# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giảng viên : Hồ Sĩ Đàm
Bộ môn Mạng và truyền thông máy tính
Trường ĐH Công Nghệ - ĐH Quốc Gia Hà Nội
Email damhs@vnu.edu.vn
Mob. 0913580373

# Giới thiệu môn học

- Cung cấp :

  - Các kiến thức cơ bản về cấu trúc dữ liệu và thuật toán;

  - Kĩ năng xây dựng, lựa chọn các cấu trúc dữ liệu và các thuật toán hợp lí.

# Giới thiệu môn học

- Chương I : Thuật toán và phân tích thuật toán
- Chương II : Đệ quy
- Chương III : Các dữ liệu có cấu trúc
- Chương IV : Danh sách
- Chương V : Cây
- Chương VI * : Bảng băm
- Chương VII : Sắp xếp
- Chương VIII : Tìm kiếm
- Chương IX : Đồ thị
- Chương X : Các kỹ thuật thiết kế thuậ toán

# Tài liệu tham khảo


- Thomas H. Cormen, Introduction to Algorithms, MIT Press, 1990
- R. Sedgevick,Algorithms Addison- Wesley, Bản dịch tiếng Việt: Cẩm nang thuật toán ( tập 1, 2).
- Hồ Sĩ Đàm, Nguyễn Việt Hà, Bùi Thế Duy
- Đinh Mạnh Tường, Đỗ Xuân Lôi

# CHƯƠNG I: THUẬT TOÁN VÀ PHÂN TÍCH THUẬT TOÁN

1. Giải bài toán trên máy tính
2. Mô hình dữ liệu
3. Cấu trúc dữ liệu
4. Bài toán và thuật toán

## *1 Giai bài toán trên máy tính*

**Bước 1.** Xác định bài toán

-Tập Input và Output

**Bước 2.** Lựa chọn/ thiết kế thuật toán

a) Lựa chọn/ thiết kế thuật toán

- Giải bài toán ← nhiều thuật toán
- Không gian ? Thời gian ?; Cài đặt ?

## *1. Giải bài toán trên máy tính*

### b) Diễn tả thuật toán

- *Input:* Hai số nguyên dương a và b;
- *Output:* q và r : a= bq+r.
- *Ý tưởng:*

  - Nếu a < b thì q = 0 và r = a. Kết thúc.

  - Nếu a > b thì a giảm đi b và q tăng lên 1. Lặp cho đến khi a < b.

## *1. Giải bài toán trên máy tính*

***) Cách liệt kê***

*Bước 1: Nhập a và b;*

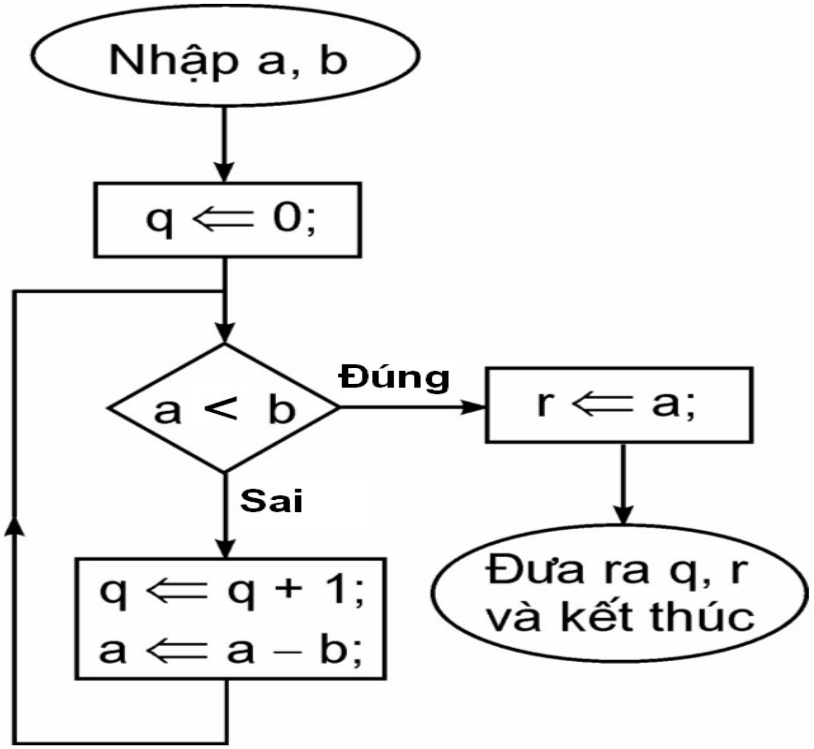
*Bước 2: q ⇐0;*

*Bước 3: Nếu a < b thì r ⇐a rồi chuyển đến b. 5;*

*Bước 4: a ⇐a - b, q ⇐q + 1 rồi quay về b.3;*

*Bước 5: Đưa ra r và q. Kết thúc.*

***) Sơ đồ khối***



## *1. Giải bài toán trên máy tính*

- Chọn CTDL
- Ngôn ngữ lập trình

**Bước 4.** Hiệu chỉnh

- Xây dựng các bộ input (test) tiêu biểu
- Chạy thử

```
Unsigned int Factorial (unsigned int n)
{
    if (n==0)
        return 1;
Else
        return n* Factorial (n-1);
}
```

**Bước 5.** Viết tài liệu

- Hướng dẫn sử dụng
- Thuật toán, Cấu trúc dữ liệu
- …….

## *2. Mô hình dữ liệu (Data model)*

- Là các trừu tượng :đồ thị, tập hợp, danh sách, cây...
- Hai khía cạnh:
  - Giá trị (kiểu dữ liệu)
  - Các phép toán ( operation)
- Chương trình có thể truy xuất đến các vùng lưu trữ.

## *3. Cấu trúc dữ liệu (Data structures)*

- Là các đơn vị cấu trúc (construct) của NNLT dùng để biểu diễn các mô hình dữ liệu

Ví dụ: mảng, bản ghi, set, file, xâu,..

## 4.1. Bài toán

Xác định rõ Input và Output

***Ví dụ:***

*Kiểm tra xem N có phải là số nguyên tố hay không?*

- *Input* : Số nguyên dương N
- *Output* : Trả lời N là số nguyên tố?

**4.2.** *Thuật toán*

*là một dãy hữu hạn các thao tác, sắp xếp theo một trật tự xác định, sau khi thực hiện, từ Input ta nhận được Output cần tìm.*

## *4. Bài toán và thuật toán*

**Ví dụ**

**- Input:N** nguyên dương, dãy $a_1,..., a_n$.

**- Output :** Tìm Max của dãy đã cho.

**Ý tưởng**

- Giả thiết Max = $a_1$. Với mỗi i, nếu $a_i$ > Max thì thay giá trị Max= $a_i$.

## 4.3. Mô tả thuật toán

### a) Cách liệt kê

*Bước 1*. Nhập N và dãy $a_1, ..., a_n$

*Bước 2*. Đặt Max = $a_1$, i = 2;

*Bước 3*. Nếu i > N thì đến B. 5;

*Bước 4*.

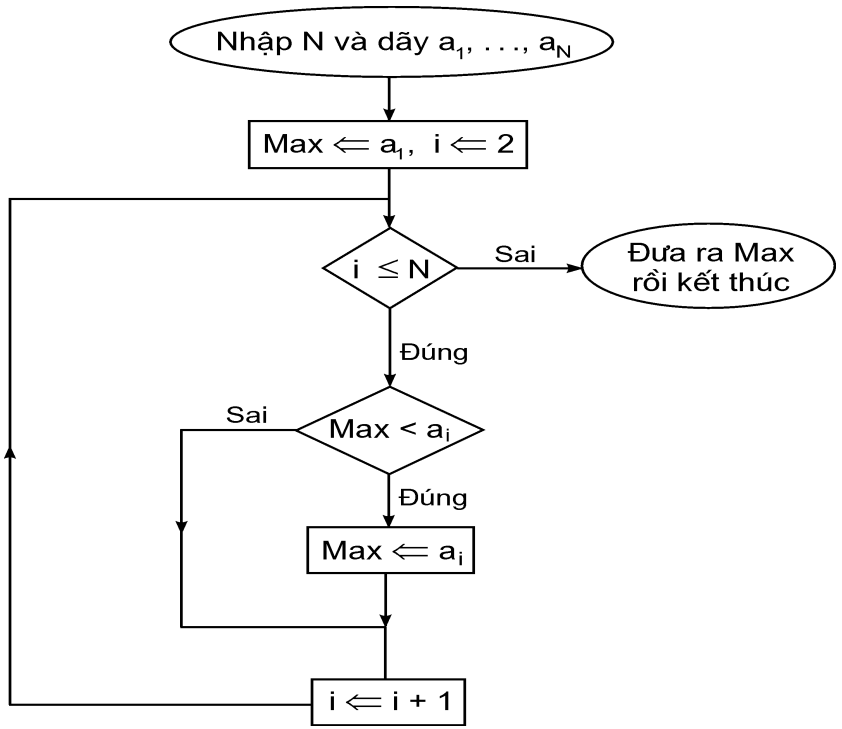
4.1. N ếu $a_i$ > Max thì Max = $a_i$.

