NGUYỄN XUÂN HUY

DIVIDE AND CONQUER

Hà Nội, 2021

MỤC LỤC

Đệ quy (Recursive)

Để xây dựng một hàm đệ quy ta cần chú ý hai điểm quan trọng:

- Xây dựng hệ thức đệ quy R(x) = R(y)
- Xác định điều kiện kết thúc

Factorial (giai thừa)

$$F(n) = 1 \times 2 \times \ldots \times n$$

Dạng thường

```
Long F(int n) {
  Long r = 1;
  for (int i = 2; i <= n; ++i) {
     r *= i;
 }
 return r;
}
```

Dạng đệ quy

```
Long RF(int n) {
  if (n < 0) return 0;
  if (n < 2) return 1;
  return RF(n-1) * n;
}
```

- Hệ thức đệ quy RF(n) = RF(n-1) * n
- Xác định điều kiện kết thúc: RF(0) = RF(1) = 1

Viết gọn

```
Long RF(int n) {
   return (n < 0) ? 0
         : ((n < 2) ? 1 : RF(n-1) * n);
}
```

Test F

```
#include <iostream>

using namespace std;

typedef long long Long;
```

```
void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

Long F(int n) {
  Long r = 1;
  for (int i = 2; i <= n; ++i) {
    r *= i;
 }
 return r;
}

Long RF(int n) {
   return (n < 0) ? 0
          : ((n < 2) ? 1 : RF(n-1) * n);
}

void Test() {
   for (int n = 0; n <= 20; ++n) {
     cout << "\n " << n << " " << F(n) << " : " << RF(n);
        }
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

Result

```
 0 1 : 1
 1 1 : 1
 2 2 : 2
 3 6 : 6
 4 24 : 24
 5 120 : 120
 6 720 : 720
 7 5040 : 5040
 8 40320 : 40320
 9 362880 : 362880
 10 3628800 : 3628800
 11 39916800 : 39916800
 12 479001600 : 479001600
 13 6227020800 : 6227020800
 14 87178291200 : 87178291200
 15 1307674368000 : 1307674368000
 16 20922789888000 : 20922789888000
 17 355687428096000 : 355687428096000
 18 6402373705728000 : 6402373705728000
 19 121645100408832000 : 121645100408832000
 20 2432902008176640000 : 2432902008176640000
 T h e   E n d
```

Dãy Fibonacci

Definition

```
f(0) = f(1) = 1
f(n) = f(n-2) + f(n-1)
f: 1 1 2 3 5 8 13 …
```

Dạng thường

```
Long Fib(int n) {
   if (n < 0) return 0;
  if (n < 2) return 1;
  Long a, b, c;
   a = b = 1;
   c = 2;
   for (int i = 2; i <= n; ++i) {
      c = a + b;
      a = b; b = c;
   }
  return c;
}
```

Dạng đệ quy

```
Long RFib(int n) {
  if (n < 0) return 0;
   if (n < 2) return 1;
   return RFib(n-2) + RFib(n-1);
}
```

Viết gọn

```
Long RFib(int n) {
  return (n < 0) ? 0
         : ((n < 2) ? 1 : RFib(n-2) + RFib(n-1));
}
```

- Hệ thức đệ quy RFib(n) = RFib(n-2) + RFib(n-1)
- Xác định điều kiện kết thúc: RFib(0) = RFib(1) = 1

Test Fib

```
#include <iostream>

using namespace std;

typedef long long Long;

void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
```

```
}

// So fibonacci thu n
// f(0) = f(1) = 1
// f(i) = f(i-2) + f(i-1)
// san duoi
Long Fib(int n) {
   if (n < 0) return 0;
  if (n < 2) return 1;
  Long a, b, c;
   a = b = 1;
   c = 2;
   for (int i = 2; i <= n; ++i) {
      c = a + b;
      a = b; b = c;
   }
  return c;
}

// De quy
Long RFib(int n) {
  return (n < 0) ? 0
         : ((n < 2) ? 1 : RFib(n-2) + RFib(n-1));
}

void Test() {
   for (int n = -10; n <= 20; ++n) {
     cout << "\n " << n << " " << Fib(n) << " : " << RFib(n);
        }
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

Result

```
 -10 0 : 0
 -9 0 : 0
 -8 0 : 0
 -7 0 : 0
 -6 0 : 0
 -5 0 : 0
 -4 0 : 0
 -3 0 : 0
 -2 0 : 0
 -1 0 : 0
 0 1 : 1
 1 1 : 1
 2 2 : 2
 3 3 : 3
 4 5 : 5
 5 8 : 8
 6 13 : 13
 7 21 : 21
```

```
8 34 : 34
9 55 : 55
10 89 : 89
11 144 : 144
12 233 : 233
13 377 : 377
14 610 : 610
15 987 : 987
16 1597 : 1597
17 2584 : 2584
18 4181 : 4181
19 6765 : 6765
20 10946 : 10946
T h e   E n d
```

Độ cao của một số

Phương án thường

```
int H(Long x, int base = 10) {
  int sum = 0;
  while (x) {
     sum += x % base;
     x /= base;
  }
  return sum;
}
```

Phương án đệ quy

```
int RH(Long x, int base = 10) {
   return (x < base) ? x : RH(x / base, base) + (x % base);
}
```

- Hệ thức đệ quy RH(x, base) = RH(x / base) + (x % base)

Độ cao của số x = Độ cao của phần còn lại của x + chữ số đơn vị của x

- Xác định điều kiện kết thúc: x < base

Số chữ số của số nguyên x trong hệ đếm base: Len(x, base), RLen(x, base)

Phương án không đệ quy: Len(x, base)

```
int Len(Long x, int base = 10) {
  int c = 0;
  while (x) {
     ++c;
     x /= base;
  }
  return c;
}
```

Phương án đệ quy: RLen(x, base)

```
int RLen(Long x, int base = 10) {
  return (x < base) ? 1 : RLen(x / base, base) + 1;
}
```

- Hệ thức đệ quy `RLen(x, base) = RLen(x / base) + 1`

  Len của số x = RLen(Phần còn lại của x) + chữ số đơn vị của x
- Xác định điều kiện kết thúc: RLen(x, base) = 1 if x < base.

