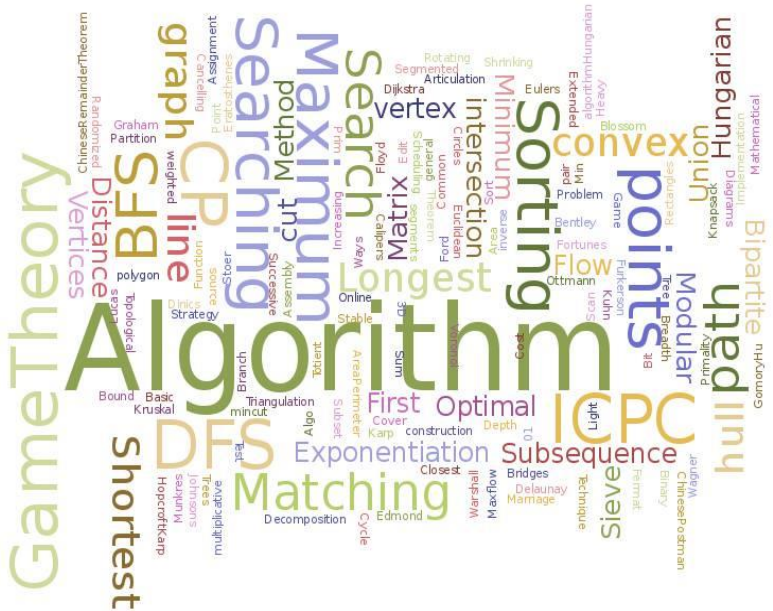
Cấu trúc dữ liệu
Và
Thuật toán

Mục tiêu bài học

- Cấu trúc dữ liệu
- Bộ sưu tập
- Phân tích một thuật toán
- Cấu trúc dữ liệu không nguyên thủy
  - ✓ Mảng
  - ✓ Danh sách liên kết
  - ✓ Cây nhị phân
  - ✓ Cây tổng hợp
  - ✓ Đống
  - ✓ Hàng đợi
  - ✓ ngăn xếp
- Thuật toán sắp xếp
- Thuật toán tìm kiếm



Phần 1

CẤU TRÚC DỮ LIỆU

Cấu trúc dữ liệu

- "Một khi bạn viết thành công các chương trình cho các thuật toán phức tạp, chúng thường chạy cực kỳ nhanh. Máy tính không cần hiểu thuật toán, nhiệm vụ của nó chỉ là chạy chương trình."
- Có một số khía cạnh của một chương trình tốt, chúng phải
  - ✓chạy chính xác
  - ✓chạy hiệu quả
  - ✓dễ đọc và dễ hiểu
  - ✓dễ dàng gỡ lỗi*Và*
  - ✓dễ dàng sửa đổi**.**

Cấu trúc dữ liệu (tiếp)

Cấu trúc dữ liệu là gì?

- Sơ đồ tổ chức các thông tin liên quan
- Cách thức tổ chức các bộ dữ liệu trong một hệ thống cụ thể
- Một tập hợp có tổ chức của các mục dữ liệu
- Một định dạng có thể hiểu được trên máy tính được sử dụng để lưu trữ, truy cập, truyền và lưu trữ dữ liệu
- Cách tổ chức dữ liệu để đảm bảo xử lý hiệu quả: điều này có thể

trong danh sách, mảng, ngăn xếp, hàng đợi hoặc cây

Cấu trúc dữ liệu là một định dạng chuyên biệt để tổ chức và lưu trữ dữ liệu để có thể truy cập và làm việc với nó theo những cách thích hợp nhằm tạo ra một chương trình hiệu quả

Cấu trúc dữ liệu (tiếp)

- Cấu trúc dữ liệu có thể được phân loại thành
  - ✓ Cấu trúc dữ liệu nguyên thủy
  - ✓ Cấu trúc dữ liệu không nguyên thủy.
- *Cấu trúc dữ liệu nguyên thủy:*
  - ✓ Đây là những cấu trúc dữ liệu có thể được thao tác trực tiếp bằng lệnh máy.
  - ✓ Trong ngôn ngữ C, các cấu trúc dữ liệu nguyên thủy khác nhau là int, float, char, double.
- *Cấu trúc dữ liệu không nguyên thủy:*
  - ✓ Đây là những cấu trúc dữ liệu không thể thao tác trực tiếp bằng lệnh máy. Mảng, danh sách liên kết, tệp, v.v., là một số cấu trúc dữ liệu không nguyên thủy và được phân loại thành *cấu trúc dữ liệu tuyến tính* Và *cấu trúc dữ liệu phi tuyến tính*.

Cấu trúc dữ liệu (tiếp)

Cấu trúc dữ liệu = Dữ liệu có tổ chức + Hoạt động được phép

- **Có hai khía cạnh thiết kế cho mọi cấu trúc dữ liệu:**

phần giao diện

Các chức năng có thể truy cập công khai của loại. Các chức năng như tạo và hủy đối tượng, chèn và xóa các phần tử (nếu là vùng chứa), gán giá trị, v.v.

- **Phần thực hiện**:

Việc triển khai nội bộ phải độc lập với giao diện. Do đó, các chi tiết về khía cạnh triển khai sẽ được ẩn khỏi người dùng.

Bộ sưu tập

- Các chương trình thường xử lý các bộ sưu tập đồ vật.
- Các bộ sưu tập này có thể được tổ chức theo nhiều cách và sử dụng nhiều cấu trúc chương trình khác nhau để thể hiện chúng, tuy nhiên, từ quan điểm trừu tượng, sẽ có một vài thao tác chung trên bất kỳ bộ sưu tập nào.

tạo nên	Tạo bộ sưu tập mới
thêm vào	Thêm một mục vào bộ sưu tập
xóa bỏ	Xóa một mục khỏi bộ sưu tập
tìm thấy	Tìm một mục phù hợp với một số tiêu chí trong bộ sưu tập
hủy hoại	Phá hủy bộ sưu tập

Phần 2

PHÂN TÍCH MỘT Thuật Toán

Phân tích một thuật toán

- **Trình tự câu lệnh đơn giản** S
  1; S2; .... ; Sk
  ✓ *Độ phức tạp là O(*1*)*miễn là*k*là hằng số
- **Vòng lặp đơn giản**
  for(i=0;i<n;i++) { s; } nó ở đâu*O(*1*)*
  ✓ Độ phức tạp là*KHÔNG(*1*)* hoặc*TRÊN)*
- **Chỉ số vòng lặp không thay đổi tuyến tính**
  h = 1;
  trong khi (h <= n) { s; h = 2 * h;}
  ✓ Độ phức tạp*O(*nhật ký*N)*
- **Vòng lặp lồng nhau (chỉ số vòng lặp phụ thuộc vào chỉ số vòng lặp bên ngoài)**
  cho(i=0;i<n;i++) f
  cho(j=0;j<n;j++) { s; }
  ✓ Độ phức tạp là *n O(n)*hoặc*TRÊN2)*

Phần 3

CẤU TRÚC DỮ LIỆU KHÔNG CHÍNH XÁC

Mảng

Mảng là hình thức đơn giản nhất để triển khai một bộ sưu tập

- Mỗi đối tượng trong một mảng được gọi là một*phần tử mảng*
- Mỗi phần tử có cùng kiểu dữ liệu (mặc dù chúng có thể có các giá trị khác nhau)
- Các phần tử riêng lẻ được truy cập theo chỉ mục bằng cách sử dụng một dãy số nguyên liên tiếp **Mảng một chiều hoặc vectơ**

```
int A[10];
vì ( i = 0; i < 10; i++)
A[i] = i +1;
```

A[0]	Một [1]	Một [2]	Một [3]			A[n-2]	A[n-1]
1	2	3	4			N-1	N

Mảng (Tiếp theo)

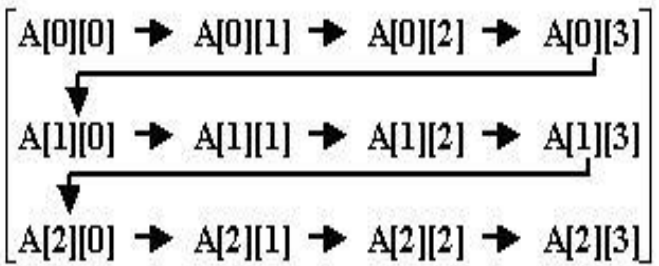
- Mảng đa chiều
- MỘT *mảng đa chiều* kích thước *N* (tức là một *N* mảng -chiều hoặc đơn giản *N*-D mảng) là tập hợp các mục được truy cập thông qua *N* biểu thức chỉ số dưới. Ví dụ, trong một ngôn ngữ hỗ trợ nó, phần tử thứ (i,j) của mảng hai chiều x được truy cập bằng cách viết x[i,j].

