

# Cấu trúc dữ liệu và giải thuật

## ÔN TẬP

Giảng viên:

Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

### Nội dung trình bày

2

- Con trỏ
- Đệ quy
- Cấu trúc
- Bài tập

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Nội dung trình bày

3

- Con trỏ

- Đệ quy

- Cấu trúc

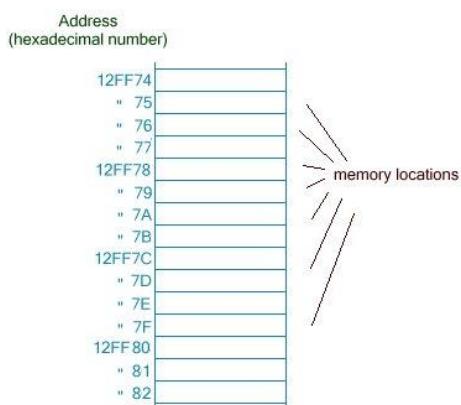
- Bài tập

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ

4

- Địa chỉ trong bộ nhớ:



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

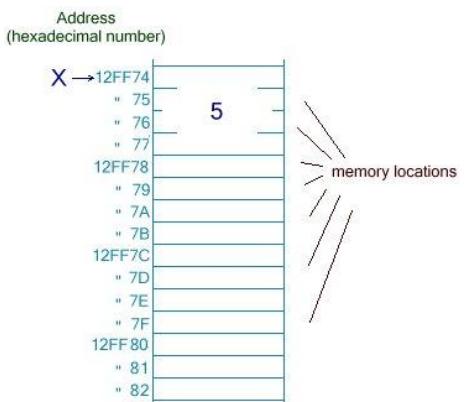
## Con trỏ

5

- Địa chỉ trong bộ nhớ:

int X;

X = 5;



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ

6

- Khái niệm đặc biệt trong C/C++.
- Biến con trỏ: loại biến dùng để **chứa địa chỉ**.
- Khai báo:  
`<KieuDuLieu> *<TenBien>;`

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ

7

### ◎ Ví dụ:

```
int *a;           /*con trỏ đến kiểu int*/  
float *b;         /*con trỏ đến kiểu float*/  
NGAY *pNgay;    /*con trỏ đến kiểu NGAY*/  
SINHVIEN *pSV;  /*con trỏ đến kiểu SINHVIEN*/
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ

8

### ◎ Lưu ý:

- ▣ Xác định địa chỉ ô nhớ: toán tử &
- ▣ Xác định giá trị của ô nhớ tại địa chỉ trong biến con trỏ: toán tử \*
- ▣ Con trỏ NULL.
- ▣ Truy cập thành phần trong cấu trúc: ->

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ

9

### ◦ Cấp phát vùng nhớ động:

- Cấp phát: toán tử **new**.
- Hủy: toán tử **delete**.

### ◦ Ví dụ:

```
int *p;  
p = new int;           //delete p;  
  
p = new int[100];     //delete []p;
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ

10

### ◦ Ví dụ:

```
int i;  
int *p;  
p = &i;  
  
int j;  
j = *p;  
  
int day = pNgay->ngay;
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ - Ví dụ

11

```
#include <stdio.h>

int main()
{
    int i,j;
    int *p;
    p = &i;
    *p = 5;
    j = i;
    printf("%d %d %d\n", i, j, *p);
    return 0;
}
```

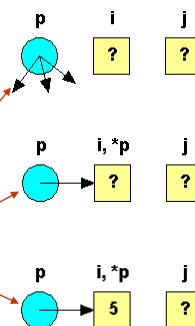
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ - Ví dụ

12

```
#include <stdio.h>

int main()
{
    int i,j;
    int *p; /* a pointer to an integer */
    p = &i;
    *p=5;
    j=i;
    printf("%d %d %d\n", i, j, *p);
    return 0;
}
```



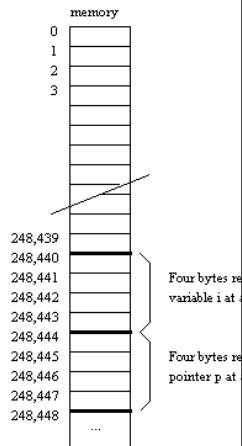
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ - Ví dụ

13

```
#include <stdio.h>

int main()
{
    int i;
    int *p;
    p = &i;
    *p=5;
    printf("%d %d %d %d", i, *p, p, &p);
    return 0;
}
```



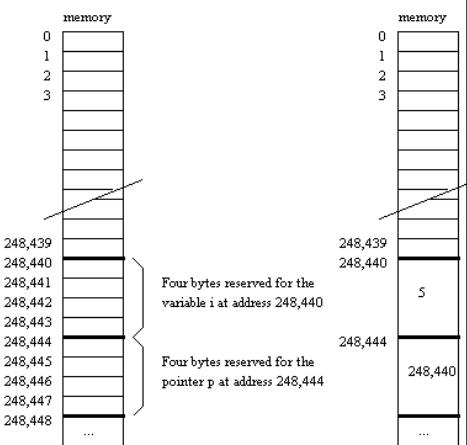
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Con trỏ - Ví dụ

14

```
#include <stdio.h>

int main()
{
    int i;
    int *p;
    p = &i;
    *p=5;
    printf("%d %d %d %d",
    return 0;
}
```



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Nội dung trình bày

15

◦ Con trỏ

◦ **Đệ quy**

◦ Cấu trúc

◦ Bài tập

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Khái niệm

16

◦ Một hàm được gọi là **đệ quy** nếu bên trong thân của hàm đó có **lời gọi hàm lại chính nó** một cách tường minh hay tiềm ẩn.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Chú ý

17

- Khi viết hàm đệ quy, cần xác định:
  - Điều kiện dừng
  - Trường hợp đệ quy

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

18

- Tính tổng  $S(n) = 1 + 2 + \dots + n$
- Ta có:
  - $S(n) = (1 + 2 + \dots + n-1) + n$
  - Trường hợp  $n > 0$ :
    - $S(n) = S(n-1) + n$  (điều kiện đệ quy)
  - Trường hợp  $n = 0$ 
    - $S(0) = 0$  (điều kiện dừng)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

19

- Tính tổng  $S(n) = 1 + 2 + \dots + n$

```
int Tong(int n)
{
    if (n == 0)//điều kiện dừng
        return 0;
    return Tong(n-1) + n;
}
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

20

- Viết hàm tính  $n!$  trong hai trường hợp: không đệ quy và đệ quy. Biết:

- $n! = 1 \times 2 \times 3 \times \dots \times n$
- $0! = 1$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

21

- Tính tổng GiaiThua( $n$ ) =  $1 \times 2 \times \dots \times n$
- Ta có:
  - GiaiThua( $n$ ) =  $(1 \times 2 \times \dots \times n-1) \times n$
  - Trường hợp  $n > 0$ :
    - GiaiThua( $n$ ) = GiaiThua( $n-1$ )  $\times n$  (điều kiện đệ quy)
  - Trường hợp  $n=0$ 
    - GiaiThua( $0$ ) = 1 (điều kiện dừng)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

22

- Cho mảng một chiều các số nguyên. Viết hàm tính tổng các số nguyên có trong mảng bằng phương pháp đệ quy.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

23

- Cho mảng một chiều các số nguyên. Viết hàm tính tổng các số nguyên có trong mảng bằng phương pháp đệ quy.
  - Input: int[] a, int n
  - Output: int (Tổng)
  - Trường hợp đệ quy:
    - $Tong(a, n) = Tong(a, n-1) + a[n-1]$
  - Điều kiện dừng:
    - $Tong(a, 0) = 0$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

24

- Cho mảng một chiều các số nguyên. Viết hàm tính tổng các số lẻ có trong mảng bằng phương pháp đệ quy.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

25

- Cho mảng một chiều các số nguyên. Viết hàm tính tổng các số lẻ có trong mảng bằng phương pháp đệ quy.
  - Input: int[] a, int n
  - Output: int (Tổng)
  - Trường hợp đệ quy:
    - Nếu  $a[n-1]$  lẻ:  $Tong(a, n) = Tong(a, n-1) + a[n-1]$
    - Nếu  $a[n-1]$  chẵn:  $Tong(a, n) = Tong(a, n-1)$
  - Điều kiện dừng:
    - $Tong(a, 0) = 0$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

26

- Viết hàm đệ quy tính số hạng thứ n của dãy Fibonacci. Biết rằng:
  - $f(0) = f(1) = 1$
  - $f(n) = f(n-1) + f(n-2)$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Nội dung trình bày

27

- Con trỏ
- Đệ quy
- **Cấu trúc**
- Bài tập

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cấu trúc

28

- Cấu trúc là phương pháp/cách thức tập hợp các thông tin dữ liệu khác nhau vào trong một dữ liệu.
  - Dễ dàng lưu trữ, truy cập, sử dụng.
  - Định nghĩa thành kiểu dữ liệu riêng
- Ví dụ:
  - NGAY gồm ngay (nguyên), thang (nguyên), nam (nguyên)
  - SINHVIEN gồm mssv (chuỗi), hoten (chuỗi), ngaysinh (NGAY), quequan (chuỗi)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

29

- Thành phần của cấu trúc:
  - Kiểu dữ liệu chuẩn.
  - Kiểu cấu trúc khác.
- Sử dụng từ khóa **struct**.
- Sử dụng như một kiểu dữ liệu tự định nghĩa.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

30

- Định nghĩa cấu trúc:

```
struct <TenCauTruc>{  
    <KieuDuLieu> <ThanhPhan1>;  
    <KieuDuLieu> <ThanhPhan2>;  
    ...  
    <KieuDuLieu> <ThanhPhanN>;  
};
```

- Ví dụ:

```
struct NGAY {  
    int ngay;  
    int thang;  
    int nam;  
};
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cấu trúc

31

- Sử dụng:

**<TenCauTruc> <BienCauTruc>;**

- Ví dụ:

NGAY NgayBatDau, NgayKetThuc;

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cấu trúc

32

- Truy cập thành phần của cấu trúc:

**NGAY** ngaysinh;  
ngaysinh.ngay = 10;  
ngaysinh.thang = 1;  
ngaysinh.nam = 1990;

**SINHVIEN** sv;

...

printf("Ho ten sinh vien : %s", sv.hoten);

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

33

## ◦ Định nghĩa cấu trúc:

- Điểm trên hệ tọa độ Oxy.
- Đoạn thẳng trên hệ tọa độ Oxy.
- Sách trong thư viện

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

34

## ◦ Định nghĩa cấu trúc:

- Điểm trên hệ tọa độ Oxy.

```
struct DIEM {  
    float x;  
    float y;  
};
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

35

- Định nghĩa cấu trúc:

- Điểm trên hệ tọa độ Oxy.

```
struct DOANTHANG {  
    DIEM BatDau;  
    DIEM KetThuc;  
};
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

36

- Một ví dụ

- Nhập vào tọa độ của một điểm và kiểm tra xem điểm này có nằm trên đường thẳng  $y=2x+1$  không?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

37

## o Một ví dụ

- Nhập vào tọa độ của một điểm và kiểm tra xem điểm này có nằm trên đường thẳng  $y=2x+1$  không?

- Nhập điểm:

```
DIEM diem;  
printf("Nhap vao mot diem: \n");  
printf("Toa do x: ");  
scanf("%f", &diem.x);  
printf("Toa do y: ");  
scanf("%f", &diem.y);
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc

38

## o Một ví dụ

- Nhập vào tọa độ của một điểm và kiểm tra xem điểm này có nằm trên đường thẳng  $y=2x+1$  không?

- Kiểm tra:

```
if (diem.y == 2 * diem.x +1)  
    printf("Diem (%f, %f) thuoc duong  
          thang\n", diem.x, diem.y);  
else  
    printf("Diem (%f, %f) khong thuoc duong  
          thang\n", diem.x, diem.y);
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Nội dung trình bày

39

- Con trỏ
- Đệ quy
- Cấu trúc
- **Bài tập**

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

40

- Cho đoạn code sau đây:

```
int i;
int *p, *q, *r;
p = &i;
q = &i;
r = p;
```
- Nếu  $*r = 5$ , hỏi  $*p$ ,  $*q$  có giá trị bao nhiêu?
- Nếu  $i = 20$  thì  $*r$  có giá trị bao nhiêu?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

41

- Cho mảng một chiều các số nguyên. Viết hàm để quy xuất mảng.
- Cho mảng một chiều các số nguyên. Viết hàm để quy xuất mảng theo thứ tự ngược (từ phải sang trái).

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

42

- Cho mảng một chiều các số nguyên. Viết hàm đếm số lượng các phần tử dương có trong mảng.
- Cho mảng một chiều các số nguyên. Viết hàm đếm số lượng các phần tử âm có trong mảng.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

43

- Cho mảng một chiều các số nguyên. Viết hàm đệ quy kiểm tra mảng có thỏa mãn tính chất 'tổng giá trị âm' hay không?
- Cho mảng một chiều các số nguyên. Viết hàm đệ quy tìm giá trị lớn nhất có trong mảng.
- Cho mảng một chiều các số nguyên. Viết hàm đệ quy tìm vị trí của phần tử có giá trị lớn nhất có trong mảng.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

44

## Hỏi và đáp

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc dữ liệu và giải thuật

## CÁC KHÁI NIỆM CƠ BẢN

Giảng viên:  
Văn Chí Nam – Nguyễn Thị Hồng Nhụng – Đặng Nguyễn Đức Tiến

### Tài liệu tham khảo

2

- Kenneth H.Rosen, *Toán rời rạc ứng dụng trong Tin học*, ltb. 5, nxb. Giáo Dục, 2007, tr. 131 - 143.
- Mark A. Weiss, *Data Structures & Algorithm Analysis in C++*, 2<sup>nd</sup> edition, Addison Wesley, 1998, p. 41 – 67.

## Nội dung

3

Tổng quan về cấu trúc dữ liệu

Tiêu chuẩn đánh giá thuật toán

Độ tăng của hàm

Độ phức tạp thuật toán

Các phương pháp đánh giá độ phức tạp

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Dẫn nhập

4

- According to [Peter J. Denning](#), the fundamental question underlying computer science is, "*What can be (efficiently) automated?*"

[[Wikipedia.org](#), tháng 9 – 2009]

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Dẫn nhập

5

- Để giải quyết nhu cầu tự động hóa, nhu cầu căn bản của Khoa học Máy tính, các nhà khoa học máy tính phải tạo ra sự **trùng tượng hóa** về những bài toán trong thế giới thực,
  - để người sử dụng máy tính có thể hiểu được
  - và có thể biểu diễn và xử lý được bên trong máy tính.

- **Ví dụ:**

- Mô hình hóa việc biểu diễn cầu thủ bóng đá
- Mô hình hóa mạch điện
- ...

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Dẫn nhập

6

- Thông thường, tìm ra một sự trùng tượng hóa thường rất khó, vì:
  - Giới hạn về khả năng xử lý của máy.
  - Phải cung cấp cho máy một mô hình về thế giới đến mức chi tiết như những gì con người có, không chỉ là sự kiện mà còn cả các nguyên tắc và mối liên hệ.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Trùu tượng hóa: sự đơn giản hóa

7

- Sự trùu tượng hóa ở đây được sử dụng là **sự đơn giản hóa**, thay thế một tình huống phức tạp và nhiều chi tiết trong thế giới thực bằng một mô hình dễ hiểu để chúng ta có thể giải quyết được bài toán trong đó.
- Có thể hiểu là chúng ta **loại bỏ** những chi tiết có tác dụng rất ít hoặc không có tác dụng gì đối với lời giải của bài toán
  - > tạo ra một mô hình cho phép chúng ta giải quyết với **bản chất** của bài toán.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Mô hình dữ liệu

8

- Mô hình dữ liệu (data model) là các trùu tượng dùng để mô tả bài toán, thông thường là mô tả cách thức mà **dữ liệu** (data) được **biểu diễn** (represented) và **truy xuất** (accessed) như thế nào.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Kiểu dữ liệu

9

- Kiểu dữ liệu (của biến) là một khái niệm trong lập trình, chỉ tập các giá trị mà biến có thể chấp nhận.
- Ví dụ:
  - Kiểu dữ liệu kiểu số nguyên,
  - Kiểu dữ liệu kiểu số thực,
  - Kiểu dữ liệu chuỗi.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Kiểu dữ liệu cơ bản

10

- Kiểu dữ liệu sơ cấp là kiểu dữ liệu mà giá trị của nó là đơn nhất.
  - Ví dụ: Trong ngôn ngữ lập trình C chuẩn, kiểu **int** gọi là kiểu sơ cấp vì kiểu này bao gồm các số nguyên từ -32768 đến 32767 và các phép toán +, -, \*, /, %...
- Mỗi ngôn ngữ đều có cung cấp sẵn các kiểu dữ liệu cơ bản (basic data type), gọi là kiểu dữ liệu chuẩn.
  - Ví dụ, trong ngôn ngữ C thì các kiểu sau là kiểu dữ liệu cơ bản: int, char, float...

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Kiểu dữ liệu có cấu trúc

11

- Kiểu dữ liệu có cấu trúc (Structured Data Type): là kiểu dữ liệu mà giá trị của nó là sự kết hợp các giá trị khác.
  - Ví dụ:
    - Kiểu dữ liệu có cấu trúc gồm các giá trị giao dịch của một phiên giao dịch (chứng khoán).
    - Kiểu dữ liệu mô tả lí lịch sinh viên.
    - ...
  - Còn được gọi là *kiểu dữ liệu tổ hợp*.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Kiểu dữ liệu trừu tượng

12

- Kiểu dữ liệu trừu tượng (abstract data type - ADT) là một mô hình toán kết hợp với các phép toán trên mô hình này.
  - ADT là sự trừu tượng các kiểu dữ liệu cơ bản (nguyên, thực,...) và các thủ tục là sự trừu tượng các phép toán nguyên thủy (+, -, ...).
  - Có thể xem ADT tương đương với khái niệm *mô hình dữ liệu* áp dụng trong lập trình.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cấu trúc dữ liệu

13

- Cấu trúc dữ liệu là các đơn vị cấu trúc của ngôn ngữ lập trình dùng để biểu diễn các mô hình dữ liệu. Ví dụ như mảng (array), tập tin (file), danh sách liên kết (linked list)...
- Các cấu trúc dữ liệu được chọn phải có khả năng biểu diễn được tập input và output của bài toán cần giải. Hơn nữa, phải phù hợp với các thao tác của thuật toán và cài đặt được bằng ngôn ngữ lập trình đã được lựa chọn.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Mô hình dữ liệu và Cấu trúc dữ liệu

14

- Mặc dù tên nghe có vẻ giống nhau, “danh sách” và “danh sách liên kết” là những khái niệm khác nhau.
  - Danh sách là Mô hình dữ liệu.
  - Danh sách liên kết là một cấu trúc dữ liệu. Cụ thể, đây là một cấu trúc thường được sử dụng trong C.
  - Có những ngôn ngữ không cần phải dùng đến một cấu trúc dữ liệu để biểu diễn một Danh sách (Mô hình dữ liệu). Ví dụ danh sách ( $a_1, a_2, \dots, a_n$ ) có thể được biểu diễn trực tiếp trong ngôn ngữ Lisp dưới dạng  $[a_1, a_2, \dots, a_n]$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Chương trình

15



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Tiêu chuẩn đánh giá thuật toán

16

- Tốc độ thực thi.
- Tính chính xác.
- Đơn giản, dễ hiểu, dễ bảo trì.
- Mức phổ dụng
- ...



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thời gian giải quyết 1 bài toán?

17

- Thời gian giải quyết một bài toán phụ thuộc vào nhiều yếu tố:
  - Tốc độ thực thi của máy tính (phần cứng lẫn phần mềm).
  - Tài nguyên (ví dụ: bộ nhớ).
  - Thuật toán.



Làm thế nào đánh giá được thời gian thực thi hiệu quả?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Đánh giá thời gian thực thi theo phép toán

18

- Đánh giá thời gian thực hiện dựa trên những phép toán quan trọng như:
  - Phép so sánh
  - Phép gán
- Đánh giá bằng cách tính số lượng các phép toán quan trọng theo **độ lớn của dữ liệu**.



Từ đó, thời gian thực hiện của một thuật toán có thể được đánh giá theo một hàm phụ thuộc vào độ lớn đầu vào.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

19

- Bước 1. Gán  $Tổng = 0$ . Gán  $i = 0$ .
- Bước 2.
  - Tăng  $i$  thêm 1 đơn vị.
  - Gán  $Tổng = Tổng + i$
- Bước 3. So sánh  $i$  với 10
  - Nếu  $i < 10$ , quay lại bước 2.
  - Ngược lại, nếu  $i \geq 10$ , dừng thuật toán.



Số phép gán của thuật toán là bao nhiêu? Số phép so sánh là bao nhiêu?

- Gán:  $f(2n + 2)$ , So sánh:  $f(n)$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ tăng của hàm

20

- Big-O.
- Một số kết quả Big-O quan trọng.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Nguồn gốc lịch sử

21

- Khái niệm Big-O lần đầu tiên được đưa ra bởi nhà toán học người Đức Paul Bachmann vào năm 1892.
- Big-O được trở nên phổ biến hơn nhờ nhà toán học Landau. Do vậy, Big-O cũng còn được gọi là ký hiệu Landau, hay Bachmann-Landau.
- Donald Knuth được xem là người đầu tiên truyền bá khái niệm Big-O trong tin học từ những năm 1970. Ông cũng là người đưa ra các khái niệm Big-Omega và Big-Theta.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Định nghĩa toán học của Big-O

22

- Cho  $f$  và  $g$  là hai hàm số từ tập các số nguyên hoặc số thực đến số thực. Ta nói  $f(x)$  là  $O(g(x))$  nếu tồn tại hằng số  $C$  và  $k$  sao cho:

$$|f(x)| \leq C |g(x)| \text{ với mọi } x > k$$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Định nghĩa toán học của Big-O

23

- Cho  $f$  và  $g$  là hai hàm số từ tập các số nguyên hoặc số thực đến số thực. Ta nói  $f(x)$  là  $O(g(x))$  nếu tồn tại hằng số  $C$  và  $k$  sao cho:

$$|f(x)| \leq C |g(x)| \text{ với mọi } x > k$$

- Ví dụ, hàm  $f(x) = x^2 + 3x + 2$  là  $O(x^2)$ .

Thật vậy, khi  $x > 2$  thì  $x < x^2$  và  $2 < 2x^2$

Do đó  $x^2 + 3x + 2 < 6x^2$ .

Nghĩa là **ta chọn được  $C = 6$  và  $k = 2$** .

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ý nghĩa của Big-O (1)

24

- Big-O giúp xác định được mối quan hệ giữa  $f(x)$  và  $g(x)$ , trong đó  **$g(x)$  thường là hàm ta đã biết trước**. Từ đó ta xác định được sự tăng trưởng của hàm  $f(x)$  cần khảo sát.
- $C$  và  $k$  trong định nghĩa của khái niệm Big-O được gọi là **bằng chứng** của mối quan hệ  $f(x)$  là  $O(g(x))$ .

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ý nghĩa của Big-O (2)

25

- Big-O phân hoạch được các hàm với các độ tăng khác nhau. Nếu có hai hàm  $f(x)$  và  $g(x)$  sao cho  $f(x)$  là  $O(g(x))$  và  $g(x)$  là  $O(f(x))$  thì ta nói hai hàm  $f(x)$  và  $g(x)$  đó là có **cùng bậc**.
- Ví dụ:  $f(x) = 7x^2$  là  $O(x^2)$  (chọn  $k = 0, C = 7$ ).  
Do vậy  $7x^2$  và  $x^2 + 3x + 2$ , và  $x^2$  là 3 hàm có cùng bậc.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ý nghĩa của Big-O (3)

26

- Lưu ý:  $7x^2$  cũng là  $O(x^3)$  nhưng  $x^3$  không là  $O(7x^2)$ .  
*Thật vậy: Nếu  $x^3$  là  $O(7x^2)$  thì ta phải tìm được  $C$  và  $k$  sao cho*  
 $|x^3| \leq C|7x^2| \Leftrightarrow x \leq 7C$  với mọi  $x > k$ .  
*Điều này không thể xảy ra vì không thể tìm được  $k$  và  $C$  nào như vậy.*
- 💡 Do vậy, trong quan hệ  $f(x)$  là  $O(g(x))$ , hàm  $g(x)$  thường được chọn là **nhỏ nhất có thể**.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Một số kết quả Big-O quan trọng

27

1. Hàm đa thức:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

Khi đó  $f(x)$  là  $O(x^n)$ .

2. Hàm giai thừa:

$$f(n) = n! \text{ là } O(n^n)$$

3. Logarit của hàm giai thừa:

$$f(n) = \log n! \text{ là } O(n \log n)$$

4. Hàm điều hòa

$$H(n) = 1 + 1/2 + 1/3 + \dots + 1/n \text{ là } O(\log n)$$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ tăng của tổ hợp các hàm

28

Cho  $f_1(x)$  là  $O(g_1(x))$  và  $f_2(x)$  là  $O(g_2(x))$ .

Khi đó:

Quy tắc tổng:

$$(f_1 + f_2)(x) \text{ là } O(\max(|g_1(x)|, |g_2(x)|))$$

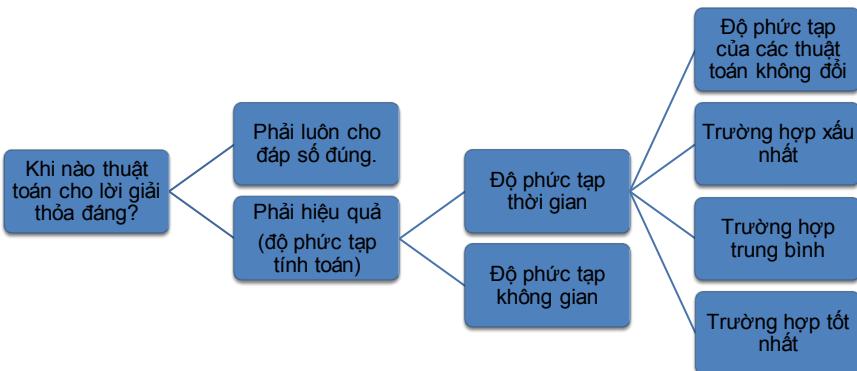
Quy tắc nhân:

$$(f_1 f_2)(x) \text{ là } O(g_1(x) g_2(x)).$$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp thuật toán

29



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp cố định của thuật toán

30

### ○ Thuật toán:

- ▣ B1. Đặt giá trị cực đại tạm thời bằng số nguyên đầu tiên trong dãy.
- ▣ B2. So sánh số nguyên tiếp sau với giá trị cực đại tạm thời. Nếu nó lớn hơn giá trị cực đại tạm thời thì đặt cực đại tạm thời bằng số nguyên đó.
- ▣ B3. Lặp lại B2 nếu còn các số nguyên trong dãy.
- ▣ B4. Dừng khi không còn số nguyên nào nữa trong dãy. Cực đại tạm thời chính là số nguyên lớn nhất của dãy.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp cố định của thuật toán

31

- Vì phép so sánh sử dụng trong thuật toán là phép so sánh, nên phép so sánh được dùng làm thước đo độ phức tạp.
- Tại mỗi số hạng, ta thực hiện 2 phép so sánh, 1 phép xem đã hết dãy hay chưa và 1 phép so với cực đại tạm thời.
- Vì hai phép so sánh được dùng từ số hạng thứ 2 đến n, và thêm 1 phép so sánh nữa để ra khỏi vòng lặp, nên ta có chính xác  $2(n-1) + 1 = 2n - 1$  phép so sánh.
- Do vậy, độ phức tạp của thuật toán là  $O(n)$ .

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp trong trường hợp xấu nhất

32

- Bước 1. Gán  $i = 1$ .
- Bước 2. Trong khi  $i \leq n$  và  $x \neq a_i$  thì tăng  $i$  thêm 1.  
$$\text{while } (i \leq n \text{ and } x \neq a_i)$$
$$i = i + 1$$
- Bước 3.
  - Nếu  $i \leq n$ , trả về giá trị là  $i$ .
  - Ngược lại,  $i > n$ , trả về giá trị 0 cho biết không tìm được  $x$  trong dãy  $a$ .

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp trong trường hợp xấu nhất

33

- Số phép so sánh dùng làm thước đo.
- Ở mỗi bước của vòng lặp, thực hiện 2 phép so sánh.
- Cuối vòng lặp, thực hiện 1 phép so sánh.
- Như vậy, nếu  $x = a_i$ , số phép so sánh thực hiện là  $(2i + 1)$ .
- Trong trường hợp xấu nhất, không tìm được  $x$  thì tổng số phép so sánh là  $2n + 2$ .
- Từ đó, thuật toán tìm kiếm tuần tự đòi hỏi tối đa  $O(n)$  phép so sánh.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp trong trường hợp tốt nhất

34

- Trong trường hợp tốt nhất, ta bắt gặp  $x$  ngay phần tử đầu tiên nên chỉ cần tốn 3 phép so sánh.
- Khi đó, ta nói thuật toán tìm kiếm tuần tự đòi hỏi ít nhất  $O(1)$  phép so sánh.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Độ phức tạp trong trường hợp trung bình

35

- Nếu  $x$  là số hạng thứ  $i$ , phép so sánh sử dụng để tìm ra  $x$  là  $2i + 1$ .
- Do đó, phép so sánh trung bình ta cần sử dụng là:

$$\frac{3+5+7+\dots+(2n+1)}{n} = \frac{2(1+2+3+\dots+n)+n}{n} = \frac{2\frac{n(n+1)}{2}+n}{n} = n+2$$

- Như vậy độ phức tạp trung bình của thuật toán tìm kiếm tuần tự là  $O(n)$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ghi chú

36

- Trong thực tế, các phép so sánh cần để xác định xem đã tới cuối vòng lặp hay chưa thường được bỏ qua, không đếm.
- Trong đa số các trường hợp không đòi hỏi sự khắt khe về tính chính xác, người ta sử dụng Big-O cho mọi trường hợp.
- Hệ số trong các hàm theo đa thức không được tính trong phân tích độ phức tạp, ví dụ  $O(n^3)$  và  $O(20000n^3)$  là như nhau, nhưng trong thực tế đôi khi hệ số rất quan trọng.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Sự phân lớp các độ phức tạp

37

Độ phức tạp	Thuật ngữ/tên phân lớp
$O(1)$	Độ phức tạp hằng số
$O(\log_2 n)$	Độ phức tạp logarit
$O(n)$	Độ phức tạp tuyến tính
$O(n \log_2 n)$	Độ phức tạp $n \log_2 n$
$O(n^a)$	Độ phức tạp đa thức
$O(a^n)$ , $a > 1$	Độ phức tạp hàm mũ
$O(n!)$	Độ phức tạp giai thừa

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Sự phân lớp các độ phức tạp

38

	logn	n	nlogn	$n^2$	$2^n$	n!
$10$	$3 \cdot 10^{-9}$	$10^{-8}$	$3 \cdot 10^{-8}$	$10^{-7}$	$10^{-6}$	$3 \cdot 10^{-3}$
$10^2$	$7 \cdot 10^{-9}$	$10^{-7}$	$7 \cdot 10^{-7}$	$10^{-5}$	$4 \cdot 10^{13}$ năm	*
$10^3$	$1,0 \cdot 10^{-8}$	$10^{-6}$	$1 \cdot 10^{-5}$	$10^{-3}$	*	*
$10^4$	$1,3 \cdot 10^{-8}$	$10^{-5}$	$1 \cdot 10^{-4}$	$10^{-1}$	*	*
$10^5$	$1,7 \cdot 10^{-8}$	$10^{-4}$	$2 \cdot 10^{-3}$	10	*	*
$10^6$	$2 \cdot 10^{-8}$	$10^{-3}$	$2 \cdot 10^{-2}$	17 phút	*	*

- Lưu ý:
  - Mỗi phép toán giả sử thực hiện trong  $10^{-9}$  giây (~ CPU 1GHz).
  - \*: thời gian lớn hơn  $100^{100}$  năm

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Một số lưu ý mở rộng

39

- Có một số thuật toán có độ phức tạp trong trường hợp xấu nhất là rất lớn nhưng trong trường hợp trung bình lại chấp nhận được.
- Đôi khi, trong thực tế ta phải tìm nghiệm gần đúng thay vì nghiệm chính xác.
- Có một số bài toán tồn tại nhưng có thể chứng minh được không có lời giải cho chúng (ví dụ bài toán Halting).
- Trong thực tế, đa số ta chỉ khảo sát các bài toán có độ phức tạp đa thức trở xuống.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các phương pháp đánh giá độ phức tạp

40

- Phương pháp đếm
- Phương pháp hàm sinh
- Một số kết quả hoán vị
- Các kết quả, định lý liên quan đến các cấu trúc dữ liệu cụ thể
- ...

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

41

1. Các hàm sau đây có là  $O(x)$  hay không?

- a)  $f(x) = 10$
- b)  $f(x) = 3x + 7$
- c)  $f(x) = 2x^2 + 2$

2. Mô tả thuật toán tìm số nhỏ nhất trong dãy hữu hạn các số tự nhiên. Có bao nhiêu phép so sánh, bao nhiêu phép gán trong thuật toán?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

42

3. Phân tích độ phức tạp của thuật toán tính tổng dãy số sau:

$$S = 1 + \frac{1}{2} + \frac{1}{6} + \dots + \frac{1}{n!}$$

4. Cho biết số phép gán, số phép so sánh trong đoạn code sau đây theo n:

```
sum = 0;
for (i = 0; i < n; i++)
{
    scanf("%d", &x);
    sum = sum + x;
}
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