Chuyển số nguyên sang dạng string trong hệ đếm base RLen(x, base)

Phương án không đệ quy: IntToStr(x, base)

```
const string DIGIT = "0123456789";
const string alpha = "abcdefghijklmnopqrstuvwxyz";
const string EDIGIT = DIGIT + alpha; // Expanded Digit

// -1234 -> "-1234"
string IntToStr(Long x, int base = 10) {
  string s = "";
  if (x == 0) return "0";
         string sign = "";
  if (x < 0) {
     sign = "-";
     x = -x;
 }
  while(x) {
    s = EDIGIT[x % base] + s;
    x /= base;
  }
  return sign + s;
}
```

Phương án đệ quy: RIntToStr(x, base)

```
const string DIGIT = "0123456789";
const string alpha = "abcdefghijklmnopqrstuvwxyz";
const string EDIGIT = DIGIT + alpha; // Expanded Digit

// 1234 -> "1234"
string RUIToStr(Long x, int base = 10) {
  return (x == 0) ? "" : RUIToStr(x / base) + EDIGIT[x % base];
}

// -1234 -> "-1234"
string RIntToStr(Long x, int base = 10) {
  if (x == 0) return "0";
```

```
 return (x < 0) ? "-" + RUIToStr(-x, base)  : RUIToStr(x, base);
}
```

Program

Xác định Len của các số trong hệ đếm 10, hệ đếm 2 và hệ 16.

```
#include <iostream>

using namespace std;

const string DIGIT = "0123456789";
const string alpha = "abcdefghijklmnopqrstuvwxyz";
const string EDIGIT = DIGIT + alpha; // Expanded Digit

typedef long long Long;

void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

int H(Long x, int base = 10) {
  int sum = 0;
  while (x) {
     sum += x % base;
     x /= base;
  }
  return sum;
}

int RH(Long x, int base = 10) {
   return (x < base) ? x : RH(x / base, base) + (x % base);
}

int Len(Long x, int base = 10) {
  int c = 0;
  while (x) {
     ++c;
     x /= base;
  }
  return c;
}

int RLen(Long x, int base = 10) {
  return (x < base) ? 1 : RLen(x / base, base) + 1;
}

// -1234 -> "-1234"
string IntToStr(Long x, int base = 10) {
  string s = "";
  if (x == 0) return "0";
  string sign = "";
  if (x < 0) {
     sign = "-";
     x = -x;
```

```
 }
  while(x) {
    s = EDIGIT[x % base] + s;
    x /= base;
  }
  return sign + s;
}

// 1234 -> "1234"
string RUIToStr(Long x, int base = 10) {
  return (x == 0) ? "" : RUIToStr(x / base) + EDIGIT[x % base];
}

// -1234 -> "-1234"
string RIntToStr(Long x, int base = 10) {
  if (x == 0) return "0";
 return (x < 0) ? "-" + RUIToStr(-x, base)  : RUIToStr(x, base);
}

/*-----------------------------------------
 Len cua cac so a:b trong 3 he dem 10, 2, 16
-----------------------------------------*/
void Test() {
  for (int x = 0; x < 224; ++x) {
    cout << "\n x = " << x;
    cout << "\n Base 10: " << x << "  Len:" << Len(x) << " : " << RLen(x);
    cout << "\n Base 2 : " << IntToStr(x,2) << " Len: " << Len(x,2) << " : "
<< RLen(x,2);
       cout << "\n Base 16: " << IntToStr(x,16) << "  Len: " << Len(x,16)
<< " : " << RLen(x,16);
             Go();
  }
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

Chuyển dạng string sang số nguyên trong hệ đếm base StrToInt(x, base), RStrToInt(x, base)

Phương án không đệ quy: StrToInt(s, base)

```
const string DIGIT = "0123456789";
const string alpha = "abcdefghijklmnopqrstuvwxyz";
const string EDIGIT = DIGIT + alpha; // Expanded Digit

// return value of digit c
int DigitVal(char c) {
  return (c <= '9') ? c-'0'
         : 10+c-'a';
}

Long StrToInt(string & s, int base = 10) {
```

```
   Long v = 0;
   int sign = 1;
   int start = 0;
  if (s[0] == '-') { // "-123"
    start = 1;
         sign = -1;
  }
   for (int i = start; i < s.length(); ++i) {
      v = v * base + DigitVal(s[i]);
   }
  return sign*v;
}
```

Phương án đệ quy: RStrTo(x, base)

```
const string DIGIT = "0123456789";
const string alpha = "abcdefghijklmnopqrstuvwxyz";
const string EDIGIT = DIGIT + alpha; // Expanded Digit

// chuyen doi str s tu vi tri j den het
// sang so nguyen khong dau
// "123" base 10 -> 123
// "9d80" base 16 -> 40320
Long RStrToUInt(string s,Long v,int base,int j) {
  return (j == s.length()) ? v
        : RStrToUInt(s,v*base+DigitVal(s[j]),base,j+1);
}

// "-123" base 10 -> -123
// "9d80" base 16 -> 40320
// "-9d80" base 16 -> -40320
Long RStrToInt(string & s, int base = 10) {
  return (s[0] == '-') ? -1*RStrToUInt(s,0,base,1)
                       : RStrToUInt(s,0,base,0);
}
```

Khi nào không dùng đệ quy ?

Xét hai dãy factorial F(n) trong hệ 16 và Fibonacci Fib(n) trong hệ 10 với n biến thiên trong khoảng 1 đến 9:
Tổng các chữ số của F(n) và Fib(n) là 2+4+9+14+23+19+18+37+40 = 166.

n	1	2	3	4	5	6	7	8	9
F (base = 10)	1	2	6	24	120	720	5040	40320	362880
F (base = 16)	1	2	6	18	78	2d0	13b0	9d80	58980
Fib (base = 10)	1	2	3	5	8	13	21	34	55
digit sum	2	4	9	14	23	19	18	37	40

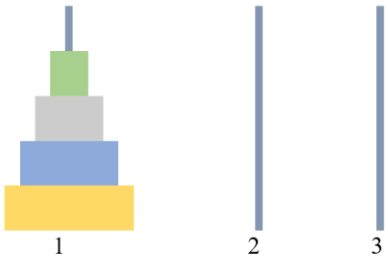
Hãy tính tổng các chữ số của F(n) và Fib(n) với n biến thiên trong khoảng [10:19] ?

Program