		Cột														
		0	1	2	3	4	5	6	7	số 8	9	10		j		N
R ồ W	0															
	1															
	2															
	:	:	:	:	:	:	:	:	:	:	:	:		:	:	:
	Tôi													x		
	tôi															

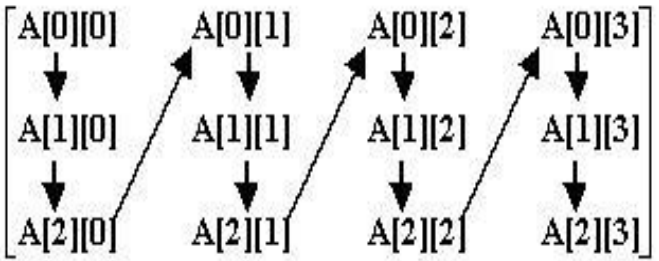
Mảng (Tiếp theo)

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]

Matrix A (3 rows x 4 columns)



Matrix A (Row Major Order)



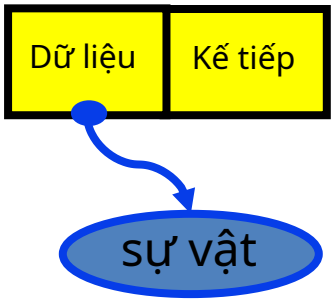
Matrix A (Column Major Order)

Mảng: Hạn chế

- Đơn giản và nhanh chóng nhưng phải xác định kích thước trong quá trình thi công
- Nếu bạn muốn chèn/xóa một phần tử vào/khỏi một vị trí cố định trong danh sách thì bạn phải di chuyển các phần tử đã có trong danh sách để nhường chỗ cho các phần tử tiếp theo trong danh sách.
- Vì vậy, trung bình, bạn có thể sao chép một nửa số phần tử.
- Trong trường hợp xấu nhất, việc chèn vào vị trí 1 yêu cầu phải di chuyển tất cả các phần tử.
- Việc sao chép các phần tử có thể khiến chương trình chạy lâu hơn nếu chèn/xóa các hoạt động diễn ra thường xuyên, đặc biệt khi bạn cho rằng chi phí sao chép là rất lớn (như khi chúng tôi sao chép chuỗi)
- Một mảng không thể được mở rộng một cách linh hoạt, người ta phải cấp phát một mảng mới của kích thước phù hợp và sao chép mảng cũ sang mảng mới

Danh sách liên kết

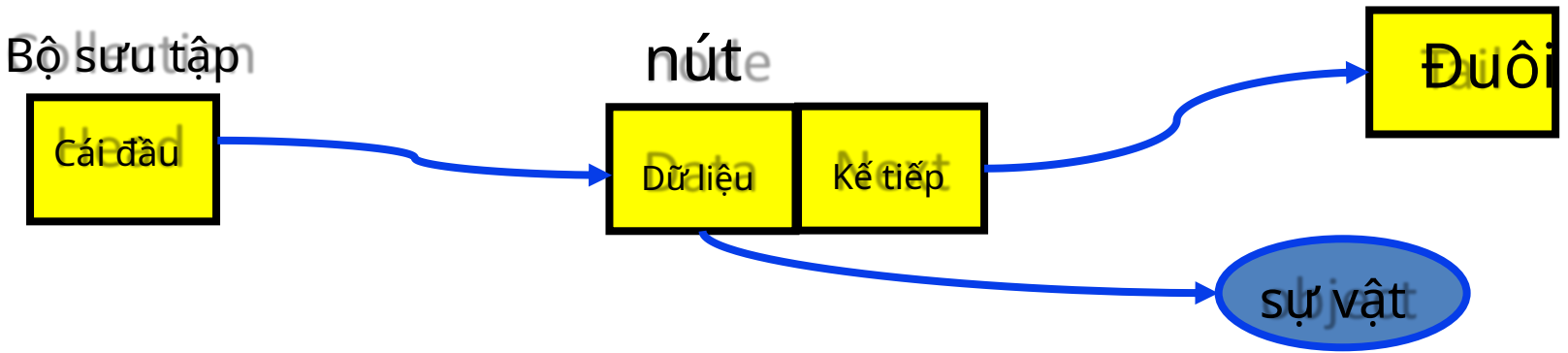
- Danh sách liên kết là một cấu trúc dữ liệu động rất linh hoạt: các mục có thể được thêm vào hoặc xóa khỏi danh sách theo ý muốn
  - ✓ Tự động phân bổ không gian cho từng phần tử khi cần thiết
  - ✓ Bao gồm một con trỏ tới mục tiếp theo
  - ✓ số lượng mục có thể được thêm vào danh sách chỉ bị giới hạn bởi dung lượng bộ nhớ khả dụng
- Danh sách liên kết có thể được coi là đã kết nối (linked)điểm giao
- Mỗinútcủa danh sách chứa
  - ✓ mục dữ liệu
  - ✓ một con trỏ tới nút tiếp theo
  - ✓ Nút cuối cùng trong danh sách chứa con trỏ NULL tới
  - ✓ chỉ ra rằng đó là sự kết thúc hoặc*đuôi*của danh sách.



Bộ sưu tập

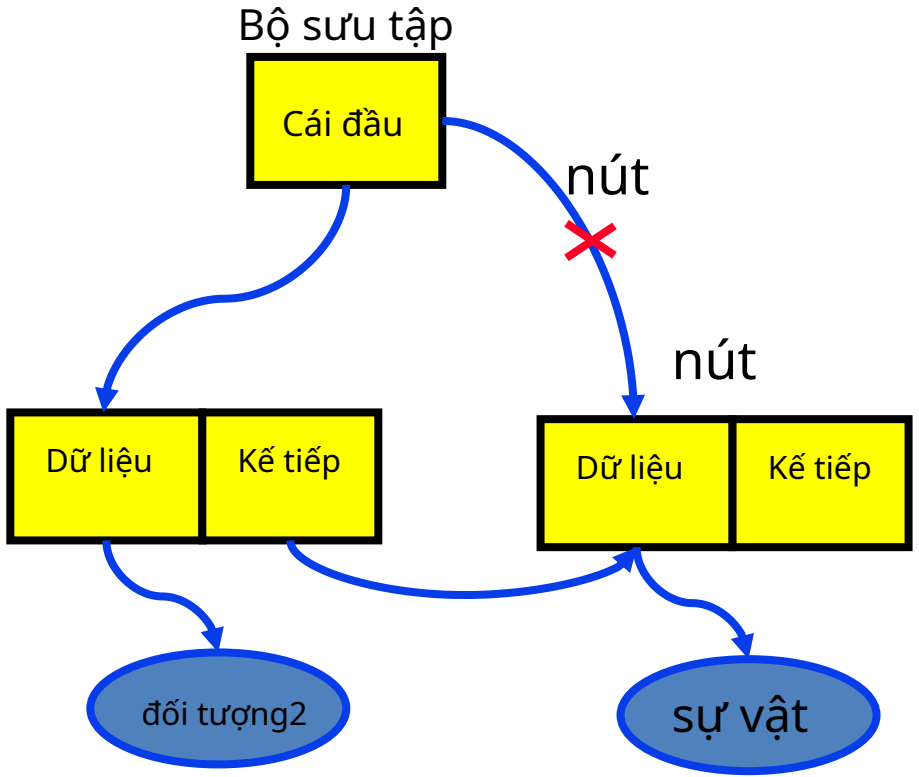
- Cấu trúc bộ sưu tập có một con trỏ tới danh sáchcái đầu
  - ✓ Ban đầu là KHÔNG
- Thêm mục đầu tiên
  - ✓ Phân bổ không gian cho nút
  - ✓ Đặt con trỏ dữ liệu của nó thành đối tượng
  - ✓ Đặt bên cạnh NULL
  - ✓ Đặt Head để trỏ tới nút mới

Biến (hoặc thẻ điều khiển) đại diện cho danh sách chỉ đơn giản là một con trỏ tới nút *ở cái đầu* của danh sách.