4.2. Đặt i=i+1 rồi quay B.3;

## *4. Bài toán và thuật toán*

### b) Sơ đồ khối

- Dùng: Ovan, Chữ nhật, Hìn thoi,Mũi tên,…



**c)**

## Ngôn ngữ điều khiển

- Dùng các ký hiệu và quy tắc
- Cách thiết lập thứ tự các thao tác

*cấu trúc điều khiển ( 03 )*

# BIỂU DIỄN BẰNG CẤU TRÚC ĐIỀU KHIỂN

Trong khi m ≠ n thì lặp lại khối sau:

Nếu m > n thì

Bớt m đi một lượng là n

Nếu ngược lại thì

Bớt n đi một lượng là m

Cho tới khi m = n thì tuyên bố USCLN chính là giá trị chung của m và n

```
Int Horner(int a[],int x)
{
   int result = a[n];
   for (i=n-1; i>=0;--1)
  result= result*x+a[i];
  return result;
}
```

Chương trình trong C++

# 4.4. Các đặc trưng chính của thuật toán

a) *Tính kết thúc (tính đóng)*

b) *Tính xác định (đơn nghĩa)*

Có đúng một thao tác để được thực hiện hoặc dừng

## *4. Bài toán và thuật toán*

c) *Tính chi tiết*

Phụ thuộc vào đối tượng thực hiện

d) *Tính phổ dụng*

với input thay đổi

e) *Đại lượng vào*

f) *Đại lượng ra*

## *4. Bài toán và thuật toán*

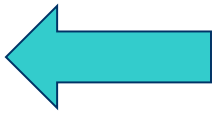
*g) Tính hiệu quả*

- Thời gian: Tốc độ xử lý
- Không gian: Dung lượng lưu trữ

## 4.5. Độ phức tạp thuật toán

### a) *Lựa chọn thuật toán*

- Dễ hiểu, dễ cài đặt và dễ ghi chép ?
- Sử dụng các tài nguyên hiệu quả.?



đặc tính của bài toán

## Phân tích theo kinh nghiệm

- Thực hiện và kết luận → dễ mắc lỗi
- Kích thước dữ liệu là quan trọng: T(n)

## *4. Bài toán và thuật toán*

*b) Ký pháp*

Giả sử T(n) là thời gian thực hiện TT và f(n), g(n), h(n) là các hàm xác định dương.

Hàm Theta lớn:

T(n) là hàm Theta lớn của g(n): $T(n) = \Theta(g(n))$

nếu $\exists$ các hằng số dương $c_1$ ,$c_2$ ,$n_0$ sao cho với mọi $n >= n_0$ :

$$\mathbf{c_1\, g(n) <= T(n) <= c_2\, g(n)}$$

## *4. Bài toán và thuật toán*

Hàm Omega lớn:

T(n) hàm Omega lớn của g(n): $T(n)=\Omega(g(n))$ nếu $\exists$ c và $n_0$ sao cho với mọi $n >= n_0$

**$T(n) >= c.g(n)$**

## *4. Bài toán và thuật toán*

Hàm O lớn:

- T(n) hàm Omega lớn của g(n), T(n) =O (g(n))

  nếu ∃ c và $n_0$ sao cho với mọi n>= $n_0$ :

  **T(n) <=c g(n)**

- g(n) giới hạn trên của T(n).

Ví dụ, nếu $T(n) = n^2 + 1$ thì $T(n) = O(n^2)$.

Chọn $c=2$ và $n_0 = 1$, khi đó với mọi $n>=1$, ta có $T(n)= n^2+1 <= 2n^2 =2g(n)$.

## *4. Bài toán và thuật toán*

### c) *Các tính chất*

(i) Tính bắc cầu: nếu f(n)= O(g(n)) và g(n)= O(h(n)) thì f(n)= O(h(n))

(ii) Tính phản xạ: f(n)=O(f(n))

## *4. Bài toán và thuật toán*

### *d) Xác định độ phức tạp*

**Quy tắc hằng số**

Nếu P có T(n)= $O(c_1 f(n))$ ⇒ P có độ phức tạp ***O(f(n))***.

**CM**: T(n)= $O(c_1 f(n))$ nên tồn tại $c_0>0$ và $n_0>0$ để $T(n) <= c_0.c_1 f(n)$ với mọi $n>= n_0$.

Đặt $c=c_0.c_1$ ta có điều cần CM

## *4. Bài toán và thuật toán*

- **Quy tắc lấy Max**

  Nếu P có T(n)= O( f(n)+g(n)) thì P có độ phức tạp là ***O( max ( f(n), g(n)))***.

**CM**: T(n) = O( f(n)+g(n)) nên tồn tại $n_0$>0 và c>0 để T(n) <= cf(n) + cg(n), với mọi n>= $n_0$ vậy T(n) <= cf(n) +cg(n) <= 2c max (f(n),g(n)) với mọi n>=$n_0$.

Từ đó suy điều cần CM.

## *4. Bài toán và thuật toán*

- **Quy tắc cộng**

  Nếu P1 có T1 (n) = O( f(n) và P2 có T2(n)= O(g(n)), khi đó: ***T1(n) +T2(n) = O(f(n) +g(n))***.

**CM**: Vì T1(n)= O(f(n)) nên ∃ các hàng số $c_1$ và $n_1$ sao cho T(n) <= $c_1$.f(n) ∀n: n>= $n_1$.

Vì T2(n) =O(g(n)) nên ∃ các hàng số $c_2$ và $n_2$ sao cho T(n) <= $c_1$.g(n) ∀n: n>= $n_2$

Chọn c= max ($c_1$,$c_2$) và $n_0$ = max($n_1$,$n_2$) ta có ∀n: n n>= $n_0$:

T(n) = T1(n) + T2(n) <= $c_1$f(n) + $c_2$g(n)

<= cf(n) +cg(n) = c(f(n)+g(n)).

## *4. Bài toán và thuật toán*

- **Quy tắc nhân**

  Nếu P có T(n)= O(f(n)). Khi đó nếu thực hiện k(n) lần P với k(n)=O(g(n)) thì độ phức tạp la ***O(f(n) g(n))***.

**CM**: Thời gian thực hiện k(n) lần đoạn chương trình P sẽ là k(n) T(n), theo định nghĩa:

$\exists c_k >= 0$ và $n_k > 0$ để $k(n) <= c_k(g(n))$ với mọi $n >= n_k$

$\exists c_T >= 0$ và $n_T > 0$ để $T(n) <= c_T f(n)$ với mọi $n >= n_T$

Vậy với mọi $n >= \max(n_T, n_k)$ ta có $k(n)T(n) <= c_k c_T(f(n)g(n))$.

## *4. Bài toán và thuật toán*

*e) Áp dụng đánh giá chương trình*

- Câu lệnh đơn thực hiện một thao tác → QT hằng số
- Câu lệnh hợp thành là dãy các câu lệnh → QT tổng
- Câu lệnh rẽ nhánh dạng If ..then..else. → QT Max
- Các câu lệnh lặp → QT Nhân

# 4. Bài toán và thuật toán

Ví dụ 1

## *Analysing an Algorithm*

- **Simple statement sequence**

  $s_1$; $s_2$; …. ; $s_k$

  - $O(1)$ as long as $k$ is constant
- **Simple loops**

  `for(i=0;i<n;i++) { s; }`

  where s is $O(1)$

  - Time complexity is $n\ O(1)$ or $O(n)$
- **Nested loops**

  ```
  for(i=0;i<n;i++)
    for(j=0;j<n;j++) { s; }
  ```

  - Complexity is $n\ O(n)$ or $O(n^2)$

This part is $O(n)$

# 4. Bài toán và thuật toán

Ví dụ 2

## *Analysing an Algorithm*

- Loop index doesn't vary linearly

```
h = 1;
while ( h <= n ) {
      s;
      h = 2 * h;
      }
```

  - h takes values 1, 2, 4, … until it exceeds $n$
  - There are $1 + \log_2 n$ iterations
  - Complexity $O(\log n)$

# 4. Bài toán và thuật toán

Ví dụ 3

## Analysing an Algorithm

- Loop index depends on outer loop index

```
for(j=0;j<n;j++)
    for(k=0;k<j;k++){
      s;
      }
```

  - Inner loop executed
    - 1, 2, 3, …., n times

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

∴ Complexity $O(n^2)$

Distinguish this case - where the iteration count increases (decreases) by a constant ⇾ $O(n^k)$ from the previous one - where it changes by a factor ⇾ $O(\log n)$

## *4. Bài toán và thuật toán*

### *Một số dạng hàm*

- Đa thức bậc k: P(n), O $(n^k)$.
- $\log_a f(n)$, O$(\log f(n))$.
- Hằng số, O$(1)$
- Hàm mũ O($2^n$.)

