

# **BÀI GIẢNG**

# **PHÂN TÍCH VÀ THIẾT KẾ GIẢI THUẬT**

Giảng viên: **NGUYỄN THÁI DƯ**

<.....>

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:  
**NGUYỄN THÁI DƯ**

### Tài liệu tham khảo

◆ Tài liệu tham khảo:

- Nguyễn Văn Linh (2022). Phân tích và Thiết kế thuật toán. Đại học Cần Thơ.
- Trương Hải Bằng (2005). Giáo trình cấu trúc dữ liệu 2. NXB ĐHQG Tp.HCM.
- Nguyễn Hồng Chương (1999). CTDL và giải thuật ứng dụng cài đặt bằng C. NXB Tp.HCM.
- Lafore, R. (1998). Data Structures and Algorithms in Java. New York: Waite Group Press.
- Niklaus, W. (1993). Cấu trúc dữ liệu + Giải thuật = Chương trình (bản dịch của Nguyễn Quốc Cường. TP.HCM: NXB Giáo dục.
- Sedgewick, R. (1999). Algorithms in C++. Sydney: Addison Wesley

### Chương 0. GIỚI THIỆU

- ◆ **Chương 1: Kỹ thuật phân tích giải thuật**
- ◆ **Chương 2: Kỹ thuật thiết kế giải thuật**
- ◆ **Chương 3: Sắp xếp ngoại**
- ◆ **Chương 4: Bảng băm**
- ◆ **Chương 5: Cây 2-3-4; B-TREE**
- ◆

### Phương pháp học tập

◆ **Giảng viên:** Cung cấp bài giảng, tài liệu.

◆ **Sinh viên:**

- Tự giác làm các bài tập
- Đọc tài liệu tham khảo liên quan;
- Trong giờ học PHẢI trả lời khi GV hỏi;
- PHẢI để điện thoại ở chế độ rung và KHÔNG nghe điện thoại trong lớp.
- KHÔNG sử dụng máy tính trong giờ lý thuyết;

◆ **GV-SV:** Giải đáp thắc mắc - Trao đổi

## Đánh giá học phần

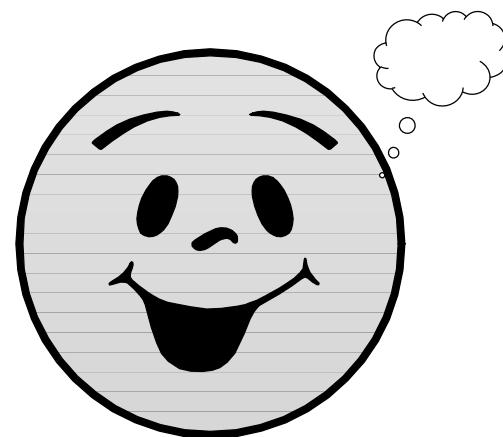
◆ Tổng cộng: 45 tiết lý thuyết

◆ Điểm thường xuyên:

- 
- 
- 
- 

◆ Thi kết thúc học phần:

- Thi lý thuyết (*Không sử dụng tài liệu*)



Cảm ơn !

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:  
**NGUYỄN THÁI DƯ**

### Sự cần thiết phải phân tích giải thuật

- ◆ Với phần lớn các bài toán, thường có nhiều giải thuật khác nhau để giải một bài toán.
- ◆ Làm cách nào để chọn giải thuật tốt nhất để giải một bài toán?
- ◆ Làm cách nào để so sánh các giải thuật cùng giải được một bài toán?  
=> Cần phải **đánh giá các giải thuật** để lựa chọn giải thuật tốt nhất.

### Chương 1 KỸ THUẬT PHÂN TÍCH GIẢI THUẬT

- ◆ Sự cần thiết phải phân tích giải thuật
- ◆ Thời gian thực hiện của giải thuật
- ◆ Khái niệm độ phức tạp của giải thuật
- ◆ Cách tính độ phức tạp của:
  - Chương trình không gọi chương trình con
  - Chương trình có gọi chương trình con không đệ quy
  - Chương trình đệ quy

### Sự cần thiết phải phân tích giải thuật

- ◆ **Đánh giá giải thuật**
  - Tính đúng đắn
    - Chạy trên dữ liệu thử
    - Chứng minh lý thuyết (bằng toán học chẳng hạn)
  - Tính đơn giản:
    - > Thường chỉ sử dụng vài lần
  - Tính nhanh chóng (thời gian thực thi)
    - Quan trọng khi chương trình được thực thi nhiều lần, chương trình có khối lượng dữ liệu nhập lớn.
    - Hiệu quả thời gian thực hiện của giải thuật

## Sự cần thiết phải phân tích giải thuật

### ◆ Đo thời gian thực hiện chương trình

- Lập trình và đo thời gian thực hiện
- Phụ thuộc vào tập lệnh của máy tính
- Kỹ năng của người lập trình
- Dữ liệu đầu vào

→ Tính độ phức tạp thời gian thực hiện của giải thuật  
= độ đo sự thực thi của giải thuật

## Đơn vị đo thời gian thực hiện

- Đơn vị của  $T(n)$  không phải là đơn vị đo thời gian bình thường như giờ, phút giây...
- Thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.
- **Ví dụ:** Khi ta nói thời gian thực hiện của một chương trình là  $T(n) = Cn$  thì có nghĩa là chương trình ấy cần  $Cn$  chỉ thị thực thi.

## Thời gian thực hiện của chương trình

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

## Thời gian thực hiện trong trường hợp xấu nhất

- ◆ Thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào.
- ◆ Dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau.
- ◆ *Ví dụ:* chương trình sắp xếp dãy số nguyên tăng dần. Nhập vào dãy số nguyên.  
=> Dãy số nhập vào có thể: có thứ tự, chưa có thứ tự, có thứ tự tăng hoặc có thứ tự giảm.  
=> Thời gian thực hiện trong các trường hợp: tốt nhất, xấu nhất, trung bình

## Thời gian thực hiện trong trường hợp xấu nhất

- ◆  $T(n)$  là thời gian thực hiện chương trình **trong trường hợp xấu nhất** trên dữ liệu vào có kích thước  $n$
- ◆  $T(n)$  là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước  $n$ .

## Tỷ suất tăng

- ◆  $T(n)$  có tỷ suất tăng  $f(n)$  nếu tồn tại hằng số  $C$  và  $N_0$  sao cho  $T(n) \leq Cf(n) \quad \forall n \geq N_0$
- ◆ Cho một hàm không âm  $T(n)$ , luôn tồn tại tỷ suất tăng  $f(n)$  của nó
- ◆ *Ví dụ 1:* Giả sử  $T(0) = 1$ ,  $T(1) = 4$ , tổng quát  $T(n) = (n+1)^2$ ,
  - Đặt  $N_0 = 1$  và  $C = 4$  thì với mọi  $n \geq 1$ , ta có:
  - $T(n) = (n+1)^2 \leq 4n^2 \quad \forall n \geq 1,$   
 $\Rightarrow$  Tỷ suất tăng của  $T(n)$  là  $n^2$ .

## Tỷ suất tăng

- ◆ *Ví dụ 2:* chứng minh rằng:

Tỷ suất tăng của  $T(n) = 3n^3 + 2n^2$  là  $n^3$

*Giải*

- Chọn  $N_0 = 0$  và  $C = 5$  với mọi  $n \geq 0$ , ta có:  
 $3n^3 + 2n^2 \leq 5n^3 \quad \forall n \geq 0$   
 $\Rightarrow$  Tỷ suất tăng của  $T(n)$  là  $n^3$ .

## Quy tắc:

$T(n)$  là đa thức của  $n$  thì tỷ suất tăng là bậc cao nhất của  $n$

## Khái niệm độ phức tạp của giải thuật

- ◆ Cùng một bài toán, có 2 thuật toán P1 và P2 với thời gian thực hiện là:
  - P1 có  $T1(n) = 100n^2$
  - P2 có  $T2(n) = 5n^3$
- ◆ Giải thuật nào chạy nhanh hơn ?
  - Xét nếu  $n \leq 20$  thì  $T1(n) > T2(n)$
  - Xét nếu  $n > 20$  thì  $T1(n) < T2(n)$
- Chọn P1 vì hầu hết các trường hợp  $T1(n) < T2(n)$
- ◆ Dùng thời gian để lựa chọn rất khó khăn vì phải so sánh 2 đa thức

## Khái niệm độ phức tạp của giải thuật

- P1 có  $T_1(n) = 100n^2$  → tỷ suất tăng là  $n^2$
- P2 có  $T_2(n) = 5n^3$  → tỷ suất tăng là  $n^3$
- ◆ Chỉ xét  $n^2$  và  $n^3$  thì rất dễ dàng lựa chọn vì  $n^2 < n^3$

=> So sánh theo tỷ suất tăng hơn là so sánh trực tiếp các hàm  $T(n)$

## Khái niệm độ phức tạp của giải thuật

- ◆ Một số hàm thể hiện độ phức tạp thường gặp:  $\log_2 n$ ,  $n$ ,  $n \log_2 n$ ,  $n^2$ ,  $n^3$ ,  $2^n$ ,  $n!$ ,  $n^n$ .
- ◆ Các hàm:  $\log_2 n$ ,  $n$ ,  $n \log_2 n$ ,  $n^2$ ,  $n^3$  gọi là hàm đa thức
- ◆ Ba hàm cuối cùng:  $2^n$ ,  $n!$ ,  $n^n$  gọi là dạng hàm mũ,
- ◆ Nhận xét:
  - Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện
  - Các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật

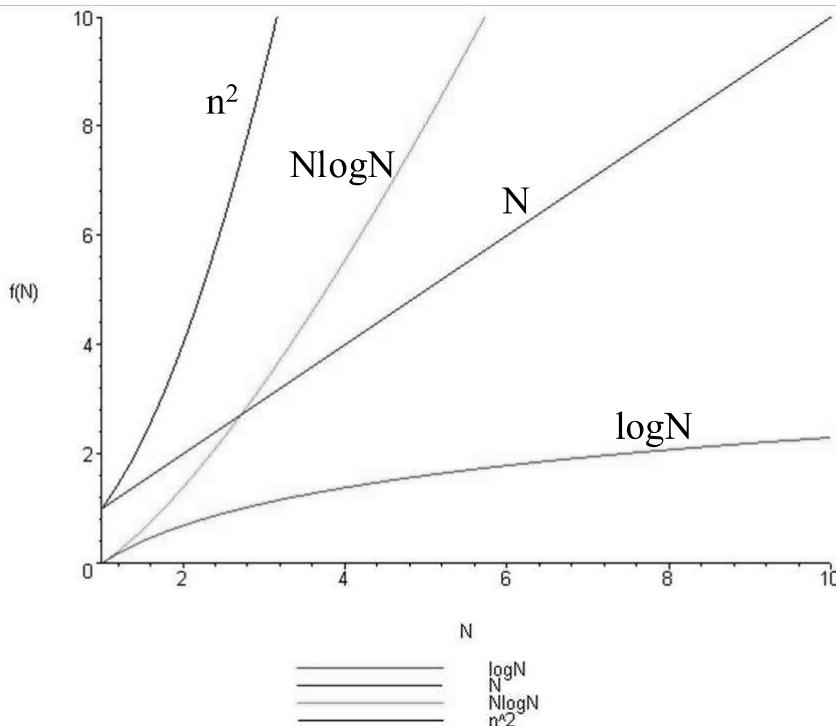
## Khái niệm độ phức tạp của giải thuật

◆ Cho một hàm  $T(n)$ ,  $T(n)$  gọi là có độ phức tạp  $f(n)$  nếu tồn tại các hằng  $C$ ,  $N_0$  sao cho  $T(n) \leq Cf(n)$  với mọi  $n \geq N_0$  (tức là  $T(n)$  có tỷ suất tăng là  $f(n)$ ) và ký hiệu  $T(n)$  là  $O(f(n))$  (đọc là “ô của  $f(n)$ ”).

◆ Ví dụ:  $T(n) = (n + 1)^2$  có tỷ suất tăng là  $n^2$  nên hàm  $T(n)$  có độ phức tạp  $O(n^2)$

### Tính chất

- $O(C.f(n)) = O(f(n))$  với  $C$  là hằng số
- $O(C) = O(1)$



## Cách tính độ phức tạp

- ◆ Chương trình không gọi chương trình con
- ◆ Chương trình có gọi chương trình con không đệ quy
- ◆ Chương trình đệ quy

### Quy tắc tổng quát để phân tích một chương trình không có chương trình con

- ◆ Các lệnh gán, nhập, xuất, return: O(1)
- ◆ Một chuỗi tuần tự các lệnh: Quy tắc cộng
  - if (điều kiện) CV1
  - else CV2
- ◆ Câu trúc IF: Max (*điều kiện, CV1, CV2*)
  - ☞ Thường thời gian kiểm tra điều kiện là O(1)
- ◆ Câu trúc vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp.
  - Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

## Cách tính độ phức tạp

- ◆ Trước hết ta có hai quy tắc quan trọng là quy tắc cộng và quy tắc nhân
- ◆ Cho 2 đoạn chương trình:
  - P1 có thời gian thực hiện  $T_1(n) = O(f_1(n))$
  - P2 có thời gian thực hiện  $T_2(n) = O(f_2(n))$
- ◆ Quy tắc cộng:
  - Thời gian thực thi P1 và P2 nối tiếp nhau sẽ là:
    - $T(n) = T_1(n) + T_2(n) = O(\max(f_1(n), f_2(n)))$
- ◆ Quy tắc nhân:
  - Thời gian thực thi P1 và P2 lồng nhau:
    - $T(n) = T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$

### Vòng lặp xác định số lần lặp

- ◆ For (*i=a; i<=b; i++*) : Số lần lặp =  $b-a+1$
- ◆ For (*i=a; i<b; i++*) : Số lần lặp =  $b-a$
- ◆ For (*i=1; i<=n; i=2\*i*)
  - Sau lần lặp thứ 1:  $i = 2$
  - Sau lần lặp thứ 2:  $i = 4$
  - .....
  - Sau lần lặp thứ k:  $i = 2^k$
  - Lặp kết thúc khi  $i = 2^k = n$
  - $k = \log_2 n \Rightarrow$  số lần lặp =  $\log_2 n$

# Vòng lặp không xác định số lần lặp

- ◆ While (điều kiện)
  - ◆ Do... while...
  - ◆ Xác định số lần lặp trong trường hợp xấu nhất

Nguyễn Thái Dư - AGU

21

## Cách tính độ phức tạp

- ◆ Ví dụ: Tính thời gian thực hiện của các đoạn chương trình sau:

- Thuật toán tính tổng các số từ 1 đến n

{1} s=0; Lệnh {1} và {2} nối tiếp nhau. Lệnh {1} tốn O(1)

{2} for (i= 1; i<=n; i++)

$\{3\}$        $s=s+i;$              $\text{long} \rightarrow \text{short}$ ,  $\text{in} \rightarrow \text{out}$ .  
Ta tiến hành tính độ phức tạp từ  
trong ra.

Lệnh {3} tồn O(1)

## **Quy tắc tổng quát để phân tích một chương trình không có chương trình con**

- ◆ Trình tự đánh giá:
    - Nối tiếp: Từ trên xuống
    - Lồng nhau: Từ trong ra

Nguyễn Thái Dư - AGU

22

## Cách tính độ phức tạp

```
{1} s=0;  
{2} for (i= 1;i<=n;i++)  
{3}     s=s+i;
```

- Lệnh {1} tồn O(1) thời gian
  - Lệnh {3} tồn O(1) thời gian
    - Vòng lặp {2} lặp n lần, mỗi lần tồn O(1) do đó vòng lặp {2} tồn  $O(n \cdot 1) = O(n)$ .
    - Lệnh {1}, {2} nối tiếp nhau  $O(\max(1, n)) = O(n)$

=> **độ phức tạp là O(n)**

Nguyễn Thái Dư - AGU

23

Nguyễn Thái Dư - AGU

24

## Cách tính độ phức tạp

❖ **Ví dụ:** Tính thời gian thực hiện của đoạn chương trình sắp xếp “nối bọt”

```
void BubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<(n-1); i++) //1
        for(j=n-1; j>i+1; j--) //2
            if(a[j-1] > a[j]) { //3
                temp = a[j-1]; //4
                a[j-1] = a[j]; //5
                a[j] = temp; //6
            }
}
```

- Cả ba lệnh đổi chỗ {4} {5} {6} tốn O(1) thời gian, do đó lệnh {3} tốn O(1).
- Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1). Do đó vòng lặp {2} tốn O((n-i).1)=O(n-i).
- Vòng lặp {1} lặp (n-1) lần vậy độ phức tạp của giải thuật là:  
$$T(n) = \sum_{i=1}^{n-1} (n-i) = (n-1) + (n-2) + \dots + (n-k-1) + \dots + 1 = \frac{n(n-1)}{2}$$

$$T(n) = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2)$$

Nguyễn Thái Dư - AGU

25

## Cách tính độ phức tạp

❖ **Ví dụ:** Tìm kiếm tuần tự. Hàm tìm kiếm Search nhận vào một mảng a có n số nguyên và một số nguyên x, hàm sẽ trả về giá trị logic TRUE nếu tồn tại một phần tử a[i] = x, ngược lại hàm trả về FALSE.

- Giải thuật tìm kiếm tuần tự là lần lượt so sánh x với các phần tử của mảng a,
  - Bắt đầu từ a[1], nếu tồn tại a[i] = x thì dừng và trả về TRUE,
  - Ngược lại nếu tất cả các phần tử của a đều khác X thì trả về FALSE.

Nguyễn Thái Dư - AGU

26

## Cách tính độ phức tạp

❖ **Ví dụ:** Hàm tìm kiếm tuần tự.

```
int LinearSearch (int a[], int n, int x)
{
    int i=0; //1
    found =0; //2
    while ((i<n) && ! found) //3
        if (a[i]==x) found =1; //4
        else i++;
    return found; //5
}
```

Các lệnh {1}, {2}, {3} và {5} nối tiếp nhau,

Ba lệnh {1}, {2} và {5} đều có độ phức tạp O(1)  
Do đó độ phức tạp của hàm Search chính là độ phức tạp lớn nhất trong 4 lệnh này.

