MỤC LỤC

[**LỜI NÓI ĐẦU** 1](#_Toc321598959)

[**1.** **Kiểm thử** 3](#_Toc321598960)

[ **Định nghĩa:** 3](#_Toc321598961)

[ **Các chú ý để thiết kế dữ liệu kiểm thử tốt:** 3](#_Toc321598962)

[ **Ví dụ:** 3](#_Toc321598963)

[**2.** **Chứng minh trực tiếp** 6](#_Toc321598964)

[**2.1.** **Phương pháp quy nạp:** 6](#_Toc321598965)

[ **Nguyên tắc:** 6](#_Toc321598966)

[ **Chú ý:** 6](#_Toc321598967)

[ **Ví dụ:** 6](#_Toc321598968)

[**2.2.** **Phương pháp bất biến vòng lặp:** 9](#_Toc321598969)

[ **Định nghĩa:** 9](#_Toc321598970)

[ **Sơ đồ minh họa bất biến vòng lặp:** 9](#_Toc321598971)

[ **Đặc trưng của bất biến vòng lặp:** 10](#_Toc321598972)

[ **Chú ý:** 10](#_Toc321598973)

[ **Các bước chứng minh tính đúng của thuật toán bằng bất biến vòng lặp:** 10](#_Toc321598974)

[ **Ví dụ:** 11](#_Toc321598975)

[**2.3.** **So sánh quy nạp và bất biến vòng lặp** 16](#_Toc321598976)

[ **Giống nhau:** 16](#_Toc321598977)

[ **Khác nhau:** 16](#_Toc321598978)

[**3.** **Kết hợp các phương pháp:** 17](#_Toc321598979)

[**Tài liệu tham khảo:** 20](#_Toc321598980)

# LỜI NÓI ĐẦU

Trong lý thuyết về thuật toán, chứng minh tính đúng của thuật toán đóng vai trò rất quan trọng.Nó trả lời cho câu hỏi quyết định là thuật toán đã cho đúng hay sai.Chính vì vậy nhóm I đã chọn đề tài “Các phương pháp chứng minh tính đúng của thuật toán” cho bài tiểu luận của mình.

Có 3 phương pháp cơ bản:

* Phương pháp kiểm thử
* Hai phương pháp chứng minh trực tiếp gồm có: quy nạp và bất biến vòng lặp.

Mỗi phương pháp được trình bày thành các phần: định nghĩa, cách tiến hành và ví dụ minh họa.Trong phần chứng minh trực tiếp có thêm phần so sánh 2 phương pháp quy nạp và bất biến vòng lặp.Phần cuối bàn về cách kết hợp các phương pháp với nhau để chứng minh tính đúng của thuật toán.

1. **Kiểm thử**

* **Định nghĩa:**
* Kiểm thử là ta chạy thử chương trình với các dữ liệu cụ thể. Nếu chương trình chạy đúng, ta khẳng định với xác suất P là thuật toán đúng.
* Xác suất khẳng định P càng cao nếu dữ liệu kiểm thử bao quát các tình huống xảy ra càng nhiều.
* Trong nhiều trường hợp khi chương trình quá lớn không thể chứng minh tính đúng một cách trực tiếp thì phải sử dụng phương pháp kiểm thử.
* **Các chú ý để thiết kế dữ liệu kiểm thử tốt:**
* Dữ liệu kiểm thử phải đủ nhỏ để chương trình có thể dừng sau một thời gian hữu hạn;
* Dữ liệu kiểm thử bao quát được càng nhiều trường hợp càng tốt.
* **Ví dụ:**

*Giải phương trình bậc 2*

Mã chương trình (Viết bằng ngôn ngữ C)