```
#include <iostream>

using namespace std;

const string DIGIT = "0123456789";
const string alpha = "abcdefghijklmnopqrstuvwxyz";
const string EDIGIT = DIGIT + alpha; // Expanded Digit

typedef long long Long;

void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

int H(int x, int base = 10) {
   int s = 0;
   while(x) {
      s += x % base;
      x /= base;
   }
   return s;
}

int Sum(int a, int b) {
  Long f = 1, fiba = 0, fibb = 1, fibc = 1;
  // tinh f va fib tu 2:a
 for (int i = 1; i <= a; ++i) {
   f *= i;
   fibc = fiba + fibb;
   fiba = fibb;
   fibb = fibc;
 }

 int s = H(f, 16) + H(fibc);

 // tinh tiep f va fibc tu (a+1):b
 for (int i = a+1; i <= b; ++i) {
   f *= i;
   fibc = fiba + fibb;
   fiba = fibb;
   fibb = fibc;
   s += H(f, 16) + H(fibc);
 }
 return s;
}

main() {
  cout << "\n " << Sum(10,19); // 404
  cout << "\n T h e   E n d";
  return 0;
}
```

Các bài toán Tháp Hà Nội

Tháp Hà Nội cổ



Tháp Hà Nội

Có ba cọc cắm tại ba vị trí là 1, 2 và 3. Trên cọc 1 có một chồng gồm n đĩa bằng gỗ hình tròn to nhỏ khác nhau được xuyên lỗ ở giữa tựa như những đồng xu và đặt chồng lên nhau để tạo ra một toà tháp. Người chơi phải chuyển được toà tháp sang cọc 2 theo các quy tắc sau:

(1) Người chơi được sử dụng 3 cọc để đặt tạm các tầng tháp.

(2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang một trong hai cọc còn lại.

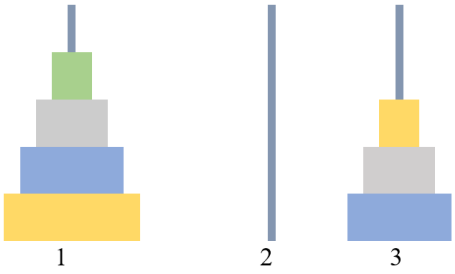
(3) Không bao giờ được đặt tầng tháp lớn lên trên tầng tháp nhỏ.

Hãy tìm cách giải bài toán trên với số lần chuyển ít nhất.

Algorithm

Chắc chắn là bạn đã biết cách giải bài toán trên. Tuy nhiên để có thể giải dễ dàng các biến thể của bài toán tháp Hà Nội chúng ta thử tìm hiểu một cách lập luận sau.

Giả sử ta quan sát một người chơi giỏi, tức là người chuyển được n tầng tháp từ cọc 1 sang cọc 2 với số lần chuyển tối thiểu. Ta dùng một chiếc máy ảnh chụp từng kết quả trung gian sau mỗi bước chuyển của người chơi này. Tổng số tấm ảnh, trừ tấm ảnh ban đầu, chính là số bước chuyển các tầng. Trong số các tấm ảnh chắc chắn phải có một tấm như hình dưới.



Một tấm ảnh giữa quy trình chuyển tháp

Tại sao vậy? Tại vì chừng nào chưa dỡ được *n* - 1 tầng tháp ở phía trên của vị trí 1 để chuyển sang vị trí 3 thì anh ta không thể chuyển được tầng tháp cuối, tức là tầng lớn nhất sang vị trí 2.

Gọi `Hn(n,a,b)` là thủ tục chuyển *n* tầng tháp từ vị trí *a* sang vị trí $b \neq a$, ta thấy:

Nếu *n* = 0: không phải làm gì;

Nếu *n* > 0 ta phải thực hiện ba bước sau:

- Thoạt tiên chuyển *n* – 1 tầng tháp từ vị trí a sang vị trí *c* = *6* – *a* – *b*:

```
Hn(n-1,a,6-a-b)
```

- Sau đó chuyển tầng lớn nhất từ vị trí *a* sang vị trí *b*:

```
a → b
```

- Cuối cùng chuyển *n* – 1 tầng tháp từ *c* sang *b*:

```
Hn(n-1,6-a-b,b)
```

Để ý rằng, do ta mã hoá các cọc là 1, 2 và 3 cho nên biết hai trong ba vị trí đó, là x, y chẳng hạn, ta dễ dàng tính được vị trí còn lại z theo hệ thức

```
z = 6-x-y
```

Ta biểu diễn cấu hình tháp n tầng dưới dạng string "**12…n**" trong đó **1** là tầng trên cùng **n** là tâng dưới cùng để minh họa thủ tục chuyển tháp dưới dạng bảng như sau:

step		cọc 1	cọc 2	cọc 3
init		**123**		
1	1 → 2	**23**	**1**	
2	1 → 3	**3**	**1**	**2**

3	2 → 3	**3**		**12**
4	1 → 2		**3**	**12**
5	3 → 1	**1**	**3**	**2**
6	3 → 2	**1**	**23**	
7	1 → 2		**123**	

Program

