Nguyễn Xuân Huy

**Divide and Conquer Method**

**Các chuyên đề chọn lọc**

Hà Nội - 2023

# Nguyện vọng của giảng viên

*Các bạn học viên thân mến,*

*Rất mong nhận được phản hôi của các bạn về các vấn đề sau:*

*1. Hiểu bài / không hiểu bài*

*2. Nội dung: dễ / khó*

*3. Tốc độ: nhanh / chậm*

*4. Chất lượng: hay / dở*

*5. Các góp ý khác.*

*Các bạn có thể gửi qua*

*Zalo: 0903203800*

*mail: nxhuy564@gmail.com*

*hoạc phone trực tiếp 0903203800*

*Cám ơn các bạn*

*Nguyễn Xuân Huy*

# Phương pháp chia để trị

Chia để trị là phương pháp thiết kế thuật toán dựa trên ý tưởng chia miền dữ liệu cần xử lý thành nhiều miền nhỏ, xử lý từng miền sau đó tổng hợp kết quả.

Một câu hỏi quan trọng liên quan đến phương thức chia là: *Chia miền dữ liệu thành bao nhiêu phần ?*

Tuỳ theo nội dung của bài toán mà có các phương thức chia khác nhau. Dưới đây là một số ý tưởng chỉ đạo cho cho các phương thức chia:

* Nên chia thành các miền dữ liệu không giao nhau để khi tổng hợp được dễ hơn.
* Chia dữ liệu đủ nhỏ để bài toán vận dụng cho dữ liệu đó trở thành tầm thường, tức là trở thành tối giản.

### Gold rings

Trong xí nghiệp sản xuất đồ trang sức người ta cần kiểm định n = 3k, k ≥ 1 chiếc nhẫn vàng hình thức giống nhau, nhưng trong số đó có một chiếc nhẫn bị lỗi nên có trọng lượng nhỏ thua các nhẫn còn lại. Bạn được dùng một chiếc cân thăng bằng 2 đĩa với số lần cân ít nhất để phát hiện ra chiếc nhẫn bị lỗi. Giới hạn của k là 10, khi đó giới hạn tối đa của n sẽ là 310 = 59049.

Chương trình của bạn cần thực hiện các thủ tục và hàm sau đây:

1. Đặt chỉ thị sau đây vào đầu chương trình để liên lạc được với BGK:

#include "BGK.H"

2. Gọi hàm int n = Gen() để ban giám khảo (BGK) sinh dữ liệu gồm n = 3k chiếc nhẫn, trong số đó có một chiếc nhẫn bị lỗi mà bạn không dược phép biết. Các chiếc nhẫn được đánh số từ 1..n = 3k, 1 ≤ k ≤ 10.

3. Mỗi lần bạn được phép cân hai nhóm nhẫn a và b thể hiện dưới dạng mảng chứa các số hiệu của nhẫn. Cụ thể là bạn cần gọi hàm Balance(int a[],int na, int b[], int nb) trong đó na và nb lần lượt là số nhẫn trong nhóm a và nhóm b. BGK sẽ thông báo cho bạn biết kết quả của phép cân Balance là 0 nếu trọng lượng của nhóm nhẫn a bằng với trọng lượng của nhóm nhẫn b; là -1 nếu nếu trọng lượng của nhóm nhẫn a nhỏ thua trọng lượng của nhóm nhẫn b; là 1 nếu nếu trọng lượng của nhóm nhẫn a lớn hơn trọng lượng của nhóm nhẫn b. Bạn được phép gọi hàm Balance không quá n/2 lần. Nhẫn được đánh số từ 1 đến n, chỉ số mảng được tính từ 0.

4. Nếu muốn, sau khi chương trình của bạn đã tìm ra số hiệu của chiếc nhẫn lỗi, chẳng hạn là s, bạn có thể gọi hàm Ask(s) để nhận được câu trả lời của BGK là 1 (kết quả đúng) hoặc 0 (kết quả sai).

#### Thuật toán

Với n = 3k chiếc nhẫn thì sau k lần cân ta sẽ phát hiện ra chiếc nhẫn bị lỗi.

Trước hết để ý rằng nếu có 3 nhóm nhẫn a, b và c với cùng số lượng, thì ta đặt hai nhóm a và b lên hai đĩa cân sẽ phát hiện ra nhóm nào chứa chiếc nhẫn bị lỗi. Cụ thể là

v = Balance(a,na,b,nb);

Nếu v = 0 thì nhóm c chứa nhẫn lỗi;

Nếu v = -1 thì nhóm a nhẹ hơn nhóm b, do đó nhóm a chứa nhẫn lỗi;

Nếu v = 1 thì nhóm a nặng hơn nhóm b, do đó nhóm b chứa nhẫn lỗi;

Vậy ta sẽ chia số nhẫn làm ba phần bằng nhau. Đến lần cân thứ k thì mỗi nhóm chỉ còn một chiếc nhẫn nên ta phát hiện ra nhẫn lỗi.