## *4. Bài toán và thuật toán*

| Lgn | | n | nlgn | $n^2$ | $n^3$ | $2^n$ |
|---|---|---|---|---|---|---|
| 0 | | 1 | 0 | 1 | 1 | 2 |
| 1 | | 2 | 2 | 4 | 8 | 4 |
| 2 | | 4 | 8 | 16 | 64 | 16 |
| 3 | | 8 | 24 | 64 | 512 | 256 |
| 4 | | 16 | 64 | 256 | 4096 | 65536 |
| 5 | | 32 | 160 | 1024 | 32768 | 2147483648 |

## *g) Độ phức tạp tính toán và dữ liệu vào*

- Trường hợp tốt nhất: T(n) là thời gian ít nhất
- Trường hợp xấu nhất: T(n) là thời gian lớn nhất
  Trường hợp trung bình: dữ liệu vào tuân theo một phân bố xác suất nào đó

## *h) Phép toán tích cực*

- Các phép toán thực hiện nhiều nhất
- Quy tắc ‘10-90’

# CHƯƠNG II : ĐỆ QUY VÀ THUẬT TOÁN ĐỆ QUY

1. Phép lặp, quy nạp và đệ quy
2. Thuật toán đệ quy

## *1. Phép lặp, quy nạp và đệ quy*

- **Lặp (interation): biến thể của cùng một thao tác.**
- **Quy nạp(induction): kĩ thuật chứng minh các mệnh đề thuộc dạng với mọi n thì P(n) là đúng.**

  Ví dụ: với mọi n, tổng n số lẻ đầu tiên bằng $n^2$.

  Bước cơ sở: Chỉ ra P(1) là đúng , vì $1^2=1$.

  Bước quy nạp: Chứng minh nếu P(n) là đúng thì kéo theo P(n+1) cũng đúng

## *1. Phép lặp, quy nạp và đệ quy*

Tổng n số lẻ là $n^2$, cần cm tổng (n+1) số lẻ là $(n+1)^2$.

Tổng n số lẻ: $1+3+5+\ldots.+(2n-1)= n^2$.

Khi đó: $[1+3+5+\ldots.+(2n-1)] +(2n+1) = n^2+2n+1= (n+1)^2$.

## *1. Phép lặp, quy nạp và đệ quy*

- Đệ quy (recursion): là một kĩ thuật định nghĩa một khái niệm trực tiếp hoặc gián tiếp theo chính nó.

## CHƯƠNG II :
## ĐỆ QUY VÀ THUẬT TOÁN ĐỆ QUY
### *2. Thuật toán đệ quy*

# a) Định nghĩa

- Nếu lời giải P được thực hiện bằng lời giải P’ có dạng giống như P thì ta nói đó là lời giải đệ quy → thuật toán đệ quy
- Chú ý:
  - P’ giống P
  - P’ “nhỏ hơn” P theo nghĩa nào đó

## HƯƠNG II :
## ĐỆ QUY VÀ THUẬT TOÁN ĐỆ QUY
### *2. Thuật toán đệ quy*

- Định nghĩa một hàm hay thủ tục đệ quy gồm hai phần:

  (i) Phần neo (anchor)

  (ii) Phần đệ quy

  ***Phần đệ quy thể hiện tính "quy nạp" của lời giải. Phần neo đảm bảo cho tính dừng của thuật toán.***

b) Ví dụ

**Tính giai thừa: N!= N(N-1)!**

```
Int fact ( int n) {
    If  ( n <= 1 )
        Return 1; /* cơ sở*/
    Else
        Return n*fact (n-1); /* quy nạp*/
}
```

## *2. Thuật toán đệ quy*

b) Ví dụ:

**Dãy số Fibonacci**:

F(n)= F(n-1)+F(n-2) (phần đệ quy)

- với n<=2 thì F(n)=1 ( phần neo).

***Recursion - Example***

- Fibonacci Numbers

*Pseudo-code*

```
fib( n ) = if ( n = 0 ) then 1
           else if ( n = 1 ) then 1
           else fib(n-1) + fib(n-2)
```

*C*

```
int fib( n ) {
     if ( n < 2 ) return 1;
     else return fib(n-1) + fib(n-2);
     }
```

*Simple, elegant solution!*

## *2. Thuật toán đệ quy*

- **Bài toán tháp Hà nội** : Ngôi đền Benares có 03 cái cọc kim cương, Thượng đế đặt n chiếc đĩa bằng vàng:

  - Có bán kính khác nhau

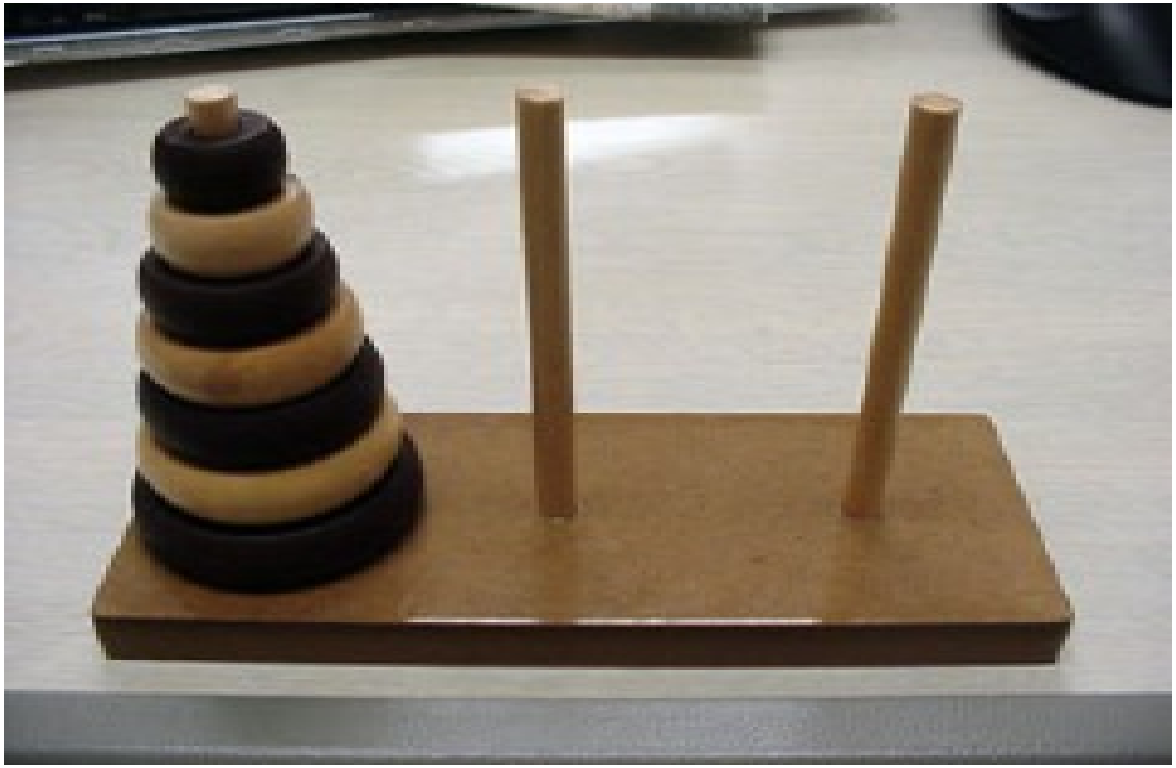
  - Chồng lên nhau ở một chiếc cọc

  - Theo thứ tự đĩa lớn ở dưới, đĩa nhỏ ở trên. Các nhà sư lần lượt chuyển các đĩa sang một cọc khác theo quy tắc sau:

## *2. Thuật toán đệ quy*

- Khi chuyển một đĩa phải đặt vào một trong 03 cọc
- Mỗi lần chỉ chuyển đúng một đĩa trên cùng tại một cọc và đặt vào trên cùng ở cọc chuyển đến.
- Đĩa lớn hơn không được phép đặt lên đĩa nhỏ hơn.

