Cấu trúc dữ liệu

Chương 1 - Mở đầu (Đọc thêm)

Bộ môn Công Nghệ Phần Mềm

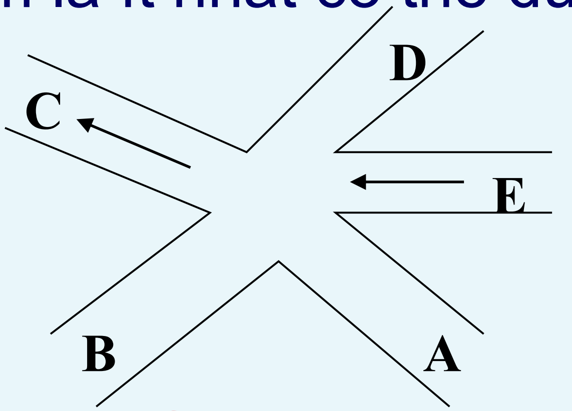
Nội dung

- Từ bài toán đến chương trình
- Kiểu dữ liệu trừu tượng
- Độ phức tạp của giải thuật

Ví dụ "Đèn Hiệu Điều khiển GT"

Một ngã năm như hình vẽ, trong đó C và E là lối đi một chiều. Hãy thiết kế một bảng đèn hiệu điều khiển giao thông một cách hợp lý để:

1. Phân chia các lối đi thành các nhóm, mỗi nhóm gồm các lối đi có thể đồng thời nhưng không xảy ra tai nạn giao thông.
2. Số lượng nhóm là ít nhất có thể được.



Ta cần làm những gì để giải bài toán này bằng chương trình máy tính?

Từ bài toán đến chương trình

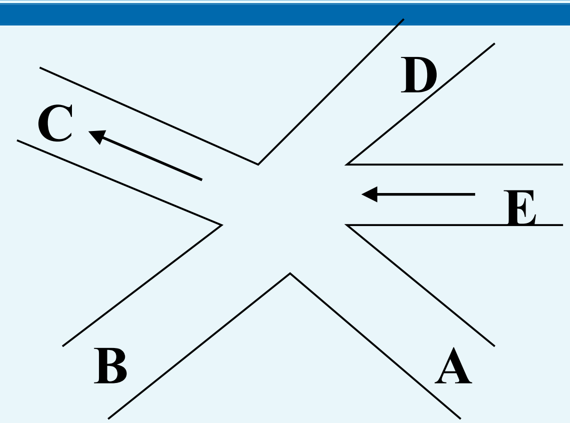
Các bước tiếp cận một bài toán:

1. **Mô hình hóa bài toán thực tế** bằng một mô hình toán học.
2. **Tìm giải thuật** (algorithms) trên mô hình này: chỉ nêu phương hướng giải hoặc các bước giải một cách tổng quát.
3. **Hình thức hoá giải thuật** bằng cách viết một thủ tục bằng *ngôn ngữ giả*, rồi chi tiết hoá dần ("mịn hoá") các bước giải tổng quát ở trên. Kết hợp việc dùng các kiểu dữ liệu trừu tượng và các cấu trúc điều khiển trong ngôn ngữ lập trình để mô tả giải thuật.
4. **Cài đặt giải thuật** trong một ngôn ngữ lập trình (NNLT) cụ thể. Các cấu trúc dữ liệu được cung cấp trong NNLT được dùng để thể hiện các kiểu dữ liệu trừu tượng, các lệnh và cấu trúc điều khiển trong NNLT được dùng để cài đặt các bước của giải thuật.

Từ bài toán đến chương trình

- Giải thuật là một chuỗi hữu hạn các thao tác để giải một bài toán nào đó.
- Các tính chất quan trọng của giải thuật là:
  - *Hữu hạn* (finiteness): giải thuật phải luôn luôn kết thúc sau một số hữu hạn bước.
  - *Xác định (definiteness):* mỗi bước của giải thuật phải được xác định rõ ràng và phải được thực hiện chính xác, nhất quán.
  - *Hiệu quả (effectiveness):* các thao tác trong giải thuật phải được thực hiện trong một lượng thời gian hữu hạn.

