Nguyễn Xuân Huy

**Sorting And Searching**

**Các chuyên đề chọn lọc**

Hà Nội - 2023

# Sắp xếp

### Bài toán sắp xếp

Cho mảng a gồm n phần tử thuộc kiểu sánh được. Ta cần viết hàm Sort(a, d, c) sắp lại các phần tử của mảng a[d..c] theo trật tự tăng dần, kể từ chỉ số d đến chỉ số c.

Ví dụ

a[0..6] = [7,2,9,1,2,5,3];

Sort(a,1,5); // a[1..5] = [2,9,1,2,5] → [1,2,2,5,9]

a[1..5] = [1,2,2,5,9];

Nếu muốn sắp giảm mảng a, ta gọi hàm Sort sau đó lật lại mảng a.

Ví dụ

a[0..6] = [7,2,9,1,2,5,3];

Sort(a,1,5); // a[1..5] = [1,2,2,5,9];

Rev(a,1,5);

a[1..5] = [9,5,2,2,1];

Một trong những yêu cầu thường gặp là *sắp tại chỗ*, nghĩa là không dùng thêm mảng phụ trong quá trình sắp xếp.

Ta tìm hiểu ba thuật toán sắp xếp quan trọng là *thuật toán chọn*, *thuật toán nổi bọt* và *thuật toán sắp nhanh*.

### Thuật toán chọn

Mảng a được chia làm hai phần: phần đầu S là dãy đã sắp, phần còn lại C là dãy chưa sắp: a = S|C. Mỗi lần chọn phần tử có trị nhỏ nhất trong C đặt vào cuối S.

Ta mường tượng dùng tay trái t trỏ từ phần tử đầu dãy d đến phần tử cuối dãy c, tay phải p đặt cạnh tay trái rồi di chuyển dần về cuối dãy. Nếu gặp phần tử phải nhỏ thua phần tử trái thì ghi nhận chỉ số nhỏ này vào biến imin. Sau mỗi lần duyệt xong tay phải ta tìm được imin trỏ vào phàn tử nhỏ nhất của phần còn lại a[t..c]. Ta đưa a[imin] vào vị trí t bằng phép đổi chỗ.

// sap tang a[d..c]

void SelectSort(int a[], int d, int c) {

for(int t = d; t < c; ++t) {

int imin = t;

for(int p = t + 1; p <= c; ++p) {

if (a[p] < a[imin]) imin = p;

}

Swap(a, imin, t);

}

}

Độ phức tạp

Thuật toán chứa hai vòng lặp lồng nhau nên cần khoảng n2 phép so sánh.

### Thuật toán nổi bọt

Mảng a cũng được chia làm hai phần: phần đầu S là dãy đã sắp, phần còn lại C là dãy chưa sắp. Mỗi lần chọn phần tử có trị nhỏ nhất trong C đặt vào cuối S.

Hãy tưởng tượng là bạn đặt mảng a vào một thùng nước theo chiều dọc, dùng tay trái t trỏ từ phần tử đầu dãy d đến phần tử cuối dãy c, tay phải p *chạy ngược* từ cuối dãy c lên đến sát tay trái. Mỗi lần so sánh hai bong bóng a[p] với phần tử sát trên là a[p-1]. Nếu bong bóng dưới nhẹ hơn bong bóng trên thì đổi chỗ cho bong bóng dưới nổi lên.

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, p-1);

}

Độ phức tạp

Thuật toán chứa hai vòng lặp lồng nhau nên cần khoảng n2 phép so sánh.

### Thuật toán sắp nhanh

Một trong các giải thuật sắp nhanh là Quick Sort do Hoare đề xuất. Thuật toán dùng phương pháp chia để trị gồm hai pha như sau:

Pha 1. Chia mảng a[d..c] thành hai phần, phần đầu S và phần cuối C. Hai phần này chưa được sắp nhưng đáp ứng tiêu chí sau đây:

Mọi phần tử x của S nhỏ thua hoặc bằng phần tử giữa m = a[(d+c)/2]: x ≤ m

Mọi phần tử y của C lớn hơn hoặc bằng phần tử giữa m: m ≤ y

Nếu ta phát hiện ra hai phần tử x trong S và y trong C không đáp ứng được tiêu chí trên, cụ thể là x > m và y < m thì ta hiểu rằng x và y là hai phần tử *đặt nhầm chỗ*. Ta đổi chỗ hai phần tử này.