```
#include <iostream>

using namespace std;

int step;

void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

void HaNoi(int n, int a, int b) {
  if (n == 0) return;
  int c = 6 - a - b;
  HaNoi(n-1, a, c);
 cout << "\n " << (++step) << ". " << a << " -> " << b;
 HaNoi(n-1, c, b);
}

void Run(int n) {
  step = 0;
  HaNoi(n, 1, 2);
  cout << "\n Total " << step << " step(s.)";
}

main() {
  Run(3);
  cout << "\n T h e   E n d";
  return 0;
}
```

Result

```
 1. 1 -> 2
 2. 1 -> 3
 3. 2 -> 3
 4. 1 -> 2
 5. 3 -> 1
 6. 3 -> 2
 7. 1 -> 2
 Total 7 step(s.)
 T h e   E n d
```

Độ phức tạp tính toán

Gọi T(n) là số bước chuyển tháp n tầng. Ta có

T(n) = (T(n-1) chuyển n-1 tầng từ a sang c)

+ 1 (chuyển tầng cuối cùng a → b)

+ (T(n-1) chuyển n-1 tầng từ c sang b)

Vậy T(n) = 2t(n-1) + 1

Hàm đệ quy RT(n)

- Hệ thức đệ quy RT(n) = 2*RT(n-1) + 1
- Xác định điều kiện kết thúc: RT(1) = 1

```
Long RT(int n) {
  return (n < 1) ? 0 : 2*RT(n-1) + 1;
}
```

Công thức truy hồi T(n)

Ta có

T(1) = 1

T(2) = 2*T(1) + 1 = 2*1 + 1 = 2 + 1

T(3) = 2*T(2) + 1 = 2*(2* + 1) + 1 = 2*2 + 2 + 1

T(4) = 2T(3) + 1 = 2*(2*2 + 2 + 1) + 1 = 2*2*2 + 2*2 + 2 + 1 = $2^3 + 2^2 + 2^1 + 1$

Ta phỏng đoán T(n) = 2^n – 1. Ta cần chứng minh điều này theo quy nạp.

n = 0: T(0) = 2^0 – 1 = 1–1 = 0. Đúng

Gỉa sử công thức đúng với n > 0: T(n) = 2^n – 1.

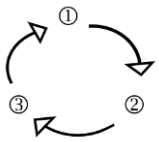
Ta cần chứng minh công thức đúng với n+1 tức là T(n+1) = 2^{n+1} – 1.

Thật vậy,

T(n+1) = 2T(n) + 1 = 2(2^n – 1) + 1 = 2^{n+1} – 2 + 1 = 2^{n+1} – 1.

Tháp Hà Nội vòng

Nội dung giống như bài toán tháp Hà Nội cổ chỉ sửa lại quy tắc (2) như sau: (2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang cọc sát nó theo chiều kim đồng hồ.



Điều kiện này quy định ba phép chuyển 1 tầng tháp giữa các cọc như sau:

Nếu cọc b kề phải cọc a thì ta chuyển ngay: a→b, cụ thể là:

Nếu b = (a % 3) + 1 thì chuyển được a→b

Có các trượng hợp sau: 1→2, 2→3, 3→1

Nếu giữa a và b có cọc c thì ta phải chuyển 2 lần: a→c, c→b, cụ thể là:

Nếu b != (a % 3) + 1 thì phải chuyển 2 lần a→c rồi c→b.

Có các trượng hợp sau: 1→3, 2→1, 3→2.

Minh họa

n = 3

step		cọc 1	cọc 2	cọc 3
init		**123**		
1	1→2	**23**	**1**	
2	2→3	**23**		**1**
3	1→2	**3**	**2**	**1**
4	3→1	**13**	**2**	
5	2→3	**13**		**2**
6	1→2	**3**	**1**	**2**
7	2→3	**3**		**12**
8	1→2		**3**	**12**
9	3→1	**1**	**3**	**2**
10	1→2		**13**	**2**
11	3→1	**2**	**13**	
12	2→3	**2**	**3**	**1**
13	1→2		**23**	**1**
14	3→1	**1**	**23**	
15	1→2		**123**	

Program

```
#include <iostream>

using namespace std;

int step;

void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

void HaNoiC(int n, int a, int b) {
  if (n == 0) return;
  int c = 6 - a - b;
  if (b == (a % 3) + 1) {
    HaNoiC(n - 1, a, c);
```

```
    cout << "\n " << (++step) << ". " << a << " -> " << b;
    HaNoiC(n - 1, c, b);
  }
  else { // a c b, c = 6-a-b
    HaNoiC(n - 1, a, b);
    cout << "\n " << (++step) << ". " << a << " -> " << c;
    HaNoiC(n - 1, b, a);
    cout << "\n " << (++step) << ". " << c << " -> " << b;
    HaNoiC(n - 1, a, b);
  }
}

void Run(int n) {
  step = 0;
  HaNoiC(n, 1, 2);
  cout << "\n Total " << step << " step(s.)";
}