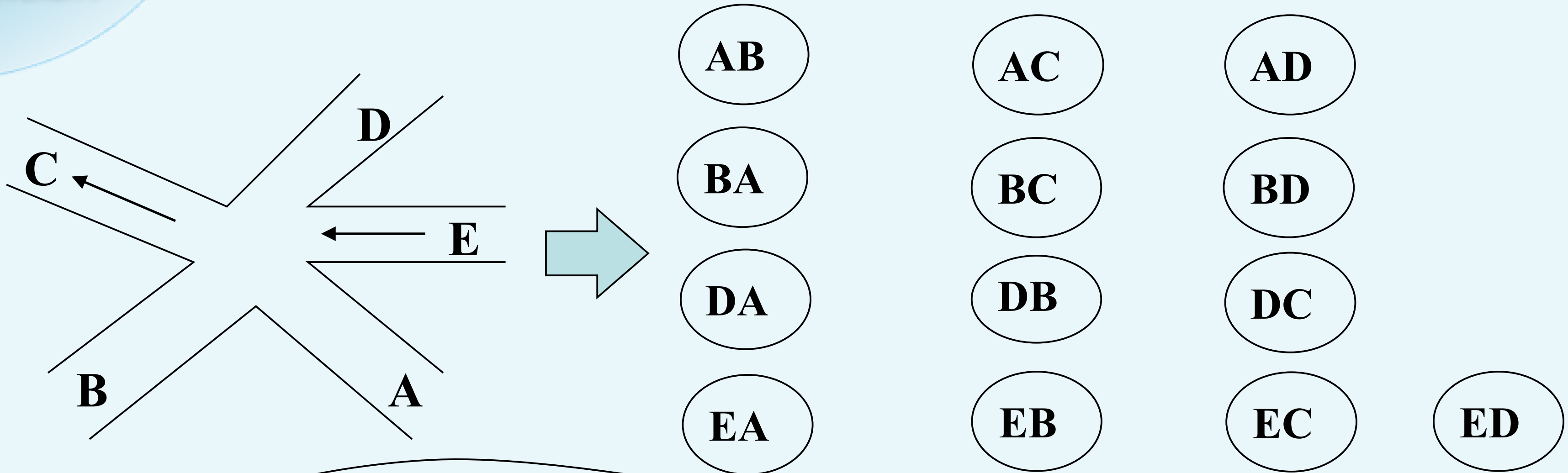
Ví dụ “Đèn Hiệu Điều khiển GT”



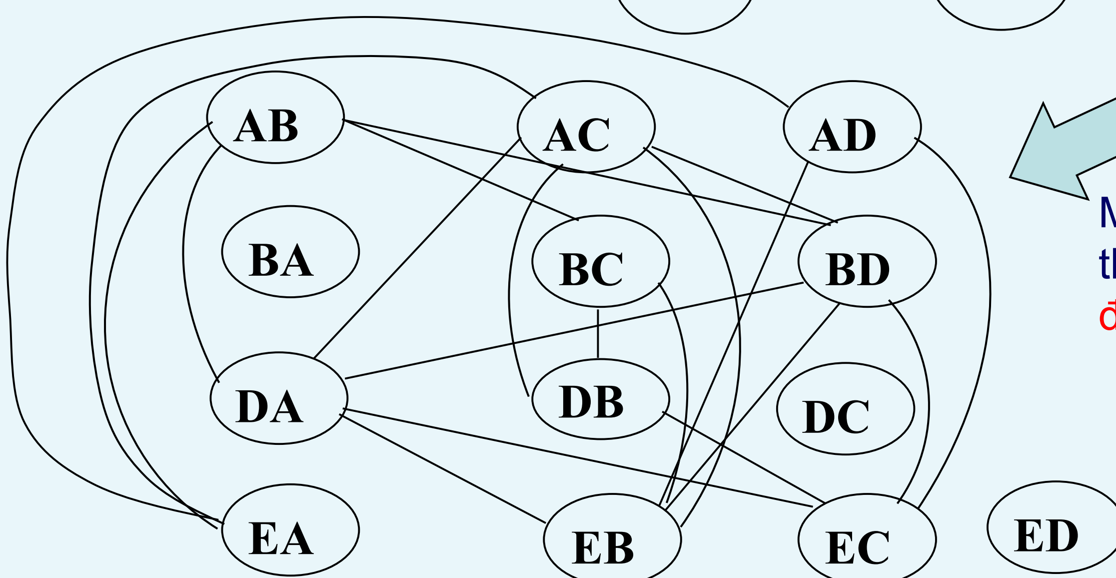
Một ngã năm như hình vẽ, trong đó C và E là lối đi một chiều. Hãy thiết kế một bảng đèn hiệu điều khiển giao thông một cách hợp lý để:

1. Phân chia các lối đi thành các nhóm, mỗi nhóm gồm các lối đi có thể đồng thời nhưng không xảy ra tai nạn giao thông.
2. Số lượng nhóm là ít nhất có thể được

Ví dụ “Đèn Hiệu Điều khiển GT”



1. Mô hình hóa bài toán thực tế



Mô hình hóa bài toán theo mô hình toán là đồ thị (Graph).