Pha 2. Sắp hai phần S và C.

void QuickSort(int a[], int d, int c) {

int i = d, j = c, m = a[(d+c)/2]; // m: phan tu giua

while(i <= j) {

while(a[i] < m) ++i; // nua trai S

while(a[j] > m) --j; // nua phai C

if (i <= j) {

Swap(a,i,j);

i++; j--;

}

}

if (d < j) QuickSort(a, d, j);

if (i < c) QuickSort(a, i, c);

}

Độ phức tạp

Gọi k là số lần chia mảng a thành hai phần S và C, ta có k = log(n). Mỗi lần chia như vậy ta cần n phép so sánh. Vậy độ phức tạp cỡ nlog(n).

### So sánh ba thuật toán sắp xếp

Nếu n = 1000 thì SelectSort và BubbleSort cần khoảng 10002 = 1000000 phép so sánh trong khi Quick Sort cần khoảng 1000log(1000) = 1000×10 = 10000 phép so sánh, nhanh hơn khoảng 100 lần.

Ta xây dựng chương trình so sánh tốc độ của ba thuật toán sắp xếp như sau:

Với mỗi phương pháp sắp xếp ta thực hiện và quan sát các bước sau đây:

* Gọi hàm Gen(n) sinh ngẫu nhiên n = 50000 số nguyên dương cho mảng a.
* Ghi nhận thời gian bắt đầu long long t1 = time(NULL);
* Gọi hàm Sort
* Ghi nhận thời gian kết thúc long long t2 = time(NULL);
* Tính tổng thời gian đã sắp: difftime(t2,t1)

Để sinh ngẫu nhiên các giá trị, trước hết bạn cần khởi động bộ sinh ngẫu nhiên bằng lệnh

srand(time(NULL)); Lệnh này lấy giá trị time(NULL) là thời gian tính từ thời điểm gốc NULL đến thời điểm hiện tại làm giá trị xuất phát. Sau đó, mỗi lần gọi hàm rand() % k, hệ thống sẽ sinh ra một số ngẫu nhiên trong khoảng 0..k-1.

Hàm difftime(t2, t1) cho ra thời gian tính từ thời điểm t1 đến t2.

#### Chương trình

// Test Sort

#include <bits/stdc++.h>

using namespace std;

const int MN = 50000;

int a[MN+1];

int n;

// Hien thi mang a[d..c] kem chu thich

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 i, int j) {

int x = a[i]; a[i] = a[j]; a[j] = x;

}

void QuickSort(int a[], int d, int c) {...}

void BubbleSort(int a[], int d, int c) {...}

void SelectSort(int a[], int d, int c) {...}

void Gen(int a[], int n) {

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

a[i] = rand() % MN + 1;

}

main() {

n = MN;

srand(time(NULL));

int d = 0, c = n-1;

long long t1, t2;

Gen(a,n);

t1 = time(NULL);

cout << "\n Begin of QuickSort...";

QuickSort(a, d, c);

t2 = time(NULL);

cout << "\n End of QuickSort: " << difftime(t2,t1);

Gen(a,n);

t1 = time(NULL);

cout << "\n Begin of BubbleSort...";

BubbleSort(a, d, c);

t2 = time(NULL);

cout << "\n End of BubbleSort: " << difftime(t2,t1);

Gen(a,n);

t1 = time(NULL);

cout << "\n Begin of SelectSort...";

SelectSort(a, d, c);

t2 = time(NULL);

cout << "\n End of SelectSort: " << difftime(t2,t1);

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

return 0;

}

## Thư viện các hàm sắp xếp trong C++

### Sort theo tiêu chí định sẵn

Đôi khi ta gọi kiểu sort này là *sắp xếp tự nhiên.*

Trong C++ để *sắp tăng* một đoạn trong mảng một chiều a từ phần tử a[i] đến phần tử a[j], ta gọi theo hàm sort với tham biến truyền theo biến (địa chỉ):

sort(a+i, a+j+1);

Muốn *sắp tăng* toàn bộ n phần tử của mảng a, ta gọi

sort(a, a+n);

Muốn *sắp giảm* mảng số thực a[i..j], ta viết

sort(a+i, a+j+1, greater<float>());

Như vậy, hàm greater<float>() sẽ so sánh hai đối tượng kiểu float trong a[] theo chiều giảm.

