2. SỐ HỌC VÀ TỔ HỢP

2.1. Kiểm tra tính nguyên tố của a

2.1.1. Lý thuyết

Số nguyên tố là số tự nhiên lớn hơn 1 chỉ có hai ước là 1 và chính nó. Để kiểm tra một số a có phải là số nguyên tố hay không, chúng ta cần kiểm tra xem a có ước số nào khác ngoài 1 và chính nó không.

- Thuật toán:

Nếu a <= 1, a không phải là số nguyên tố.

Duyệt qua các số từ 2 đến sqrt(a). Nếu a chia hết cho bất kỳ số nào trong khoảng này, thì a không phải là số nguyên tố.

Nếu không tìm thấy ước số nào ngoài 1 và a, thì a là số nguyên tố.

2.1.2. Vận dụng và thực hành

- Chương trình C++ kiểm tra tính nguyên tố:

```
#include <bits\stdc++.h>
using namespace std;
int main() {
   int a;
   bool isPrime = true;
   // Nhập số cần kiểm tra
   cout << "Nhập một số nguyên dương: ";
   cin >> a;
   // Kiểm tra số nhỏ hơn hoặc bằng 1
   if (a <= 1) {
      isPrime = false;
   } else {
      for (int i = 2; i <= …….; i++) {        // Duyệt qua các số từ 2 đến sqrt(a)
         if (a % i == 0) {
            isPrime = false;  // Nếu a chia hết cho i, nó không phải số nguyên tố
            break;  // Thoát khỏi vòng lặp vì không cần kiểm tra thêm
         }
      }
   }
   if (isPrime) {
      cout << a << " là số nguyên tố." << endl;
   } else {
      cout << a << " không phải là số nguyên tố." << endl;
```

0985154409

```
    }
    return 0;
}
```

Ví dụ đầu ra:

Nhập một số nguyên dương: 17 là số nguyên tố.

2.2. Sàng số nguyên tố

2.2.1. Lý thuyết

Sàng nguyên tố (Sieve of Eratosthenes) là một trong những thuật toán hiệu quả nhất để tìm tất cả các số nguyên tố nhỏ hơn hoặc bằng một số cho trước n. Nó đặc biệt hữu ích khi cần tìm nhiều số nguyên tố trong một khoảng.

- Ý tưởng chính:

Tạo một mảng đánh dấu (isPrime[]) với độ dài n + 1 để lưu trạng thái của các số từ 0 đến n (mặc định tất cả đều là số nguyên tố).

Đầu tiên, giả định tất cả các số từ 2 đến n đều là số nguyên tố.

Bắt đầu từ số nhỏ nhất là i=2: Nếu số đó chưa được đánh dấu là không phải số nguyên tố, thì đó là số nguyên tố. Đánh dấu tất cả các bội số của nó từ 2 lên đến n là không phải số nguyên tố.

Tiếp tục quá trình này cho đến i = sqrt{n}

Sau khi hoàn thành, những số nào vẫn chưa bị đánh dấu là không phải số nguyên tố sẽ là số nguyên tố.

2.2.2. Vận dụng và thực hành:

- Chương trình C++ không sử dụng hàm:

```
#include <bits\stdc++.h>
using namespace std;
int main() {
    int n;
    cout << "Nhập số nguyên dương n: ";
    cin >> n;
    bool isPrime[n + 1];
    memset(isPrime, true, sizeof(isPrime));  // Gán tất cả giá trị là true
    isPrime[0] = isPrime[1] = false; // Đặt 0 và 1 không phải số nguyên tố
    // Bắt đầu từ số 2, tiến hành sàng lọc
    for (int i = 2; …….. <= n; i++) {
        if (isPrime[i]) {
            // Đánh dấu tất cả các bội số của i là không phải số nguyên tố
            for (int j = ……..; j <= n; j += i) {
```


0851540 09

```
            isPrime[j] = false;
        }
    }
  }
}
```

2.3. Đồng dư

2.3.1. Lý thuyết Modulo

Modulo là phép toán được sử dụng để tìm phần dư khi một số chia cho một số khác. Ký hiệu của phép toán này thường là %.