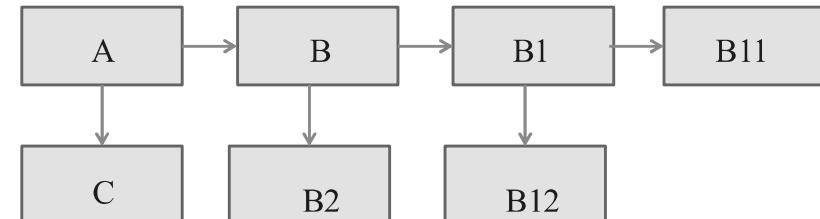
Lệnh {4} có độ phức tạp O(1)  
Lệnh {3} thực hiện n lần  
Vậy ta có  $T(n) = O(n)$ .

Nguyễn Thái Dư - AGU

27

## Cách tính độ phức tạp

- Chương trình có gọi chương trình con (không đệ quy)
- Quy tắc: tính từ trong ra ngoài



- Để tính thời gian thực hiện của A, ta tính theo các bước sau:
  - Tính thời gian thực hiện của C, B2, B11 và B12.
  - Tính thời gian thực hiện của B1.
  - Tính thời gian thực hiện của B.
  - Tính thời gian thực hiện của A.

Nguyễn Thái Dư - AGU

28

## Cách tính độ phức tạp

◆ **Ví dụ:** Ta có thể viết lại chương trình sắp xếp bubble như sau:

- Trước hết chúng ta viết thủ tục HoanDoi để thực hiện việc hoàn đổi hai phần tử cho nhau,
- Sau đó trong thủ tục Bubble, khi cần ta sẽ gọi đến thủ tục HoanDoi này.

```
void HoanDoi(int &x, int &y)
{
    int tam = x; x = y; y = tam; }
```

## Cách tính độ phức tạp

```
void BubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<(n-1); i++) //1
        for(j=n-1; j>i+1; j--) //2
            if(a[j-1] > a[j]) HoanDoi(a[j-1], a[j]); //3
}
```

## Cách tính độ phức tạp

```
void HoanDoi(int &x, int &y)
{
    int tam = x; x = y; y = tam; }

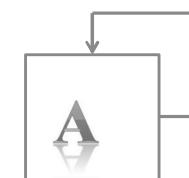
void BubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<(n-1); i++) //1
        for(j=n-1; j>i+1; j--) //2
            if(a[j-1] > a[j])
                HoanDoi(a[j-1], a[j]); //3
}
```

- Chương trình Bubble gọi chương trình con Swap
- Thời gian thực hiện của Swap là O(1) vì nó chỉ bao gồm 3 lệnh gán
- Trong Bubble, lệnh {3} gọi Swap nên chỉ tốn O(1)
- Lệnh {2} thực hiện n-i lần, mỗi lần tốn O(1), nên tốn O(n-i).
- Lệnh {1} thực hiện n-1 lần nên

$$T(n) = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2)$$

## Phân tích các chương trình đệ quy

- ◆ Với các chương trình có gọi các chương trình con đệ quy, ta không thể áp dụng cách tính như trình bày phần trước vì một chương trình đệ quy sẽ gọi chính bản thân nó.
- ◆ Chương trình đệ quy có thể minh họa như hình ảnh sau:



## Phân tích các chương trình đệ quy

- Chương trình đệ quy
  - Lập phương trình đệ quy  $T(n)$
  - Giải phương trình đệ quy tìm nghiệm
  - Suy ra tỷ suất tăng  $f(n)$  hay  $O(f(n))$

Ví dụ:

```
1. int giao_thua(int n) {  
2.     if (n == 0)  
3.         return 1;  
4.     else  
5.         return n * giao_thua(n - 1);  
6. }
```

## Thành lập phương trình đệ quy

- ◆ Thông thường đối với một chương trình đệ quy để giải bài toán kích thước  $n$ ,
  - phải có ít nhất một trường hợp dừng ứng với một  $n$  cụ thể
  - và lời gọi đệ quy để giải bài toán kích thước  $k$  ( $k < n$ ).

## Thành lập phương trình đệ quy

- ◆ Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa  $T(n)$  và  $T(k)$ , trong đó
  - $T(n)$  là thời gian thực hiện chương trình với kích thước dữ liệu nhập là  $n$ ,
  - $T(k)$  thời gian thực hiện chương trình với kích thước dữ liệu nhập là  $k$ , với  $k < n$ .
  - Để thành lập được phương trình đệ quy, ta phải căn cứ vào chương trình đệ quy.

## Thành lập phương trình đệ quy

- ◆ Để thành lập phương trình đệ quy, ta gọi
  - $T(n)$  là thời gian để giải bài toán kích thước  $n$
  - $T(k)$  là thời gian để giải bài toán kích thước  $k$
- ◆ Khi đệ quy dừng, ta phải xem xét khi đó chương trình làm gì và tốn hết bao nhiêu thời gian, chẳng hạn thời gian này là  $c(n)$
- ◆ Khi đệ quy chưa dừng thì phải xem có bao nhiêu lời gọi đệ quy với kích thước  $k$  ta sẽ có bấy nhiêu  $T(k)$ 
  - Ngoài ra ta còn phải xem xét đến thời gian để phân chia bài toán và tổng hợp các lời giải, chẳng hạn thời gian này là  $d(n)$ .

## Thành lập phương trình đệ quy

- ◆ Dạng tổng quát của một phương trình đệ quy sẽ là:

$$T(n) = \begin{cases} C(n) \\ F(T(k)) + d(n) \end{cases}$$

- ◆ Trong đó:

- $C(n)$  là thời gian thực hiện chương trình ứng với trường hợp đệ quy dừng
- $F(T(k))$  là một đa thức của các  $T(k)$ .
- $d(n)$  là thời gian để phân chia bài toán và tống hợp các kết quả.

```
int Giai_thua (int n) {  
    if (n==0)    return 1;  
    else return n*Giai_thua(n-1);  
}
```

- ❖ Gọi  $T(n)$  là thời gian thực hiện việc tính  $n$  giai thừa
- ❖  $T(n-1)$  là thời gian thực hiện việc tính  $n-1$  giai thừa
- ❖ Trường hợp  $n=0$  thì chương trình chỉ thực hiện một lệnh gán  $Giai\_thua:=1$  nên tốn  $O(1)$ , do đó ta có  $T(0) = C_1$
- ❖ Trường hợp  $n>0$  chương trình phải gọi đệ quy  $Giai\_thua(n-1)$ , việc gọi đệ quy này tốn  $T(n-1)$ 
  - Sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với  $n$  và gán cho  $Giai\_thua$
  - Thời gian để thực hiện phép nhân và phép gán là một hằng  $C_2$ .
- ❖ Vậy ta có phương trình đệ quy để tính thời gian thực hiện của chương trình đệ quy  $Giai\_thua$  là:  
$$T(n) = \begin{cases} C_1 & \text{nếu } n = 0 \\ T(n-1) + C_2 & \text{nếu } n > 0 \end{cases}$$

## Thành lập phương trình đệ quy

- ◆ **Ví dụ:** Xét hàm tính giai thừa viết bằng giải thuật đệ quy như sau:

```
int Giai_thua (int n) {  
    if (n==0)    return 1;  
    else  
        return n*Giai_thua(n-1);  
}
```

## Giải phương trình đệ quy

- ◆ Có ba phương pháp giải phương trình đệ quy:
  1. Phương pháp truy hồi
  2. Phương pháp đoán nghiệm.
  3. Lời giải tổng quát của một lớp các phương trình đệ quy.

## Giải phương trình đệ quy

- Phương pháp truy hồi
  - Triển khai  $T(n)$  theo  $T(n - 1)$  rồi  $T(n - 2)$  ... cho đến  $T(1)$  hoặc  $T(0)$
  - Suy ra nghiệm
- Phương pháp đoán nghiệm
  - Dự đoán nghiệm  $f(n)$
  - Áp dụng định nghĩa tỷ suất tăng và chứng minh  $f(n)$  là tỷ suất tăng của  $T(n)$
- Lời giải tổng quát cho một lớp các phương trình đệ quy

## Phương pháp truy hồi

- Triển khai  $T(n)$  theo  $T(n-1)$  rồi đến  $T(n-2)$  tiếp đến ... cho đến  $T(1)$

## Phương pháp truy hồi

- ◆ Ví dụ: Giải phương trình

$$T(n) = \begin{cases} C_1 & \text{nếu } n = 0 \\ T(n-1) + C_2 & \text{nếu } n > 0 \end{cases}$$

- ◆ Ta có  $T(n) = T(n-1) + C_2$

$$T(n) = [T(n-2) + C_2] + C_2 = T(n-2) + 2C_2$$

$$T(n) = [T(n-3) + C_2] + 2C_2 = T(n-3) + 3C_2$$

.....

$$T(n) = T(n-i) + iC_2$$

- ◆ Quá trình trên kết thúc khi  $n - i = 0$  hay  $i = n$ . Khi đó ta có

$$T(n) = T(0) + nC_2 = C_1 + nC_2 = O(n)$$

## Phương pháp truy hồi

- ◆ Ví dụ: Giải phương trình

$$T(n) = \begin{cases} C_1 & \text{nếu } n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2 n & \text{nếu } n > 1 \end{cases}$$

- ◆ Ta có

$$T(n) = 2T\left(\frac{n}{2}\right) + C_2 n$$

$$T(n) = 2[2T\left(\frac{n}{4}\right) + C_2 \left(\frac{n}{2}\right)] + C_2 n = 4T\left(\frac{n}{4}\right) + 2C_2 n$$

$$T(n) = 4[2T\left(\frac{n}{8}\right) + C_2 \left(\frac{n}{4}\right)] + 2C_2 n = 8T\left(\frac{n}{8}\right) + 3C_2 n$$

.....