#### Chương trình minh hoạ

// ArraySort.CPP

#include <bits/stdc++.h>

using namespace std;

const int N = 10;

int a[] = {2, 6, 0, 8, 1, 7, 4, 9, 3, 5};

float b[] = {2.12, 6.08, 0, 8, 1, 7, 3.14, 9, 3.17, 5};

// Hien thi mang 1D a kem thong bao

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

cout << msg;

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

}

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

cout << msg;

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

}

int main() {

Print(a, 0, N-1, "\n Given a: ");

sort(a, a+N);

Print(a, 0, N-1, "\n Sorted a: ");

Print(b, 0, N-1, "\n Given b: ");

sort(b, b+N, greater<float>());

Print(b, 0, N-1, "\n dec sorted b: ");

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

return 0;

}

#### Kết quả

Given a: 2 6 0 8 1 7 4 9 3 5

Sorted a: 0 1 2 3 4 5 6 7 8 9

Given b: 2.12 6.08 0 8 1 7 3.14 9 3.17 5

dec sorted b: 9 8 7 6.08 5 3.17 3.14 2.12 1 0

T h e E n d

### Sort theo tiêu chí tự đặt

Đôi khi bạn muốn thay đổi tiêu chí so sánh các phần tử của mảng, ví dụ bạn muốn sắp tăng mảng nguyên không âm a theo *độ cao* H của mỗi số.

Bạn muốn cài đặt hàm H theo tùy biến, cụ thể là H(x) sẽ cho ra tổng các chữ số của số nguyên x theo hệ đếm 10 (hệ thập phân), còn H(x, 2) sẽ cho ra tổng các chữ số của số nguyên x theo hệ đếm 2. Nói cách khác, H(x, 2) cho biết trong dạng nhị phân biểu diễn số nguyên x có bao nhiêu bit 1.

Ví dụ, H(19) = 1 + 9 = 10, H(19, 2) = 3, vì 192 = 100112.

Việc cài đặt hàm H(x) khá dễ, bạn chỉ việc lấy tổng các chữ số của x là x mod base tính theo hệ đếm nào thì ta chỉ việc chia theo hệ đếm đó. Hàm x div base bỏ đi chữ số đơn vị của x.

// Do cao theo base

int H(int x, int base = 10) {

return (x == 0) ? 0 : (x % base) + H(x/base, base);

}

Tiếp đến bạn phải tự cài đặt phương thức so sánh hai phần tử của mảng a. Bạn quy định rằng x < y khi và chỉ khi H(x) < H(y):

bool Less(int x, int y) { return H(x) < H(y); }

Đến đây bạn gọi hàm sort:

sort(a, a+n, Less);

Nếu muốn sắp giảm bạn gọi

sort(a, a+n, Great);

với hàm Great do bạn tự viết như sau:

bool Great(int x, int y) { return H(x) > H(y); }

#### Minh hoạ

// ArraySort.CPP

#include <bits/stdc++.h>

using namespace std;

const int N = 10;

int a[] = {20, 16, 10, 0, 111, 73, 46, 19, 33, 15};

int b[N];

// Do cao theo base

int H(int x, int base = 10) {

return (x == 0) ? 0 : (x % base)+H(x/base, base);

}

// Hien thi a[i1]...a[i2] va do cao theo base 10 | 2

void Print(int a[], int d, int c, char \* msg = "", int base = 10) {

cout << msg;

for (int i = d; i <= c; ++i) {

cout << " " << a[i] << ":" << H(a[i], base);

}

}

// x < y theo base 10

bool Less(int x, int y) {

return H(x) < H(y);

}

// x > y theo base 10

bool Great(int x, int y) {

return H(x) > H(y);

}

// x < 2 theo base 2

bool Less2(int x, int y) {

return H(x, 2) < H(y, 2);

}

// x > y theo base 2

bool Great2(int x, int y) {

return H(x, 2) > H(y, 2);

}

int main() {

memcpy(b, a, sizeof(a));

// b = a

Print(a, 0, N-1, "\n Given a in base 10: ");

sort(a, a+N, Less);

Print(a, 0, N-1, "\n inc sorted a by H of base 10: ");

sort(a, a+N, Great);

Print(a, 0, N-1, "\n dec sorted a by H of base 10: ");

Print(a, 0, N-1, "\n Now a in base 2: ", 2);

sort(a, a+N, Less2);

Print(a, 0, N-1, "\n inc sorted a by H of base 2: ", 2);

sort(a, a+N, Great2);