#include<iostream.h>

#include<conio.h>

#include<math.h>

float a,b,c,x1,x2,delta;

int main(){

cout<<endl<<"Chuong trinh giai phuong trinh bac hai ax^2+bx+c=0";

cout<<endl<<"Nhap cac he so:";

cout<<endl<<"a = ";

cin>>a;

cout<<endl<<"b = ";

cin>>b;

cout<<endl<<"c = ";

cin>>c;

cout<<"Phuong trinh da cho la:";

cout<<a<<"\*x^2 + "<<b<<"\*x + "<<c<<" = 0";

if (a==0){

if (b==0)

if (c==0) cout <<endl<<"Phuong trinh da cho co vo so nghiem";

else cout <<endl<<"Phuong trinh da cho vo nghiem ";

else{

x1 = -c/b;

cout<<endl<<"Phuong trinh da cho co nghiem x = "<<x1;

}

}

else{

delta = b\*b-4\*a\*c;

if (delta<0) cout<<endl<<"Phuong trinh da cho vo nghiem";

if (delta==0){

x1 = -b/(2\*a);

cout<<endl<<"Phuong trinh da cho co nghiem kep x1 = x2 = "<<x1;

}

if (delta>0){

x1=(sqrt(delta)-b)/(2\*a);

x2=(-sqrt(delta)-b)/(2\*a);

cout<<endl<<"Phuong trinh da cho co hai nghiem phan biet: x1 = "<<x1<<", x2 = "<<x2;

}

}

getch();

return 0;

}

Ta thiết kế dữ liệu kiểm thử với 6 dữ liệu đầu vào:

|  |  |  |
| --- | --- | --- |
| **STT** | **Dữ liệu kiểm thử** | **Kết quả** |
| 1 |  | Phuong trinh da cho vo nghiem |
| 2 |  | Phuong trinh da cho co nghiem kep |
| 3 |  | Phuong trinh da cho co hai nghiem phan biet: |
| 4 |  | Phuong trinh da cho co vo so nghiem |
| 5 |  | Phuong trinh da cho co nghiem |
| 6 |  | Phuong trinh da cho vo nghiem |

Ta thấy các dữ liệu kiểm thử trên đã bao quát hết các trường hợp có thể xảy ra.Cả 6 trường hợp, thuật toán đều cho ra kết quả đúng chứng tỏ thuật toán là đúng.

Bên cạnh đó, có những thuật toán mà đầu vào bao gồm nhiều loại dữ liệu đến nỗi ta không thể thiết kế hết các dữ liệu kiểm thử để kiểm tra.Điều này dẫn đến nhu cầu phải chứng minh thuật toán đó đúng bằng các phương pháp toán học chặt chẽ. Phần tiếp theo của tiểu luận sẽ trình bày về các phương pháp này.

1. **Chứng minh trực tiếp**
   1. **Phương pháp quy nạp:**
   * **Nguyên tắc:**

* Nguyên tắc chung: Chứng minh tính đúng của thuật toán theo kích cỡ dữ liệu đầu vào ( ví dụ như số phần tử trong mảng. số bít trong số nguyên, số chiều của ma trận v…v..
* Phương pháp quy nạp thường được sử dụng trong việc chứng minh thuật toán **c**ó dạng đệ quy.
* Các bước chứng minh tính đúng của thuật toán bằng phương pháp quy nạp :

+) Chứng minh được cơ sở của phép quy nạp (tương ứng với trường hợp suy biến của phép đệ quy) là đúng

+) Giả thiết quy nạp:”Giả sử thuật toán đúng với kích cỡ dữ liệu đầu vào (input) có kích thước là n”. Với giả thiết này, ta chứng minh thuật toán đúng với kích thước dữ liệu đầu vào là .

* + **Chú ý:**
* Trong giả thiết quy nạp thêm điều kiện n không vượt quá kích thước của trường hợp cơ sở.
  + **Ví dụ****:**
* **Ví dụ 2.1.1**: Chứng minh thuật toán tính giai thừa:

Mã giả

*Mã giả*

*int factorial (int n)//n:số cần tính*

*Begin*

*if(n=0) then return 1;*

*endif.*

*else*

*Begin.*

*return (n\*factorial(n-1));*

*End.*

*End.*

*End.*

Chứng minh thuật toán trả lại giá trị đúng bằng

.

