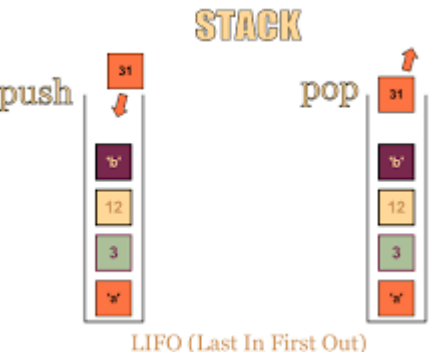
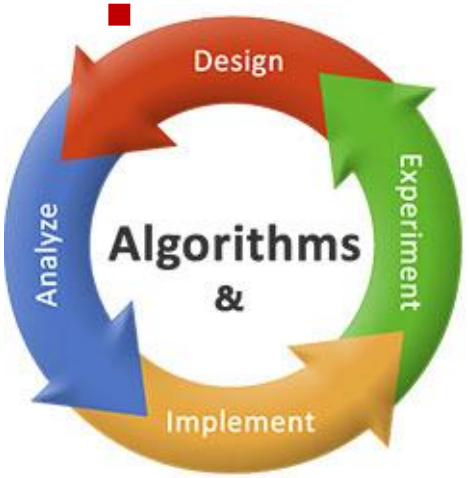
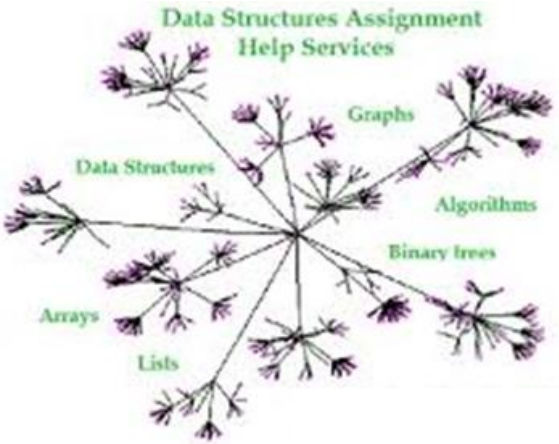


CẤU TRÚC DỮ LIỆU

&

GIẢI THUẬT





Lê Văn Hạnh

levanhanhvn@gmail.com

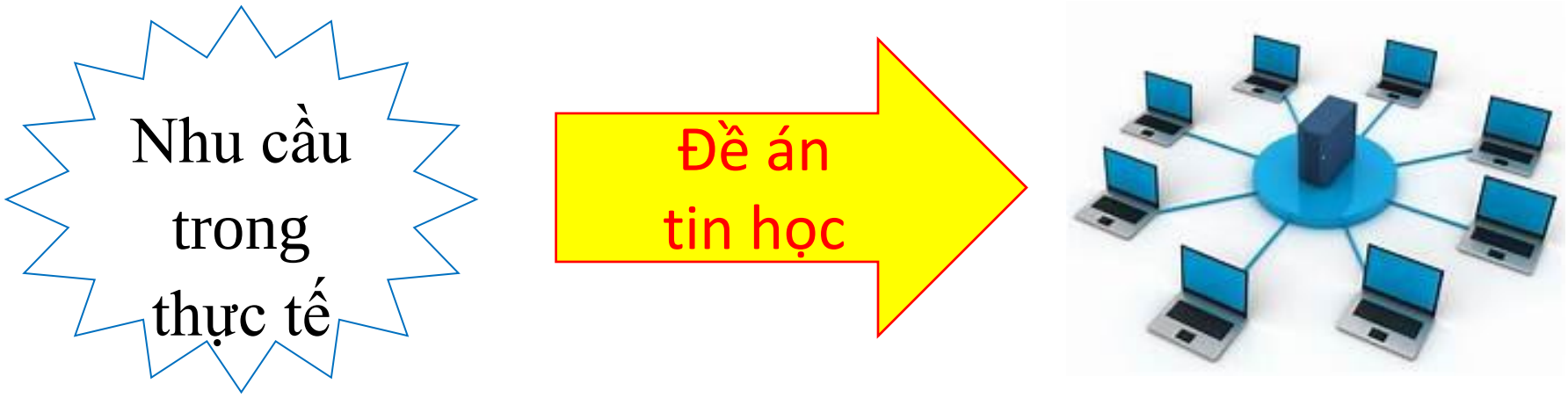
NỘI DUNG MÔN HỌC

- Chương 1: Ôn tập ngôn ngữ lập trình C
- Chương 2: Kiểu dữ liệu con trỏ
- **Chương 3: Tổng quan về cấu trúc dữ liệu và giải thuật**
- Chương 4: Danh sách kề (Danh sách tuyến tính)
- Chương 5: Các giải thuật tìm kiếm trên danh sách kề
- **Chương 6**: Các giải thuật sắp xếp trên danh sách kề
- **Chương 7**: Danh sách liên kết động (*Linked List*)
- **Chương 8**: Ngăn xếp *(Stack)*
- **Chương 9:** Hàng đợi *(Queue)*
- **Chương 10**: Cây nhị phân tìm kiếm (*Binary Search Tree*)
- **Chương 11**: Cây cân bằng (*Binary Search Tree*)
- **Chương 12**: Bảng băm (*Hash Table*)