#### Chương trình

Trong thi đấu, dĩ nhiên BGK sẽ cung cấp các hàm Gen, Balance, Ask và giữ bí mật số hiệu của chiếc nhẫn lỗi. Bạn không được phép tự viết các hàm này, hoặc tìm cách truy nhập đến số hiệu của chiếc nhẫn lỗi.

Để thiết kế chương trình, bạn cần đóng vai trò BGK. Cụ thể là bạn cần viết trước các hàm Gen, Balance và Ask như sau:

// Chuong trinh cua BGK

#ifndef \_\_BGK\_\_H

#define \_\_BGK\_\_H

#include <bits/stdc++.h>

using namespace std;

class \_\_RINGS {

protected:

int failedRing;

int \_\_Bcall;

int \_\_N2;

public:

// sinh ngau nhien du lieu

int \_\_Gen(bool BGK = 0) {

srand(time(NULL));

int k = rand() % 10 + 1;

int n = int(pow(3,k)); // n = 3^k

\_\_N2 = n / 2;

\_\_Bcall = 0;

failedRing = rand() % n + 1; // chon ngau nhien 1 nhan loi

if (BGK) cout << "\n " << failedRing;

return n;

}

// can 2 nhom a va b

int \_\_Balance(int a[], int na, int b[], int nb) {

\_\_Bcall++;

if (\_\_Bcall > \_\_N2) {

cerr << "\n Qua so lan can.";

return 0;

}

int t = 0;

// trong luong dia can trai

for(int i = 0; i < na; ++i)

t += (a[i] == failedRing) ? 1 : 2;

int p = 0;

// trong luong dia can phai

for(int i = 0; i < nb; ++i)

p += (b[i] == failedRing) ? 1 : 2;

return (t == p) ? 0

: ((t < p) ? -1 : 1);

}

bool \_\_Ask(int result) {

cout << "\n So lan can " << \_\_Bcall << endl;

return (result == failedRing);

}

};

\_\_RINGS \_\_R;

#define Gen \_\_R.\_\_Gen

#define Balance \_\_R.\_\_Balance

#define Ask \_\_R.\_\_Ask

#endif

#### Giải thích

* Bạn ghi chương trình này với tên BGK.h
* Hàm Gen() sinh tự động giá trị ngẫu nhiên n = 3k và chiếc nhẫn lỗi failedRing. Chiếc nhẫn lỗi này được khai báo với chỉ thị protected, do đó mọi truy nhập nằm ở ngoài file này đều bị cấm.

Ví dụ, lời gọi Gen() sẽ sinh ra các giá trị n = 81 và failedRing = 47, trong đó giá trị n sẽ được cung cấp cho bạn, còn giá trị failedRing sẽ được giữ riêng.

* Tuy nhiên, muốn giúp bạn niềm tin rằng chương trình của BGK sẽ sinh dữ liệu chuẩn mực, ta cài thêm tham số mặc định BGK cho hàm Gen. Bạn thử gọi Gen(1) để thấy chương trình của BGK sẽ hiển thị số hiệu của chiếc nhẫn lỗi.
* Các chỉ thị #define x y sẽ chuyển đổi các bí danh x thành y. Ví dụ, trong chương trình của ban, bạn viết Gen() chương trình dịch sẽ chuyển thành lời gọi hàm \_\_R.\_\_Gen(). Hàm này sẽ được thực thi trong đói tượng \_\_R thuộc lớp \_\_RINGS.
* Sau lời gọi n = Gen() bạn đặt các số hiệu nhẫn từ 1..n vào mảng r rồi chia r thành ba đoạn dài bằng nhau, ghi chỉ số các nhẫn vào a, b và c, tạm ký hiệu là: r = a|b|c. Tiếp đén là gọi hàm Balance(a, n3, b, n3), n3 = n/3 để BGK xác định kết quả cân...

