Bài 1. CÁC TOÁN TỬ XỬ LÝ BIT VÀ ỨNG DỤNG

1.1. Số nguyên và độ dài bit

Ta biết rằng mỗi số nguyên được biểu diễn bằng một dãy bit, độ dài của dãy bit phụ thuộc vào kiểu số nguyên đó.

- int8_t, uint8_t: 8 bit
- int16_t, uint16_t: 16 bit
- int32_t, uint32_t: 32 bit
- int64_t, uint64_t: 64 bit

C++ có những tên chuẩn mô tả kiểu dữ liệu số nguyên như **int**, **long long**, hay **unsigned long long**. Tùy theo hệ điều hành và chương trình dịch, các kiểu dữ liệu chuẩn này sẽ được ánh xạ vào một trong 8 kiểu số nguyên nói trên, chẳng hạn với trình dịch g++ 32 bit, kiểu **int** tương đương với **int32_t**, kiểu **long long** tương đương với **int64_t** còn kiểu **unsigned long long** tương đương với **uint64_t**.

Với hai kiểu dữ liệu tương đương, dùng loại tên nào phải tùy theo mục đích sử dụng và bảo trì. Hiện tại thì int và int32_t là tương đương, nhưng không có gì đảm bảo chúng tương đương trong các hệ điều hành và trình dịch thế hệ kế tiếp: int là kiểu số nguyên có dấu tính nhanh nhất trên máy còn int32_t là kiểu số nguyên có dấu chiếm đúng 32 bit. Vậy nên muốn khai báo biến số nguyên mang kiểu có dấu tính nhanh nhất thì nên dùng kiểu int, còn nếu một biến số nguyên có dấu chắc chắn chiếm 32 bit bất kể trên hệ thống nào thì nên dùng int32_t. Tương tự như vậy, nếu muốn một biến số nguyên không dấu mang kiểu lớn nhất trên máy thì nên dùng kiểu unsigned long long, còn nếu muốn một biến số nguyên không dấu chắc chắn chiếm 64 bit thì nên dùng uint64_t. Tất nhiên đây là văn hóa sử dụng kiểu, còn nếu biết chắc chương trình của mình chỉ chạy trên hệ điều hành và trình dịch nào thì chẳng có gì khác nhau về cách dùng tên giữa hai kiểu tương đương*.

Trong bài học này, ta quan tâm tới các số nguyên không âm, tức là nếu một số nguyên biểu diễn bởi z bit: $b_{z-1}b_{z-2}\ldots b_1b_0$ thì giá trị số nguyên đó sẽ bằng:

$$\sum_{i=0}^{z-1} b_i.2^i = b_{z-1}.2^{z-1} + b_{z-2}.2^{z-2} + \cdots + b_1.2 + b_0$$

Trong hầu hết các ví dụ trong bài , bạn cũng có thể dùng các kiểu số nguyên có dấu hay không dấu đều được do ta chỉ quan tâm tới các bit mà không quan tâm tới giá trị số. Tuy nhiên cẩn

---

* Các tên gọi như int, long long, unsigned, … ta đã quen thuộc. Hệ thống tên có kèm kích thước kiểu cũng rất dễ nhớ: Tiền tố int hay uint chỉ ra kiểu có dấu hay không dấu (có biểu diễn được số âm hay không), tiếp theo là kích thước kiểu 8, 16, 32 hoặc 64, cuối cùng là hậu tố _t

thận với kiểu số nguyên có dấu nếu bạn phải sử dụng lẫn cả phép toán bit và các phép toán số học (+, -, *, /, %)

1.2. Các toán tử xử lý bit

Các toán tử xử lý bit (Bitwise operators) trong một vài trường hợp sẽ làm chương trình hiệu quả hơn hẳn cả về tốc độ và tính đơn giản. Cơ sở của chúng là thực hiện các phép toán logic trên các bit với quy ước bit 1 là true và bit 0 là false.

Các toán tử xử lý bit đều có độ phức tạp $O(1)$, hơn thế chúng xử lý nhanh hơn cả các phép toán số học +, -, *, /, %. Dưới đây ta khảo sát sơ đồ bit biểu diễn số nguyên và các toán tử xử lý bit.