Thật vậy, dựa trên theo nguyên tắc trên ta có:

Cơ sở: thì luôn đúng

Giả thiết quy nạp:Giả sử trả lại giá trị của

Khi đó, trả lại giá trị là:

Vậy thuật toán trên là đúng.

* **Ví dụ 2.1.2:** Chứng minh thuật toán tính số fibonacci thứ :

*Mã giả*

int Fibonacci (int n)//n:số cần tính

Begin

if(n<=1) then return 1;

endif.

else

Begin.

return (Fibonacci(n-1)+Fibonacci(n-2))

End.

End.

End.

*End.*

Chứng minh thuật toán trên trả lại giá trị của số fibonacci thứ n.

Áp dụng nguyên tắc trên ta có:

Cơ sở: thì

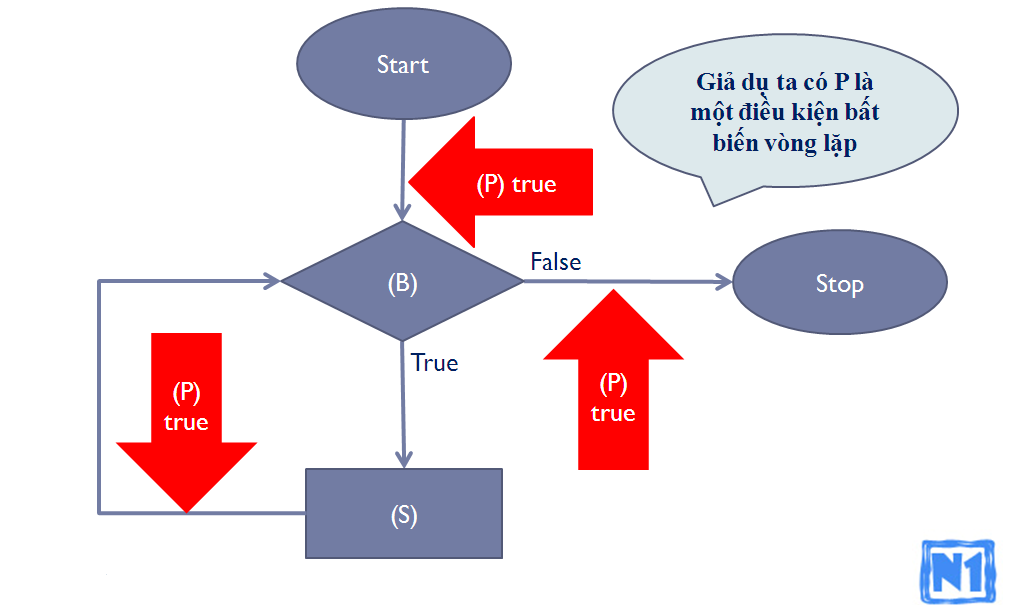
Giả thiết quy nạp: Giả sử trả lại giá trị của số fibonacci thứ

Vậy:

là giá trị của số fibonacci thứ

* 1. **Phương pháp bất biến vòng lặp:**
  + **Định nghĩa:**
* Bất biến vòng lặp là một mệnh đề lôgíc (của các biến được sử dụng trong vòng lặp) đúngtrước khi vào vòng lặp, sau mỗi bước lặp và khi kết thúc vòng lặp.
  + **Sơ đồ minh họa bất biến vòng lặp:**
* Giả sử ta có 1 vòng lặp while như sau. (Các vòng lặp khác có thể xét tương tự)
* While (B)
* { S }
* Trong đó:
  + B là điều kiện lặp.
  + S là các lệnh trong vòng lặp.

Bây giờ ta minh họa bất biến vòng lặp bằng sơ đồ sau:



Từ hình vẽ ta có, bất biến vòng lặp (P) phải đúng trước lần lặp đầu tiên, trong vòng lặp và sau khi kết thúc vòng lặp.Do vậy bất biến vòng lặp sẽ có những đặc trưng cơ bản được nêu sau đây.

