
n = sc.nextInt();

a = new int[m][];
for(int i = 0; i < m; i++)
    a[i] = new int[n];

for(int i = 0; i < m; i++){
    for(int j = 0; j < n; j++){
        a[i][j] = sc.nextInt();
    }
}
```

Minh họa



```
a[0][0] = sc.nextInt();
```



```
a[0][1] = sc.nextInt();
```

10	20		

```
a[0][2] = sc.nextInt();
```

10	20	30	

```
a[0][3] = sc.nextInt();
```

10	20	30	40

```
a[1][0] = sc.nextInt();
```

10	20	30	40
50			

```
a[1][1] = sc.nextInt();
```

10	20	30	40
50	60		

```
a[1][2] = sc.nextInt();
```

10	20	30	40
50	60	70	

```
a[1][3] = sc.nextInt();
```

10	20	30	40
50	60	70	80

```
a[2][0] = sc.nextInt();
```

10	20	30	40
50	60	70	80
90			

```
a[2][1] = sc.nextInt();
```

10	20	30	40
50	60	70	80
90	100		

```
a[2][2] = sc.nextInt();
```

10	20	30	40
50	60	70	80
90	100	110	

```
a[2][3] = sc.nextInt();
```

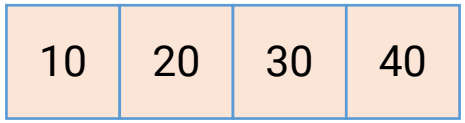
10	20	30	40
50	60	70	80
90	100	110	120

6. Đọc mảng 2 chiều (Python)

```
3 4
10 20 30 40
50 60 70 80
90 100 110 120
```

```python
m, n = map(int, input().split())
a = []
for i in range(m):
    temp = list(map(int, input().split()))
    a.append(temp)
```

Đọc được dòng 0: temp = [10, 20, 30, 40]



Đọc được dòng 1: temp = [50, 60, 70, 80]

10	20	30	40
50	60	70	80

Đọc được dòng 2: temp = [90, 100, 110, 120]

10	20	30	40
50	60	70	80
90	100	110	120

Các kĩ thuật xử lí trên mảng 2 chiều

BT1 – TÍNH TỔNG CÁC DÒNG

- Viết chương trình tính tổng của từng dòng trong ma trận số nguyên.

Kĩ thuật xử lí trên từng dòng

Kĩ thuật xử lí trên từng dòng

- Đây là kĩ thuật dùng để tìm dòng hoặc tìm giá trị thuộc dòng nào đó thỏa mãn điều kiện cho trước.
- Khi xử lí trên dòng, tức là đã xác định trên dòng nào, do đó, chỉ cần duyệt các phần tử trên dòng đó.

4	18	9	3
12	45	74	15
84	87	75	67

BT1 – Gợi ý

1. Đọc vào mảng 2 chiều a, có m dòng, n cột.
2. Sử dụng vòng lặp **for i**, duyệt theo số dòng m.
    1. Xem a[i] là mảng 1 chiều có n phần tử. Sử dụng vòng lặp **for j**, duyệt theo n, để tính tổng các phần tử **a[i]**[j].
    2. In tổng của dòng i ra màn hình.

BT1 – Minh họa

(2) for j: 0..n-1

j = 0	j = 1	j = 2	j = 3

(1) for i: 0..m-1

i = 0
i = 1
i = 2

4	18	9	3
12	45	74	15
84	87	75	67

BT1 – Minh họa – Tính tổng từng dòng

4	18	9	3
12	45	74	15
84	87	75	67

```
i = 0 # tính tổng của dòng 0
    sum = 0
    j = 0, sum += a[0][0] = 0 + 4 = 4
    j = 1, sum += a[0][1] = 4 + 18 = 22
    j = 2, sum += a[0][2] = 22 + 9 = 31
    j = 3, sum += a[0][3] = 31 + 3 = 34
In theo format 0: 34
```

4	18	9	3
12	45	74	15
84	87	75	67

```
i = 1 # tính tổng của dòng 1
    sum = 0
    j = 0, sum += a[1][0] = 0 + 12 = 12
    j = 1, sum += a[1][1] = 12 + 45 = 57
    j = 2, sum += a[1][2] = 57 + 74 = 131
    j = 3, sum += a[1][3] = 131 + 15 = 146
In theo format 1: 146
```

BT1 – Minh họa – Tính tổng từng dòng

4	18	9	3
12	45	74	15
84	87	75	67