#### Chương trình

// Rings

#include <bits/stdc++.h>

#include "BGK.H"

using namespace std;

const int MN = 10; // 3^10 = 59049

void Go() {

cout << " ? "; fflush(stdin);

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

}

void Print(int x[], int d, int c, const char \* msg="") {

cout << msg;

for(int i = d; i <= c; ++i) cout << " " << x[i];

}

// Tim so hieu cua chiec nhan loi

int Find(int r[], int n) {

int n3 = n/3;

int a[n3], b[n3], c[n3];

// chia 3 doan dai bang nhau a, b, c

for (int i = 0; i < n3; ++i) {

a[i] = r[i];

b[i] = r[i+n3];

c[i] = r[i+n3+n3];

}

int v = Balance(a, n3, b, n3); // can a va b

int res;

if (n3 == 1) {

res = (v == 0) ? c[0]

: ((v < 0) ? a[0] : b[0]);

return res;

}

// chon nhom a, b, c ?

if (v == 0) return Find(c, n3); // a = b: chon c

else if (v < 0) return Find(a,n3); // a < b: chon a

else return Find(b, n3); // a > b: chon b

}

void Run() {

int n = Gen();

int \*r = new int[n];

for(int i = 0; i < n; ++i) r[i] = i+1;

int res = Find(r, n);

cout << "\n Result = " << res << " ? " << Ask(res);

}

main() {

Run();

cout << "\n T h e E n d";

return 0;

}

#### Output

Result = 2799 ? 1

T h e E n d

## Luyện tập Phương pháp chia để trị

### Nhân Ấn ĐỘ

Người Ấn Độ từ xa xưa đã biết cách nhân nhanh hai số tự nhiên x và y để thu được tích z, z = x × y như sau. Ta biết tích x × y chính là tổng của x số hạng y, z = x × y = y + y +...+y. Giả sử bạn phải làm phép nhân z = 33×25, tức là bạn phải múc 33 gáo nước, mỗi gáo 25 lít đổ vào thùng z. Bạn có thể làm nhanh theo cách sau. Bạn thay gáo 25 lit bằng gáo to gấp đôi, gáo 50 lít thì bạn sẽ chỉ phải múc 33/2 lần. Tuy nhiên 33 là số lẻ, nên để tránh sai số của phép chia đôi, bạn múc trước một gáo 25 lít vào thùng z sau đó bạn chỉ còn phải múc 32 gáo loại 25 lít. Thay bằng gáo gấp đôi là 50 lít thì bạn chỉ phải múc 16 lần. Vị chi là từ 33 lần múc bạn chỉ còn phải múc 16 lần gáo 50 lít + 1 lần gáo 25 lít. Nếu thay gáo 50 lít bằng gáo 100 lít thì bạn chỉ phải múc 8 lần, gáo 200 lít còn 4 lần, và gáo 800...

|  |  |  |  |
| --- | --- | --- | --- |
|  | x | y | z |
| lẻ | 33 | 25 | 25 |
|  | 16 | 50 |  |
|  | 8 | 100 |  |
|  | 4 | 200 |  |
|  | 2 | 400 |  |
| lẻ | 1 | 800 | 800 |
|  |  | + | 825 |

Thuật toán đó như sau:

Để tìm tích z = xy ta viết hai cột x và y

Lần lượt chia đôi x đến khi gặp 1

Nhân đôi y tương ứng với các dòng x

Đánh dấu các dòng x lẻ

Lấy tổng các dòng y ứng với x lẻ làm kết quả

Thuật toán này dựa trên hệ thức

Hãy cài đặt hàm Mult(int x, int y) tính tích của hai số tự nhiên x và y theo thuật toán Ấn Độ.

### Chia Ấn Độ

Người Ấn Độ làm phép chia x:y để tìm thương q và dư r như sau:

Ý tưởng: dùng thước y để đo tấm vải x sẽ được thương q là số lần đặt thước y trên x, và dư r là phần vải thừa.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| vải x: | ---- | ---- | ... | ---- | --- |
| thước y: | y | y | ... | y | r |
|  | q lần đặt thước y | | | | |
|  | x = qy + r | | | | |

Muốn đo nhanh ta gấp đôi thước y. Thuật toán sẽ gồm hai pha như sau

// x : y = thương q, dư r

DivMod(x, y, q, r):

Pha 1:

thước mới w = y

Gấp đôi thước mới w đến khi w > x

while w ≤ x do w = w\*2 end while

Pha 2:

Ta có: vì thước w > tấm vải x nên

q = 0, r = x

Giảm dần (chia 2) thước mới w đến y

while w > y do

w /= 2 // chia đôi thước w

q \*= 2 // nhân đôi số lần đặt thước

if w ≤ r then

q += 1

r -= w

end if

end while

end DivMod

Minh hoạ phép chia (x = 90) : (y = 16) → (thương q = 5, dư r = 10).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | w = y = 16 |  | q |  | w < r ? | r = r-w |
| ↓ | 16 |  | 4+1 = 5 | ↑ | 16 < 26 | 26-16 = 10 |
| ↓ | 32 |  | 2 | ↑ | 32 > 26 |  |
| ↓ | 64 |  | 0+1 = 1 | ↑ | 64 < 90 | 90-64 = 26 |
| ↓ | 128 > 90 |  | 0 | ↑ | 128 | 90 |
|  |  |  | q |  |  | r |

Hãy viết hàm DivMod(int x, int y, int &q, int &r) thực hiện phép chia hai số tự nhiên x:y theo phương pháp Ấn Độ để thu được thương q và dư r.