* + **Đặc trưng của bất biến vòng lặp:**
* Bất biến vòng lặp có những đặc trưng cơ bản sau:
* Khởi tạo: bất biến của vòng lặp phải đúng trước lần lặp đầu tiên.
* Duy trì: Nếu nó đúng trước một vòng lặp, nó vẫn còn đúng trước vòng lặp tiếp theo.
* Kết thúc: Khi vòng lặp kết thúc, bất biến này cho chúng ta một tính chất hữu ích giúp chứng minh được thuật toán là đúng đắn.
  + **Chú ý:**
* Đối với những thuật toán có nhiều vòng lặp lồng nhau ta phải bắt đầu chứng minh tính đúng từ vòng lặp trong ra vòng lặp ngoài vì nếu các vòng lặp trong không đúng ta không thể chứng minh các vòng lặp ngoài đúng được.Do vậy, thuật toán của chúng ta không thể đúng được.
* Khi sử dụng bất biến vòng lặp phải chỉ ra thuật toán dừng và cho chúng ta một kết quả tính toán đầu ra (output).
  + **Các bước chứng minh tính đúng của thuật toán bằng bất biến vòng lặp:**
* Xác định bất biến vòng lặp
* Chứng minh ba đặc trưng của bất biến vòng lặp:
* Chứng minh khởi tạo đúng
* Chứng minh duy trì đúng
* Chứng minh kết thúc đúng sau hữu hạn lần lặp

Để nắm rõ phương pháp bất biến vòng lặp ta đi sâu thêm một số ví dụ cơ bản sau đây:

* + **Ví dụ:**
* **Ví dụ 2.2.1:** Chứng minh thuật toán tìm kiếm một phần tử bằng v trên một mảng số nguyên:

*Mã giả*

int findElement(int a[ ],int n)// n:số phần tử. a[1…n ] là mảng số nguyên

Begin

index = 0;

i = 1;

while (index = 0) and (i ≤ n)

Begin

if (v = a[i])

then index = i;

i = i + 1;

endif.

endwhile

return index;

End

**Bất biến vòng lặp:**

Sau bước lặp thứ i thì:

Bất biến trên thỏa mãn 3 tính chất:

**Khởi tạo:**Trước khi vào vòng lặp, coi như i = 0, do {a[1], … a[i]} =Ø nên index = 0 thỏa mãn tính bất biến vì v Ø

**Duy trì:**Sau bước lặp thứ i, nếu còn thực hiện tiếp bước lặp thứ i + 1, chứng tỏ là sau bước lặp thứ i (vì nếu khác đi thì vòng lặp dừng lại luôn):



Xét 2 trường hợp xảy ra sau bước lặp thứ i:

Nếu a[i+1] ≠ v như vậy index = 0 và rõ ràng v {a[1], a[2], … a[i], a[i+1]}

Nếu a[i+1] = v thì index = i + 1

Vậy bất biến của vòng lặp có tính duy trì.

**Kết thúc:**

Sau bước lặp thứ i, nếu vòng lặp dừng lại, ta có 2 trường hợp

Nếu index = 0 thì v { a[1], a[2], … a[i]}, do vòng lặp dừng lại nên i = n, chứng tỏ v {a[1], a[2], … a[n]}, như vậy ta khẳng định được trong dãy n số (a[1], a[2], … a[n]) không có số nào bằng v

Nếu index = i thì v = a[i] do vòng lặp dừng lại và ta đưa ra được chỉ số ai = v.

Bất biến của vòng lặp thỏa mãn đủ 3 tính chất cần thiết, chứng tỏ thuật toán của ta là đúng đắn

* **Ví dụ 2.2.2:**Chứng minh tính đúng của thuật toán bubblesort

procedure bubblesort(A[1..n])

comment Sort A[1], A[2], ... , A[n] into nondecreasing order

1. for i := 1 to n – 1 do

2. for j := 1 to n – i do

3. if A[j] > A[j + 1] then

4. Swap A[j] with A[j + 1]

**Bất biến vòng lặp đối với vòng lặp trong**

**Mệnh đề**

Trước khi bắt đầu lần lặp thứ j, A[j] lưu phần tử lớn nhất trong A[1],A[2],…,A[j].