Ví dụ “Đèn Hiệu Điều khiển GT”

Mô hình hóa bài toán thực tế: phải giải quyết bài toán “Tô màu cho đồ thị” sao cho:

- Các lối đi được phép đi đồng thời sẽ được tô cùng một màu => *2 đỉnh có cạnh nối nhau sẽ không được tô cùng màu*.
- Số nhóm là ít nhất <=> ta phải tính toán để dùng *số màu ít nhất*.

Ví dụ “Đèn Hiệu Điều khiển GT”

2. Tìm giải thuật

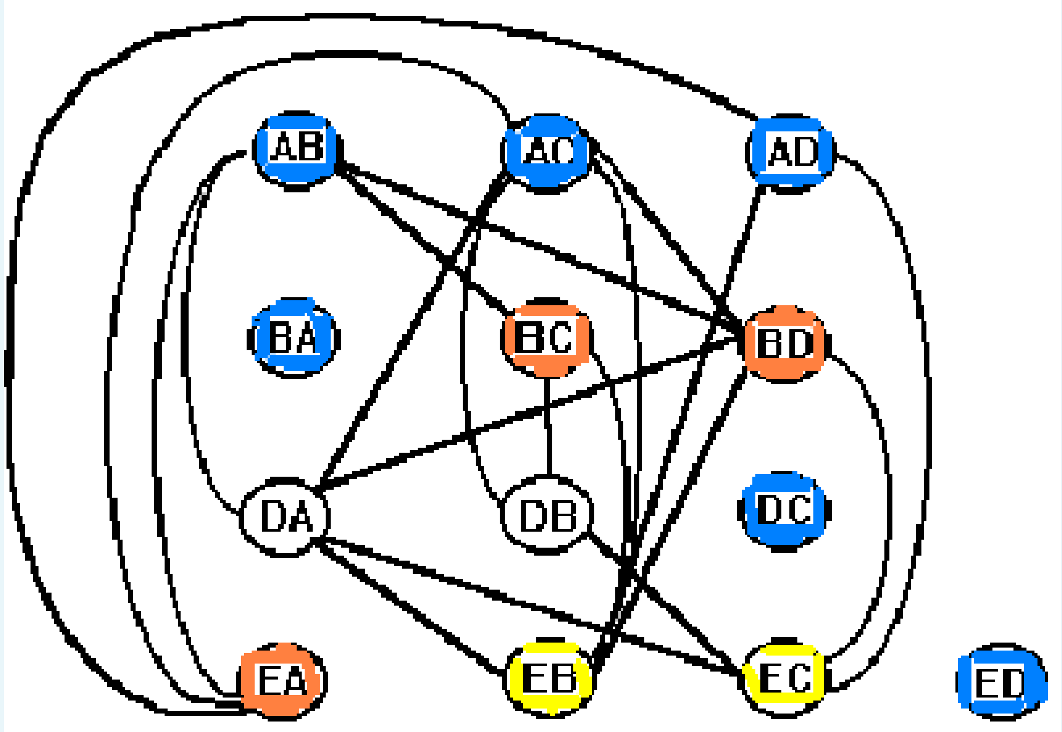
Xây dựng một giải thuật “Gready” để tô màu đồ thị

- Chọn một đỉnh chưa tô màu và tô nó bằng một màu mới C nào đó.
- Duyệt danh sách các đỉnh chưa tô màu. Đối với một đỉnh chưa tô màu, xác định xem nó có kề với một đỉnh nào được tô bằng màu C đó không. Nếu không có, tô nó bằng màu C đó.

