CHƯƠNG I
TỔNG QUAN VỀ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Nội dung chính

1.1 Mô hình hóa bài toán thực tế

- Giải quyết bài toán thực tế theo hướng tin học hóa
- Xác định bài toán:
  - Phải làm gì?
  - Phải làm như thế nào?

Ví dụ: Tô màu bản đồ thế giới.

1.2 Cấu trúc dữ liệu

- **Kiểu dữ liệu**: Là một tập hợp các giá trị và một tập hợp các phép toán trên các giá trị đó.
- Kiểu dữ liệu có hai loại:
  - Kiểu dữ liệu sơ cấp:

    Ví dụ: Kiểu Boolean, Integer, character, float….
  - Kiểu dữ liệu có cấu trúc (CTDL): Là kiểu dữ liệu mà giá trị dữ liệu của nó là sự kết hợp của các giá trị khác.

    Ví dụ: Kiểu mảng, kiểu cấu trúc, danh sách liên kết….

Vai trò của việc lựa chọn CTDL

- Kết hợp và đưa ra cách giải quyết cho bài toán.
- Hỗ trợ cho thuật toán thao tác trên các đối tượng được hiệu quả hơn.

Các tiêu chuẩn lựa chọn CTDL

- Phải biểu diễn được đầy đủ thông tin nhập và xuất của bài toán.
- Phải phù hợp với các thao tác của thuật toán mà ta lựa chọn.
- Phù hợp với điều kiện cho phép của NNLT đang sử dụng.
- Tiết kiệm tài nguyên hệ thống.

Phân loại cấu trúc dữ liệu

- **Cấu trúc dữ liệu tuyến tính**: Là cấu trúc dữ liệu trong đó các phần tử được liên kết tuần tự nối tiếp với nhau
- **Cấu trúc dữ liệu dạng cây**: Mỗi phần tử có thể liên kết với nhiều phần tử khác theo từng mức.
- **Cấu trúc dữ liệu bảng băm**: Là một cấu trúc dữ liệu tương tự như mảng nhưng kèm theo một hàm băm để ánh xạ nhiều giá trị vào cùng một phần tử trong mảng.
- **Cấu trúc dữ liệu dạng đồ thị**: Là dạng cấu trúc dữ liệu bao gồm một tập các đối tượng được gọi là các đỉnh (hoặc nút) nối với nhau bởi các cạnh (hoặc cung).

1.3 Giải thuật

- **Tại sao sử dụng máy tính để xử lý dữ liệu?**
  - Nhanh hơn.
  - Nhiều hơn.
  - Giải quyết những bài toán mà con người không thể hoàn thành được.
- **Làm sao đạt được những mục tiêu đó?**
  - Nhờ vào sự tiến bộ của kỹ thuật: tăng cấu hình máy ⇨ chi phí cao ☹
  - Nhờ vào các thuật toán hiệu quả: thông minh và chi phí thấp ☺

"Một máy tính siêu hạng vẫn không thể cứu vãn một thuật toán tồi !"

Khái niệm thuật toán

Giải thuật hay thuật toán là một chuỗi hữu hạn các thao tác để giải một bài toán nào đó.

Tính chất của thuật toán

- **Hữu hạn**: Giải thuật phải luôn luôn kết thúc sau một số hữu hạn bước.
- **Xác định**: Mỗi bước của giải thuật phải được xác định rõ ràng và phải được thực hiện chính xác, nhất quán.
- **Đúng**: Giải thuật phải đảm bảo tính đúng và chính xác;
- **Hiệu quả** : Các thao tác trong giải thuật phải được thực hiện trong một lượng thời gian hữu hạn.
- …

Ngoài ra thuật toán còn phải có dữ liệu đầu vào và đầu ra

Biểu diễn giải thuật

➢ **Đặc tả không hình thức (Unformal specification)**

Sử dụng ngôn ngữ tự nhiên: Liệt kê tuần tự các bước để giải quyết bài toán => dài dòng và đôi khi khó hiểu.