main() {
  Run(3);
  cout << "\n T h e   E n d";
  return 0;
}
```

Result

```
1. 1 -> 2
2. 2 -> 3
3. 1 -> 2
4. 3 -> 1
5. 2 -> 3
6. 1 -> 2
7. 2 -> 3
8. 1 -> 2
9. 3 -> 1
10. 1 -> 2
11. 3 -> 1
12. 2 -> 3
13. 1 -> 2
14. 3 -> 1
15. 1 -> 2
Total 15 step(s.)
T h e   E n d
```

Độ phức tạp tính toán

Tháp Hà Nội ngược

Nội dung giống như bài toán tháp Hà Nội cổ chỉ sửa lại quy tắc (2) như sau: (2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang cọc sát nó về hướng ngược chiều kim đồng hồ.

Tháp Hà Nội thẳng

Nội dung giống như bài toán tháp Hà Nội cổ chỉ sửa lại quy tắc (2) như sau: (2) Mỗi lần chỉ được chuyển 1 tầng tháp từ một cọc sang cọc kề nó, không được vòng từ 3 sang 1 hay 1 sang 3.

1 – 2 – 3

Tháp Hà Nội sắc màu (Hà Nội Cầu vồng)

Người ta sơn mỗi tầng tháp một màu và quy định luật chuyển cho mỗi loại tầng theo màu như sau:

x xanh: xuôi theo chiều kim đồng hồ: 1→2→3→1.

n nâu: ngược chiều kim đồng hồ: 1←2←3←1

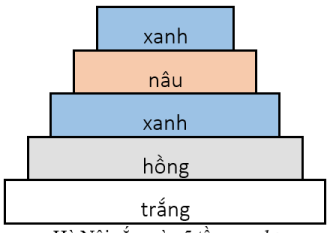
t trắng: chuyển thẳng: 1←2←3

f hồng: chuyển tự do theo hà nội kinh điển: 1↔2↔3↔1

Ngoài ra, dĩ nhiên vẫn phải theo quy định là tầng to không được đặt lên trên tầng nhỏ.

Hãy tìm cách giải bài toán với số lần chuyển ít nhất.

Thí dụ, với các tầng tháp được tô màu từ trên (tầng nhỏ nhất) xuống dưới (tầng lớn nhất) là:



Hà Nội sắc màu 5 tầng *xnxht*

và cần chuyển tháp từ cọc 1 sang cọc 2 thì phải thực hiện tối thiểu 31 lần chuyển các tầng như sau:

Đọc thêm

Lược sử

Một số bài toán về tháp Hà Nội đã được đưa vào các kì thi Olympic Tin học tại một số quốc gia. Chúng ta thử tìm hiểu cội nguồn của các bài toán thuộc loại này.

Năm 1883, trên một tờ báo ở Paris có đăng bài mô tả một trò chơi toán học của giáo sư Claus với tên là Tháp Hà Nội. Nội dung trò chơi được mọi người say mê làm thử chính là bài toán Tháp Hà Nội cổ.

Thời đó ở thủ đô Paris dân chúng đổ xô nhau mua đồ chơi này và suốt ngày ngồi chuyển tháp. Trong lịch sử về các trò chơi thông minh đã từng có những cơn sốt như vậy. Tính trung bình mỗi thế

kỉ có một vài cơn sốt trò chơi. Thế kỉ thứ XX có cơn sốt Rubic, thế kỉ XIX là các trò chơi 15 và tháp Hà Nội. Bài toán này nổi tiếng đến mức trở thành kinh điển trong các giáo trình về thuật giải đệ quy và được trình bày trong các thông báo chính thức của các phiên bản chuẩn của các ngữ trình như ALGOL-60, ALGOL-68, Pascal, Delphy, C, C++, Ada,... khi muốn nhấn mạnh về khả năng đệ quy của các ngôn ngữ đó.

Theo nhà nghiên cứu Henri De Parville công bố vào năm 1884 thì tác giả của trò chơi tháp Hà Nội có tên thật là nhà toán học Eduard Lucas, người có nhiều đóng góp trong lĩnh vực số luận. Mỗi khi viết về đề tài giải trí thì ông đổi tên là Claus. Bạn có để ý rằng Claus là một hoán vị các chữ cái của từ Lucas.

De Parville còn kể rằng bài toán tháp Hà Nội bắt nguồn từ một tích truyền kì ở Ấn Độ. Một nhóm cao tăng Ấn Độ giáo được giao trọng trách chuyển dần 64 đĩa vàng giữa ba cọc kim cương theo các điều kiện đã nói ở bài toán Tháp Hà Nội cổ. Khi nào hoàn tất công việc, tức là khi chuyển xong toà tháp vàng 64 tầng từ vị trí ban đầu sang vị trí kết thúc thì cũng là thời điểm tận thế. Sự việc này có xảy ra hay không ta sẽ xét ở bài tập 8.10.

Lời giải được công bố đầu tiên cho bài toán tháp Hà Nội là của Allardice và Frase, năm 1884.

Năm 1994 David G. Poole ở Đại học Trent, Canada đã viết một bài khảo cứu cho tờ Mathematics Magazine số tháng 12 nhan đề "Về các tháp và các tam giác của giáo sư Claus" cùng với một phụ đề "Pascal biết Hà Nội". Poole đã liệt kê 65 công trình khảo cứu bài toán tháp Hà Nội đăng trên các tạp chí toán-tin trong khoảng mười năm. Tác giả cũng chỉ ra sự liên quan giữa các công thức tính số lần chuyển các tầng tháp và một phương pháp quen biết dùng để tính các hệ số của dạng khai triển nhị thức Newton $(a + b)^n$. Phương pháp này được gọi là Tam giác Pascal, mang tên nhà toán học kiêm vật lí học Pháp Blaise Pascal (1623-1662), người đã chế tạo chiếc máy tính quay tay đầu tiên trên thế giới.

Một số nhà nghiên cứu trong và ngoài nước có bàn luận về địa danh Hà Nội. Theo tôi vấn đề này vẫn còn ngỏ. Hầu hết các bài viết xoay quanh đề tài chuyển tháp nói trên đều dùng thuật ngữ bài toán tháp Hà Nội. Khi giới thiệu về bài toán Hà Nội nhiều tháp Dudeney đặt tên là bài toán đố của Reve (The Reve's Puzzle). Tuy nhiên, nhiều nhà nghiên cứu cho rằng tốt hơn cả là nên đặt tên và phân loại theo tên nguyên thuỷ của bài toán, nghĩa là Tháp Hà Nội.

Ngoài các dạng Tháp Hà Nội đã liệt kê ở phần trên một số tác giả còn đề xuất những dạng khá kì lạ, chẳng hạn như bài toán sau đây.

Hà Nội nhiều tháp

Trong trò chơi này người ta làm thêm những cọc, chẳng hạn thay vì ba ta dùng bốn cọc và cũng có thể bố trí tháp tại nhiều cọc. Ý kiến này do H.E. Dudeney, một tác giả hàng đầu về toán học giải trí người Anh đưa ra vào năm 1908. Đã có nhiều bài đăng lời giải cho bài toán này, có những bài mới xuất hiện gần đây vào những năm 1988 và 1989. Dù vậy chưa ai chứng minh được rõ ràng số lần chuyển của bài giải là tối thiểu như đã làm với các dạng tháp Hà Nội khác.

Bài tập

Bạn hãy thử lập công thức tính số lần chuyển các tầng tối thiểu cho các bài toán sau:

Tháp Hà Nội,
Tháp Hà Nội Xuôi,
Tháp Hà Nội Ngược *và*
Tháp Hà Nội Thẳng.

Lời cảm ơn Các tư liệu trên và một số tư liệu khác trong bài được trích dẫn từ các bài viết của giáo sư viện sĩ Nguyễn Xuân Vinh, Khoa Kỹ thuật không gian, Đại học Michigan, cộng tác viên NASA, Hoa Kỳ. Tác giả xin chân thành cám ơn giáo sư đã cho phép trích dẫn và chỉ giáo về các phương pháp truyền thụ tri thức khoa học cho giới trẻ.

Tổ hợp chặp k của n phần tử: C(n,k)

Formula

$$C(n,k) = \binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1}$$

0! = 1.

Ta có

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \frac{n!}{(k-1)!\,(n-k+1)!} \times \frac{n-k+1}{k} = \binom{n}{k-1} \times \frac{n-k+1}{k}$$

Nếu sử dụng cách ghi tuyến tính *C*(*n*, *k*) cho hàm tính tổ hợp chập *k* của *n*, ta có:

$$C(n, k) = C(n, k-1)(n-k+1) \text{ div } k \quad (1)$$

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

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

Một phương án đệ quy cho hàm *C*(*n*, *k*) là:

C(n, k) = (k == 1) ? n : C(n, k – 1) * (n – k + 1) div k.

Thay *k* trong (1) bằng *i* + 1 ta thu được

$$C(n, i+1) = C(n, i)(n-i) \text{ div } (i+1) \quad (2)$$

Trong dạng không đệ quy, vận dụng (2), hàm *C*(*n*, *k*) sẽ được tính như sau:

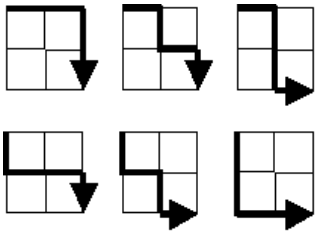
```
// non recursive of C(n, k), n > 0, k > 0
Long C(int n, int k) {
  Long r = 1;
  for (int i = 0; i < k; ++i) {
    r = r * (n-i) / (i+1);
  }
  return r;
}