# Tháp Hà nội



## *2. Thuật toán đệ quy*

- Ví dụ, với trường hợp n=2 ta có thể chuyển như sau:
- Chuyển đĩa nhỏ sang cọc 3
- Chuyển đĩa lớn sang cọc 2
- Chuyển đĩa từ cọc 3 sang cọc 2

## *2. Thuật toán đệ quy*

- Nếu n=1 thì chuyển đĩa duy nhất đó từ cọc 1 sang cọc 2. Kết thúc.

Giả thiết ta có cách chuyển (n-1) đĩa từ cọc 1 sang cọc 3.

- Chuyển đĩa lớn nhất đang ở cọc 1 sang cọc 2
- Chuyển (n-1) đĩa từ cọc 3 sang cọc 2.
- Kết thúc.

## *2. Thuật toán đệ quy*

c) Đánh giá về đệ quy

Ưu điểm:

- Mạnh, rõ ràng, chặt chẽ
- Thiết kế TT đơn giản

## *2. Thuật toán đệ quy*

Nhược điểm:

- Lời gọi hàm tốn rất nhiều thời gian.
- Thận trọng đừng để chạy vô hạn.

# CHƯƠNG III :KiỂU DỮ LIỆU CẤU TRÚC

1. Các kiểu dữ liệu chuẩn
2. Kiểu mảng (array) và biến chỉ số
3. Kiểu xâu (string)
4. Kiểu cấu trúc (struct)

## *1. Các kiểu dữ liệu chuẩn*

- Kiểu dữ liệu chuẩn: nguyên, thực, lôgic,
- Ví dụ,
  - Khai báo biến int x, float y,….
  - Với kiểu nguyên có các phép toán: cộng, trừ, nhân, chia x div y ( chia nguyên ), x mod y ( lấy phần dư của phép chia).

## *2. Kiểu mảng (array) và biến chỉ số*

# 2.1. Kiểu mảng một chiều

- Dãy liên tiếp các phần tử cùng kiểu.
- Đặt tên và mỗi phần tử  có một chỉ số.
- Tham chiếu tên mảng và chỉ số **[ ]** .
- Số lượng các phần tử cho trước.

## *2. Kiểu mảng (array) và biến chỉ số*

- ***Ví dụ***

  int a[10] : mảng có tên là a, có 10 phần tử có kiểu số nguyên

  float b[20] : mảng có tên là b, có 20 phần tử có kiểu số thực

## *2. Kiểu mảng (array) và biến chỉ số*

### 2.2. Kiểu mảng hai chiều

- Bảng  nxm các phần tử cùng kiểu dữ liệu.
- Tham chiếu

  Tên mảng cùng với hai chỉ số, **[ ]**.

## *2. Kiểu mảng (array) và biến chỉ số*

- ***Ví dụ***: int a[10][15] :
  float b[20][10]:

## 2.3. Các ghi chú về kiểu mảng

- Địa chỉ các phần tử là liên tiếp nhau.
- Các phần tử  được sắp xếp theo hàng.
- Bộ nhớ là cố định suốt cả quá trình

## *2. Kiểu mảng (array) và biến chỉ số*

- Chỉ số mảng không được vượt quá phạm vi  Ví dụ: a[5] * b[8][3] là hợp lệ.
- Cấu trúc đơn giản, truy nhập nhanh.
- Thiếu mềm dẻo trong các phép toán như xóa, chèn.
- Có thể dùng phép gán cho cả mảng.

## *3. Kiểu xâu (chuỗi – string)*

- Kiểu mảng đặc biệt mà kiểu phần tử của mảng này là ký tự. Thông thường xâu được đặt trong cặp dấu ‘ và ’,

  Ví dụ: S=‘Tin hoc’
- Các ký tự trong xâu được viết liên tiếp và có thể xuất hiện nhiều lần
- Số lượng các ký tự là độ dài của xâu, có xâu rỗng

## *3. Kiểu xâu (chuỗi – string)*

- Phép toán trên xâu:
  - Ghép xâu: Char*strcat(char* s1, char s2)
  - Copy (S,M,N): giá trị là xâu con của sâu S gồm các ký tự từ vị trí M đến N
  - Delete (S,M,N): xóa liên tiếp N kí tự của S bắt đầu từ kí tự thứ M
  - Phép chèn Insert (S1, S2, N): Chèn xâu S1 vào trước ký tự N của xâu S2

## *4. Kiểu cấu trúc (struct)*

- Tập các đối tượng có cùng một số thuộc tính có thể có các kiểu dữ liệu khác nhau.
- Mỗi đối tượng  mô tả bằng một cấu trúc.
- Mỗi thuộc tính tương ứng với một thành phần .

## *4. Kiểu cấu trúc (struct)*

- Khai báo kiểu cấu trúc:

```
STRUCT tên_kiểu_cấu trúc {
    Khai báo các thành phần
};
```

## *4. Kiểu cấu trúc (struct)*

- Truy nhập:
  - Xác định: viết tên cấu trúc, dấu chấm (.) và sau cùng là tên thành phần.
  - Thành phần của một cấu trúc có thể là một cấu trúc.

# CHƯƠNG IV: DANH SÁCH (LIST)

1. Khái niệm danh sách
2. Biểu diễn danh sách trong máy tính.
3. Một số nhận xét.
4. Kiểu dữ liệu con trỏ (pointer) và việc cấp phát/thu hồi bộ nhớ động.

## *1. Khái niệm danh sách*

a) Định nghĩa

- Danh sách là một tập sắp thứ tự các phần tử cùng kiểu.
- Các phần tử được sắp theo thứ tự "trước- sau"
- Danh sách con gồm các phần tử liên tiếp từ $a_i$ đến $a_j$ của danh sách.
  - Nếu i=1 gọi là phần đầu (prefix)
  - Nếu i=n gọi là phần cuối (postfix).

## *1. Khái niệm danh sách*

- Dãy con là một DS tạo thành bằng cách loại từ DS một số phần tử. Ví dụ, DS = (a,b,c,d,e,f). Khi đó:
  - (c,d,e) là một danh sách con của DS
  - (a, b) là một phần đầu của DS
  - (c,d,e,f) là một phần cuối của DS
  - (a,c,f) là một dãy con của DS

## *1. Khái niệm danh sách*

**b) Các phép toán**

- Phép khởi tạo tạo ra một danh sách rỗng.
- Xác định độ dài
- Xóa
- Chèn
- Tìm kiếm
- Kiểm tra  tính trạng thái rỗng
- Kiểm tra trạng thái tràn
- Duyệt danh sách.
- Sắp xếp
-

## *2. Biểu diễn danh sách trong máy tính*

# 2.1. Cài đặt bằng mảng một chiều

- VD. DS = ( A, B, C, D, E, F, G, H, I, J, K)
- Mảng M gồm 11 phần tử:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K |

## *2. Biểu diễn danh sách trong máy tính*

### 2.1. Cài đặt bằng mảng một chiều

***a) Chèn***

- Dồn các phần tử từ vị trí P đến cuối sang phải một vị trí:

P

| A | B | C | D | E | F | | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|

- Đặt V vào vị trí P

-Tăng n lên 1

## 2. Biểu diễn danh sách trong máy tính

***b) Xóa***

- Mảng ban đầu

P

| A | B | C | D | E | F | G | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|---|

- Chuyển tất cả các phần tử từ vị trí P+1 đến cuối sang trái 1 vị trí

| A | B | C | D | E | F | H | I | J | K | L |
|---|---|---|---|---|---|---|---|---|---|---|

-Giảm n đi 1.

- Nếu không cần bảo lưu thứ tự các phần tử sau khi xóa thì chỉ cần tráo đổi giá trị phần tử cần xóa cho phần tử cuối cùng và giảm n đi 1.