### Các lũy thừa tự thân (Self powers)

*Problem 48 Project Euler*

Tổng 11 + 22 + 33 + ... + 1010 = 10405071317.

Tìm mười chữ số cuối của tổng 11 + 22 + 33 + ... + 10001000.

## Bài giải Phương pháp chia để trị

### Nhân Ấn Độ

#### Thuật toán

Các số nguyên trong máy tính được biểu diễn dưới dạng nhị phân nên ta có:

x >> k tương đương với x div 2k

x << k tương đương với x\*2k

x & 1 tương ddương với x % 2

int Mult(int x, int y) {

int z = 0; // tich

while(x > 0) {

if(x & 1) z += y; // x le: cong them y

x >>= 1; // giam x 2 lan: x = x/2

y <<= 1; // gap doi y: y = y\*2

}

return z;

}

Theo phương án đệ quy ta có

int RMult(int x, int y) {

return (x == 0 || y == 0) ? 0

: ((x & 1) ? RMult(x/2, y+y) + y

: RMult(x/2, y+y));

}

#### Độ phức tạp

n (số bit biểu diễn số nguyên). Ví dụ, nhân hai số nguyên trên máy 32 bit ta cần cỡ 32 phép toán dịch bit và cộng.

### Chia Ấn Độ

#### Chương trình

// Nhan, Chia An Do: Thuong va du

#include <bits/stdc++.h>

using namespace std;

int Mult(int x, int y) {

int z = 0;

while(x > 0) {

if(x & 1) z += y;

x >>= 1; // x = x/2

y <<= 1; // y = y\*2

}

return z;

}

// de quy

int RMult(int x, int y) {

return (x == 0 || y == 0) ? 0

: ((x & 1) ? RMult(x/2, y+y) + y

: RMult(x/2, y+y));

}

// x = qy + r

void DivMod(int x, int y, int &q, int &r) {

if(y == 0) {

cerr << "\n Error: divide by zero.";

exit(3);

}

int w = y;

while (w <= x) w <<= 1; // w = w\*2

q = 0; r = x;

while(w > y) {

w >>= 1; // chia doi thuoc w

q <<= 1; // nhan doi thuong

if (w <= r) { ++q; r -= w; }

}

}

int main () {

int x = 123, y = 456;

int q, r;

DivMod(x\*y+37, y, q, r);

cout << "\n " << x\*y+37 << ":" << y << " -> thuong q = "

<< q << ", du r = " << r;

cout << "\n " << "y\*q + r = " << Mult(y,q)+r << " =? " << x\*y+37;

cout << "\n T h e E n d";

return 0;

}

### Output

56125:456 -> thuong q = 123, du r = 37

y\*q + r = 56125 =? 56125

T h e E n d

#### Độ phức tạp

n (số bit biểu diễn số nguyên).

### Các lũy thừa tự thân (Self powers)

#### Thuật toán

Các số dạng được gọi là các *lũy thừa tự thân.* *k* chữ số cuối của số nguyên dương *x* là số Trong bài này *k* = 10. Do đó ta cần tính toán theo *phép chia dư* *m* (modulo *m*) với các số nguyên 64 bits, trong đó *m* = = 10000000000 (10G = mười tỷ).

Các quy tắc đơn giản dưới đây được vận dụng để tính toán:

(*a* + *b*) mod *m* = ((*a* mod *m*) + (*b* mod *m*)) mod *m*

(*a* × *b*) mod *m* = ((*a* mod *m*) × (*b* mod *m*)) mod *m*

*ak* mod *m* = (*a* mod *m*)*k* mod *m*

*m* = 10G.

Tóm lại, nếu bạn chỉ quan tâm đến mười chữ số cuối cùng của kết quả thì mọi phép tính cộng, trừ và nhân trung gian (không kể phép chia) đều phải được tính toán theo mdulo *m* = = 10G.