1.2.1. Sơ đồ bit

Một số nguyên z bit được biểu diễn bởi sơ đồ sau:

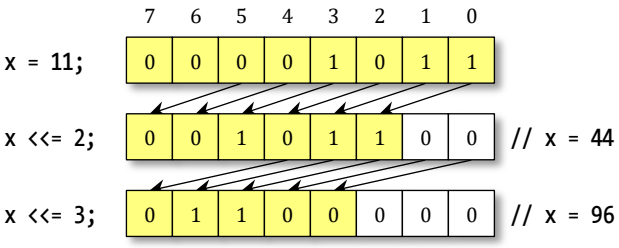
Bit số 0 (bit cuối) được gọi là bit đơn vị hay bit thấp nhất. Bit số $z-1$ được gọi là bit cao nhất. Để phù hợp với ký pháp toán học, sơ đồ bit trong bài này được đánh số từ phải qua trái dù rằng trên thực tế, bộ nhớ lưu giá trị dãy bit từ trái qua phải.

1.2.2. Phép dịch trái và dịch phải

Với x và k là số nguyên.

Biểu thức $x \ll k$ cho kết quả là số nguyên tạo thành từ dãy bit của x dịch trái k vị trí: k bit đầu tiên bị cắt bỏ và k bit 0 được thêm vào cuối dãy bit.

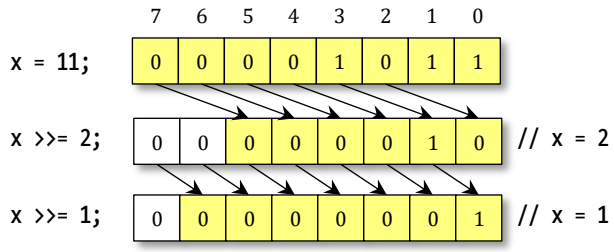
Ví dụ với x kiểu uint8_t



Phép toán $x \ll k$ hoạt động giống như $x.2^k$ (những bit tràn bị cắt bỏ) nhưng phép dịch trái thực hiện nhanh hơn nhiều so với phép nhân.

Biểu thức $x \gg k$ cho kết quả là số nguyên tạo thành từ dãy bit của x dịch phải k vị trí: k bit cuối cùng bị cắt bỏ và k bit 0 được thêm vào đầu dãy bit.

Ví dụ với x kiểu uint8_t



Phép toán $x \ll k$ hoạt động giống như $x/2^k$ nhưng phép dịch phải thực hiện nhanh hơn nhiều so với phép chia nguyên.

Phép dịch trái và dịch phải có độ ưu tiên tương đương trong biểu thức (cấp 7*), độ ưu tiên của chúng thấp hơn các phép toán số học +, -, *, /, % và cao hơn các phép so sánh >, <, <=, >=, ==, !=.

Ví dụ $1 + 5 \ll 2 + 3 = (1 + 5) \ll (2 + 3) = 192$

1.2.3. Phép đảo bit

Phép đảo bit thực hiện trên số nguyên x, ký pháp $\sim x$ cho giá trị bằng một số nguyên tạo thành từ x bằng cách đảo hết bit 0 thành 1 và bit 1 thành 0.

Chú ý rằng trong biểu thức số nguyên, các kiểu dữ liệu nhỏ hơn int sẽ được tính theo kiểu int, vì vậy nếu biến x có kiểu uint32_t, int64_t hay uint64_t thì ~x sẽ nhận giá trị kiểu đó, nếu không biểu thức $\sim x$ sẽ mang giá trị kiểu int. Muốn $\sim x$ mang kiểu cùng với x, phải sử dụng toán tử ép kiểu (typecasting)

Phép đảo bit trong biểu thức số nguyên có độ ưu tiên cấp 3, ngang với phép lấy số đối (-), tức là ưu tiên hơn các phép toán số học +, -, *, /, %, ưu tiên hơn phép toán dịch trái, dịch phải cũng như các phép toán bit khác.

1.2.4. Phép toán AND bit

Phép toán AND bit, ký hiệu x & y thực hiện trên từng cặp bit tương ứng của x và y để tính giá trị các bit kết quả. Nếu coi bit 0 ứng với hằng logic false và bit 1 ứng với hằng logic true thì phép toán AND bit hoạt động giống như phép logic AND (&&)

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

Tức là giá trị bit kết quả bằng 1 nếu và chỉ nếu cả hai bit toán hạng đều là 1.