## *2. Biểu diễn danh sách trong máy tính*

### *c) Nhận xét*

- Truy cấp trực tiếp đến mọi phần tử
- Chèn và xóa đều phải dịch chuyển một số các phần tử
- Kích thước mảng trong mọi ngôn ngữ lập trình đều là cố định

## *2. Biểu diễn danh sách trong máy tính*

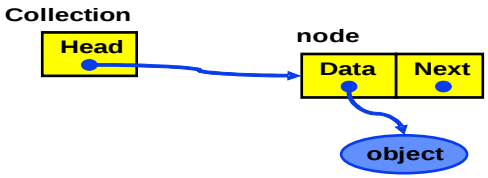
# 2.2. DS nối đơn

- DS các phần tử được nối với nhau theo một chiều.
- Mỗi phần tử là một Struct (bản ghi)
  - Một trường con trỏ chứa thông tin liên kết
  - Các trường chứa thông tin
- Đầu (Head) và Cuối (Tail)
- Trường NEXT của phần tử cuối chứa giá trị đặc biệt (Nill, Null)

# 2. Biểu diễn danh sách trong máy tính

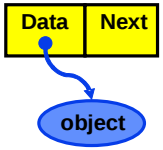
### Linked Lists

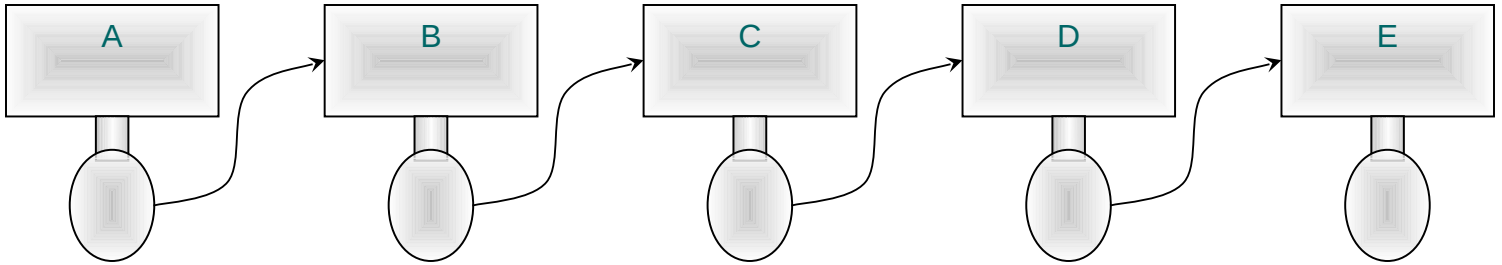
- Collection structure has a pointer to the list head
  - Initially NULL
- Add first item
  - Allocate space for node
  - Set its data pointer to object
  - Set Next to NULL
  - Set Head to point to new node




### Linked Lists

- Flexible space use
  - Dynamically allocate space for each element as needed
  - Include a pointer to the next item
- Linked list
  - Each node of the list contains
    - the data item (an object pointer in our ADT)
    - a pointer to the next node



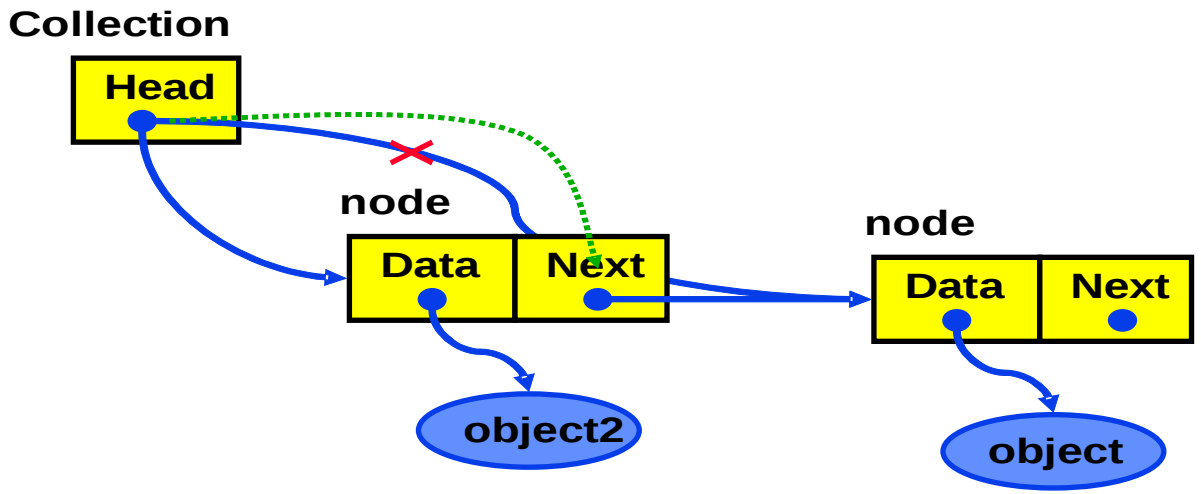



# 2. Biểu diễn danh sách trong máy tính

## a) Chèn

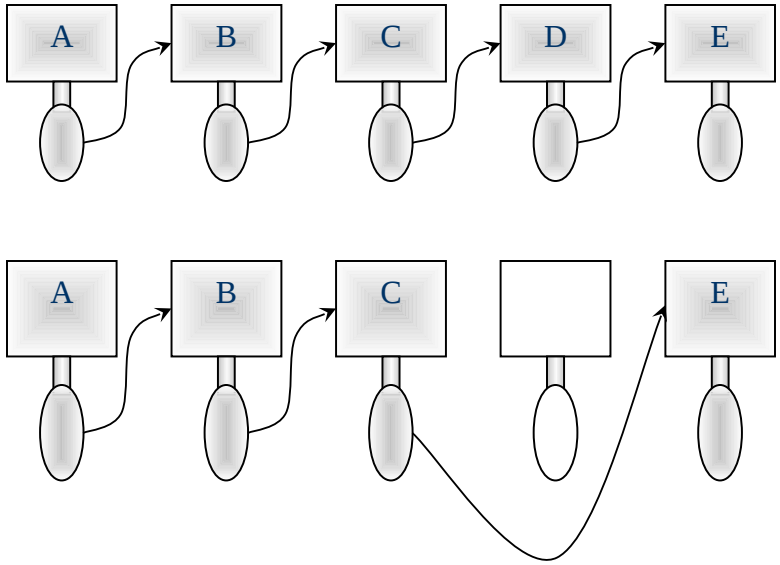
### Linked Lists

- **Add second item**
  - Allocate space for node
  - Set its data pointer to object
  - Set Next to current Head
  - Set Head to point to new node



# 2. Biểu diễn danh sách trong máy tính

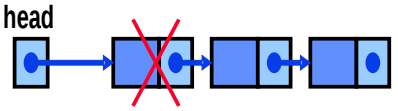
## b) Xóa




### *Linked Lists -* `Delete` *implementation*

- Implementation

```
void *DeleteFromCollection( Collection c, void *key ) {
    Node n, prev;
    n = prev = c->head;
    while ( n != NULL ) {
      if ( KeyCmp( ItemKey( n->item ), key ) == 0 ) {
            prev->next = n->next;
            return n;
            }
      prev = n;
      n = n->next;
      }
    return NULL;
    }
```




Minor addition needed to allow for deleting this one! An exercise!

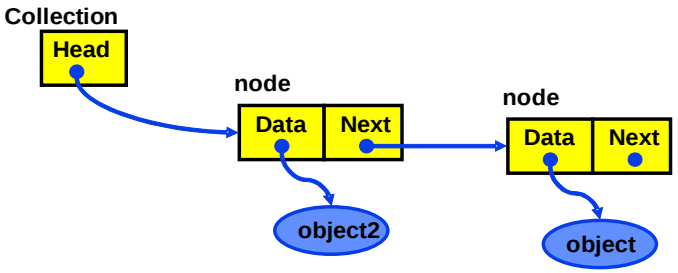
# 2. Biểu diễn danh sách trong máy tính

c) **Xác định**

- Tìm kiếm bắt đầu từ Head, Theo con trỏ của trường NEXT
  → duyệt tuần tự.