➢ **Đặc tả hình thức (Formal specification)**

- **Sử dụng lưu đồ** (flowchart): => trực quan và dễ hiểu , tuy nhiên hơi cồng kềnh.
- **Sử dụng mã giả**: => đỡ cồng kềnh, tuy nhiên không trực quan.
- **Sử dụng ngôn ngữ đặc tả**: => đòi hỏi phải có kiến thức về ngôn ngữ được sử dụng.

Đánh giá độ phức tạp của thuật toán

Thời gian chạy một chương trình phụ thuộc vào các yếu tố sau:

- Khối lượng của dữ liệu đầu vào.
- Chất lượng của mã máy được tạo ra bởi trình dịch.
- Tốc độ thực thi lệnh của máy.
- Độ phức tạp về thời gian của thuật toán

Một thuật toán được gọi là hiệu quả nếu **chi phí cần sử dụng tài nguyên của máy là thấp** .

Đánh giá độ phức tạp của thuật toán

Để mô tả việc đánh giá độ phức tạp của thuật toán người ta sử dụng một hàm f(N), trong đó N là khối lượng dữ liệu cần được xử lý.

Có hai phương pháp để đánh giá độ phức tạp của thuật toán gồm:

- Phương pháp thực nghiệm:
- Phương pháp xấp xỉ toán học:

Phương pháp thực nghiệm

- Cài thuật toán rồi chọn các bộ dữ liệu thử nghiệm, thống kê các thông số nhận được khi chạy các bộ dữ liệu đó.
- *Ưu điểm*: Dễ thực hiện.
- *Nhược điểm*:
  - Chịu sự hạn chế của ngôn ngữ lập trình.
  - Ảnh hưởng bởi trình độ của người lập trình.
  - Chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán: khó khăn và tốn nhiều chi phí.
  - Phụ thuộc vào phần cứng.

Phương pháp xấp xỉ toán học

- Đánh giá giá thuật toán theo hướng tiệm xấp xỉ tiệm cận qua các khái niệm O().
- *Ưu điểm*: Ít phụ thuộc môi trường cũng như phần cứng hơn.
- *Nhược điểm*: Phức tạp.
- Các trường hợp độ phức tạp quan tâm:
  - Trường hợp tốt nhất (phân tích chính xác)
  - Trường hợp xấu nhất (phân tích chính xác)
  - Trường hợp trung bình (mang tính dự đoán)

Ví dụ về O lớn (Big O)

- Xét độ phức tạp của đoạn code sau

```
S=0;
for (int i=0; i<n; i++){
    S=S+i*i;
}
cout<< S;
```

- Số thao tác cơ bản:
  - 1 phép gán ngoài vòng lập.
  - n bước lắp mỗi bước gồm 3 thao tác cơ bản (nhân, cộng và gán)
  - Độ phức tạp = 3n+1 = O(n)

Ví dụ về O lớn (Big O)

- Xét độ phức tạp của đoạn code sau

```
s = 0;
for (i=0; i<=n;i++){
        p =1;
        for (j=1;j<=i;j++)
                p=p * x / j;
        s = s+p;
}
```

- 1 phép gán s=0
- n+1 phép gán p=1 và s=s+p
- 0+1+2+…+n =n(n-1)/2=n^2/2-n/2 phép toán p=p*x/j

= O(3n+4+(n^2/2-n/2)*3)=O(n^2)

Phân lớp độ phức tạp của GT

- **Sử dụng ký hiệu BigO**
  - Hằng số : O(c)
  - logN : O(logN)
  - N : O(N)
  - NlogN : O(NlogN)
  - N^2 : O(N^2)
  - N^3 : O(N^3)
  - 2^N : O(2^N)
  - N! :O(N!)