CANTHO UNIVERSITY

Ví dụ “Đèn Hiệu Điều khiển GT”

Một lời giải cho bài toán “Đèn Hiệu Điều khiển GT” khi áp dụng giải thuật “Gready”



- Dùng 4 màu tô, mỗi màu tượng trưng cho một màu đèn
- Các lối đi có cùng một màu sẽ được đi đồng thời

Ví dụ “Đèn Hiệu Điều khiển GT”

3. Hình thức hóa giải thuật

Giải thuật mức 0:

```
void GREEDY (GRAPH *G,
                  SET     *Newclr )
{
/*1*/ Newclr = ∅;
/*2*/ while (còn đỉnh v chưa
                tô màu của G)
/*3*/  if (v không được nối với một
         đỉnh nào trong Newclr){
/*4*/   đánh dấu v đã được tô màu;
/*5*/   thêm v vào Newclr;
         }
}
```

Thủ tục GREEDY với ngôn ngữ giả C

Giải thuật mức 1:

```
void GREEDY (GRAPH *G,
                  SET     *Newclr )
{
/*1*/     Newclr= ∅;
/*2*/     while (còn đỉnh v chưa tô màu của G){
/*3.1*/      found=0;
/*3.2*/      while (mỗi đỉnh w trong Newclr)
/*3.3*/          if (có cạnh nối giữa v và w)
/*3.4*/             found=1;
/*3.5*/      if (found==0) {
/*4*/            đánh dấu v đã được tô màu;
/*5*/            thêm v vào Newclr;
             }//if
          }//while
}
```

Chi tiết hóa bước 3 của giải thuật mức 0

Ví dụ “Đèn Hiệu Điều khiển GT”

CANTHO UNIVERSITY

3. Hình thức hóa giải thuật

Giải thuật mức 2:

```
void GREEDY ( GRAPH *G,  LIST  *Newclr )
{    int found;  int v, w ;
          Newclr= ∅;
          v= đỉnh đầu tiên chưa được tô màu trong G;
          while (v<>null) {
               found=0;
               w=đỉnh đầu tiên trong Newclr;
               while( w<>null) && (found=0) {
                    if (có cạnh nối giữa v và w)
                              found=1;
                    else w= đỉnh kế tiếp trong Newclr;
               } //while
               if (found==0) {
                              Đánh dấu v đã được tô màu;
                              Thêm v vào Newclr;
               }//if
               v= đỉnh chưa tô màu kế tiếp trong G;
          }//while
}
```

Chi tiết hóa giải thuật mức 1

Kiểu dữ liệu trừu tượng (Abstract Data Types - ADT)

- Trừu tượng hóa trong tin học là đơn giản hóa: che đi những chi tiết, làm nổi bật cái tổng thể.
  - Trừu tượng hóa chương trình: phân chia chương trình thành các chương trình con => che dấu tất cả các lệnh cài đặt chi tiết trong các chương trình con ở cấp độ chương trình chính.
  - Trừu tượng hóa dữ liệu là định nghĩa các **kiểu dữ liệu trừu tượng**.

Kiểu dữ liệu trừu tượng (Abstract Data Types - ADT)

- Kiểu dữ liệu (Data type): là một tập hợp các giá trị và một tập hợp các phép toán trên các giá trị đó.
- Có 2 loại kiểu dữ liệu:
  - Kiểu dữ liệu *sơ cấp*
    - Giá trị dữ liệu là đơn nhất.
    - Ví dụ: char, int, …
  - Kiểu dữ liệu *có cấu trúc* (**cấu trúc dữ liệu**)
    - Giá trị dữ liệu là sự kết hợp của các giá trị khác.
    - Ví dụ: mảng.

Kiểu dữ liệu trừu tượng (Abstract Data Types - ADT)

- Kiểu dữ liệu trừu tượng (ADT): mô hình toán học cùng với một tập hợp các phép toán trừu tượng được định nghĩa trên mô hình đó.
- Cài đặt kiểu dữ liệu trừu tượng:
  – Tổ chức lưu trữ: một **cấu trúc dữ liệu**.
  – Viết chương trình con thực hiện các phép toán.