Với *m* = 10G cho trước, ta cần triển khai các hàm sau đây:

Plus(a, b) = (a + b) mod m,

Mult(a, b) = (a × b) mod m, và

Exp(a, k) = ak mod m.

với yêu cầu là các hàm trên không được sinh ra các kết quả tính toán trung gian vượt quá giá trị *m.*

Để ý rằng mặc dù 0 ≤ *a, b* < *m*, nhưng có thể *a*+*b* ≥ *m*, nghĩa là tổng *a*+*b* có thể gây ra hiện tượng tràn số. Do đó ta cần thay phép kiểm tra *a*+*b* ≥ *m* bằng phép kiểm tra *a* ≥ *m* − *b* và thay phép tính (*a* + *b*) mod *m* bằng phép tính (*a* − (*m* − *b*)) vì ta luôn luôn có 0 ≤ *a, b* < *m.*

Ta cũng có thể xen trực tiếp đoạn mã của hàm Plus vào chương trình nhằm giảm thời gian truyền tham số.

Nếu x là bội của 10 thì ta bỏ qua, vì khi x = y10k, thì xx sẽ có trên 10 chữ số 0 tận cùng, do đó xx mod 10G = 0.

Giả sử ta cần tính nhanh *z* = (*xy*) *mod m* và không gây hiện tượng tràn số, vì tích trung gian xy có thể lớn. Ta dựa trên hệ thức

Phép luỹ thừa theo modulo *z* = *ax* *mod m* cũng được tính phương pháp chia để trị với hệ thức quen thuộc sau

#### Chương trình

// Self powers

#include <iostream>

using namespace std;

typedef unsigned long long UL;

const UL m = 10000000000; // 10G

#define Plus(a,b) (((a) >= m-(b)) ? (a)-(m-(b)) : ((a)+(b)));

// z = ab mod m

UL Mult(UL a, UL b) {

UL z = 0;

while (a > 0) {

if (a & 1) // a is odd

z = Plus(z,b); // z = (z+b) mod m

a >>= 1; // a = a div 2

b = Plus(b,b); // b = 2b mod m

} // while

return z;

}

// z = a^k mod m

UL Exp(UL a, int k) {

UL z = 1;

while (k > 0) {

if (k & 1) // k is odd

z = Mult(z,a); // z = (z\*a) mod m

k >>= 1; // k = k div 2

a = Mult(a,a); // a = a\*a mod m

} // while

return z;

} // Exp

void SelfPower(int n) {

UL sum = 1; // sum = 1^1

for (int x = 2; x <= n; ++x)

if (x % 10) sum = Plus(sum, Exp(x,x));

cout << "\n " << sum;

}

main() {

SelfPower(1000); // 9110846700, Time = 0.012 sec.

cout << "\n T H E E N D.";

return 0;

}

### Triangle Number

Giải bài Số tam giác bằng PP chia để trị

Cho số x tìm số n để T(n) = n\*(n+1) / 2 = x ?

/\*

Name: Triangle Number

Copyright: (C) 2023

Author: DevCPP Fan

Date: 25-06-23 15:43

Description:

\*/

#include <bits/stdc++.h>

using namespace std;

void Go() {

cout << " ? ";

fflush(stdin);

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

}

inline int T(int n) {return n\*(n+1) / 2; }

// n ? T(n) = n(n+1)/ 2 = x

int TNum(int x) {

if(x == 0 || x == 1) return x;

int d = 1, c = x, n, y;

while(d < c) {

n = (d+c) / 2;

if(T(n) < x) d = n+1;

else c = n;

}

// d = c

return(T(d) == x) ? d : -1;

}

void Run() {

for(int n = 0; n < 60; ++n) {

int x = T(n);

cout << "\n n = " << n << " ? " << TNum(x) << " " << TNum(x-1);

}

}

main() {

Run();

cout << endl << "\n\n T h e E n d \n";

return 0;

}

### Secret number

*Câu hỏi YN là câu hỏi chỉ cần nhận được câu trả lời là Y (Yes) / N (No). Trọng tài ghi nhận một số nằm trong khoảng d..c. Bạn được phép dùng các câu hỏi Y/N để tìm ra số đó. Tối đa bạn phải hỏi bao nhiêu lần?*

#### *Thuật toán*

Phương pháp tìm kiếm nhị phân cho phép ta hỏi không quá log(n), n = c-d+1 câu hỏi Y/N phát hiện ra số bí mật của trọng tài.

Thuật toán tìm kiếm nhị phân Search(d, c, ask) dưới đây có thêm tham biến tuỳ biến ask là nội dung câu hỏi để vận dụng linh hoat trong một số tình huống.