Độ phức tạp tăng dần

1.4 Chương trình

- Theo **Niklaus Wirth**:

  Cấu trúc dữ liệu + Thuật toán = Chương trình

Tiêu chuẩn của một chương trình

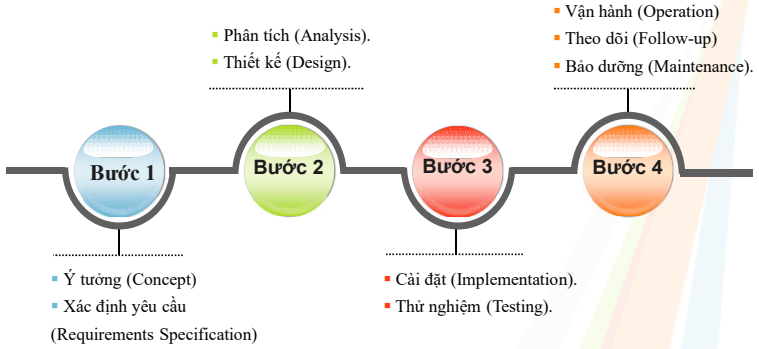
Tính tin cậy: Chạy đúng như dự định, mô tả chính xác một giải thuật đúng.

Tính uyển chuyển: Dễ sửa đổi, giảm bớt công sức của lập trình viên khi phát triển chương trình, đáp ứng các quy trình làm phần mềm.

Tính trong sáng: Chương trình viết ra phải dễ đọc, dễ hiểu.

Tính hữu hiệu: Chạy nhanh và ít tốn tài nguyên bộ nhớ

Quy trình làm phần mềm



Bài tập

Trình bày giải thuật giải các bài toán sau:

1. Giải phương trình bậc hai: $ax^2+bx+c = 0$.
2. Tìm số lớn nhất trong 4 số nguyên.
3. Tính tổng $s=1+2+3+\ldots+n$.
4. Tìm số lớn nhất trong n số (n nguyên dương).
5. Tìm bội chung nhỏ nhất của 2 số nguyên dương.
6. Tìm ước chung lớn nhất của 2 số nguyên dương.
7. Tìm phần tử thứ n của dãy Fibonacci.

Chương II
GIẢI THUẬT TÌM KIẾM

Nội dung chính

- Bài toán tìm kiếm
- Tìm kiếm tuyến tính
- Tìm kiếm nhị phân
- Đánh giá giải thuật tìm kiếm

2.1 Bài toán tìm kiếm

Cho một dãy gồm n bản ghi r[1.. n]. Mỗi bản ghi r[i] tương ứng với một khóa r[i].k Hãy tìm bản ghi có khóa X cho trước.

X được gọi là khóa tìm kiếm hay đối trị tìm kiếm (argument).

Bài toán tìm kiếm

Công việc tìm kiếm sẽ hoàn thành nếu như có một trong hai tình huống sau sảy ra:

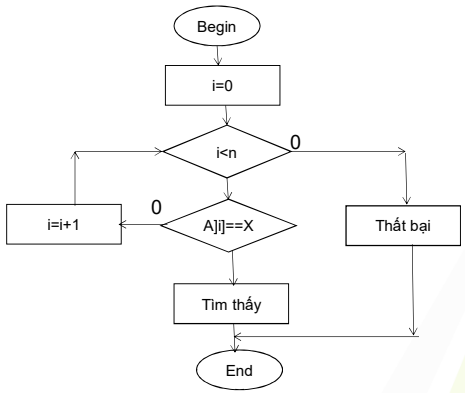
- *Tìm được bản ghi có khóa tương ứng bằng X, lúc đó phép tìm kiếm **thành công**;*
- *Không tìm được bản ghi nào có khóa bằng X cả, phép tìm kiếm **thất bại**.*

Để đơn giản cho việc trình bày giải thuật thì bài toán có thể phát biểu như sau: Cho mảng A[n], hãy tìm trong A phần tử có khóa bằng X.