*Linked Lists*

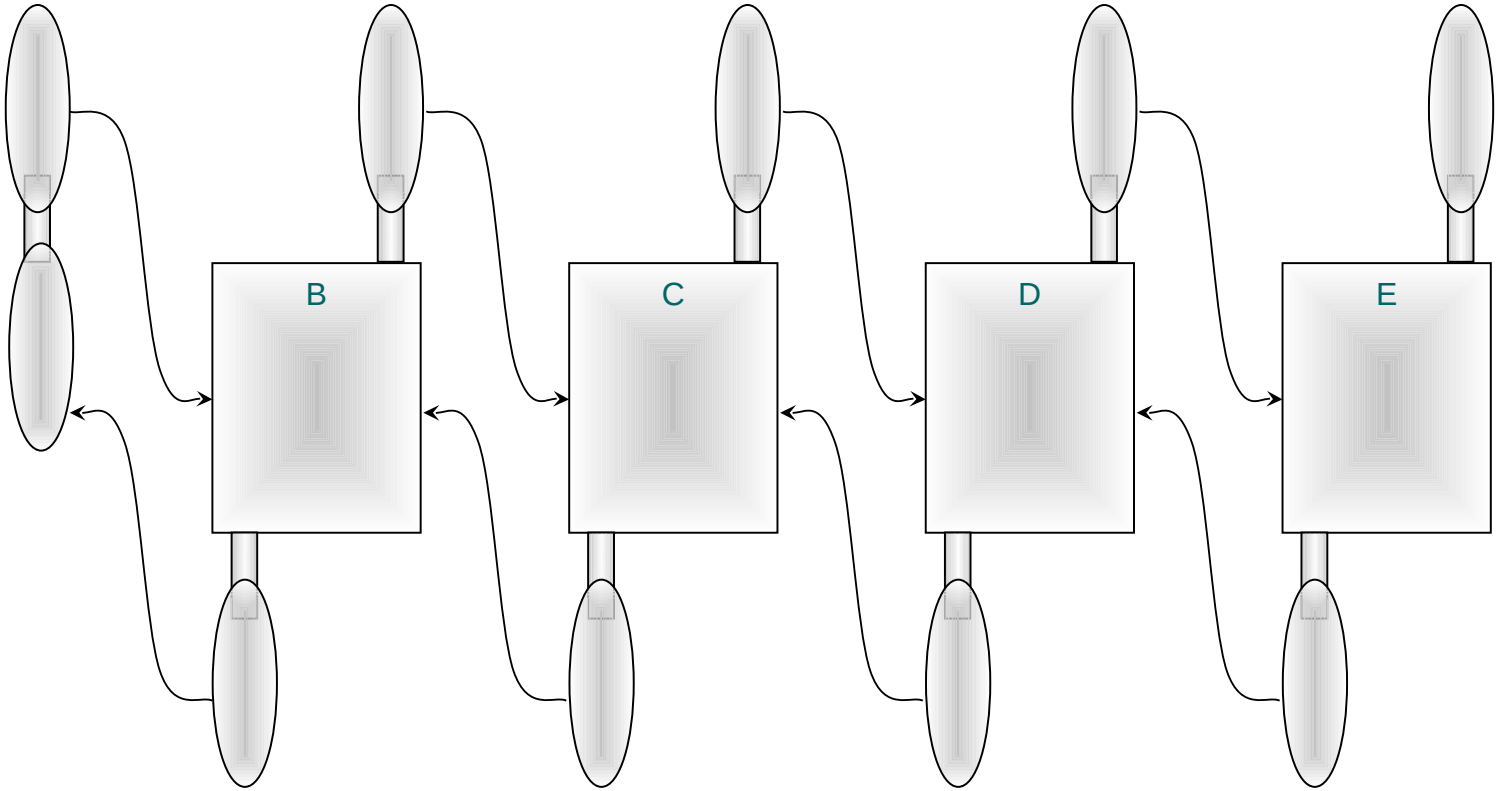
- Add time
  - Constant - independent of n
- Search time
  - Worst case - n




```
void
    .
    while ( n != NULL ) {
      if ( KeyCmp( ItemKey( n->item ), key ) == 0 ) {
            return n->item;
      n = n->next;
      }
    return NULL;
    }
```

# 2.3. DS nối kép

- DS gồm các phần tử được nối với nhau theo hai chiều.
- Mỗi phần tử là một STRUCT có hai trường liên kết (LINK) liên kết tới 2 phần tử trước và sau nó và trường DATA (INFO) lưu trữ thông tin.


B
C
D
E

## *2. Biểu diễn danh sách trong máy tính*

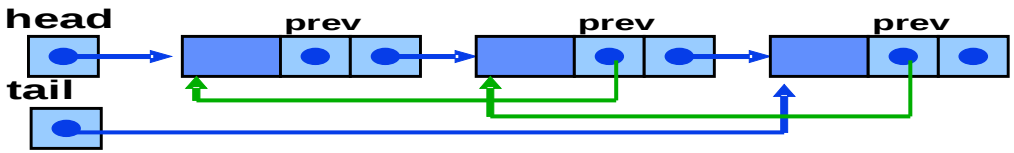
- **Có hai cách duyệt**:
  - Bắt đầu từ nút First, dựa vào NEXT để đi đến khi duyệt qua nút Last;
  - Hoặc bắt đầu từ nút Last dựa vào Prev để đi đến khi duyệt qua nút First.

## *Linked Lists - Doubly linked*

- **Doubly linked lists**
  - **Can be scanned in both directions**

```
struct t_node {
     void *item;
     struct t_node *prev,
                   *next;
     } node;

typedef struct t_node *Node;
struct collection {
     Node head, tail;
     };
```



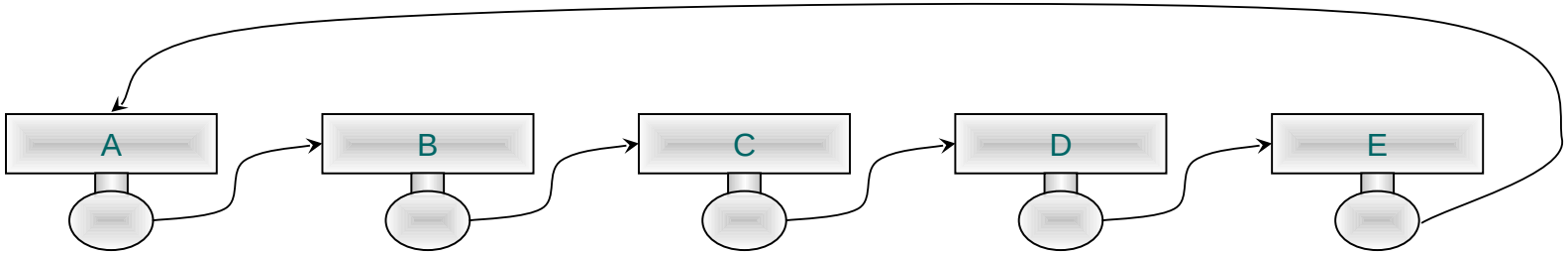
## CHƯƠNG IV: DANH SÁCH (LIST)
### *2. Biểu diễn danh sách trong máy tính*

Chèn/xóa bằng cách chỉnh lại các liên kết của các nút liên quan.

## *2. Biểu diễn danh sách trong máy tính*

# 2.4. DS nối vòng một hướng

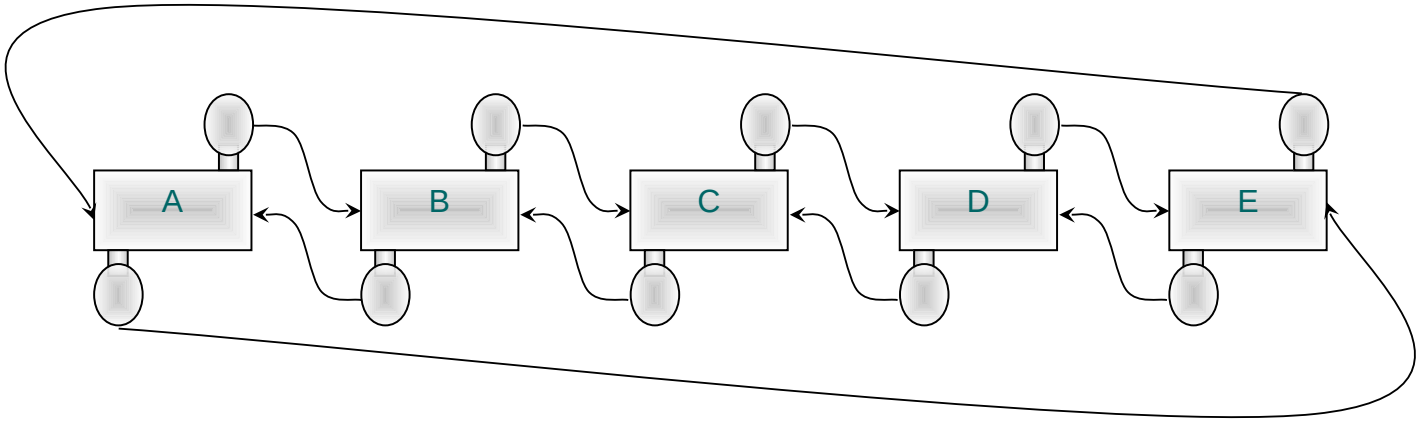
- Phần tử cuối  trỏ về phần tử đầu tiên



- Chỉ cần xác định được một phần tử nào đó
- Chèn/xóa bằng cách chỉnh lại các liên kết của các nút liên quan.

# 2.5. DS nối vòng hai hướng

- Tạo từ DS nối kép, PREV của nút First trỏ tới nút Last, NEXT của nút Last trỏ tới First.