#### Chương trình

// Secret Number

#include <bits/stdc++.h>

using namespace std;

inline void Go() {

cout << " ? ";

fflush(stdin);

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

}

int Search(int d, int c, const char \* ask = "") {

int h = 0; // so cau hoi

int m; // diem giua

char res; // cau tra loi

while(d < c) {

m = (d+c)/2;

++h;

while(true) {

cout << "\n " << h << ". " << ask << m << " ? [Y|N]: ";

fflush(stdin); // lau vung dem

res = toupper(cin.get()); // nhan Y/N chuyen sang chu HOA

if (res == 'Y' || res == 'N') break;

}

if (res == 'Y') d = m+1;

else c = m; // res = 'N'

}

return d;

}

void SecretNumber(int d, int c) {

cout <<"\n You must remember a secret number between " << d <<".."<<c;

cout << "\n Now, press a key to begin: "; Go();

int x = Search(d, c, " Is your secret number bigger than ");

cout << "\n Your secret number is = " << x;

}

main() {

SecretNumber(1, 32); // so bi mat x trong khoang 1..32

cout << "\n T h e E n d";

return 0;

}

#### *Output*

You must remember a secret number between 1..32

Now, press a key to begin: ?

1. Is your secret number bigger than 16 ? [Y|N]: n

2. Is your secret number bigger than 8 ? [Y|N]: n

3. Is your secret number bigger than 4 ? [Y|N]: y

4. Is your secret number bigger than 6 ? [Y|N]: n

5. Is your secret number bigger than 5 ? [Y|N]: y

Your secret number is = 6

T h e E n d

### Sinh nhật

*Cần bao nhiêu hỏi YN để có thể đoán được sinh nhật (gồm ngày và tháng) của một người ?*

#### *Thuật toán*

Để xác định ngày sinh ta cần tối đa log(31) = 5 câu hỏi, xác định tháng sinh cần log(12) = 4 câu hỏi (ký hiệu log(x) cho giá trị nguyên đầu tiên m thỏa 2m ≥ x). Tổng cộng ta cần 9 câu hỏi.

#### *Chương trình*

// Birthday

#include <bits/stdc++.h>

using namespace std;

int Search(int d, int c, const char \* ask = "") {...}

void Birthday() {

int ngay = Search(1, 31, "Ban sinh sau ngay ");

int thang = Search(1, 12, "\*\* Ban sinh sau thang ");

cout << "\n Ban sinh ngay " << ngay << " thang " << thang;

}

main() {

Birthday();

cout << "\n T h e E n d";

return 0;

}

#### *Output*

1. Ban sinh sau ngay 16 ? [Y|N]: n

2. Ban sinh sau ngay 8 ? [Y|N]: y

3. Ban sinh sau ngay 12 ? [Y|N]: y

4. Ban sinh sau ngay 14 ? [Y|N]: n

5. Ban sinh sau ngay 13 ? [Y|N]: y

1. \*\* Ban sinh sau thang 6 ? [Y|N]: n

2. \*\* Ban sinh sau thang 3 ? [Y|N]: n

3. \*\* Ban sinh sau thang 2 ? [Y|N]: n

4. \*\* Ban sinh sau thang 1 ? [Y|N]: y

Ban sinh ngay 14 thang 2

T h e E n d

### Tom And Jerry

*Giaỉ mã văn bản bí mật sau*

🐈🖲🖲🐈🖲🐈🐈🖲🐈🐈🖲🖲🖲🖲🖲🐈🐈🐈🐈🖲🖲🖲🐈🐈🖲🐈🖲🐈🖲🖲🐈🖲🖲🖲🐈🐈🖲🐈🖲🐈

*Biết*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | C | E | H | M | O | T | U | V | \* |
| 🐈🐈🐈 | 🐈🐈🖲 | 🐈🖲🐈🐈 | 🐈🖲🐈🖲 | 🐈🖲🖲 | 🖲🐈🐈 | 🖲🐈🖲🐈 | 🖲🐈🖲🖲 | 🖲🖲🐈 | 🖲🖲🖲 |

Ta thấy văn bản mật chỉ gồm hai ký hiệu là mèo 🐈và chuột máy 🖲, do đó ta dễ dàng chuyển đổi thành hai trị: 0 cho mèo 🐈và 1 cho chuột máy 🖲. Đề bài khi đó sẽ được chuyển thành:

*Giaỉ mã*

🐈🖲🖲🐈🖲🐈🐈🖲🐈🐈🖲🖲🖲🖲🖲🐈🐈🐈🐈🖲🖲🖲🐈🐈🖲🐈🖲🐈🖲🖲🐈🖲🖲🖲🐈🐈🖲🐈🖲🐈

🡪 0110100100111110000111001010110111001010

*Biết*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | C | E | H | M | O | T | U | V | \* |
| 🐈🐈🐈 | 🐈🐈🖲 | 🐈🖲🐈🐈 | 🐈🖲🐈🖲 | 🐈🖲🖲 | 🖲🐈🐈 | 🖲🐈🖲🐈 | 🖲🐈🖲🖲 | 🖲🖲🐈 | 🖲🖲🖲 |
| 000 | 001 | 0100 | 0101 | 011 | 100 | 1010 | 1011 | 110 | 111 |

Ta vận dụng *cây tìm kiếm nhị phân* cho bài toán này.