Ví dụ:

· 7 mod 3 = 1 (bởi vì 7 chia cho 3 được 2 dư 1)

· 10 mod 4 = 2 (bởi vì 10 chia cho 4 được 2 dư 2)

Phép đồng dư đúng với phép *, + , - nhưng không đúng phép chia.

(a + b) % m = (a % m + b % m) % m.....

(a - b) % m = (a % m - b % m + m) % m, mục đích cộng thêm m để khử âm .....

(a * b) % m = ((a % m) * (b % m)) % m, ưu tiên tính trong ngoặc trước .....

2.3.2. Phần tử nghịch đảo Modulo

Phần tử nghịch đảo (hoặc phần tử đối) của một số a theo một modulo m là một số b sao cho:

$$(a*b) \% m=1$$

Điều này có nghĩa là nếu bạn nhân a với b và lấy kết quả modulo m, bạn sẽ được 1.

Điều kiện tồn tại phần tử nghịch đảo Modulo

Phần tử nghịch đảo tồn tại nếu và chỉ nếu a và m là coprime, tức là gcd(a,m)=1 (gcd là ước số chung lớn nhất).

2.3.3. Tìm phần tử nghịch đảo Modulo bằng Định lý Fermat nhỏ

Định lý Fermat nhỏ phát biểu rằng nếu p là số nguyên tố và a là số nguyên không chia hết cho p, thì:

$$a^{p-1} \% p = 1$$

Từ đó, ta có thể suy ra rằng phần tử nghịch đảo của a modulo p sẽ là:

$$a^{p-2} \% p$$

2.3.4. Vận dụng và thực hành:

Viết chương trình nhập vào số nguyên dương a. Tìm phần tử nghịch đảo của a % p, với p = 1e9+7.

```
const int MOD = 1e9 + 7;
// Hàm tính lũy thừa a^b mod MOD chỉ dùng vòng lặp
int power(int a, int b, int mod) {
```

.....

.....

.....

.....

```
}
int main() {
    int a;
    cout << "Nhap so nguyen duong a: ";
    cin >> a;
    // Tìm nghịch đảo của a mod MOD
    int inverse = power(……, ……., ……..);
    cout << "Nghich dao cua " << a << " mod " << MOD << " la: " << inverse << endl;
    return 0;
}
```

2.4. Tổ hợp chập k của n

2.4.1. Lý thuyết

Trong toán học, tổ hợp chập k của n là số cách chọn ra k phần tử từ một tập hợp gồm n phần tử mà không phân biệt thứ tự. Điều này có nghĩa là chỉ quan tâm đến việc chọn những phần tử nào, không cần biết thứ tự sắp xếp của chúng.

- Công thức tính:

  Số tổ hợp chập k của n được ký hiệu là C(n,k) và được tính theo công thức:

  $$C(n,k)=n!/(k!(n-k)!)$$

  Trong đó:

  n! (giai thừa của n) là tích của tất cả các số từ 1 đến n: $n!=n\times(n-1)\times\square\times2\times1$

  k! và (n−k)! lần lượt là giai thừa của k và (n-k).

- Ví dụ:

  Cho tập hợp có 5 phần tử: {A, B, C, D, E}, cần chọn 3 phần tử từ tập này.

  Số tổ hợp chập 3 của 5 là:

  $$C(5,3)=5!/(3!(5-3)!) = 5\times4\times3!/(3!\times2!) =10$$

  Các tổ hợp gồm: {A, B, C}, {A, B, D}, {A, B, E}, {A, C, D}, {A, C, E}, {A, D, E}, {B, C, D}, {B, C, E}, {B, D, E}, {C, D, E}.

- Tính đối xứng:

  $$C(n,k)=C(n,n-k)$$

  Điều này có nghĩa là số cách chọn k phần tử từ n phần tử cũng bằng số cách chọn (n-k) phần tử từ n phần tử.

0985154409

- Quan hệ truy hồi (Recursive Formula):

$$C(n,k)=C(n-1,k-1)+C(n-1,k)$$

Điều này cho phép tính các giá trị tổ hợp bằng cách sử dụng các giá trị nhỏ hơn.

- Trường hợp đặc biệt:

$$C(n,0)=C(n,n)=1$$

Chỉ có một cách để chọn 0 phần tử hoặc chọn tất cả các phần tử.