```
i = 2 # tính tổng của dòng 2
    sum = 0
    j = 0, sum += a[2][0] = 0 + 84 = 84
    j = 1, sum += a[2][1] = 84 + 87 = 171
    j = 2, sum += a[2][2] = 171 + 75 = 246
    j = 3, sum += a[2][3] = 246 + 67 = 313
In theo format 2: 313
```

BT2 – CỘT TOÀN ÂM

- Viết chương trình liệt kê các cột toàn âm của ma trận các số nguyên.

Kĩ thuật xử lí trên từng cột

Kĩ thuật xử lí trên từng cột

- Đây là kĩ thuật dùng để tìm cột hoặc tìm giá trị thuộc cột nào đó thỏa mãn điều kiện cho trước.
- Khi xử lí trên cột, tức là đã xác định trên cột nào, do đó, chỉ cần duyệt các các phần tử trên cột đó.

4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67

BT2 – Gợi ý

1. Đọc vào mảng 2 chiều a, có m dòng, n cột.
2. Sử dụng vòng lặp (**for j**), duyệt theo số cột n.
    1. Sử dung vòng lặp (**for i**), duyệt theo số dòng, dùng **kĩ thuật đặt cờ hiệu**, tìm một ô **a**[i]**[j]** >= 0 để chỉ ra cột j ko toàn âm. (L05P04)
    2. In j nếu cột j là toàn âm.

BT2 – Minh họa

(1) for j: 0..n-1

j = 0	j = 1	j = 2	j = 3

(2) for i: 0..m-1

i = 0
i = 1
i = 2

4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67

BT2 – Minh họa

4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67

j = 0 # kiểm tra toàn âm trên cột 0
flag = True
i = 0, a[0][0] = 4 >= 0: True, flag = False
i = 1, a[1][0] = -7 >= 0: False
i = 2, a[2][0] = -8 >= 0: False
Không in cột 0

4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67

j = 1 # kiểm tra toàn âm trên cột 1
flag = True
i = 0, a[0][1] = 18 >= 0: True, flag = False
i = 1, a[1][1] = -2 >= 0: False
i = 2, a[2][1] = 87 >= 0: True, flag = False
Không in cột 1

BT2 – Minh họa

4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67

j = 2 # kiểm tra toàn âm trên cột 2
flag = True
i = 0, a[0][2] = -9 >= 0: False
i = 1, a[1][2] = -4 >= 0: False
i = 2, a[2][2] = -5 >= 0: False
In cột 2

4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67

j = 3 # kiểm tra toàn âm trên cột 3
flag = True
i = 0, a[0][3] = 3 >= 0: True, flag = False
i = 1, a[1][3] = -1 >= 0: False
i = 2, a[2][3] = 67 >= 0: True, flag = False
Không in cột 3

BT3 – ĐẾM NGUYÊN TỐ TRÊN BIÊN

- Đếm xem trên biên của ma trận có bao nhiêu phần tử là số nguyên tố.

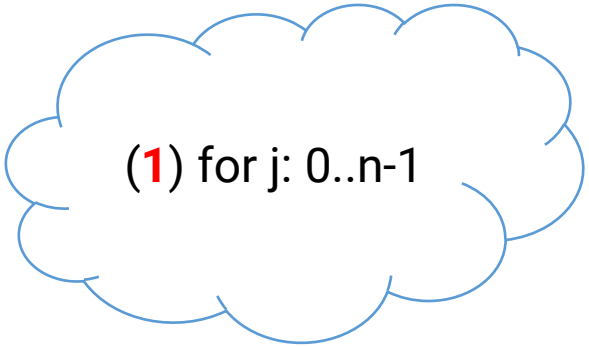
Kĩ thuật xử lí trên 4 đường biên

Kĩ thuật xử lí trên đường biên

- Đây là kĩ thuật dùng để xử lý các giá trị thuộc phía viền bên ngoài của ma trận.