Hãy mường tượng là bạn cần lập một sơ đồ đường dẫn để tìm một số đối tượng. Điểm xuất phát được gọi là *gốc* của sơ đồ. Mỗi bước ta chỉ có thể di chuyển *xuống dưới* theo một trong hai hướng đi là *rẽ trái* theo chỉ dẫn 0 hoặc *rẽ phải* theo chỉ dẫn 1. Để đi từ gốc đến vị trí của điểm A = 000 bạn cần ba lần rẽ trái. Muốn đi từ gốc đến điểm M = 011 bạn cần rẽ trái, rẽ phải, rẽ phải... Sơ đồ này được gọi là *cây nhị phân tìm kiếm,* vì mỗi vị trí trong cây chỉ có không quá hai khả năng di chuyển xuống dưới là rẽ trái (0) hoặc rẽ phải (1). Mỗi hình tròn trong sơ đồ được gọi là một *đỉnh* của cây nhị phân. Đỉnh đầu tiên, số hiệu 1 dược gọi là *gốc* của cây nhị phân. Các đỉnh cuối cùng từ đó không còn đỉnh kế tiếp được gọi là *lá* hoặc *đỉnh treo* của cây nhị phân. Dựa theo bảng mã, bạn có thể xây dựng được cây nhị phân như trong hình với 10 đỉnh treo là A, C, E, H, M, O, T, U, V,\*.

Diagram, shape

Description automatically generated

Có sơ đồ trong tay, bạn dễ dàng giải mã được văn bản mật theo thuật toán sau đây.

Decode(s):

Input: văn bản mật s

Output: văn bản rõ t

t = ""; // string rỗng

Xuất phát từ gốc i = 1

for each symbol b in s do

if b == '0' then rẽ trái

else rẽ phải // b == '1'

end if

if gặp lá then

xuất ký tự trên lá

quay lại gốc i = 1

end if

end for

end Decode

Ví dụ

Bạn hãy theo sơ đồ trên, thử giải vài ký tự đầu tiên của văn bản mật: "0110100100111110...”

'0' (trái) '1' (phải) '1' (phải) (xuất **M**)

'0' (trái) '1' (phải) '0' (trái) '0' (trái) (xuất **E**)

'1' (phải) '0' (trái) '0' (trái) (xuất **O**)

'1' (phải) '1' (phải) '1' (phải) (xuất **\***)

'1' (phải) '1' (phải) '0' (trái) (xuất **V**)…

#### Biểu diễn cây

Trong bài này cây nhị phân được cài đặt ra sao?

Chỉ cần một mảng tham chiếu. Ta gọi mảng đó là a. Ta gán trị a[v] = c trong đó v là số hiệu của (đỉnh) lá, c là ký tự trên lá. Ví dụ trên cho ta cấu trúc của mảng a như sau :

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| v : Số hiệu đỉnh | … | 8 | 9 | … | 11 | 12 | … | 14 | 15 | … | 20 | 21 | … | 26 | 27 | … |
| Ký tự | # | A | C | # | M | O | # | V | \* | # | E | H | # | T | U | # |

trong đó dấu # biểu diễn cho giá trị trống dùng để gán cho các đỉnh không phải là lá.

Việc còn lại chỉ là xác định các *số hiệu* cho các đỉnh.

|  |  |
| --- | --- |
| Diagram  Description automatically generated | Mỗi *đỉnh* i trong cây nhị phân có tối đa *hai đỉnh con* là *con trái* và *con phải*. i được gọi là đỉnh *cha.* Số hiệu của mỗi đỉnh được tính như sau:  Đỉnh cha: i  Đỉnh con trái theo nhánh 0: 2i  Đỉnh con phải theo nhánh 1: 2i+1  Đỉnh gốc: 1 |

Điều thú vị là ta có thể tính trực tiếp *số hiệu đỉnh treo* cho mỗi ký tự theo mã của chúng theo cùng một thuật toán giống như thuật toán giải mã. Thật vậy, giả sử ký tự tại đỉnh treo trong cây có mã là .