2.4.2. Vận dụng và thực hành:

2.4.2.1. Viết chương trình nhập vào số nguyên dương n và k. Tính tổ hợp chập k của n?

.....

.....

.....

.....

.....

.....

2.4.2.2. Viết chương trình nhập vào số nguyên dương n và k. Tính tổ hợp chập k của n đồng dư cho p, với p = 1e9+7.

.....

.....

.....

.....

.....

.....

2.5. Hoán vị của n phần tử

Hoán vị của n phần tử là một cách sắp xếp tuần tự của tất cả các phần tử đó. Nếu tập hợp các phần tử là S={1,2,…,n}, thì một hoán vị của S là một dãy sắp xếp lại các phần tử của S.

Ví dụ, với tập hợp S={1,2,3}, các hoán vị có thể có là:

(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)

Số lượng hoán vị: Số lượng hoán vị của n phần tử được tính bằng công thức:

$$P(n)=n!$$

$$n!=n\times(n-1)\times(n-2)\times\square\times1$$

Ví dụ:

Với n=3, số lượng hoán vị là $3!=3\times2\times1=6$

Với n=4, số lượng hoán vị là $4!=4\times3\times2\times1=24$

2.5.1. Viết chương trình nhập vào số nguyên dương n. Tính số hoán vị của n phần tử đồng dư cho p, với p = 1e9+7.

0985154409

.....

.....

.....

.....

2.6. Chỉnh hợp chập k của n phần tử

Chỉnh hợp chập k của n phần tử là cách sắp xếp k phần tử từ n phần tử có phân biệt thứ tự. Khác với tổ hợp, chỉnh hợp chập k của n phần tử quan tâm đến thứ tự các phần tử trong mỗi cách chọn.

Ký hiệu chỉnh hợp chập k của n phần tử là A(n,k).

Công thức: $A(n,k)=n!/(n-k)!$, trong đó n! là giai thừa của n, và (n−k)! là giai thừa của n−k.

Ví dụ: Giả sử n=5, k=3. Ta muốn tính số cách sắp xếp 3 phần tử từ 5 phần tử. Khi đó:

$$A(5,3)=5!/(5-3)!=5\times4\times3\times2\times1/(2\times1)=5\times4\times3=60$$

Vậy, có 60 cách sắp xếp 3 phần tử từ 5 phần tử. Có thể liệt kê: (1 2 3), (1 3 2), …, (5 4 3)

2.6.1. Viết chương trình nhập vào số nguyên dương n và k. Tính chỉnh hợp chập k của n đồng dư cho p, với p = 1e9+7.

.....

.....

.....

.....

.....

2.7. Hoán vị lặp

Là cách sắp xếp các phần tử trong một tập hợp mà một số phần tử có thể xuất hiện nhiều lần. Khi đó, thứ tự của các phần tử vẫn được phân biệt, nhưng vì có sự trùng lặp, số lượng hoán vị sẽ giảm so với hoán vị của các phần tử duy nhất.

Công thức tính số lượng hoán vị lặp: Giả sử ta có n phần tử, trong đó:

a_1 phần tử giống nhau thuộc loại 1,

a_2 phần tử giống nhau thuộc loại 2,

...

a_k phần tử giống nhau thuộc loại k.

Số lượng hoán vị có thể có được tính theo công thức:

$$P=n!/(a_1!\times a_2!\times\ldots\times a_k!)$$

Ví dụ: Giả sử ta có 6 phần tử gồm: A, A, B, B, C, D. Tổng số phần tử n=6. Số lần xuất hiện của các phần tử là: A: 2 lần, B: 2 lần, C: 1 lần, D: 1 lần. Số lượng hoán vị của 6 phần tử này là:

$P=6!/(2!\times2!\times1!\times1!)=180$. Vậy có 180 cách sắp xếp các phần tử này.

2.7.1. Viết chương trình nhập vào một xâu các ký tự in HOA không có khoảng trắng. Tính xem có bao nhiêu xâu hoán vị từ xâu này đồng dư cho p, với p = 1e9+7.

0935154409

.....

.....

.....

.....

.....