- Vòng lặp 1: xử lí trên dòng đầu tiên 0 và dòng cuối m-1.
- Vòng lặp 2: xử lí trên cột đầu tiên 0 và cột cuối n-1.
- Cẩn thận, đừng xử lí lặp lại 2 lần các giá trị ở 4 góc.

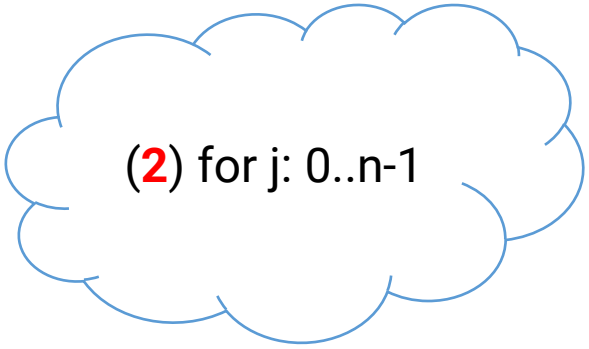
4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67
-8	7	19	28

BT3 – Minh họa



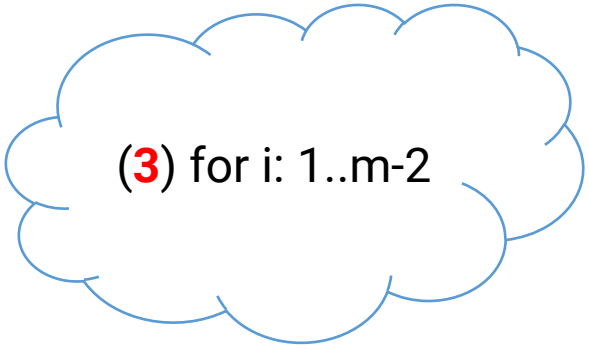
4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67
-8	7	19	28

BT3 – Minh họa



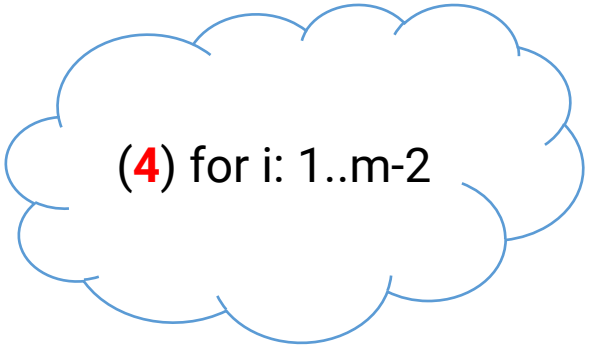
4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67
-8	7	19	28

BT3 – Minh họa



4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67
-8	7	19	28

BT3 – Minh họa



4	18	-9	3
-7	-2	-4	-1
-8	87	-5	67
-8	7	19	28

BT3 – Gợi ý

1. Đọc vào mảng 2 chiều a, m dòng, n cột.
2. Sử dụng **kĩ thuật đếm**, khởi tạo count = 0.
3. Vòng lặp 1 (for j), duyệt theo số cột n.
   - Xem **dòng đầu a[0] và dòng cuối a[m-1]** là các mảng 1 chiều, sử dụng **kĩ thuật đếm** (L05P05), nếu **a[i]**[j] là số nguyên tố thì tăng count lên 1.
4. Vòng lặp 2 (for i), duyệt theo số dòng m.
   - Xem **cột đầu a[0] và cột cuối a[n-1]** là các mảng 1 chiều, sử dụng **kĩ thuật đếm**, nếu **a**[i]**[j]** là số nguyên tố thì tăng count lên 1.
5. In kết quả.

BT4 – SỐ NGUYÊN TỐ TRÊN ĐƯỜNG CHÉO CHÍNH

- Viết chương trình đếm xem trên đường chéo chính của ma trận những số nào là số nguyên tố.

Kĩ thuật xử lí trên đường chéo chính

- Khi ma trận có số dòng và số cột bằng nhau thì ta gọi đó là ma trận vuông.
- Ma trận vuông sẽ có 2 loại đường chéo như sau:
  - Đường chéo chính.
  - Đường chéo phụ.

Đường chéo trong ma trận vuông

	j = 0	j = 1	j = 2	j = 3
i = 0	4	18	-9	3
i = 1	-7	-2	-4	-1
i = 2	-8	87	-5	67
i = 3	-8	7	19	28

Đường chéo chính
j = i

	j = 0	j = 1	j = 2	j = 3
i = 0	4	18	-9	3
i = 1	-7	-2	-4	-1
i = 2	-8	87	-5	67
i = 3	-8	7	19	28

Đường chéo chính
j = n – 1 - i

BT4 – Gợi ý

1. Đọc vào mảng 2 chiều a, có n dòng, n cột.
2. Sử dụng vòng lặp (for i), duyệt theo n.
   - Sử dụng kĩ thuật đếm, nếu a[i][i] là số nguyên tố thì tăng đếm lên 1.

BT5 – TÍCH SỐ NGUYÊN TỐ TRÊN ĐƯỜNG CHÉO PHỤ

- Viết chương trình tính tích các số nguyên tố trên **đường chéo phụ** của ma trận vuông các số nguyên.

BT5 – Gợi ý

1. Đọc vào mảng 2 chiều a, có n dòng, n cột.
2. Sử dụng vòng lặp (for i), duyệt theo n.
   - Sử dụng kĩ thuật đếm, nếu a[i][n-i-1] là số nguyên tố thì nhân vào ans.

Hỏi đáp