43

5. Cho biết số phép gán, số phép so sánh trong đoạn code sau đây theo n:

```
for (i = 0; i < n ; i++)
    for (j = 0; j < n; j++)
    {
        C[i][j] = 0;
        for (k = 0; k < n; k++)
            C[i][j] = C[i][j] +
                        A[i][k]*B[k][j];
    }
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

44

6. Hãy cho biết các hàm g(n) cho các hàm f(n) dưới đây ( $f(n) = O(g(n))$ ).

- ▣  $f(n) = (2 + n) * (3 + \log_2 n)$
- ▣  $f(n) = 11 * \log_2 n + n/2 - 3542$
- ▣  $f(n) = n * (3 + n) - 7 * n$
- ▣  $f(n) = \log_2(n^2) + n$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

45

## Hỏi và Đáp

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc dữ liệu và giải thuật

## CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

Giảng viên:

Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

### Nội dung trình bày

2

Danh sách liên kết

Ngăn xếp

Hàng đợi

3

## Danh sách liên kết

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

4

## Nội dung

- Giới thiệu
- Các loại danh sách liên kết
- Các thao tác trên danh sách liên kết
- So sánh danh sách liên kết và mảng
- Ứng dụng

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giới thiệu

5

- Mảng: cấu trúc dữ liệu quen thuộc

- Tập có thứ tự
  - Số lượng phần tử cố định (tĩnh)
  - Cấp phát vùng nhớ liên tục
  - Truy xuất phần tử thông qua chỉ số

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giới thiệu

6

- Đánh giá thao tác trên mảng:

- Truy xuất phần tử?
  - Cập nhật?
  - Chèn phần tử?
  - Xoá phần tử?

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Giới thiệu

7

## ◦ Thực tế:

- Không xác định được chính xác số lượng phần tử
  - Danh sách bệnh nhân: tăng/giảm.
  - Danh sách sinh viên: tăng/giảm.
- Vùng nhớ thay đổi trong quá trình sử dụng  
=> Không đủ vùng nhớ cấp phát liên tục.

=> Cấu trúc dữ liệu động đáp ứng nhu cầu

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Các loại danh sách liên kết

8

## ◦ Danh sách liên kết đơn

- singly linked list
- uni-directional linked list

## ◦ Danh sách liên kết kép

- doubly linked list
- bi-directional linked list

## ◦ Danh sách liên kết vòng

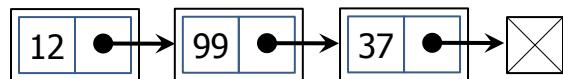
- circularly linked list
- ring list

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Danh sách liên kết đơn

9

- Mỗi phần tử có MỘT liên kết đến phần tử phía sau nó.

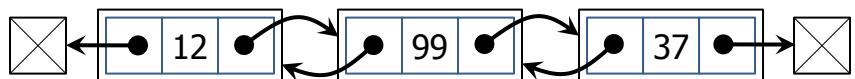


Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Danh sách liên kết kép

10

- Mỗi phần tử có HAI liên kết đến phần tử đứng sau và trước nó.

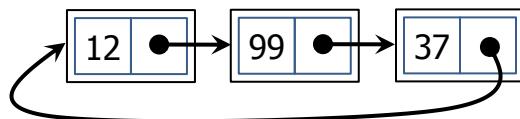


Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Danh sách liên kết vòng

11

- Có mối liên kết giữa phần tử cuối và phần tử đầu



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

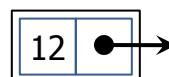
## Phần tử trên danh sách liên kết

12

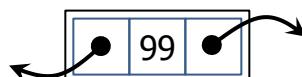
- Phần tử (Node, Element)
  - ▣ Phần tử = Dữ liệu + Liên kết

- ▣ Ví dụ:

- Phần tử có 1 liên kết



- Phần tử có 2 liên kết



- Phần tử rỗng



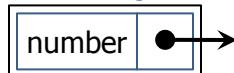
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ

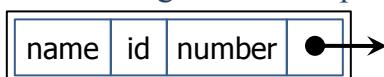
13

- Ví dụ:

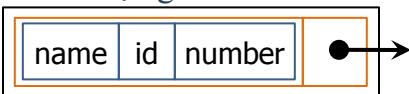
- Phần tử có dữ liệu gồm 1 thành phần



- Phần tử có dữ liệu gồm 3 thành phần



- Phần tử có dữ liệu gồm 1 cấu trúc



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Cài đặt

14

- Sinh viên tự viết phần cài đặt cho các ví dụ trên

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tổ chức

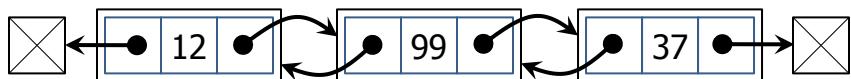
15

- Mỗi danh sách liên kết bao gồm:

- Con trỏ đến phần tử đầu (hoặc/và cuối) danh sách.

- (Các) phần tử trên danh sách

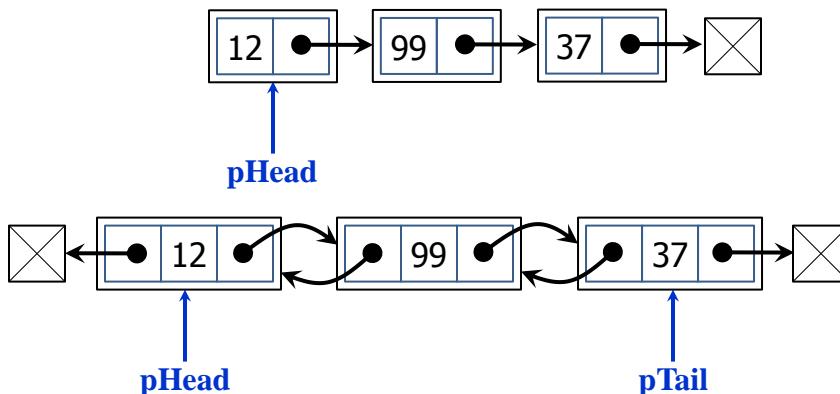
- Dữ liệu
  - Các mối liên kết



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tổ chức

16



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Các thao tác trên danh sách liên kết

17

- Thêm phần tử
- Duyệt danh sách
- Xoá phần tử
- Truy xuất phần tử
- Xoá danh sách

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thêm phần tử

18

- Vào đầu danh sách
- Sau một phần tử
- Vào cuối danh sách

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

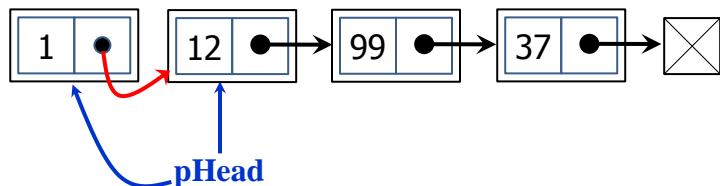
## Thêm phần tử

19

### ◦ Vào đầu danh sách:

- Nếu danh sách rỗng
  - Phần tử vừa thêm là phần tử đầu danh sách

### ▫ Ngược lại,



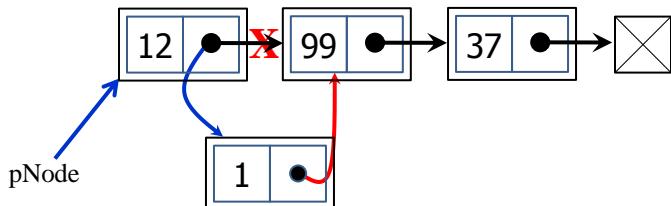
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thêm phần tử

20

### ◦ Sau một phần tử (pNode):

- Nếu danh sách rỗng?
- Nếu danh sách khác rỗng?
  - Tạo node mới có dữ liệu là Data.
  - Cập nhật lại liên kết của pNode và node vừa tạo.



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Duyệt danh sách

21

- Đảm bảo việc truy xuất đến tất cả các phần tử trên danh sách
- Thuật toán:
  - Bắt đầu từ phần tử đầu tiên
  - Trong khi chưa hết danh sách
    - Xử lý phần tử hiện hành
    - Di chuyển đến phần tử kế tiếp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Xoá phần tử

22

- Đầu danh sách
- Cuối danh sách
- Sau một phần tử
- Theo khóa k

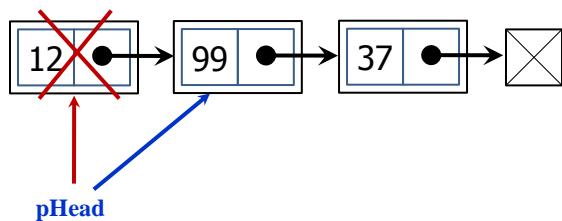
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Xoá phần tử

23

### ○ Đầu danh sách

- Nếu danh sách rỗng:
- Nếu danh sách khác rỗng:
  - Cập nhật lại pHead
  - Xóa con trỏ pHead cũ



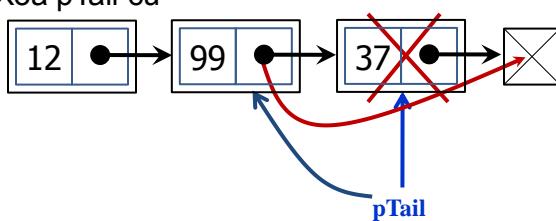
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Xoá phần tử

24

### ○ Cuối danh sách:

- Danh sách rỗng?
- Danh sách khác rỗng:
  - tìm con trỏ cuối danh sách pTail
  - Cập nhật lại pTail
  - Xóa pTail cũ



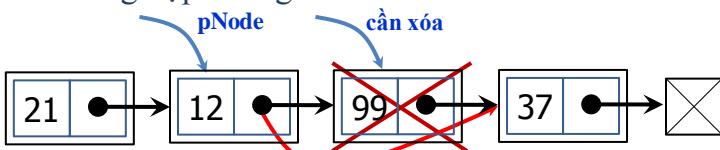
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Xoá phần tử

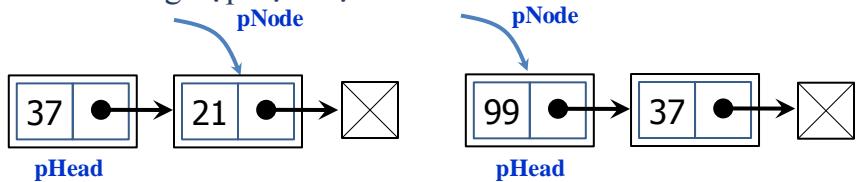
25

- Sau một phần tử (pNode)

- Trường hợp chung:



- Trường hợp đặc biệt:



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Xoá danh sách

26

- Danh sách liên kết bao gồm các phần tử được cấp phát động.

- Phải xoá các phần tử trên danh sách sau khi đã sử dụng xong danh sách.

- Thuật toán:

- Duyệt qua các phần tử trên danh sách
  - Xoá từng phần tử

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Danh sách liên kết là gì?

27

- Một dãy tuần tự các phần tử (node)
- Giữa hai phần tử có liên kết với nhau.
- Các phần tử không cần phải lưu trữ liên tiếp nhau trong bộ nhớ
- Có thể mở rộng tùy ý (chỉ giới hạn bởi dung lượng bộ nhớ)
- Thao tác Chèn/Xóa không cần phải dịch chuyển phần tử
- Có thể truy xuất đến các phần tử khác thông qua các liên kết

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## So sánh danh sách liên kết và mảng

28

### Danh sách liên kết

- Số phần tử không cần xác định trước.
- Cấp phát vùng nhớ riêng lẻ cho từng phần tử.
- Truy xuất tuần tự, danh sách liên kết đơn chỉ có thể duyệt 1 chiều.
- Cần nhiều bộ nhớ hơn để lưu trữ các liên kết
- Thêm/xóa phần tử cuối: O(1)
- Thêm/xóa phần tử giữa: O(1)

### Mảng

- Cần xác định trước số phần tử.
- Cần cấp phát vùng nhớ liên tục đủ lớn để lưu trữ mảng → lãng phí nếu không dùng hết.
- Truy xuất ngẫu nhiên, đơn giản, nhanh chóng.
- Không cần
- Thêm/xóa phần tử cuối: O(1)
- Thêm/xóa phần tử giữa: O(n)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ứng dụng

29

- Là cấu trúc dữ liệu chính cho ngôn ngữ lập trình LISP (List Processing Language) – ngôn ngữ lập trình hàm.
- Giúp nâng cao hiệu quả của một số thuật toán sắp xếp: Quick Sort, Radix Sort

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập 1

30

- Cho một DSLK đơn, mỗi node trong DSLK lưu thông tin là 1 số nguyên và con trỏ đến node kế. Tạo 2 DSLK đơn mới (không phá huỷ DSLK đã cho).
  - ▣ Một danh sách chứa các số lẻ của danh sách đã cho.
  - ▣ Một danh sách chứa các số chẵn của danh sách đã cho.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập 1

31

- In ra các *đường chạy* tự nhiên từ DSLK đã cho:

VÍ DỤ: DSLK ban đầu biểu diễn các số: 1 5 6 4 8 3  
7

In ra các dãy số: 1 5 6

4 8

3 7

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập 2

32

- Cho danh sách liên kết đơn L, lập giải thuật thực hiện các phép sau đây:

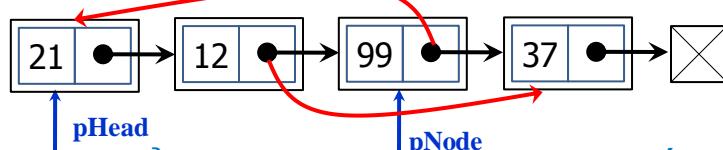
- Tính số lượng các nút của danh sách.
- Tìm tới nút thứ k trong danh sách, nếu có nút thứ k thì cho biết địa chỉ của nút đó, ngược lại trả về null.
- Bổ sung một nút vào sau nút k.
- Loại bỏ nút đứng trước nút k.
- Đảo ngược danh sách đã cho.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập 3

33

- Hàm **MoveToFront** có tác dụng di chuyển 1 node trong xâu lên đầu xâu, như hình sau:



- Chọn kiểu khai báo hàm phù hợp và viết code  
`void MoveToFront(NODE * pHed, NODE * pTail, NODE * pNode )`

Lưu ý: các kí hiệu `*` có thể là `*`, `&` hoặc khoảng trắng

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

34

## Ngăn xếp - stack

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Nội dung

35

- Giới thiệu
- Các thao tác cơ bản
- Ký pháp Ba Lan

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giới thiệu

36

- Một số hình ảnh thông dụng:

- Một chồng sách vở ở trên bàn



- Một chồng đĩa
  - Cơ cấu của một hộp chứa đạn súng trường.



Nhận xét gì từ các ví dụ trên?

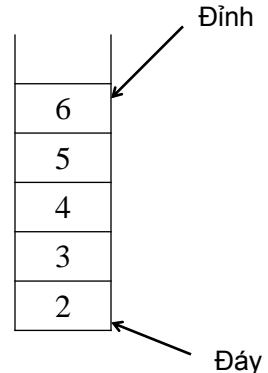
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Giới thiệu

37

## ◦ Định nghĩa:

- Ngăn xếp là vật chứa các đối tượng làm việc theo cơ chế “vào sau ra trước” (Last In First Out)
- Đối tượng có thể được thêm vào bất kì lúc nào, nhưng chỉ có *đối tượng vào sau cùng mới được phép lấy ra khỏi ngăn xếp.*



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Các thao tác trên ngăn xếp

38

## ◦ Các thao tác cơ bản:

- Push: Thêm 1 phần tử vào ngăn xếp
- Pop: Lấy 1 phần tử ra khỏi ngăn xếp

## ◦ Các thao tác khác:

- Lưu trữ ngăn xếp
- Kiểm tra ngăn xếp rỗng
- Lấy thông tin của phần tử đầu ngăn xếp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lưu trữ ngăn xếp

39

### ◦ Lưu trữ bằng mảng

- Khai báo mảng 1 chiều với kích thước tối đa N.
- t là địa chỉ của phần tử đỉnh của ngăn xếp → t sẽ thay đổi khi ngăn xếp hoạt động.
  - Ngăn xếp rỗng thì giá trị của t là 0
- Tạo ngăn xếp S và quản lý ngăn xếp bằng biến t:  
Data S[N];  
int t;

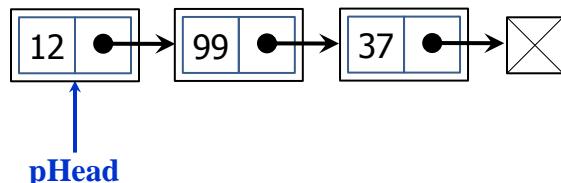
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lưu trữ ngăn xếp

40

### ◦ Lưu trữ bằng DSLK:

- Dùng con trỏ pHead lưu địa chỉ của đỉnh ngăn xếp
- Ngăn xếp rỗng khi pHead = NULL



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kiểm tra ngăn xếp rỗng

41

- Input:

- Output:

- TRUE nếu ngăn xếp rỗng
- FALSE nếu ngăn xếp không rỗng

- Ngăn xếp rỗng:

- Mảng: số lượng phần tử mảng là 0
- DSLK: pHead = NULL

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kiểm tra ngăn xếp đầy

42

- Input:

- Output:

- TRUE nếu ngăn xếp đầy
- FALSE nếu ngăn xếp còn chỗ trống

- Ngăn xếp đầy:

- Mảng: đã lưu hết các phần tử mảng
- DSLK: không cấp phát được vùng nhớ mới cho ngăn xếp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thêm phần tử vào ngăn xếp (push)

43

- Input: phần tử cần thêm

- Output:

- Giải thuật:

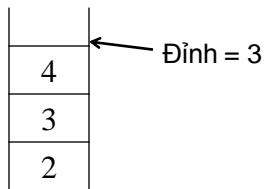
- Kiểm tra ngăn xếp có đầy không?
- Nếu không
  - Bổ sung phần tử mới vào
  - Cập nhật địa chỉ của con trỏ đến đỉnh ngăn xếp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

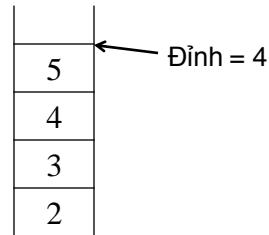
## Thêm phần tử vào ngăn xếp (push)

44

- Ví dụ:



Ngăn xếp ban đầu



Ngăn xếp sau khi thêm push(5)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lấy phần tử ra khỏi ngăn xếp (pop)

45

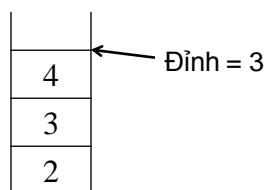
- Input:
- Output: giá trị của đối tượng đầu ngăn xếp
  
- Thuật toán:
  - Kiểm tra ngăn xếp rỗng hay không?
  - Nếu không:
    - Cập nhật địa chỉ của con trỏ đến đỉnh ngăn xếp
    - Xóa phần tử ở đỉnh khỏi ngăn xếp
    - Trả về giá trị của phần tử ở đỉnh

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

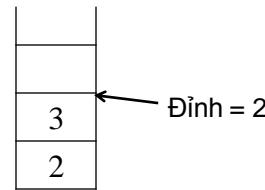
## Lấy phần tử ra khỏi ngăn xếp (pop)

46

- Ví dụ:



Ngăn xếp ban đầu



Ngăn xếp sau khi pop()

return 4;

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lấy thông tin đỉnh ngăn xếp

47

- Chỉ lấy giá trị của phần tử đầu mà *không hủy nó* khỏi ngăn xếp.
- Input:
- Output: giá trị tại đỉnh ngăn xếp
- Giải thuật:
  - Kiểm tra xem ngăn xếp có rỗng không?
  - Trả về giá trị của phần tử ở đỉnh ngăn xếp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

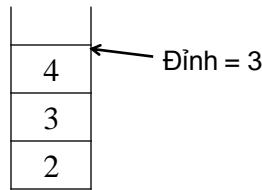
## Lấy thông tin đỉnh ngăn xếp

48

- Ví dụ



Ngăn xếp ban đầu



Ngăn xếp sau khi gettop()

return 4;

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập

49

- Cho biết nội dung của stack sau khi thực hiện các thao tác trong dãy (từ trái sang phải):

EAS\*Y\*\*QUE\*\*\*ST\*\*\*I\*ON

Mỗi chữ cái hoặc dấu \* tương ứng một thao tác trên stack trong đó:

- Một chữ cái tượng trưng cho thao tác thêm chữ cái đó vào stack
- Dấu \* tượng trưng cho thao tác lấy nội dung một phần tử trong stack ra rồi in lên màn hình.

Cho biết kết quả xuất ra màn hình sau khi hoàn tất chuỗi trên?

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập

50

- Cho biết nội dung của stack và giá trị của các biến sau khi thực hiện liên tiếp các thao tác sau:
- (Ban đầu các biến được khởi tạo: A= 5, B = 3, C= 7)
  - a. Tạo stack
  - b. push A
  - c. push C\*C
  - d. pop rồi lưu trữ vào biến B
  - e. push B+A
  - f. pop rồi lưu trữ vào biến A
  - g. pop rồi lưu trữ vào biến B

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ký pháp Ba Lan

51

- Biểu thức dạng *trung tố*: dấu của các phép toán hai ngôi luôn được *đặt giữa* 2 toán hạng
  - Ví dụ:  $A + B * C \quad A + B * C - D$   
 $(A+B) * C \quad (A + B )^* (C - D)$
  - ⇒ Qui định thứ tự ưu tiên của các phép toán
  - ⇒ Dùng dấu ngoặc để phân biệt thứ tự thực hiện.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ký pháp Ba Lan

52

- Biểu thức dạng *tiền tố*:

Trung tố	Tiền tố
$A + B$	$+ A B$
$(A+B)*C$	$*+ A B C$
$(A + B )^* (C - D)$	$* + A B - C D$

Không cần  
thiết phải  
dùng dấu  
 ngoặc

- Biểu thức dạng *hậu tố*:

Trung tố	Hậu tố
$A + B$	$A B +$
$(A+B)*C$	$A B + C *$
$(A + B )^* (C - D)$	$A B + C D - *$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Chuyển trung tố sang hậu tố

53

- Mã giả: P là biểu thức trung tố ban đầu, Q là biểu thức kết quả dạng hậu tố

```
0.1 push('(');
0.2 Thêm ')' vào P
while (chưa hết biểu thức P)
{
    1. đọc 1 kí tự x trong P (từ trái qua phải)
    2. if (x là toán hạng)
        Thêm x vào Q
    3. if (x là dấu ngoặc mở)
        push(x);
```

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Chuyển trung tố sang hậu tố

54

- Mã giả:

```
4. if (x là toán tử)
    4.1 while( thứ tự ưu tiên tại đỉnh >= x)
        4.1.1 w = pop();
        4.1.2 Thêm w vào Q
    4.2 push(x);
5. if (x là dấu ngoặc đóng)
    5.1 while( chưa gặp ngoặc mở)
        5.1.1 w = pop();
        5.1.2 Thêm w vào Q
    5.2 pop(); //dãy ngoặc mở ra khỏi ngăn xếp
}
```

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

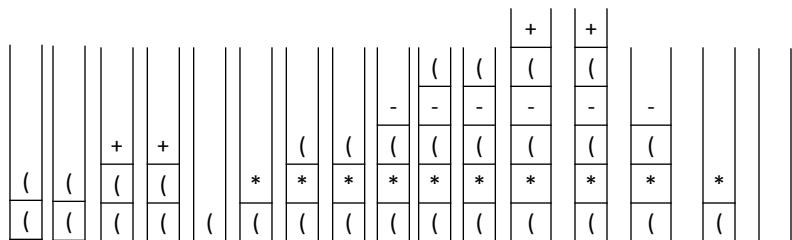
## Chuyển trung tố sang hậu tố

55

- ⦿ Ví dụ 1:  $P = (A + B)^* (C - (D + A))$

Kí tự đc đc ( A + B ) \* ( C - ( D + A ) )

## Trạng thái của ngăn xếp



$Q = A B + C D A + - *$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Chuyển trung tố sang hậu tố

56

- Ví dụ 2: đổi biểu thức trung tố

$$P = A + (B * C - (D / E \wedge F) * G) * H$$

sang biểu thức dạng hậu tố

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ứng dụng khác của ngăn xếp

57

- Dùng biến đổi cơ số
- Lượng giá biểu thức hậu tố
- Trong trình biên dịch, ngăn xếp được sử dụng để lưu mô trường các thủ tục.
- Dùng trong một số bài toán của lý thuyết đồ thị.
- Khử đệ qui đuôi.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

58

## Hàng đợi - Queue

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Nội dung

59

- Giới thiệu
- Các thao tác cơ bản
- Ứng dụng

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giới thiệu

60

### ◦ Giới thiệu:

- Là vật chứa các đối tượng làm việc theo qui tắc *vào trước ra trước* (FIFO).
- Các đối tượng có thể được thêm vào hàng đợi bất kì lúc nào nhưng chỉ có đối tượng *thêm vào đầu tiên* mới được *lấy ra khỏi hàng đợi*



- Việc thêm vào diễn ra ở cuối, việc lấy ra diễn ra ở đầu

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Các thao tác trên hàng đợi

61

### ◦ Thao tác cơ bản:

- Enqueue: Thêm 1 đối tượng vào cuối hàng đợi
- Dequeue: Lấy đối tượng ở đầu ra khỏi hàng đợi

### ◦ Thao tác khác:

- Lưu trữ hàng đợi
- Kiểm tra hàng đợi rỗng
- Kiểm tra hàng đợi đầy
- Lấy thông tin của đối tượng ở đầu hàng đợi

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

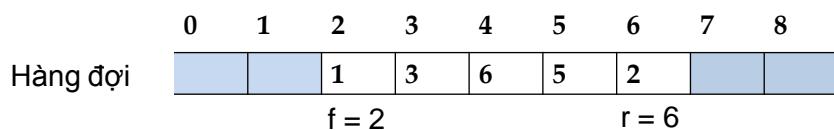
## Lưu trữ hàng đợi

62

### ◦ Lưu trữ bằng mảng:

- Khai báo mảng 1 chiều với kích thước tối đa N.

- f là địa chỉ của phần tử nằm ở đầu, r là địa chỉ của phần tử nằm ở cuối



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lưu trữ hàng đợi

63

### ◦ Lưu trữ bằng mảng:

- Các phần tử của hàng đợi sẽ di chuyển khắp các ô nhớ  
→ coi không gian dành cho hàng đợi theo dạng xoay vòng.
- Hàng đợi khi xoay vòng:

0	1	2	3	4	5	6	7	8
		1	3	6	5	2		

$r = 2$                                      $f = 6$

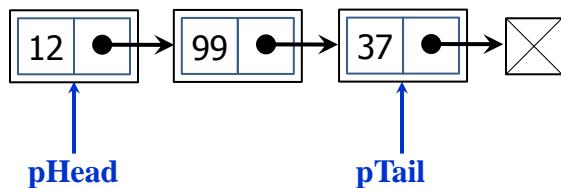
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lưu trữ hàng đợi

64

### ◦ Lưu trữ hàng đợi bằng danh sách liên kết đơn

- Phần tử đầu DSLK sẽ là phần tử đầu hàng đợi.
- Phần tử cuối DSLK sẽ là phần tử cuối hàng đợi



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kiểm tra hàng đợi rỗng

65

- Input:

- Output:

- TRUE nếu hàng đợi rỗng
- FALSE nếu hàng đợi không rỗng

- Hàng đợi rỗng:

- Mảng: ô nhớ đầu tiên không chứa dữ liệu
- DSLK: pHead = NULL

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kiểm tra hàng đợi đầy

66

- Input:

- Output:

- TRUE nếu hàng đợi đầy
- FALSE nếu hàng đợi không đầy

- Hàng đợi đầy:

- Mảng: ô nhớ cuối hàng đợi đã chứa dữ liệu
- DSLK: không cấp phát được vùng nhớ cho phần tử mới

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thêm phần tử vào cuối hàng đợi

67

- Input: giá trị cần thêm
- Output:
- Giải thuật thêm phần tử (EnQueue)
  - Kiểm tra hàng đợi đã đầy chưa?
  - Trong trường hợp lưu trữ bằng mảng: kiểm tra điều kiện xoay vòng.
  - Thêm phần tử vào cuối hàng đợi
  - Cập nhật địa chỉ phần tử cuối hàng đợi

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thêm phần tử vào cuối hàng đợi

68

- Ví dụ:

	0	1	2	3	4	5	6	7	8
Hàng đợi ban đầu			1	3	6	5	2		

$f = 2$        $r = 6$

EnQueue(9)			1	3	6	5	2	9	

$f = 2$        $r = 7$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lấy phần tử đầu ra khỏi hàng đợi

69

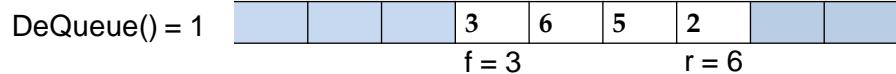
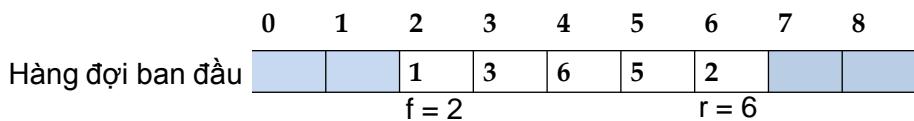
- Input:
- Output: giá trị của phần tử đầu hàng đợi
- Giải thuật lấy phần tử ở đầu (DeQueue)
  - Kiểm tra hàng đợi có rỗng không?
  - Xóa phần tử đầu ra khỏi hàng đợi
  - Cập nhật địa chỉ phần tử đầu hàng đợi
  - Trong trường hợp lưu trữ bằng mảng: kiểm tra điều kiện xoay vòng

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lấy phần tử đầu ra khỏi hàng đợi

70

- Ví dụ:



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lấy thông tin đầu hàng đợi

71

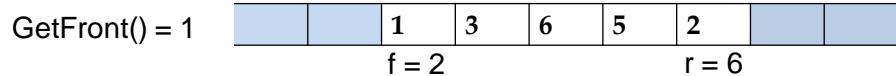
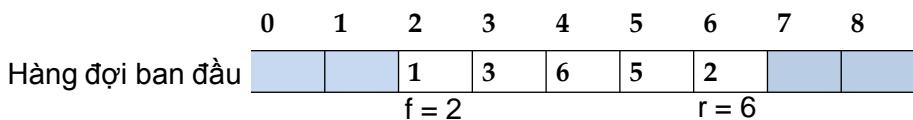
- Chỉ lấy thông tin của đối tượng đầu hàng đợi mà *không hủy* đối tượng khỏi hàng đợi.
- Input: hàng đợi
- Output: giá trị của đối tượng đầu hàng đợi
- Giải thuật:
  - Kiểm tra hàng đợi rỗng?
  - Trả về giá trị của phần tử đầu hàng đợi

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Lấy thông tin đầu hàng đợi

72

- Ví dụ:



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ứng dụng của hàng đợi

73

- Bộ đệm bàn phím của máy tính
- Xử lý các lệnh trong máy tính: hàng đợi thông điệp trong Windows, hàng đợi tiến trình ...
- Thường dùng trong các hệ mô phỏng.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập

74

Cho hàng đợi ban đầu như sau: (hàng đợi có tối đa 6 phần tử)

0	1	2	3	4	5
	A	B	C		

$f = 1$                        $r = 3$

Vẽ tình trạng của hàng đợi, cho biết giá trị  $f$ ,  $r$  tương ứng với mỗi lần thực hiện thao tác sau:

- a. Bổ sung E vào hàng đợi
- b. Loại 2 phần tử khỏi hàng đợi
- c. Bổ sung I, J, K vào hàng đợi
- d. Loại 2 phần tử khỏi hàng đợi
- e. Bổ sung O vào hàng đợi
- f. Loại 2 phần tử khỏi hàng đợi

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Hỏi và Đáp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Cấu trúc dữ liệu và giải thuật

## CẤU TRÚC CÂY

Giảng viên:  
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

### Nội dung trình bày

2

Khái niệm

Phép duyệt cây và Biểu diễn cây

Cây nhị phân và Cây nhị phân tìm kiếm

Cây AVL

Cây AA

3

# Khái niệm

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

4

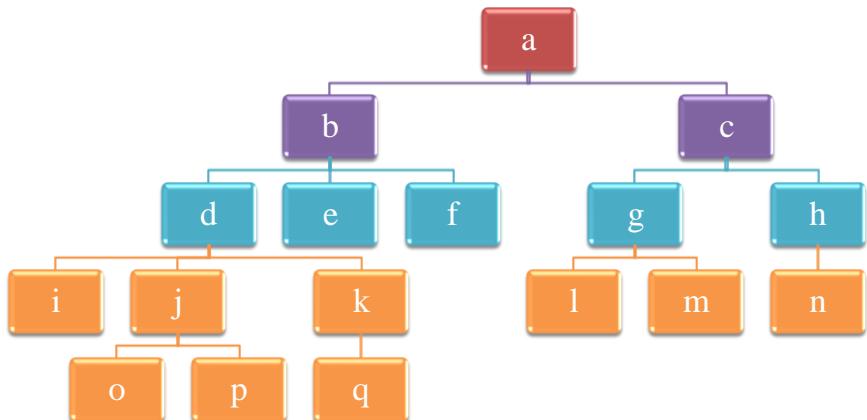
## Một số thuật ngữ

- Tree
- Search tree
- Binary search tree
- Balanced tree
- AVL tree
- AA tree
- Red-Black tree
- ...

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cây tổng quát

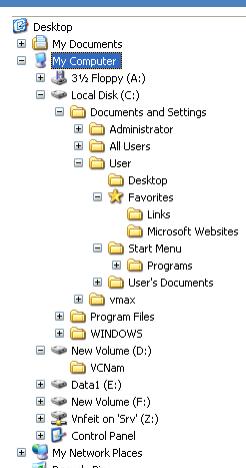
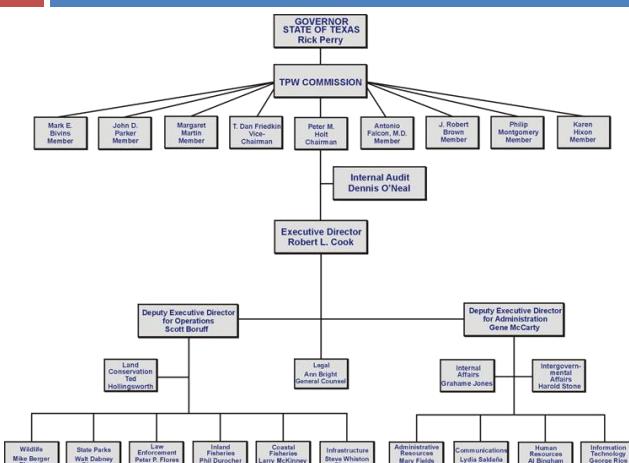
5



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cây tổng quát

6



# Cây thư mục

Sơ đồ tổ chức  
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Định nghĩa

7

Cây (cây có gốc) được xác định đệ quy như sau:

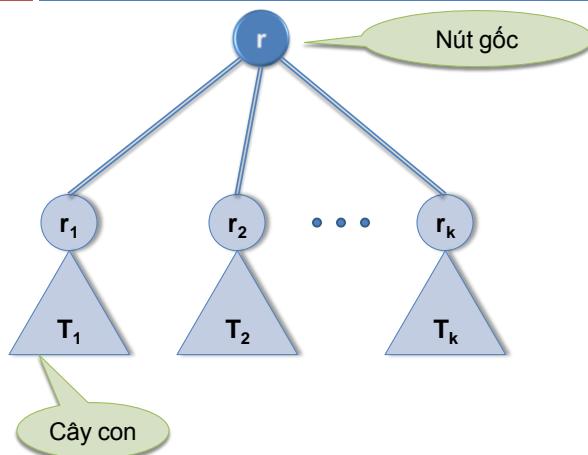
1. Tập hợp gồm 1 **đỉnh** là một cây. Cây này có **gốc** là đỉnh duy nhất của nó.
2. Gọi  $T_1, T_2, \dots, T_k$  ( $k \geq 1$ ) là các cây không cắt nhau có gốc tương ứng  $r_1, r_2, \dots, r_k$ .

Giả sử  $r$  là một đỉnh mới không thuộc các cây  $T_i$ . Khi đó, tập hợp  $T$  gồm đỉnh  $r$  và các cây  $T_i$  tạo thành một cây mới với gốc  $r$ . Các cây  $T_1, T_2, \dots, T_k$  được gọi là cây con của gốc  $r$ .

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Định nghĩa

8



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các khái niệm

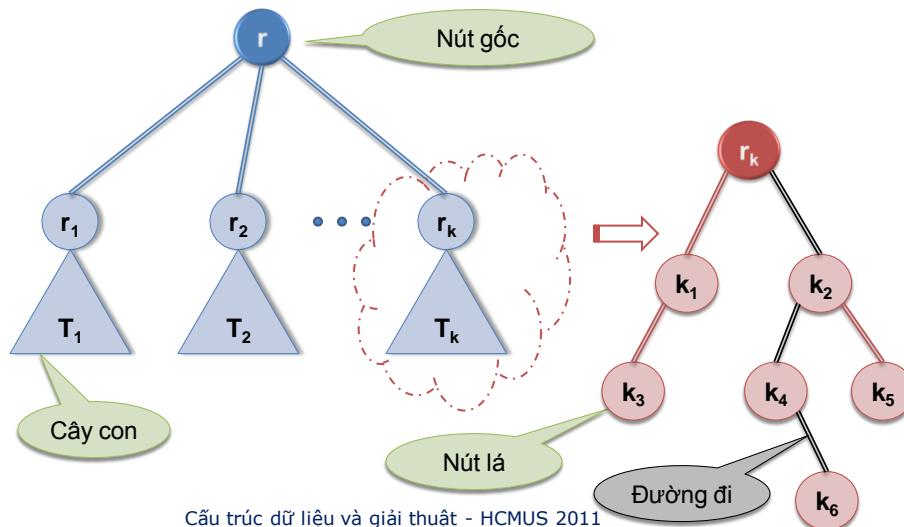
9

- node: đỉnh
- root: gốc cây
- leaf: lá
- inner node/internal node: đỉnh trong
- parent: đỉnh cha
- child: đỉnh con
- path: đường đi

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các khái niệm

10



## Các khái niệm

11

- degree/order: bậc

- Bậc của node: Số con của node

- Bậc của cây: bậc lớn nhất trong số các con

- depth/level: độ sâu/mức

- Mức (độ sâu) của node: Chiều dài của đường đi từ node gốc đến node đó cộng thêm 1.

- height: chiều cao

- Chiều cao cây:

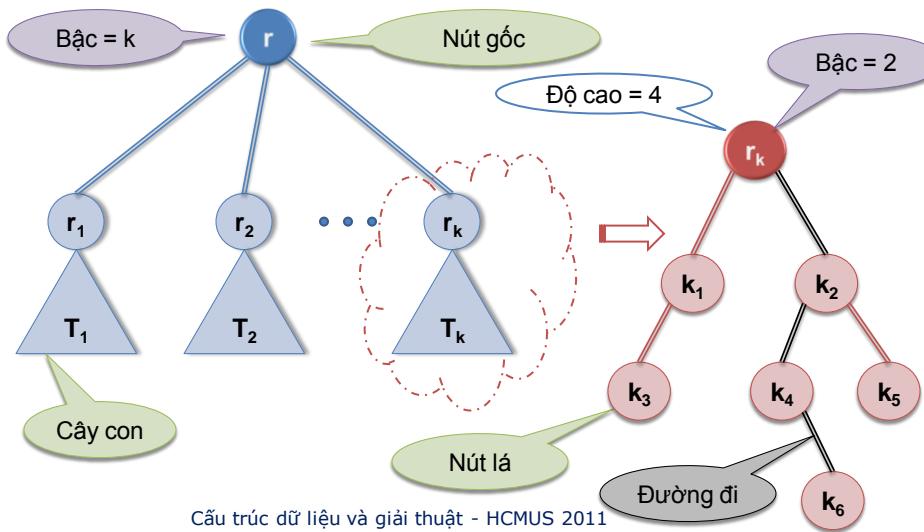
- Cây rỗng: 0

- Cây khác rỗng: Mức lớn nhất giữa các node của cây

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các khái niệm

12



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

13

## Phép duyệt cây

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

14

## Phép duyệt cây

- Đảm bảo đến mỗi node trên cây **chính xác một lần** một cách **có hệ thống**.
- Nhiều thao tác xử lý trên cây cần phải sử dụng đến phép duyệt cây.
- Các phép cơ bản:
  - Duyệt trước (Pre-order)
  - Duyệt giữa (In-order)
  - Duyệt sau (Post-order)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép duyệt cây

15

Tìm cha một đỉnh.

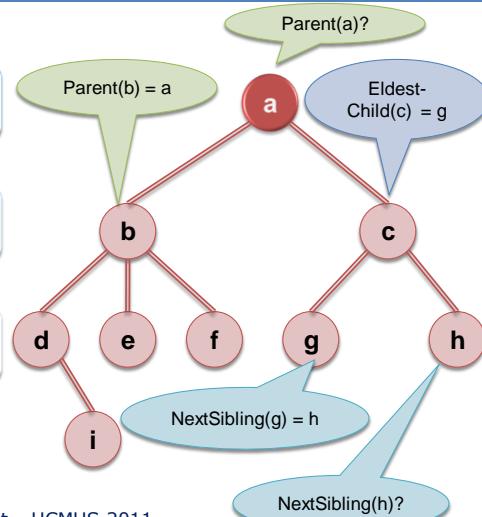
- $\text{Parent}(x)$

Tìm đỉnh con trái nhất.

- $\text{EldestChild}(x)$

Tìm đỉnh kê phải.

- $\text{NextSibling}(x)$



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép duyệt cây

16

Duyệt trước

- $a b d e i j c f g k h$

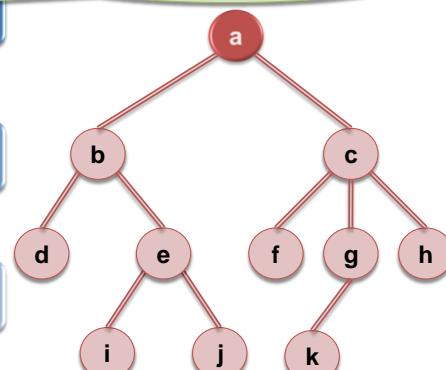
Duyệt theo chiều sâu

Duyệt giữa

- $d b i e j a f c k g h$

Duyệt sau

- $d i j e b f k g h c a$



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Phép duyệt cây

17

## Pre-order