// Recursive of C(n, k), n > 0, k > 0
Long RC(int n, int k) {
  return (k == 1)? n : RC(n,k-1)*(n-k+1)/k;
}
```

Commented [WU1]: Việt hóa dòng này

Commented [WU2]: Việt hóa dòng này

Problem 15, Project Euler

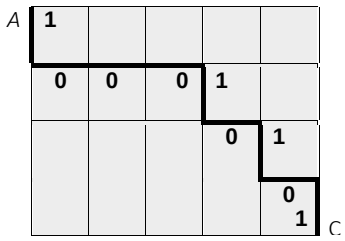
Xuất phát từ góc trên trái của lưới kích thước 2×2 có 6 đường (không quay lui) dẫn đến góc dưới phải như sau:



Có bao nhiêu đường như vậy trong lưới 20×20?

Algorithm

Ta gắng giải bài này với lưới tổng quát có kích thước *a*×*b*, trong đó *a* là chiều rộng, *b* là chiều dài của lưới. Trước hết ta gọi một đường không quay lui từ *A* đến *C* là *đường hợp lệ*. Trong mỗi đường, ta gán nhãn 0 cho cạnh đơn vị ngang và 1 cho cạnh dọc.



Đường 100010101 trong lưới 4×5. a = 4, b = 5.

Ta thấy mọi đường hợp lệ đều đi qua đúng *a* cạnh dọc và đúng *b* cạnh ngang. Từ đây suy ra mọi đường hợp lệ đều có chiều dài *a+b*. Vậy bài toán có thể được phát biểu lại như sau:

Có bao nhiêu xâu nhị phân dài a+b chứa đúng a số 1 (hoặc *b* số 0).

Đáp số khi đó sẽ là $\binom{n}{k}$ - *tổ hợp chập k của n* với công thức tính

Program

```
#include <iostream>

using namespace std;

#include <iostream>

using namespace std;

typedef long long Long;

// non recursive of C(n, k)
Long C(int n, int k) {
  Long r = 1;
  for (int i = 0; i < k; ++i) {
    r = r * (n-i) / (i+1);
  }
  return r;
}

// Recursive of C(n, k)
Long RC(int n, int k) {
  return (k == 1)? n : RC(n,k-1)*(n-k+1)/k;
}

main() {
```

```
  Long n = 40, k = 20;
  cout << C(n, k) << " " << RC(n,k); // 137846528820
  cout << "\n T h e   E n d";
  return 0;
}
```

Ví dụ

Trong các số nhị phân từ 1 đến 2^9 có bao nhiêu số chứa đúng 5 bit 1 ?

Đáp số

```
C(9, 4) or RC(9, 4)
```

Problem 2. Int To Rome

Chuyển đổi số nguyên dương sang dạng số La Mã.