Chương 3

TỔNG QUAN
VỀ CẤU TRÚC DỮ LIỆU
& GIẢI THUẬT

1. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU TRONG MỘT ĐỀ ÁN TIN HỌC



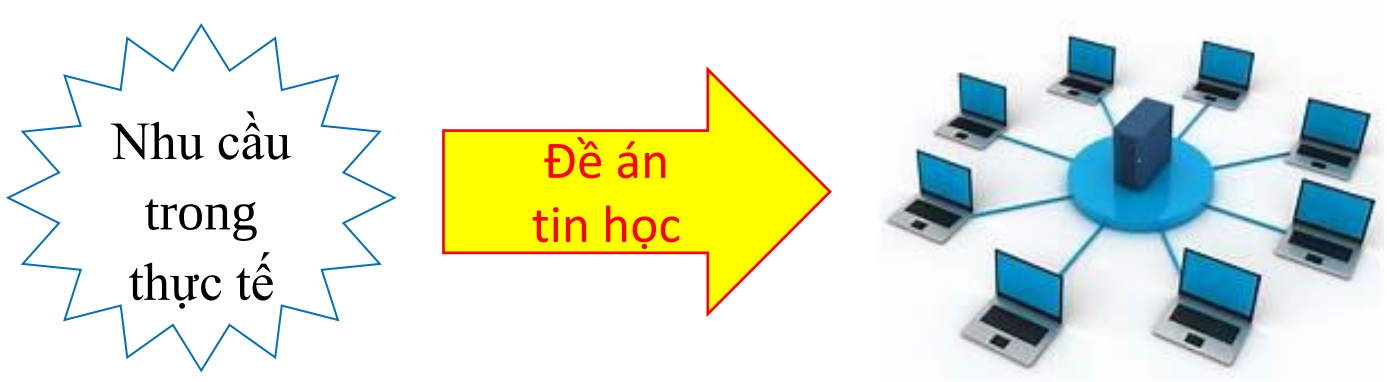
Đề án tin học: 2 vấn đề cần quan tâm

i. Tổ chức biểu diễn các đối tượng thực tế:

- Dữ liệu trong thực tế:
  - Muôn hình vạn trạng, đa dạng, phong phú
  - Thường có chứa đựng quan hệ với nhau

Đây là bước xây dựng cấu trúc dữ liệu cho bài toán thực tế được chuyển vào giải quyết trong máy tính

1. Vai trò của cấu trúc dữ liệu trong một đề án tin học



ii.Xây dựng các thao tác xử lý dữ liệu:

- Dựa trên yêu cầu cụ thể, xác định các trình tự giải quyết vấn đề trên máy tính để đưa kết quả mong muốn.
- Cần phải tổ chức biểu diễn thành cấu trúc thích hợp nhất nhằm mục đích:
  - Phản ánh chính xác dữ liệu thực tế
  - Xây dựng giải thuật ⇒ hướng đến các phép xử lý
  - Dễ dàng xử lý trong máy tính

Đây là bước xây dựng giải thuật cho bài toán

1. VAI TRÒ CỦA CẤU TRÚC DỮ LIỆU TRONG MỘT ĐỀ ÁN TIN HỌC

Thực tế

Chọn loại (kiểu) dữ liệu gì để biểu diễn thực tế

Cấu trúc dữ liệu

Là đối tượng xử lý

Giải thuật

Giải thuật phù hợp với kiểu dữ liệu đã chọn ⇒ Cấu trúc dữ liệu thay đổi ⇒ giải thuật thay đổi

Những thao tác nào sẽ tác động đến kiểu dữ liệu đã chọn

dễ dàng xử lý trên máy tính

Cấu trúc dữ liệu + Giải thuật = Chương trình

2. TIÊU CHUẨN ĐÁNH GIÁ CẤU TRÚC DỮ LIỆU

- ***Phản ánh đúng thực tế***
  - Thể hiện được đầy đủ thông tin nhập/xuất của bài toán.
  - Là tiêu chuẩn quan trọng nhất, quyết định tính đúng đắn của toàn bộ bài toán.
- ***Phù hợp với các thao tác xử lý***:
  - Giúp việc phát triển các thuật toán đơn giản, tự nhiên hơn; chương trình đạt hiệu quả cao hơn về tốc độ xử lý.
  - Tăng tính hiệu quả của chương trình → hiệu quả của dự án tin học.
- ***Tiết kiệm tài nguyên hệ thống:*** tùy vào tình huống cụ thể khi thực hiện đề án để chọn nên tối ưu thời gian xử lý hay tối ưu bộ nhớ (hoặc cả hai).

3. KIỂU DỮ LIỆU CƠ BẢN

- Thường đơn giản và không có cấu trúc, được NNLT xây dựng sẵn, nên còn gọi là kiểu dữ liệu dựng sẵn.
- Thường là các trị vô hướng:
  - số nguyên
  - số thực
  - ký tự
  - giá trị logic

3. KIỂU DỮ LIỆU CƠ BẢN

Tên kiểu	KT	Miền giá trị	Ghi chú
char	**1**	-128 đến 127	Có thể dùng như số nguyên 1 byte có dấu hay kiểu ký tự
unsigned char	**1**	0 đến 255	Số nguyên 1 byte ko dấu
int	**2**	-32768 đến 32767	
unsigned int	**2**	0 đến 65355	Gọi tắt là unsigned
long	**4**	-2^{32} đến $2^{31}-1$	
unsigned long	**4**	0 đến $2^{32}-1$	
float	**4**	3.4E-38 … 3.4E38	Giới hạn chỉ trị tuyệt đối.Các giá trị <3.4E-38 được coi = 0. Tuy nhiên kiểu float chỉ có 7 chữ số có nghĩa.
double	**8**	1.7E-308 … 1.7E308	
long double	**10**	3.4E-4932 …1.1E4932	