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + iC_2 n$$

- ◆ Quá trình kết thúc khi  $\frac{n}{2^i} = 1$  hay  $2^i = n$  và do đó  $i = \log_2 n$ . Khi đó ta có:

$$T(n) = nT(1) + \log_2 n C_2 n = C_1 n + C_2 n \log_2 n = O(n \log n).$$

- Thủ đoán 1 nghiệm  $f(n)$
- Sau đó tìm cách chứng minh  $T(n) \leq f(n)$ 
  - Chứng minh mình bằng quy nạp
- Thông thường ta chọn  $f(n)$  có dạng:  $n$ ,  $\log n$ ,  $n \log n$ ,  $2^n$ , ...

## Bài toán đệ quy tổng quát

- ◆ Để giải một bài toán kích thước  $n$ , ta chia bài toán đã cho thành  $a$  bài toán con, mỗi bài toán con có kích thước  $n/b$ . Giải các bài toán con này và tổng hợp kết quả lại để được kết quả của bài toán đã cho.
- ◆ Với các bài toán con chúng ta cũng sẽ áp dụng phương pháp đó để tiếp tục chia nhỏ ra nữa cho đến các bài toán con kích thước 1. Kỹ thuật này sẽ dẫn chúng ta đến một giải thuật đệ quy.
- ◆ Giả thiết rằng mỗi bài toán con kích thước 1 lấy một đơn vị thời gian
- ◆ Giả thiết thời gian để chia bài toán kích thước  $n$  thành các bài toán con kích thước  $n/b$  và tổng hợp kết quả từ các bài toán con để được lời giải của bài toán ban đầu là  $d(n)$ .

- ◆ Trong mục này, chúng ta sẽ nghiên cứu các phần sau:
  - Bài toán đệ quy tổng quát.
  - Thành lập phương trình đệ quy tổng quát.
  - Giải phương trình đệ quy tổng quát.
  - Các khái niệm về nghiệm thuần nhất, nghiệm riêng và hàm nhán.
  - Nghiệm của phương trình đệ quy tổng quát khi  $d(n)$  là hàm nhán.
  - Nghiệm của phương trình đệ quy tổng quát khi  $d(n)$  không phải là hàm nhán.

## Thành lập phương trình đệ quy tổng quát

- ◆ Nếu gọi  $T(n)$  là thời gian để giải bài toán kích thước  $n$
- ◆ Thì  $T(n/b)$  là thời gian để giải bài toán con kích thước  $n/b$ .
- ◆ Khi  $n = 1$  theo giả thiết trên thì thời gian giải bài toán kích thước 1 là 1 đơn vị, tức là  $T(1) = 1$ .
- ◆ Khi  $n$  lớn hơn 1, ta phải giải đệ quy a bài toán con kích thước  $n/b$ , mỗi bài toán con tốn  $T(n/b)$  nên thời gian cho a lời giải đệ quy này là  $aT(n/b)$ .
- ◆ Ngoài ra ta còn phải tốn thời gian để phân chia bài toán và tổng hợp các kết quả, thời gian này theo giả thiết trên là  $d(n)$ . Vậy ta có phương trình đệ quy:

$$T(n) = \begin{cases} 1 & \text{nếu } n = 1 \\ aT\left(\frac{n}{b}\right) + d(n) & \text{nếu } n > 1 \end{cases}$$

## Giải phương trình đệ quy tổng quát

$$T(n) = aT\left(\frac{n}{b}\right) + d(n)$$

$$T(n) = \begin{cases} 1 & \text{neu } n=1 \\ aT\left(\frac{n}{b}\right) + d(n) & \text{neu } n>1 \end{cases}$$

$$T(n) = a \left[ aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right) \right] + d(n) = a^2 T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n)$$

$$\begin{aligned} T(n) &= a^2 \left[ aT\left(\frac{n}{b^3}\right) + d\left(\frac{n}{b^2}\right) \right] + ad\left(\frac{n}{b}\right) + d(n) \\ &= a^3 T\left(\frac{n}{b^3}\right) + a^2 d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n) \end{aligned}$$

.....

$$T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$

Nguyễn Thái Dư - AGU

49

## Nghiệm thuần nhất và nghiệm riêng

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

Nghiệm thuần nhất  
 $a^k = n^{\log_b a}$

Nghiệm riêng

Nghiệm của phương trình là: MAX(NTN,NR).

Nguyễn Thái Dư - AGU

51

## Giải phương trình đệ quy tổng quát (tt)

$$T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j d\left(\frac{n}{b^j}\right)$$

Giả sử  $n = b^k$ , quá trình suy rộng trên sẽ kết thúc khi  $i = k$ . Khi đó ta được:

$$T\left(\frac{n}{b^i}\right) = T\left(\frac{n}{b^k}\right) = T\left(\frac{b^k}{b^k}\right) = T(1) = 1$$

Thay vào trên ta có:

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d(b^{k-j})$$

Nguyễn Thái Dư - AGU

50

## Hàm nhân

◆ Một hàm  $f(n)$  được gọi là **hàm nhân** (multiplicative function) nếu  $f(m.n) = f(m).f(n)$  với mọi số nguyên dương  $m$  và  $n$ .

### Ví dụ:

- Hàm  $f(n) = n^k$  là một hàm nhân, vì  $f(m.n) = (m.n)^k = m^k.n^k = f(m).f(n)$ .
- Hàm  $f(n) = \log n$  không phải là một hàm nhân, vì  $f(n.m) = \log(n.m) = \log n + \log m \neq \log n \cdot \log m = f(n).f(m)$

Nguyễn Thái Dư - AGU

52

## Tính nghiệm riêng khi $d(n)$ là hàm nhân

◆ Khi  $d(n)$  là hàm nhân, ta có:

$$◆ d(b^{k-j}) = d(b \cdot b \cdot b \dots b) = d(b) \cdot d(b) \dots d(b) = [d(b)]^{k-j}$$

$$NR = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} a^j [d(b)]^{k-j} = [d(b)]^k \sum_{j=0}^{k-1} \left[ \frac{a}{d(b)} \right]^j = [d(b)]^k \left[ \frac{\frac{a}{d(b)}}{1 - \frac{a}{d(b)}} \right] - 1$$

$$\text{Hay } NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

## Ba trường hợp (tt)

$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

**Trường hợp 3:**  $a = d(b)$

Công thức trên không xác định nên ta phải tính trực tiếp nghiệm riêng:

$$NR = [d(b)]^k \sum_{j=0}^{k-1} \left[ \frac{a}{d(b)} \right]^j = a^k \sum_{j=0}^{k-1} 1 = a^k k \quad (\text{do } a = d(b))$$

Do  $n = b^k$  nên  $k = \log_b n$  và  $a^k = n^{\log_b a}$ .

Vậy  $NR$  là  $n^{\log_b a} \log_b n > NTN$ .

Do đó  $T(n)$  là  $O(n^{\log_b a} \log_b n)$ .

## Ba trường hợp

$$NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

◆ **Trường hợp 1:**  $a > d(b)$

Trong công thức trên ta có  $a^k > [d(b)]^k$ , theo quy tắc lấy độ phức tạp ta có  $NR$  là  $O(a^k) = O(n^{\log_b a}) = NTN$ .

Do đó  $T(n)$  là  $O(n^{\log_b a})$ .

$$T(n) = \begin{cases} 1 & \text{neu } n=1 \\ aT\left(\frac{n}{b}\right) + d(n) & \text{neu } n>1 \end{cases}$$

◆ **Trường hợp 2:**  $a < d(b)$

Trong công thức trên ta có  $[d(b)]^k > a^k$ , theo quy tắc lấy độ phức tạp ta có  $NR$  là  $O([d(b)]^k) = O(n^{\log_b d(b)}) > NTN$ .

Do đó  $T(n)$  là  $O(n^{\log_b d(b)})$ .

## Bài tập

• Giải các phương trình đệ quy sau với  $T(1) = 1$

$$1/- T(n) = 4T\left(\frac{n}{2}\right) + n$$

$$4/ \quad T(1) = 1$$

$$2/- T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

$$3/- T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

## Nghiệm của phương trình đệ quy tổng quát khi $d(n)$ không phải là hàm nhân

- Trong trường hợp hàm tiến triển không phải là một hàm nhân thì chúng ta không thể áp dụng các công thức ứng với ba trường hợp nói trên mà chúng ta phải tính trực tiếp NR, sau đó lấy MAX(NR, NTN).