Danh sách liên kết (tiếp)

- **Thêm một nút**
  - ✓ Phân bổ không gian cho nút
  - ✓ Đặt con trỏ dữ liệu của nó thành đối tượng
  - ✓ Đặt Bên cạnh Đầu hiện tại
  - ✓ Đặt Head để trỏ tới nút mới



Danh sách liên kết - *Thêm vào* thực hiện

- **Thực hiện**

```
cấu trúc t_node {
        làm mất hiệu lực * mục;
        struct t_node *next;
        } nút;
typedef struct t_node *Node; bộ sưu
tập cấu trúc {
        Đầu nút;
        ……
        };
int AddToCollection( Bộ sưu tập c, void *item ) {
        Nút mới = malloc( sizeof( struct t_node ) );
        mới-> mục = mục;
        mới->tiếp theo = c->đầu;
        c->đầu = mới;
        trả về ĐÚNG;
        }
```

Định nghĩa kiểu đệ quy - C cho phép điều đó!

Kiểm tra lỗi, xác nhận được bỏ qua cho rõ ràng!

Danh sách liên kết - *Tìm thấy* thực hiện

- **Thực hiện**

✓ *Một đệ quy thực hiện là cũng có thể!*

```
void *FidinCollection( Bộ sưu tập c, void
* chìa khóa ) {
        Nút n = c->đầu; trong khi ( n !
        = NULL ) {
        if ( KeyCmp( ItemKey( n->item ), key ) == 0
) {
            trở lại      n->mục;
                  n =    n->tiếp theo;
        }
            trở lại      VÔ GIÁ TRỊ;
        }
```

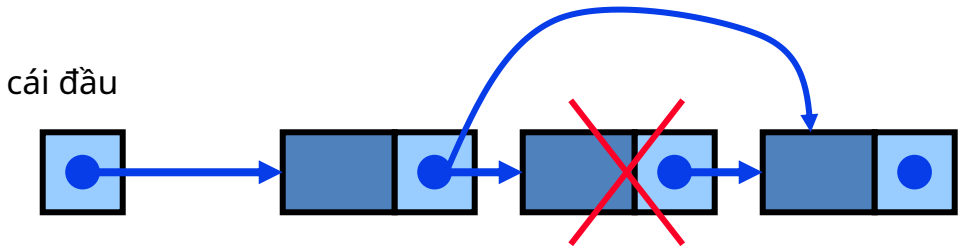
Thêm thời gian Hằng số - độc lập với n

thời gian tìm kiếmTrường hợp xấu nhất - n

Danh sách liên kết - *Xóa bỏ* thực hiện

- **Thực hiện**

```
void *DeleteFromCollection( Bộ sưu tập c, void *key ) {
        Nút n, trước;
        n = prev = c->head; trong khi
        ( n != NULL ) {
        if ( KeyCmp( ItemKey( n->item ), key ) == 0 ) {
                trước->tiếp theo = n->tiếp theo;
                trả lại n;
                        }
            trước   = n;
            N  =  n->tiếp theo;
            }
        trở lại      VÔ GIÁ TRỊ;
        }
```



Danh sách liên kết - Biến thể

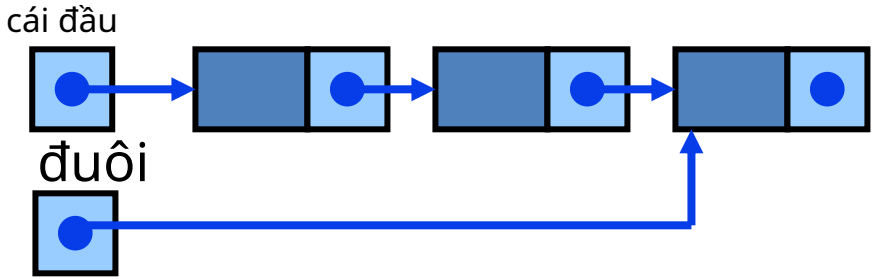
- Thực hiện đơn giản nhất
  - ✓ Thêm vào đầu
  - ✓ Ngữ nghĩa vào sau ra trước (LIFO)
- Sửa đổi
  - ✓ Nhập trước xuất trước (FIFO)
  - ✓ Giữ một con trỏ đuôi

```
cấu trúct_node {
        làm mất hiệu lực * mục;
        cấu trúc t_node      * Kế tiếp;
} nút;
typedef struct t_node *Node; cấu trúc
bộ sưu tập {
        Nút đầu, đuôi;
};
```

Bằng cách đảm bảo rằng phần đuôi của danh sách luôn trỏ đến phần đầu, chúng ta có thể xây dựng một danh sách liên kết vòng

đầu là đuôi-> tiếp theo

LIFO hoặc FIFO sử dụng MỘT con trỏ



Danh sách liên kết - Liên kết đôi

- Liên kết đôidanh sách
  - ✓ Có thể quét vàocả hai hướng

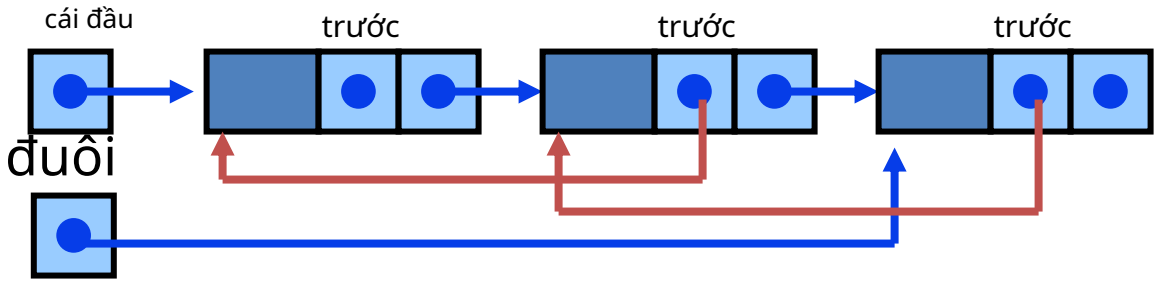
```
cấu trúc t_node {
        làm mất hiệu lực * mục;
        cấu trúc    nút t_node    * trước đó,
                                  * Kế tiếp;
        } nút;
typedef     cấu trúc t_node
* Nút;
cấu trúc    bộ sưu tập {
        Nút     đầu, đuôi;
        };
```

Ứng dụng yêu cầu

tìm kiếm cả hai chiều

Ví dụ. Tìm kiếm tên

trong danh bạ điện thoại



Cây nhị phân

- Dạng đơn giản nhất của Cây là Cây nhị phân

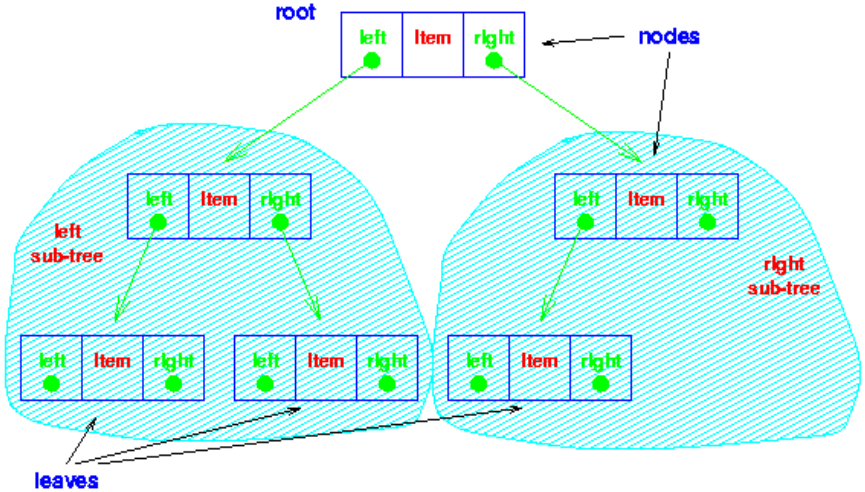
  ✓ Cây nhị phân bao gồm

  - Nút (được gọi là nút ROOT)
  - Cây con trái và phải
  - Cả hai cây con đều là cây nhị phân
  - Các nút ở mức thấp nhất của cây (các nút không có cây con) được gọi là lá

Lưu ý

đệ quy

sự định nghĩa!