4. KIỂU DỮ LIỆU CÓ CẤU TRÚC

- Trong nhiều trường hợp, các **kiểu dữ liệu cơ sở không đủ để phản ánh tự nhiên** và đầy đủ bản chất của sự vật thực tế, dẫn đến nhu cầu phải xây dựng các kiểu dữ liệu mới dựa trên việc tổ chức, liên kết các thành phần dữ liệu có kiểu dữ liệu đã được định nghĩa.
- Kiểu cấu trúc:
  - Kết hợp nhiều kiểu dữ liệu cơ bản để phản ánh bản chất của đối tượng thực tế
  - Đa số các ngôn ngữ đều cài đặt sẵn một số kiểu có cấu trúc cơ bản: mảng, chuỗi, tập tin, bản ghi…
  - Ngoài ra cung cấp cơ chế cho người lập trình cài đặt các dữ liệu cấu trúc khác.

4. Kiểu dữ liệu có cấu trúc

- Ví dụ: Để mô tả một đối tượng sinh viên:

```
typedef struct tagDate
{     char ngay;
  char thang;
  char thang;
}Date;

typedef struct tagSinhVien
{  char masv[15];
   char tensv[15];
   Date ngaysinh;
   char noisinh[15];
   int Diem thi;
}SinhVien;
```

5. TRỪU TƯỢNG HÓA DỮ LIỆU

- Trừu tượng hoá là ý niệm về sự vật hay hiện tượng bao gồm các trạng thái tĩnh (*info*) và các tác vụ (*operation*) lên dữ liệu đó.
- Trừu tượng hoá giúp
  - Làm đơn giản hóa, sáng sủa, dễ hiểu hơn.
  - Che đi phần chi tiết, làm nổi bật cái tổng thể

5. Trừu tượng hóa dữ liệu

- Kết quả của quá trình trừu tượng hoá giúp chúng ta xây dựng một mô hình cho một kiểu dữ liệu mới gọi là kiểu dữ liệu trừu tượng (*Abstract Data Type - ADT*), mỗi kiểu dữ liệu trừu tượng có mô tả dữ liệu và các tác vụ liên quan.
- *Abstract Data Type* : gồm
  - Mô tả dữ liệu
  - Tác vụ liên quan

5. Trừu tượng hóa dữ liệu

- Ví dụ 1: Trừu tượng hoá dữ liệu phân số

KIỂU DỮ LIỆU TRỪU TƯỢNG VỀ PHÂN SỐ (PhanSo)

Mô tả dữ liệu:

- Tử số.
- Mẫu số (mẫu số phải khác 0).

Mô tả tác vụ:

- Tác vụ cộng: **add**(PhanSo1, PhanSo2)
  - Đầu vào:
    - a,b là tử và mẫu của PhanSo1
    - c,d là tử và mẫu của PhanSo2
  - Xử lý:
    - t = ad+bc là tử của phân số kết quả (chưa rút gọn).
    - m = bd là mẫu của phân số kết quả(chưa rút gọn).
    - tìm ước số chung lớn nhất (us) của t và m.
  - Đầu ra:
    - t/us: là tử của phân số kết quả.
    - m/us: là mẫu của phân số kết quả.
- Tác vụ nhân: **mul**(PhanSo1, PhanSo2)

  Đầu vào: …

  Đầu ra: …

…

5. Trừu tượng hóa dữ liệu

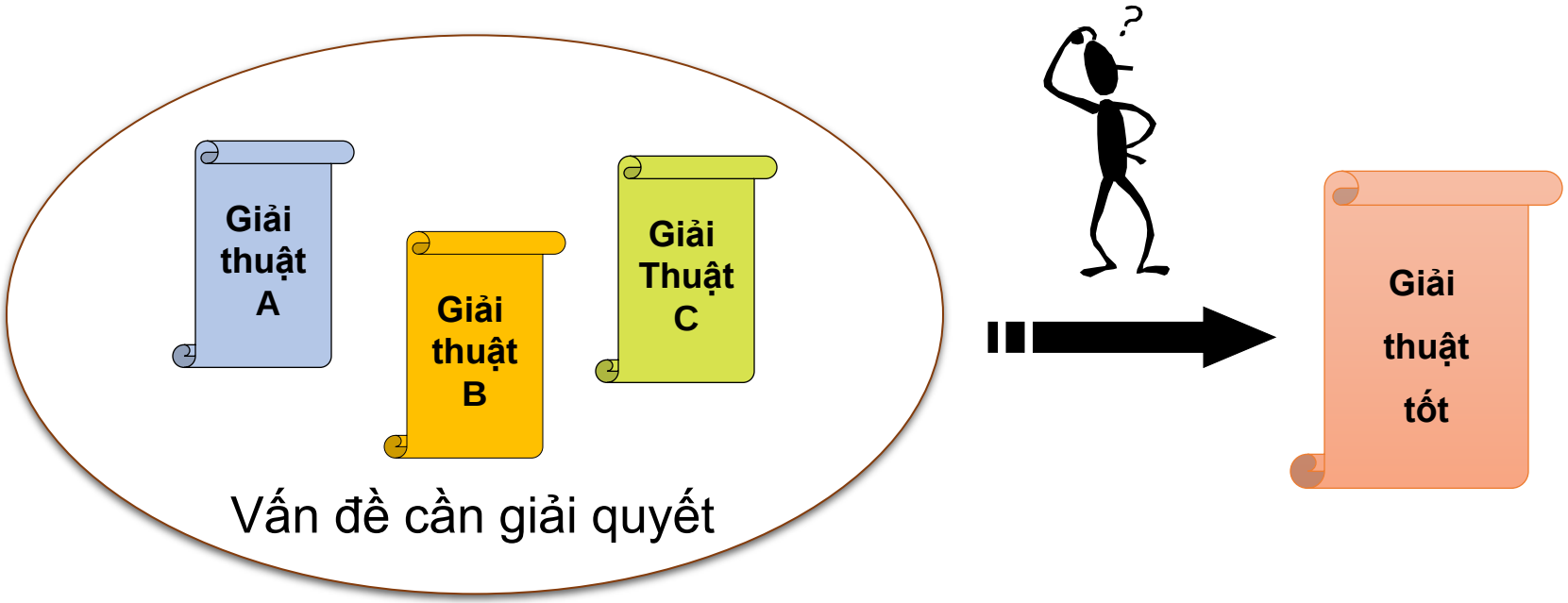
- Ví dụ 2: mô tả kiểu dữ liệu trừu tượng về số hữu tỉ a/b với các tác vụ: +,*, /
  - Mô tả dữ liệu
    - Tử số
    - Mẫu số (≠0)
  - Mô tả tác vụ