2.8. Dãy Fibonacci

Là một dãy số trong toán học được định nghĩa bởi các phần tử đầu tiên là 0 và 1, và mỗi phần tử tiếp theo là tổng của hai phần tử trước nó. Dãy này được đặt theo tên của nhà toán học người Ý, Leonardo Fibonacci, và có nhiều ứng dụng trong toán học, khoa học máy tính, và tự nhiên.

Dãy Fibonacci được xác định như sau:

$F0=0$

$F1=1$

$Fn=Fn-1+Fn-2$, với $n\geq 2$

Với quy tắc trên, các số đầu tiên của dãy Fibonacci là:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

2.8.1. Viết chương trình nhập vào số n, với n < 1e9. Tính F_n đồng dư cho p, với p = 1e9+7.

.....

.....

.....

.....

.....

2.8.2. Có bao nhiêu cách lát một cái sân có kích thước 2xn bằng các viên gạch 1x2, biết rằng không được để trống, n < 1e9. Kết quả đồng dư cho p, với p = 1e9+7.

.....

.....

.....

.....

.....

2.9. Số Catalan

2.9.1. Số Catalan

Số Catalan là một dãy số tự nhiên xuất hiện trong nhiều bài toán tổ hợp khác nhau, đặc biệt liên quan đến các cấu trúc cây, dấu ngoặc, và đường đi lưới. Các số này được đặt theo tên nhà toán học người Bỉ Catalan.

Số Catalan thứ n được định nghĩa bởi công thức:

$$Catalan(n)=C(2n, n)/(n+1) = (2n)!/((n+1)!*n!)$$

với $Catalan(0)=1$.

0985154409

Các số Catalan đầu tiên là:

1, 1, 2, 5, 14, 42, 132, 429, …

Công thức truy hồi:

Catalan(n)=sum(Catalan(i)*Catalan(n-1-i)), với i: 0..n-1.

2.9.2. Ý nghĩa tổ hợp của số Catalan

Số Catalan xuất hiện trong nhiều bài toán tổ hợp. Một vài ứng dụng phổ biến bao gồm:

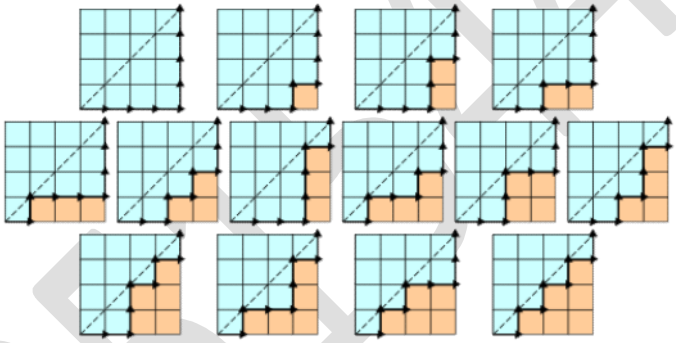
✓ Cây nhị phân: Số cách khác nhau để tạo cây nhị phân có n+1 lá với n nút bên trong.

✓ Dấu ngoặc đúng: Số cách đặt n cặp dấu ngoặc sao cho chúng hợp lệ.

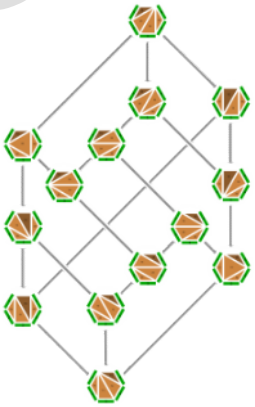
Ví dụ: Giả sử n=3: Catalan(3)=C(2*3, 3)/(3+1)=5

((())), (()()), (())(), ()(()), ()()()

✓ Đường đi lưới: Số cách đi từ điểm (0, 0) đến điểm (n, n) mà không vượt quá đường chéo trong mặt phẳng tọa độ.



✓ Phân hoạch đa giác: Số cách phân hoạch một đa giác lồi với n+2 cạnh thành các tam giác bằng cách kẻ các đường chéo không cắt nhau.



2.9.3. Viết chương trình nhập vào số nguyên dương n. Tính Catalan(n) đồng dư cho p, với p=1e9+7.

.....

.....

.....

.....

0985-1409