## Ví dụ: GPT với $T(1) = 1$ và

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

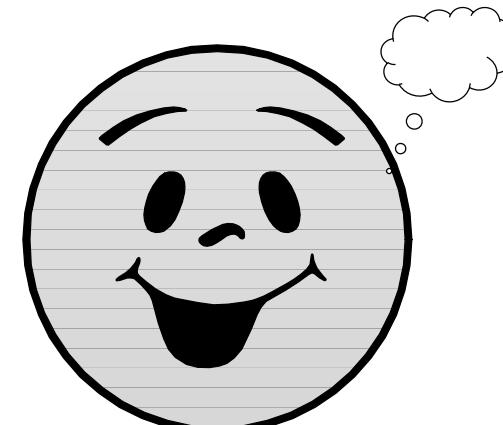
- PT thuộc dạng phương trình tổng quát nhưng  $d(n) = n\log n$  không phải là một hàm nhân.
- $NTN = n^{\log_b a} = n^{\log 2} = n$
- Do  $d(n) = n\log n$  không phải là hàm nhân nên ta phải tính nghiệm riêng bằng cách xét trực tiếp

## Ví dụ (tt)

$$NR = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j}$$
$$a = b = 2$$
$$d(n) = n\log n$$

$$NR = 2^k \sum_{j=0}^{k-1} (k-j) = 2^k \frac{k(k+1)}{2} = O(2^k k^2)$$

- Theo giải phương trình đệ quy tổng quát thì  $n = b^k$  nên  $k = \log_b n$ , ở đây do  $b = 2$  nên  $2^k = n$  và  $k = \log n$ ,
- $NR = O(n\log^2 n) > NTN$
- $T(n) = O(n\log^2 n)$ .



Cảm ơn !

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:  
**NGUYỄN THÁI DƯ**

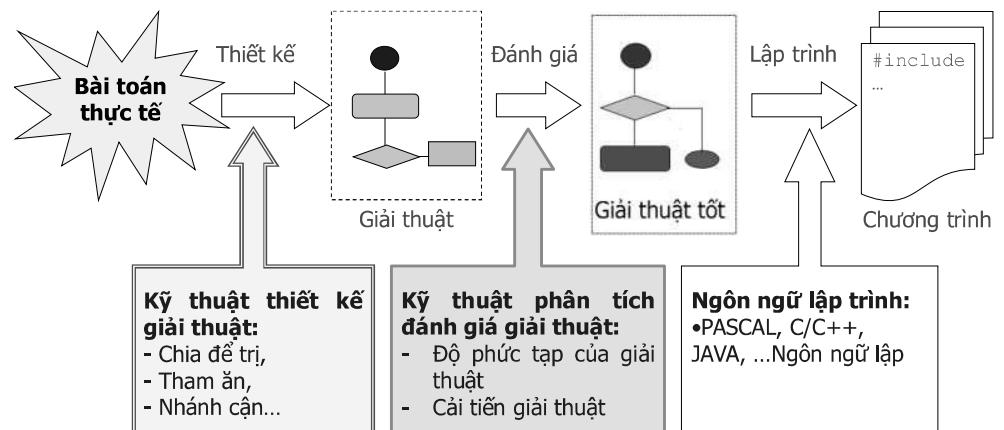
### Mục tiêu

- ◆ Biết các kỹ thuật thiết kế giải thuật: từ ý tưởng cho đến giải thuật chi tiết.
- ◆ Hiểu rõ nguyên lý của các kỹ thuật phân tích thiết kế giải thuật
- ◆ Vận dụng kỹ thuật phân tích thiết kế để giải các bài toán thực tế: các bài toán dạng nào thì có thể áp dụng kỹ thuật này.

## Chương 2. KỸ THUẬT THIẾT KẾ GIẢI THUẬT

- ◆ Mục tiêu
- ◆ Từ bài toán đến chương trình
- ◆ Các kỹ thuật thiết kế giải thuật
  - Chia để trị
  - Tham ăn (gready)
  - Quay lui
    - Vét cạn
    - Cắt tỉa Alpha-Beta
    - Nhánh cận

### Mô hình từ bài toán đến chương trình



## Kỹ thuật chia để trị

### ◆ Yêu cầu:

- Cần phải giải bài toán có kích thước n.

### ◆ Phương pháp:

- Ta chia bài toán ban đầu thành một số bài toán con đồng dạng với bài toán ban đầu có kích thước nhỏ hơn n.
- Giải các bài toán con được các lời giải con
- Tổng hợp lời giải con → ta có được lời giải của bài toán ban đầu.

### ◆ Chú ý

- Đối với từng bài toán con, ta lại chia chúng thành các bài toán con nhỏ hơn nữa.
- Quá trình phân chia này sẽ dừng lại khi kích thước bài toán **đủ nhỏ** mà ta có thể dễ dàng giải → gọi là **bài toán cơ sở**.

Nguyễn Thái Dư - AGU

5

## Kỹ thuật chia để trị

```
solve(n) {  
    if (n đủ nhỏ để có thể giải được)  
        giải bài toán → KQ  
        return KQ;  
    else {  
        Chia bài toán thành các bài toán con  
        kích thước n1, n2, ...  
  
        KQ1 = solve(n1); //giải bài toán con 1  
        KQ2 = solve(n2); //giải bài toán con 2  
        ...  
        Tổng hợp các kết quả KQ1, KQ2, ... → KQ  
        return KQ;  
    }  
}
```

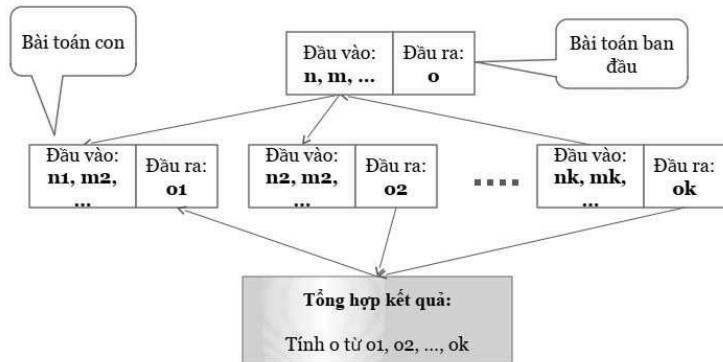
Nguyễn Thái Dư - AGU

7

## Kỹ thuật chia để trị

### ◆ Kỹ thuật chia để trị bao gồm hai quá trình:

- Phân tích bài toán đã cho thành các bài toán cơ sở
- **Tổng hợp** kết quả từ bài toán cơ sở để có lời giải của bài toán ban đầu



Nguyễn ... - AGU

6

## Nhìn lại giải thuật QuickSort và MergeSort

### ◆ Giải thuật QuickSort

- Sắp xếp dãy n số theo thứ tự tăng dần

### ◆ Áp dụng kỹ thuật chia để trị

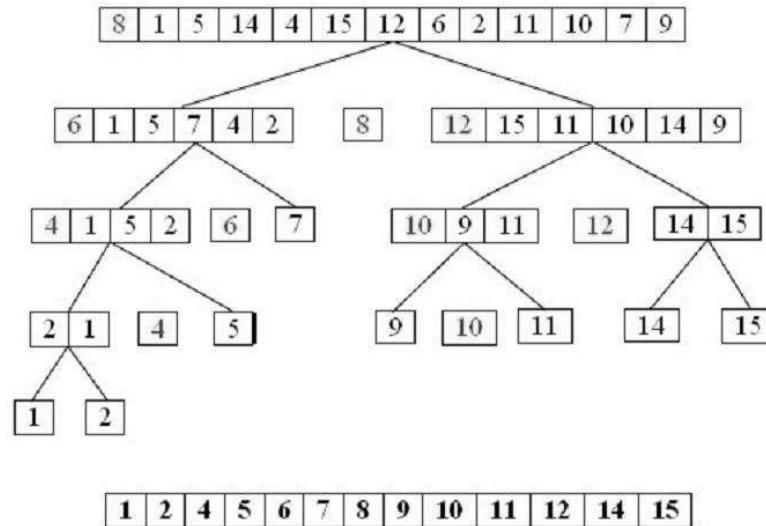
- **Phân chia:** Tìm một giá trị chốt và phân hoạch danh sách đã cho thành hai danh sách con “bên trái” và “bên phải”
  - Sắp xếp “bên trái” và “bên phải” thì ta được danh sách có thứ tự.
  - Bài toán cơ sở: Sắp xếp danh sách có 1 phần tử hoặc nhiều phần tử có giá trị giống nhau.
- **Tổng hợp:** đã bao hàm trong giai đoạn phân chia.