Mỗi cây con

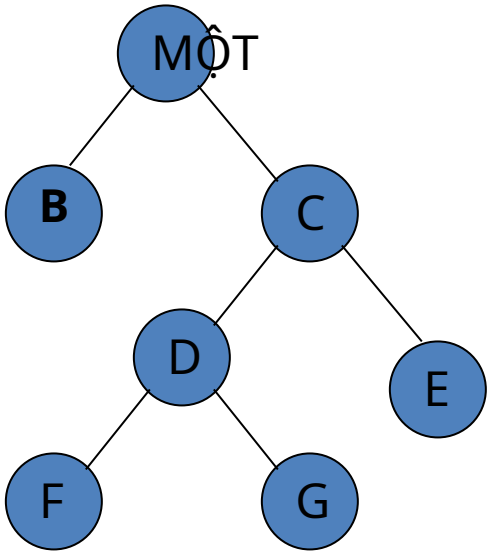
là chính nó

cây nhị phân

- trong một *cây nhị phân có thứ tự* khóa của tất cả các nút trong
  - ✓ cây con bên trái nhỏ hơn cây con gốc
  - ✓ khóa của tất cả các nút trong cây con bên phải đều lớn hơn khóa của nút gốc,
  - ✓ cây con bên trái và bên phải được sắp xếp theo thứ tự cây nhị phân.

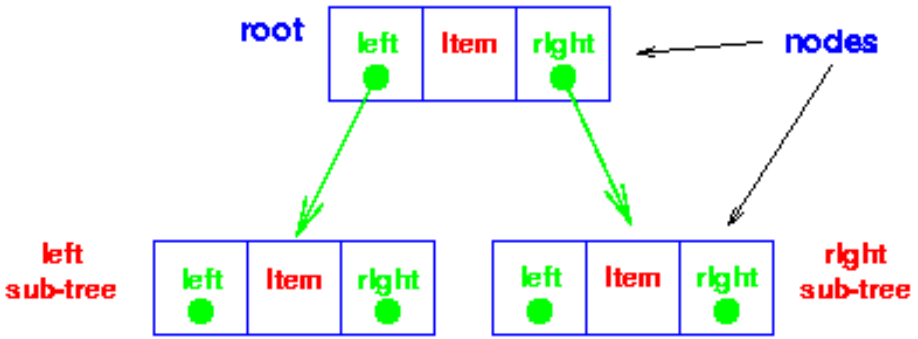
Bộ sưu tập

- Nếu A là gốc của cây nhị phân và B là gốc của cây con trái/phải của nó thì
  - ✓ A là *bố* của B
  - ✓ B là *con trai trái/phải* của A
- Hai nút là *anh em* nếu họ là con trai trái và phải của cùng một người cha
- Nút n1 là một *tổ tiên* của n2 (và n2 là *hậu duệ* của n1) nếu n1 là bố của n2 hoặc bố của tổ tiên nào đó của n2
- *Cây nhị phân nghiêm ngặt*: Nếu mọi nút không có lá trong cây nhị phân đều không có nút nào trống và cây con bên phải
- *Mức độ* của một nút: Root có cấp 0. Cấp của bất kỳ nút nào cao hơn cấp một của cha nó
- *Chiều sâu*: Mức tối đa của bất kỳ lá nào trên cây
  - ✓ Một cây nhị phân có thể chứa tối đa $2^{tôi}$ các nút ở cấp độ l
  - ✓ Tổng số nút của cây nhị phân có độ sâu d = $2^{d+1} - 1$



Cây nhị phân - Thực hiện

```
cấu trúct_node {
    làm mất hiệu lực * mục;
    struct t_node *left;
    struct t_node *đúng;
};

cấu trúc typedeft_node *Nút;

cấu trúct_bộ sưu tập {
    Nút gốc;
    ......
};
```



Cây nhị phân - Thực hiện

- **Tìm thấy**

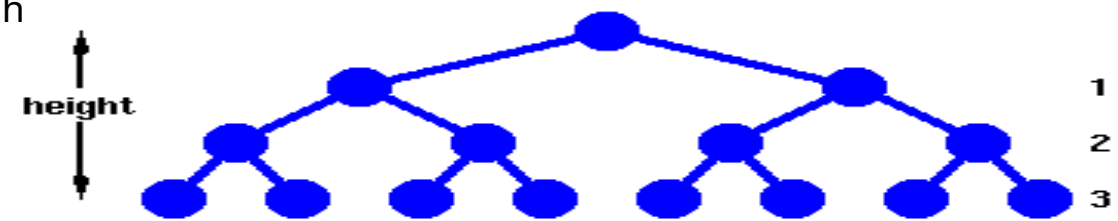
```
bên ngoàiint KeyCmp(void *a, void *b );
/* Trả về -1, 0, 1 cho a < b, a == b, a > b */

trống rỗng*FindInTree( Nút t, void *key ) {
    if ( t == (Node)0 ) trả về NULL; switch( KeyCmp( key,
    ItemKey(t->item) ) ) {
        case -1 : return FindInTree( case 0:    t->trái   , chìa khóa );
        return t->item; case +1 : return
        FindInTree( }                           t->phải   , chìa khóa );

    }

trống rỗng*FindInCollection(bộ sưu tập c, void *key ) {
    return FindInTree( c->root, key );
}
```

Ít hơn, tìm kiếm bên trái

Lớn hơn, tìm kiếm bên phải

Cây nhị phân - Hiệu suất

- **Tìm thấy**
  - ✓ Cây hoàn chỉnh



  - ✓ Chiều cao, h
    - Các nút đi qua đường đi từ gốc đến lá
  - ✓ Số lượng nút, h
    - $N = 1 + 2_1 + 2_2 + \ldots + 2_h = 2_{h+1} - 1$
    - h = tầng ($\log_2 N$)
  - ✓ Vì chúng ta cần nhiều nhất $h+1$ so sánh, tìm trong $O(h+1)$ hoặc $O(\log N)$

Cây nhị phân - Đi ngang

Traverse: Đi qua cây, liệt kê từng nút một lần

- **Đặt hàng trước (còn gọi là đặt hàng theo chiều sâu)**
  1. Truy cập gốc
  2. Duyệt cây con trái theo thứ tự trước
  3. Duyệt cây con bên phải theo thứ tự trước
- **InOrder (còn gọi là thứ tự đối xứng)**
  1. Duyệt cây con trái theo thứ tự
  2. Truy cập gốc
  3. Duyệt cây con bên phải theo thứ tự
- **PostOrder (còn gọi là thứ tự đối xứng)**
  1. Duyệt cây con trái theo thứ tự sau
  2. Duyệt cây con bên phải theo thứ tự sau
  3. Truy cập gốc

Cây nhị phân - Ứng dụng

- Cây nhị phân là một cấu trúc dữ liệu hữu ích khi phải đưa ra các quyết định hai chiều.