2.2 Tìm tuần tự (Sequential/Linear search)

Ý tưởng : So sánh X lần lượt với phần tử thứ 1, thứ 2,…của mảng A cho đến khi gặp được khóa cần tìm (tìm thấy), hoặc tìm hết mảng mà không thấy (không tìm thấy).

Tìm kiếm tuần tự



Cài đặt giải thuật

```
int SequentialSearch(int A[], int n, int x)
{
        int i=0;
        while(i<n && A[i]!=x)
              i++;
        if(i==n)
              return 0; //Tìm không thấy x
        else
              return 1; //Tìm thấy
}
```


Cài đặt giải thuật

Nhận xét: Số phép so sánh của thuật toán trong trường hợp xấu nhất là 2*n. Để giảm thiểu số phép so sánh trong vòng lặp cho thuật toán, ta thêm phần tử "lính canh" vào cuối dãy.

```
int SequentialSearch(int A[], int n, int x)
{
    int i=0, A[n]=x;//A[n] là phần tử lính canh
    while(A[i]!=x)
          i++;
    if(i==n) return 0; //Tìm không thấy x
    else return 1; //Tìm thấy
}
```

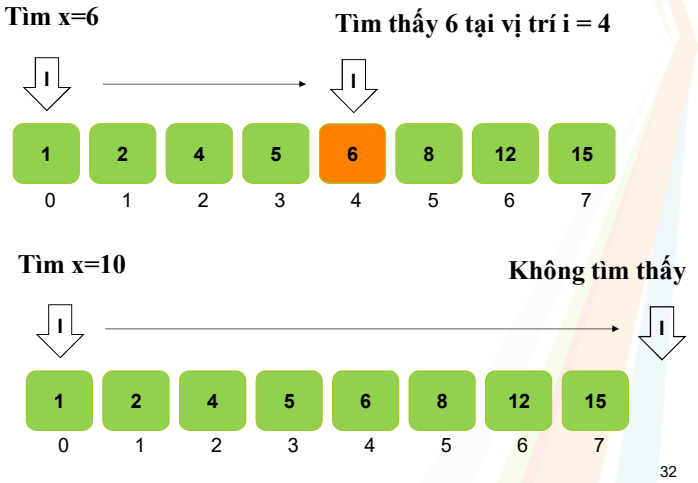

Cài đặt giải thuật

Cài đặt giải thuật bằng for:

```
int SequentialSearch(int A[], int n, int x)
{
    for(int i=0; i<n; i++)
          if(A[i]==x) return 1;
    return 0;
}
```

Ghi chú: *Tùy vào yêu cầu của bài toán mà ta có thể áp dụng giải thuật tìm kiếm tuyến tính một cách linh hoạt và phù hợp hơn.*

Minh họa tìm kiếm tuần tự



2.3 Tìm nhị phân (binary search)

Giải thuật tìm kiếm nhị phân được áp dụng trên mảng đã sắp xếp thứ tự. Giả sử mảng A[n] được sắp xếp tăng dần, khi đó ta có:

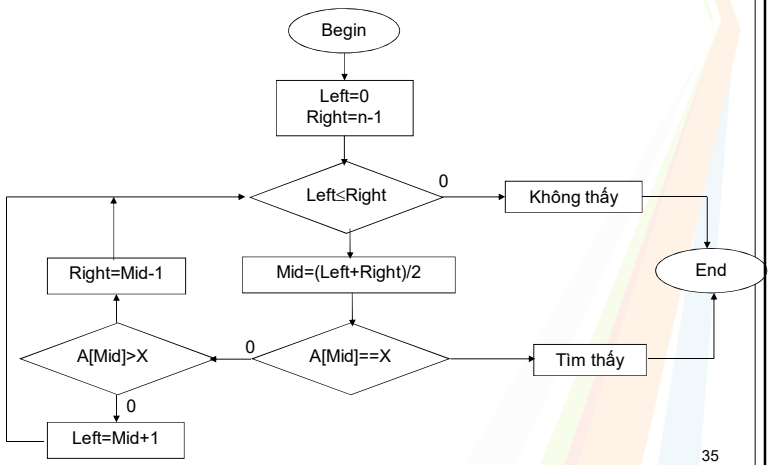
$$A[i-1] < A[i] < A[i+1]$$

Như vậy nếu X>A[i] thì X chỉ có thể nằm trong đoạn [A[i+1], A[n-1]], còn nếu X<A[i] thì X chỉ có thể nằm trong đoạn [A[0], A[i-1]].