```
IntToRome(2021) = "MMXXI"
IntToRome(3964) = "MMMCMLXIV"
```

Algorithm Rót nước

Program

```
/**********************************
 Name: INITOROME.CPP
 Copyright: (C) 2021
 Author: DevcFan
 Date: 11/10/21 10:06
 Description:
 int to Rome
 IntToRome(2021) = "MMXXI"
  IntToRome(3964) = "MMMCMLXIV"
**********************************/

#include <iostream>
// #include <fstream>
//#include <bitset>
//#include <cmath>
//#include <windows.h>
//#include <cstring>
#include <bits/stdc++.h>
//#include <time.h>
//#include <algorithm>

using namespace std;

const int MN = 13;
// Roles 49 AB = B-A; AB = A+B
int ival[] =    {1000,900,500,400,100,90,   50,  40,  10,  9,  5,   4,1};
string rval[] = {"M","CM","D","CD","C","XC","L",
"XL","X","IX","V","IV","I"};

void Go() {
  cout << " ? ";
  fflush(stdin);
  if (cin.get() == '.') exit(0);
```

```
}

string IntToRome(int b) {
  string r = "";
  for (int i = 0; i < MN; ++i) {
     while (b >= ival[i]) {
         b -= ival[i];
         r += rval[i];
     }
  }
  return r;
}

main() {
  int n = 2021;
  cout << "\n " << n << " -> " <<  IntToRome(n); // "MMXXI"
  n = 3964;
  cout << "\n " << n << " -> " << IntToRome(3964); // "MMMCMLXIV"
  for (int i = 1; i <= 4000; ++i) {
     cout << "\n " << i << " -> " << IntToRome(i); Go();
 }
 // --------------------------------
    cout << endl << "\n\n T h e   E n d \n";
    return 0;
}
```

Result

```
2021 -> MMXXI
3963 -> MMMCMLXIV
1 -> I ?

2 -> II ?

3 -> III ?

4 -> IV ?

5 -> V ?

6 -> VI ?

7 -> VII ?
```

Divide And Conquer

Idea

Cần tính F(D)

- Chia miền data cần xử lý thành các miền nhỏ: D = D1 | D2 | … | Dk
- Xử lý từng miền nhỏ F(Di) 1 ≤ i ≤ k
- Tổng hợp kết quả

k = ?

Chia đến khi nhận được kết quả tầm thường (hiển nhiên)

Problem 1. Balancing

Trong số 27 đồng tiền vàng hình dạng giống nhau, có 26 đồng nặng bằng nhau, 1 đồng nhẹ hơn các đồng khác. Cần cân mấy lần trên loại cân thăng bằng 2 đĩa để phát hiện ra đồng tiền nhẹ ?

	Đĩa A	Đĩa B	?	Trên bàn C
1	9	9	A = B	C = 9
			A < B	A = 9
			else	B = 9
2	3	3		3
3	1	1		1

54 ?

27000 ?

Chia 3: $\log_3(n)$; Chia 2: $\log(n)$; Chia k: $\log_k(n)$

Problem 2. Min, Max of Array

Xác định giá trị nhỏ nhất trong mảng nguyên a[d:c].

Method: Chia đôi, xử lý nửa trái, nửa phải. Tổng hợp kết quả.

```
MinVal(a, d, c) = min {MinVal(a,d,m), MinVal(a,m+1,c)}
m = (d+c) / 2; // điểm giữa
```

Function

```
int MinVal(int a[], int d, int c) {
  if (d == c) return a[d];
 int m = (d+c) / 2;
 int m1 = MinVal(a, d, m);
 int m2 = MinVal(a, m+1, c);
 return min(m1, m2);
}
```

Reducing (viết gọn)

```
int MinVal(int a[], int d, int c) {
 int m = (d+c) / 2;
 return (d == c) ? a[d]
        : min(MinVal(a, d, m), MinVal(a, m+1, c));
}
```

Tìm index i trong mảng nguyên a[d:c] so cho a[i] đạt min.

Function

```
int Min(int a[], int d, int c) {
  if (d == c) return d;
 int m = (d+c) / 2;
 int i1 = Min(a, d, m);
 int i2 = Min(a, m+1, c);
 return (a[i1] < a[i2]) ? i1 : i2;
}
```

Test

```
#include <iostream>
// #include <windows.h>
// #include <algorithm>

using namespace std;

int a[] = {5, 2, -1, 9, 6, -7, 12, 19, -14, 40};

void Go() {
   cout << " ? ";
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

void Print(int a[], int d, int c, const char * msg = "") {
  cout << msg;
  for (int i = d; i <= c; ++i)
     cout << " " << a[i];
}

int MinVal(int a[], int d, int c) {
 int m = (d+c) / 2;
 return (d == c) ? a[d]
        : min(MinVal(a, d, m), MinVal(a, m+1, c));
}


int Min(int a[], int d, int c) {
  if (d == c) return d;
 int m = (d+c) / 2;
 int i1 = Min(a, d, m);
 int i2 = Min(a, m+1, c);
 return (a[i1] < a[i2]) ? i1 : i2;
}
```

```
void Test() {
   int d = 0;
   int n = 10;
   for (int c = 0; c < n; ++c) {
     Print(a, d, c, "\n a: ");
     int id = Min(a, d, c);
     int m = MinVal(a, d, c);
              cout << ", Min Val = " << m << " at " << id;
              if (a[id] == m) cout << " CORRECT.";
              else cout << " ERROR.";
  }
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

result

```
 a:  5, Min Val = 5 at 0 CORRECT.
 a:  5 2, Min Val = 2 at 1 CORRECT.
 a:  5 2 -1, Min Val = -1 at 2 CORRECT.
 a:  5 2 -1 9, Min Val = -1 at 2 CORRECT.
 a:  5 2 -1 9 6, Min Val = -1 at 2 CORRECT.
 a:  5 2 -1 9 6 -7, Min Val = -7 at 5 CORRECT.
 a:  5 2 -1 9 6 -7 12, Min Val = -7 at 5 CORRECT.
 a:  5 2 -1 9 6 -7 12 19, Min Val = -7 at 5 CORRECT.
 a:  5 2 -1 9 6 -7 12 19 -14, Min Val = -14 at 8 CORRECT.
 a:  5 2 -1 9 6 -7 12 19 -14 40, Min Val = -14 at 8 CORRECT.
 T h e   E n d
```

Problem 3. Secret Number

Bạn nghĩ một số x trong khoảng a:b. Với ít nhất các câu hỏi Yes/No chương trình tìm ra được số x.

Số câu hỏi = log(b-a+1).

Ví dụ, chỉ cần tối đa 10 câu hỏi có thể đóan được số tùy ý trong khoảng 1 đến 1000.