  được thực hiện tại mỗi điểm trong một quá trình

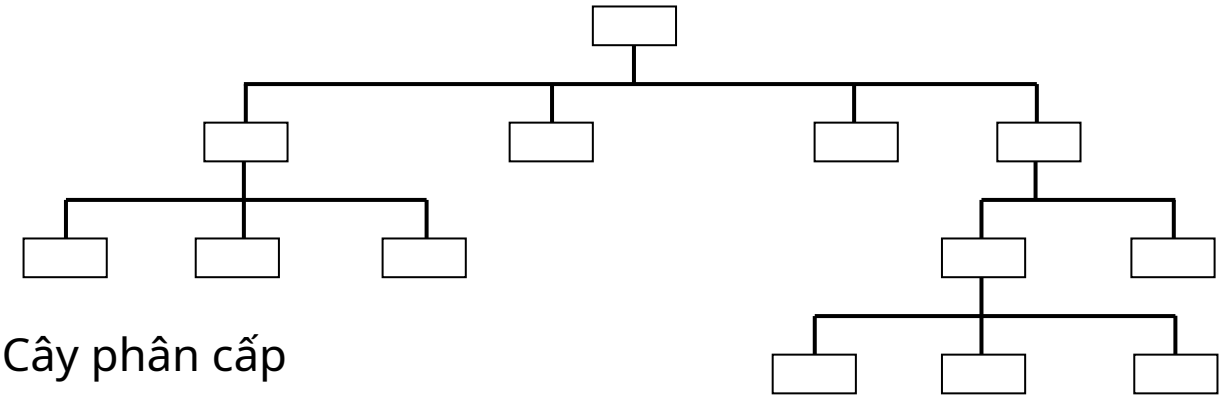
  ✓ Ví dụ: Tìm số trùng lặp trong danh sách số

- Cây nhị phân có thể được sử dụng để biểu diễn một biểu thức chứa toán hạng (lá) và toán tử (nút không có lá).
- Việc duyệt cây sẽ dẫn đến các dạng biểu thức trung tố, tiền tố hoặc hậu tố
- Hai cây nhị phân là MIRROR SIMILAR nếu cả hai đều trống hoặc nếu chúng

  không trống, cây con bên trái của mỗi cây là bản sao tương tự như cây con bên phải

Cây tổng hợp

- Cây là một tập hợp hữu hạn khác rỗng các phần tử trong đó có một phần tử được gọi là ROOT và phần tử còn lại được phân chia thành m >=0 các tập con rời rạc, mỗi cái đều là một cái cây
- Các loại cây khác nhau – cây nhị phân, cây n-ary, cây đỏ đen, cây AVL



Cây phân cấp

Đống

Đống được dựa trên khái niệm về một**cây hoàn chỉnh**

Cây nhị phân là**Hoàn toàn đầy đủ**nếu nó có chiều cao,*h*, và có 2*h*nút +1-1.

- Cây nhị phân có chiều cao*h*, là**hoàn thành***nếu*
  - ✓ nó trống rỗng*hoặc*
  - ✓ cây con bên trái của nó có chiều cao đầy đủ*h*-1 và cây con bên phải của nó có chiều cao hoàn toàn*h*-2*hoặc*
  - ✓ cây con bên trái của nó hoàn toàn có chiều cao*h*-1 và cây con bên phải của nó có chiều cao đầy đủ*h*-1.
- Một cây hoàn chỉnh được điền từ bên trái:
  - ✓ tất cả những chiếc lá đều bật
  - ✓ cùng cấp độ*hoặc*hai cái liền kề*Và*
  - ✓ tất cả các nút ở mức thấp nhất càng xa bên trái càng tốt.
- Cây nhị phân có**thuộc tính đống***nếu*
  - ✓ nó trống rỗng*hoặc*
  - ✓ khóa ở gốc lớn hơn khóa ở cả hai cây con và cả hai cây con đều có thuộc tính heap.

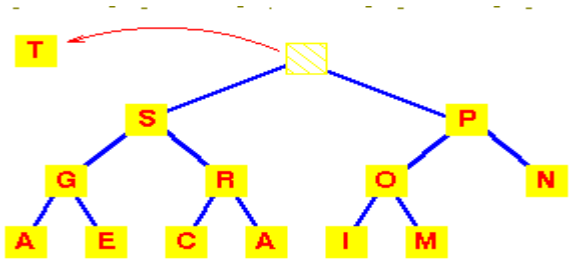
Đống (Tiếp theo)

- Một đống có thể được sử dụng làm hàng đợi ưu tiên:
- mục có mức độ ưu tiên cao nhất nằm ở thư mục gốc và được trích xuất một cách tầm thường. Nhưng nếu gốc là bị xóa, chúng ta còn lại hai cây con và chúng ta phải *hiệu quả* tạo lại một cây duy nhất với thuộc tính heap.
- Giá trị của cấu trúc heap là chúng ta có thể trích xuất mục có mức độ ưu tiên cao nhất và chèn một cái mới vào **O(log *N*)** thời gian.

Ví dụ:

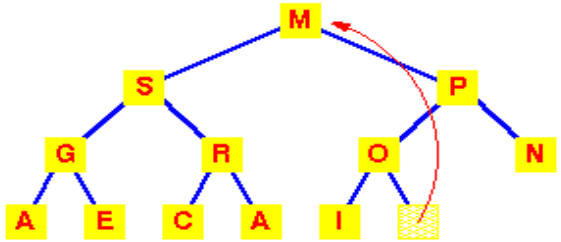
Việc xóa sẽ loại bỏ

T ở gốc

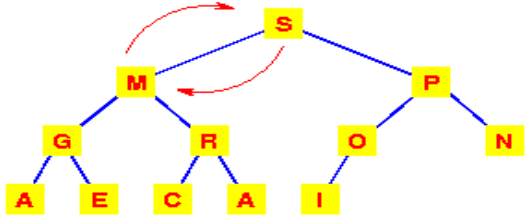


Đống (Tiếp theo)

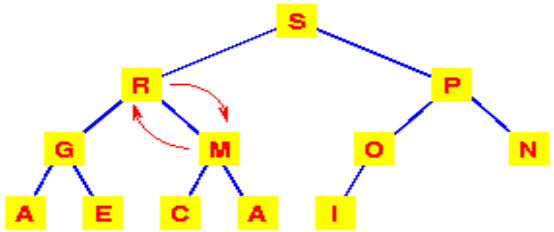
- Để tìm ra cách duy trì thuộc tính heap, hãy sử dụng thực tế là một cây hoàn chỉnh được điền từ bên trái. Vì vậy, vị trí phải trống là vị trí do M chiếm giữ. Đặt nó vào vị trí gốc còn trống.



- Điều này đã vi phạm điều kiện gốc phải lớn hơn mỗi con của nó. Vì vậy hãy hoán đổi M với số lớn hơn trong số con của nó.



- Cây con bên trái hiện đã mất thuộc tính heap. Vì vậy, một lần nữa hoán đổi M với số lớn hơn trong số con của nó.
- Chúng ta cần phải thực hiện nhiều nhất *h* trao đổi gốc của cây con với một trong các cây con của nó để khôi phục hoàn toàn thuộc tính heap.



- **O(h) hoặc O(log n)**

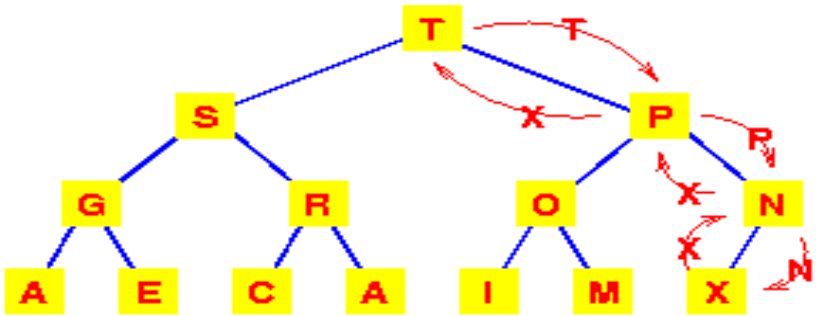
Đống (Tiếp theo)

- **Bổ sung vào Heap**

✓ Để thêm một mục vào heap, chúng ta làm theo quy trình ngược lại.

