THUẬT TOÁN TÌM KIẾM

1. Giới thiệu bài toán

Việc tìm kiếm là thao tác nền móng cho rất nhiều tác vụ tính toán, tìm kiếm có nghĩa là tìm một hay nhiều mẩu tin từ một số l- ợng lớn thông tin đã đ- ợc l- u trữ. Thông th- ờng thông tin đ- ợc chia thành các mẩu tin (bản ghi), mỗi mẩu tin có một khoá (key) dùng cho việc tìm kiếm. Mục đích của việc tìm kiếm là tìm tất cả các mẩu tin mà khoá của chúng đồng nhất với một khoá đã cho tr- ớc. Sau khi một mẩu tin đã tìm thấy, thông tin bên trong nó sẽ cung cấp cho một quá trình xử lý nào đó.

Việc tìm kiếm đ- ợc áp dụng rất đa dạng và rộng rãi. Ví dụ một nhà băng cần theo dõi tất cả các cân đối tài khoản của các khách hàng và cần phải tìm kiếm để kiểm tra các biến động. Một hệ thống bán vé máy bay cũng có các yêu cầu t- ơng tự.

Ta có thể phát biểu bài toán tìm kiếm như sau: "Cho một bảng gồm n bản ghi $r_1, r_2, ..., r_n$. Mỗi bản ghi r_i t- ơng ứng với một khoá a_i (i =1, 2, ..., n). Hãy tìm bản ghi có giá trị khoá t- ơng ứng bằng x cho trước", x đ- ợc gọi là khoá tìm kiếm và công việc tìm kiếm là hoàn thành khi có 1 trong 2 tình huống sau đây xảy ra:

- Tìm được bản ghi có giá trị khoá t- ơng ứng bằng x. Lúc đó ta nói phép tìm kiếm đ- ợc thoả (Successful).
- Không tìm đ- ợc bản ghi nào có giá trị khoá bằng x cả, phép tìm kiếm không thoả (Unsuccessful). Sau một phép tìm kiếm không thoả có khi xuất hiện các yêu cầu bổ sung thêm bản ghi mới có giá trị khoá bằng x vào bảng. Thuật toán thực hiện cả hai yêu cầu này được gọi là "tìm kiếm có bổ sung".

T- ơng tự nh- bài toán sắp xếp, ta coi các khoá đại diện cho các bản ghi đó và trong các phần tiếp theo ta cũng chỉ nói tới khoá. Để thuận tiện ta coi các khoá a_i là các số nguyên khác nhau. Trong bài này ta chỉ xét các thuật toán tìm kiếm cơ bản và phổ dụng đối với dữ liệu ở bộ nhớ trong, tức là tìm kiếm trong, còn tìm kiếm ngoài ta không đề cập đến.

2. Tìm kiếm tuần tư (sequential searching)

Tìm kiếm tuần tự là kỹ thuật đơn giản và cổ điển. Nội dung của nó là duyệt xuyên qua toàn bộ bảng một cách tuần tự, lần l- ợt so sánh khoá tìm kiếm với khoá t- ơng ứng của các bản ghi trong bảng, cho tới khi tìm đ- ợc bản ghi mong muốn hoặc đã hết bảng mà ch- a tìm thấy.

Trong cách cài đặt sau đây, ta sẽ đ- a thêm khoá tìm vào cuối dãy gọi là phần tử *cầm canh* (hoặc cờ). Khi đó việc tìm kiếm lúc nào cũng thành công. Hàm tìm kiếm sẽ trả lại chỉ số nếu nhỏ hơn hay bằng n thì đó là chỉ số của mẫu tin cần tìm, ng- ợc lại trong bảng đã cho không có mẫu tin nào có giá trị khoá cần tìm:

```
function seqSearch(x : integer) : integer;
var
  i : integer;
begin
  a[n+1] := x;
  for i := 1 to n+1 do
    if a[i] = x then break;
  seqSearch := i;
end;
```

Nhận xét: Với thuật toán trên thuận lợi chỉ cần 1 phép so sánh $C_{min} = 1$, còn xấu nhất thì cần $C_{max} = n+1$ phép so sánh. Nếu khoá tìm kiếm trùng với một khoá nào đó của bảng là đồng khả năng thì $C_{tb} = (n+1)/2$. Tóm lại trong tr- ờng hợp xấu cũng nh- trung bình, cấp độ lớn của thời gian thực hiện là $O(n)$.

3. Tìm kiếm nhị phân (binary searching)

Nếu các khoá đã đ- ợc sắp xếp thứ tự (tăng dần) thì ta có thể giảm tổng số thời gian tìm kiếm bằng cách dựa trên sơ đồ "chia để trị". Thuật toán tìm kiếm nhị phân là một ph- ơng pháp tìm kiếm khá thông dụng dựa trên ý t- ởng đó. Nó t- ơng tự nh- cách ta tra từ trong từ điển. Chỉ có điều hơi khác là trong cách tra từ điển thì ta so sánh khoá tìm kiếm với một từ chọn hú hoạ, còn với việc tìm kiếm nhị phân thì nó luôn chọn khoá ở giữa bảng đang xét để thực hiện so sánh với khoá tìm kiếm.