Từ bài toán đến chương trình

CANTHO UNIVERSITY

Các bước tiếp cận một bài toán:

1. **Mô hình hóa bài toán thực tế** bằng một mô hình toán học.
2. **Tìm giải thuật** (algorithms) trên mô hình này: chỉ nêu phương hướng giải hoặc các bước giải một cách tổng quát.
3. **Hình thức hoá giải thuật** bằng cách viết một thủ tục bằng ngôn ngữ giả, rồi chi tiết hoá dần ("mịn hoá") các bước giải tổng quát ở trên. Kết hợp việc dùng các kiểu dữ liệu trừu tượng và các cấu trúc điều khiển trong ngôn ngữ lập trình để mô tả giải thuật.
4. **Cài đặt giải thuật** trong một ngôn ngữ lập trình (NNLT) cụ thể. Các cấu trúc dữ liệu được cung cấp trong NNLT được dùng để thể hiện các kiểu dữ liệu trừu tượng, các lệnh và cấu trúc điều khiển trong NNLT được dùng để cài đặt các bước của giải thuật.

Độ phức tạp của giải thuật

- Một bài toán có thể có nhiều cách giải khác nhau.
- Ta cần phải phân tích, đánh giá giải thuật để:
  – Lựa chọn một giải thuật tốt nhất trong các giải thuật để cài đặt chương trình giải quyết bài toán đặt ra.
  – Cải tiến giải thuật hiện có để được một giải thuật tốt hơn.
- Độ phức tạp của giải thuật (algorithm complexity) liên quan đến mức độ nhanh hay chậm thực hiện một giải thuật.

Độ phức tạp của giải thuật

Ví dụ: Hãy sắp xếp mảng có n phần tử theo thứ tự tăng dần.

- Giải thuật sắp xếp thông dụng dùng 2 vòng lặp i:1->n và j:1->n để đổi chỗ.

```
void  Sort(int a[], int n){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            if (a[j] > a[i]) {
                int tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;     }
        }
    }
}
```

có độ phức tạp là $O(n^2)$

- Giải thuật sắp xếp Quicksort có độ phức tạp O(n*log(n)).

Độ phức tạp của giải thuật

- *Thời gian thực hiện một giải thuật* là một hàm của kích thước dữ liệu vào, ký hiệu $T(n)$ trong đó n là kích thước (độ lớn) của dữ liệu vào.
  - Ví dụ : Chương trình tính tổng của n số có thời gian thực hiện là $T(n) = cn$ trong đó c là một hằng số.
- Thời gian thực hiện giải thuật là một hàm không âm, tức là $T(n) \geq 0 \ \forall \ n \geq 0$.

Độ phức tạp của giải thuật

- Thời gian thực hiện giải thuật không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào.
- Vì vậy thường ta coi T(n) là thời gian thực hiện chương trình trong trường hợp xấu nhất trên dữ liệu vào có kích thước n, tức là: T(n) là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n.

Độ phức tạp của giải thuật

- Trong những ứng dụng thực tiễn, chúng ta *không cần biết chính xác hàm T(n)* mà *chỉ cần biết một ước lượng đủ tốt* của nó.
- Ước lượng thường dùng nhất là ước lượng cận trên O-lớn.
- Cho một hàm T(n), T(n) gọi là có độ phức tạp f(n) nếu tồn tại các hằng C, N_0 sao cho $T(n) \leq Cf(n)$ với mọi $n \geq N_0$ (tức là T(n) có tỷ suất tăng là f(n)) và kí hiệu $T(n)=O(f(n))$.
- Chú ý: $O(Cf(n))=O(f(n))$ với C là hằng số. Đặc biệt $O(C)=O(1)$.

Độ phức tạp của giải thuật

Ví dụ: cho một hàm $T(n) = n^2 + 2n + 1$. Hãy chứng minh rằng $T(n) = O(n^2)$?

- Tìm C và N_0 sao cho $T(n) \leq Cf(n)$ hay $n^2 + 2n + 1 \leq c*n^2$ với mọi $n \geq N_0$.
- Đặt $N_0=1$, thì $1 \leq n \leq n^2$ với mọi $n \geq 1$. Do đó, ta có:

  $$1 + 2n + n^2 \leq n^2 + 2n^2 + n^2 = 4n^2$$

  Vậy, C=4.

Độ phức tạp của giải thuật

Bài tập: Kiểm tra xem những câu sau là đúng hay sai?

a) $10 + n + n^2 = O(n^2)$

b) $10 + n + \log n = O(n^2)$

c) $7\log^2 n + 2n\log n = O(n)$

d) $(1/3)^n + 100 = O(1)$

e) $3^n + 100 = O(1)$

f) $2^n + 100n^2 + n^{100} = O(n^{101})$

Độ phức tạp của giải thuật

- Các dạng phức tạp thường gặp

Dạng phức tạp	Hàm thể hiện độ phức tạp	Thời gian thực hiện
Hằng		O(1)
Lôgarit	log(n)	O(log(n))
Tuyến tính	n	O(n)
	n*log(n) Dạng đa thức	O(n*log(n))
Bậc hai	n^2	$O(n^2)$
Khối	n^3	$O(n^3)$
Mũ	$2^n, n!, n^k$ Dạng mũ	$O(2^n)$, $O(n!)$, $O(n^k)$

- Một giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật.