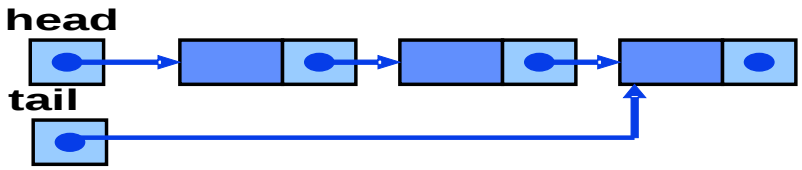
# CHƯƠNG IV: DANH SÁCH (LIST)
## *2. Biểu diễn danh sách trong máy tính*

## 2.6. Ngăn xếp và hàng đợi

### *Linked Lists - LIFO and FIFO*

- **Simplest implementation**
  - Add to head
  - Last-In-First-Out (LIFO) semantics
- **Modifications**
  - First-In-First-Out (FIFO)
  - Keep a tail pointer

```
struct t_node {
     void *item;
     struct t_node *next;
     } node;
typedef struct t_node *Node;
struct collection {
     Node head, tail;
     };
```




tail is set in the AddToCollection method if head == NULL

# CHƯƠNG IV: DANH SÁCH (LIST)
## *2. Biểu diễn danh sách trong máy tính*

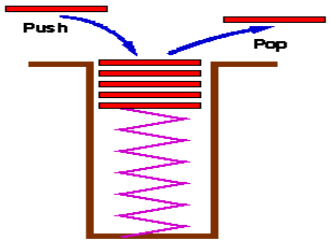
### 2.6. Ngăn xếp và hàng đợi

#### a) *Ngăn xếp (Stack)*

- Một kiểu DS
- Bổ sung thêm và lấy ra một phần tử cũng ở cuối DS.
- Hoạt động theo nguyên tắc “vào sau-ra trước” (LIFO)

## *Stacks*

- Stacks are a special form of collection with LIFO semantics
- Two methods
  - `int push( Stack s, void *item );`
    - add item to the top of the stack
  - `void *pop( Stack s );`
    - remove an item from the top of the stack
- Like a plate stacker
- Other methods




```
int IsEmpty( Stack s );
/* Return TRUE if empty */
void *Top( Stack s );
/* Return the item at the top,
    without deleting it */
```

## *2. Biểu diễn danh sách trong máy tính*

**Mô tả Stack bằng mảng**

(i) Thêm (Push) vào Stack = thêm vào cuối mảng

(ii) Lấy ra (Pop) khỏi Stack = loại bỏ cuối mảng

(iii) Overstack khi mảng đã đầy

(iv) EmptyStack khi trong mảng không có phần tử nào cả.

## *2. Biểu diễn danh sách trong máy tính*

- **Mô tả Stack bằng DS nối đơn_**
  - Stack chỉ tràn khi vùng không gian nhớ dùng cho các biến động không còn đủ
  - Không gian bộ nhớ dùng cho các biến động là rất lớn nên bỏ qua việc kiểm tra tràn Stack.

## *2. Biểu diễn danh sách trong máy tính*

### *b) Hàng đợi (Queue)*

- Một kiểu danh sách
- Thêm một phần tử vào cuối danh sách (Rear) và Lấy ra một phần tử ở đầu danh sách.
- Hoạt động theo nguyên tắc vào trước-ra trước (FIFO - First In First Out).

## *2. Biểu diễn danh sách trong máy tính*

- **Cài đặt  Queue bằng mảng**

  Sử dụng hai chỉ số Front để lưu chỉ số đầu và Rear để lưu chỉ số cuối. Khởi tạo đặt Front là 1 còn Rear là 0.

  (i) Thêm ( Push) vào Queue: tăng Rear lên 1 và đưa giá trị của phần tử cần bổ sung vào phần tử có chỉ số là Rear.

## *2. Biểu diễn danh sách trong máy tính*

ii) Lấy ra (Pop) :lấy giá trị phần tử có chỉ số là Front và sau đó tăng Front lên 1.

(iii) Khi tăng chỉ số Rear lên hết khoảng cho phép thìOverQueue

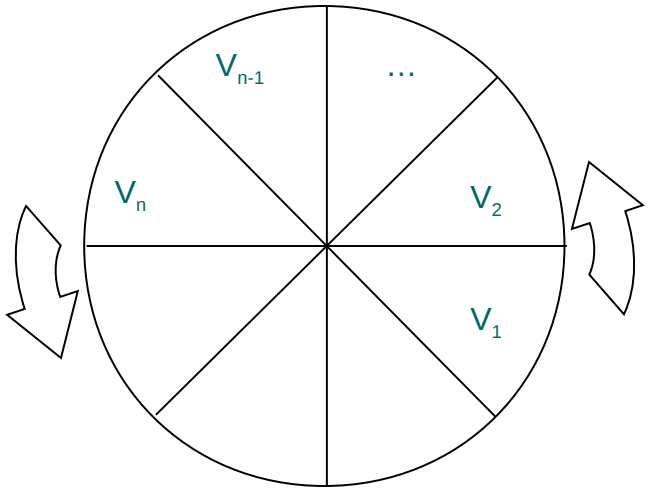
(iv) Khi Front > Rear thì EmptyQueue

## *2. Biểu diễn danh sách trong máy tính*

- **Queue vòng tròn**
  - Các phần tử xếp quanh vòng tròn theo một hướng.
  - Các phần tử nằm trên cung tròn từ vị trí Front đến Rear là thuộc Queue.

# 2. Biểu diễn danh sách trong máy tính

- *Khi thêm :dịch chuyển chỉ số Rear theo vòng tròn 1 ví trí rồi đặt giá trị cần thêm vào đó.*
- *- Khi lấy ra : lấy phần tử có chỉ số là Front  rồi dịch chuyển chỉ số Front theo vòng tròn 1 vị trí.*



## *2. Biểu diễn danh sách trong máy tính*

- **Để cài đặt** :

  (i) Một phương tiện để chia bộ nhớ thành các nút, mỗi nút có phần lưu trữ data, phần lưu trữ các liên kết (con trỏ) và cách cài đặt cho con trỏ

  (ii) Cài đặt các thao tác để truy nhập giá trị (cả data và pointer).

  (iii) Một phương tiện nào đó để "đánh dấu" vùng bộ nhớ

## *3. Một số nhận xét*

- Cài đặt danh sách bằng mảng tạo ra hạn chế:
  - Độ dài của danh sách.
  - Kiểm tra “tràn” và “rỗng” khi thực hiện chèn và xóa.
  - Khi thực hiện “chèn“ và “Xóa” đều phải thực hiện phép “dịch chuyển”.

## *3. Một số nhận xét*

- Trong các cấu trúc DS, để thực hiện các thao tác cơ bản cần các thao tác tối thiểu:
  - Định vị phần tử đầu tiên của DS
  - Cho trước vị trí của một phần tử bất kì trong DS, xác định được phần tử tiếp theo.

## *4. Kiểu dữ liệu con trỏ và việc cấp phát/thu hồi bộ nhớ động*

- Để khắc phục, cần
  - Một thủ tục tiền định để cấp phát bộ nhớ ( New (p) trong TP, trong C có các hàm Void * calloc, * void maloc ) và một thủ tục để giải phóng bộ nhớ ( Dispose(p) trong Tp và hàm Void Free  trong C)
  - Dùng biến con trỏ để truy cập đến vùng nhớ này.