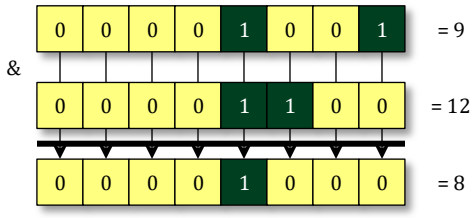
Tính chất: với b là một bit

```
b & 0 = 0 & b = 0
b & 1 = 1 & b = b
```

---

* Cấp càng nhỏ độ ưu tiên càng cao

Ví dụ trên kiểu số nguyên uint8_t, phép toán 9 & 12 sẽ cho kết quả là 8:



Trong biểu thức số nguyên, phép toán & có độ ưu tiên cấp 10

1.2.5. Phép toán OR bit

Phép toán OR bit, ký hiệu $x \mid y$ thực hiện trên từng cặp bit tương ứng của x và y để tính giá trị các bit kết quả. Nếu coi bit 0 ứng với hằng logic false và bit 1 ứng với hằng logic true thì phép toán OR bit hoạt động giống như phép logic OR (||)

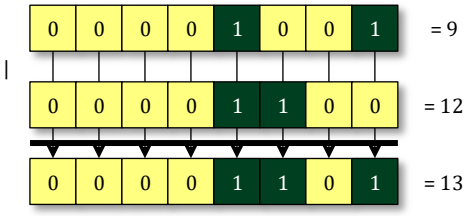
```
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1
```

Tức là giá trị bit kết quả bằng 0 nếu và chỉ nếu cả hai bit toán hạng đều là 0.

Tính chất:

```
b | 0 = 0 | b = b
b | 1 = 1 | b = 1
```

Ví dụ trên kiểu số nguyên uint8_t, phép toán 9 | 12 sẽ cho kết quả là 13:



Trong biểu thức số nguyên, phép toán | có độ ưu tiên cấp 12

1.2.6. Phép toán XOR bit

Phép toán XOR (eXclusive-OR) bit, ký hiệu $x \wedge y$ thực hiện trên từng cặp bit tương ứng của x và y để tính giá trị các bit kết quả.

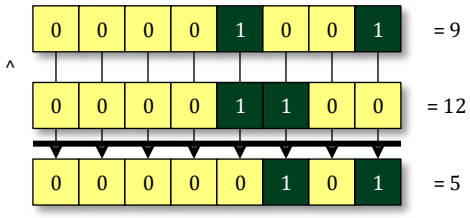
```
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 0
```

Tức là giá trị bit kết quả bằng 0 nếu và chỉ nếu cả hai bit toán hạng giống nhau (cùng là 0 hoặc cùng là 1).

Tính chất:

```
x | 0 = 0 | x = x
x | 1 = 1 | x = ~x
```

Ví dụ trên kiểu số nguyên uint8_t, phép toán 9 ^ 12 sẽ cho kết quả là 5:



Trong biểu thức số nguyên, phép toán ^ có độ ưu tiên cấp 11

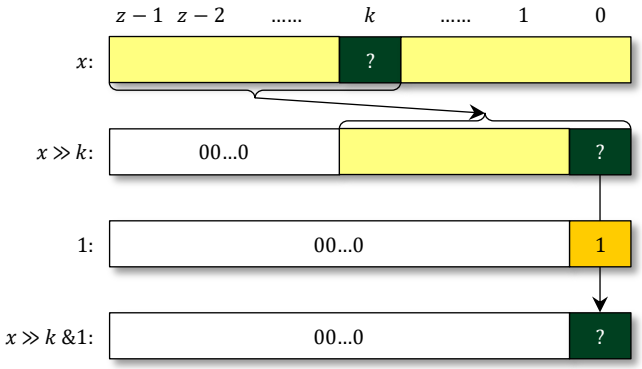
1.2.7. Một vài kỹ thuật cơ bản

❁ **Đọc giá trị bit thứ k của x**

$$x \gg k \,\&\, 1$$

Giải thích: Dựa vào tính chất: $b \,\&\, 1 = b$ và $b \,\&\, 0 = 0$

Phép toán $x \gg k$ dịch dãy bit của x sang phải k bước, bit thứ k của x trở thành bit đơn vị. Giá trị 1 có bit đơn vị là 1 và các bit khác đều bằng 0. Phép toán "& 1" sẽ tách được bit đơn vị (Hình 1-1).