**Khởi tạo**

Trước lần lặp thứ nhất, j = 1, nên A[1] lưu phần tử lớn nhất trong A[1].

**Duy trì**

Trước lần lặp thứ j, A[j] lưu phần tử lớn nhất trong A[1],A[2],…,A[j]. Sau khi chạy qua bước 3-4, nếu A[j] mà lớn hơn A[j+1] thì đổi chỗ nó cho A[j+1], như vậy sau vòng lặp thứ j, ta có A[j +1] chứa phần tử lớn nhất trong A[1],A[2],…,A[j+1].

**Kết thúc**

Khi thoát khỏi vòng lặp, j = n-i, A[n-i+1] lưu phần tử lớn nhất trong A[1],A[2],…,A[n-i+1].

**Bất biến vòng lặp đối với vòng lặp ngoài**

**Mệnh đề**

Trước khi bắt đầu lần lặp thứ i, dãy A[n-i+2],A[n-i+3],…,A[n] sắp xếp theo thứ tự không giảm và các số A[1],A[2],…,A[n-i+1] đều không lớn hơn A[n-i+2].

**Khởi tạo**

Trước lần lặp thứ nhất, i = 1, dãy A[n-i+2],A[n-i+3],…,A[n] rỗng nên ta coi như nó thỏa mãn mệnh đề.

**Duy trì**

Trước lần lặp thứ i, dãy A[n-i+2],A[n-i+3],…A[n] sắp xếp theo thứ tự không giảm và các số A[1],A[2],…,A[n-i+1] đều không lớn hơn A[n-i+2]. Sau khi chạy qua 2-4, theo bất biến của vòng lặp trong ta có A[n-i+1] lưu phần tử lớn nhất trong A[1],A[2],…,A[n-i+1], vòng lặp trong này không đả động gì đến các phần tử A[n-i+2],A[n-i+3],…,A[n], do đó:

Do trước vòng lặp A[1],A[2],…,A[n-i+1] đều không vượt quá A[n-i+2], do đó sau vòng lặp A[n-i+1] cũng không vượt quá A[n-i+2], kết quả là dãy A[n-i+1], A[n-i+2],A[n-i+3],…A[n] sắp xếp theo thứ tự không giảm;

A[n-i+1] đã lưu phần tử lớn nhất trong A[1],A[2],…,A[n-i+1] do đó không có phần từ nào trong A[1],A[2],…,A[n-i] lớn hơn A[n-i+1].

Chứng tỏ vòng lặp có tính duy trì.

**Kết thúc**

Khi thoát khỏi vòng lặp, i = n, theo bất biến trên:

A[2],A[3],…,A[n] sắp theo thứ tự không giảm;

A[1] không lớn hơn A[2];

Từ hai điều trên, ta kết luận là dãy A[1],A[2],A[3],…,A[n] đã được sắp theo thứ tự không giảm. Chứng tỏ thuật toán đã cho là đúng.

* 1. **So sánh quy nạp và bất biến vòng lặp**
  + **Giống nhau:**
* Về cơ bản các bước khởi tạo, duy trì, kết thúc của phương pháp bất biến vòng lặp cũng có điểm tương đồng trong việc chứng minh với các bước cơ sở quy nạp, giả thiết quy nạp, tổng quát của phương pháp quy nạp.
  + **Khác nhau:**

|  |  |
| --- | --- |
| Quy nạp: | Bất biến vòng lặp: |
| - Mệnh đề quy nạp luôn là thuật toán đã cho đúng với dữ liệu đầu vào với kích thước n.  - Nếu ta kí hiệu thuật toán với kích thước dữ liệu đầu vào n là A(n). Phương pháp quy nạp sẽ đi chứng minh thuật toán A đúng với mỗi giá trị của n=1,2,..,k,k+1:  A(1),A(2),…,A(k),A(k+1),… . | -Mệnh đề của bất biến vòng lặp thường là một tính chất đúng trong trước, trong và sau vòng lặp và từ tính chất này ta suy ra tính đúng của thuật toán.  -Đi sâu từng bước của vòng lặp trong thuật toán A để chứng minh thuật toán đúng. |

1. **Kết hợp các phương pháp:**

Trên thực tế, có những thuật toán để chứng minh tính đúng ta không thể chỉ dùng riêng lẻ một phương pháp mà cần phải kết hợp nhiều phương pháp với nhau.