Tìm nhị phân

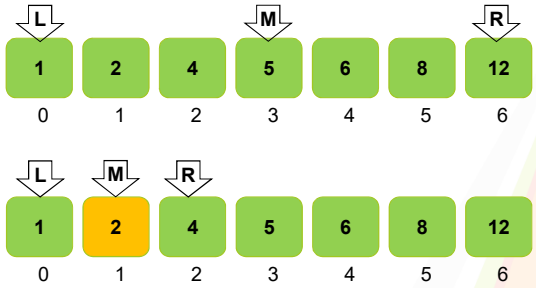
Ý tưởng của giải thuật là tại mỗi bước ta so sánh X với phần tử đứng giữa trong dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này mà ta quyết định giới hạn dãy tìm kiếm ở nửa trước hay nửa sau của dãy tìm kiếm hiện hành.

Tìm nhị phân



Minh họa thuật toán

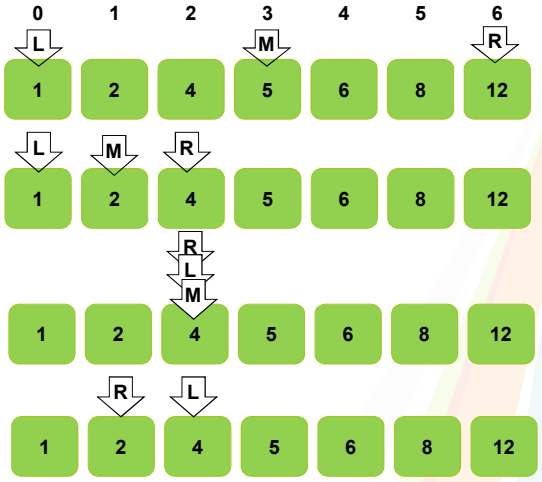
Tìm x=2



Tìm thấy tại vị trí 1

Minh họa thuật toán

Tìm x=3



Không tìm thấy

Cài đặt giải thuật

```
int BinarySearch(int a[], int n, int x){
    int mid, left=0, right=n-1;
    do {
          mid=(left+right)/2;
          if(a[mid]==x) return 1;
          else
                if(a[mid]<x)  left = mid+1;
                else  right = mid-1;
    } while(left<=right);
    return 0;
}
```


2.4 Đánh giá các giải thuật

Giải thuật tìm kiếm	Trường hợp tốt nhất	Trường hợp trung bình	Trường hợp xấu nhất
Tìm kiếm tuyến tính	O(1)	O((N+1)/2)	O(N)
Tìm kiếm nhị phân	O(1)	$O(\log_2 N/2)$	$O(\log_2 N)$

Bài tập

Bài 1. Viết lại hàm tìm kiếm nhị phân bằng đệ quy.

Bài 2. Viết chương trình nhập dữ liệu cho mảng số nguyên A[n], với 0<n<100. Hãy tìm trong A các phần tử là số lẻ và lưu vào mảng B.

Bài 3. Nhập dữ liệu và sắp xếp mảng A[n] tăng dần. Sử dụng giải thuật tìm kiếm nhị phân để tìm phần tử có giá trị bằng X ở trong mảng A sau đó xóa nó khỏi A nếu tìm thấy.

Bài 4. Nhập vào một chuỗi S bất kì. Đếm xem trong chuỗi S có bao nhiêu kí tự khoảng trống, bao nhiêu kí tự số, bao nhiêu kí tự là chữ cái in hoa ?


Thank You!
41