Hình 1-1. Sơ đồ đọc giá trị bit thứ k của x

Chú ý là phép ">>" ưu tiên hơn phép & nên không cần dấu ngoặc trong biểu thức trên

Một điều nữa cần lưu ý là nếu muốn kiểm tra bit thứ k của x có phải là 0 hay không, ta không thể viết:

```
if (x >> k & 1 == 0) …;
```

Lý do là vì phép & ưu tiên thấp hơn phép so sánh == lệnh if khi đó được hiểu là:

```
if ((x >> k) & (1 == 0)) …;
```

Cách viết đúng phải là:

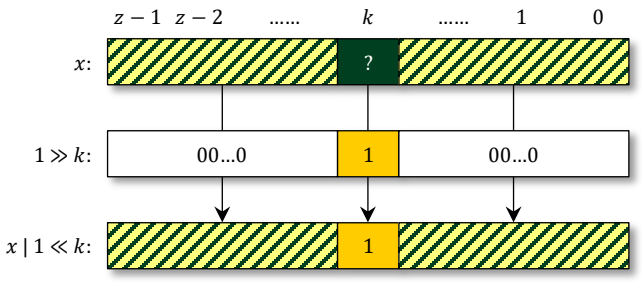
```
if ((x >> k & 1) == 0) …;
```

❁ **Đặt bit thứ k của x thành 1**

$$x \mid= 1 \ll k$$

Giải thích: Dựa vào tính chất $b \mid 1 = 1$ và $b \mid 0 = b$.

$1 \ll k$ là giá trị có duy nhất bit thứ k bằng 1 còn tất cả các bit khác bằng 0. Khi OR bit x với giá trị này, bit thứ k của x được OR bit với 1 nên nó sẽ được đặt thành 1 bất kể trước đó bit này bằng bao nhiêu, các bit khác được OR bit với 0 nên vẫn giữ nguyên giá trị (Hình 1-2)



Hình 1-2. Bật bit thứ k của x thành 1

Chú ý duy nhất là khi thực hiện phép toán $x \mid= 1 \ll k$ là phải cẩn thận với kiểu. Nếu x là kiểu uint64_t và k là kiểu int32_t, biểu thức $1 \ll k$ sẽ được tính theo kiểu int và sẽ cho giá trị 0 nếu $k > 32$ do lỗi tràn. Những viết đúng là:

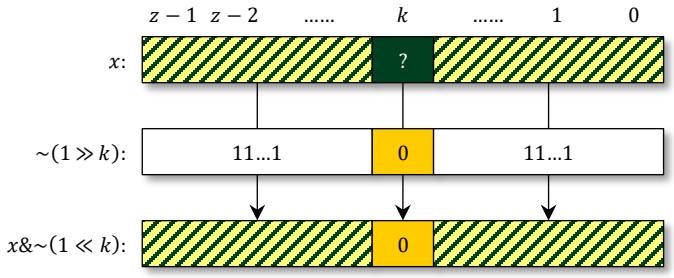
```
x |= 1ULL << k;
x |= 1LL << k;
x |= uint64_t(1) << k;
x |= int64_t(1) << k;
```

❀ **Đặt bit thứ k của x thành 0**

$$x \&= \sim(1 \ll k)$$

Giải thích: Dựa vào tính chất $b \,\&\, 1 = b$ và $b \,\&\, 0 = 0$.

$1 \ll k$ là giá trị có duy nhất bit thứ k bằng 1 còn tất cả các bit khác bằng 0. Khi đảo bit, $\sim(1 \ll k)$ sẽ có duy nhất bit thứ k bằng 0 còn tất cả các bit khác bằng 1. Khi AND bit x với giá trị này, bit thứ k của x sẽ được AND bit với 0 nên nó sẽ được đặt thành 0 bất kể trước đó bit này bằng bao nhiêu, các bit khác được AND bit với 1 nên vẫn giữ nguyên giá trị.