Print(a, 0, N-1, "\n dec sorted a by H of base 2: ", 2);

Print(b, 0, N-1, "\n Given b in base 10: ");

sort(b+2, b+8, Less);

Print(b, 2, 7, "\n inc sorted b[2:7] by H of base 10: ");

sort(b+2, b+8, Great);

Print(b, 2, 7, "\n dec sorted b[2:7] by H of base 10: ");

Print(b, 0, N-1, "\n Now b in base 2: ", 2);

sort(b+2, b+8, Less2);

Print(b, 2, 7, "\n inc sorted b[2:7] by H of base 2: ", 2);

sort(b+2, b+8, Great2);

Print(b, 2, 7, "\n dec sorted b[2:7] by H of base 2: ", 2);

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

return 0;

}

#### Kết quả

Given a in base 10: 20:2 16:7 10:1 0:0 111:3 73:10 46:10 19:10 33:6 15:6

inc sorted a by H of base 10: 0:0 10:1 20:2 111:3 33:6 15:6 16:7 73:10 46:10 19:10

dec sorted a by H of base 10: 73:10 46:10 19:10 16:7 33:6 15:6 111:3 20:2 10:1 0:0

Now a in base 2: 73:3 46:4 19:3 16:1 33:2 15:4 111:6 20:2 10:2 0:0

inc sorted a by H of base 2: 0:0 16:1 33:2 20:2 10:2 73:3 19:3 46:4 15:4 111:6

dec sorted a by H of base 2: 111:6 46:4 15:4 73:3 19:3 33:2 20:2 10:2 16:1 0:0

Given b in base 10: 20:2 16:7 10:1 0:0 111:3 73:10 46:10 19:10 33:6 15:6

inc sorted b[2:7] by H of base 10: 0:0 10:1 111:3 73:10 46:10 19:10

dec sorted b[2:7] by H of base 10: 73:10 46:10 19:10 111:3 10:1 0:0

Now b in base 2: 20:2 16:1 73:3 46:4 19:3 111:6 10:2 0:0 33:2 15:4

inc sorted b[2:7] by H of base 2: 0:0 10:2 73:3 19:3 46:4 111:6

dec sorted b[2:7] by H of base 2: 111:6 46:4 73:3 19:3 10:2 0:0

T h e E n d .

### Sort theo chỉ dẫn

Trong quá trình sort, các đối tượng trong mảng thường phải đổi chỗ với nhau cho nên chi phí cho các phép toán copy thường làm tăng thời gian sắp xếp, đặc biệt là với những đối tượng có kích thước lớn như string hoặc bản ghi. Sort theo chỉ dẫn cho phép chúng ta chỉ ra *một trật tự*, ví dụ, tăng dần, các đối tượng của mảng nhưng vẫn *giữ nguyên vị trí ban đầu* của chúng.

Ví dụ

Ta có mảng name chứa danh sách các bạn trẻ. Ta cần duyệt danh sách này theo trật tự từ điển như dòng 4 trong bảng dưới đây:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| init name | HOA | KHANH | AN | TUAN | BINH | VINH | CHINH | VAN | DUNG | MAI |
| Init id | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| sorted id | 2 | 4 | 6 | 8 | 0 | 1 | 9 | 3 | 7 | 5 |
| Print name by id | AN | BINH | CHINH | DUNG | HOA | KHANH | MAI | TUAN | VAN | VINH |

Để thực hiện điều này ta cần một mảng phụ tạm gọi là id để quản lý số thứ tự (chỉ số) của mảng.

Quy trình sort sẽ được thực hiện qua các bước sau:

Bước 1. Khởi trị cho mảng id:

id[i] = i, 0 ≤ i < n

trong đó n là số phần tử của mảng.

Dòng 2 của bảng cho ta kết quả của bước khởi trị id.

Bước 2. Cài đặt hàm Less(i,j) chỉ rõ phương thức so sánh hai đối tượng name[i] và name[j], cụ thể là cho biết name[i] < name[j] ?

bool Less(int i, int j) {

return name[i] < name[j];

}

Bước 3. Gọi hàm sort mảng id chứ không phải mảng name:

sort(id, id+n, Less);

với ý nghĩa sắp xếp lại mảng phụ id từ vị trí đầu đến cuối theo phương thức so sánh Less.

Như vậy chỉ có *các phần tử trong id được sắp xếp lại*, còn bản thân các string trong mảng name vẫn được giữ nguyên.