Ví dụ thuật toánA có hai loại dữ liệu là L1 và L2 mà:

* Ta chỉ có thể chứng minh thuật toán đúng với L1 bằng phương pháp chứng minh trực tiếp (vì L1 bao gồm quá nhiều trường hợp).
* còn với L2, không thể chứng minh trực tiếp được mà phải kiểm tra bằng dữ liệu kiểm thử.

Như vậy, với thuật toán A ta phải sử dụng cả hai phương pháp chứng minh trực tiếp và kiểm thử.

Mặt khác, kể cả khi ta đã chứng minh thuật toán đúng về mặt lí thuyết, vẫn đòi hỏi phải sử dụng các dữ liệu kiểm thử để tăng thêm độ tin cậy về tính đúng của thuật toán.

Thông thường, kể cả khi quyết định dùng phương pháp chứng minh trực tiếp, ta sử dụng kiểm thử trước khi sử dụng các công cụ toán học.

Bên cạnh đó, ngay cả với phương pháp trực tiếp, có những thuật toán đòi hỏi phải kết hợp cả quy nạp với bất biến vòng lặp.

Ta xét ví dụ về thuật toán nổi bọt có dạng đệ quy chứa vòng lặp

1. procedure bubblesort(n)
2. comment Sort A[1…n]
3. if(n>1) then
4. for j:=1 to n-1 do
5. if A[j]>A[j+1] then
6. swap A[j] with A[j+1]
7. bubblesort(n-1)

Ta sẽ chứng minh thuật toán trên bằng việc kết hợp 2 phương pháp bất biến vòng lặp và quy nạp. Lược đồ chung là sử dụng quy nạp, trong bước giả thiết quy nạp lại sử dụng bất biến vòng lặp.

**Cơ sở:**

n=1:dãy chỉ có một phần tử A[1] đã được sắp xếp

**Giả thiết:**

Giả sử câu lệnh 5.bubblesort(n-1) là đúng với n>1 tức là dãy A[1],A[2],…,A[n-1] đã được sắp xếp.Ta phải chứng minh với n-1+1 =n (n>1)phần tử đầu vào dãy A[1],A[2],…,A[n] được sắp xếp(\*).Thật vậy, ta có bất biến vòng lặp cho vòng lặp for là:

*Mệnh đề*

Trước khi bắt đầu lần lặp thứ j, A[j] lưu phần tử lớn nhất trong A[1],A[2],…,A[j].

*Khởi tạo*

Trước lần lặp thứ nhất, j = 1, nên A[1] lưu phần tử lớn nhất trong A[1].

*Duy trì*

Trước lần lặp thứ j, A[j] lưu phần tử lớn nhất trong A[1],A[2],…,A[j]. Sau khi chạy qua bước 3-4, nếu A[j] mà lớn hơn A[j+1] thì đổi chỗ nó cho A[j+1], như vậy sau vòng lặp thứ j, ta có A[j +1] chứa phần tử lớn nhất trong A[1],A[2],…,A[j+1].

*Kết thúc*

Khi thoát khỏi vòng lặp, j = n, A[n] lưu phần tử lớn nhất trong A[1],A[2],…,A[n].

Kết hợp với (\*) ta thu được dãy A[1],A[2],…,A[n] đã được sắp xếp.

Vậy thuật toán trên đúng.

# Tài liệu tham khảo:

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms (Second Edition) McGraw Hill - MIT Press © 2001 (ISBN:0262032937)

[2] Ian Parberry and William Gasarch, Problems on Algorithms, ebooks 2002

[3] Slide lecture2 - Algorithm Analysis

[4] Giáo trình một số vấn đề thuật toán – Nguyễn Hữu Điền