Nguyễn Thái Dư - AGU

8

## Nhìn lại giải thuật QuickSort và MergeSort

### ◆ Ví dụ QuickSort:



Ngu

1 2 4 5 6 7 8 9 10 11 12 14 15

## Độ phức tạp của QuickSort

### ◆ Xấu nhất

- Dãy n số đã đúng thứ tự tăng dần
- Phân hoạch bị lệch: phần tử chót là phần tử nhỏ nhất  
=> cần n phép so sánh để biết nó là phần tử đầu tiên
- Độ phức tạp trong trường hợp này là: **O(n<sup>2</sup>)**

### ◆ Tốt nhất:

- Phân hoạch cân bằng: phần tử chót là phần tử giữa dãy  
=> C(n) = 2C(n/2) + n
- Độ phức tạp trong trường hợp này là: **O(nlogn)**

## Nhìn lại giải thuật QuickSort và MergeSort

### ◆ Giải thuật MergeSort

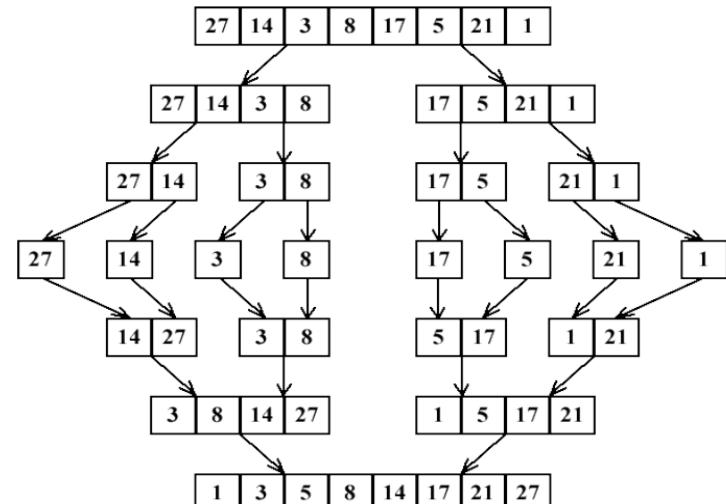
- Sắp xếp dãy n số theo thứ tự tăng dần

### ◆ Áp dụng kỹ thuật chia để trị

- Phân chia:** chia danh sách có n phần tử thành 2 danh sách có n/2 phần tử.
  - Quá trình phân chia sẽ dẫn đến các danh sách chỉ có 1 phần tử, là bài toán cơ sở.
- Tổng hợp:** trộn (merge) 2 danh sách có thứ tự thành một danh sách có thứ tự.

## Nhìn lại giải thuật QuickSort và MergeSort

### ◆ Ví dụ Merge Sort:



Ngu

11

Ngu

12

## Bài toán xếp lịch thi đấu thể thao

### ◆ Bài toán:

- Xếp lịch thi đấu vòng tròn 1 lượt cho n đấu thủ.
- Mỗi đấu thủ phải đấu với n-1 đấu thủ còn lại
- Mỗi đấu thủ chỉ đấu nhiều nhất là 1 trận mỗi ngày.

### ◆ Yêu cầu

- Xếp lịch sao cho số ngày thi đấu là ít nhất.

### ◆ Chia để trị

- chia
  - Xếp cho  $n/2, n/4, n/8, \dots$
  - Cuối cùng xếp cho 2 đấu thủ
- Tổng hợp

## Giải thuật chia để trị cho bài toán xếp lịch thi đấu

### ◆ Xuất phát từ bài toán cơ sở:

- Lịch thi đấu cho 2 đấu thủ 1 và 2 trong ngày thứ 1
- Như vậy ta có  $O(1,1) = "2"$  và  $O(2,1) = "1"$ .

2 đấu thủ

1	1
2	2
2	1

## Giải thuật chia để trị cho bài toán xếp lịch thi đấu

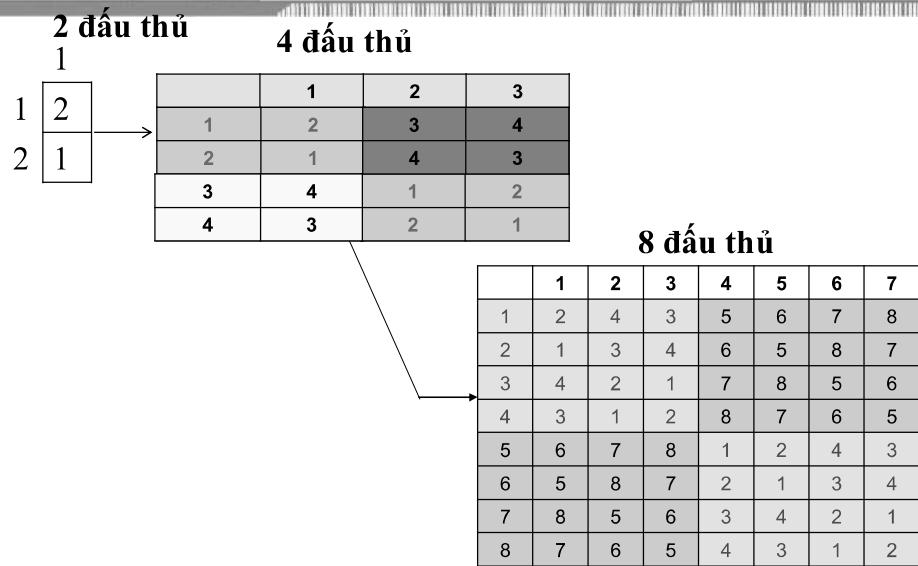
- ◆ Lịch thi đấu là 1 bảng gồm n dòng (tương ứng với n đấu thủ) và n-1 cột (tương ứng với n-1 ngày). Ô  $(i,j)$  biểu diễn đấu thủ mà i phải đấu trong ngày j.
- ◆ Chia để trị: thay vì xếp cho n người, ta sẽ xếp cho  $n/2$  người sau đó dựa trên kết quả của lịch thi đấu của  $n/2$  người ta xếp cho n người.
- ◆ Quá trình phân chia sẽ dừng lại khi ta phải xếp lịch cho 2 đấu thủ. Việc xếp lịch cho 2 đấu thủ rất dễ dàng: ta cho 2 đấu thủ này thi đấu 1 trận trong 1 ngày.
- ◆ Bước khó khăn nhất chính là bước xây dựng lịch cho 4, 8, 16, ... đấu thủ từ lịch thi đấu của 2 đấu thủ.

## Giải thuật chia để trị cho bài toán xếp lịch thi đấu

- ◆ Xuất phát từ lịch thi đấu cho hai đấu thủ ta có thể xây dựng lịch thi đấu cho 4 đấu thủ như sau:

- Lịch thi đấu cho 4 đấu thủ sẽ là một bảng 4 dòng, 3 cột.
- Lịch thi đấu cho 2 đấu thủ 1 và 2 trong ngày thứ 1 chính là lịch thi đấu của hai đấu thủ (**bài toán cơ sở**).
- Như vậy ta có  $O(1,1) = "2"$  và  $O(2,1) = "1"$ .
- Tương tự ta có lịch thi đấu cho 2 đấu thủ 3 và 4 trong ngày thứ 1. Nghĩa là  $O(3,1) = "4"$  và  $O(4,1) = "3"$ .
- Bây giờ để hoàn thành lịch thi đấu cho 4 đấu thủ, ta lấy góc trên bên trái của bảng lấp vào cho góc dưới bên phải và lấy góc dưới bên trái lấp cho góc trên bên phải.

## Xây dựng lịch thi đấu



## Bài toán con cân bằng

- ◆ Sẽ tốt hơn nếu ta chia bài toán cần giải thành các bài toán con có kích thước gần bằng nhau.
- ◆ Ví dụ:
  - **MergeSort** phân chia bài toán thành hai bài toán con có cùng kích thước  $n/2$  và do đó thời gian của nó chỉ là  $O(n\log n)$ .
  - Ngược lại trong trường hợp xấu nhất của **QuickSort**, khi mảng bị phân hoạch lệch thì thời gian thực hiện là  $O(n^2)$ .
- ◆ Nguyên tắc chung: Chia bài toán thành các bài toán con có kích thước xấp xỉ bằng nhau thì hiệu suất sẽ cao hơn.

## Bài toán tối ưu tổ hợp

- Cho hàm  $f(X)$ , là hàm mục tiêu, xác định trên một tập hữu hạn các phần tử  $D$ .
- Mỗi phần tử  $X \in D$  có dạng  $X = (x_1, x_2, \dots, x_n)$  được gọi là một phương án.
- Cần tìm một phương án  $X^* \in D$  sao cho  $f(X^*)$  đạt min/max.
  - Phương án  $X^*$  như thế được gọi là phương án tối ưu.
- Có nhiều phương pháp để giải
- Phương pháp “vết cạn” cần một thời gian mű.

## Kỹ thuật “tham ăn”/“háu ăn” (Greedy)

- ◆ Đây là một kỹ thuật được dùng nhiều để giải các bài toán tối ưu tổ hợp.
- ◆ Áp dụng kỹ thuật này tuy không cho chúng ta lời giải tối ưu nhưng sẽ cho một lời giải “tốt”; bù lại chúng ta được lợi khá nhiều về thời gian.

## Kỹ thuật “tham ăn”/“háu ăn” (Greedy)

### ◆ Phương pháp Greedy:

- Giải bài toán tối ưu tổ hợp: xây dựng một phương án X.
- Phương án X được xây dựng bằng cách:
  - Sắp xếp các lựa chọn cho mỗi bước theo thứ tự nào đó “có lợi” (tăng dần hoặc giảm dần tùy theo cách lập luận)
  - Lựa chọn từng  $X_i$  cho đến khi đủ n thành phần  
$$X = (x_1, x_2, \dots x_n)$$
  - Với mỗi  $X_i$ , ta sẽ chọn  $X_i$  tối ưu.  
→ Với cách này thì có thể ở bước cuối cùng ta không còn gì để chọn mà phải chấp nhận một giá trị cuối cùng còn lại.