Dòng 3 của bảng cho ta kết quả của bước 3

id = [2, 4, 6, 8, 0, 1, 9, 3, 7, 5]

với ý nghĩa:

* id[0] = 2 cho biết phần tử name[2] (AN) sẽ đứng ở vị trí đầu tiên,
* id[1] = 4 cho biết phần tử name[4] (BINH) sẽ đứng ở vị trí thứ hai...

Ngoài ưu thế tiết kiệm thời gian, sort theo chỉ dẫn còn được ứng dụng trong các bài toán cần tham chiếu đến vị trí ban đầu của dữ liệu.

#### Chương trình minh hoạ

// ArraySort theo chi dan

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

string name[] = {"HOA", "KHANH", "AN", "TUAN", "BINH",

"VINH", "CHINH", "VAN", "DUNG", "MAI"};

int n = 10;

int \*id;

// Hien thi mang string a kem chu thich

void Print(string a[], int n, char \* msg = "") {...}

// Hien thi mang int a kem chu thich

void Print(int x[], int n, char \* msg = "") {...}

// Hien thi mang string a theo chi dan

void PrintById(string \*a, int \*id, int n, char \* msg = "") {

cout << msg;

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

cout << " " << a[id[i]];

}

}

bool Less(int i, int j) {

return name[i] < name[j];

}

int main() {

Print(name, n, "\n Given a: ");

id = new int[n];

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

Print(id, n, "\n Init id: ");

sort(id, id+n, Less);

Print(id, n, "\n Now id: ");

PrintById(name, id, n, "\n Now Names: ");

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

return 0;

}

#### Kết quả

Given a: HOA KHANH AN TUAN BINH VINH CHINH VAN DUNG MAI

Init id: 0 1 2 3 4 5 6 7 8 9

Now id: 2 4 6 8 0 1 9 3 7 5

Now Names: AN BINH CHINH DUNG HOA KHANH MAI TUAN VAN VINH

T h e E n d .

### Tổng kết các hàm Sort trong thư viện C++

|  |  |
| --- | --- |
| **sort array a[i..j]** |  |
| sort(a+i, a+j+1) | *sắp tăng* |
| sort(a+i, a+j+1, Less) | *sắp tăng theo tiêu chí tự đặt* |
| sort(a+i, a+j+1, greater<type>()) | *sắp giảm* |
| sort(a+i, a+j+1, Great) | *sắp giảm theo tiêu chí tự đặt* |
| type: int, float, string,...  Less, Great: tự đặt | |

|  |  |
| --- | --- |
| **ID sort array a[i:j]** |  |
| Init: id[i] = i, 0 ≤ i < n |  |
| sort(id+i, id+j, Less) | *sắp tăng* |
| sort(id+i, id+j, Great) | *sắp giảm* |
| Less, Great tự đặt |  |

# Tìm kiếm

### Bài toán tìm kiếm

Cho tập đối tượng S và một phần tử x. Hãy cho biết x ∈ S? Nếu thuộc thì x xuất hiện tại vị trí nào trong S?

Tuỳ theo cấu trúc của tập S mà ta có các thuật toán tìm kiếm khác nhau.

Nếu S là tập không được sắp thứ tự thì ta phải duyệt tuần tự từng phần tử trong S gặp phần tử đầu tiên S(i) = x ta return i, ngược lại, sau khi duyệt hết S mà không tìm được giá trị x trong S ta retturn -1. Thuật toán này gọi là tìm kiếm tuần tự và có độ phức tạp n = len(S).

Nếu S là mảng được sắp thì ta có thể thực hiện tìm kiếm nhị phân với độ phức tạp log(len(S)).

#### Tìm kiếm tuần tự

### Substring

Xâu x được gọi là xâu con của xâu y nếu sau khi xoá đi một số ký tự trong y thì thu được x.

Hãy viết hàm Substr(string x, string y) cho ra số ký tự cần xoá khỏi y để thu được x. Nếu không thể xoá được, hàm cho ra giá trị -1.

Ví dụ

Substr("abc","a12b34bb5c6") = 8;

Substr("abc","abc") = 0;

Substr("abc","a1cb") = -1;

#### Thuật toán

Gọi lenx và leny lần lượt là chiều dái của các string x và y. Xét ba trường hợp

* lenx > leny: bài toán vô nghiệm, return -1;
* lenx = leny: return (x == y) ? 0 : -1;
* lenx < leny: gọi x[i] là ký tự đang chờ xét của x. Duyệt tuần tự các ký tự j của y. Nếu y[j] = x[i] thì ta chuyển qua xét ký tự chờ tiếp theo của x. Nếu x hết ký tự chờ là thành công. Số ký tự cần xoá sẽ là leny-lenx.