Hình 1-3. Tắt bit thứ k của x thành 0

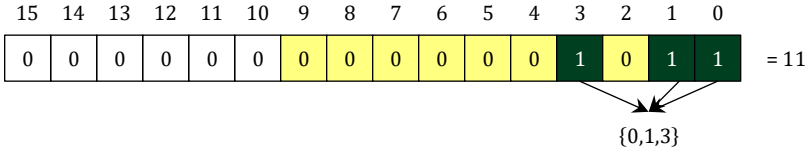
Tương tự như trên, ta cần cẩn thận ép kiểu khi tính toán với các số kích thước lớn hơn int.

1.3. Một số bài toán ví dụ

1.3.1. Dùng bit biểu diễn tập hợp

Khi biểu diễn một tập các giá trị trong phạm vi từ 0 tới $z-1$ với z không quá lớn (≤ 64), ta có thể dùng kiểu số nguyên dài z bit. Bit thứ k bằng 1 tức là phần tử k thuộc tập hợp còn bit thứ k bằng 0 tức là phần tử k không thuộc tập hợp. Với cách biểu diễn như vậy, z có thể làm tròn lên thành 8, 16, 32 hay 64 để phù hợp với kiểu số nguyên được khai báo.

Chẳng hạn với $z = 10$. Số nguyên kiểu int16_t mang giá trị 11 được dùng để biểu diễn tập hợp {0,1,3}



Với cách biểu diễn như vậy

- Phần bù của tập S có thể viết là $\sim S$
- Hợp hai tập A và B ($A \cup B$) có thể viết là $A|B$
- Giao của hai tập A và B ($A \cap B$) có thể viết là $A\&B$
- Hiệu của hai tập A và B ($A - B$) có thể viết là $A\&\sim B$
- Tập gồm những phần tử chỉ thuộc một trong hai tập A và B ($(A - B) \cup (B - A)$) có thể viết là $A\hat{}B$

Trong C++ STL, lớp mẫu bitset<n> đã cài đặt tập hợp chứa các giá trị từ 0 tới $n-1$ với các phép toán trình bày ở trên

Bài toán

Ông John sở hữu n con bò đen và n con bò trắng và ông ta tin rằng màu của mỗi con bò được quyết định bởi một vị trí nào đó trong bộ gen của bò. Ông ta đã lập bản đồ gen từ DNA của cả $2n$ con bò, mỗi con bò có bộ gen là một chuỗi m ký tự $\in \{A, C, G, T\}$ đánh số từ 0 tới $m-1$. Mỗi ký tự tượng trưng cho một đa phân tử hữu cơ gọi là đơn phân (nucleotide).

Chẳng hạn

```
 Vị trí: 0 1 2 3 4 5 6 ... m-1
-----------------------------
  Đen 1: A A T C C C A ... T
  Đen 2: G A T T G C A ... A
  Đen 3: G G T C G C A ... A
Trắng 1: A C T C C C A ... G
Trắng 2: A C T C G C A ... T
Trắng 3: A C T T C C A ... T
```

Ta gọi một vị trí i ($0 \leq i < m$) là vị trí phân loại nếu không tồn tại một con bò đen và một con bò trắng nào có cùng nucleotide ở vị trí đó. Như ví dụ trên thì vị trí 1 có thể là vị trí phân loại, trong khi đó vị trí 0 không phải vị trí phân loại do con bò đen 1 và con bò trắng 1 đều có

nucleotide số 0 là A, vị trí cuối cũng không phải vị trí phân loại do con bò đen 1 và con bò trắng 2 đều có nucleotide tương ứng là T.

Yêu cầu: Đếm số vị trí phân loại

Input

- Dòng 1 chứa hai số nguyên dương $n \leq 100; m \leq 100$
- n dòng tiếp theo, mỗi dòng chứa m ký tự $\in \{A, C, G, T\}$ ứng với bộ gen của một con bò đen
- n dòng tiếp theo, mỗi dòng chứa m ký tự $\in \{A, C, G, T\}$ ứng với bộ gen của một con bò trắng

Output

Ví dụ

Sample Input	Sample Output
3 8 AATCCCAT GATTGCAA GGTCGCAA ACTCCCAG ACTCGCAT ACTTCCAT	1

Thuật toán

Với mỗi vị trí i ta gọi $a[i]$ là tập những nucleotide tại vị trí i của những con bò đen và $b[i]$ là tập những nucleotide tại vị trí i của những con bò trắng. Như ví dụ minh họa, ta có $a[0] = \{A, G\}$ và $b[0] = \{A\}$, hay $a[1] = \{A, G\}$ và $b[1] = \{C\}$

Điều kiện cần và đủ để vị trí i là vị trí phân loại chính là $a[i] \cap b[i] = \emptyset$.