```
void Preorder(NODE A)
{
    NODE B;
    Visit(A);
    B = EldestChild(A);
    while (B != Ø) {
        Preorder(B);
        B = NextSibling(B);
    }
}
```

## Post-order

```
void Postorder(NODE A)
{
    NODE B;
    B = EldestChild(A);
    while (B != Ø) {
        Postorder(B);
        B = NextSibling(B);
    }
    Visit(A);
}
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Phép duyệt cây

18

## In-Order

```
void Inorder(NODE A)
{
    NODE B;
    B = EldestChild(A);
    if (B != Ø) {
        Inorder(B);
        B = NextSibling(B);
    }
    Visit(A);
    while (B != Ø) {
        Inorder(B);
        B = NextSibling(B);
    }
}
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

19

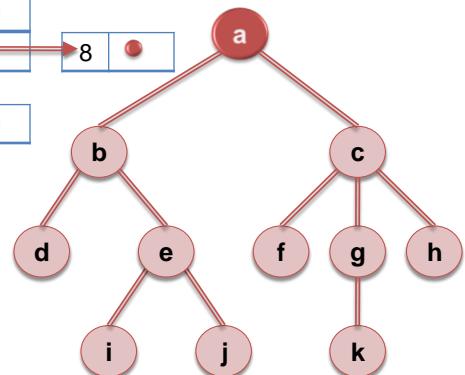
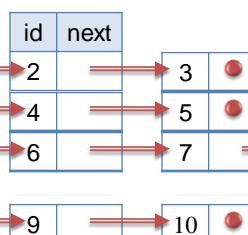
## Biểu diễn cây

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

20

## Bảng danh sách cây con

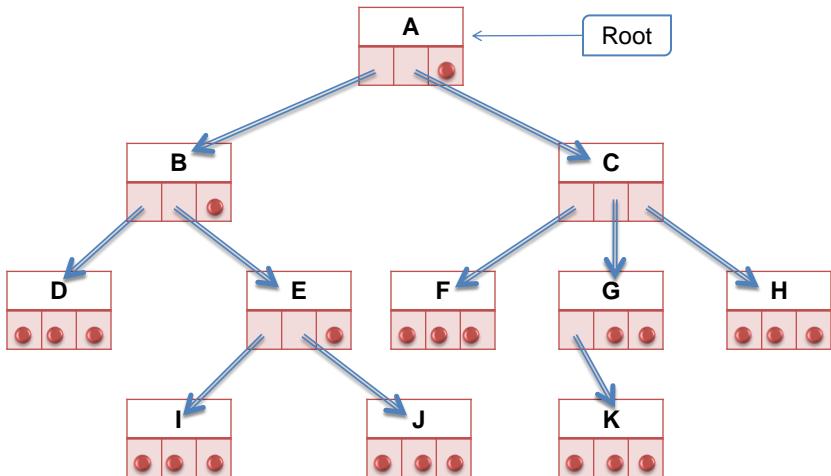
	info	child
1	a	2
2	b	4
3	c	6
4	d	•
5	e	9
6	f	•
7	g	11
8	h	•
9	i	•
10	j	•
11	k	•



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bảng danh sách cây con

21

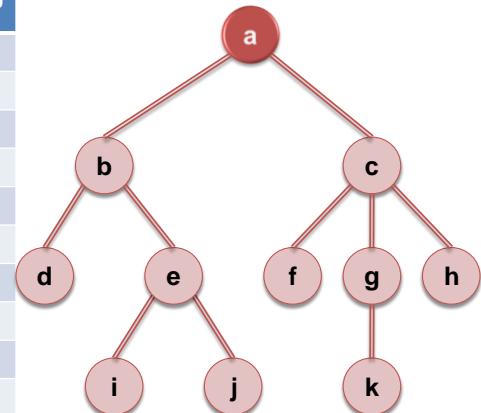


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bảng định trái nhất và định kè phải

22

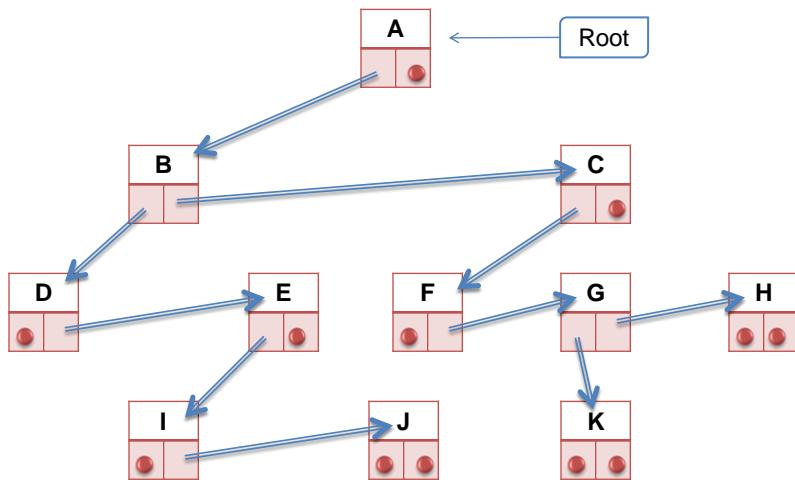
	Info	Eldest Child	Next Sibling
1	a	2	0
2	b	4	3
3	c	6	0
4	d	0	5
5	e	9	0
6	f	0	7
7	g	11	8
8	h	0	0
9	i	0	10
10	j	0	0
11	k	0	0



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bảng đỉnh trái nhất và đỉnh kề phải

23

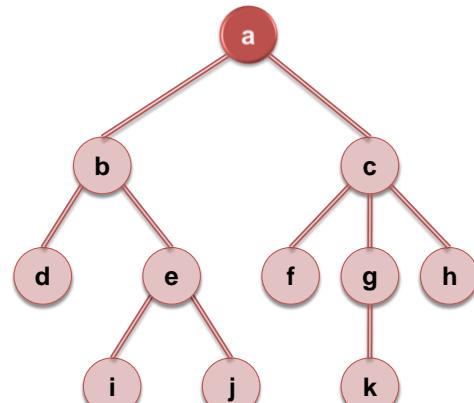


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bảng cha mỗi đỉnh

24

	Info	Parent
1	a	0
2	b	1
3	c	1
4	d	2
5	e	2
6	f	3
7	g	3
8	h	3
9	i	5
10	j	5
11	k	7



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

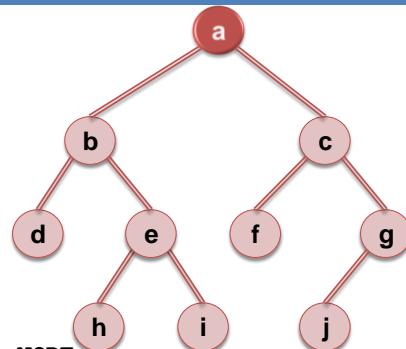
# Cây nhị phân

Binary tree

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cây nhị phân

- Là cây mà mỗi đỉnh có bậc tối đa bằng 2.
- Các cây con được gọi là cây con trái và cây con phải.
- Có toàn bộ các thao tác cơ bản của cây.



```

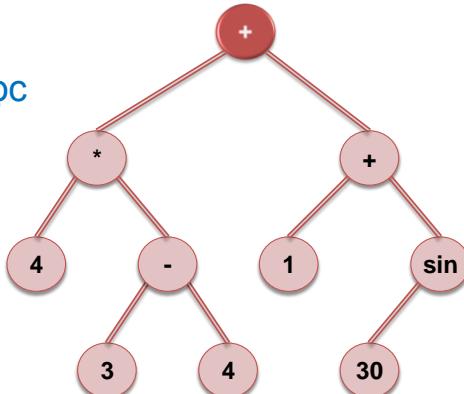
struct NODE
{
    Data key;
    NODE *pLeft;
    NODE *pRight;
};
  
```

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Một số ứng dụng

27

- Cây tổ chức thi đấu
- Cây biểu thức số học
- Lưu trữ và tìm kiếm thông tin.



Cây biểu thức:  
 $4 * (3 - 4) + (1 + \sin(30))$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cây nhị phân tìm kiếm

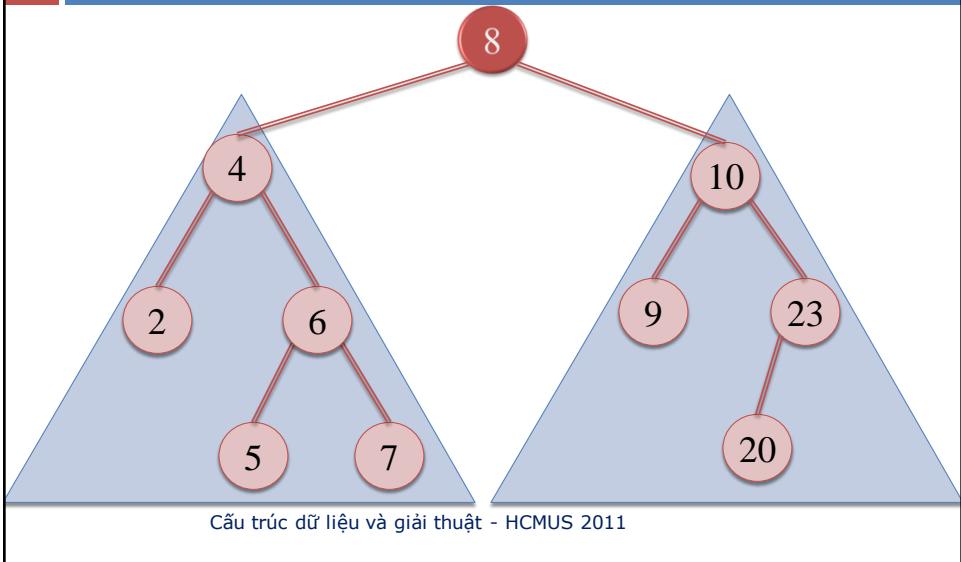
28

- Cây nhị phân tìm kiếm là cây nhị phân thỏa mãn các điều kiện sau:
  1. Khóa của các đỉnh thuộc cây con trái nhỏ hơn khóa gốc.
  2. Khóa của gốc nhỏ hơn khóa các đỉnh thuộc cây con phải.
  3. Cây con trái và cây con phải của gốc cũng là cây nhị phân tìm kiếm.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cây nhị phân tìm kiếm

29



## Cây nhị phân tìm kiếm

30

### ◦ Đặc điểm:

- Có thứ tự
- Không có phần tử trùng
- Dễ dàng tạo dữ liệu sắp xếp, và tìm kiếm

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Thao tác trên cây nhị phân tìm kiếm

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các thao tác

32

- Thêm phần tử (khóa)
- Tìm kiếm phần tử (khóa)
- Xóa phần tử (khóa)
- Sắp xếp
- Duyệt cây
- Quay cây

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thêm phần tử

33

- **Bước 1:** Bắt đầu từ gốc
- **Bước 2:** So sánh dữ liệu (khóa) cần thêm với dữ liệu (khóa) của node hiện hành.
  - Nếu bằng nhau => Đã tồn tại. Kết thúc
  - Nếu nhỏ hơn => Đi qua nhánh trái, Tiếp bước 2.
  - Nếu lớn hơn => Đi qua nhánh phải, Tiếp bước 2.
- **Bước 3:** Không thể đi tiếp nữa => Tạo node mới với dữ liệu (khóa) cần thêm. Kết thúc

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Tìm kiếm phần tử

34

- **Bước 1:** Bắt đầu từ gốc
- **Bước 2:** So sánh dữ liệu (khóa) cần tìm với dữ liệu (khóa) của node hiện hành.
  - Nếu bằng nhau => Tìm thấy. Kết thúc
  - Nếu nhỏ hơn => Đi qua nhánh trái, Tiếp bước 2.
  - Nếu lớn hơn => Đi qua nhánh phải, Tiếp bước 2.
- **Bước 3:** Không thể đi tiếp nữa => Không tìm thấy. Kết thúc.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xóa phần tử

35

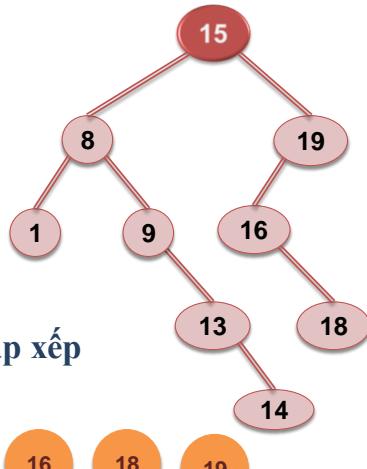
- Tìm đến node chứa dữ liệu (khóa) cần xóa.
- Xét các trường hợp:
  - Node lá
  - Node chỉ có 1 con
  - Node có 2 con: dùng phần tử thê mạng để xóa thê.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Sắp xếp

36

- Cho cây nhị phân tìm kiếm
- Thứ tự duyệt các node nếu sử dụng Duyệt giữa?
- Nếu nhận xét
  - Có thể dễ dàng tạo dữ liệu sắp xếp nếu dùng phép duyệt giữa

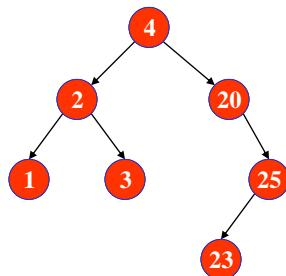


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Phép duyệt cây

37

## ◦ Duyệt trước

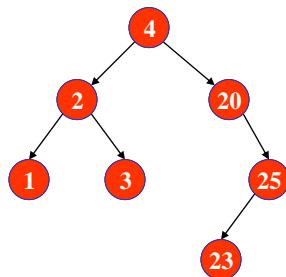


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Phép duyệt cây

38

## ◦ Duyệt giữa

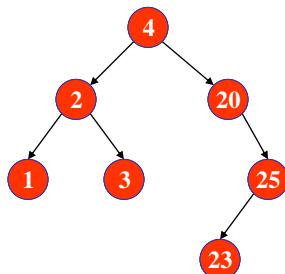


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép duyệt cây

39

### ◦ Duyệt sau

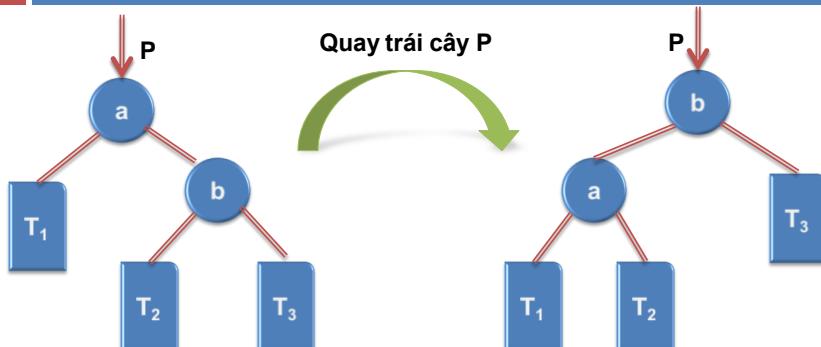


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép quay trái

40

Quay trái cây P

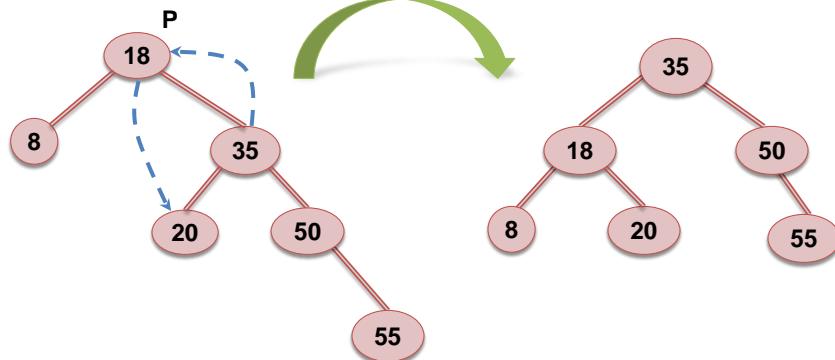


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép quay trái

41

Quay trái cây P



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép quay phải

42

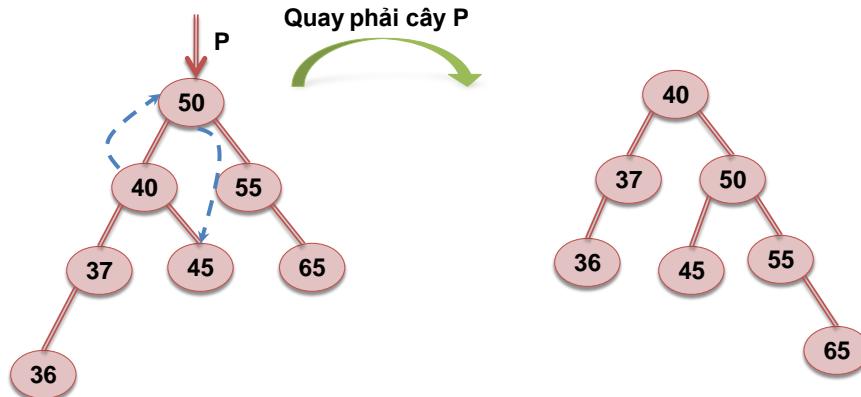
Quay phải cây P



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Phép quay phải

43



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thời gian thực hiện các phép toán

44

### ◦ Đối với phép tìm kiếm:

- Trường hợp tốt nhất: mỗi nút (trừ nút lá) đều có 2 con: **O(log<sub>2</sub>n)** (chính là chiều cao của cây).
- Trường hợp xấu nhất: cây trở thành danh sách liên kết: **O(n)**.
- Trường hợp trung bình là bao nhiêu?  
**O(log<sub>2</sub>n)**

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

45

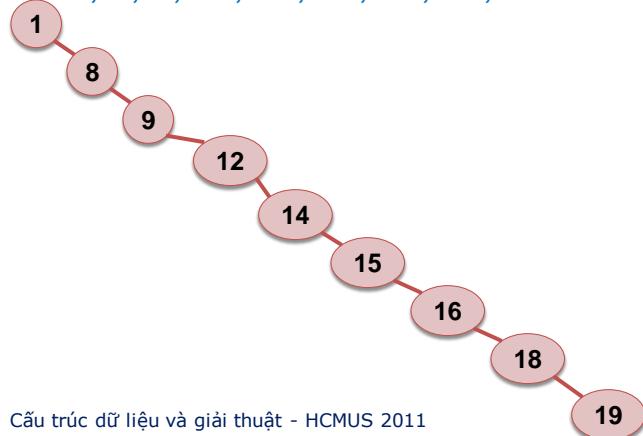
- Tạo cây nhị phân tìm kiếm theo thứ tự nhập như sau: 1, 8, 9, 12, 14, 15, 16, 18, 19

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

46

- Tạo cây nhị phân tìm kiếm theo thứ tự nhập như sau: 1, 8, 9, 12, 14, 15, 16, 18, 19



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

47

## Cây AVL

AVL tree

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

48

## Giới thiệu

- Do G.M. Adelsen Velskii và E.M. Lendis đưa ra vào năm 1962, đặt tên là cây AVL.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Định nghĩa

49

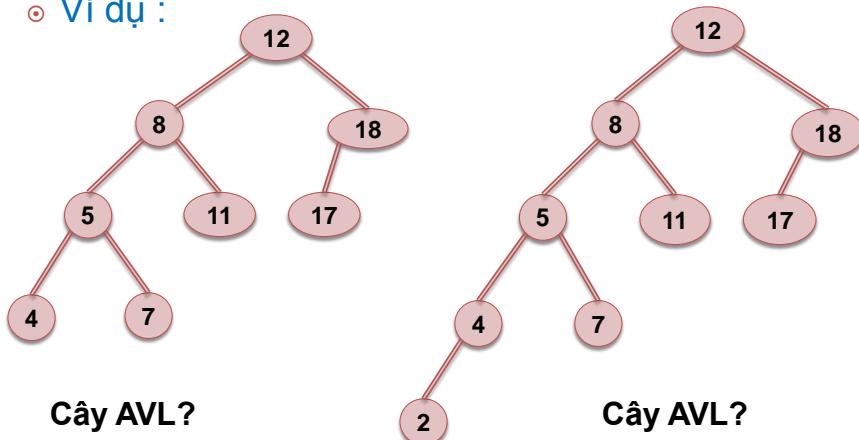
- Cây cân bằng AVL là cây nhị phân tìm kiếm mà tại mỗi đỉnh của cây, độ cao của cây con trái và cây con phải **không chênh lệch quá 1**.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Cây AVL

50

- Ví dụ :



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xây dựng cây cân bằng

51

- Việc xây dựng cây cân bằng dựa trên cây nhị phân tìm kiếm, chỉ bổ sung thêm 1 giá trị cho biết sự cân bằng của các cây con như thế nào.

- Cách làm gợi ý:

```
struct NODE {  
    Data key;  
    NODE *pLeft, *pRight;  
    int bal;  
};
```

- Trong đó giá trị bal (balance, cân bằng) có thể là: 0: cân bằng; 1: lệch trái; 2: lệch phải

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các trường hợp mất cân bằng

52

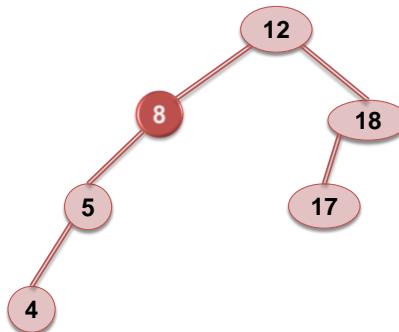
- Mất cân bằng trái-trái (L-L)
- Mất cân bằn trái-phải (L-R)
- Mất cân bằng phải-phải (R-R)
- Mất cân bằng phải-trái (R-L)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các trường hợp mất cân bằng

53

- Mất cân bằng trái-trái (L-L)

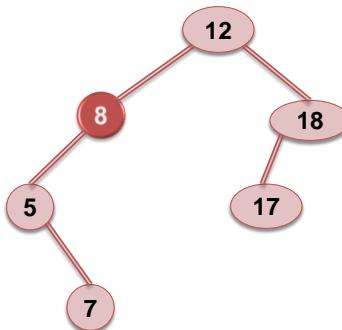


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các trường hợp mất cân bằng

54

- Mất cân bằng trái-phải (L-R)

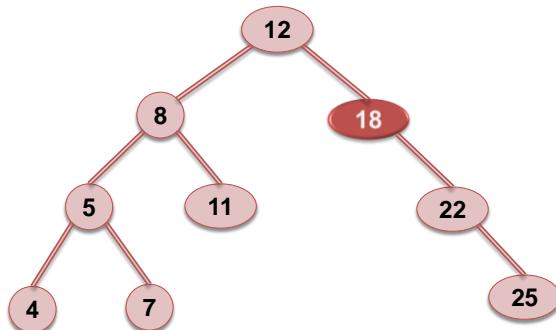


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các trường hợp mất cân bằng

55

- Mất cân bằng phải-phải (R-R)

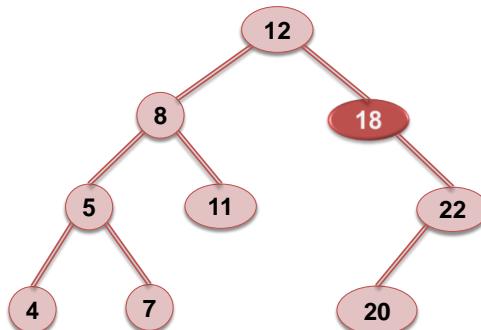


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các trường hợp mất cân bằng

56

- Mất cân bằng phải-trái (R-L)



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

57

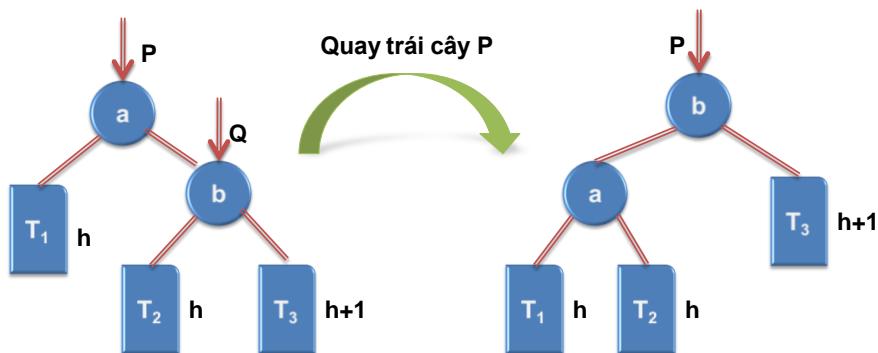
- Giả sử tại một node cây xảy ra mất cân bằng bên phải (cây con phải chênh lệch với cây con trái hơn một đơn vị):
  - Mất cân bằng phải-phải (RR)
    - Quay trái
  - Mất cân bằng phải-trái (R-L)
    - Quay phải
    - Quay trái

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

58

- P mất cân bằng phải-phải (RR):

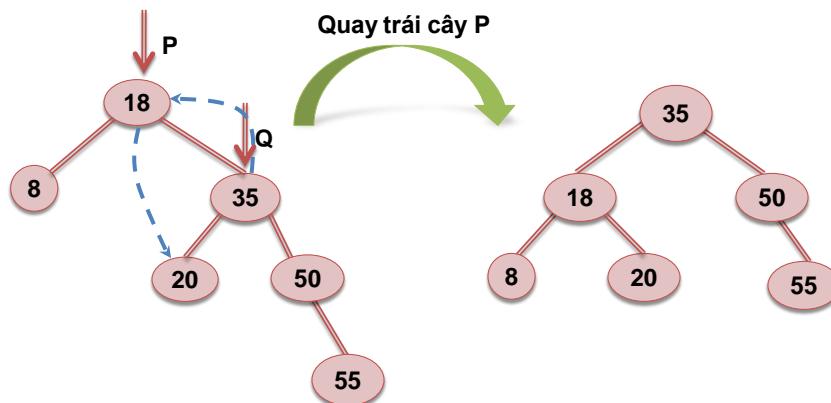


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

59

### ◦ P mất cân bằng phải-phải (RR):



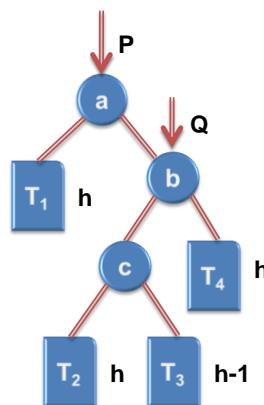
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

60

### ◦ P mất cân bằng phải-trái (RL):

- Bước 1: quay phải Q
- Bước 2: quay trái cây P



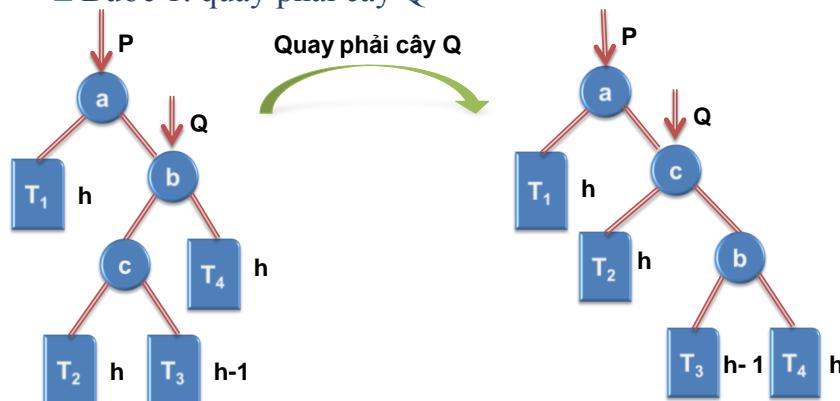
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

61

### ◦ P mất cân bằng phải-trái (RL):

- Bước 1: quay phải cây Q



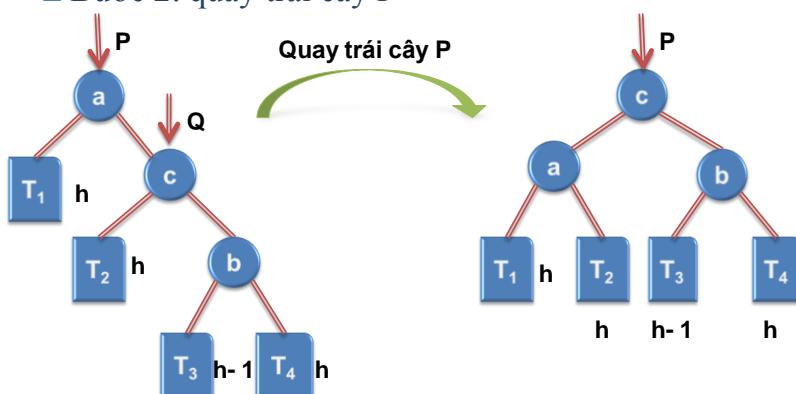
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

62

### ◦ P mất cân bằng phải-trái (RL):

- Bước 2: quay trái cây P

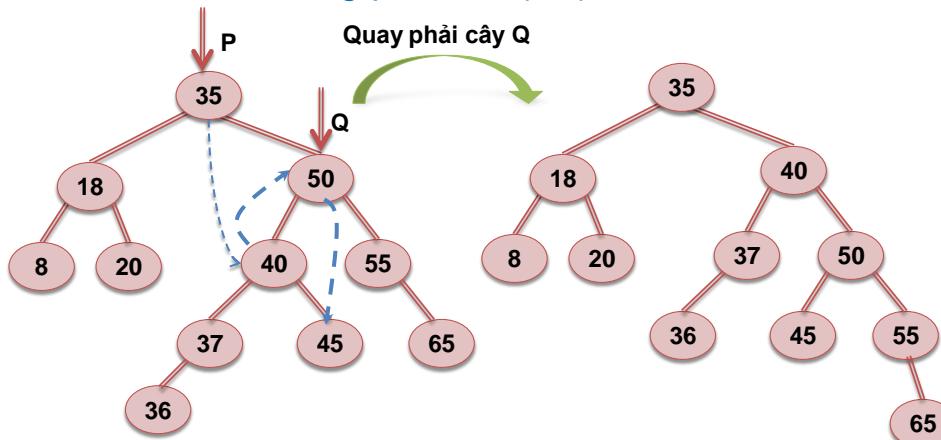


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

63

- P mất cân bằng phải-trái (RL) – Bước 1:

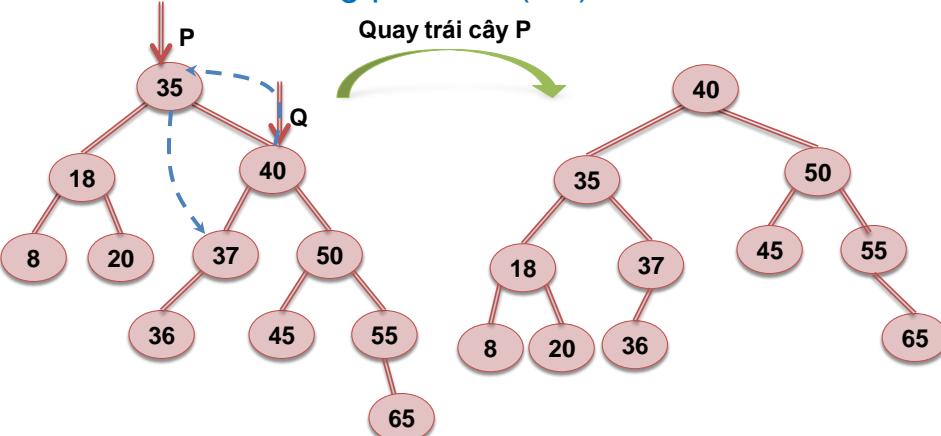


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xử lý mất cân bằng

64

- P mất cân bằng phải-trái (RL) - Bước 2:



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Xử lý mất cân bằng

65

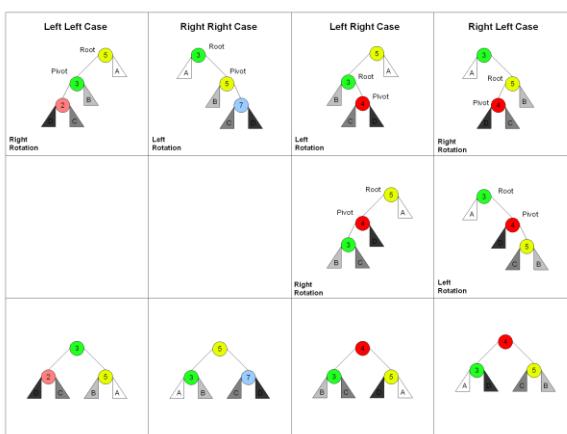
- Khi một node cây xảy ra mất cân bằng bên trái (cây con trái chênh lệch với cây con phải hơn một đơn vị): (thực hiện đối xứng với trường hợp mất cân bằng bên phải)
  - Mất cân bằng trái-trái (LL)
    - Quay phải
  - Mất cân bằng trái-trái (L-R)
    - Quay trái
    - Quay phải

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Xử lý mất cân bằng

66

There are 4 cases in all, choosing which one is made by seeing the position of the first 2 nodes from the balanced node to the newly inserted node and matching them to the top most row.  
Root is the initial parent before a rotation. Pivot is the child to take the root's place.



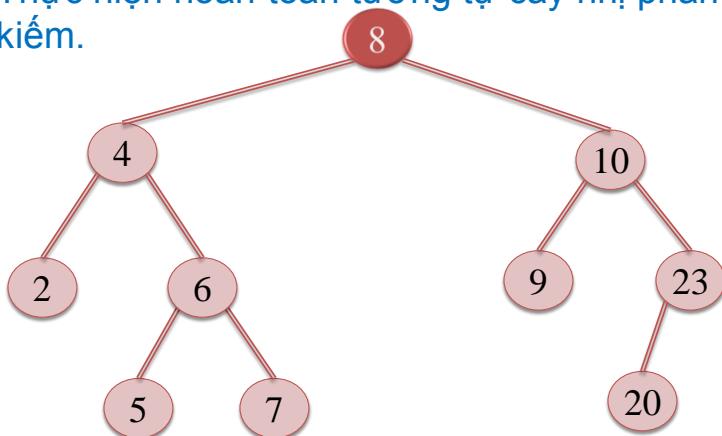
Theo Wikipedia

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thao tác tìm kiếm

67

- Thực hiện hoàn toàn tương tự cây nhị phân tìm kiếm.



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thao tác thêm phần tử

68

- Thực hiện tương tự với việc thêm phần tử của cây nhị phân tìm kiếm.
- Nếu xảy ra việc mất cân bằng thì xử lý bằng các trường hợp mất cân bằng đã biết.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thao tác xóa phần tử

69

- Thực hiện tương tự cây nhị phân tìm kiếm: xét 3 trường hợp, và tìm phần tử thế mạng nếu cần.
- Sau khi xóa, nếu cây mất cân bằng, thực hiện cân bằng cây.
- Lưu ý: việc cân bằng sau khi hủy có thể xảy ra dây chuyền.

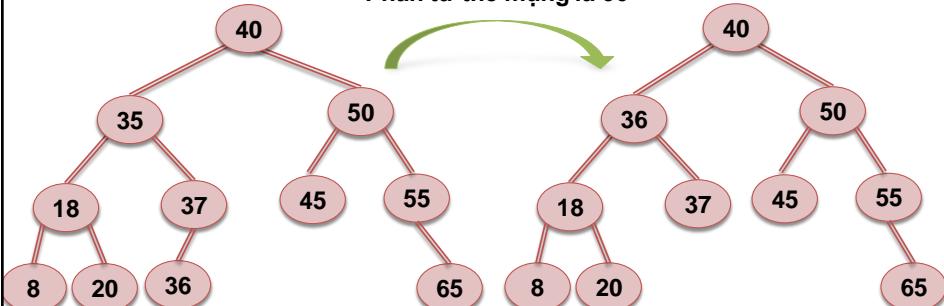
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thao tác xóa phần tử

70

- Ví dụ: xóa 35

Phần tử thế mạng là 36



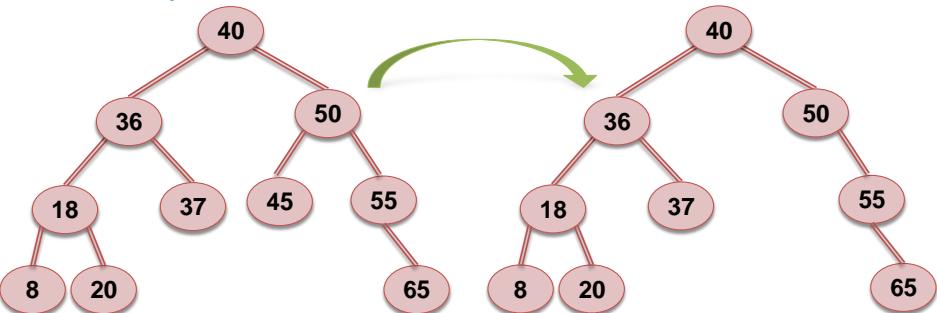
Cây vẫn cân bằng nên  
không phải hiệu chỉnh

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thao tác xóa phần tử

71

### ◦ Xóa phần tử 45



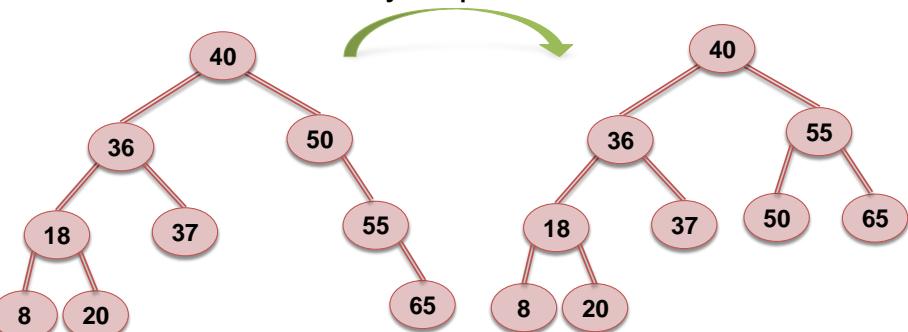
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thao tác xóa phần tử

72

### ◦ Xóa phần tử 45: cân bằng lại cây

Quay trái tại node 50



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

73

## Cây AA

AA tree

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

74

## Hình thành

- Được đặt tên theo tác giả Arne Anderson (Thụy Điển).
- Công trình được công bố năm 1993 (Balanced Search Trees Made Simple).

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các khái niệm

75

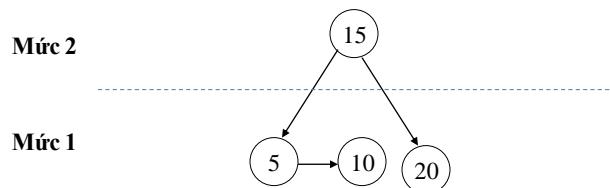
- Mức của node
- Liên kết ngang

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các khái niệm

76

- Mức của node:
  - Số liên kết **trái** từ node đó đến node NULL.
    - Mức của NULL là 0.
    - Mức của node lá là 1.



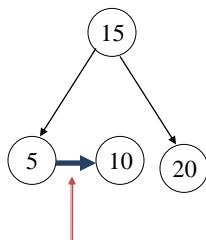
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các khái niệm

77

- **Liên kết ngang:**

- Liên kết giữa node cha và node con có cùng mức.



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Tính chất

78

- Cây AA là cây nhị phân tìm kiếm thỏa mãn các tính chất sau:

- [1] Mức của node con trái bắt buộc phải nhỏ hơn mức của node cha.

- [2] Mức của node con bên phải nhỏ hơn hoặc bằng mức của node cha.

Liên kết ngang bắt buộc hướng sang phải.

- [3] Mức của node cháu bên phải bắt buộc nhỏ hơn mức của node ông.

Không tồn tại 2 liên kết ngang liên tiếp.

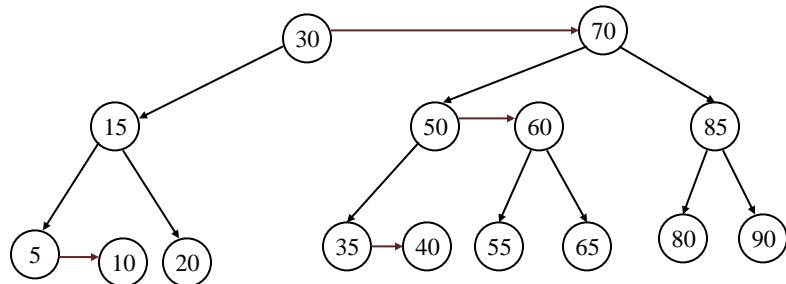
- [4] Mọi node có mức lớn hơn 1 phải có 2 node con.

- [5] Nếu một node không có liên kết ngang phải thì cả hai node con của nó phải cùng mức.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

79

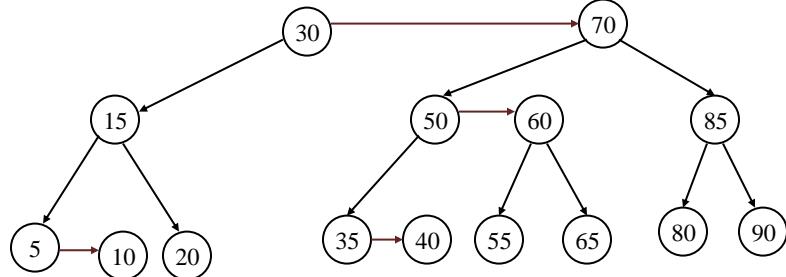


Mức của các node?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

80

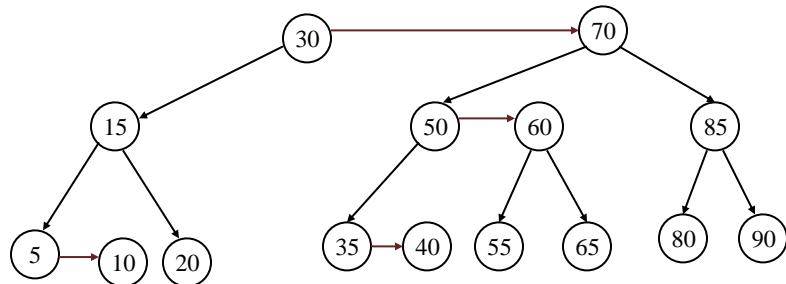


So sánh mức của node con trái với mức của node cha trực tiếp của nó?  
Các cặp node: 15 và 30, 5 và 15, 50 và 70, 35 và 50, 55 và 60, 80 và 85

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

81

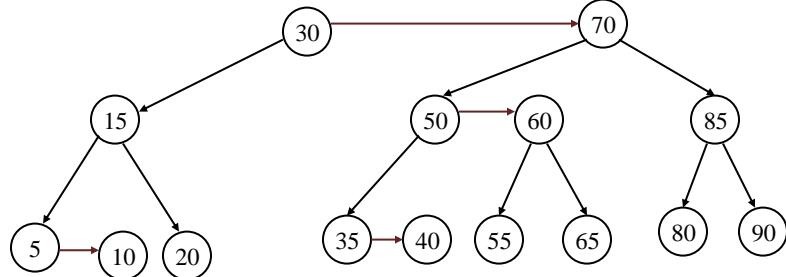


Các liên kết ngang?  
Hướng của liên kết ngang?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

82

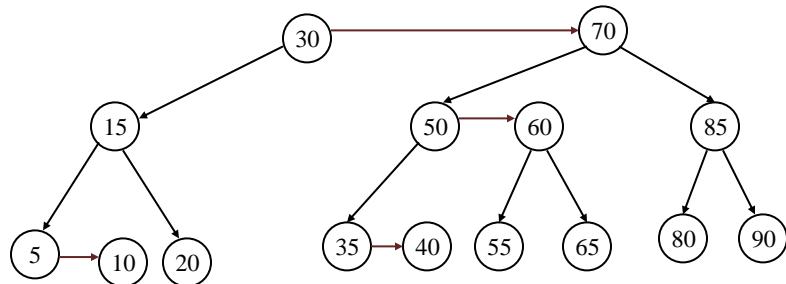


Có tồn tại 2 liên kết ngang liên tiếp?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

83

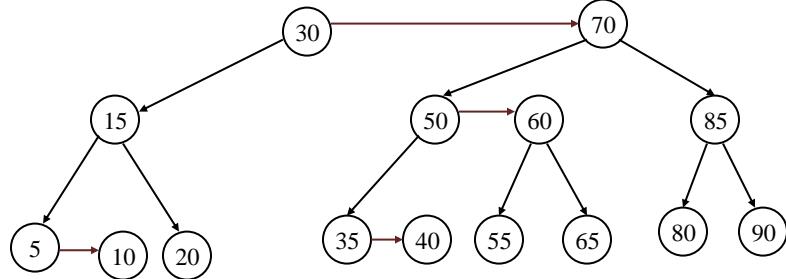


Mọi node có mức lớn hơn 1 đều có 2 node con?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

84



So sánh mức của các node con của các node: 15, 70, 60, 85?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các phép biến đổi cây

85

- Skew

- Split

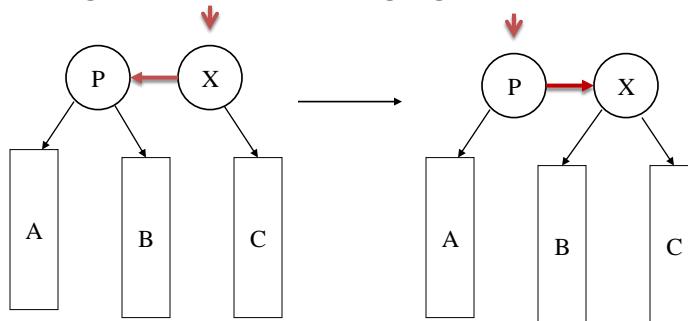
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các phép biến đổi cây

86

- Skew:

- Dùng để loại bỏ liên kết ngang trái.



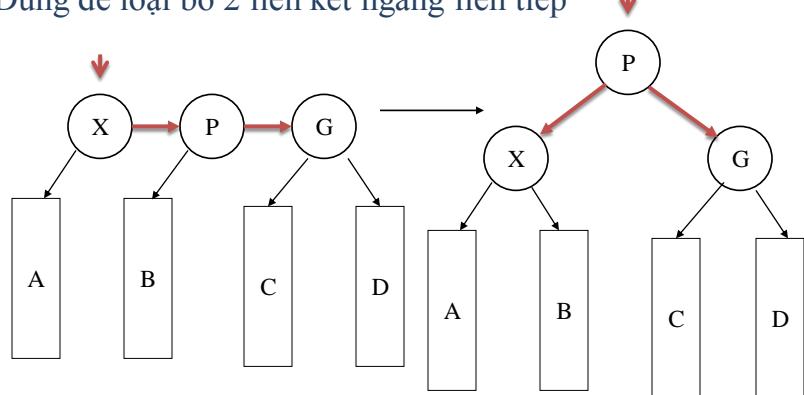
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các phép biến đổi cây

87

- Split:

- Dùng để loại bỏ 2 liên kết ngang liên tiếp



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các phép biến đổi cây

88

- Skew: dùng để loại bỏ liên kết ngang bên trái.

- Split: dùng để loại bỏ 2 liên kết ngang (phải) liên tiếp.

- Biến đổi theo thứ tự Skew -> Split (nếu có).

- Khi thực hiện thao tác Split, node giữa được tăng thêm một mức.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Các thao tác trên cây

89

- Duyệt cây, Tìm kiếm:
  - Tương tự cây nhị phân tìm kiếm
- Thêm phần tử
- Xóa phần tử

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thêm phần tử

90

- Thực hiện tương tự trên cây nhị phân tìm kiếm.
- Phần tử được thêm vào luôn ở mức 1.
- Sau khi thêm, thực hiện các thao tác Skew và/hoặc Split để đảm bảo tính chất của cây.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

91

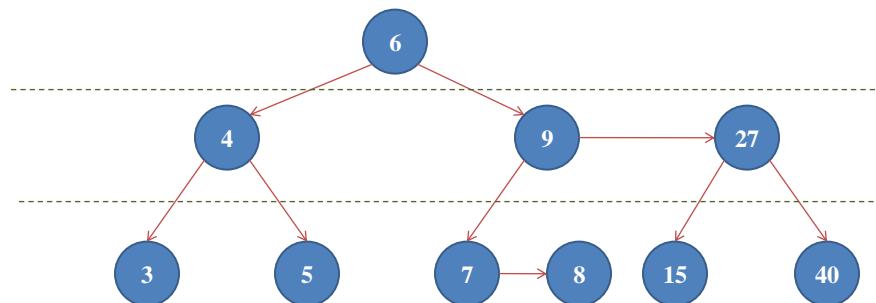
- Vẽ cây AA theo thứ tự nhập sau đây:

4, 7, 6, 3, 5, 9, 15, 27, 8, 40

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

92



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

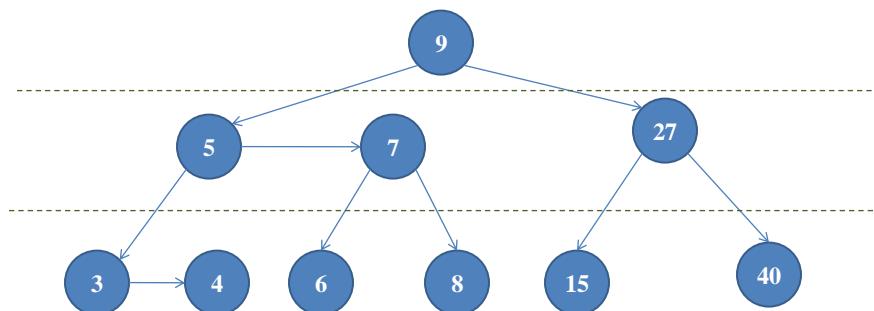
93

- Hãy vẽ cây AA theo thứ tự nhập sau đây:
  - 40, 8, 27, 15, 9, 5, 3, 6, 7, 4

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

94



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Xóa phần tử

95

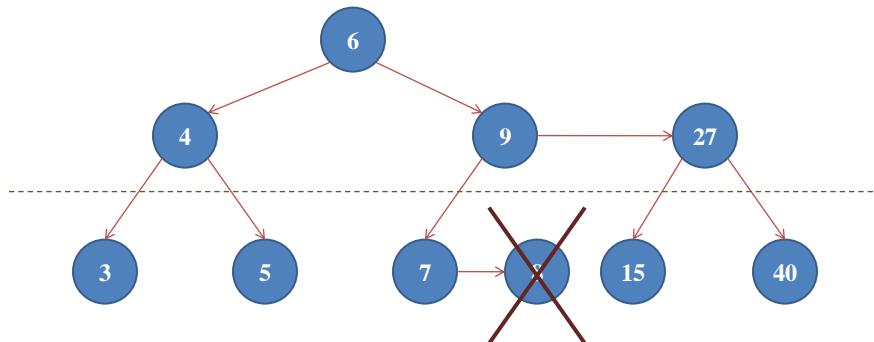
- Nếu không phải là node lá (mức của node là 1), tìm phần tử thế mạng:
  - Phần tử lớn nhất bên nhánh trái (node lá).
- Xóa node lá:
  - Giảm mức của node cha nếu mức của node lá nhỏ hơn.
  - Thực hiện các thao tác Skew, Split cần thiết

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

96

- Xóa phần tử 8

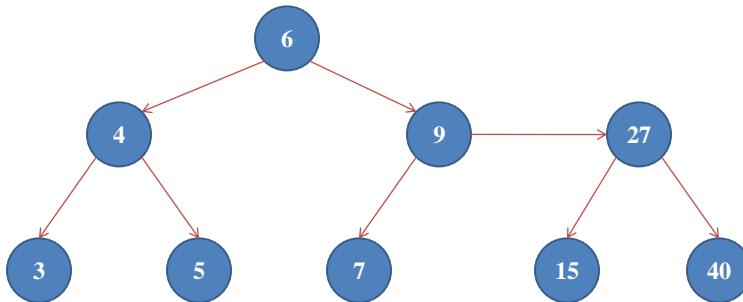


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

97

### ◦ Xóa phần tử 8

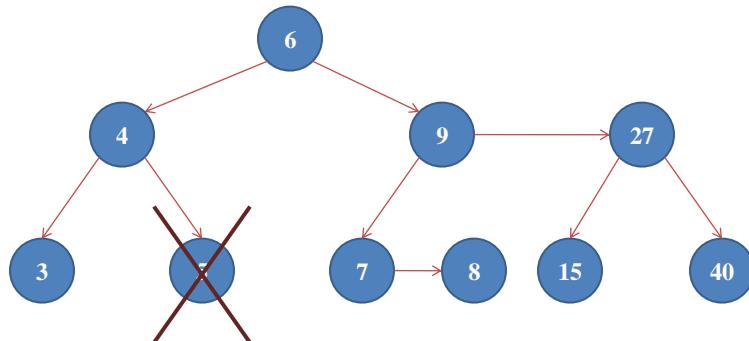


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

98

### ◦ Xóa phần tử 5

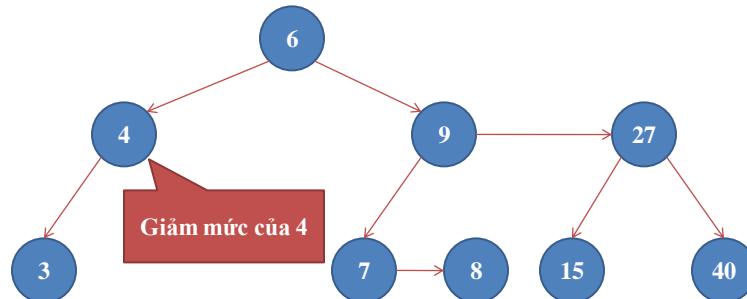


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

99

### ◦ Xóa phần tử 5

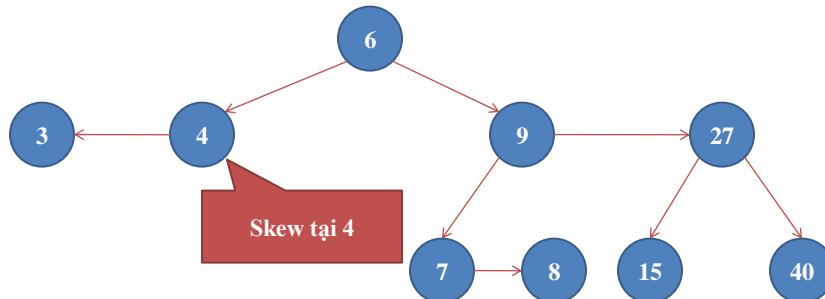


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

100

### ◦ Xóa phần tử 5

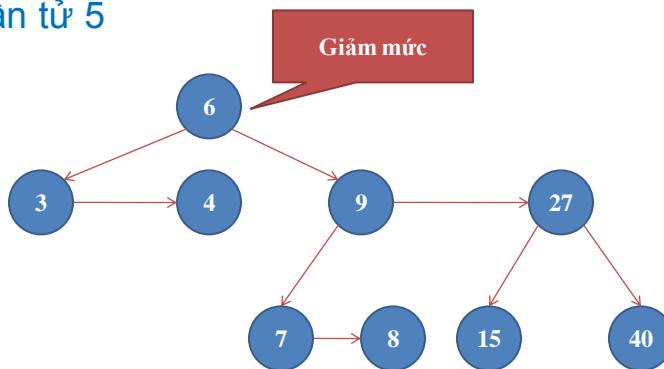


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

101

### ◦ Xóa phần tử 5

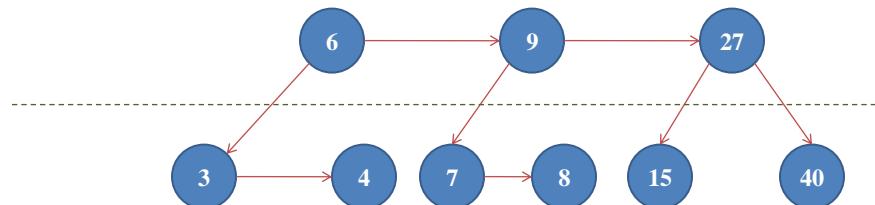


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

102

### ◦ Xóa phần tử 5

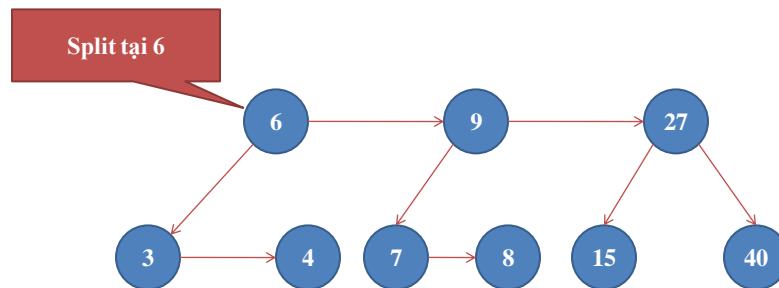


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

103

### ◦ Xóa phần tử 5

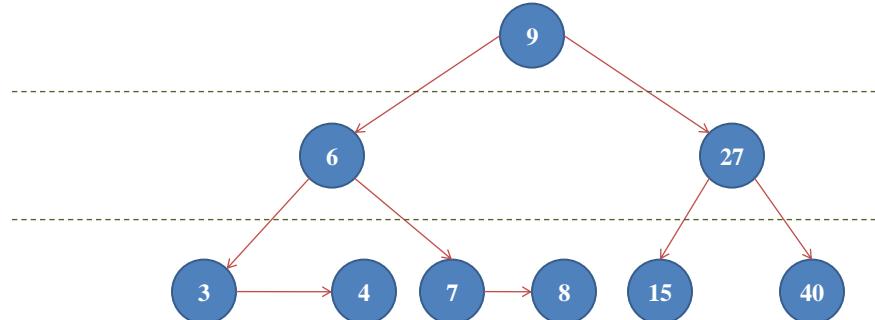


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Ví dụ

104

### ◦ Xóa phần tử 5



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

105

## Bài tập

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

106

## Bài tập

1. Xây dựng giải thuật xóa một đỉnh với khóa cho trước ra khỏi cây nhị phân tìm kiếm.
2. Hãy chứng tỏ rằng trường hợp tìm kiếm trung bình cho cây nhị phân tìm kiếm là  $O(\log_2 n)$ ?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

107

3. Biểu diễn tình trạng cây nhị phân tìm kiếm sau khi thực hiện các thao tác sau:

- ❑ Lần lượt thêm các node theo trình tự: M G B K S P D C A H L F X N T W R.
- ❑ Xóa M.
- ❑ Xóa S.
- ❑ Cho biết kết quả sau khi duyệt cây theo các trình tự giữa, trước và sau.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

108

4. Xây dựng giải thuật thực hiện các thao tác sau trên cây nhị phân tìm kiếm:

- Đếm số node lá.
- Tính độ cao cây.
- Tính độ cao của 1 node trong cây.
- Xuất ra các node có cùng độ cao.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

109

5. Biểu diễn tình trạng cây cân bằng AVL/cây AA sau khi thực hiện các thao tác sau:

- ❑ Lần lượt thêm các node theo trình tự: 13 7 2 11 19 16 4 3 1 8 12 6 24 14 20 23 18
- ❑ Xóa 13.
- ❑ Xóa 19

Lưu ý: cho biết các trường hợp mất cân bằng.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Bài tập

110

6. Hãy vẽ cây AVL với 12 nút có chiều cao cực đại trong tất cả các cây AVL 12 nút.

7. Tìm 1 dãy N khoá sao cho khi lần lượt dùng thuật toán thêm vào cây AVL sẽ phải thực hiện mỗi thao tác cân bằng (LL, LR, RL, RR) lại ít nhất 1 lần.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

111

## Hỏi và Đáp

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc dữ liệu và giải thuật

## NÉN DỮ LIỆU

Giảng viên:  
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

### Nội dung trình bày

2

Giới thiệu

Một số khái niệm

Giải thuật nén Huffman  
tĩnh

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Giới thiệu

3

## ◦ Thuật ngữ:

- Data compression
- Encoding
- Decoding
- Lossless data compression
- Lossy data compression
- ...

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Giới thiệu

4

## ◦ Nén dữ liệu

- Nhu cầu xuất hiện ngay sau khi hệ thống máy tính đầu tiên ra đời.
- Hiện nay, phục vụ cho các dạng dữ liệu đa phương tiện
- Tăng tính bảo mật.

## ◦ Ứng dụng:

- Lưu trữ
- Truyền dữ liệu

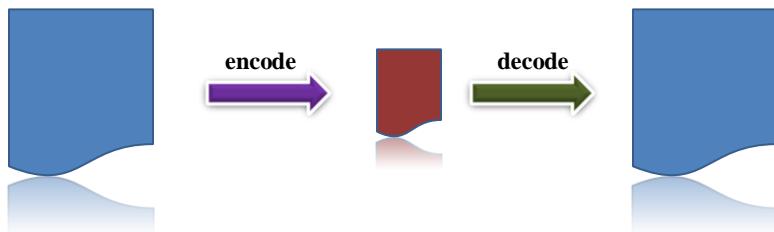
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Giới thiệu

5

### ◦ Nguyên tắc:

- Encode và decode sử dụng cùng một scheme.



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Khái niệm

6

### ◦ Tỷ lệ nén (Data compression ratio)

- Tỷ lệ giữa kích thước của dữ liệu nguyên thủy và của dữ liệu sau khi áp dụng thuật toán nén.

- Gọi:

- N là kích thước của dữ liệu nguyên thủy,
- $N_1$  là kích thước của dữ liệu sau khi nén.

- Tỷ lệ nén R:  
$$R = \frac{N}{N_1}$$

- Ví dụ:

- Dữ liệu ban đầu 8KB, nén còn 2 KB. Tỷ lệ nén: 4-1

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Khái niệm

7

### ◦ Tỷ lệ nén (Data compression ratio)

- Về khả năng tiết kiệm không gian: Tỷ lệ của việc giảm kích thước dữ liệu sau khi áp dụng thuật toán nén.

- Gọi:

- $N$  là kích thước của dữ liệu nguyên thủy,
- $N_1$  là kích thước của dữ liệu sau khi nén.

- Tỷ lệ nén R:

$$R = 1 - \frac{N_1}{N}$$

- Ví dụ:

- Dữ liệu ban đầu 8KB, nén còn 2 KB. Tỷ lệ nén: 75%

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Khái niệm

8

### ◦ Nén dữ liệu không mất mát thông tin (Lossless data compression)

- Cho phép dữ liệu nén được phục hồi nguyên vẹn như dữ liệu nguyên thủy (lúc chưa được nén).

- Ví dụ:

- Run-length encoding
- LZW
- ...

- Ứng dụng:

- Ảnh PCX, GIF, PNG,..
- Tập tin \*.ZIP
- Ứng dụng gzip (Unix)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Khái niệm

9

- Nén dữ liệu mất mát thông tin (Lossy data compression)
  - Dữ liệu nén được phục hồi
    - không giống hoàn toàn với dữ liệu nguyên thủy;
    - gần đủ giống để có thể sử dụng được.
  - Ứng dụng:
    - Dùng để nén dữ liệu đa phương tiện (hình ảnh, âm thanh, video):
      - Ảnh: JPEG, DjVu;
      - Âm thanh: AAC, MP2, MP3;
      - Video: MPEG-2, MPEG-4

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

10

## Nén Huffman tinh

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Giới thiệu

11

### ◦ Mong muốn:

- Một giải thuật nén bảo toàn thông tin;
- Không phụ thuộc vào tính chất của dữ liệu;
- Ứng dụng rộng rãi trên bất kỳ dữ liệu nào, với hiệu suất tốt.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Giới thiệu

12

### ◦ Tư tưởng chính:

- Phương pháp cũ: dùng 1 dãy bit cố định để biểu diễn 1 ký tự
  - David Huffman (1952): tìm ra phương pháp xác định mã tối ưu trên dữ liệu tĩnh :
    - Sử dụng vài bit để biểu diễn 1 ký tự (gọi là “mã bit” – bit code)
    - Độ dài “mã bit” cho các ký tự không giống nhau:
    - Ký tự xuất hiện nhiều lần: biểu diễn bằng mã ngắn;
    - Ký tự xuất hiện ít : biểu diễn bằng mã dài
- => Mã hóa bằng mã có độ dài thay đổi (Variable Length Encoding)

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Giới thiệu

13

- Giả sử có dữ liệu sau đây:

ADDAABBCCBAAABBCCCCBBCDAADDEEAA

Ký tự	Tần số xuất hiện
A	10
B	8
C	6
D	5
E	2

- Biểu diễn 8 bit/ký tự cần:

$$(10 + 8 + 6 + 5 + 2) * 8 = 248 \text{ bit}$$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Giới thiệu

14

- Dữ liệu:

ADDAABBCCBAAABBCCCCBBCDAADDEEAA

- Biểu diễn bằng chiều dài thay đổi:

Ký tự	Tần số	Mã
A	10	11
B	8	10
C	6	00
D	5	011
E	2	010

$$(10*2 + 8*2 + 6*2 + 5*3 + 2*3) = 69 \text{ bit}$$

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén

15

[B1]: Duyệt tập tin -> Lập bảng thống kê tần số xuất hiện của các ký tự.

[B2]: Xây dựng cây Huffman dựa vào bảng thống kê tần số xuất hiện

[B3]: Phát sinh bảng mã bit cho từng ký tự tương ứng

[B4]: Duyệt tập tin -> Thay thế các ký tự trong tập tin bằng mã bit tương ứng.

[B5]: Lưu lại thông tin của cây Huffman cho giải nén

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

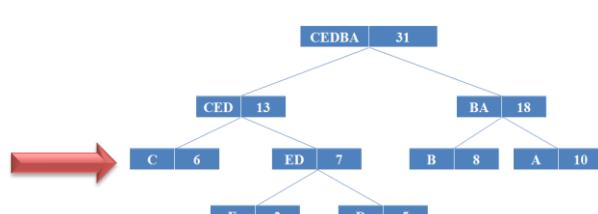
## Thuật toán nén

16

ADDAABBCCCBAAABBCCCCBBCDAADDEEEAA



Ký tự	Tần số
A	10
B	8
C	6
D	5
E	2



Ký tự	Mã
A	11
B	10
C	00
D	011
E	010

11011011111010000010111111010000  
000101010000111110110110100101111



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Thuật toán nén – Thống kê tần số

17

## ◦ Dữ liệu:

ADDAABBCCBAAABBCCCCBBBCDAADDEEAA

Ký tự	Tần số xuất hiện
A	10
B	8
C	6
D	5
E	2

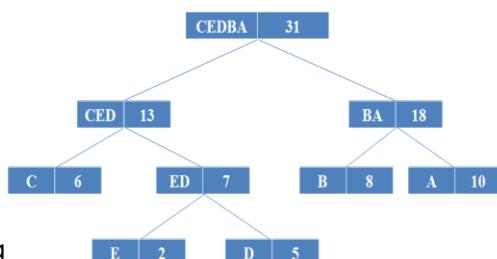
Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Thuật toán nén – Tạo cây Huffman

18

## ▫ Cây Huffman: cây nhị phân

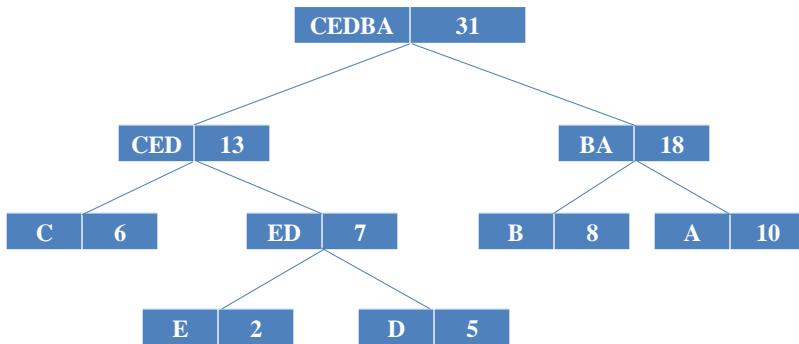
- Mỗi node lá chứa 1 ký tự
- Mỗi node cha chứa các ký tự của những node con.
- Trọng số của node:
  - Node con: tần số xuất hiện của ký tự tương ứng
  - Node cha: Tổng trọng số của các node con.



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

19



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

20

### ○ Phát sinh cây:

- ▣ Bước 1: Chọn trong bảng thống kê hai phần tử  $x,y$  có trọng số thấp nhất.
- ▣ Bước 2: Tạo 2 node của cây cùng với node cha  $z$  có trọng số bằng tổng trọng số của hai node con.
- ▣ Bước 3: Loại 2 phần tử  $x,y$  ra khỏi bảng thống kê.
- ▣ Bước 4: Thêm phần tử  $z$  vào trong bảng thống kê.
- ▣ Bước 5: Lặp lại Bước 1-4 cho đến khi còn 1 phần tử trong bảng thống kê.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

21

### ◦ Quy ước:

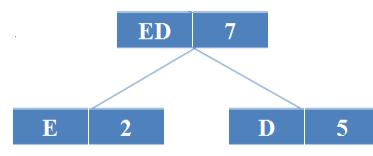
- Node có trọng số nhỏ hơn sẽ nằm bên nhánh trái. Node còn lại nằm bên nhánh phải.
- Nếu 2 node có trọng số bằng nhau
  - Node nào có ký tự nhỏ hơn thì nằm bên trái
  - Node có ký tự lớn hơn nằm bên phải.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

22

Ký tự	Tần số
A	10
B	8
C	6
D	5
E	2

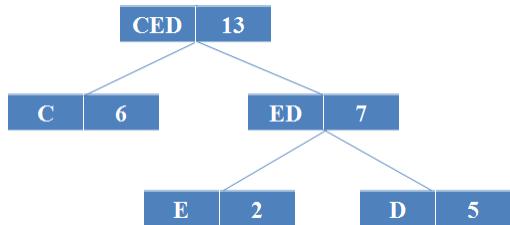


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

23

Ký tự	Tần số
A	10
B	8
<b>ED</b>	<b>7</b>
<b>C</b>	<b>6</b>

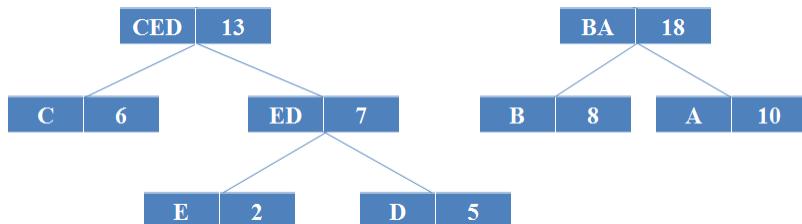


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

24

Ký tự	Tần số
CED	13
<b>A</b>	<b>10</b>
<b>B</b>	<b>8</b>

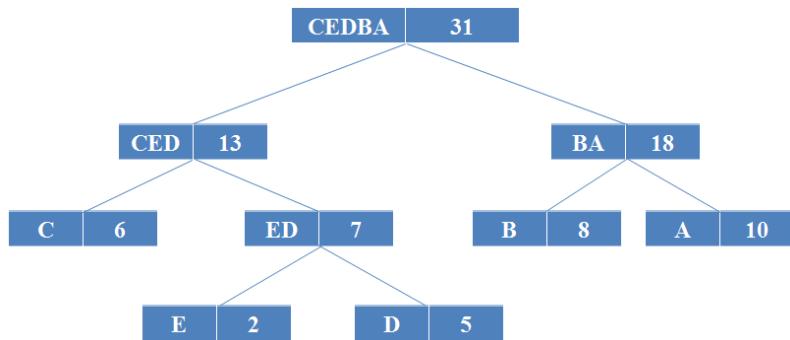


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

25

Ký tự	Tần số
BA	18
CED	13

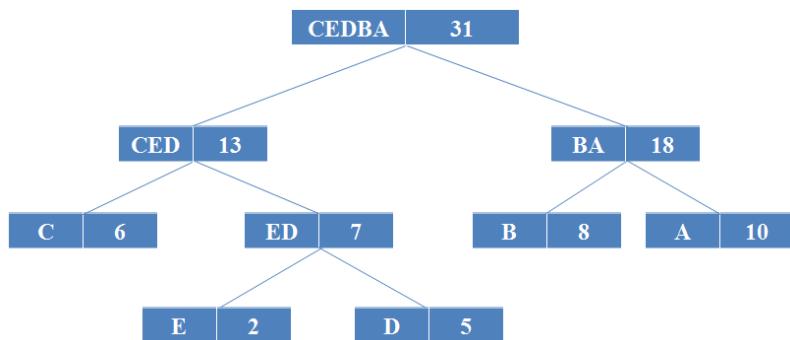


Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Tạo cây Huffman

26

Ký tự	Tần số
CEDBA	31



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Phát sinh mã bit

27

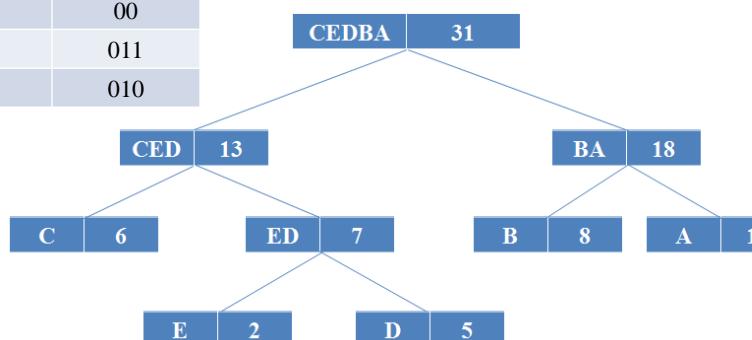
- Mã bit của từng ký tự: đường đi từ node gốc của cây Huffman đến node lá của ký tự đó.
- Cách thức:
  - ▣ Bit 0 được tạo ra khi đi qua nhánh trái
  - ▣ Bit 1 được tạo ra khi đi qua nhánh phải

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Phát sinh mã bit

28

Ký tự	Mã
A	11
B	10
C	00
D	011
E	010



Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Nén dữ liệu

29

- Duyệt tập tin cần nén
- Thay thế tất cả các ký tự trong tập tin bằng mã bit tương ứng của nó.

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán nén – Lưu lại thông tin

30

- Phục vụ cho việc giải nén.
- Cách thức:
  - Cây Huffman
  - Bảng tần số

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Thuật toán giải nén

31

- Phục hồi cây Huffman dựa trên thông tin đã lưu trữ.

- **Lắp**

- Đi từ gốc cây Huffman
- Đọc từng bit từ tập tin đã được nén
  - Nếu bit 0: đi qua nhánh trái
  - Nếu bit 1: đi qua nhánh phải
  - Nếu đến node lá: xuất ra ký tự tại node lá này.

- Cho đến khi nào hết dữ liệu

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Vấn đề khác

32

- Có thể không lưu trữ cây Huffman hoặc bảng thống kê tần số vào trong tập tin nén hay không?

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

## Vấn đề khác

33

- Thống kê săn trên dữ liệu lớn và tính toán săn cây Huffman cho bộ mã hóa và bộ giải mã.

- **Ưu điểm:**

- Giảm thiểu kích thước của tập tin cần nén.
  - Giảm thiểu chi phí của việc duyệt tập tin để lập bảng thống kê

- **Khuyết điểm:**

- Hiệu quả không cao trong trường hợp khác dạng dữ liệu đã thống kê

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

34

## Hỏi và Đáp

Cấu trúc dữ liệu và giải thuật - HCMUS 2011

# Cấu trúc dữ liệu và giải thuật

## CÁC CHIẾN LƯỢC TÌM KIẾM

Giảng viên:  
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

### Nội dung trình bày

2

Giới thiệu

Tìm kiếm tuần tự

Tìm kiếm nhị phân

Tìm kiếm theo bảng băm

Tổng kết

## Giới thiệu

3

- Thao tác tìm kiếm rất phổ biến trong cuộc sống hàng ngày.
  - Tìm kiếm hồ sơ, tập tin.
  - Tìm kiếm tên người trong danh sách.
  - ...

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán tìm kiếm

4

- Có nhiều loại:
  - Tìm kiếm tuần tự (Sequential/ Linear Search)
  - Tìm kiếm nhị phân (Binary Search)
  - ...
- Mục tiêu:
  - Tìm hiểu về 2 thuật toán tìm kiếm cơ bản.
  - Phân tích thuật toán để lựa chọn thuật toán phù hợp khi áp dụng vào thực tế.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

5

# Tìm kiếm tuần tự

Sequential Search

Linear Search

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

6

## Thuật toán tìm kiếm tuần tự

### ◦ Input:

- Dãy A,  $n$  phần tử
- Giá trị  $x$  cần tìm

### ◦ Output:

- Nếu  $x$  xuất hiện trong A: trả về vị trí xuất hiện đầu tiên của  $x$
- Nếu không: trả về  $n$  hoặc -1

### ◦ Thuật toán:

- Vét cạn (exhaustive)
- Dùng lính canh (sentinel)

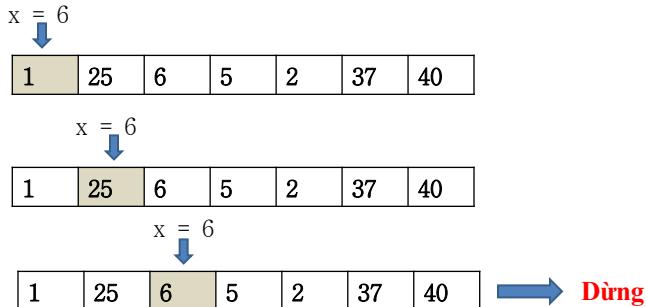
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Vết cạn

7

### ○ Thuật toán:

- Lần lượt so sánh  $x$  với các phần tử của mảng  $A$  cho đến khi gặp được phần tử cần tìm, hoặc hết mảng.
- Ví dụ:  $A = \{1, 25, 6, 5, 2, 37, 40\}$ ,  $x = 6$



## Tìm kiếm tuần tự - Vết cạn

8

### Thuật toán: Linear Exhaustive

- **Bước 1.** Khởi tạo biến chỉ số:  $i = 0$
- **Bước 2.** Kiểm tra xem có thực hiện hết mảng hay chưa: **So sánh  $i$  và  $n$** 
  - Nếu chưa hết mảng ( $i < n$ ), sang bước 3.
  - Nếu đã hết mảng ( $i \geq n$ ), thông báo không tìm thấy giá trị  $x$  cần tìm.
- **Bước 3. So sánh giá trị  $a[i]$  với giá trị  $x$  cần tìm**
  - Nếu  $a[i]$  bằng  $x$ : Kết thúc chương trình và thông báo đã tìm thấy  $x$ .
  - Nếu  $a[i]$  khác  $x$ , **tăng  $i$  thêm 1** và quay lại bước 2.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Vết cạn

9

- Nhận xét: Phép so sánh là phép toán sơ cấp được dùng trong thuật toán. Suy ra, số lượng các phép so sánh sẽ là thước đo độ phức tạp của thuật toán.
- Mỗi vòng lặp có 2 điều kiện cần kiểm tra:
  - ▣ Kiểm tra cuối mảng (bước 2)
  - ▣ Kiểm tra phần tử hiện tại có bằng  $x$ ? (bước 3)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Vết cạn

10

- Trường hợp  $x$  nằm ở 2 biên của mảng A: rất hiếm khi xuất hiện.
- Ước lượng số vòng lặp trung bình sẽ hữu ích hơn.
- Số phép so sánh trung bình:
$$2(1+2+ \dots + n)/n = n+1$$
$$\Rightarrow$$
 Số phép so sánh tăng/giảm tuyến tính theo số phần tử

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Vét cạn

11

- Vậy độ phức tạp của thuật toán là:
  - Tốt nhất:  $O(1)$ .
  - Trung bình:  $O(n)$ .
  - Xấu nhất:  $O(n)$ .

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Lính canh

12

- Trong thuật toán vét cạn, có 2 điều kiện được kiểm tra.
- Có thể bỏ việc kiểm tra điều kiện cuối mảng bằng cách dùng “lính canh”.
- Lính canh là phần tử có giá trị bằng với phần tử cần tìm và đặt ở cuối mảng.

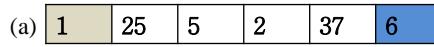
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Lính canh

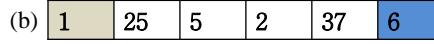
13

◦ Ví dụ:  $A = \{1, 25, 5, 2, 37\}$ ,  $x = 6$

$$x = 6$$



$$x = 6$$



$$x = 6$$



$$x = 6$$

$$x = 6$$

$$x = 6$$

→ return 5;

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Lính canh

14

Thuật toán: **LinearSentinel**

- **Bước 1.** Khởi tạo biến chỉ số:  $i = 0$
- **Bước 2.** So sánh giá trị  $a[i]$  với giá trị  $x$  cần tìm
  - Nếu  $a[i]$  bằng  $x$ :
    - Nếu  $i < n$ : Kết thúc chương trình và thông báo đã tìm thấy  $x$ .
    - Nếu  $i \geq n$ : Thông báo không tìm thấy  $x$  trong mảng.
  - Nếu  $a[i]$  khác  $x$ , **tăng  $i$  thêm 1** và quay lại bước 2.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tìm kiếm tuần tự - Lính canh

15

- Thực nghiệm cho thấy trong trường hợp  $n$  lớn, thời gian tìm kiếm giảm khi dùng phương pháp lính canh.
  - Với  $n = 15000$ : nhanh hơn khoảng 20% (0,22s so với 0,28s)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

16

## Tìm kiếm nhị phân

Binary Search

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán tìm kiếm nhị phân

17

- Với dãy A được sắp xếp thứ tự (ví dụ: tăng dần), độ phức tạp của thuật toán tìm kiếm tuần tự không đổi.
- Tận dụng thông tin của mảng đã được sắp xếp để giới hạn vị trí của giá trị cần tìm trong mảng.  
-> Thuật toán tìm kiếm nhị phân.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán tìm kiếm nhị phân

18

- **Input:**
  - Dãy A, n phần tử **đã được sắp xếp**
  - Giá trị x cần tìm
- **Output:**
  - Nếu x xuất hiện trong A: trả về một vị trí xuất hiện của x
  - Nếu không: trả về n hoặc -1

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

19

## ◦ Ý tưởng:

- So sánh  $x$  với phần tử chính giữa mảng  $A$ .
  - Nếu  $x$  là phần tử giữa thì dừng.
- Nếu không: xác định xem  $x$  có thể thuộc nửa trái hay nửa phải của  $A$ .
- Lặp lại 2 bước trên với nửa đã được xác định.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

20

Thuật toán: BinarySearch( $A[]$ ,  $n$ ,  $x$ )

- **Bước 1.** Khởi gán  $left = 0$  và  $right = n - 1$ .
- **Bước 2.** Trong khi  $left \leq right$ , thực hiện:
  - 2.1. Đặt  $mid = (left + right)/2$
  - 2.2. *So sánh giá trị  $x$  và  $a[mid]$ :*
    - Nếu  $x < a[mid]$ , gán  $right = mid - 1$ .
    - Nếu  $x > a[mid]$ , gán  $left = mid + 1$ .
    - Nếu  $x = a[mid]$ , thông báo đã tìm thấy  $x$  và kết thúc.
- Kết quả trả về không tìm thấy  $x$  nếu  $left > right^*$ .

\*Điều này có nghĩa là không còn phần tử nào trong mảng:  $x$  không có trong mảng

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

21

Cài đặt đệ quy: BinarySearch(A[], left, right, x)

- **Bước 1.** Nếu  $left > right$ : thông báo không tìm thấy x và thoát khỏi hàm.

- **Bước 2.**

- 2.1. Đặt  $mid = (left + right)/2$
- 2.2. So sánh giá trị x và  $a[mid]$ :
  - Nếu  $x < a[mid]$ , Gọi  $\text{BinarySearch}(A, left, mid - 1, x)$
  - Nếu  $x > a[mid]$ , Gọi  $\text{BinarySearch}(A, mid + 1, right, x)$
  - Nếu  $x = a[mid]$ , thông báo đã tìm thấy x và kết thúc (trả lại giá trị mid)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

22

- **Minh họa:**

- $A[] = \{1, 2, 6, 26, 28, 37, 40\}$ ,  $x = 2$

index	0	1	2	3	4	5	6
$A[i]$	1	2	6	26	28	37	40
Vòng 1	left			mid			right
Vòng 2	left	mid	right				

↓

$x = a[1] \rightarrow \text{return } 1$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

23

## ○ Minh họa:

- $A[] = \{1, 2, 6, 26, 28, 37, 40\}$ ,  $x = 40$

index	0	1	2	3	4	5	6
$A[i]$	1	2	6	26	28	37	40
Vòng 1	left			mid			right
Vòng 2					left	mid	right
Vòng 3							left mid right

$x = a[6] \rightarrow \text{return } 6$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

24

## ○ Minh họa:

- $A[] = \{1, 2, 6, 26, 28, 37, 40\}$ ,  $x = -7$

index	0	1	2	3	4	5	6
$A[i]$	1	2	6	26	28	37	40
Vòng 1	left			mid			right
Vòng 2	left	mid	right				
Vòng 3	left mid right						

right = -1, left = 0

=> right < left => thoát khỏi while,  
**return -1**

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán tìm kiếm nhị phân

25

### ◦ Phân tích thuật toán tuyến tính:

- Mỗi lần lặp thì chiều dài của mảng con giảm *khoảng*  $\frac{1}{2}$  so với mảng trước đó.
- $n = 2^k + m$  ( $0 \leq m < 2$ )
- $2^k \leq n < 2^{k+1} \Rightarrow k \leq \log_2 n < k+1 \Rightarrow k = \lfloor \log_2 n \rfloor$   
 $\Rightarrow$  mảng A ban đầu được chia nửa *khoảng*  $k$  lần.
- Số lần thực hiện vòng while là khoảng  $k$  lần, mỗi vòng lặp thực hiện 1 phép so sánh.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán tìm kiếm nhị phân

26

### ◦ Phân tích thuật toán tuyến tính:

- Trường hợp tốt nhất:  $k = 1 \Leftrightarrow x$  là phần tử chính giữa của mảng.
- Trường hợp xấu nhất:  $k = \lfloor \log_2 n \rfloor + 1 \Leftrightarrow x$  không thuộc mảng hoặc  $x$  là phần tử cuối cùng của mảng  
 $\Rightarrow$  Số phép so sánh tăng theo hàm logarit

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Thuật toán tìm kiếm nhị phân

27

## ◦ Độ phức tạp của tìm kiếm nhị phân

- Trường hợp tốt nhất:  $O(1)$
- Trường hợp trung bình:  $O(\log_2 n)$
- Trường hợp xấu nhất:  $O(\log_2 n)$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# So sánh hiệu suất

28

## ◦ So sánh trường hợp xấu nhất của 2 thuật toán:

Kích thước mảng	T/h xấu nhất	
	Tuần tự	Nhi phân
100.000	100.000	16
200.000	200.000	17
400.000	400.000	18
800.000	800.000	19
1.600.000	1.600.000	20

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tổng kết

29

- Có nhiều thuật toán tìm kiếm, ước lượng số phép so sánh của mỗi thuật toán cho biết hiệu suất của thuật toán.
- Thuật toán tuần tự tìm kiếm cho đến khi tìm thấy giá trị cần tìm hoặc hết mảng
- Hiệu suất của tìm kiếm tuần tự trong trường hợp xấu nhất là 1 hàm tuyến tính theo số phần tử mảng.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Tổng kết

30

- Nếu mảng đã được sắp xếp thì nên dùng tìm kiếm nhị phân.
- Tìm kiếm nhị phân dùng kết quả của phép so sánh để thu hẹp vùng tìm kiếm kế tiếp.
- Hiệu suất của tìm kiếm nhị phân là một hàm logarit theo số phần tử mảng.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

31

## Tìm kiếm theo bảng băm

Hash Table

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

32

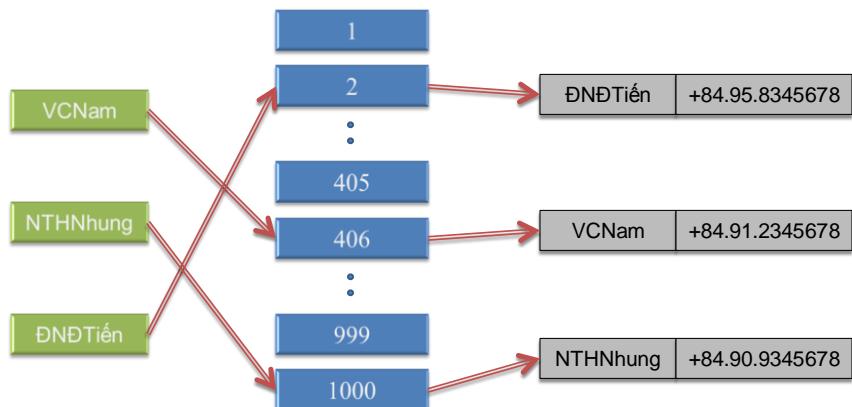
## Khái quát về hash

- **Vấn đề:** Cho trước 1 tập S gồm các phần tử được đặc trưng bởi giá trị khóa. Trên giá trị các khóa này có quan hệ thứ tự. Tổ chức S như thế nào để tìm kiếm 1 phần tử có khóa k cho trước có độ phức tạp ít nhất trong giới hạn bộ nhớ cho phép?
- **Ý tưởng:** Biến đổi khóa k thành một số (bằng hàm hash) và sử dụng số này như là địa chỉ để tìm kiếm trên bảng dữ liệu.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ về một bảng băm

33



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Độ phức tạp

34

- Chi phí tìm kiếm trung bình:  $O(1)$
- Chi phí tìm kiếm trong trường hợp xấu nhất:  $O(n)$  (rất ít gặp).

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Hàm băm (hash function)

35

- **Định nghĩa:** là hàm biến đổi khóa k của phần tử thành địa chỉ trong *bảng băm*.

Ví dụ:  $H(k) = k \text{ mod } M$ .

- **Tổng quát về phép biến đổi khóa:** Là 1 ánh xạ thích hợp từ tập các khóa U vào tập các địa chỉ A.

$$H: U \rightarrow A$$

$$k \rightarrow a = h(k)$$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Khó khăn của hàm băm

36

- Tập các giá trị khóa (U) có thể lớn hơn rất nhiều so với số khóa thực tế (K) rất nhiều.

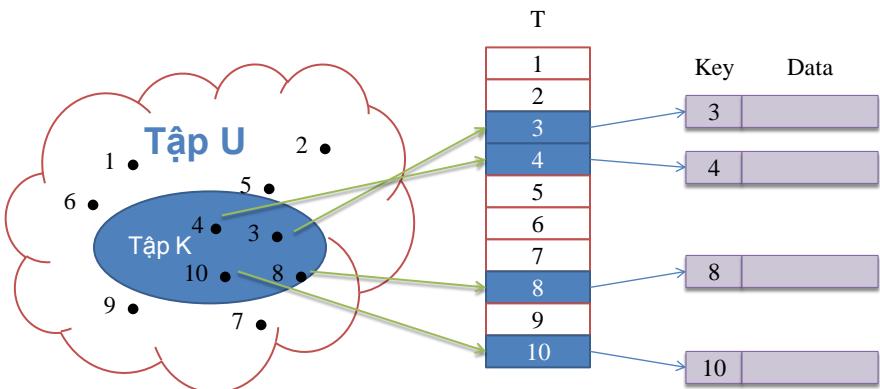
- Ví dụ: Quản lý danh sách 1000 sinh viên, mã sinh viên gồm 7 chữ số.

Có  $U = 10^7$  khóa so với  $K = 1000$ .

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ

37



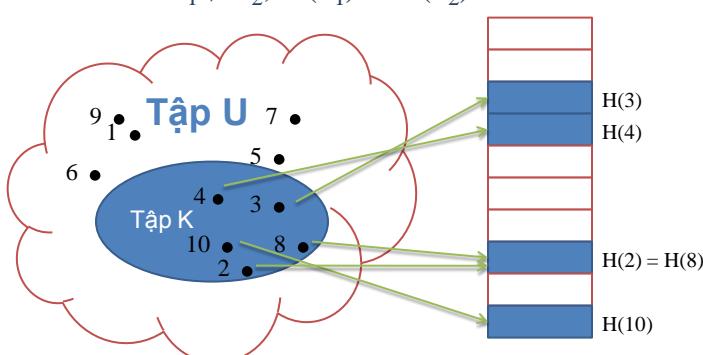
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Sự đụng độ (collision)

38

- $\exists k_1, k_2 \in K:$

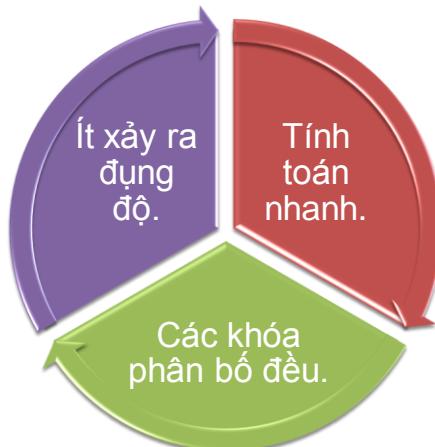
$$k_1 \neq k_2, H(k_1) = H(k_2)$$



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Những yêu cầu đối với hàm băm

39



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ về hàm băm

40

- Xét lại ví dụ về danh sách sinh viên:

Với kích thước bảng là  $M = 1000$ , ta có thể chọn hàm băm như sau:

$$H(k) = k \bmod M.$$

- Khóa này thỏa mãn yêu cầu tính toán nhanh và trải đều trên bảng.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Các phương pháp xử lý đụng độ

41

- Phương pháp nối kết (chaining)
- Phương pháp địa chỉ mở (Open-addressing)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Phương pháp nối kết

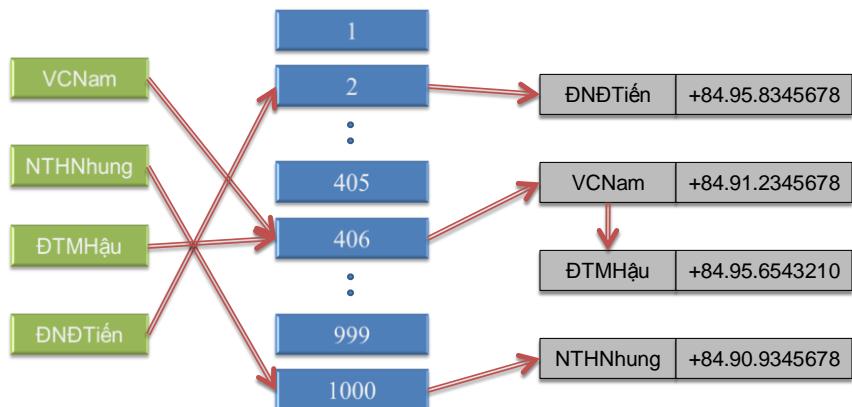
42

- Ứng với mỗi địa chỉ của bảng, ta có một danh sách liên kết chứa các phần tử có khóa khác nhau mà có cùng địa chỉ đó.
- Ta sẽ có danh sách (bảng băm) gồm  $M$  phần tử chứa địa chỉ đầu của các danh sách liên kết.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giải quyết đụng độ với phương pháp nối kết

43



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Phương pháp địa chỉ mở

44

- Tên gọi khác:
  - ▣ Phương pháp dò
  - ▣ Phương pháp thử
- Ý tưởng:
  - ▣ Khi đụng độ xảy ra, ta sẽ thử tìm đến vị trí kế tiếp nào đó trong bảng cho đến khi tìm thấy vị trí nào còn trống.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Các cách thực hiện

45

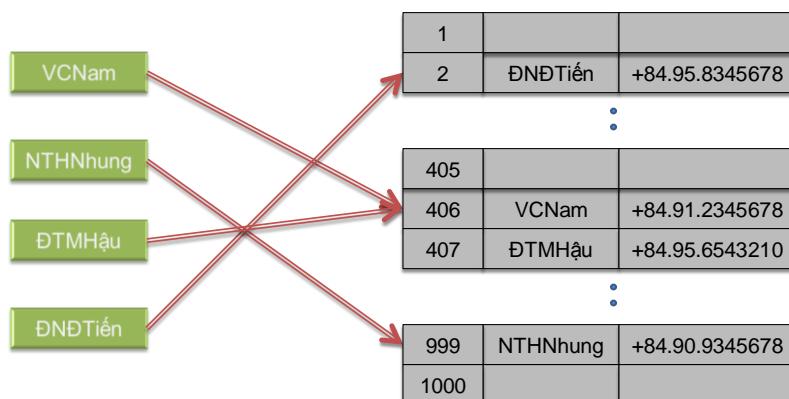
- Phương pháp dò tuyến tính (Linear probing)
- Phương pháp dò bậc 2 (Quadratic probing)
- Phương pháp băm kép (Double hashing)

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giải quyết đụng độ bằng phương pháp dò tuyến tính

46

- Ý tưởng:  $H(k, i) = (h(k) + i) \text{ mod } M$



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Phương pháp dò bậc 2 và băm kép

47

- *Phương pháp dò bậc 2:*

$$H(k, i) = (h(k) + i^2) \bmod M$$

- *Phương pháp băm kép:*

$$H(k, i) = (h_1(k) + i * h_2(k)) \bmod M$$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ưu thế của phương pháp địa chỉ mở so với phương pháp nối kết

48

- Đơn giản khi cài đặt.
- Sử dụng cấu trúc dữ liệu cơ bản.
- Phương pháp địa chỉ mở giải quyết được độ nhung lục có thể gây ra đụng độ mới.
- Phương pháp nối kết không bị ảnh hưởng về tốc độ khi mảng gần đầy.
- Ít tốn bộ nhớ khi mảng thừa (ít phần tử).

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập

49

1. Cho bảng băm có kích thước  $M = 11$ . Hàm băm:  $h(k) = k \bmod M$ . Dùng phương pháp địa chỉ mở. Cho biết kết quả sau khi thêm vào bảng băm các khóa 10, 22, 31, 4, 15, 28, 17, 88, 59, với 3 phương pháp xử lý đụng độ:
  - a. Dò tuyến tính.
  - b. Dò bậc 2.
  - c. Băm kép  $h_2(k) = (k \bmod 19)+1$ .

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập

50

2. Cho từ điển Anh – Việt có 15.000 từ, hãy tổ chức cấu trúc dữ liệu bảng băm và cho biết hàm băm thích hợp giúp cho việc tra từ hiệu quả nhất.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Hỏi và Đáp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

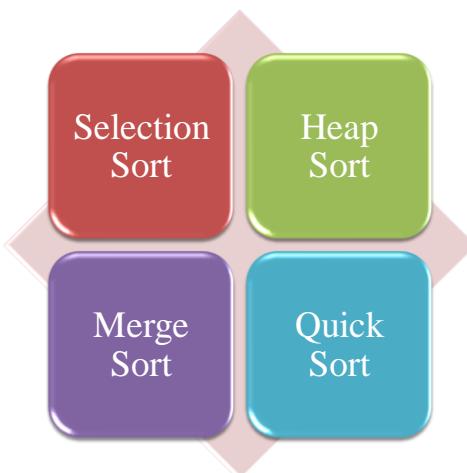
# Cấu trúc dữ liệu và giải thuật

## CÁC THUẬT TOÁN SẮP XẾP

Giảng viên:  
Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến

### Nội dung

2



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

3

# Giới thiệu

Bài toán sắp xếp

Các thuật toán sắp xếp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

4

# Giới thiệu

- Bài toán sắp xếp: Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt chúng theo một thứ tự thỏa yêu cầu cho trước
- Ví dụ: danh sách trước khi sắp xếp:  
 $\{1, 25, 6, 5, 2, 37, 40\}$   
Danh sách sau khi sắp xếp:  
 $\{1, 2, 5, 6, 25, 37, 40\}$
- Thông thường, sắp xếp giúp cho việc tìm kiếm được nhanh hơn.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giới thiệu

5

### ◦ Các phương pháp sắp xếp thông dụng:

- Buble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- Merge Sort
- Heap Sort
- Radix Sort

 Cần tìm hiểu các phương pháp sắp xếp và lựa chọn phương pháp phù hợp khi sử dụng.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

6

## Sắp xếp chọn

Selection Sort

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ý tưởng

7

- Mô phỏng cách sắp xếp tự nhiên nhất trong thực tế
  - ▣ Chọn phần tử nhỏ nhất và đưa về vị trí đúng là đầu dãy hiện hành.
  - ▣ Sau đó xem dãy hiện hành chỉ còn  $n-1$  phần tử.
  - ▣ Lặp lại cho đến khi dãy hiện hành chỉ còn 1 phần tử.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán

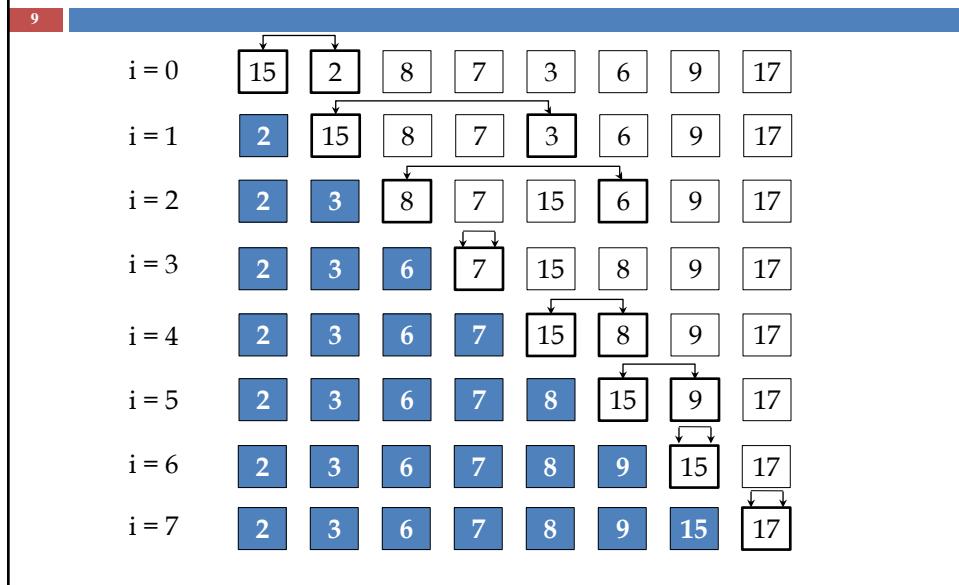
8

Các bước của thuật toán:

- **Bước 1.** Khởi gán  $i = 0$ .
- **Bước 2.** Bước lặp:
  - ▣ 2.1. *Tìm  $a[min]$  nhỏ nhất trong dãy từ  $a[i]$  đến  $a[n-1]$*
  - ▣ 2.2. *Hoán vị  $a[min]$  và  $a[i]$*
- **Bước 3.** So sánh  $i$  và  $n$ :
  - ▣ Nếu  $i \leq n$  thì *tăng  $i$  thêm 1* và lặp lại bước 2.
  - ▣ Ngược lại: Dừng thuật toán.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ



## Đánh giá

10

### ◦ Đánh giá giải thuật:

#### ▫ Số phép so sánh:

- Tại lượt  $i$  bao giờ cũng cần  $(n-i-1)$  số lần so sánh
- Không phụ thuộc vào tình trạng dãy số ban đầu

$$\text{Số phép so sánh} = \sum_{i=0}^{n-1} (n-i-1) = \frac{n(n-1)}{2}$$

## Đánh giá

11

### ◦ Số phép gán:

▪ Tốt nhất:  $\sum_{i=0}^{n-1} 4 = 4n$

▪ Xấu nhất:

$$\sum_{i=0}^{n-1} (4 + n - i - 1) = \frac{n(n+7)}{2}$$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

12

## Sắp xếp vun đống

Heap Sort

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ý tưởng

13

- Ý tưởng: khi tìm phần tử nhỏ nhất ở bước i, phương pháp Selection sort không tận dụng được các thông tin đã có nhờ vào các phép so sánh ở bước i-1 → cần khắc phục nhược điểm này.
- J. Williams đã đề xuất phương pháp sắp xếp Heapsort.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Heap

14

- Định nghĩa Heap:

▪ Giả sử xét trường hợp sắp xếp tăng dần, Heap được định nghĩa là một dãy các phần tử  $a_l, a_{l+1}, \dots, a_r$  thỏa: với mọi  $i$  thuộc  $[l, r]$  (chỉ số bắt đầu từ 0)

$$a_i \geq a_{2i+1}$$

$$a_i \geq a_{2i+2} \quad \{(a_i, a_{2i+1}), (a_i, a_{2i+2}) \text{ là các cặp phần tử liên đới}\}$$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Các tính chất của Heap

15

- Nếu  $a_i, a_{i+1}, \dots, a_r$  là một heap thì phần tử  $a_i$  (đầu heap) luôn là phần tử lớn nhất.
- Mọi dãy  $a_i, a_{i+1}, \dots, a_r$  với  $2i + 1 > r$  là heap.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán

16

- Giai đoạn 1: Hiệu chỉnh dãy ban đầu thành heap (bắt đầu từ phần tử giữa của dãy)
- Giai đoạn 2: sắp xếp dựa trên heap.
  - ▣ Bước 1: đưa phần tử lớn nhất về vị trí đúng ở cuối dãy
  - ▣ Bước 2:
    - Loại bỏ phần tử lớn nhất ra khỏi heap:  $r = r - 1$
    - Hiệu chỉnh lại phần còn lại của dãy.
  - ▣ Bước 3: So sánh  $r$  và  $l$ :
    - Nếu  $r > l$  thì lặp lại bước 2.
    - Ngược lại, dừng thuật toán.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Heap Sort

17

- ◎ Mã giả (Tựa ngôn ngữ lập trình C):