#### Độ phức tạp

Cỡ leny phép so sánh.

#### Chương trình

// Substring

#include <bits/stdc++.h>

using namespace std;

int Substr(const string & x, const string & y) {

cout << "\n x = " << x;

cout << "\n y = " << y;

int lenx = x.length(), leny = y.length();

if (lenx > leny) return -1;

if (lenx == leny) return (x == y) ? 0 : -1;

// lenx < leny

int i = 0;

for(int j = 0; j < leny; ++j) {

if (x[i] == y[j]) {

++i;

if (i == lenx) return leny-lenx;

}

}

return -1;

}

int main() {

cout << "\n Result: " << Substr("abc","a12b34bb5c6");

cout << "\n Result: " << Substr("abc","abc");

cout << "\n Result: " << Substr("abc","a1cb3");

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

return 0;

}

#### Output

x = abc

y = a12b34bb5c6

Result: 8

x = abc

y = abc

Result: 0

x = abc

y = a1cb3

Result: -1

T h e E n d .

#### Tìm kiếm nhị phân

Tìm kiếm nhị phân được vận dụng trong tập dữ liệu được sắp S. Muốn tìm phần tử x trog S, ta chia S thành hai phần A và B, S = A|B rồi kiểm tra xem x có thể thuộc phần nào. Nếu x ∈ A hoặc x ∈ B ta lại chia tiếp nửa này thành hai phần và lặp lại bước trước cho đến khi chỉ còn lại một phần tử. Do mỗi lần chia 2 ta bỏ đi một nữa miền tìm kiếm nên độ phức tạp sẽ là cỡ log(len(S)).

### Binary Search

Hàm BinarySearch(a, d, c, x) tìm kiếm một chỉ số i trong mảng sắp tăng a[d..c] thoả điều kiện a[i] = x. Nếu không tìm được hàm cho ra giá trị -1.

Ta chọn điểm giữa mảng a là m = (d+c)/2, như vậy đoạn a[d..c] sẽ được chia thành hai nửa A = a[d..m] và B = a[m+1..c]. Ta xét xem x có trong nửa nào bằng phép kiểm tra a[m] < x?

Nếu a[m] < x thì ta chọn nửa B để xét tiếp; ngược lại ta chọn nửa A. Sau mỗi bước lặp hai đầu d và c sẽ xích lại gần nhau. Khi d = c ta thực hiện phép kiẻm tra cuối cùng a[d] = x? vì rất có thể x không xuất hiện trong a.

int BinarySearch(int a[], int d, int c, int x) {

int m;

while(d < c) {

m = (d+c) / 2; // diem giua

if (a[m] < x) d = m + 1; // lay nua cuoi B

else c = m; // lay nua dau A

}

// d = c

return (a[d] == x) ? d : -1;

}

Dưới đây là phương án đệ quy của hàm Search:

// de quy

int RSearch(int a[], int d, int c, int x) {

if (d == c)

return (a[d] == x) ? d : -1;

// d < c

int m = (d+c)/2; // diem giua

return (x > a[m]) ? RSearch(a, m+1, c, x) // tim trong nua cuoi B

: RSearch(a, d, m, x); // tim trong nua dau A

}

#### Chương trình

// Tim kiem nhi phan

#include <bits/stdc++.h>

using namespace std;

int a[] = {3, 8, 10, 10, 10, 14, 25, 25, 50}; // sap tang

int n = sizeof(a) / sizeof(int);

void Go() {...}

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

int BinarySearch(int a[], int d, int c, int x) {...}

// de quy

int RSearch(int a[], int d, int c, int x) {...}

void Run() {

int d = 0, c = n-1;

int x;

while(true) {

Print(a, d, c, "\n a: ");

cout << "\n (Get x < 0 to stop) x = " ;

fflush(stdin);

cin >> x;

if (x < 0) return;

// cout << " Found " << BinarySearch(a, d, c, x);

cout << " Found " << RSearch(a, d, c, x);

}

}

main() {

Run();

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

return 0;