    Cộng(Số_Hữu_Tỉ 1, Số_Hữu_Tỉ 2)
    - Nhập:
      - a, b là tử số và mẫu số của Số_Hữu_Tỉ 1
      - c, d là tử số và mẫu số của Số_Hữu_Tỉ 2
    - Xuất:
      - ad+bc là tử số của số hữu tỉ kết quả
      - bd là mẫu số của số hữu tỉ kết quả

4. ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

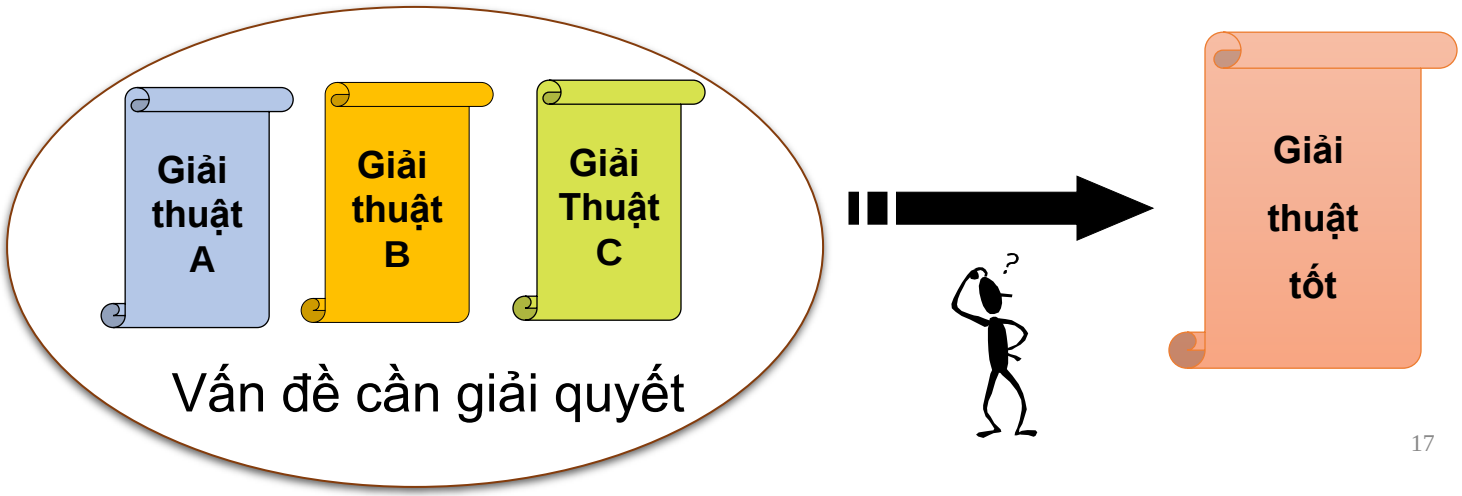
4.1. Giải thuật

- Hầu hết các bài toán đều có nhiều giải thuật khác nhau để giải quyết chúng. Vậy làm thế nào chọn được một giải thuật tốt nhất?



4.1. Giải thuật

- Việc chọn lựa phụ thuộc vào nhiều yếu tố như:
  - Độ phức tạp tính toán của giải thuật
  - Dung lượng bộ nhớ bị chiếm dụng
  - Tần suất sử dụng
  - Tính đơn giản
  - Tốc độ thực hiện
  - ...



4.1. Giải thuật

- Thông thường mục tiêu chọn lựa giải thuật là:
  i. Giải thuật đúng
  ii. Giải thuật rõ ràng, dễ hiểu, dễ mã hóa và hiệu chỉnh.
  iii. Giải thuật sử dụng có hiệu quả tài nguyên của máy tính và đặc biệt thời gian thực hiện càng nhanh càng tốt.

(i) Bắt buộc với mọi giải thuật.

(ii) Quan trọng khi chương trình ít khi thực hiện (số lần chạy ít).

(iii) Quan trọng khi chương trình được thực hiện nhiều lần (số lần chạy nhiều)

4.1. Giải thuật: Ví dụ kiểm tra tính nguyên tố của một số nguyên dương n (n>=2):

```
bool IsPrime(int n)
{  if (n<=1)
        return false;
   for(int i=2; i<n;i++)
        if (n %i==0)
            return false;
   return true;
}
```

Nếu n là số nguyên tố, cần n-2 phép mod.