✓ Đặt nó ở vị trí lá tiếp theo và di chuyển nó lên.

✓ Một lần nữa, chúng tôi yêu cầuO(*h*) hoặc O(log*N*) trao đổi.



So sánh cấu trúc dữ liệu

	Mảng	Danh sách liên kết	Cây
	Đơn giản, nhanh chóng	Đơn giản	Vẫn đơn giản
	Không linh hoạt	Linh hoạt	Linh hoạt
Thêm vào	O(1) TRÊN)*sắp xếp inc*	O(1) *sắp xếp -> không có tiến bộ*	O(logn)
Xóa bỏ	TRÊN)	O(1) -*bất kì* TRÊN) -*cụ thể*	O(logn)
Tìm thấy	TRÊN) O(đăng nhập) *Tìm kiếm nhị phân*	TRÊN) *(không tìm kiếm thùng rác)*	O(logn)

Hàng đợi

Hàng đợi là các bộ sưu tập động có một số khái niệm về thứ tự

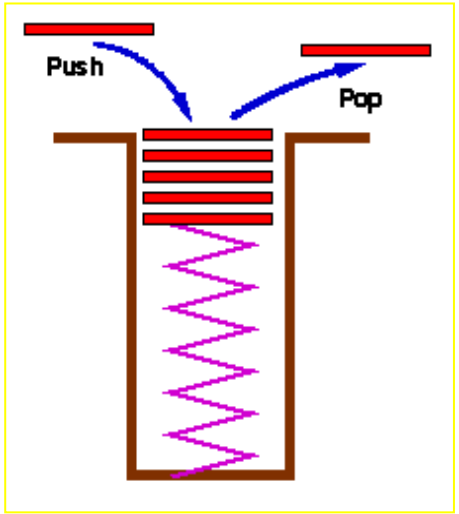
- **hàng đợi FIFO**
  - ✓ Hàng đợi trong đó mục được thêm vào đầu tiên luôn là mục được đưa ra đầu tiên.
- **hàng đợi LIFO**
  - ✓ Hàng đợi trong đó mục được thêm gần đây nhất luôn là mục được đưa ra đầu tiên.
- **Hàng đợi ưu tiên**
  - ✓ Một hàng đợi trong đó các mục được sắp xếp sao cho mục có mức độ ưu tiên cao nhất luôn là mục tiếp theo được trích xuất.

Hàng đợi có thể được triển khai bằng Danh sách liên kết

ngăn xếp

- **Ngăn xếp là một dạng thu thập đặc biệt với ngữ nghĩa LIFO**
- Hai phương pháp
  - ✓ int push(Stack s, void *item); **thêm mục vào đầu ngăn xếp**
  - ✓ void *pop(Stack s); **xóa mục được đẩy gần đây nhất khỏi đầu ngăn xếp**
- Giống như một cái máy xếp đĩa
- Các phương pháp khác
  - ✓ int IsEmpty(Ngăn xếp s); **Xác định xem ngăn xếp có gì trong đó không**
  - ✓ void *Top(Ngăn xếp s); **Trả lại mục ở trên cùng mà không xóa nó**

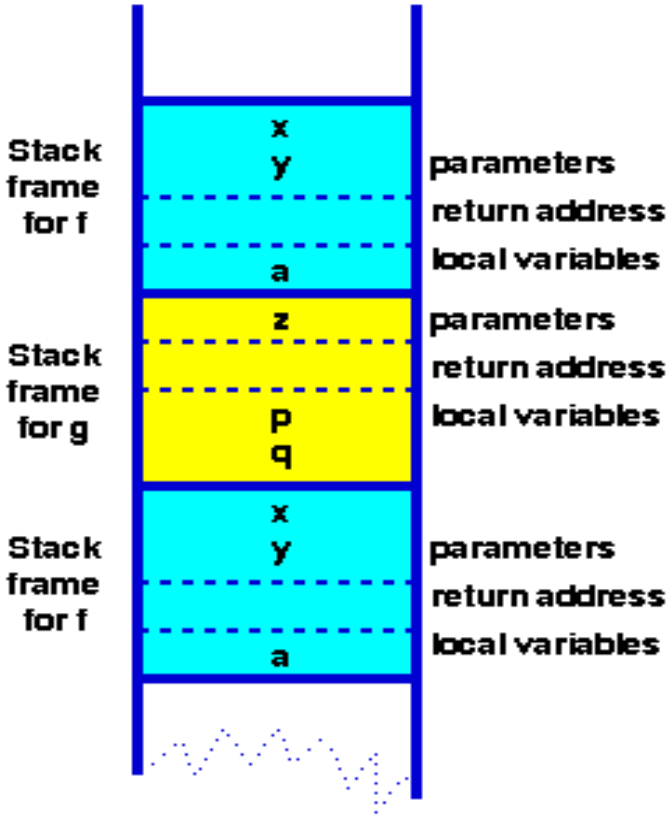
*** Ngăn xếp được thực hiện bằng Mảng hoặc Danh sách liên kết**



Ngăn xếp (Tiếp theo)

- Stack rất hữu ích cho đệ quy
- Phím gọi/trả về trong hàm & thủ tục

```
chức năngf( int x, int y) {
        int một;
        if ( term_cond ) trả về ...; a = ....;

        trả về g( a );
}
chức năngg( int z ) {
        int p, q;
        p = .... ; q = .... ; trở lại
                        f(p,q);
}
```



phần 4

Thuật toán sắp xếp

Thuật toán sắp xếp

- Một tập tin được cho là SẮP XẾP trên khóa nếu i < j ngụ ý rằng k[i] đứng trước k[j] theo thứ tự nào đó của các khóa
- **Các kiểu sắp xếp khác nhau**

✓ **Sắp xếp trao đổi**

- Sắp xếp bong bóng
- Sắp xếp nhanh chóng

✓ **Sắp xếp chèn**

✓ **Sắp xếp lựa chọn**

- Sắp xếp đống
- Sắp xếp cây nhị phân

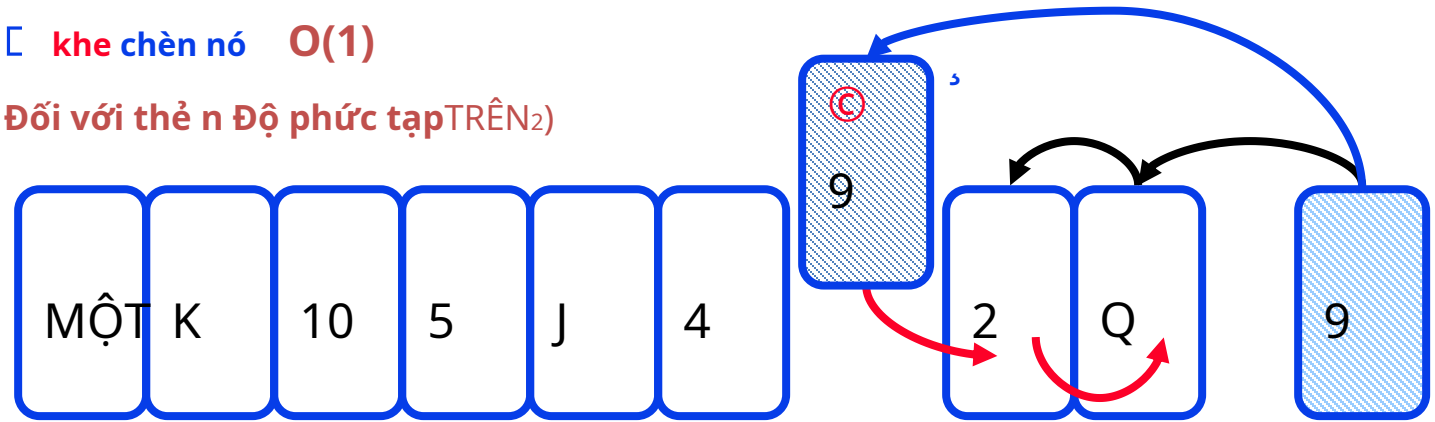
✓ **Hợp nhất và sắp xếp cơ số**

Sắp xếp chèn

- Thẻ đầu tiên đã được sắp xếp
- Với tất cả những gì còn lại,
  - **Quét lại từ đầu cho đến khi tìm thấy thẻ đầu tiên lớn hơn thẻ mới**