```
void HeapSort(int a[], int n)
{
    TaoHeap(a, n-1);
    r = n-1;
    while(r > 0)
    {
        HoanVi(a[0], a[r]);
        r = r - 1;
        HieuChinh(a, 0, r);
    }
}
```

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Heap Sort

18

- ◎ Mã giả:

```
void TaoHeap(int a[], int r)
{
    int l = r/2;
    while(l > 0)
    {
        HieuChinh(a, l, r);
        l = l - 1;
    }
}
```

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Heap Sort

19

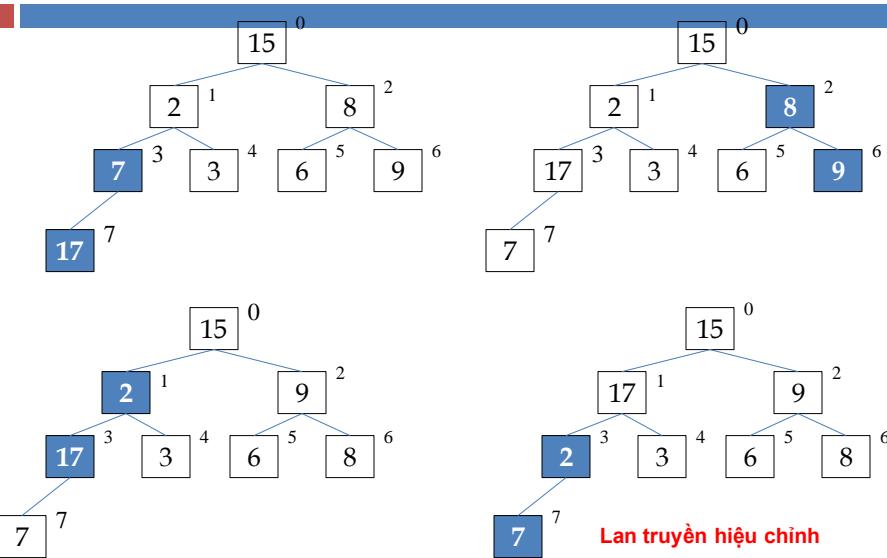
◦ Mã giả:

```
void HiệuChinh(int a[], int l, int r)
{
    i = l; j = 2*i+1; x = a[i];
    while(j <= r)
    {
        if(có dù 2 phần tử liên đới)
            //xác định phần tử liên đới lớn nhất
        if(a[j] < x) //thỏa quan hệ liên đới
            //dừng
        else
            //hiệu chỉnh
        //xét khả năng hiệu chỉnh lan truyền
    }
}
```

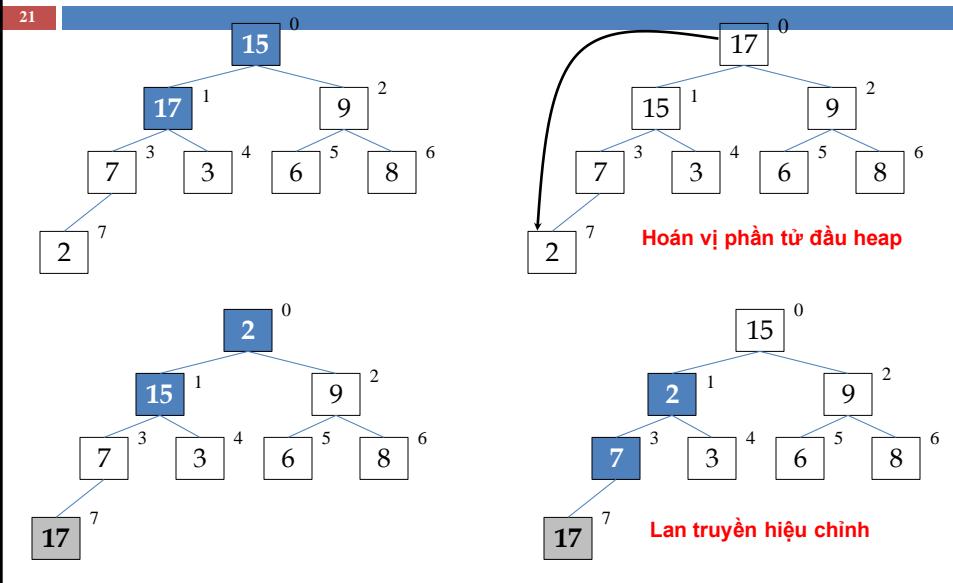
Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ

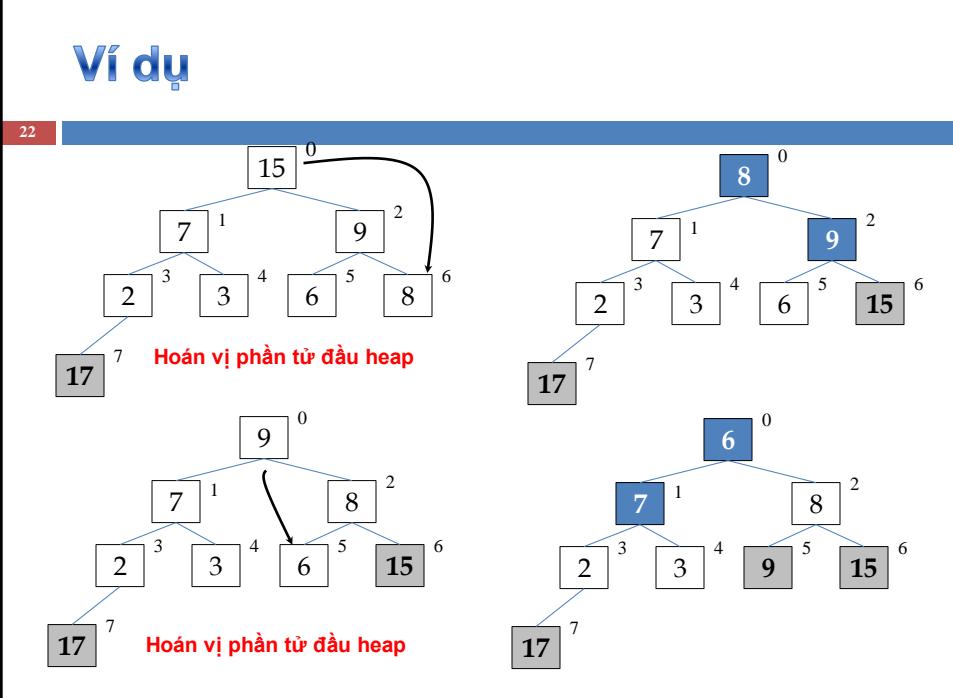
20



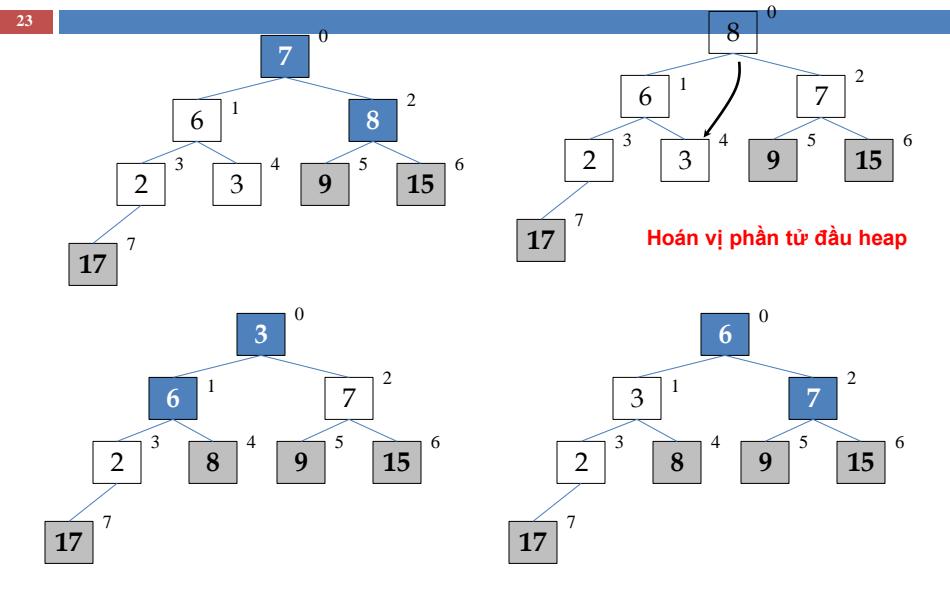
## Ví dụ



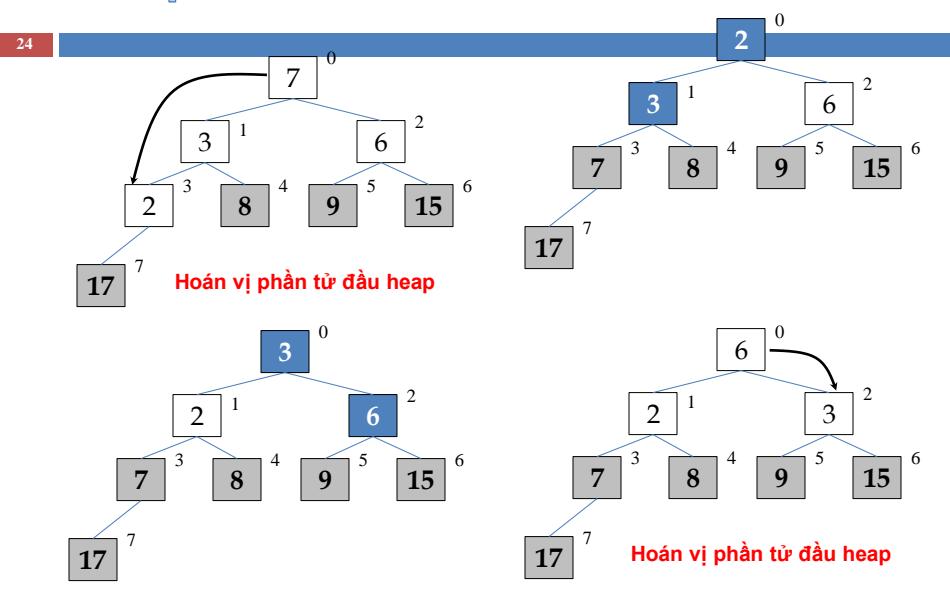
## Ví dụ



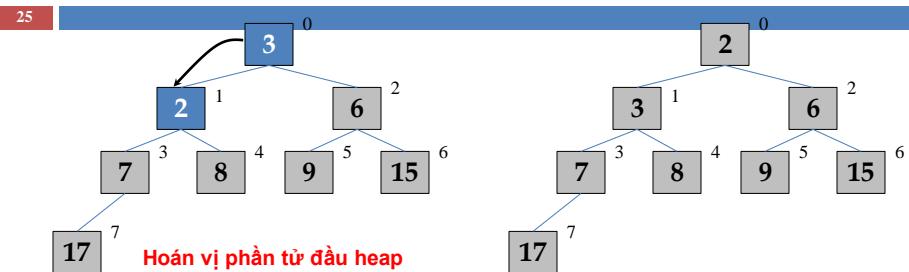
## Ví dụ



## Ví dụ



## Ví dụ



Mảng sau khi sắp xếp:

2 3 6 7 8 9 15 17

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Heap Sort

26

### ◦ Đánh giá giải thuật:

- Độ phức tạp của giải thuật trong trường hợp xấu nhất là  $O(n \log_2 n)$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

27

## Sắp xếp nhanh

Quick Sort

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

28

## Ý tưởng

○ Giải thuật: dựa trên việc phân hoạch dãy ban đầu thành 2 phần:

- Dãy con 1:  $a_0, a_1, \dots, a_i$  có giá trị nhỏ hơn  $x$
- Dãy con 2:  $a_j, \dots, a_{n-1}$  có giá trị lớn hơn  $x$ .

Dãy ban đầu được phân thành 3 phần:

$a_k < x$	$a_k = x$	$a_k > x$
$k = 0 \dots i$	$k = i+1 \dots j$	$k = j+1, \dots n-1$

- Phần 2 đã có thứ tự
- Phần 1, 3: cần sắp thứ tự, tiến hành phân hoạch từng dãy con theo cách phân hoạch dãy ban đầu

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Thuật toán – Giai đoạn phân hoạch

29

1. Chọn phần tử  $a[k]$  trong dãy là giá trị mốc,  $0 \leq k \leq r-1$ 
  - Gán  $x = a[k]$ ,  $i = l$ ,  $j = r$ .
  - Thường chọn phần tử ở giữa dãy:  $k = (l+r)/2$
2. Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  sai vị trí:
  - 2.1. Trong khi ( $a[i] < x$ ), tăng  $i$ .
  - 2.2. Trong khi ( $a[j] > x$ ), giảm  $j$ .
  - 2.3. Nếu  $i \leq j$  thì:
    - Hoán vị  $a[i], a[j]$ ,
    - Tăng  $i$  và giảm  $j$
3. So sánh  $i$  và  $j$ :
  - Nếu  $i < j$ : lặp lại bước 2
  - Ngược lại: dừng phân hoạch.

## Thuật toán Quick Sort

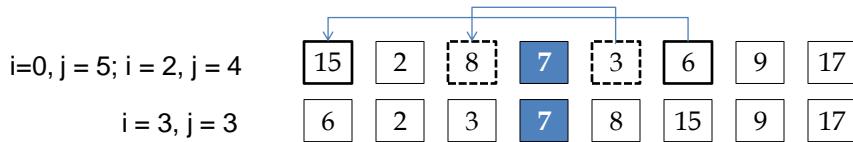
30

- Bước 1: phân hoạch dãy ban đầu thành 3 dãy:
  - Dãy 1:  $a_l \dots a_j < x$
  - Dãy 2:  $a_{j+1} \dots a_{i-1} = x$
  - Dãy 3:  $a_i \dots a_r > x$
- Bước 2: sắp xếp:
  - Nếu  $l < j$  : phân hoạch dãy  $a_l \dots a_j$
  - Nếu  $i < r$  : phân hoạch dãy  $a_i \dots a_r$

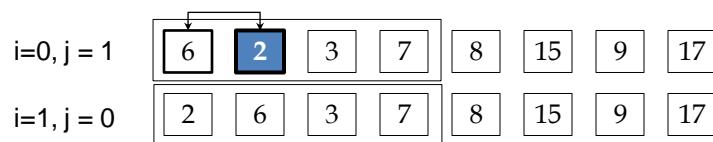
## Ví dụ

31

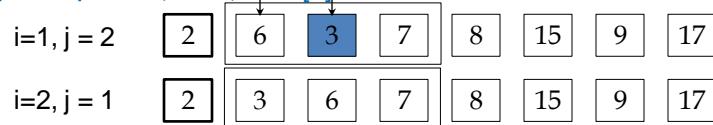
Phân hoạch dãy ban đầu:  $i = 0, r = 7, x = a[3]$



Phân hoạch đoạn  $i = 0, r = 3, x = a[1]$



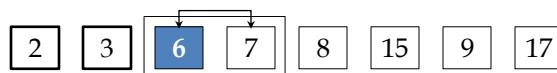
Phân hoạch đoạn  $i = 1, r = 3, x = a[2]$



## Ví dụ (tt)

32

◦ Phân hoạch đoạn  $i = 2, r = 3, x = a[2]$



◦ Phân hoạch đoạn  $i = 3, r = 7, x = a[5]$



◦ Phân hoạch đoạn  $i = 3, r = 4, x = a[3]$



◦ Phân hoạch đoạn  $i = 5, r = 7, x = a[6]$



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Bài tập

33

- Chạy tay thuật toán Quick Sort để sắp xếp mảng A trong 2 trường hợp tăng dần và giảm dần.

$$A = \{2, 9, 5, 12, 20, 15, -8, 10\}$$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Quick Sort

34

- Đánh giá giải thuật:

- Hiệu quả phụ thuộc vào việc chọn giá trị mốc
  - Tốt nhất là phần tử median.
  - Nếu phần tử mốc là cực đại hay cực tiểu thì việc phân hoạch không đồng đều.
- Bảng tổng kết:

	Độ phức tạp
Tốt nhất	$n * \log(n)$
Trung bình	$n * \log(n)$
Xấu nhất	$n^2$

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

35

## Sắp xếp trộn

Merge Sort

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

36

## Giới thiệu

- Thực hiện theo hướng chia để trị.
- Do John von Neumann đề xuất năm 1945.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giải thuật

37

- Nếu dãy có chiều dài là 0 hoặc 1: đã được sắp xếp.
- **Ngược lại:**
  - Chia dãy thành 2 dãy con (chiều dài tương đương nhau).
  - Sắp xếp trên từng dãy con bằng thuật toán Merge Sort.
  - Trộn 2 dãy con (đã được sắp xếp) thành một dãy mới đã được sắp xếp.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Giải thuật

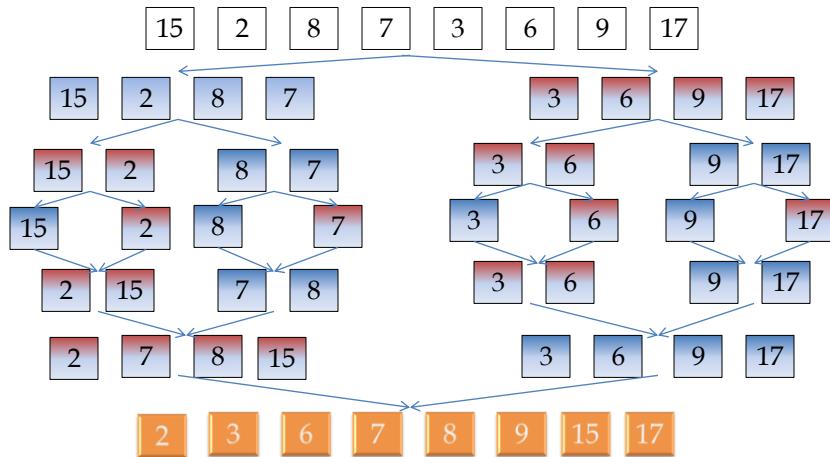
38

- Input: Dãy A và các chỉ số left, right (sắp xếp dãy A gồm các phần tử có chỉ số từ *left* đến *right*).
- Output: Dãy A đã được sắp xếp
  - MergeSort(A, left, right)
  - {
  - if (left < right) {
  - mid = (left + right)/2;
  - MergeSort(A, left, mid);
  - MergeSort(A, mid+1, right);
  - Merge(A, left, mid, right);
  - }
  - }

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Ví dụ

39



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Đánh giá

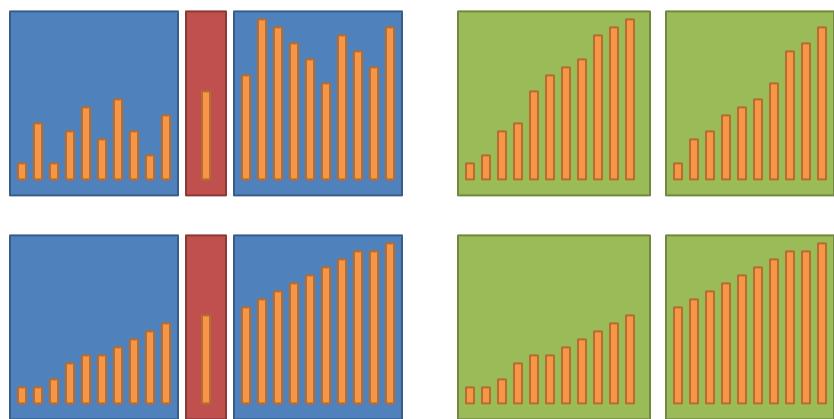
40

- Số lần chia các dãy con:  $\log_2 n$
- Chi phí thực hiện việc trộn hai dãy con đã sắp xếp tỷ lệ thuận với  $n$ .
- Chi phí của Merge Sort là  $O(n \log_2 n)$
- Thuật toán không sử dụng thông tin nào về đặc tính của dãy cần sắp xếp => chi phí thuật toán là không đổi trong mọi trường hợp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## So sánh tư tưởng sắp xếp giữa Quick sort và Merge sort

41



Cấu trúc dữ liệu và giải thuật – HCMUS 2011

42

## Kết luận

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kết luận

43

- Các thuật toán Bubble sort, Selection sort, Insertion sort
  - Cài đặt thuật toán đơn giản.
  - Chi phí của thuật toán cao:  $O(n^2)$ .
- Heap sort được cải tiến từ Selection sort nhưng chi phí thuật toán thấp hơn hẳn ( $O(n \log_2 n)$ )

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kết luận

44

- Các thuật toán Quick sort, Merge sort là những thuật toán theo chiến lược chia để trị.
  - Cài đặt thuật toán phức tạp
  - Chi phí thuật toán thấp:  $O(n \log_2 n)$
  - Rất hiệu quả khi dùng danh sách liên kết.
  - Trong thực tế, Quick sort chạy nhanh hơn hẳn Merge sort và Heap sort.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

## Kết luận

45

- Người ta chứng minh  $O(n \log_2 n)$  là ngưỡng chặng dưới của các thuật toán sắp xếp dựa trên việc so sánh giá trị của các phần tử.

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

46

## Hỏi và Đáp

Cấu trúc dữ liệu và giải thuật – HCMUS 2011

# Bài 01

## ÔN TẬP KỸ THUẬT LẬP TRÌNH

### MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu và sử dụng kiểu con trỏ trong C++.
- Phân biệt truyền tham biến và truyền tham trị.
- Thao tác đọc/ghi trên tập tin văn bản.
- Hiểu rõ về lập trình đệ quy, viết được các chương trình đệ quy.

### THỜI GIAN THỰC HÀNH

Từ 6-15 tiết, gồm

- Con trỏ: 2-5 tiết
- Truyền tham biến và truyền tham trị: 1-3 tiết
- Thao tác đọc/ghi trên tập tin văn bản: 2-4 tiết
- Lập trình đệ quy: 1-3 tiết

## 1. CON TRỎ

Con trỏ là khái niệm đặc biệt trong C/C++, là loại biến dùng để chứa địa chỉ. Các thao tác với con trỏ lần lượt qua các bước:

- Khai báo biến con trỏ
- Khởi tạo con trỏ dùng cấp phát vùng nhớ
- Truy xuất giá trị ô nhớ từ biến con trỏ
- Hủy vùng nhớ đã xin cấp phát

### 1.1. Khai báo biến con trỏ trong C++

<KiểuDữLieu> \*<TênBiến>;

Ví dụ:

```
int* pa; // con trỏ đến kiểu int  
DIEM *pd; // con trỏ đến kiểu DIEM
```

Để xác định địa chỉ của một ô nhớ: toán tử &

Ví dụ:

```
int a = 1;  
int* pa = &a; // con trỏ trỏ đến ô nhớ của biến a
```

### 1.2. Khởi tạo biến con trỏ dùng cấp phát vùng nhớ (cấp phát động)

Sử dụng toán tử new

Ví dụ:

```
int* pInt = new int; // xin cấp phát vùng nhớ cho 1 số nguyên  
DIEM *pDiem = new DIEM; // xin cấp phát vùng nhớ cho 1 biến kiểu cấu trúc DIEM
```

Toán tử new còn có thể sử dụng thẻ cấp phát vùng nhớ cho **nhiều** phần tử.

```
int* arr = new int[5]; // xin cấp phát vùng nhớ cho 5 số nguyên
```

Lưu ý:

Để kiểm tra cấp phát vùng nhớ thành công hay không, ta dùng con trỏ đặc biệt NULL.

NULL là con trỏ đặc biệt, có thể được gán cho các biến con trỏ của các kiểu dữ liệu khác nhau.

Ví dụ:

```
int* pInt = NULL;  
DIEM* pDiem = NULL;
```

đều hợp lệ.

Để kiểm tra cấp phát thành công, ta làm như sau:

```
DIEM* pDiem = NULL; // khai báo con trỏ và gán bằng NULL  
pDiem = new DIEM; // xin cấp phát vùng nhớ  
if (pDiem == NULL) // nếu pDiem vẫn bằng NULL thì xin cấp phát không thành công  
    printf("Cap phat khong thanh cong");
```

### 1.3. Truy xuất giá trị ô nhớ từ biến con trỏ

#### 1.3.1. Đối với các kiểu dữ liệu cơ bản (như kiểu int, float, ...)

Để xác định giá trị ô nhớ tại địa chỉ trong biến con trỏ: toán tử \*

Ví dụ:

Với khai báo các biến a, pa

```
int a = 1;  
int* pa = &a; // con trỏ trỏ đến ô nhớ của biến a
```

câu lệnh

```
printf("%d\n", *pa);
```

sẽ xuất ra “1”

Giải thích:

```
int a = 1;
```

Với khai báo này, một ô nhớ sẽ được cấp phát và nội dung ô nhớ là 1

```
int* pa = &a;
```

Sau khai báo này, biến pa sẽ giữ địa chỉ ô nhớ vừa được cấp phát cho biến a

Khi đó, \*pa sẽ lấy nội dung của ô nhớ được trỏ đến bởi biến pa, mà biến pa giữ địa chỉ ô nhớ được cấp phát cho biến a.

Vậy \*pa = a = 1.

#### 1.3.2. Đối với các kiểu dữ liệu cấu trúc (như kiểu SINHVIEN, DIEM, ...)

Để truy xuất các thành phần của kiểu cấu trúc, dùng ->

Ví dụ:

Với kiểu cấu trúc DIEM được định nghĩa như sau

```
struct DIEM  
{
```

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010

```

        int hoanhDo, tungDo;
    } ;
DIEM *pDiem = new DIEM;

```

Để truy xuất thành phần dùng

```
pDiem->hoanhDo và pDiem->tungDo
```

#### 1.4. Hủy vùng nhớ đã xin cấp phát

Để hủy vùng nhớ đã xin cấp phát, dùng toán tử delete

Với khai báo

```

int* pa = new int;
int* pb = new int[5];

```

Cách hủy tương ứng sẽ là

```

delete pa;
delete pb[];

```

#### Bài tập (code mẫu: ConTroCoBan)

```

#include <stdio.h>

struct DIEM
{
    int hoanhDo, tungDo;
} ;

void main()
{
// khai tao cac bien gia tri

    int a = 1;
    DIEM d;
    d.hoanhDo = 1;
    d.tungDo = 2;

// khai bao bien con tro va tro den vung nho cua cac bien gia tri da co
    int *pa = &a;
    int *pb = pa;
    DIEM *pd = &d;

// xac dinh dia chi o nho: toan tu %
    printf("Dia chi o nho: %d \n", &a);

// truy xuat gia tri o nho tu bien con tro: toan tu *
    printf("Gia tri a: %d \n", *pa);

// truy xuat thanh phan trong cau truc

    printf("Diem D: (%d,%d) \n", d.hoanhDo, d.tungDo); //doi voi bien gia tri: .
    printf("Diem D: (%d,%d) \n", pd->hoanhDo, pd->tungDo); // doi voi bien con
tro: ->
}

```

1. Biên dịch đoạn chương trình trên.

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010

## 2. Néu lệnh

```
int a = 1;  
được đổi thành
```

```
int a = 10;  
Cho biết giá trị của *pa.
```

## 3. Néu dòng

```
int *pa = &a;  
được sửa lại thành
```

```
int *pa;  
Cho biết kết quả biên dịch chương trình? Chương trình có báo lỗi khi thực thi không? Nếu có, cho  
biết tại sao lỗi.
```

## 4. Néu trước dòng

```
printf("Gia tri a: %d \n", *pa);  
ta thêm dòng code
```

```
*pb = 2;  
Cho biết kết quả của lệnh xuất
```

```
printf("Gia tri a: %d \n", *pa);  
Giải thích tại sao có kết quả xuất như vậy.
```

### 1.5. Con trỏ với mảng (cấp phát mảng động)

Cách làm trước đây khi không sử dụng cấp phát động với mảng 1 chiều

```
int a[100]; // xin 100 ô nhớ cho mảng tối đa 100 phần tử  
int n;  
printf("Nhập số lượng phần tử: ");  
scanf("%d", &n);  
for (int i = 0; i < n; i++)  
{  
    printf("Nhập a[%d]: ", i);  
    scanf("%d", &a[i]);  
}
```

Cách làm này có nhiều hạn chế như: cấp phát thừa trong trường hợp n nhập vào < 100 và không cho phép n nhập vào lớn hơn một số lượng định trước được cài đặt trong code (100).

Để cấp phát mảng động (số lượng phần tử cấp phát đúng bằng với n nhập vào và không giới hạn giá trị n), ta làm như sau

```
int n;  
printf("Nhập số lượng phần tử: ");  
scanf("%d", &n);  
  
//khai bao bien con tro a va xin cap phat vung nho chua n so interger  
int* a = new int[n];  
  
//dung vong lap de nhap cac gia tri a[i]  
for (int i = 0; i < n; i++)  
{  
    Tài liệu hướng dẫn thực hành môn Cấu trúc dữ liệu và giải thuật  
    HCMUS 2010
```

```

        printf("Nhập a[%d]: ", i);
        scanf("%d", &a[i]);
    }
}

```

Sau khai báo

```
int* a = new int[n];

```

một vùng nhớ chứa n số nguyên sẽ được cấp phát, con trỏ a sẽ chỉ đến phần tử đầu tiên của dãy n số.

Các phần tử của mảng được truy xuất qua toán tử [] như với mảng trước đây đã dùng.

### Bài tập (code mẫu: ConTroVoiMang)

Nhập mảng một chiều các số nguyên dùng cấp phát mảng động.

```
#include <stdio.h>

void main()
{
// MANG 1 CHIEU

    int n;
    printf("Nhập số lượng phần tử: ");
    scanf("%d", &n);

    //khai bao bien con tro a va xin cap phat vung nho chua n so interger
    int* a = new int[n];

    //dung vong lap de nhap cac gia tri a[i]
    for (int i = 0; i < n; i++)
    {
        printf("Nhập a[%d]: ", i);
        scanf("%d", &a[i]);
    }

    printf("a[0] = %d\n", a[0]);
    printf("*a = %d\n", *a);

    //xuat cac gia tri a[i]
    for (int i = 0; i < n; i++)
    {
        printf("a[%d] = %d\n", i, a[i]);
    }
}
```

- Biên dịch đoạn chương trình trên.
  - Nhập vào một vài mảng số nguyên, nhận xét về kết quả của 2 lệnh xuất sau các lần chạy.
- ```

printf("a[0] = %d\n", a[0]);
printf("*a = %d\n", *a);

```
- Giải thích tại sao có thể rút ra kết luận ở câu 2.
  - Sửa lại đoạn chương trình trên để nhập vào một mảng số nguyên và xuất ra tổng các số trong mảng đó.
  - Viết chương trình cho phép nhập vào một mảng 2 chiều các số nguyên dùng cấp phát động.
- Gợi ý:

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010

Mảng 2 chiều mxn các số nguyên được khai báo như sau

```
int** b = new int*[m];
```

trong đó mỗi  $b[i]$  (kiểu  $\text{int}^*$ ) là một mảng một chiều n số nguyên

```
b[i] = new int[n];
```

## 2. TRUYỀN THAM SỐ: TRUYỀN THAM BIẾN vs TRUYỀN THAM TRỊ

Giả sử ta có đoạn chương trình:

```
void BinhPhuong(int a)
{
    a = a * a;
}
void main()
{
    int a = 2;
    BinhPhuong(a);
    printf("%d", a);
}
```

với hàm BinhPhuong nhận vào 1 tham số kiểu int và tính bình phương của số đó ( $a = a * a$ ).

Ta mong muốn kết quả xuất ra là 4. Tuy nhiên, thực tế kết quả xuất ra lại là 2.

Giải thích:

Khi gọi thực hiện hàm Funtciton(a), giá trị của a sẽ được truyền cho hàm, không phải là bản thân biến a. Do đó, dù câu lệnh

```
a = a * a;
```

được thực hiện và giá trị của a trong hàm BinhPhuong có thay đổi nhưng do ta chỉ truyền giá trị chứ không phải bản thân biến nên khi ra khỏi hàm BinhPhuong(), giá trị của biến a khi thực hiện câu lệnh in ra vẫn là 2.

Cách truyền tham số a vào hàm BinhPhuong như trên gọi là cách **truyền tham trị** (chỉ truyền giá trị vào hàm, các thao tác làm thay đổi giá trị của biến bên trong hàm không ảnh hưởng đến giá trị biến khi kết thúc hàm).

Tuy nhiên trong trường hợp này, ta muốn những thay đổi giá trị biến a trong hàm BinhPhuong vẫn có tác dụng khi ra khỏi hàm. Ta sửa lại đoạn chương trình trên như sau

```
void BinhPhuong(int& a)
{
    a = a * a;
}
void main()
{
    int a = 2;
    BinhPhuong(a);
    printf("%d", a);
}
```

thì kết quả xuất ra là 4.

trong đó cách khai báo

```
void BinhPhuong(int& a) // dùng toán tử &
cho biết biến a sẽ được truyền theo kiểu tham biến (truyền trực tiếp biến vào hàm, do đó những thay đổi giá trị của biến bên trong hàm sẽ ảnh hưởng đến giá trị biến kể cả khi kết thúc hàm).
```

**Bài tập (code mẫu: ThamBien\_ThamTri)**

```
#include <stdio.h>
```

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010

```

struct DIEM
{
    int x, y;
};

void TruyenThamTri(int a)
{
    a = a * 10;
}

void TruyenThamBien(int &a)
{
    a = a * 10;
}

void ThamTriConTro(DIEM* d)
{
    d->x = d->x * 10;
    d->y = d->y * 10;
}

void ThamBienConTro(DIEM* &d, DIEM* p)
{
    d->x = d->x * 10;
    d->y = d->y * 10;
    d = p;
}

void main()
{
// tham tri, tham bien voi bien du lieu
    int a = 1, b = 10;

    printf("a = %d\n", a);

    TruyenThamTri(a);
    printf("a sau ham TruyenThamTri = %d\n", a);

    TruyenThamBien(a);
    printf("a sau ham TruyenThamBien = %d\n", a);

// bien con tro

    DIEM* d2 = new DIEM;
    d2->x = 5; d2->y = 5;
    printf("Diem d2(%d, %d)\n", d2->x, d2->y);

    ThamTriConTro(d2);
    printf("d2 sau khi goi ham ThamTriConTro: (%d,%d)\n", d2->x, d2->y);
    printf("\n");

    DIEM* d1 = new DIEM;
    d1->x = 5; d1->y = 5;
    d2->x = 5; d2->y = 5;
    printf("Diem d2(%d, %d)\n", d2->x, d2->y);
    ThamBienConTro(d2, d1);
    printf("d2 sau khi goi ham ThamBienConTro: (%d,%d)\n", d2->x, d2->y);

}

```

1. Biên dịch đoạn chương trình trên.

2. Cho biết kết quả của lệnh in

Tài liệu hướng dẫn thực hành môn **Cấu trúc dữ liệu và giải thuật**  
HCMUS 2010

```
    printf("a sau ham TruyenThamTri = %d\n", a);  
và
```

```
    printf("a sau ham TruyenThamBien = %d\n", a);
```

3. Cho biết kết quả của lệnh in

```
    printf("d2 sau khi goi ham ThamTriConTro: (%d,%d)\n", d2->x, d2->y);  
Nhận xét giá trị của d2->x và d2->y có bị thay đổi không? Nếu giá trị của d2->x và d2->y có bị  
thay đổi, giải thích tại sao khi khai báo hàm
```

```
void ThamTriConTro(DIEM* d)
```

bíen d truyền theo kiêu tham trị (không sử dụng &) nhưng d2->x, d2->y lại bị thay đổi

Gợi ý: d là kiêu con trỏ.

4. Cho biết kết quả của lệnh in

```
    printf("d2 sau khi goi ham ThamBienConTro: (%d,%d)\n", d2->x, d2->y);  
Nhận xét giá trị của d2->x và d2->y có bị thay đổi không? Nếu giá trị của d2->x và d2->y không bị  
thay đổi, giải thích tại sao khi khai báo hàm
```

```
void ThamBienConTro(DIEM* &d, DIEM* p)
```

bíen d truyền theo kiêu tham bíen (sử dụng &) nhưng d2->x, d2->y lại không bị thay đổi

Gợi ý

Thay đổi

```
    d1->x = 5;    d1-> y = 5;  
thành giá trị khác rồi quan sát kết quả của lệnh in để rút ra nhận xét.
```

### 3. NHẬP/XUẤT TRÊN TẬP TIN VĂN BẢN

#### 3.1. Nhập

Để đọc một file văn bản, cần thực hiện các bước sau

B1:

```
#include <stdio.h>
```

```
// mo file de doc  
FILE* fi = fopen("input.txt", "rt");
```

trong đó hàm fopen nhận 2 tham số: tham số đầu tiên (kiểu char\*) là tên file (input.txt), tham số thứ 2 là chuỗi “rt” (read + text: đọc dạng file văn bản)

Trong trường hợp muốn viết hàm đọc file và truyền tên file như một tham số vào hàm:

```
void DocFile(char* tenFile)  
{  
    FILE* fi = fopen(tenFile, "rt");  
    ...  
}
```

B2:

```
//doc du lieu  
int n;  
fscanf(fi, "%d", &n);
```

dùng hàm fscanf để đọc dữ liệu. Cách dùng hàm fscanf tương tự hàm scanf (chỉ khác có thêm tham số đầu tiên kiểu FILE\* là con trỏ đến tập tin đã mở ở trên)

B3:

```
//dong file  
fclose(fi);
```

#### 3.2. Xuất

Tương tự với nhập dữ liệu, ta cũng có các bước

B1:

```
//mo file de ghi  
FILE* fo = fopen(tenFile, "wt"); // wt = write (ghi) + text (dạng văn bản)
```

B2:

```
//ghi du lieu ra file  
fprintf(fo, "%d ", n);
```

Dùng hàm fprintf() tương tự printf()

B3:

```
// dong file  
fclose(fo);
```

### 3.3. Cách quản lý tập tin nhập/xuất trong project của VS2005

Khi làm việc với tập tin, ta cần phải đặt tập tin ở đúng thư mục để có thể debug/thực thi chương trình. Ta có thể sử dụng VS2005 để tạo tập tin txt, VS2005 sẽ tự động đặt file đó ở đúng thư mục và ta có thể quản lý file dễ dàng hơn.

Để tạo file txt, từ cửa sổ Solution Explorer, nhấn chuột phải vào Resources Files, chọn Add -> New Item

Chọn Text File (.txt) và gõ tên (không gõ đuôi .txt)

Sau đó gõ nội dung tập tin.

Với các file mà ta xuất kết quả ra, ta cũng có thể thêm file đó vào project để dễ quản lý.

Cách làm tương tự, nhấn chuột phải vào Resources Files, chọn Add -> Existing Item

Chọn file mà ta đã xuất ra.

(Xem Demo)

#### Bài tập (code mẫu: NhapXuatFile)

Đọc từ file “input.txt” mảng một chiều các số thực. Tập tin input.txt có nội dung như sau:

- Dòng đầu chứa 1 số nguyên là số lượng phần tử của mảng
- Dòng sau chứa các phần tử của mảng cách nhau bởi khoảng trắng

#### Ví dụ:

```
5
1.2 2.3 3.4 4.5 5.6
```

```
#include <stdio.h>

void XuatFile(char* tenFile, float* arr, int n)
{
    //mo file de ghi
    FILE* fo = fopen(tenFile, "wt"); // wt = write (ghi) + text (dang van ban)

    //ghi du lieu ra file
    for (int i = 0; i < n; i++)
        fprintf(fo, "%0.1f ", arr[i]);

    // dong file
    fclose(fo);
```

```

}

void main()
{
    // mo file de doc
    FILE* fi = fopen("input.txt", "rt");

    //doc du lieu
    int n;
    fscanf(fi, "%d", &n);
    float* arr = new float[n];
    for (int i = 0; i < n; i++)
        fscanf(fi, "%f", &arr[i]);

    //dong file
    fclose(fi);

    // in ra man hinh de kiem tra
    for (int i = 0; i < n; i++)
        printf("%0.1f ", arr[i]);
    printf("\n");

    // xuat ra file
    XuatFile("output.txt", arr, n);
}

```

1. Biên dịch đoạn chương trình trên.
2. Tạo tập tin dữ liệu mới “MSSV.txt” thay cho file “input.txt”. Nhập dữ liệu cho file MSSV.txt và chạy chương trình.
3. Xuất ra file “out\_MSSV.txt” thay cho file “output.txt”. Thêm file output.txt vào project để có thể xem kết quả xuất từ Visual Studio thay vì phải dùng Windows Explorer và Notepad.
3. Sửa lại chương trình để chỉ xuất ra file các tập tin có chỉ số lẻ của mảng. (chỉ in ra arr[1], arr[3], ...)
4. Sửa lại chương trình để tính tổng các phần tử và xuất ra file tổng đó.

#### **3.4. Đọc đến hết file**

Nếu bài toán đọc mảng các số thực ở trên không có thông tin số lượng phần tử thì ta sẽ giải quyết theo hướng cứ đọc vào đến khi nào hết file thì dừng. Vậy ta cần phải biết dấu hiệu kết thúc file: hàm feof()

Xem đoạn chương trình mẫu sau: đọc một mảng không biết trước số lượng các số thực và in ra màn hình.