Giả sử máy tính có tốc độ xử lý 100 ngàn tỷ phép mod (10^{14})/giây.

Giả sử số n có 25 chữ số

⇒Thời gian để kiểm tra n là số nguyên tố:

$$\frac{10^{25}}{10^{14} \times 60 \times 60 \times 24 \times 365} \approx 3170 \text{ năm}$$

4.1. Giải thuật: Cải tiến bài toán kiểm tra tính nguyên tố của một số nguyên dương n (n>=2):

```
bool IsPrime(int n)
{  if (n<=1)
        return false;
   for(int i=2; i<=sqrt(n);i++)
        if (n %i==0)
            return false;
   return true;
}
```

Vẫn với tốc độ xử lý 100 ngàn tỷ phép mod (10^{14})/giây và với n có 25 chữ số

⇒Thời gian để kiểm tra n là số nguyên tố:

$$\frac{\sqrt{10^{25}}}{10^{14}} \approx 0.03 \text{ giây}$$

4.2. Đánh giá thời gian thực hiện của thuật toán

i. ***Cách 1***: thông qua thực nhiệm: viết và chạy chương trình với nhiều dữ liệu khác nhau trên cùng 1 máy tính.

ii. ***Cách 2***: dựa trên lý thuyết:

- Sử dụng ký hiệu O(f(n)) để mô tả độ lớn của hàm T(n).
- Coi thời gian thực hiện thuật toán như một hàm số của cỡ dữ liệu vào (VD trong bài toán kiểm tra số nguyên tố thì cỡ của dữ liệu vào là n.
- Gọi T(n) là hàm số biểu diễn thời gian nhiều nhất cần thiết để thực hiện chương trình.

VD: Giả sử $T(n)= n^2+2n \Rightarrow n^2+2n \leq n^2+2n^2 \leq 3n^2 \ \forall n>=1$

$\Rightarrow T(n)=O(n^2)$

Thuật toán có thời gian thực hiện cấp n^2

4.3. Độ phức tạp của thuật toán (*algorithm complexity*)

- Độ phức tạp của thuật toán (*algorithm complexity*) hay độ phức tạp về thời gian thực thi giúp ước lượng số phép tính cần thực hiện hay đánh giá độ phức tạp giải thuật.
- Độ phức tạp giải thuật càng thấp $\Rightarrow$ thời gian thực hiện chương trình cành nhanh và ngược lại.

4.3. Độ phức tạp của thuật toán (*algorithm complexity*)

- Thường dựa vào số phép so sánh và số phép gán
- Một số công thức thường dùng để tính độ phức tạp:

$$\sum_{i=1}^{n} 1 = n$$

$$\sum_{i=a}^{b} 1 = \sum_{i=1}^{b} 1 - \sum_{i=1}^{a} 1 = b - a + 1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=a}^{n} i = \frac{n(n+1) - (a-1)a}{2}$$

$$\sum_{i=1}^{n^2} i = \frac{n^2(n^2+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$$

4.4. Các độ phức tạp giải thuật thường gặp

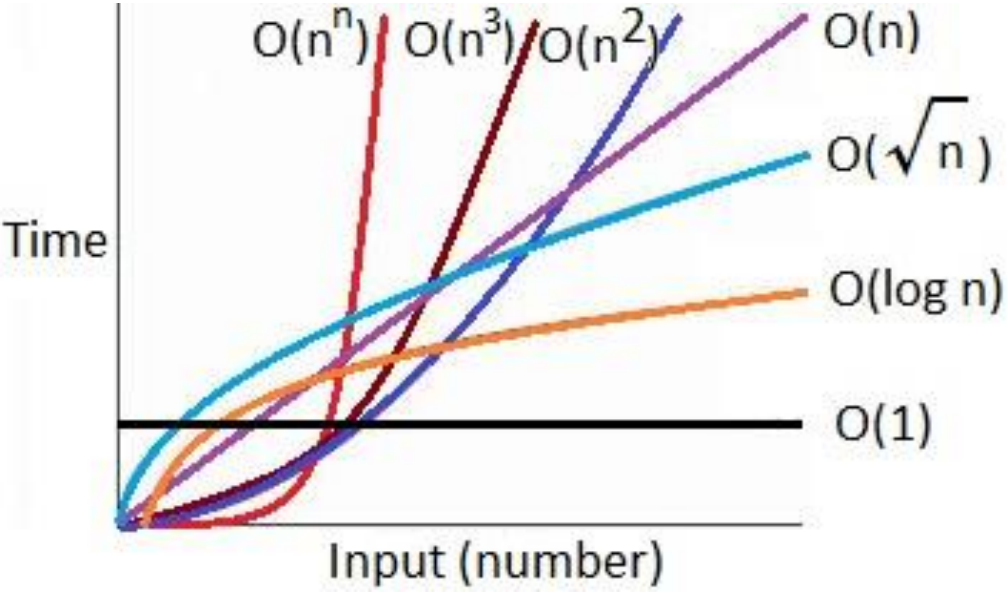
- ***Hằng số***: là mục tiêu phấn đấu để đạt được trong việc thiết kế thuật toán.
- ***logN***
  - Thời gian chạy chương trình tiến chậm khi N lớn dần.
  - Xuất hiện trong các chương trình mà việc giải 1 bài toán lớn bằng cách chuyển nó thành bài toán nhỏ hơn, bằng cách cắt bỏ kích thước bớt 1 hằng số nào đó.
  - Ví dụ: Khi N là 1000 thì $\log_{10}N$ là 3; và $\log_2N$ là 10