    TRÊN)
  - **Di chuyển tất cả những cái thấp hơn lên một** **TRÊN)**
  - **khe chèn nó** **O(1)**

Đối với thẻ n Độ phức tạpTRÊN2)



Sắp xếp bong bóng

- Từ phần tử đầu tiên
  - ✓ Trao đổi các cặp nếu chúng không còn phù hợp
  - ✓ Lặp lại từ đầu tiên đến n-1
  - ✓ Dừng lại khi bạn chỉ có một yếu tố để kiểm tra

Vòng ngoài/Mần lặp lại

Vòng trong
N-1, *N*-2, *N*-3, ..., 1 lần lặp

O(1) tuyên bố

```
/* Sắp xếp bong bóng cho số nguyên */
# xác định SWAP(a,b) { int t; t=a; a=b; b=t; }

bong bóng trống( int a[], int n ) {
  int tôi, j;
  for(i=0;i<n;i++) {/* n truyền qua mảng */ /* Từ
  đầu đến cuối phần chưa được sắp xếp */
        for(j=1;j<(ni);j++) {
        /* Nếu các mục liền kề không đúng thứ tự, hãy hoán đổi */
             if( a[j-1]>a[j] ) SWAP(a[j-1],a[j]);
        }
    }
}
```

Tổng thể *TRÊN2)*

Sắp xếp lựa chọn

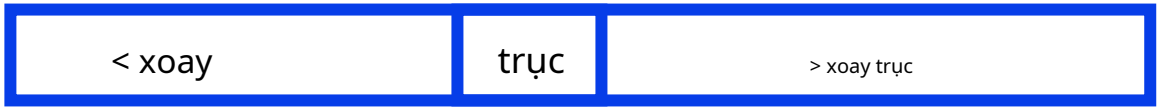
- Thuật toán:
  - ✓ Đi qua các phần tử một cách tuần tự;
  - ✓ bên trong *Tôi quần que* vượt qua, chúng tôi chọn phần tử có giá trị thấp nhất trong A[i] đến A[n], sau đó hoán đổi giá trị thấp nhất với A[i].
- Độ phức tạp thời gian: O(N^2)
- Ví dụ: Sắp xếp danh sách {25, 57, 48, 37, 12}

Sắp xếp nhanh 1/2

- Sắp xếp nhanh, còn được gọi là sắp xếp phân vùng, sắp xếp bằng cách sử dụng chiến lược chia để trị.
- Thuật toán:
  - ✓ Chọn một phần tử trục từ đầu vào;
  - ✓ Phân vùng tất cả các phần tử đầu vào khác sao cho các phần tử nhỏ hơn trục xoay đứng trước trục quay và những phần tử lớn hơn trục xoay đứng sau nó (các giá trị bằng nhau có thể đi theo một trong hai cách);
  - ✓ Sắp xếp đệ quy danh sách các phần tử trước trục và danh sách các phần tử sau trục.
  - ✓ Quá trình đệ quy kết thúc khi danh sách chứa 0 hoặc một phần tử.
- Độ phức tạp thời gian: O(*N*nhật ký*N*) hoặc O(N^2)
- Thử nghiệm:http://pages.stern.nyu.edu/~panos/java/Quicksort/
- Ví dụ: Sắp xếp danh sách {25, 57, 48, 37, 12}

Sắp xếp nhanh 2/2

- Ví dụ củaPhân chia và chinh phụcthuật toán
- Hai giai đoạn
  - ✓ Giai đoạn phân vùng
    - Chiacông việc thành một nửa



  - ✓ Giai đoạn sắp xếp
    - Chinh phụcmột nửa!

< xoay

```
quicksort( void *a, int low, int high )
{ int trục;
if ( cao > thấp ) /* Điều kiện kết thúc! */ {
        trục = phân vùng (a, thấp, cao);
        quicksort(a, low, Pivot-1 );
        quicksort( a, trục+1, cao ); }

}
```

Sắp xếp đống

Heap cũng cung cấp một phương tiện sắp xếp:

- xây dựng một đống,
- thêm từng mục vào đó (duy trì thuộc tính heap!),
- khi tất cả các mục đã được thêm vào, hãy xóa từng mục một (khôi phục thuộc tính heap khi từng mục bị xóa).
- Việc thêm và xóa đều là**O(log*N*)**hoạt động. Chúng ta cần thực hiện ***N*** thêm và bớt, dẫn đến**O(*N*nhật ký*N*)**thuật toán
- Nói chung là chậm hơn

So sánh sắp xếp

chèn	*TRÊN$_2$)*	***Đảm bảo***	
bong bóng	*TRÊN$_2$)*	***Đảm bảo***	
Đống	*TRÊN*nhật ký*N)*	***Đảm bảo***	
Nhanh	*TRÊN*nhật ký*N)*	***Hầu hết thời gian!***	*TRÊN$_2$)*
Thùng rác	*TRÊN)*	***Phím trong phạm vi nhỏ***	*O(n+m)*
Cơ số	*TRÊN)*	***Khóa/trùng lặp bị giới hạn***	*TRÊN*nhật ký*N)*

Thuật toán tìm kiếm

- Hoạt động cơ bản
- Tìm một phần tử trong một tập hợp (rất lớn) các phần tử khác
  - ✓ Mỗi phần tử trong tập hợp có một khóa
- Tìm kiếm là tìm kiếm một phần tử có khóa cho trước
  - ✓ các phần tử riêng biệt có thể có (chia sẻ) cùng một khóa
  - ✓ làm thế nào để xử lý tình huống này?
    - đầu tiên, cuối cùng, bất kỳ, liệt kê, ...
- Có thể sử dụng cấu trúc dữ liệu chuyên dụng
- *Những điều cần cân nhắc*
  - ✓ thời gian trung bình
  - ✓ thời điểm trong trường hợp xấu nhất và
  - ✓ thời gian tốt nhất có thể.

Tìm kiếm tuần tự

- Lưu trữ các phần tử trong một mảng
  - ✓ Không có thứ tự

```
// return first element with key 'k' in 't[]';
// return 'NULL' if not found
// 't[]' is from 1 to 'N'
element find(element* t, int N, int k) {
   t[0].key = k; t[0].value = NULL; // sentinel
   int i = N;
   while (t[i--].key != k);
   // 'i' has been decreased!
   return t[i + 1];
}
```

Phân tích tìm kiếm tuần tự

- Thuật toán đơn giản chung
- Độ phức tạp của không gian: O(1)
- Độ phức tạp thời gian
  - ✓ Thời gian tỷ lệ thuận với *N*
  - ✓ Chúng tôi gọi đây là độ phức tạp thời gian *TRÊN)*
    - Trường hợp xấu nhất: so sánh N + 1
    - Trường hợp tốt nhất: 1 so sánh
    - Trường hợp trung bình (thành công): (1+2+...+N)/N = (N+1)/2
- Cả mảng (chưa sắp xếp) và danh sách liên kết

Tìm kiếm tuần tự trong Danh sách liên kết (được sắp xếp) 1/2

- Giữ danh sách được sắp xếp
  - ✓ Dễ dàng thực hiện với danh sách liên kết (***bài tập: làm đi)!***

```
// trở lại nút đầu tiên có khóa 'k' trong 'l'; // trả về
'NULL' nếu không tìm thấy // 'l' là sắp xếp

nút tìm (danh sách l, int k) {
    nút z = list_end(l); node_setKey(z, k); //
    trọng điểm cho (nút n = list_start(l);

       node_getKey(n) > k;
       n = node_next(n));
    if (node_getKey(n) != k) trả về NULL; trả lại n;

}
```