◆ Kỹ thuật Greedy: thường chọn một khả năng mà xem như tốt nhất tại lúc đó. → Tức là, giải thuật chọn một khả năng tối ưu cục bộ với hy vọng sẽ dẫn đến một lời giải tối ưu toàn cục.

## Bài toán trả tiền của máy rút tiền tự động ATM

### Ví dụ: Máy rút tiền ATM

- ◆ Gọi  $X = (X_1, X_2, X_3, X_4)$  là một phương án trả tiền.
- $X_1$  là số tờ giấy bạc 100.000 đồng,
  - $X_2$  là số tờ giấy bạc 50.000 đồng,
  - $X_3$  là số tờ giấy bạc 20.000 đồng,
  - $X_4$  là số tờ giấy bạc 10.000 đồng.

→ Mục tiêu là  $X_1 + X_2 + X_3 + X_4$  đạt nhỏ nhất với ràng buộc là:

$$X_1 * 100.000 + X_2 * 50.000 + X_3 * 20.000 + X_4 * 10.000 = n.$$

## Bài toán trả tiền của máy rút tiền tự động ATM

### Ví dụ: Máy rút tiền ATM

- Trong máy ATM, có sẵn các loại tiền có mệnh giá 100.000 đồng, 50.000 đồng, 20.000 đồng và 10.000 đồng.
- Giả sử mỗi loại tiền đều có số lượng không hạn chế.
- Khách hàng cần rút một số tiền  $n$  đồng (tính chẵn đến 10.000 đồng, tức là  $n$  chia hết cho 10.000).
- Hãy tìm một phương án trả tiền sao cho trả đủ  $n$  đồng và số tờ giấy bạc phải trả là ít nhất.

## Bài toán trả tiền của máy rút tiền tự động ATM

### ◆ Ý tưởng:

- Để  $(X_1 + X_2 + X_3 + X_4)$  nhỏ nhất thì các tờ giấy bạc mệnh giá lớn phải được chọn nhiều nhất.
- Trước hết ta chọn tối đa các tờ giấy bạc 100.000 đồng, nghĩa là  $X_1$  là số nguyên lớn nhất sao cho  $X_1 * 100.000 \leq n$ . Tức là  $X_1 = n \text{ DIV } 100.000$ .
- Xác định số tiền cần rút còn lại là hiệu  $n - X_1 * 100000$
- Chuyển sang chọn loại giấy bạc 50.000 đồng, và cứ tiếp tục như thế cho các loại mệnh giá khác...

## Bài toán trả tiền của máy rút tiền tự động ATM

Ví dụ: Khách hàng cần rút 1.290.000 đồng

$n = 1.290.000$ , phương án trả tiền như sau:

- $X_1 = 1.290.000 \text{ DIV } 100.000 = 12$ 
  - Số tiền còn lại là  $1.290.000 - 12 * 100.000 = 90.000$
- $X_2 = 90.000 \text{ DIV } 50.000 = 1$ 
  - Số tiền còn lại là  $90000 - 1 * 50000 = 40000$
- $X_3 = 40.000 \text{ DIV } 20.000 = 2$ 
  - Số tiền còn lại là  $40.000 - 2 * 20.000 = 0$
- $X_4 = 0 \text{ DIV } 10.000 = 0$
- Vậy ta có  $X = (12, 1, 2, 0)$  tức là máy ATM sẽ trả cho khách hàng 12 tờ 100.000 đồng, 1 tờ 50.000 đồng và 2 tờ 20.000 đồng.

Nguyễn Thái Dư - AGU

25

## Mô hình hóa Bài toán cái ba lô

◆ Gọi  $X=(X_1, X_2, \dots, X_n)$  với  $X_i$  là số nguyên không âm, là một phương án.  $X_i$  là số đồ vật thứ  $i$ .

◆ Cần tìm  $X$  sao cho:

- $X_1g_1 + X_2g_2 + \dots + X_ng_n \leq W$
- $F(X) = X_1v_1 + X_2v_2 + \dots + X_nv_n \rightarrow \text{Max}$

Nguyễn Thái Dư - AGU

27

## Kỹ thuật tham ăn - Bài toán cái ba lô

### Bài toán cái ba lô

- ◆ Cho một cái ba lô có thể đựng một trọng lượng  $W$
- ◆ Có  $n$  loại đồ vật (tất cả các loại đồ vật đều có số lượng không hạn chế), mỗi đồ vật  $i$  có một
  - Trọng lượng  $g_i$
  - Giá trị  $v_i$ .
- ◆ **Vấn đề:** Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các loại đồ vật nào, mỗi loại lấy bao nhiêu sao cho tổng trọng lượng không vượt quá  $W$  và tổng giá trị là lớn nhất.

Nguyễn Thái Dư - AGU

26

## Bài toán cái ba lô

❖ Áp dụng kỹ thuật Greedy

1. Tính đơn giá cho các loại đồ vật.
2. Xét các loại đồ vật theo **thứ tự đơn giá từ lớn đến nhỏ** (giảm dần)
3. Với mỗi đồ vật được xét sẽ lấy một số lượng tối đa mà trọng lượng còn lại của ba lô cho phép.
4. Xác định trọng lượng còn lại của ba lô và quay lại bước 3 cho đến khi không còn có thể chọn được đồ vật nào nữa.

Nguyễn Thái Dư - AGU

28

## Bài toán cái ba lô

◆ **Ví dụ:** Ta có một ba lô có trọng lượng là 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng bên dưới:

Loại đồ vật	Trọng lượng	Giá trị
A	15	30
B	10	25
C	2	2
D	4	6

## Bài toán cái ba lô

Loại đồ vật	Trọng lượng	Giá trị	Đơn giá
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

❖ Theo bảng thứ tự ưu tiên là **B, A, D** và **C**:

- Vật B, chọn tối đa là 3. Mỗi vật B có trọng lượng là 10  
→ Trọng lượng còn lại của ba lô là  $37 - 3*10 = 7$ .
- Vật A, Không chọn được vì trọng lượng vật A là 15 trong khi ba lô chỉ còn 7.
- Vật D, chọn được 1. → Trọng lượng còn lại của ba lô:  $7-4 = 3$ .
- Vật C, chọn được 1
- Tổng trọng lượng là:  $3*10 + 0 + 1*4 + 1*2 = 36$
- Tổng giá trị là:  $3*25 + 0 + 1*6 + 1*2 = 83$

## Bài toán cái ba lô

◆ Từ bảng trên ta tính đơn giá và sắp lại theo đơn giá

Loại đồ vật	Trọng lượng	Giá trị	Đơn giá
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

## Biến thể của bài toán cái ba lô

◆ Có một số biến thể của bài toán cái ba lô như sau:

- Mỗi đồ vật **i** chỉ có một số lượng  $s_i$ .
  - Với bài toán này khi lựa chọn vật **i** ta không được lấy một số lượng vượt quá  $s_i$ .
- Mỗi đồ vật chỉ có một cái.
  - Với bài toán này thì với mỗi đồ vật ta chỉ có thể chọn hoặc không chọn.

## Kỹ thuật quay lui (backtracking)

- ◆ Kỹ thuật quay lui (backtracking) là một quá trình phân tích đi xuống và quay lui trở lại theo con đường đã đi qua.
- ◆ Tại mỗi bước phân tích chúng ta chưa giải quyết được vấn đề do còn thiếu dữ liệu nên cứ phải phân tích cho tới các điểm dừng, nơi chúng ta xác định được lời giải của chúng hoặc là xác định được là không thể (*hoặc không nên*) tiếp tục theo hướng này.
- ◆ Từ các điểm dừng này chúng ta quay ngược trở lại theo con đường mà chúng ta đã đi qua để giải quyết các vấn đề còn tồn đọng và cuối cùng ta sẽ giải quyết được vấn đề ban đầu.