Giả thiết bảng đang xét là a_l, ..., a_r thì khoá ở giữa bảng là a_i với $i = (l+r)$ *div* 2. Việc tìm sẽ kết thúc nếu $x = a_i$. Nếu $x < a_i$ thì tìm kiếm thực hiện tiếp với a_l, ..., a_{i-1}; còn nếu $x > a_i$ thì tìm kiếm lại đ- ợc làm với a_{i+1}, ..., a_r. Quá trình cứ tiếp tục khi tìm thấy khoá mong muốn hoặc bảng khoá là trở nên rỗng (không tìm thấy).

```
function binarySearch(x : integer) : integer;
var
  l, r, c : integer;
begin
  l := 1; r := n;
  repeat
    c := (l+r) div 2;
    if x < a[c] then r := c-1
                else l := c+1;
  until (a[c] = x) or (l > r);

  if x = a[c] then binarySearch := c
              else binarySearch := n+1;
end;
```

Nhận xét:
Rõ ràng tìm kiếm nhị phân chi phí ít hơn nhiều so với tìm kiếm tuần tự. Ng- ời ta đã chứng minh đ- ợc rằng ph- ơng pháp tìm kiếm nhị phân không bao giờ dùng nhiều hơn $\log_2(n) + 1$ phép so sánh. Trong các ph- ơng pháp tìm kiếm dựa trên so sánh giá trị khoá thì không có ph- ơng pháp nào khác cho đ- ợc kết quả tốt hơn.

Bên cạnh đó ta phải chú ý rằng tr- ớc khi sử dụng ph- ơng pháp tìm kiếm nhị phân thì bảng khoá phải đ- ợc sắp xếp rồi, nghĩa là thời gian chi phí cho việc sắp xếp cũng phải đ- ợc kể đến. Nếu bảng khoá luôn biến động, tức là phép bổ sung và loại bỏ luôn tác động, thì lúc đó chi phí cho việc sắp xếp lại nổi lên rất rõ. Điều này đã bộc lộ nh- ợc điểm của ph- ơng pháp tìm kiếm này.

Thời điểm gặp mặt

File vào MEETING.INP
File ra MEETING.OUT
File ch□ơng trình MEETING.PAS
Giới hạn thời gian 5 giây

Một nhóm gồm n bạn học sinh của một lớp, cùng tham gia câu lạc bộ tin học vào dịp nghỉ hè. Biết rằng khoảng thời gian mà bạn thứ i có mặt tại câu lạc bộ là $[a_i, b_i]$ ($a_i < b_i$ và chúng t- ơng ứng là các thời điểm đến và rời khỏi câu lạc bộ).

Cô giáo chủ nhiệm lớp muốn tới thăm các bạn trong nhóm này. Hãy giúp cô giáo xác định thời điểm đến câu lạc bộ, sao cho tại thời điểm đó cô giáo có thể gặp đ- ợc nhiều bạn trong nhóm nhất.

Dữ liệu: File vào gồm các dòng:
- Dòng đầu tiên ghi số nguyên d- ơng n ($n \le 1000$);
- Dòng thứ i trong số n dòng tiếp theo ghi hai số nguyên không âm a_i, b_i ($i = 1, 2, ..., n$).

Kết quả: File ra gồm các dòng:
- Dòng đầu tiên ghi số nguyên d- ơng k là số bạn đang có mặt ở câu lạc bộ tại thời điểm cô giáo đến;
- Trong k dòng tiếp theo ghi chỉ số của k bạn có mặt ở câu lạc bộ tại thời điểm cô giáo đến, theo thứ tự tăng và mỗi dòng ghi chỉ số của một bạn.

Ví dụ:

MEETING.INP	MEETING.OUT
6 1 2 2 3 2 5 5 7 6 7 9 11	3 1 2 3

MEETING.INP	MEETING.OUT
5 1 2 3 5 7 9 11 15 17 21	1 1

Lỗ thủng

File vào: lothung.in
File ra: lothung.out
File ch□ơng trình: lothung.pas

Giới hạn thời gian: 1 giây
Giới hạn bộ nhớ: 64 KB

Cho một dãy A gồm n số nguyên d- ơng a_1, a_2, ..., a_n. Hãy tìm số nguyên d- ơng nhỏ nhất không là tổng của một số số hạng của dãy A. Tổng này có thể chỉ gồm một số hạng và nếu có nhiều hơn, các số hạng không nhất thiết liên tiếp nhau nh- ng mỗi số hạng của dãy không xuất hiện quá một lần.

Dữ liệu: File vào gồm các dòng:
- Dòng đầu tiên ghi số n ($1 \le n \le 10.000$);
- Các dòng tiếp theo chứa các số a_1, a_2, ..., a_n. Các số cách nhau bởi dấu cách hoặc dấu xuống dòng.

Kết quả: File ra chứa đúng một số nguyên là số nhỏ nhất tìm đ- ợc.

Ví dụ:

lothung.in	lothung.out
4 2 4 8 1	16