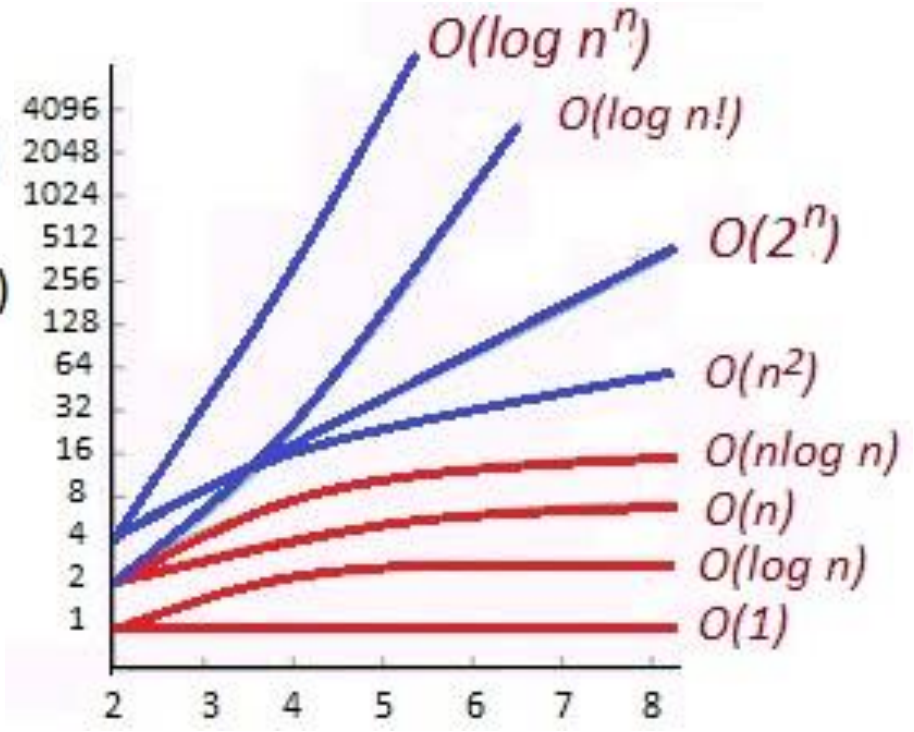
4.4. Các độ phức tạp giải thuật thường gặp

- N
  - Thời gian chạy của chương trình là tuyến tính.
  - Xuất hiện trong trường hợp mà một số lượng nhỏ các xử lý được làm cho mỗi phần tử dữ liệu nhập.
- N^2: Xuất hiện trong các thuật toán mà xử lý tất cả các cặp phần tử dữ liệu (có thể là 2 vòng lặp lồng nhau).
- N^3:
  - Có thể thuật toán dùng 3 vòng lặp lồng nhau
  - Chỉ có ý nghĩa thực tế trong các bài toán nhỏ.
- 2^n Có thể xem các thuật toán có độ phức tạp này như là "sự ép buộc thô bạo" để giải bài toán.

4.4. Các độ phức tạp giải thuật thường gặp

hằng số, $\log_2 n$, n, $n\log_2 n$, n^2, n^3, $\log n!$, $\log n^n$, $n!$, 2^n, n^n





- Thông thường thuật giải có độ phức tạp đa thức thì có thể cài đặt.
- Độ phức tạp ở mức hàm mũ thì phải cải tiến giải thuật!

4.5. Quy tắc tính độ phức tạp giải thuật

- ***Mỗi lệnh*** gán, *printf* (hay *cout*), *scanf* (hay *cin*) được tính là *O(1)*.
- ***Lệnh if***: là thời gian thực hiện lệnh điều kiện cộng với thời gian kiểm tra điều kiện.
- ***Lệnh if ... else ...*** là thời gian kiểm tra điều kiện cộng với **thời gian lớn nhất** của 1 trong 2 lệnh rẽ nhánh *true* và *false*.

```
void ThucHien()
{
{1} printf("Nhap n");
{2} scanf ("%d",&n);
{3} SLChan=0;
{4} SLLe=0;
{5} tam = n;
{6} while (n>0)
    {
{7}    so=n%10;
{8}    if (so%2==0)
{9}       SLChan++;
       else
       {
{10}      SLLe++;
{11}      printf("%d", so);
       }
{12}   n=n%10;
    }
{13} printf("Chan=%d,Le=%d",SLChan,SLLe);
}
```

- ***Một khối lệnh*** xác định theo ***quy tắc tổng***; nghĩa là thời gian chạy của khối là **thời gian lớn nhất** của 1 lệnh nào đó trong khối lệnh.

4.5. Quy tắc tính độ phức tạp giải thuật

- ***Vòng lặp***:
  - ***Đơn***: là tổng thời gian thực hiện thân vòng lặp và thời gian kiểm tra kết thúc vòng lặp.
  - ***Lồng nhau***: là tích thời gian thực hiện của các vòng lặp.

```
void ThucHien()
{
{1}    int n;
{2}    print("Nhap n");
{3}    scanf ("%d",&n);
{4}    for(i = 1; i <=n ; i++)
{5}    {
{6}        if (so%2==0)
{7}           SLChan++;
{8}        else
{9}           SLLe++;
       }
{10} printf("Chan=%d,Le=%d",SLChan,SLLe);
}
```

```
void SelectionSort(int a[],int n)
{
{1}   int min,i,j;
{2}    for(i=0;i<n-1;i++)
{3}    {
{4}       min = i;
{5}      for(j = i+1; j <n ; j++)
{6}         if (a[j ] < a[min])
{7}            min=j;
{8}      Swap(a[min],a[i]);
{9}    }
}
```