```

void DocHetFile1(char* tenFile)
{
    FILE* fi = fopen(tenFile, "rt");
    float temp;
    while (!feof(fi)) // khi chua ket thuc file
    {
        fscanf(fi, "%f", &temp); // doc so thuc vao bien temp
        printf("%0.1f ", temp);
    }
}

```

```

        printf("\n");
        fclose(fi);
    }

```

Trong đó, ta sử dụng vòng while và điều kiện để thực hiện là chưa hết file: !feof(fi)

Hàm feof() nhận 1 tham số kiểu FILE\* là file đang đọc và trả về true/false nếu kết thúc/chưa kết thúc file.

### Bài tập (code mẫu: NhậpXuấtFile)

```

#include <stdio.h>

void DocHetFile1(char* tenFile)
{
    FILE* fi = fopen(tenFile, "rt");
    float temp;
    while (!feof(fi))
    {
        fscanf(fi, "%f", &temp);
        printf("%0.1f ", temp);
    }
    printf("\n");
    fclose(fi);
}

void DocHetFile2(char* tenFile)
{
    FILE* fi = fopen(tenFile, "rt");
    float temp;
    while (!feof(fi))
    {
        if (fscanf(fi, "%f", &temp) > 0)
            printf("%0.1f ", temp);
        else
            break;
    }
    printf("\n");
    fclose(fi);
}

void main()
{
    printf("Đọc khi không biết số lượng phần tử, đọc đến hết file thì
dừng:\n");

    DocHetFile1("input2.txt");
    DocHetFile1("input3.txt");
}

```

Trong đó

input2.txt có nội dung

```
1.2 2.3 3.4 4.5 5.6
```

input3.txt có nội dung (có 1 dấu khoảng trắng ở cuối file)

1. Biên dịch đoạn chương trình trên.
2. Nhận xét về kết quả xuất ra màn hình (2 dòng tương ứng với 2 file input2.txt và input3.txt). 2 dòng kết quả xuất ra có giống nhau không?
3. Nếu 2 dòng kết quả xuất ra không giống nhau, giải thích tại sao với dữ liệu vào như nhau (xem nội dung file input2.txt và input3.txt) kết quả xuất lại không giống nhau.

Gợi ý: file input3.txt có một dấu khoảng trắng ở cuối file nên ở vòng lặp cuối sau khi đọc 5.6 điều kiện (!feof(fi)) vẫn trả về đúng (vì hết số 5.6 chưa là cuối file).

4. Sửa lại 2 dòng

```
DocHetFile1("input2.txt");
DocHetFile1("input3.txt");
```

thành

```
DocHetFile2("input2.txt");
DocHetFile2("input3.txt");
```

và chạy lại chương trình. Nhận xét kết quả xuất. Từ đó rút ra kết luận hàm nào đọc hết file chính xác hơn.

Gợi ý:

Hàm DocHetFile2 và DocHetFile1 chỉ khác nhau ở chỗ hàm DocHetFile2 có kiểm tra lệnh đọc số thực từ file có thành công hay không như sau:

Hàm fscanf() có giá trị trả về là số lượng biến đọc thành công

```
if (fscanf(fi, "%f", &temp) > 0)
```

Ở đây ta đọc 1 biến, do đó chỉ cần kiểm tra giá trị trả về  $> 0$  là đọc thành công.

#### 4. ĐỆ QUY

Một hàm được gọi là đệ quy nếu bên trong thân của hàm đó có lời gọi hàm lại chính nó một cách tường minh hay tiềm ẩn

Khi viết hàm đệ quy, cần xác định:

- Điều kiện dừng
- Công thức đệ quy

Ví dụ:

Tính tổng  $S(n) = 1 + 2 + \dots + n$

Ta có

$$S(n) = (1 + 2 + \dots + n-1) + n$$

hay  $S(n) = S(n-1) + n$  (công thức đệ quy)

$$S(0) = 0 \quad (\text{điều kiện dừng})$$

Ta có chương trình tương ứng với công thức đệ quy trên như sau:

```
int Tong(int n )  
{  
    if (n == 0)  
        return 0;  
    return Tong(n-1) + n;  
}
```

#### Bài tập (code mẫu: DeQuy)

```
#include <stdio.h>  
  
int Tong(int* a, int n)  
{  
    if ( n == 0)  

```

```
    printf("%d\n", Tong(a, n-1));  
}
```

1. Biên dịch đoạn chương trình trên.
2. Cho biết đoạn chương trình trên thực hiện tác vụ gì.
3. Viết công thức đệ quy và điều kiện dừng của chương trình đệ quy trên. Nếu chương trình đệ quy không có điều kiện dừng thì điều gì sẽ xảy ra?
4. Sửa lại chương trình để tính tích của các phần tử của một dãy số nguyên bằng phương pháp đệ quy
5. Sửa lại chương trình để tính tổng các số lẻ có trong mảng bằng phương pháp đệ quy

# DANH SÁCH LIÊN KẾT

## MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu được các thành phần của danh sách liên kết.
- Thành thạo các thao tác trên danh sách liên kết: thêm phần tử, xóa phần tử, duyệt danh sách liên kết.
- Áp dụng cấu trúc dữ liệu danh sách liên kết vào việc giải quyết một số bài toán đơn giản.

Thời gian thực hành: **từ 120 phút đến 400 phút**

## TÓM TẮT

Danh sách liên kết là cấu trúc dữ liệu dùng để lưu trữ một danh sách (tập hợp hữu hạn) dữ liệu. Điểm đặc biệt của cấu trúc này là khả năng chứa của nó **động** (có thể mở rộng và thu hẹp dễ dàng).

Có các loại danh sách liên kết:

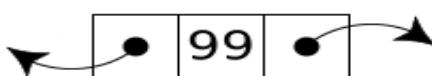
- Danh sách liên kết đơn
- Danh sách liên kết kép
- Danh sách liên kết vòng

Mỗi danh sách liên kết là tập hợp các phần tử (node) chứa thông tin lưu trữ của dữ liệu. Giữa các phần tử có một hoặc nhiều liên kết để đảm bảo danh sách liên kết có thể giữ các phần tử này một cách chặt chẽ.

Ví dụ 1:



Phần tử có một liên kết

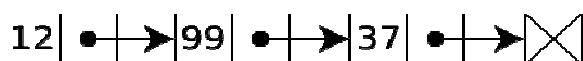


Phần tử có hai liên kết

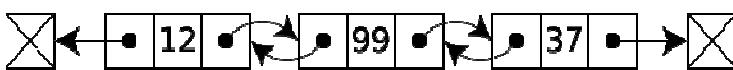


Phần tử rỗng

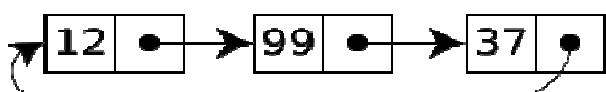
Ví dụ 2:



Danh sách liên kết đơn



Danh sách liên kết kép



Danh sách liên kết vòng

Trong mỗi phần tử của danh sách liên kết, thông tin liên kết là vô cùng quan trọng. Chỉ cần một xử lý không cẩn thận có thể làm mất phần liên kết này thì danh sách liên kết sẽ bị ‘gãy’ từ phần tử đó (không thể truy xuất tiếp các phần tử từ phần tử đó trở về trước hoặc trở về sau).

Các thao tác cơ bản trên danh sách liên kết:

- Thêm phần tử: vào đầu danh sách liên kết, vào cuối danh sách liên kết, vào trước/sau một phần tử trên danh sách liên kết.
- Xóa phần tử: ở đầu danh sách liên kết, ở cuối danh sách liên kết, một phần tử trên danh sách liên kết.
- Duyệt danh sách liên kết: để có thể đi được hết các phần tử trên danh sách liên kết.

# NỘI DUNG THỰC HÀNH

## Cơ bản

Sinh viên đọc kỹ phát biểu bài tập và thực hiện theo hướng dẫn:

*Tổ chức một danh sách liên kết đơn trong đó mỗi phần tử chứa thông tin dữ liệu nguyên.*

*Người dùng sẽ nhập các giá trị nguyên từ bàn phím. Với mỗi giá trị nguyên được nhập vào, giá trị đó được thêm vào **phía đầu** của danh sách liên kết. Nếu người dùng nhập vào giá trị **-1**, quá trình nhập dữ liệu sẽ kết thúc.*

*Sau đó, in ra các phần tử đang có trên danh sách liên kết.*

*Khi chương trình kết thúc, tất cả các phần tử trên danh sách liên kết bị **xóa bỏ** khỏi bộ nhớ.*

## Phân tích

- Danh sách liên kết đơn gồm mỗi phần tử chứa dữ liệu nguyên. Thông tin của mỗi phần tử được khai báo theo ngôn ngữ C/C++ như sau:

```
struct NODE{
    int Key;
    NODE *pNext;
};
```

- Danh sách liên kết được khai báo như sau:

```
struct LIST{
    NODE *pHead;
    NODE *pTail;
};
```

- Thao tác cần thực hiện: **thêm** phần tử nguyên vào **đầu** danh sách liên kết (**AddHead**), **in** các phần tử của danh sách liên kết (**PrintList**), **loại bỏ tất cả** các phần tử trên danh sách liên kết (**RemoveAll**).

## Chương trình mẫu

```
#include "stdafx.h"

struct NODE{
    int Key;
    NODE *pNext;
};

struct LIST{
    NODE *pHead;
    NODE *pTail;
};

NODE* CreateNode(int Data)
{
    NODE* pNode;
    pNode = new NODE; //Xin cấp phát bộ nhớ động để tạo một phần tử (node) mới
    if (pNode == NULL)
        return NULL;
    pNode->Key = Data;
    pNode->pNext = NULL;
    return pNode;
}

bool AddHead(LIST &L, int Data)
{
```

```

NODE *pNode;
pNode = CreateNode(Data);
if (pNode == NULL)
    return false;
if (L.pHead == NULL)
{
    L.pHead = pNode;
    L.pTail = pNode;
}
else {
    pNode->pNext = L.pHead;
    L.pHead = pNode;
}
return true;
}

void PrintList(LIST L)
{
    NODE *pNode;
    pNode = L.pHead;
    while (pNode != NULL)
    {
        printf("%5d", pNode->Key);
        pNode = pNode->pNext; //Ghi chú: thao tác này dùng để làm gì?
    }
}

void RemoveAll(LIST &L) //Ghi chú: Ý nghĩa của ký hiệu &
{
    NODE *pNode;
    while (L.pHead != NULL)
    {
        pNode = L.pHead;
        L.pHead = L.pHead->pNext;
        delete pNode;
    }
    L.pHead = NULL; //Ghi chú: Tại sao phải thực hiện phép gán này?
    L.pTail = NULL;
}

int _tmain(int argc, _TCHAR* argv[])
{
    LIST L;
    //Ghi chú: Tại sao lại phải thực hiện phép gán phía dưới?
    L.pHead = NULL;
    L.pTail = NULL;
    int Data;
    do
    {
        printf("Nhập vào dữ liệu, -1 để kết thúc: ");
        scanf("%d", &Data);
        if (Data == -1)
            break;
        AddHead(L, Data);
    }while (Data != -1);
    printf("\nDữ liệu đã được nhập: \n");

    //Ghi chú: Chức năng của dòng lệnh phía dưới
    PrintList(L);

    //Ghi chú: Chức năng của dòng lệnh phía dưới
    RemoveAll(L);
}

```

```

        return 0;
}

```

### **Yêu cầu**

1. Biên dịch đoạn chương trình nêu trên.
2. Cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:  

```

-1
5      -1
7      10     -23    -25    -4      1      -1
1      2      3      4      -1

```
3. Nếu nhận xét ngắn gọn mối liên hệ giữa thứ tự nhập dữ liệu vào với thứ tự in dữ liệu ra màn hình.
4. Vẽ hình danh sách liên kết theo dữ liệu được nhập ở câu 2.
5. Nếu trong hàm main (\_tmain) thứ tự hai dòng lệnh sau đây bị hoán đổi cho nhau thì kết quả xuất ra màn hình sẽ như thế nào đối với dữ liệu câu 2? Giải thích lý do?

```

PrintList(L);
RemoveAll(L);

```

6. Nếu trong hàm main (\_tmain) vòng lặp do...while được thay đổi như dưới đây thì kết quả xuất ra màn hình sẽ như thế nào đối với dữ liệu câu 2? Giải thích lý do?

```

do
{
    printf("Nhập vào dữ liệu, -1 để kết thúc: ");
    scanf("%d", &Data);
    AddHead(L, Data);
    if (Data == -1)
        break;
} while (Data != -1);

```

7. Với các hàm CreateNode, AddHead được cung cấp sẵn, hãy cho biết ý nghĩa của các giá trị trả về của hàm.
8. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú (*//Ghi chú*) trong các hàm RemoveAll, PrintList, \_tmain.
9. Kết quả sẽ như thế nào nếu hàm RemoveAll được thay đổi như dưới đây? Giải thích lý do

```

void RemoveAll(LIST &L)
{
    while (L.pHead != NULL)
    {
        L.pHead = L.pHead->pNext;
        delete L.pHead;
    }
    L.pHead = NULL;
}

```

10. Giá trị cuối cùng của biến pRoot trong đoạn chương trình mẫu là gì? Giải thích lý do.

### **Áp dụng – Nâng cao**

1. Bổ sung chương trình mẫu cho phép tính **tổng giá trị** các phần tử trên danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm PrintList để viết hàm **SumList**.

2. Bổ sung chương trình mẫu cho phép tìm **giá trị nguyên lớn nhất** trong số các phần tử nguyên trên danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **PrintList** để viết hàm **MaxList**.

3. Bổ sung chương trình mẫu cho phép tính **số lượng các phần tử** của danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **PrintList** để viết hàm **CountList**.

4. Bổ sung chương trình mẫu cho phép **thêm vào cuối** danh sách liên kết đơn một giá trị nguyên.

Gợi ý: tham khảo hàm **AddHead** để viết hàm **AddTail**.

5. Bổ sung chương trình mẫu cho phép **xóa phần tử đầu** danh sách liên kết đơn.

6. Bổ sung chương trình mẫu cho phép **xóa phần tử cuối** danh sách liên kết đơn.

7. Bổ sung chương trình mẫu cho biết **số lượng các phần tử** trên danh sách liên kết đơn có giá trị trùng với giá trị  $x$  được cho trước.

Gợi ý: tham khảo thao tác duyệt danh sách liên kết trong hàm **PrintList**.

8. Bổ sung chương trình mẫu cho phép tạo một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên trong đó không có cặp phần tử nào mang giá trị giống nhau.

Gợi ý: sử dụng hàm **AddHead** hoặc **AddTail** có bổ sung thao tác kiểm tra phần tử giống nhau.

9. Cho sẵn một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên và một giá trị nguyên  $x$ . Hãy tách danh sách liên kết đã cho thành 2 danh sách liên kết: một danh sách gồm các phần tử có giá trị nhỏ hơn giá trị  $x$  và một danh sách gồm các phần tử có giá trị lớn hơn giá trị  $x$ . Giải quyết trong 2 trường hợp:

- Danh sách liên kết ban đầu không cần tồn tại.
- Danh sách liên kết ban đầu bắt buộc phải tồn tại.

## BÀI TẬP THÊM

1. Đề xuất cấu trúc dữ liệu thích hợp để biểu diễn đa thức  $(a_nx^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0)$  bằng danh sách liên kết (đơn hoặc kép). Cài đặt các thao tác trên danh sách liên kết đơn biểu diễn đa thức:

- In đa thức
- Rút gọn đa thức
- Cộng hai đa thức
- Nhân hai đa thức

2. Thông tin của một quyển sách trong thư viện gồm các thông tin:

- Tên sách (chuỗi)
  - Tác giả (chuỗi, tối đa 5 tác giả)
  - Nhà xuất bản (chuỗi)
  - Năm xuất bản (số nguyên)
- Hãy tạo danh sách liên kết (đơn hoặc kép) chứa thông tin các quyển sách có trong thư viện (được nhập từ bàn phím).
  - Cho biết số lượng các quyển sách của một tác giả bất kỳ (nhập từ bàn phím).
  - Trong năm YYYY (nhập từ bàn phím), nhà xuất bản ABC (nhập từ bàn phím) đã phát hành những quyển sách nào.

# STACK và QUEUE

## MỤC TIÊU

Hoàn tất phần thực hành này, sinh viên có thể:

- Hiểu được cách thức sử dụng stack và queue trên cơ sở sử dụng danh sách liên kết để cài đặt.
- Hiểu và vận dụng các cấu trúc stack và queue trong những bài toán đơn giản.

Thời gian thực hành: 120 phút đến 360 phút.

Lưu ý: yêu cầu vận dụng thành thạo danh sách liên kết ở Lab02.

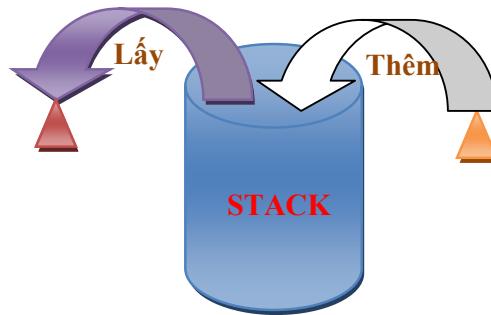
## TÓM TẮT

- Stack (ngăn xếp) và queue (hang đợi) là những cấu trúc dữ liệu dùng để lưu trữ các phần tử của tập hợp theo những nguyên tắc đặc biệt khi thêm phần tử cũng như lấy phần tử ra khỏi cấu trúc.
- Stack (last in, first out – LIFO): phần tử vào stack sau cùng, là phần tử được lấy ra khỏi stack trước nhất.
- Queue (first in, first out – FIFO): phần tử vào queue trước nhất, là phần tử được lấy ra khỏi queue trước nhất.

© Lab03 là phần vận dụng danh sách liên kết đã thực hành ở Lab02 để cài đặt Stack và Queue.

(Lưu ý: chúng ta cũng có thể dùng mảng để cài đặt stack và queue, nhưng mảng đặc trưng cho cơ chế tĩnh, do vậy danh sách liên kết – cơ chế động – là cấu trúc tốt hơn mảng khi hiện thực Stack và Queue).

Ví dụ: minh họa Stack



+ Phần tử mới được thêm vào đỉnh của ngăn xếp.

+ Thao tác lấy phần tử ra khỏi ngăn xếp, nếu ngăn xếp khác rỗng thì phần tử ở đầu ngăn xếp được lấy ra, ngược lại, ngăn xếp rỗng thì thao tác lấy phần tử thất bại.

Ví dụ: minh họa Queue



+ Phần tử được thêm vào ở đầu queue. Do vậy, phần tử vào đầu tiên sẽ ở đáy của queue. Do vậy, khi lấy phần tử ra, nếu queue khác rỗng thì phần tử ở đáy queue được lấy ra, ngược lại, queue bị rỗng thì thao tác lấy phần tử ra khỏi queue thất bại.

## NỘI DUNG THỰC HÀNH

### Cơ bản

Yêu cầu: cài đặt stack và queue bằng danh sách liên kết.

Do đặc trưng của stack và queue, chúng ta cần xây dựng 2 thao tác chính là thêm 1 phần tử vào stack hoặc queue, và lấy 1 phần tử ra khỏi stack hoặc queue.

Dựa vào nguyên tắc thêm và lấy phần tử ra khỏi stack/queue, ta cần xây dựng các hàm sau:

- Đối với Stack
  - o Thêm phần tử: thêm phần tử vào đầu danh sách liên kết.
  - o Lấy phần tử: lấy phần tử ở đầu danh sách ra khỏi danh sách liên kết.

(Lưu ý: ta cũng có thể thêm phần tử vào cuối danh sách liên kết, do vậy thao tác lấy phần tử, ta thực hiện lấy phần tử ở cuối danh sách liên kết).

- Đối với Queue
  - o Thêm phần tử: thêm vào đầu danh sách liên kết.
  - o Lấy phần tử: lấy phần tử ở cuối danh sách liên kết.

(Lưu ý: ta cũng có thể thực hiện việc thêm phần tử vào cuối danh sách liên kết và lấy ra ở đầu danh sách liên kết).

Sử dụng bài tập Lab02

### Chương trình mẫu

```
#include <stdio.h>

struct NODE{
    int Key;
    NODE *pNext;
};

NODE* CreateNode(int Data)
{
    NODE* pNode;
    pNode = new NODE;
    if (pNode == NULL)
        return NULL;
    pNode->Key = Data;
    pNode->pNext = NULL;
    return pNode;
}

bool AddHead(NODE* &pHead, int Data)
{
    NODE *pNode;
    pNode = CreateNode(Data);
    if (pNode == NULL)
        return false;
    if (pHead == NULL)
        pHead = pNode;
    else {
        pNode->pNext = pHead;
        pHead = pNode;
    }
}
```

```

        }
        return true;
    }

NODE* RemoveHead(NODE* &pHead)
{
    if(pHead == NULL)
        return NULL;
    NODE* pResult = pHead;
    pHead = pHead->pNext;
    return pResult;
}

NODE* RemoveTail(NODE* &pHead)
{
    NODE *pNode;
    if(pHead == NULL) //<1>
    {
        return NULL;
    }
    else if(pHead->pNext == NULL) //<2>
    {
        pNode = pHead;
        pHead = NULL;
        return pNode;
    }

    pNode = pHead;
    while(pNode->pNext->pNext != NULL) //<3>
    {
        pNode = pNode->pNext;
    }

    NODE* pResult = pNode->pNext;
    pNode->pNext = NULL;
    return pResult;
}

//-----STACK :
//----PUSH tương ứng AddHead
//----POP tương ứng RemoveHead

bool PushStack(NODE* &pStack, int Data)
{
    return AddHead(pStack, Data);
}

NODE* PopStack(NODE* &pStack)
{
    return RemoveHead(pStack);
}

//-----QUEUE :
//----ENQUEUE tương ứng AddHead
//----DEQUEUE tương ứng RemoveTail
bool EnQueue(NODE* &pQueue, int Data)
{
    return AddHead(pQueue, Data);
}

NODE* DeQueue(NODE* &pQueue)
{
    return RemoveTail(pQueue);
}

```

```

void main()
{
    NODE* pStack = NULL;
    NODE* pQueue = NULL;

    int n = 10;
    while(n!=0)
    {
        PushStack(pStack, n);
        EnQueue(pQueue, n);
        n--;
    }

    NODE* pNode = DeQueue(pQueue);
    if(pNode != NULL)           //<4>
        printf("\nGia tri phan tu (Queue) : %d\n", pNode->Key);
    else
        printf("\nNULL\n");

    NODE* pNode2 = PopStack(pStack);
    if(pNode2 != NULL)
        printf("\nGia tri phan tu (Stack) : %d\n", pNode2->Key);
    else
        printf("\nNULL\n");
}

```

1. Biên dịch đoạn chương trình trên.
2. Thay giá trị n=10 thành n=1 trong main, đọc giá trị Queue và Stack in ra màn hình.
3. Giải thích <4> khi nào giá trị pNode khác NULL, khi nào pNode bằng NULL, ý nghĩa của mỗi trường hợp trên.
4. Giải thích hàm RemoveTail ở các điểm <1>, <2>, <3> cho biết ý nghĩa của chúng.
5. Sử dụng các hàm PushStack, PopStack, EnQueue, DeQueue để cài đặt.
  - a. Về Stack: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lệnh thêm phần tử vào stack), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi stack, nếu có, in giá trị phần tử ra màn hình, nếu không có (stack rỗng), in ra màn hình “STACK RONG, KHONG LAY DUOC PHAN TU”.
  - b. Về Queue: Trong hàm main, thực hiện việc thêm vào 3 giá trị do người dùng nhập vào (thực hiện 3 lần lệnh thêm phần tử vào queue), sau đó thực hiện 4 lần lệnh lấy giá trị phần tử ra khỏi queue, nếu có, in giá trị phần tử ra màn hình, nếu không có (queue rỗng), in ra màn hình “QUEUE RONG, KHONG LAY DUOC PHAN TU”.

## Áp dụng – Nâng cao

1. Cài đặt hàm AddTail để có phiên bản cài đặt Stack (thêm phần tử vào cuối danh sách và lấy phần tử ở cuối danh sách liên kết) cũng như áp dụng 1 phiên bản khác khi cài đặt Queue (thêm phần tử vào cuối danh sách liên kết và lấy phần tử ở đầu danh sách liên kết).
2. Nhận xét cách cài đặt trên ở phần 1 (áp dụng – nâng cao) so với chương trình mẫu đối với trường hợp stack cũng như queue.
3. Sử dụng cấu trúc Stack để chuyển giá trị từ cơ số 10 sang cơ số 2.  
Gợi ý : thực hiện việc chia liên tiếp giá trị trong cơ số 10 cho 2, lấy phần dư đưa vào stack, cho đến khi giá trị đem đi chia là 0. In giá trị trong stack ra (đó chính là kết quả khi chuyển số từ hệ cơ số 10 sang hệ cơ số 2).

## BÀI TẬP THÊM

Tìm đường trong mê cung (thực hiện loang theo chiều rộng <sử dụng queue> hoặc loang theo chiều sâu <sử dụng stack>).

Bài toán: cho ma trận mxn, mỗi phần tử là số 0 hoặc 1.

Giá trị 1 : có thể đi tới và giá trị 0 : không thể đi tới được.

Câu hỏi:

Từ ô ban đầu có tọa độ  $(x_1, y_1)$  có thể đi tới ô  $(x_2, y_2)$  không?

Biết rằng từ 1 ô  $(x, y)$  chỉ có thể đi qua ô có chung cạnh với ô đang đứng và mang giá trị là 1, ngược lại không có đường đi.

# CÂY NHỊ PHÂN TÌM KIẾM

## MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu được các thành phần của cây nhị phân tìm kiếm.
- Thành thạo các thao tác trên cây nhị phân tìm kiếm: tạo cây, thêm phần tử, xóa phần tử, duyệt cây nhị phân tìm kiếm.
- Áp dụng cấu trúc dữ liệu cây nhị phân tìm kiếm vào việc giải quyết một số bài toán đơn giản.

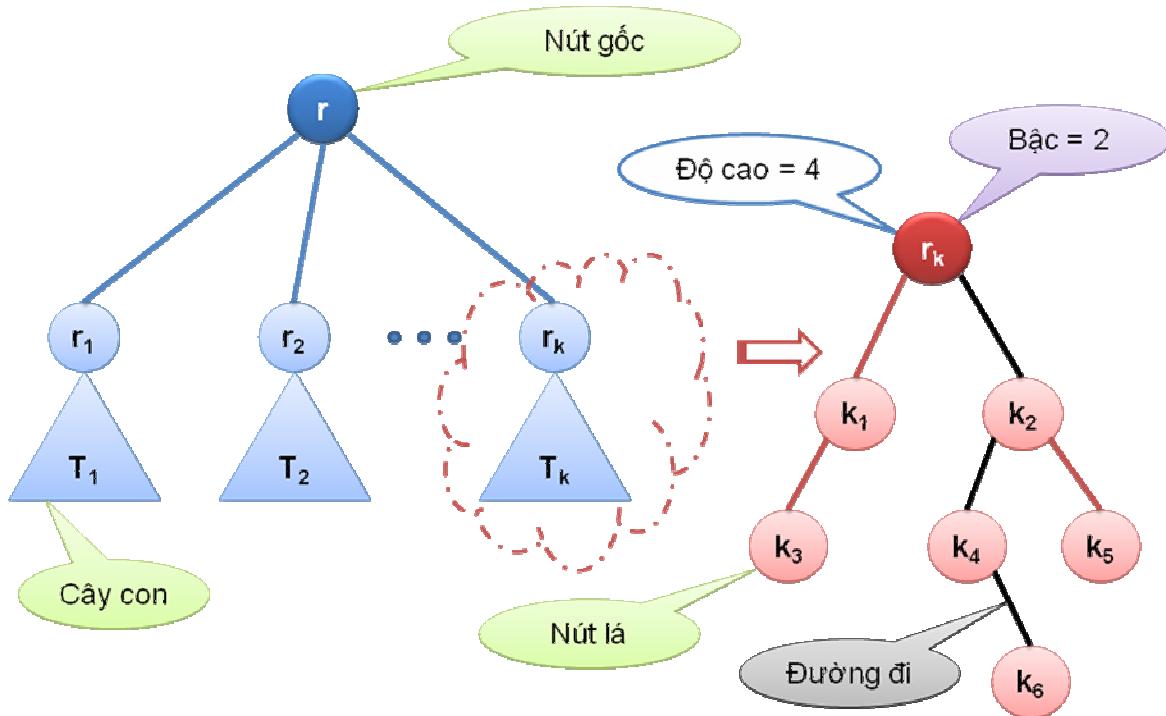
Thời gian thực hành: **từ 120 phút đến 400 phút**

## TÓM TẮT

Cây nhị phân tìm kiếm là cây có tối đa 2 nhánh (cây con), nhánh trái và nhánh phải. Cây nhị phân tìm kiếm có các tính chất sau:

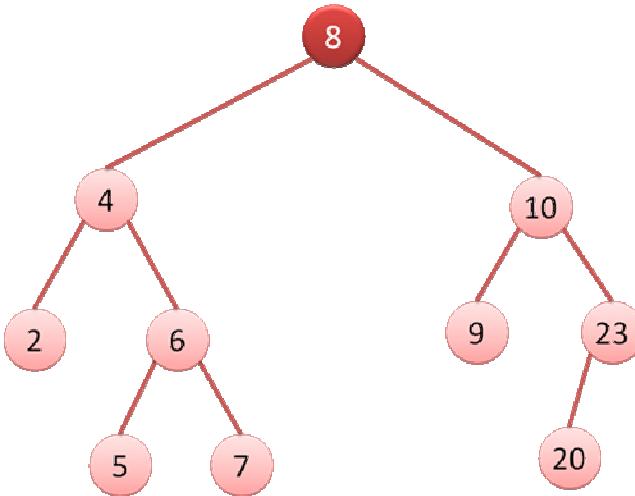
- Khóa của tất cả các nút thuộc cây con trái nhỏ hơn khóa nút gốc.
- Khóa của nút gốc nhỏ hơn khóa của tất cả các nút thuộc cây con phải.
- Cây con trái và cây con phải của nút gốc cũng là cây nhị phân tìm kiếm

Một số khái niệm:



- Nút lá có độ cao bằng 1

Ví dụ cây nhị phân tìm kiếm:



Trong mỗi nút của cây nhị phân tìm kiếm, thông tin liên kết là vô cùng quan trọng. Chỉ cần một xử lý không cẩn thận có thể làm mất phần liên kết này thì cây sẽ bị ‘gãy’ cây con liên quan ứng với liên kết đó (không thể truy xuất tiếp tất cả các nút của nhánh con bị mất).

Các thao tác cơ bản trên cây nhị phân tìm kiếm:

- Thêm 1 nút: dựa vào tính chất của cây nhị phân tìm kiếm để tìm vị trí thêm nút mới.
  - o Tạo cây: từ cây rỗng, lần lượt thêm các nút vào cây bằng phương thức thêm nút vào cây nhị phân tìm kiếm
- Xóa 1 nút: là nút lá, là nút có 1 nhánh con, là nút có 2 nhánh con.
- Duyệt cây nhị phân tìm kiếm: để có thể đi được hết các phần tử trên cây nhị phân tìm kiếm: duyệt trước (NLR), duyệt giữa (LNR), duyệt sau (LRN). Do tính chất của cây nhị phân tìm kiếm, phép duyệt giữa cho phép duyệt các khóa của cây theo thứ tự tăng dần

## NỘI DUNG THỰC HÀNH

### Cơ bản

Sinh viên đọc kỹ phát biểu bài tập và thực hiện theo hướng dẫn:

Tổ chức một cây nhị phân tìm kiếm trong đó mỗi phần tử chứa thông tin dữ liệu là số nguyên.

Người dùng sẽ nhập các giá trị nguyên từ bàn phím. Với mỗi giá trị nguyên được nhập vào, giá trị đó được thêm vào cây nhị phân tìm kiếm mà vẫn đảm bảo cây sau khi thêm vẫn là cây nhị phân tìm kiếm. Nếu người dùng nhập vào giá trị -1, quá trình nhập dữ liệu sẽ kết thúc. Cây ban đầu là cây rỗng (chưa có nút nào)

Sau đó, in ra các phần tử đang có trên cây bằng phương pháp duyệt trước.

Cho người dùng nhập vào 1 giá trị nguyên từ bàn phím, cho biết giá trị này có trong cây hay không. Nếu có, cho biết nút đó có độ cao bao nhiêu. Sau đó, xóa nút khỏi cây, xuất cây sau khi xóa bằng phương pháp duyệt trước

### Phân tích

- Cây nhị phân tìm kiếm có mỗi nút chứa dữ liệu nguyên. Thông tin của mỗi nút được khai báo theo ngôn ngữ C/C++ như sau:

```
struct NODE{  
    int Key;  
    NODE *pLeft;  
    NODE *pRight;  
};
```

- Thao tác cần thực hiện:
  - o Khai báo, khởi tạo cây
  - o (lặp) **thêm** nút có khóa nguyên vào **cây nhị phân tìm kiếm (Insert)**,
  - o **in** các nút của cây nhị phân tìm kiếm (**NLR**),
  - o **tìm** 1 giá trị, nếu có:
    - tính **độ cao** của nút đó (**Height**)
    - **xóa** nút khỏi cây (**RemoveNode**)
    - **in** các nút của cây sau khi xóa (**NLR**)

### Chương trình mẫu

```
#include "stdio.h"  
  
struct NODE{  
    int Key;  
    NODE *pLeft;  
    NODE *pRight;  
};  
  
void Init (NODE *&TREE)  
{  
    TREE = NULL;  
}  
  
void Insert (NODE *&pRoot, int x)  
{  
    if (pRoot == NULL)
```

```

{
    NODE *q;
    q = new NODE;
    q->Key = x;
    q->pLeft = q->pRight = NULL;
    pRoot = q;
}
else
{
    if (x < pRoot->Key)
        Insert (pRoot->pLeft, x);
    else if (x > pRoot->Key)
        Insert (pRoot->pRight, x);
}
}

void CreateTree(NODE *&pRoot)
{
    int Data;
    do{
        printf("Nhập vào địa điểm, -1 để kết thúc: ");
        scanf("%d", &Data);
        if (Data == -1)
            break;
        Insert(pRoot, Data);
    } while(1);
}

void NLR(NODE* pTree)
{
    if(pTree != NULL)
    {
        printf("%4d", pTree->Key);
        NLR(pTree->pLeft);
        NLR(pTree->pRight);
    }
}

NODE* Search(NODE* pRoot, int x)
{
    if(pRoot == NULL)
        return NULL;
    if(x < pRoot->Key)
        Search(pRoot->pLeft, x);
    else
        if(x > pRoot->Key)
            Search(pRoot->pRight, x);
        else
    {
        //Ghi chú: Trong trường hợp nào dòng bên dưới được thực hiện?
        return pRoot;
    }
}

int Height(NODE* pNode)
{
    if(pNode == NULL)
        return 0;
    int HL, HR;
    HL = Height(pNode->pLeft);
    HR = Height(pNode->pRight);
    if(HL > HR)
        return (1 + HL);
    return (1 + HR);
}

```

```

}

void SearchStandFor(NODE* &Tree, NODE* &q)
{
    if (Tree->pRight)
        SearchStandFor(Tree->pRight, q);
    else
    {
        q->Key = Tree->Key;
        q = Tree;
        Tree = Tree->pLeft;
    }
}

void RemoveNode(NODE* &Tree, int x)
{
    NODE* p;
    if(Tree == NULL)
        printf("%d khong co trong cay", x);
    else
    {
        if (x < Tree->Key)
            RemoveNode(Tree->pLeft, x);
        else
            if (x > Tree->Key)
                RemoveNode(Tree->pRight, x);
            else
            {
                //Ghi chú: Mục đích phép gán này là gì?
                p = Tree;
                if(p->pRight == NULL)
                    Tree = p->pLeft;
                else
                    if (p->pLeft == NULL)
                        Tree = p->pRight;
                    else {
                        //Ghi chú: Hàm bên dưới dùng để làm gì?
                        SearchStandFor(Tree->pLeft, p);
                    }
                delete p;
            }
    }
}

void main()
{
    NODE* pTree, *p;
    int x;

    Init(pTree);
    CreateTree(pTree);
    NLR(pTree);

    printf("Nhập vào 1 giá trị để tìm: ");
    scanf("%d", &x);
    p = Search(pTree, x);
    if(p != NULL)
    {
        printf ("%d có xuất hiện trong cây.\n", x);
        printf("Chiều cao của nút %d là %d\n", x, Height(p));
        RemoveNode(pTree, x);
        NLR(pTree);
    }
    else
}

```

```

        printf("%d khong co trong cay.\n", x);
}

```

### **Yêu cầu**

1. Biên dịch đoạn chương trình nêu trên.
2. Cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:

```

-1
-1
-----
5      -1
-1
-----
7      10     -23    -25    -4      1      -1
-23
-----
-23    7      10     -25    -4      1      -1
-23
-----
7      10     -23    -4      1      -25    -1
-23
-----
1      2      3      4      -1      5
3

```

3. Nếu nhận xét ngắn gọn mối liên hệ giữa thứ tự nhập dữ liệu vào (với cùng tập dữ liệu – dữ liệu thứ 3, 4, và 5) với thứ tự in dữ liệu ra màn hình.
4. Vẽ hình cây nhị phân kiếm theo dữ liệu được nhập ở câu 2.
5. Nếu bỏ `Init(pTree)` trong hàm `main` kết quả có thay đổi hay không? Giải thích lý do?
6. Nếu trong hàm `CreateTree` vòng lặp `do...while` được thay đổi như dưới đây thì kết quả sẽ xuất ra màn hình sẽ như thế nào đối với dữ liệu câu 2? Giải thích lý do?