Tìm kiếm tuần tự trong Danh sách liên kết (được sắp xếp) 2/2

- Độ phức tạp của không gian: O(1)
- Độ phức tạp thời gian
  - ✓ Trường hợp tốt nhất: 1 so sánh
  - ✓ Trường hợp trung bình (thành công): giống như tìm kiếm tuần tự trong danh sách không có thứ tự (mảng): (N+1)/2
  - ✓ Trường hợp xấu nhất (không thành công):
    - coi trọng điểm là một phần của danh sách
    - thì việc tìm kiếm luôn “thành công” (tìm trọng điểmít nhất)
    - Do đó: (N+2)/2

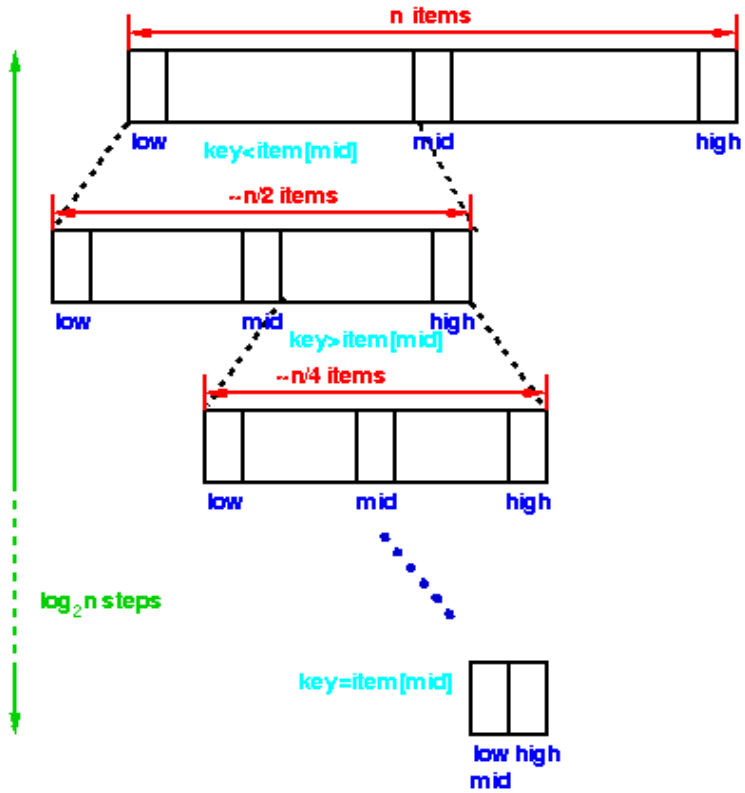
Cải tiến tìm kiếm tuần tự

- Bộ nhớ đệm tĩnh
  - ✓ Sử dụng tần suất truy cập tương đối của các phần tử
    - lưu trữ các phần tử thường được truy cập nhất ở vị trí đầu tiên
- Bộ nhớ đệm động
  - ✓ Đối với mỗi lần truy cập, hãy di chuyển phần tử đến vị trí đầu tiên
    - Cần cấu trúc dữ liệu danh sách liên kết để có hiệu quả
- Rất khó phân tích sự phức tạp về mặt lý thuyết: rất hiệu quả trong thực tế

Tìm kiếm nhị phân

- Mảng được sắp xếp trên một phím
- đầu tiên so sánh khóa với mục ở vị trí giữa của mảng
- Nếu có trận đấu, chúng tôi có thể quay lại ngay lập tức.
- Nếu khóa nhỏ hơn khóa giữa thì mục cần tìm phải nằm ở nửa dưới của mảng
- nếu lớn hơn thì mục tìm kiếm phải nằm ở nửa trên của mảng
- Lặp lại quy trình ở nửa dưới (hoặc trên) của mảng -**đệ quy**

Độ phức tạp thời gian *O*(nhật ký *N*)



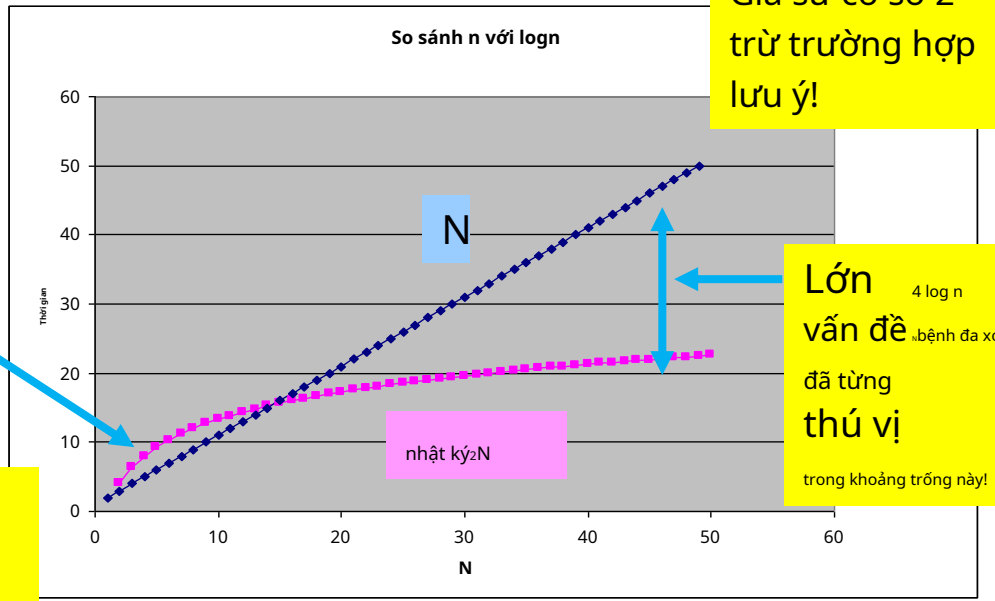
Triển khai tìm kiếm nhị phân

```
static void *bin_search( bộ sưu tập c, int low, int high, void *key ) {
        int giữa;
        nếu (thấp > cao) trả về NULL; /* Kiểm tra kết thúc */
        mid = (cao+thấp)/2;
        switch (memcmp(ItemKey(c->items[mid]),key,c->size)) {
            trường hợp 0: return c->items[mid]; /* So khớp, trả lại mục đã tìm thấy */
            trường hợp -1: return bin_search( c, low, mid-1, key); /* tìm kiếm nửa dưới */ trường
            hợp 1: return bin_search( c, mid+1, high, key ); /* tìm kiếm nửa trên */ mặc định :
            return NULL;
            }
        }
void *FindInCollection( bộ sưu tập c, void *key ) { /*
Tìm một mục trong bộ sưu tập
        Điều kiện trước:
            c là tập hợp được tạo bởi ConsCollection c được
            sắp xếp theo thứ tự tăng dần của khóa key !=
            NULL
        Hậu điều kiện: trả về một mục được xác định bằng khóa nếu có tồn tại, nếu không thì trả về NULL */

             int thấp, cao;
             thấp = 0; cao = c->item_cnt-1; return
            bin_search( c, low, high, key );
         }
```

Tìm kiếm nhị phân và tìm kiếm tuần tự

- **Tìm phương pháp**
  - ✓ **Tìm kiếm tuần tự**
    - Thời điểm xấu nhất: $c_1 N$
  - ✓ **Tìm kiếm nhị phân**
    - Thời điểm xấu nhất: $c_2 \text{nhật ký}_2 N$

Nhật ký
Cơ sở 2 cho đến nay là phổ biến nhất trong khóa học này.
Giả sử cơ sở 2 trừ trường hợp lưu ý!



Bé nhỏ
các vấn đề - đã không thú vị!

Lớn 4 log n
vấn đề „bệnh đa xơ cứng - đã từng thú vị
trong khoảng trống này!

Tìm kiếm nhị phân
Phức tạp hơn
Hệ số không đổi cao hơn

Cảm ơn

Hỏi đáp