4.5. Quy tắc tính độ phức tạp giải thuật

- ***Hàm không đệ quy***: là thời gian lớn nhất của hàm đang xét và các hàm được gọi trong hàm đang xét.
- ***Hàm đệ quy***: *sẽ được giới thiệu sau.*
- ***Quy ước cách tính*** *O(n)* ***từ*** *T(n)*
  - Giữ lại giá trị của n có số mũ cao nhất (X)
  - Xóa hằng số trong X để có O(n)
  - Ví dụ

$T(n)= 4n^3 + 7n - 5$ *rút gọn* $T(n^3) \Rightarrow O(n^3)$

$T(n)= 2n^2 + 2n + 6$ *rút gọn* $T(n^2) \Rightarrow O(n^2)$

$T(n)= 1/2n^3 + 4n^2 + 3n + 9$ *rút gọn* $T(n^3) \Rightarrow O(n^3)$

4.6. Một số ví dụ

- **Ví dụ 1**: Tính độ phức tạp của hàm cho người dùng nhập giá trị cho các phần tử của mảng số nguyên A, gồm n phần tử

```
void TaoMang (int A[], int n)
{
    {1}   for(int i=0;i<n;i++)
    {2}   {
    {3}       printf("Nhap so: ");
    {4}       scanf("%d",&n);
    {5}   }
}
```

4.6. Một số ví dụ

- **Ví dụ 2:** Tính độ phức tạp của hàm in các số lẻ có trong mảng số nguyên A, gồm n phần tử

```
void InSoLe(int A[], int n)
{
    {1}   for(int i=0;i<n;i++)
    {2}   {
    {3}       if(A[i]%2!=0)
    {4}           printf("%5d",A[i]);
    {5}   }
}
```

4.6. Một số ví dụ

- **Ví dụ 3**: Xét hàm sắp xếp sau:

```
void SelectionSort(int a[],int n)
{    int  min,i,j;
{1}  for(i=0;i<n-1;i++)
{2}  {    min = i;
{3}       for(j = i+1; j <n ; j++)
{4}            if (a[j ] < a[min])
{5}                 min=j;
{6}       Swap(a[min],a[i]);
     }
}
void Swap (int &a, int &b)
{
{7}  int temp=a;
{8}  a=b;
{9}  b=temp;
}
```

4.6. Một số ví dụ

- **Ví dụ 4**: Phân tích thời gian thực hiện của chương trình sau:

```
void Tinh(int A[], int n)
{       long i, j, n, s1, s2;
    {1} scanf("%ld",&n);
    {2} s1=0;
    {3} for (i=1; i<=n; i++)
    {4}     s1+= i;
    {5} s2=0;
    {6} for (j=n;j>=1;j--)
    {7}     s2+= j*j;
    {8} printf("1+2+..+%ld=%ld",n,s1);
    {9} printf("1^2+2^2+..+%ld^2=%ld",n,s2);
}
```

4.6. Một số ví dụ

- **Ví dụ 5:** Tính độ phức tạp của hàm tính P, với:

$$P = 1! + 2! + 3! + ... + n!$$

```
long TinhP(int n)
{
{1} long P=0;
{2} for(int i=1;i<=n;i++)
{3} {
{4}     long t=1;
{5}     for(int j=1;j<=i;j++)
{6}        t=t*j;
{7}     P=P+t;
{8} }
{9} return P;
}
```

```
long TinhP(int n)
{
{1}   long P=0;
{2}   long t=1;
{3}   for(int i=1;i<n;i++)
{4}   {
{5}        t=t*i;
{6}        P=P+t;
{7}   }
{8}   return P;
}
```

4.6. Thực hành

Bài toán gà chó. Nội dung bài toán như sau:

Vừa gà vừa chó
Bó lại cho tròn
Đúng ba sáu con
Một trăm chân chẵn.

i. Viết chương trình tìm tất cả các trường hợp bài toán có nghiệm (vét cạn). Trong mỗi trường hợp, cho biết số lượng gà, số lượng chó?
ii. Có thể cải tiến chương trình đang có để giảm độ phức tạp?

4.6. Thực hành Bài toán gà chó. Nội dung bài toán như sau:

Vừa gà vừa chó
Bó lại cho tròn
Đúng ba sáu con
Một trăm chân chẵn.

i. Viết chương trình tìm tất cả các trường hợp?
ii. Có thể cải tiến chương trình đang có để giảm độ phức tạp?

```
void main()
{  int ga, cho;
   printf("Dap an: ");
   for (cho=0;cho<=36;cho++)
      for (ga=0;ga<=36;ga++)
         if ((ga*2 + cho*4 == 100) &&(ga+cho==36))
            printf("SL Ga=%d, SL Cho=%d\n", ga, cho);
}
```

```
void main()// chương trình cải tiến
{  int ga, cho;
   printf("Dap an: ");
   for (cho=0;cho<= 25 ;cho++)
   {  ga=36-cho;
      if (ga*2 + cho*4==100)
        printf("SL Ga=%d, SL Cho=%d\n", ga, cho);
   }
}
```

4.6. Thực hành Bài toán gà chó. Nội dung bài toán như sau:

Vừa gà vừa chó
Bó lại cho tròn
Đúng ba sáu con
Một trăm chân chẵn.

i. Viết chương trình tìm tất cả các trường hợp?
ii. Có thể cải tiến chương trình đang có để giảm độ phức tạp?