# CHƯƠNG V CẤU TRÚC DỮ LiỆU BẢNG BĂM

1. Bảng băm mở
    - 1.1. Bảng băm
    - 1.2. Hàm băm
    - 1.3. Xung đột
    - 1.4. Một số hàm băm thông dụng
1. Bảng băm đóng
    - 2.1. Băm lại tuyến tính.
    - 2.2. Băm lại bình phương
    - 2.3. Băm lại bằng cách tạo vùng mới

# CHƯƠNG VI:
# CẤU TRÚC DỮ LiỆU BẢNG BĂM
## *1. Bảng băm mở*

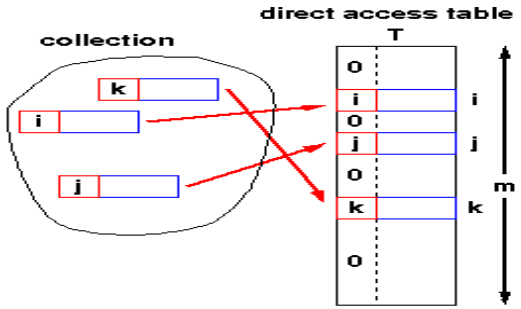
- **Bảng băm** (Hash Table):

- Mảng B gồm m phần tử

-Lưu trữ chỉ số định vị phần tử dữ liệu có khóa phân biệt thuộc tập số nguyên { 0,1,2…m-1}

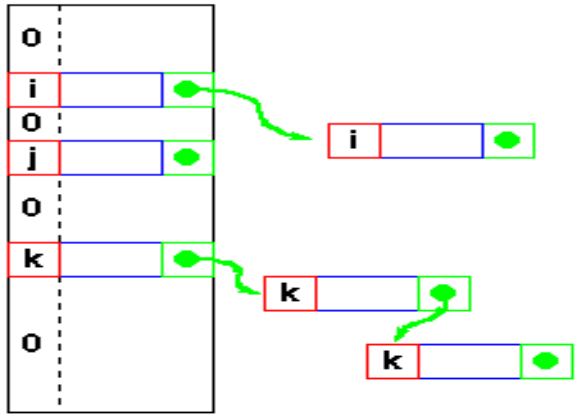
## *Hash Tables - Structure*

- **Simplest case:**
  - Assume items have integer keys in the range $1 .. m$
  - Use the value of the key itself to select a slot in a direct access table in which to store the item
  - To search for an item with key, $k$, just look in slot $k$
    - If there's an item there, you've found it
    - If the tag is 0, it's missing.
  - Constant time, $O(1)$



# *Hash Tables - Relaxing the constraints*

- **Keys are integers**
  - Need a hash function
    $h(key) \longrightarrow integer$
    *ie* one that maps a key to an integer
  - Applying this function to the key produces an address
  - If $h$ maps each key to a *unique integer* in the range $0 .. m-1$ then search is $O(1)$



## *1. Bảng băm mở*

- **Hàm băm**

  (Hash function):

H(x) cho giá trị là một chỉ số phần tử của B

## *1. Bảng băm mở*

- **Xung đột** (collision):
  - x1<>x2 nhưng H(x1)=H(x2)

# 1. Bảng băm mở

## Hash Tables - Collision handling

- **Collisions**
  - Occur when the hash function maps two different keys to the same address
  - The table must be able to recognise and resolve this
  - Recognise
    - Store the actual key with the item in the hash table
    - Compute the address
      - k = h( key )
    - Check for a hit
      - *if (* table[k].key == key *) then hit*
        *else try next entry*
  - Resolution
    - Variety of techniques

We'll look at various "*try next entry*" schemes

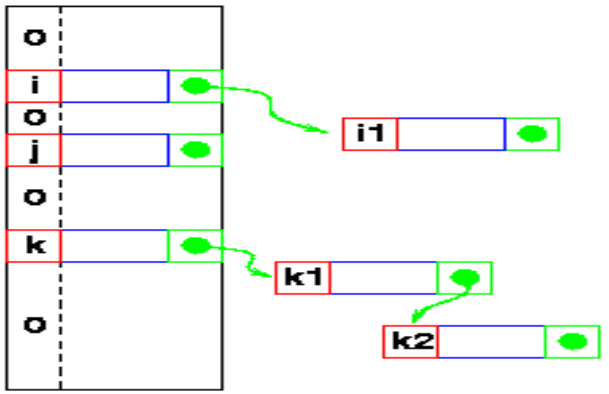
## *1. Bảng băm mở*

Xung đột:

- Giải quyết:
  - liên kết các danh sách có các khóa khác nhau nhưng có cùng giá trị hàm băm thành một danh sách
  - liên kết trong bẳng băm B sẽ trỏ tới danh sách đầu tiên.

# Hash Tables - Linked lists

- Collisions - Resolution
  1. Linked list attached to each primary table slot
     - h(i) == h(i1)
     - h(k) == h(k1) == h(k2)
  - Searching for i1
    - Calculate h(i1)
    - Item in table, i, doesn't match
    - Follow linked list to i1
  - If NULL found, key isn't in table



# CHƯƠNG VI:
# CẤU TRÚC DỮ LiỆU BẢNG BĂM
## *1. Bảng băm mở*

**Một số hàm băm thông dụng:**

- **Hàm cắt bỏ** bỏ bớt một phần nào đó của khóa.
  - Ví dụ: x=842615, bỏ bớt chẳng hạn các chữ số hàng lẻ (1,3,5…), số còn lại sẽ là 821. Vậy H(x) = H(842615) = 821.
  - ***Nhận xét***: khó có phân bố đều.

- **Hàm gấp** chia số nguyên đó thành một số đoạn tùy chọn, sau đó kết hợp
- Ví dụ: Số các hàng lẻ: 465 và số các hàng chẵn: 821, vậy H(x)=465+821=1286.
  - ***Nhận xét***: Tính chất thứ hai có thể thỏa mãn tốt hơn

## *1. Bảng băm mở*

- **Hàm phần dư** của phép chia x/m
- Nên chọn m là số nguyên tố.

  - ***Nhận xét***: Các cách lấy phần dư cho khả năng tránh hiện tượng xung đột

## *Hash Tables - Reducing the range to* [ 0, $m$ )

① Division

- Use a mod function

$$h(k) = k \bmod m$$

- Choice of $m$?
  - Powers of 2 are generally not good!

    $h(k) = k \bmod 2^n$

    selects last $n$ bits of $k$

    $k \bmod 2^8$ selects these bits

    01100101111000011010

  - All combinations are not generally equally likely
- Prime numbers close to $2^n$ seem to be good choices

  *eg* want ~4000 entry table, choose $m$ = 4093

# CHƯƠNG VI:
# CẤU TRÚC DỮ LiỆU BẢNG BĂM
## *2. Bảng băm đóng*

- Bảng băm mở: chỉ dùng để lưu trữ các liên kết trỏ đến các thành phần dữ liệu có khóa tương ứng.
- Bảng băm đóng: bảng băm mà mỗi thành phần của nó lưu trữ chính các thành phần dữ liệu.

# CHƯƠNG VI:
# CẤU TRÚC DỮ LiỆU BẢNG BĂM
## *2. Bảng băm đóng*

**Các phương pháp xử lý:**

***a) Băm lại tuyến tính***

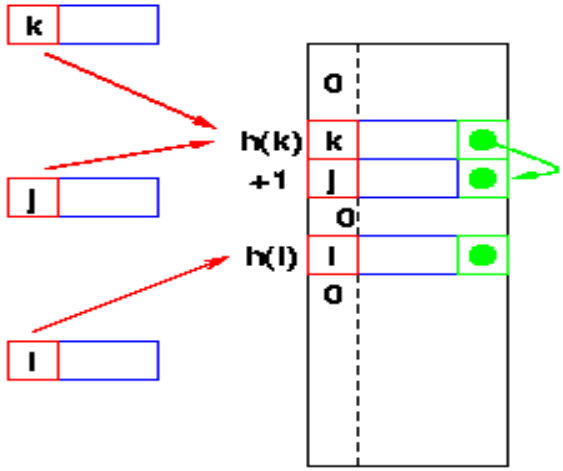
Hi (x) = (H(x)+i) mod m

- **Nhận xét C**ác giá trị hàm băm xếp thành từng đoạn con, nên việc tìm kiếm ví trí  rỗng sẽ rất chậm.

# CHƯƠNG VI:
# CẤU TRÚC DỮ LiỆU BẢNG BĂM
## *2. Bảng băm đóng*

### *Hash Tables - Re-hash functions*

- ❸ The re-hash function
  - Many variations
- Linear probing
  - h'(x) is +1
  - Go to the next slot until you find one empty




- Can lead to bad clustering
- Re-hash keys fill in gaps between other keys and exacerbate the collision problem

## 2. Bảng băm đóng

### b) Băm lại bình phương

$$Hi(x) = ( H(x)+i^2) \bmod m$$

**Hash Tables - Re-hash functions**

- The re-hash function
  - Many variations
- Quadratic probing
  - $h'(x)$ is $c\ i^2$ on the $i^{th}$ probe
  - Avoids primary clustering
  - Secondary clustering occurs
    - All keys which collide on $h(x)$ follow the same sequence
    - First
      - $a = h(j) = h(k)$
    - Then $a + c,\ a + 4c,\ a + 9c, \ldots.$
    - Secondary clustering generally less of a problem

## 2. Bảng băm đóng

# c) Băm lại bằng cách tạo vùng mới

- Ngoài bảng B cần tạo ra một vùng không gian mới

### Hash Tables - Overflow area

- Overflow area
  - Linked list constructed in special area of table called overflow area
- h(k) == h(j)
- k stored first
- Adding j
  - Calculate h(j)
  - Find k
  - Get first slot in overflow area
  - Put j in it
  - k's pointer points to this slot
- Searching - same as linked list