}

#### *Output*

a: 3 8 10 10 10 14 25 25 50

(Get x < 0 to stop) x = 8

Found 1

a: 3 8 10 10 10 14 25 25 50

(Get x < 0 to stop) x = 10

Found 2

a: 3 8 10 10 10 14 25 25 50

(Get x < 0 to stop) x = 55

Found -1

a: 3 8 10 10 10 14 25 25 50

(Get x < 0 to stop) x = 2

Found -1

a: 3 8 10 10 10 14 25 25 50

(Get x < 0 to stop) x = -1

T h e E n d

Các hàm trên tìm phần tử i đầu tiên trong mảng sắp tăng a thoả a[i] = x. Nếu mảng được sắp tăng trong đó có dãy các phần tử bằng nhau thì ta có thể tìm phần tử cuối dãy thoả a[i] = x.

// Tim chi so i phai nhat a[i] = x

int RightmostSearch(int a[], int d, int c, int x) {

int m;

while(d < c) {

m = (d+c+1) / 2; // diem giua

if (a[m] <= x) d = m; // lay nua cuoi B

else c = m-1; // lay nua dau A

}

// d = c

return (a[d] == x) ? d : -1;

}

// de quy: Tim chi so i phai nhat a[i] = x

int RRightmostSearch(int a[], int d, int c, int x) {

if (d == c) return (a[d] == x) ? d : -1;

// d < c

int m = (d+c+1)/2; // diem giua

return (a[m] <= x) ? RRightmostSearch(a, m, c, x) // lay nua cuoi B

: RRightmostSearch(a, d, m-1, x); // Lay nua dau A

}

## Luyện tập Tìm kiếm

### 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?*

### Sinh nhật

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

### Tom And Jerry

*Giaỉ mã văn bản bí mật sau*

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

*Biết*

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

### Fib

*Bulgaria*

*Dãy số nguyên vô hạn tại hai đầu*

*...,f-2, f-1, f0, f1, f2, ...*

*đựơc gọi là tựa Fibonacci nếu fi = fi-1+fi-2. Biết 5 giá trị i, fi, j, fj, n, i ≠ j. Tính fn ?*

*Ví dụ*

*i* = 0, *f0* = 7, *j* = -5, *f*-5 = -1, *n* = 3. Khi đó *f*3 = 29. Thật vậy, ta có dãy

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *…* | *-5* | *-4* | *-3* | *-2* | *-1* | *0* | *1* | *2* | *3* | *…* |  |
| *fi* |  | -1 | 2 | 1 | 3 | 4 | 7 | 11 | 18 | 29 |  |  |

### Segment Sum

*Kyrgyzstan*

*Cho a là dãy n số nguyên không âm và số nguyên dương d. Cần chọn ít nhất k số liên tiếp nhau trong dãy a để tổng của chúng bằng d. Chỉ số của dãy a được tính từ 0.*

*Ví dụ, n =* 20*, d =* 15*.*

*Ta có đáp số là k = 2, sum(a*[10..11]*) =* 4 + 11 = 15.

|  |  |  |
| --- | --- | --- |
| SUM.INP | OUTPUT | *Giải thích* |
| 20 15  1 1 9 7 0  0 2 12 1 0  4 11 5 8 2  7 2 10 2 3 | 2  10 11 | *k = 2*  *a*[10]+*a*[11] = 15  *Nếu vô nghiệm thì hiển thị k = 0.* |

### Segment Sum by Modulo

Poland

Cho dãy a gồm n số nguyên dương và số nguyên dương d. Cần chọn ít nhất k số liên tiếp trong dãy để tổng của chúng chia hết d. Chỉ số được tính từ 1.

Ví dụ

n = 10, a = [1, 5, 1, 3, 4, 21, 9, 1, 30, 7], d = 10.

Có những đoạn ứng viên sau đây:

a[1..4]: sum(a[1..4]) % 10 = (1+5+1+3) % 10 = 10 % 10 = 0, k = 4

a[6..7]: sum(a[6..7]) % 10 = (21+9) % 10 = 30 % 10 = 0, k = 2

a[7..8]: sum(a[7..8]) % 10 = (9+1) % 10 = 10 % 10 = 0, k = 2

a[9..9] : sum(a[9..9]) % 10 = 30 % 10 = 0, k = 1

Đoạn ngắn nhất là k = 1:

Result: 1

9 9

#### Input

|  |  |
| --- | --- |
| MOD.INP | OUTPUT |
| 10 10  1 5 1 3 4  21 9 1 30 7 | 1  9 9 |