Vì các tập $a[.], b[.]$ có tối đa 4 phần tử $\{A, C, G, T\}$ ta có thể mã hóa các chữ cái này thành các số 0, 1, 2, 3 và từ đó biểu diễn các tập hợp như sau:

```
{A} = 0b0001 = 1 << 0
{C} = 0b0010 = 1 << 1
{G} = 0b0100 = 1 << 2
{T} = 0b1000 = 1 << 3
```

Qua đó, tập $\{A, C, T\}$ có thể biểu diễn bởi $\{A\} \cup \{C\} \cup \{T\} = 1011$

💻 CLASSIFY1.CPP ✓ Đếm số vị trí phân loại

```
#include <iostream>
#include <string>
using namespace std;
const int maxM = 100;
int n, m, a[maxM], b[maxM];
string s;

inline int Code(char c)
{
    switch (c)
    {
        case 'A': return 0b0001;
        case 'C': return 0b0010;
        case 'G': return 0b0100;
```

```
        case 'T': return 0b1000;
    }
    return 0; //disable warnings
}

int main()
{
    cin >> n >> m;
    getline(cin, s); //xuống dòng 2
    fill(a, a + m, 0); //Khởi tạo các tập a[.] đều rỗng
    for (int k = n; k > 0; k--)
    {
        getline(cin, s); //Đọc bộ gen của 1 con bò đen
        for (int i = 0; i < m; i++)
            a[i] |= Code(s[i]); //Bổ sung nucleotide thứ i vào tập a[i]
    }
    fill(b, b + m, 0); //Khởi tạo các tập b[.] đều rỗng
    for (int k = n; k > 0; k--)
    {
        getline(cin, s); //Đọc bộ gen của 1 con bò trắng
        for (int i = 0; i < m; i++)
            b[i] |= Code(s[i]); //Bổ sung nucleotide thứ i vào tập b[i]
    }
    int res = 0;
    for (int i = 0; i < m; i++)
        if ((a[i] & b[i]) == 0) res++; //a[i] ∩ b[i] = ∅ ⇔ a[i] * b[i] == 0
    cout << res;
}
```

💻 CLASSIFY2.CPP ✓ Đếm số vị trí phân loại dùng bitset

```
#include <iostream>
#include <string>
#include <bitset>
using namespace std;
const int maxM = 100;
int n, m;
typedef bitset<4> TSet;
TSet a[maxM], b[maxM];
string s;

inline int Code(char c)
{
    switch (c)
    {
        case 'A': return 0;
        case 'C': return 1;
        case 'G': return 2;
        case 'T': return 3;
    }
    return 0; //disable warnings
}

int main()
{
    cin >> n >> m;
    getline(cin, s); //xuống dòng 2
    for (int i = 0; i < m; i++)
    {
        a[i].reset();
        b[i].reset();
    }
```

```
    for (int k = n; k > 0; k--)
    {
        getline(cin, s); //Đọc bộ gen của 1 con bò đen
        for (int i = 0; i < m; i++)
            a[i].set(Code(s[i])); //Bổ sung nucleotide thứ i vào tập a[i]
    }
    fill(b, b + m, 0); //Khởi tạo các tập b[.] đều rỗng
    for (int k = n; k > 0; k--)
    {
        getline(cin, s); //Đọc bộ gen của 1 con bò trắng
        for (int i = 0; i < m; i++)
            b[i].set(Code(s[i])); //Bổ sung nucleotide thứ i vào tập b[i]
    }
    int res = 0;
    for (int i = 0; i < m; i++)
        if ((a[i] & b[i]).none()) res++; //a[i] ∩ b[i] = Ø
    cout << res;
}
```