Code(m):

Input: mã 0/1 m

Output: v số hiệu của đỉnh treo

v = 1

for each symbol b in m do

if b == '0' then v = v\*2 // rẽ trái

else v = v\*2 + 1 // rẽ phải

end if

end for

return v

end Code

Ví dụ

Giả sử ta cần xác định số hiệu v cho đỉnh treo (lá) ký tự T có mã 1010.

Ta đặt giá trị (**1**) cho v vào bên trái dãy mã 1010: (**1**)1010.

Tiếp theo ta dịch qua phải từng bit b của T và cập nhật **v** theo quy tắc :

(**v**) '0' (**2v**): nếu b = '0' thì cập nhật **v** = **2v**

(**v**) '1' (**2v+1**): nếu b ='1' thì cập nhật **v** = **2v+1**

Sơ đồ chi tiết của các bước được thể hiện như sau:

(**1**) "1010" → "1" (**3**) "010" → "10" (**6**) "10" → "101" (**13**) "0" → "1010" (**26**).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Xác định số hiệu cho đỉnh treo T có mã 1010* | | | | | | | | |
| m | m1 |  | m2 |  | m3 |  | m4 |  |
| (**1**) | '1' | (**3**) | '0' | (**6**) | '1' | (**13**) | '0' | (**26**) |
| v |  | v |  | v |  | v |  | v |
| (v)'0' → (2v); (v)'1' → (2v+1)  "1010" → 26 | | | | | | | | |

Vậy đỉnh treo T có số hiệu v = 26.

#### Chú ý

Các ô trống trong mảng a không ứng với ký tự nào cần được khởi trị bằng một ký tự không xuất hiện trong văn bản, ví dụ, ‘#’.

Ta ghi dữ liệu vào cho ví dụ trên trong text file TOMJERRY.INP như sau:

|  |  |
| --- | --- |
| TOMJERRY.INP | *Giải thích* |
| 10  A 000  C 001  E 0100  H 0101  M 011  O 100  T 1010  U 1011  V 110  \* 111  0110100100111110000111001010110111001010 | *10 ký tự*  *mã số của A*  *mã số của C*  *mã số của E*  *mã số của H*  *mã số của M*  *mã số của O*  *mã số của T*  *mã số của U*  *mã số của V*  *mã số của \**  *mã số của văn bản cần giải mã* |

Các bước của thuật toán tạo cây và giải mã sẽ như sau:

|  |  |
| --- | --- |
| Bước 1 | Tạo cây  Khởi trị mảng a toàn ký tự ‘#’  Xác định số hiệu v cho mỗi ký tự c theo mã của chúng.  Gán trị a[v] = c |
| Bước 2 | Giải mã  Xuất phát từ gốc v = 1.  Mỗi lần xét một ký hiệu b trong văn bản mật:  Nếu b = '0': v = 2v (rẽ trái)  Nếu b = '1': v = 2v+1 (rẽ phải)  Nếu a[v] ≠ '#': xuất a[v]; v → 1 (quay lại gốc). |

#### Chương trình

// TomJerry.CPP

#include <bits/stdc++.h>

using namespace std;

#define toint(c) (c)-'0'

const int MN = 1024;

const char \* FN = "TOMJERRY.INP";

char a[MN];

// So hieu cua dinh

int Code(string &s) {

int v = 1;

for(int i = 0; i < s.length(); ++i)

v = (v \* 2) + toint(s[i]);

return v;

}

void Decode(string &s) {

cout << "\n Decode " << s << "\n ";

int v = 1;

for(int i = 0; i < s.length(); ++i) {

v = (v \* 2) + toint(s[i]);

if (a[v] != '#') {

cout << a[v];

v = 1;

} // if

} // for

}

void Run() {

fill(a, a+MN, '#');

ifstream f(FN);

string s;

char c;

int n;

f >> n; // so ky tu

for (int i = 1; i <= n; ++i) {

f >> c >> s; // ky tu c co ma s

cout << "\n " << c << " : " << s;

int v = Code(s);

a[v] = c;

cout << " -> " << v;

}

// Doc doan code can giai ma

f >> s;

f.close();

Decode(s);

}

main() {

Run();

cout << "\n T h e E n d.";

return 0;

}

#### Output

A : 000 -> 8

C : 001 -> 9

...

V : 110 -> 14

\* : 111 -> 15

Decode ...

MEO\*VA\*CHUOT

T h e E n d.

#### Độ phức tạp

Tổ chức tìm kiếm mỗi phần tử trên cây nhị phân có độ phức tạp cỡ log(n).