CANTHO UNIVERSITY

Ví dụ: nếu chúng ta lấy một máy tính trung bình từ năm 2008 với giả định rằng nó có thể thực hiện khoảng 50,000,000 phép toán cơ bản mỗi giây.

Algorithm	n=10	n=20	n=50	n=100	1,000	10,000	100,000
O(1)	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.
O(log(n))	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.
O(n)	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.
O(n*log(n))	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.
$O(n^2)$	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	< 1 sec.	2 sec.	3-4 min.
$O(n^3)$	< 1 sec.				20 sec.	5.55 hours	231.5 days
$O(2^n)$	< 1 sec.	< 1 sec.	260 days	treo	treo	treo	treo
O(n!)	< 1 sec.	treo	treo	treo	treo	treo	treo
$O(n^n)$	3-4 min.	treo	treo	treo	treo	treo	treo

$10{,}000^3/(60phut*60giay) =$ 5.55 giờ

https://introprogramming.info/english-intro-csharp-book/read-online/chapter-19-data-structures-and-algorithm-complexity/

Tính độ phức tạp của giải thuật

Bài tập: giả sử máy tính có thể thực hiện 1 hoạt động chính của giải thuật trong 1 micro giây.

- Xác định thời gian (số năm) thực hiện giải thuật có độ phức tạp $O(2^n)$, $O(3^n)$ và $O(n!)$ khi $n=10$, $n=40$ và $n=100$?
- Nếu tuổi thọ của một hành tinh khoảng 20 tỷ năm thì hành tinh đó vẫn còn tồn tại khi giải thuật có độ phức tạp $O(n!)$ với $n=40$ thực hiện xong?

CANTHO UNIVERSITY

Tính độ phức tạp của giải thuật

- **Quy tắc cộng**: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1 và P2; và T1(n)=O(f(n)), T2(n)=O(g(n)) thì thời gian thực hiện của đoạn hai chương trình đó nối tiếp nhau là T(n)=O(max(f(n),g(n))).
- **Quy tắc nhân**: Nếu T1(n) và T2(n) là thời gian thực hiện của hai đoạn chương trình P1và P2 và T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện của đoạn hai đoạn chương trình đó lồng nhau là T(n) = O(f(n).g(n)).

Tính độ phức tạp của giải thuật

- Thời gian thực hiện của mỗi lệnh gán, nhập, xuất là O(1).
- Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng.
- Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau IF hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là O(1).
- Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

Tính độ phức tạp của giải thuật

Ví dụ: tính độ phức tạp của hàm tìm phần tử lớn nhất trong một mảng có n số nguyên?

```
int FindMaxElement(int[] a, int n)
{
    int max = int.MinValue;          O(1)
    for (int i = 0; i < n; i++){
        if (a[i] > max){             O(1)
            max = a[i];              O(1)
        }                            O(max(1,1))=O(1)
    }                                O(n*1) =O(n)
    return max;                      O(1)
}                                    O(max(1, n, 1)) =O(n)
```

https://introprogramming.info/english-intro-csharp-book/read-online/chapter-19-data-structures-and-algorithm-complexity/

Tính độ phức tạp của giải thuật

Ví dụ: tính độ phức tạp của đoạn chương trình sau:

```
/*1*/ Sum=0;
/*2*/ for(i=1;i<=n;i++) {
/*3*/       scanf("%d",&x);
/*4*/       Sum=Sum+x;
          }
```

- Dòng 3,4 là các thao tác nhập, gán nên độ phức tạp là O(1). Áp dụng quy tắc cộng, độ phức tạp cho cả 3 và 4 là O(1)
- Vòng lặp 2 thực hiện n lần mỗi lần O(1) => theo quy tắc nhân độ phức tạp của 2 là O(n)
- Dòng 1 là phép gán => O(1)
- Áp dụng quy tắc cộng cho 1 và 2 -> T(n) = O(n)