Nhắc lại: Cho hệ phương trình 2 ẩn số

$$\begin{cases} ax+by=c \\ dx+ey=f \end{cases}$$

Sử dụng định thức để giải hệ phương trình trên:

B1: tính D, Dx, Dy, với:

$$D=\begin{vmatrix} a & b \\ d & e \end{vmatrix} = ae - db \;;\quad Dx=\begin{vmatrix} c & b \\ f & e \end{vmatrix} = ce - fb \;;\quad Dy=\begin{vmatrix} a & c \\ d & f \end{vmatrix} = af - dc$$

B2:

Nếu D!=0 $\Rightarrow$ x = Dx / D; y = Dy / D

Ngược lại nếu Dx!=0 hoặc Dy !=0 $\Rightarrow$ PT vô nghiệm

Ngược lại, $\Rightarrow$ PT vô định (vô số nghiệm)

4.6. Thực hành Bài toán gà chó. Nội dung bài toán như sau:

Vừa gà vừa chó
Bó lại cho tròn
Đúng ba sáu con
Một trăm chân chẵn.

Thay các giá trị tương ứng vào bài toán đang xét

$$\begin{cases} ax+by=c \\ dx+ey=f \end{cases}$$

Ta có hệ phương trình của bài toán là: $\begin{cases} g+c=36 \\ 2*g+4*c=100 \end{cases}$

B1: tính D, Dx, Dy, với:

$$D=\begin{vmatrix} a & b \\ d & e \end{vmatrix} = ae - db \;;\quad Dx=\begin{vmatrix} c & b \\ f & e \end{vmatrix} = ce - fb \;;\quad Dy=\begin{vmatrix} a & c \\ d & f \end{vmatrix} = af - dc$$

$$D=\begin{vmatrix} 1 & 1 \\ 2 & 4 \end{vmatrix} = 4-2=2$$

$$Dx=\begin{vmatrix} 36 & 1 \\ 100 & 4 \end{vmatrix} = 144-100=44 \;;\quad Dy=\begin{vmatrix} 1 & 36 \\ 2 & 100 \end{vmatrix} = 100-72=28$$

B2:

Do D=2!=0 $\Rightarrow$ g = Dx / D = 44/2 = 22; c = Dy / D = 28/2 = 14

4.6. Thực hành Bài toán gà chó. Nội dung bài toán như sau:

Vừa gà vừa chó
Bó lại cho tròn
Đúng ba sáu con
Một trăm chân chẵn.

Chương trình

```
void main()
{
	int D, Dg, Dc;
	int a=1, b=1, c = 36;
	int d = 2, e = 4, f = 100;
	D = a*e - d*b;
	if (D != 0)
	{
		Dg = c*e - f*b;
		Dc = a*f - d*c;
		printf("SL Ga = %d, SL Cho = %d\n", Dg/D, Dc/D);
	}
	else
		printf("He phuong trinh vo nghiem");
	getch();
}
```

4.7. Bài tập

Cài đặt hàm và tính độ phức tạp cho các hàm sau sao cho độ phức tạp là thấp nhất có thể

4.7.1. Bài toán trăm trâu. Nội dung bài toán như sau:

Trăm trâu trăm cỏ

Trâu đứng ăn năm

Trâu nằm ăn ba

Lụ khụ trâu già

Ba con một bó

Xác định số lượng trâu mỗi loại?

4.7.2. Tìm những tam giác có 3 cạnh là số nguyên có kích thước nằm trong khoảng từ 1-10 và 3 cạnh đó tạo thành tam giác vuông?

4.7. Bài tập

Cài đặt hàm và tính độ phức tạp cho các hàm sau sao cho độ phức tạp là thấp nhất có thể

4.7.3. Tính

a/- $$S = x + \frac{x^2}{1+2} + \frac{x^3}{1+2+3} + \ldots + \frac{x^n}{1+2+3+\ldots+n}$$

b/- S = (1) + (1+2) +(1+2+3) + (1+2+3+4) + (1+2+3+4+5) + … + (1+2+3+… +n)

c/- S = (1) + (1x2) + (1x2x3)+ . . . +(1x2x3x. . . xn)

4.7. Bài tập

Cài đặt hàm và tính độ phức tạp cho các hàm sau sao cho độ phức tạp là thấp nhất có thể

4.7.4. Giả sử gọi **X** là số may mắn khi **X** chỉ chứa các ký số 6 hoặc 8 (VD: 6, 86, 868, 6868,…). Viết chương trình cho nhập số nguyên dương **n**. In ra số may mắn nhỏ hơn hoặc bằng và gần với **n** nhất sao cho độ phức tạp là thấp nhất. (*trích trong đề thi ACM-ICPC năm 2018, khu vực miền Nam*)

4.7.5. Cho một dãy số gồm **n** số nguyên dương, xác định xem có tồn tại một dãy con liên tiếp có tổng bằng **k** hay không?

4.7. Bài tập

Cài đặt hàm và tính độ phức tạp cho các hàm sau sao cho độ phức tạp là thấp nhất có thể

4.7.6. Viết chương trình cho nhập số nguyên dương n. Tìm X sao cho:

- X là bội số nhỏ nhất của n.
- X chỉ chứa các ký số 0 hoặc 9.

Ví dụ

```
n= 15 ⇒ X=90  (vì 90/15=6)
n=123⇒ X=99999 (vì 99999/123=813)
n=230⇒ X=9909090 (vì 9909090/230=430833)
```


Q&A
How
Why
When
Where
What
Who