```

do
{
    printf("Nhập vào dữ liệu, -1 để kết thúc: ");
    scanf("%d", &Data);
    Insert(pRoot, Data);
    if (Data == -1)
        break;
} while (1);

```

7. Trong hàm NLR nếu ta đổi trật tự như bên dưới thì kết quả thế nào?

```

void NLR(NODE* pTree)
{
    if (pTree != NULL)
    {
        NLR(pTree->pLeft);
        printf("%4d", pTree->Key);
        NLR(pTree->pRight);
    }
}

```

```
}
```

8. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú (*//Ghi chú*) trong các hàm Search, RemoveNode.

9. Nếu trong hàm RemoveNode thay đổi như sau, kết quả có thay đổi không? Nếu có, chỉ ra cách để kết quả không thay đổi. Nếu không, giải thích lý do

```
else {
    //Ghi chú: Hàm bên dưới dùng để làm gì?
    SearchStandFor(Tree->pRight, p);
}
```

10. Trong hàm RemoveNode nếu không có dòng `delete p;` thì kết quả có gì khác? Dòng đó dùng để làm gì?

### Áp dụng – Nâng cao

1. Bổ sung chương trình mẫu cho phép tính **tổng giá trị** các nút trên cây nhị phân gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **NLR** để viết hàm **SumTree**.

2. Bổ sung chương trình mẫu cho phép tìm **giá trị nguyên lớn nhất và nhỏ nhất** trong số các phần tử nguyên trên cây nhị phân tìm kiếm gồm các giá trị nguyên.

Gợi ý: dựa vào tính chất 1, 2 của cây nhị phân tìm kiếm.

3. Bổ sung chương trình mẫu cho phép tính **số lượng các nút** của cây nhị phân gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **NLR** để viết hàm **CountNode**.

4. Bổ sung chương trình mẫu cho biết **số lượng các nút lá** trên cây nhị phân.

Gợi ý: tham khảo thao tác duyệt cây nhị phân **NLR**.

5. Sử dụng cây nhị phân tìm kiếm để giải bài toán:

- Đếm có bao nhiêu giá trị phân biệt trong dãy số cho trước
- Với mỗi giá trị phân biệt, cho biết số lượng phần tử

## BÀI TẬP THÊM

1. Sử dụng cây nhị phân tìm kiếm để giải bài toán đếm (thống kê) số lượng ký tự có trong văn bản (Không dấu).

- Xây dựng cây cho biết mỗi ký tự có trong văn bản xuất hiện mấy lần
- Nhập vào 1 ký tự. Kiểm tra ký tự đó xuất hiện bao nhiêu lần trong văn bản

2. Bài toán tương tự như trên nhưng thống kê số lượng tiếng có trong văn bản (không dấu)

Ví dụ:

Văn bản có nội dung như sau: “hoc sinh di hoc mon sinh hoc”

Kết quả cho thấy như sau:

di: 1

hoc: 3

mon: 1

sinh: 2

# CÂY CÂN BẰNG AVL

## MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu được các thao tác quay cây (quay trái, quay phải) để hiệu chỉnh cây thành cây cân bằng.
- Cài đặt hoàn chỉnh cây cân bằng AVL.

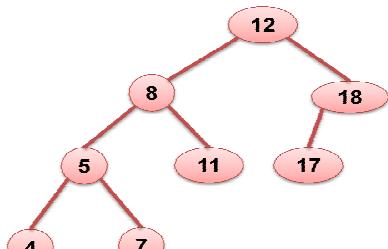
Thời gian thực hành: 120 phút – 360 phút

Lưu ý: Sinh viên phải thực hành bài tập về **Cây nhị phân** và **Cây nhị phân tìm kiếm** trước khi làm bài này.

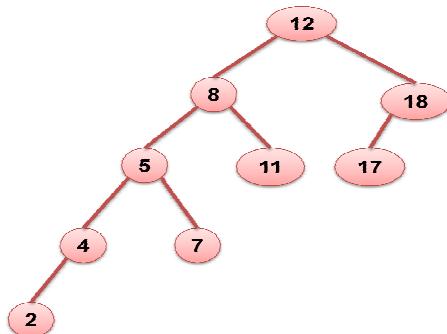
## TÓM TẮT

Cây cân bằng AVL là **cây nhị phân tìm kiếm** (NPTK) mà tại mỗi đỉnh của cây, **độ cao** của cây con trái và cây con phải **khác nhau không quá 1**.

Ví dụ 1: cây cân bằng AVL



Ví dụ 2: cây không cân bằng

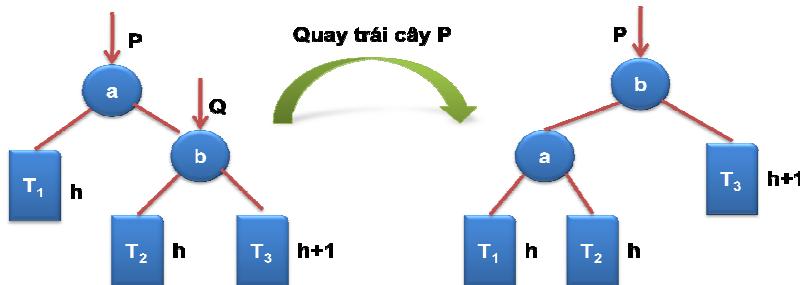


Khi thêm node mới vào cây AVL có thể xảy ra các trường hợp mất cân bằng như sau:

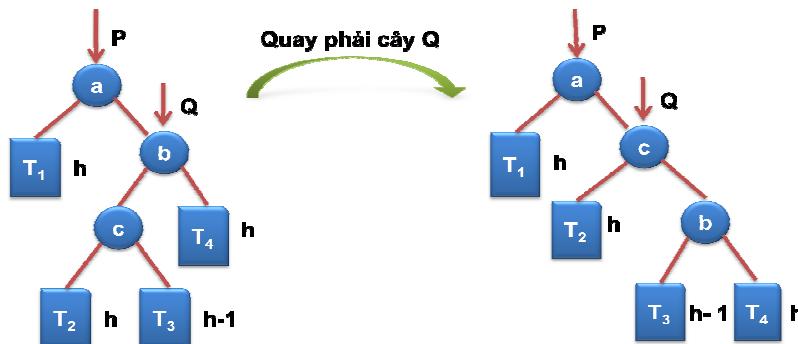
|                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Mất cân bằng phải-phải (R-R)                                                                                                                                                                                                                                                                                                                          | Mất cân bằng phải-trái (R-L)                                                                                                                                                                                                                                                                                                                          |
| <pre>graph TD; 12((12)) --&gt; 8((8)); 12 --&gt; 18((18)); 8 --&gt; 5((5)); 8 --&gt; 11((11)); 18 --&gt; 22((22)); 22 --&gt; 25((25)); 5 --&gt; 4((4)); 5 --&gt; 7((7));</pre> <p>An AVL tree with root 12. Node 12 has children 8 and 18. Node 8 has children 5 and 11. Node 18 has child 22. Node 22 has child 25. Node 5 has children 4 and 7.</p> | <pre>graph TD; 12((12)) --&gt; 8((8)); 12 --&gt; 18((18)); 8 --&gt; 5((5)); 8 --&gt; 11((11)); 18 --&gt; 22((22)); 22 --&gt; 20((20)); 5 --&gt; 4((4)); 5 --&gt; 7((7));</pre> <p>An AVL tree with root 12. Node 12 has children 8 and 18. Node 8 has children 5 and 11. Node 18 has child 22. Node 22 has child 20. Node 5 has children 4 and 7.</p> |
| Mất cân bằng trái-trái (L-L)                                                                                                                                                                                                                                                                                                                          | Mất cân bằng trái-phải (L-R)                                                                                                                                                                                                                                                                                                                          |

Xử lý mất cân bằng bằng cách sử dụng các phép quay cây

a. Quay trái



b. Quay phải



Xử lý cụ thể cho các trường hợp mất cân bằng như sau:

| MẤT CÂN BẰNG PHẢI                                                                                                   |                                                                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Mất cân bằng phải-phải (R-R) <ul style="list-style-type: none"> <li>- Quay trái tại node bị mất cân bằng</li> </ul> | Mất cân bằng phải-trái (R-L) <ul style="list-style-type: none"> <li>- Quay phải tại node con phải của node bị mất cân bằng</li> <li>- Quay trái tại node bị mất cân bằng</li> </ul> |
| MẤT CÂN BẰNG TRÁI                                                                                                   |                                                                                                                                                                                     |
| Mất cân bằng trái-trái (L-L) <ul style="list-style-type: none"> <li>- Quay phải tại node bị mất cân bằng</li> </ul> | Mất cân bằng trái-phải (L-R) <ul style="list-style-type: none"> <li>- Quay trái tại node con trái của node bị mất cân bằng</li> <li>- Quay phải tại node bị mất cân bằng</li> </ul> |

Giống với cây NPTK, các thao tác trên cây cân bằng bao gồm:

- Thêm phần tử vào cây
- Tìm kiếm 1 phần tử trên cây
- Duyệt cây
- Xóa 1 phần tử trên cây

## NỘI DUNG THỰC HÀNH

### Cơ bản

Sinh viên đọc kỹ phát biểu bài tập và thực hiện theo hướng dẫn:

Tổ chức một cây cân bằng AVL trong đó mỗi node trên cây chứa thông tin dữ liệu nguyên.

*Người dùng sẽ nhập các giá trị nguyên từ bàn phím. Với mỗi giá trị nguyên được nhập vào, phải tạo cây AVL theo đúng tính chất của nó. Nếu người dùng nhập -1 quá trình nhập dữ liệu sẽ kết thúc. Sau đó, xuất thông tin các node trên cây.*

*Khi chương trình kết thúc, tất cả các node trên cây bị xóa bỏ khỏi bộ nhớ.*

### Phân tích

- Các node trên cây cân bằng cũng giống như các node trên cây NPTK. Tuy nhiên, do mỗi lần thêm node vào cây chúng ta cần kiểm tra độ cao của node vừa thêm để kiểm soát tính cân bằng của cây nên cần bổ sung thêm giá trị cho biết sự cân bằng tại node đó vào cấu trúc của node. Cụ thể như sau:

```
struct AVLNODE
{
    int key;
    int bal; // thuộc tính cho biết giá trị cân bằng
    // 0: cân bằng, 1: lệch trái, 2: lệch phải
    NODE* pLeft;
    NODE* pRight;
};
```

- Các thao tác cần cài đặt: xoay trái cây (**RotateLeft**), xoay phải cây (**RotateRight**), thêm 1 node mới vào cây (**InsertNode**), duyệt cây theo (**Traverse**), xóa toàn bộ node trên cây (**RemoveAll**)
- Để chương trình mạch lạc và rõ ràng hơn, chúng ta sẽ cài đặt 2 hàm xử lý cân bằng khi cây lệch trái và lệch phải theo bảng phân loại ở trang 2. Như vậy trong chương trình sẽ có thêm 2 hàm **BalanceLeft** và **BalanceRight**.

### Chương trình tham khảo

```
#include <stdio.h>

struct AVLNODE
{
    int Key;
    int bal; // thuộc tính cho biết giá trị cân bằng
    // 0: cân bằng, 1: lệch trái, 2: lệch phải
    AVLNODE* pLeft;
    AVLNODE* pRight;
};

AVLNODE* CreateNode(int Data)
{
    AVLNODE* pNode;
    pNode = new AVLNODE; // Xin cấp phát bộ nhớ động để tạo một phần tử (node) mới
    if (pNode == NULL) {
        return NULL;
    }
    pNode->Key = Data;
    pNode->pLeft = NULL;
    pNode->pRight = NULL;
    pNode->bal = 0; // Ghi chú: giải thích ý nghĩa của thao tác này
    return pNode;
}

void LeftRotate(AVLNODE* &P)
```

```

AVLNODE *Q;
Q = P->pRight;
P->pRight = Q->pLeft;
Q->pLeft = P;
P = Q;
}
void RightRotate (AVLNODE* &P)
{
    //Ghi chú: sinh viên tự code cho hàm này
}
void LeftBalance (AVLNODE* &P)
{
    switch(P->pLeft->bal){
        case 1: //mất cân bằng trái trái
            RightRotate (P);
            P->bal = 0;
            P->pRight->bal = 0;
            break;
        case 2: //Ghi chú: cho biết đây là trường hợp mất cân bằng nào?
            LeftRotate (P->pLeft);
            RightRotate (P);
            switch(P->bal){
                case 0:
                    P->pLeft->bal= 0;
                    P->pRight->bal= 0;
                    break;
                case 1:
                    P->pLeft->bal= 0;
                    P->pRight->bal= 2;
                    break;
                case 2:
                    P->pLeft->bal= 1;
                    P->pRight->bal= 0;
                    break;
            }
            P->bal = 0;
            break;
    }
}

void RightBalance (AVLNODE* &P)
{
    switch(P->pRight->bal){
        case 1: //Ghi chú: cho biết đây là trường hợp mất cân bằng nào?
            RightRotate (P->pRight);
            LeftRotate (P);
            switch(P->bal){
                case 0:
                    P->pLeft->bal= 0;
                    P->pRight->bal= 0;
                    break;
                case 1:
                    P->pLeft->bal= 0;
                    P->pRight->bal= 2;
                    break;
                case 2:
                    P->pLeft->bal= 1;
                    P->pRight->bal= 0;
                    break;
            }
            P->bal = 0;
            break;
        case 2: //Ghi chú: cho biết đây là trường hợp mất cân bằng nào?
            LeftRotate (P);
    }
}

```

```

        P->bal = 0;
        P->pLeft->bal = 0;
        break;
    }
}

int InsertNode(AVLNODE* &tree, int x)
{
    int res;
    if(tree==NULL){ //Ghi chú: cho biết ý nghĩa của câu lệnh này
        tree = CreateNode(x);
        if(tree==NULL){
            return -1; //thêm ko thành công vì thiếu bộ nhớ
        }
        return 2;//thêm thành công và làm tăng chiều cao cây
    }
    else {
        if(tree->Key==x){
            return 0; //khóa này đã tồn tại trong cây
        }
        else if(tree->Key > x){
            res = InsertNode(tree->pLeft,x);
            if(res < 2) {
                return res;
            }
            switch(tree->bal){ //Ghi chú: giải thích ý nghĩa của câu lệnh
switch này
                case 0:
                    tree->bal = 1;
                    return 2;
                case 1:
                    LeftBalance(tree);
                    return 1;
                case 2:
                    tree->bal = 0;
                    return 1;
            }
        }
        else{
            res = InsertNode(tree->pRight,x);
            if(res<2){
                return res;
            }
            switch(tree->bal){
                case 0:
                    tree->bal=2;
                    return 2;
                case 1:
                    tree->bal = 0;
                    return 1;
                case 2:
                    RightBalance(tree);
                    return 1;
            }
        }
    }
}
void Traverse(AVLNODE* t)
{
    if(t!=NULL)
    {
        Traverse(t->pLeft);
        printf("Khoa: %d, can bang: %d\n", t->Key,t->bal);
        Traverse(t->pRight);
    }
}

```

```

}

void RemoveAll(AVLNODE* &t)
{
    if(t!=NULL) {
        RemoveAll(t->pLeft);
        RemoveAll(t->pRight);
        delete t;
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    AVLNODE *tree;
    //Ghi chú: Tại sao lại phải thực hiện phép gán phía dưới?
    tree = NULL;
    int Data;
    do
    {
        printf("Nhập vào địa chỉ, -1 để kết thúc: ");
        scanf("%d", &Data);
        if (Data == -1)
            break;
        InsertNode(tree, Data);
    }while (Data != -1);

    printf("\nCây AVL vừa tạo: \n");
    Traverse(tree);

    RemoveAll(tree);

    return 0;
}

```

### **Yêu cầu**

1. Biên dịch đoạn chương trình nêu trên.
2. Cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:  
-1  
10    30    35    32    20    8    -1  
30    40    50    -10    -5    -1
3. Nhận xét trình tự các node được xuất ra màn hình? Giải thích tại sao lại in ra được trình tự như nhận xét?
4. Sinh viên hoàn tất hàm **RightRotate** trong source code.  
*Gợi ý:* RightRotate tương tự hàm LeftRotate.
  5. Biên dịch lại chương trình sau khi hoàn thành câu 3 và cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:  
50    20    30    10    -5    7    15    35    57    65    55    -1
  6. Vẽ hình cây AVL được tạo ra từ phần nhập liệu ở câu 5.
  7. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú (*//Ghi chú*) trong các hàm InsertNode, BalanceLeft, BalanceRight, \_tmain.
  8. Sinh viên cài đặt lại các hàm dùng cho cây nhị phân và cây NPTK để áp dụng cho cây AVL.

## Áp dụng – Nâng cao

- Sinh viên tự cài đặt thêm chức năng cho phép người dùng nhập vào khóa x và kiểm tra xem khóa x có nằm trong cây AVL hay không.

Cho dãy A như sau:

1      3      5      7      9      12      15      17      21      23      25      27

- Tạo cây AVL từ dãy A. Cho biết số phép so sánh cần thực hiện để tìm phần tử 21 trên cây AVL vừa tạo.
  - Tạo cây nhị phân tìm kiếm từ dãy A dùng lại đoạn code tạo cây của bài thực hành trước). Cho biết số phép so sánh cần thực hiện để tìm phần tử 21 trên cây nhị phân tìm kiếm vừa tạo.
  - So sánh 2 kết quả trên và rút ra nhận xét?
- Cài đặt chương trình đọc các số nguyên từ tập tin input.txt (không biết trước số lượng số nguyên trên tập tin) và tạo cây AVL từ dữ liệu đọc được
  - Cài đặt cây cân bằng AVL trong đó mỗi node trên cây lưu thông tin sinh viên.
  - Tự tìm hiểu và cài đặt chức năng xóa một node ra khỏi cây AVL.

## BÀI TẬP THÊM

- Viết chương trình cho phép tạo, tra cứu và sửa chữa từ điển Anh-Việt (sinh viên liên hệ với GVLT để chép file từ điển Anh-Việt)
- Cài đặt lại các bài tập thêm của cây NPTK bằng cách dùng cây AVL

## NÉN HUFFMAN

### MỤC TIÊU

Hoàn thành bài thực hành này, sinh viên có thể:

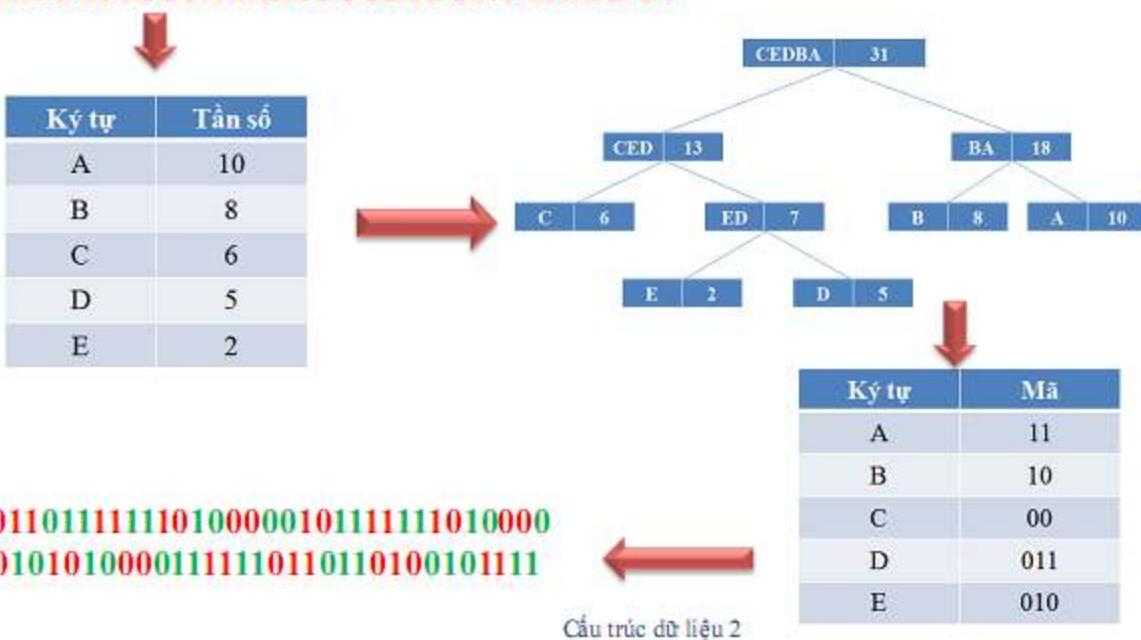
- Nắm rõ các bước của thuật toán nén huffman tĩnh.

### THỜI GIAN THỰC HÀNH

Từ 120 phút đến 400 phút

### TÓM TẮT

ADD AABBCCBAAABBCCCBBBCDAADDEEAA



**Nén Huffman** là phương pháp mã hóa bằng **mã có độ dài thay đổi** (variable length encoding) trong đó chỉ sử dụng vài bit để biểu diễn 1 ký tự và độ dài mã bit cho các ký tự không giống nhau (ký tự xuất hiện nhiều lần được biểu diễn bằng mã ngắn và ngược lại).

Thuật toán nén Huffman bao gồm 5 bước:

- B1: Lập bảng thống kê tần số xuất hiện của các ký tự.
- B2: Xây dựng cây Huffman dựa vào bảng thống kê tần số xuất hiện.
- B3: Phát sinh bảng mã bit cho từng ký tự tương ứng.
- B4: Duyệt tập tin, thay thế các ký tự trong tập tin bằng mã bit tương ứng.
- B5: Lưu lại thông tin của cây Huffman cho giải nén.

### NỘI DUNG THỰC HÀNH

Cài đặt thuật toán nén Huffman. Dữ liệu được nhập từ file “input.txt”. Xuất ra màn hình chuỗi bit tương ứng với nội dung nhập.

Ví dụ

Với file input.txt có nội dung:

ABBCCCDDDD

Chuỗi bit tương ứng khi xuất ra sẽ là

10010110 11111110 000

Trong đó A (100) B (101) C(11) và D(0)

## PHÂN TÍCH

Để biểu diễn một nút trong cây huffman, ta dùng:

```
struct NODE {  
    unsigned char    c;           // ký tự của node  
    int             freq;        // số lần xuất hiện  
    bool            used;        // đánh dấu node có thuộc bảng thống kê ko  
                                // used = true: ko thuộc bảng thống kê  
    int             nLeft;       // chỉ số nút con nằm bên trái  
    int             nRight;      // chỉ số nút con nằm bên phải  
};
```

Cây Huffman được lưu trữ dưới dạng mảng

NODE huffTree[MAX\_NODE];

Trong đó MAX\_NODE là 1 số nguyên > số lượng nút tối đa của cây Huffman.

(Ta có thể thấy số lượng nút tối đa của cây Huffman sẽ  $< 2^8 * 9$ )

Để biểu diễn một mã bit, ta dùng

```
struct MABIT {  
    char*          bits;        // chua mang bit  
    int            soBit;       // so luong bit cua ma  
};
```

Bảng mã bit: mã bit của 256 ký tự

MABIT bangMaBit[256];

### BUỚC 1: LẬP BẢNG THỐNG KÊ TẦN SỐ XUẤT HIỆN

Ta tạo ra 256 node tương ứng với 256 ký tự ASCII. Sau đó, đọc từng ký tự tương ứng từ tập tin nhập và tăng tần số xuất hiện của node tương ứng ký tự nhập.

Khởi tạo node:

```
for (int i = 0; i < 256; i++) {  
    huffTree[i].c = i;  
    huffTree[i].freq = 0;
```

```
    ...
}
```

Thông kê tần số:

```
while (1) {
    fscanf(fi, "%c", &c);
    if (feof(fi)) {
        break;
    }
    huffTree[c].freq++; // tang tan so xuat hien cua ky tu c
}
```

Nhận xét:

Việc tạo ra 256 node có thể dư do trong văn bản hiếm khi xảy ra trường hợp 256 ký tự ASCII cùng xuất hiện nhưng cho phép ta truy xuất nhanh chóng đến node tương ứng với ký tự c (với quy ước node huffTree[c] tương ứng với ký tự c). Điều này rất quan trọng khi làm việc với các tập tin có độ dài lớn.

## BUỚC 2: XÂY DỰNG CÂY HUFFMAN

Các bước phát sinh cây:

- B1: Chọn trong bảng thống kê 2 phần tử có tần suất thấp nhất.
- B2: Tạo 2 node của cây cùng với node cha z có trọng số bằng tổng trọng số 2 nút con.
- B3: Loại 2 phần tử x, y khỏi bảng thống kê.
- B4: Thêm phần tử z vào bảng thống kê.
- B5: lặp lại bước 1 đến bước 4 cho đến khi còn 1 phần tử trong bảng thống kê.

Với cách lưu trữ cây Huffman dùng mảng như ở trên, các thao tác được thực hiện như sau:

- Chọn trong bảng thống kê có tần suất thấp nhất

Tìm 2 phần tử trong mảng huffTree có tần suất thấp nhất. Chỉ xét các phần tử thuộc bảng thống kê (huffTree[i].used == false) và có xuất hiện trong file dữ liệu (huffTree[i].freq > 0)

- Tạo node mới của cây Huffman có 2 nút con i, j

Thêm node được tạo vào cuối mảng, sử dụng nLeft và nRight để lưu vị trí 2 nút con.

```
huffTree[nNode].freq = huffTree[i].freq + huffTree[j].freq;
huffTree[nNode].nLeft = i;
huffTree[nNode].nRight = j;
```

- Loại phần tử x, y khỏi bảng thống kê.

Để loại x, y khỏi bảng thống kê, gán

```
huffTree[i].used = true;
huffTree[j].used = true;
```

- Thêm phần tử nNode vào bảng thống kê

```
huffTree[nNode].used = false;
```

Ví dụ: với dữ liệu vào ABBCCCCDDDD, bảng thống kê ban đầu:

|        |       |       |       |       |     |       |  |
|--------|-------|-------|-------|-------|-----|-------|--|
|        | 65    | 66    | 67    | 68    | ... | 255   |  |
| C      | A     | B     | C     | D     |     | ?     |  |
| Freq   | 1     | 2     | 3     | 4     |     | 0     |  |
| Used   | FALSE | FALSE | FALSE | FALSE |     | FALSE |  |
| nLeft  | -1    | -1    | -1    | -1    |     | -1    |  |
| nRight | -1    | -1    | -1    | -1    |     | -1    |  |

Tìm 2 phần tử có tần suất thấp nhất: A (1) và B (2). Tạo node mới (nút 256) . Loại A, B khỏi bảng thống kê và thêm nút 256 vào bảng thống kê

|        |      |      |       |       |     |       |       |  |
|--------|------|------|-------|-------|-----|-------|-------|--|
|        | 65   | 66   | 67    | 68    | ... | 255   | 256   |  |
| C      | A    | B    | C     | D     |     | ?     | A     |  |
| Freq   | 1    | 2    | 3     | 4     |     | 0     | 3     |  |
| Used   | TRUE | TRUE | FALSE | FALSE |     | FALSE | FALSE |  |
| nLeft  | -1   | -1   | -1    | -1    |     | -1    | 65    |  |
| nRight | -1   | -1   | -1    | -1    |     | -1    | 66    |  |

Tìm 2 phần tử có tần suất thấp nhất: C (3) và 256 (3) (Lưu ý A, B có used = TRUE không được xét) . Tạo node mới (nút 257) . Loại C, 256 khỏi bảng thống kê và thêm nút 257 vào bảng thống kê

|        |      |      |      |       |     |       |      |       |  |
|--------|------|------|------|-------|-----|-------|------|-------|--|
|        | 65   | 66   | 67   | 68    | ... | 255   | 256  | 257   |  |
| C      | A    | B    | C    | D     |     | ?     | A    | A     |  |
| Freq   | 1    | 2    | 3    | 4     |     | 0     | 3    | 6     |  |
| Used   | TRUE | TRUE | TRUE | FALSE |     | FALSE | TRUE | FALSE |  |
| nLeft  | -1   | -1   | -1   | -1    |     | -1    | 65   | 256   |  |
| nRight | -1   | -1   | -1   | -1    |     | -1    | 66   | 67    |  |

Tương tự, sau khi tạo nút 258, chỉ còn 1 phần tử thuộc bảng thống kê (258). Quá trình tạo cây dừng.

|        |      |      |      |      |     |       |      |      |       |
|--------|------|------|------|------|-----|-------|------|------|-------|
|        | 65   | 66   | 67   | 68   | ... | 255   | 256  | 257  | 258   |
| C      | A    | B    | C    | D    |     | ?     | A    | A    | A     |
| Freq   | 1    | 2    | 3    | 4    |     | 0     | 3    | 6    | 10    |
| Used   | TRUE | TRUE | TRUE | TRUE |     | FALSE | TRUE | TRUE | FALSE |
| nLeft  | -1   | -1   | -1   | -1   |     | -1    | 65   | 256  | 68    |
| nRight | -1   | -1   | -1   | -1   |     | -1    | 66   | 67   | 256   |

Sau quá trình tạo cây, nút huffTree[nNode – 1] chính là nút gốc của cây Huffman.

### ***BUỚC 3: PHÁT SINH BẢNG MÃ BIT***

---

Duyệt cây Huffman từ nút gốc đến nút lá của các ký tự.

Nút gốc của cây Huffman: huffTree[nNode - 1]

Kiểm tra một nút là nút lá: huffTree[i].nLeft == -1 && huffTree[i].rRight == -1

Duyệt đệ quy từ nút gốc của cây Huffman. Thêm bit 0 vào khi đi qua nhánh trái, thêm bit 1 khi đi qua nhánh phải.

#### ***BUỚC 4: THAY THẾ KÝ TỰ BẰNG MÃ BIT***

Sử dụng 1 biến unsigned char out để lưu chuỗi bit xuất ra. Duyệt các ký tự, bật (gán = 1) các bit tương ứng của biến out (tùy theo mã bit của ký tự). Xuất ra biến out khi chuỗi bit có độ dài 8 (đủ 1 byte).

#### ***Ôn tập: THAO TÁC XỬ LÝ BIT***

Tham khảo các phép xử lý bit cơ bản

(<http://forums.congdongcviet.com/showthread.php?t=316>)

2 thao tác cần thực hiện:

- Bật bit i của số nguyên out

`out = out | (1 << i);`

Ví dụ: bật bit 2 của biến out

|                   |   |   |   |   |   |   |   |   |
|-------------------|---|---|---|---|---|---|---|---|
|                   | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Out               | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $1 \ll 2$         | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $out   (1 \ll 2)$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

- Lấy bit i của số nguyên out

`(out >> i) & 1`

Ví dụ: lấy bit 2 của biến out

|                  |   |   |   |   |   |   |   |   |
|------------------|---|---|---|---|---|---|---|---|
|                  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Out              | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| $out >> 2$       | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1                | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $(out >> 2) & 1$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

#### ***CODE THAM KHẢO (StaticHuffman.rar)***

Xem code đi kèm.

#### ***YÊU CẦU***

1. Biên dịch chương trình tham khảo.
2. Nhập dữ liệu input.txt có nội dung “ACBCDBDCDD”. Chạy tay thuật toán nén Huffman, cho biết kết quả bảng thông kê, cây Huffman, bảng mã bit và chuỗi bit xuất ra.
3. Trong hàm NenHuffman()

Bỏ dấu comment (//) ở các dòng

```
// XuatBangThongKe();  
// XuatCayHuffman(nRoot, 0);  
// XuatBangMaBit();
```

Chạy chương trình, so sánh kết quả xuất ra màn hình với kết quả của câu 2.

4. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm ThongKeTanSoXuatHien() và XuatBangThongKe().
5. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm TaoCayHuffman() và Tim2PhanTuMin().
6. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm PhatSinhMaBit() và DuyetCayHuffmanx().
7. Trả lời câu hỏi ở các dòng có ghi chú ở 2 hàm NenHuffman() và MaHoa1KyTu() và XuatByte().

### Áp dụng - Nâng cao

8. Sửa lại chương trình để xuất nội dung nén ra file “output.txt” thay vì xuất ra màn hình.  
Ví dụ: với ví dụ ở trên, thay vì xuất ra màn hình chuỗi bit 10010110 11111110 000, xuất ra file 3 byte

βÔ

trong đó β (10010110) Ô (11111110) (byte thứ 3 là 0 nên không thể hiển thị)

9. Sửa lại chương trình để xuất nội dung nén và các thông tin cần thiết ra file output.huf” sao cho có thể sử dụng file output.huf cho quá trình giải nén.
10. Viết hàm giải nén: đọc nội dung từ file output.huf ở câu 9 và xuất nội dung giải nén ra file decode.txt. So sánh nội dung decode.txt và input.txt