Method: Chia đôi, xử lý nửa trái, nửa phải. Tổng hợp kết quả.

```
if (x > m) chọn nửa phải [m+1:c]
    else chọn nửa trái [d:m]
```

```
#include <iostream>

using namespace std;

void Go() {
   cout << " ? ";
   fflush(stdin);
```

```
   if (cin.get() == '.') exit(0);
}

int SecretNum(int a, int b) {
    int m; // middle
    int q; // question
    int ch;
    while(a < b) {
       m = (a+b) / 2;
       ++q;
       do {
         cout << "\n " << q << ". Your number > "
                          << m << " ? [Y/N]: ";
         fflush(stdin);
         ch = toupper(cin.get());
        } while (ch != 'Y' && ch != 'N');
       if (ch == 'Y') a = m+1; else b = m;
    }
    cout << "\n Your number is " << b;
    return a;
}

void Test() {
  SecretNum(1, 500);
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

Problem 4. Birthday

Với không quá 9 câu hỏi Yes / No có thể đoán được sinh nhật của một người.

```
#include <iostream>
using namespace std;

int SecretNum(int a, int b, string ask = " Your number > ") {
    int m; // middle
    int q; // question
    int ch;
    while(a < b) {
       m = (a+b) / 2;
       ++q;
       do {
         cout << "\n " << q << ask << m << " ? [Y/N]: ";
         fflush(stdin);
         ch = toupper(cin.get());
       } while(ch != 'Y' && ch != 'N');
      if (ch == 'Y') a = m+1; else b = m;
    }
    cout << "\n I've find " << a;
    return a;
}
```

```
void Birthday() {
   int month = SecretNum(1,12, " * Your born month > ");
   int day = SecretNum(1,31, " ** Your born day > ");
   cout << "\n Your Birthday is " << day << " / " << month;
}

void Test() {
  //SecretNum(1, 500);
  Birthday();
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

Binary Searching

Algorithm

Mảng được sắp tăng (hoặc giảm)

Program

```
#include <iostream>
#include <algorithm>

using namespace std;

int a[] = {5, 2, -1, 9, 6, -7, 12, 19, -14, 40};

void Go(char * msg = " ? ") {
   cout << msg;
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

void Print(int a[], int d, int c, const char * msg = "") {
  cout << msg;
  for (int i = d; i <= c; ++i)
     cout << " " << a[i];
}

int BinSearch(int a[], int d, int c, int x) {
  int m;
   while(d < c) {
      m = (d + c) / 2;
      if (x > a[m]) d = m + 1;
      else c = m;
   }
  return (a[d] == x) ? d : -1;
}

void Test() {
```

```
  int n = 10;
  sort(a, a+n);
  int x;
  while(true) {
     Print(a, 0, n-1,"\n a: ");
     cout << " Search x = ";
         fflush(stdin);
         cin >> x;
         cout << " Found at: " <<  BinSearch(a, 0, n-1, x);
     Go("\n Press dot key [.] to stop \n or any key to continue: ");
  }
}

main() {
  Test();
  cout << "\n T h e   E n d";
  return 0;
}
```

Quick Sort

Method:

```
1. Lấy giá trị m tại điểm giữa mảng.
2. Chuyển các a[i] < m về nửa trái, a[j] > m về nửa phải.
3. Sắp tăng nửa trái
4. Sắp tăng nửa phải
```

Program

```
#include <iostream>
#include <algorithm>
#include <time.h>

using namespace std;

const int MN = 300000;

int a[MN] = {5, 2, -1, 9, 6, -7, 12, 19, -14, 40};

void Go(char * msg = " ? ") {
   cout << msg;
   fflush(stdin);
   if (cin.get() == '.') exit(0);
}

void Print(int a[], int d, int c, const char * msg = "") {
  cout << msg;
  for (int i = d; i <= c; ++i)
     cout << " " << a[i];
}

void Swap(int &a, int &b) {
 int t = a; a = b; b = t;
}
```

```
// O(n^2)
void MinSort(int a[], int d, int c) {
   for (int t = d; t <= c; ++t) {
      for (int p = t + 1; p <= c; ++p) {
         if (a[p] < a[t]) Swap(a[p], a[t]);
      }
   }
}

// O(n^2)
void BubbleSort(int a[], int d, int c) {
   for (int t = d; t <= c; ++t) {
      for (int p = c; p > t; --p) {
         if (a[p] < a[p-1]) Swap(a[p], a[p-1]);
      }
   }
}

// O(nlog(n))
void QuickSort(int a[], int d, int c) {
  int t = d, p = c;
 int m = a[(t+p) / 2];
 while (t <= p) {
    while(a[t] < m) ++t;
    while(a[p] > m) --p;
    if (t <= p) {
       Swap(a[t], a[p]);
       ++t; --p;
    }
 }
 if (d < p) QuickSort(a, d, p);
  if (t < c) QuickSort(a, t, c);
}

void Test1() {
  int n = 10;
  Print(a, 0, n-1,"\n Given a: ");
  // MinSort(a, 0, n-1);
  // BubbleSort(a, 0, n-1);
  QuickSort(a, 0, n-1);
  Print(a, 0, n-1,"\n Sorted a: ");
}


void Gen(int a[], int n) {
   srand(time(NULL));
   for (int i = 0; i < n; ++i)
     a[i] = rand() % 10000;
}

void Test2() {
  int n = 30000;
  Gen(a, n);
  // Print(a, 0, 200, "\n Init: ");
  int t = time(NULL);
  // MinSort(a, 0, n-1);
  // BubbleSort(a, 0, n-1);
  QuickSort(a, 0, n-1);
```

```
  cout << "\n End of QuickSort, time = " << difftime(time(NULL),t);
  // cout << "\n End of MinSort, time = " << difftime(time(NULL),t);
  //cout << "\n End of BubbleSort, time = " << difftime(time(NULL),t);
}

main() {
  // Test1();
  Test2();
  cout << "\n T h e   E n d";
  return 0;
}
```

Độ phức tạp tính toán

MinSort and BubbleSort: O(n2); QuickSort: O(nlog(n)).

nxhuy564@gmail.com