Tính độ phức tạp của giải thuật

Ví dụ: tính độ phức tạp cho đoạn chương trình sau:

```
/*1*/ Sum=0;
/*2*/ for(i=1;i<=n;i++) {
/*3*/     if (i%2==0)
/*4*/         Sum=Sum+i;
          }
```

- Dòng 3 rẽ nhánh với kiểm tra điều kiện mất O(1), công việc thực hiện khi điều kiện đúng mất O(1) => theo quy tắc cộng độ phức tạp của 3 là O(1)
- Vòng lặp 2 thực hiện n lần mỗi lần O(1) => theo quy tắc nhân độ phức tạp của 2 là O(n)
- Dòng 1 là phép gán => O(1)
- Áp dụng quy tắc cộng cho 1 và 2 => T(n) = O(n)

Tính độ phức tạp của giải thuật

Ví dụ: tính độ phức tạp của đoạn chương trình sau:

```
/*1*/ Sum=0;
/*2*/ for(i=1;i<=n;i=i*2) {
/*3*/     scanf("%d",&x);
/*4*/     Sum=Sum+x;
        }
```

- Cách tính tương tự như ví dụ trang 29, tuy nhiên ở đây do vòng lặp 2 thực hiện ~ logn lần, do đó T(n) = O(logn)

Bài tập: tính độ phức tạp của đoạn chương trình sau:

```
/*1*/ Sum1=0;
/*2*/ k=1;
/*3*/ while (k<=n) {
/*4*/    for(j=1;j<=n;j++)
/*5*/      Sum1=Sum1+1;
/*6*/    k=k*2;
      }
```

Tính độ phức tạp của giải thuật

Bài tập: tính độ phức tạp của hàm Sum3 sau?

```
long Sum3(int n)
{
    long sum = 0;
    for (int a = 1; a < n; a++) {
        for (int b = 1; b < n; b++){
            for (int c = 1; c < n; c++){
                sum += a * b * c;
            }
        }
    }
    return sum;
}
```

Tính độ phức tạp của giải thuật

Ví dụ: tính độ phức tạp của đoạn chương trình sau?

```
int count = 0;
for (int i = 0; i < n; i++)
   for (int j = 0; j < i; j++)
      count++;
```

Khi i=0, đoạn chương trình chạy 0 lần lệnh count++.
Khi i=1, đoạn chương trình chạy 1 lần lệnh count++.
Khi i=2, đoạn chương trình chạy 2 lần lệnh count++.
…
Khi i=n-1, đoạn chương trình chạy n-1 lần lệnh count++.
Tổng số lần chạy lệnh count++ là: $0 + 1 + 2 + \cdots + (n - 1) = \frac{n*(n-1)}{2}$ => Độ phức tạp của đoạn chương trình O(n^2)

https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/#:~:text=Time%20complexity%20of%20an%20algorithm,the%20length%20of%20the%20input.&text=Let%20each%20operation%20takes%20time.

Tính độ phức tạp của giải thuật

Bài tập: tính độ phức tạp đoạn chương trình sau?

```
int count = 0;
for (int i = n; i > 0; i/=2)
   for (int j = 0; j < i; j++)
      count++;
```

Tính độ phức tạp của giải thuật

CANTHO UNIVERSITY

Ví dụ: để giải một bài toán thực tế, ta có thể sử dụng một trong hai giải thuật A hoặc B. Giải thuật A có $T_A(n)=100*n*n$ ms = $O(n^2)$; giải thuật B có $T_B(n)=5*n*n*n$ ms = $O(n^3)$. Ta nên chọn giải thuật A hay giải thuật B?

http://www.dcs.gla.ac.uk/~pat/52233/complexity.html

n	A	B
1	0.1s	0.005s
2	0.4s	0.04s
3	0.9s	0.135s
4	1.6s	0.32s
5	2.5s	0.625s
6	3.6s	1.08s
7	4.9s	1.715s
8	6.4s	2.56s
9	8.1s	3.645s
10	10s	5s
11	12.1s	6.655s
12	14.4s	8.64s
13	16.9s	10.985s
14	19.6s	13.72s
15	22.5s	16.875s
16	25.6s	20.48s
17	28.9s	24.565s
18	32.4s	29.16s
19	36.1s	34.295s
20	40s	40s
21	44.1s	46.305s
…	…	…
30	90s	135s
40	160s	320s

Q&A?