Nguyễn Thái Dư - AGU

33

## Kỹ thuật quay lui (backtracking)

- ◆ Để giải bài toán A ta cần giải các bài toán con  $A_1, \dots, A_n$
- ◆ Một số bài toán con  $A_i$  chưa giải được ta phải đi giải các bài toán con  $A_{i1}, A_{i2}, \dots, A_{ik}$
- ◆ Sau đó quay lại giải  $A_i$
- ◆ Trong quá trình giải:
  - Giải tất cả các bài toán con  $\rightarrow$  vét cạn
  - một số bài toán con không cần giải ta bỏ qua  $\rightarrow$  cắt tỉa

Nguyễn Thái Dư - AGU

34

## Kỹ thuật quay lui (backtracking)

- Giải bài toán tối ưu
  - Tìm một **lời giải tối ưu** trong số các lời giải
  - Mỗi **lời giải** gồm thành *n* **thành phần**.
  - Quá trình xây dựng một lời giải được xem như quá trình tìm *n* thành phần. Mỗi thành phần được tìm kiếm trong 1 bước.
    - Các **bước** phải có **dạng giống nhau**.
    - Ở mỗi bước, ta có thể dễ dàng chọn lựa một thành phần.
  - Sau khi thực hiện đủ *n* bước ta được 1 lời giải

Nguyễn Thái Dư - AGU

35

## Kỹ thuật quay lui (backtracking)

- ◆ **2 kỹ thuật quay lui:**
  - “Vét cạn” (*brute force*) là kỹ thuật phải đi tới tất cả các điểm dừng rồi mới quay lui.
    - Tìm tất cả các lời giải
    - Độ phức tạp là thời gian lũy thừa
  - “Cắt tỉa Alpha-Beta” và “Nhánh-Cận” (*branch and bound*) là hai kỹ thuật cho phép không cần thiết phải đi tới tất cả các điểm dừng, mà chỉ cần đi đến một số điểm nào đó và dựa vào một số suy luận để có thể quay lui sớm.
    - Chỉ tìm lời giải có lợi
    - Cải tiến thời gian thực hiện

Nguyễn Thái Dư - AGU

36

## Kỹ thuật quay lui - Kỹ thuật vét cạn

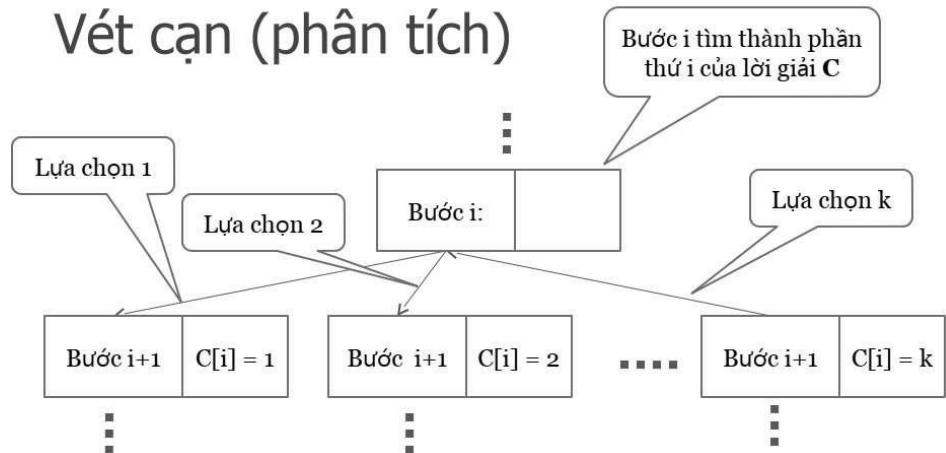
- **Ý tưởng:**
  - Gần giống chia để trị nhưng xây dựng lời giải từ dưới lên trong khi chia để trị là phân tích từ trên xuống
- **Một phương án/lời giải C:**
  - Gồm  $n$  thành phần  $C[1], C[2], \dots, C[n]$
- **Ở mỗi bước  $i$ , có một số lựa chọn cho thành phần  $i$ .**
  - Chọn một giá trị nào đó cho thành phần  $i$
  - Gọi đệ quy để tìm thành phần  $i + 1$
  - Hủy bỏ sự lựa chọn, quay lui lại bước 1 chọn giá trị khác cho thành phần  $i$
- **Chú ý:**
  - Quá trình đệ quy kết thúc khi  $i > n$
  - Khi tìm được lời giải, so sánh với các lời trước đó để chọn lời giải tối ưu

Nguyễn Thái Dư - AGU

37

## Kỹ thuật vét cạn

### Vét cạn (phân tích)



Nguyễn Thái Dư - AGU

38

## Kỹ thuật vét cạn

### ◆ Giải thuật vét cạn

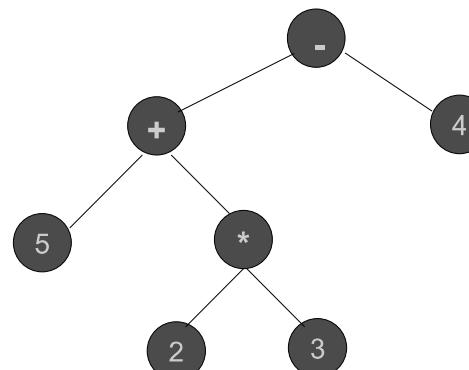
```
search(int i) {  
    if (i > n)  
        Kiểm tra, so sánh lời giải với các  
        lời giải hiện có ➔ Lời giải tối ưu  
    else {  
        for (j ∈ lựa chọn có thể có của bước i) {  
            C[i] = j; //Lựa chọn p/a j cho bước i  
            search(i + 1); //Gọi đệ quy  
            C[i] = null; //Hủy bỏ lựa chọn  
        }  
    }  
}
```

Nguyễn Thái Dư - AGU

39

## Kỹ thuật quay lui - Kỹ thuật vét cạn

- ◆ **Kỹ thuật vét cạn:** là kỹ thuật phải đi tới tất cả các điểm dừng rồi mới quay lui.
- ◆ **Ví dụ:** định trị cây biểu thức số học  $5 + 2 * 3 - 4$



Nguyễn Thái Dư - AGU

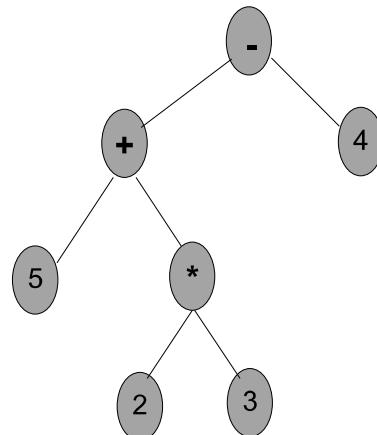
40

## Kỹ thuật Vét cạn – Định vị cây biểu thức số học

- ◆ Trong ngôn ngữ lập trình đều có các biểu thức số học, việc dịch các biểu thức này đòi hỏi phải đánh giá (định trị) chúng.
- ◆ Để làm được điều đó cần phải có một biểu diễn trung gian cho biểu thức.
- ◆ Một trong các biểu diễn trung gian cho biểu thức là cây biểu thức.

## Kỹ thuật Vét cạn – Định vị cây biểu thức số học

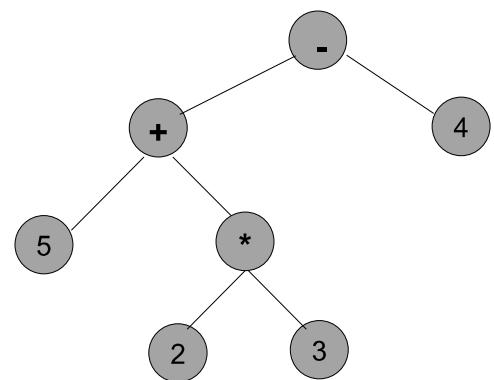
- ◆ Trị của nút lá chính là trị của toán hạng mà nút đó biểu diễn
- ◆ Trị của một nút trong có được bằng cách lấy toán tử mà nút đó biểu diễn áp dụng vào các con của nó.
- ◆ Trị của nút gốc chính là trị của biểu thức.



## Kỹ thuật Vét cạn – Định vị cây biểu thức số học

- ◆ Cây biểu thức số học là một cây nhị phân, trong đó:
  - Các nút lá biểu diễn cho các toán hạng,
  - Các nút trong biểu diễn cho các toán tử.

- ◆ Biểu thức  $5 + 2 * 3 - 4$  sẽ được biểu diễn bởi cây trong hình bên

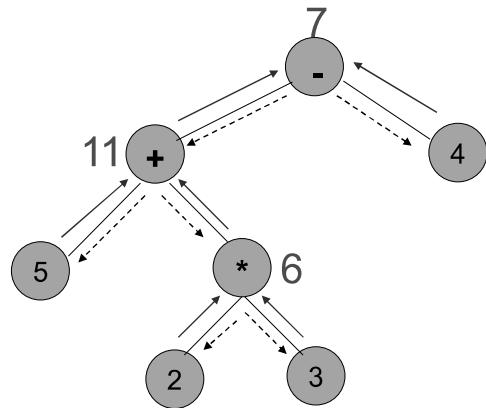


## Kỹ thuật Vét cạn – Định vị cây biểu thức số học

- ◆ Định trị cho nút gốc bằng cách:
  - Định trị cho hai con của nó,
  - đối với mỗi con ta xem nó có phải là nút lá hay không, nếu không phải ta lại phải xét hai con của nút đó.
- ◆ Quá trình cứ tiếp tục như vậy cho tới khi gặp các nút lá mà giá trị của chúng đã được biết,
  - quay lui để định trị cho các nút cha của các nút lá
  - và cứ như thế mà định trị cho tổ tiên của chúng.
- ◆ Đó chính là kỹ thuật quay lui vét cạn, vì chúng ta phải lùi đến tất cả các nút lá mới định trị được các nút trong và từ đó mới định trị được cho nút gốc.

## Kỹ thuật Vét cạn – Định vị cây biểu thức số học

Ví dụ: định trị cây biểu thức số học  $5 + 2 * 3 - 4$



### Quá trình định trị cây biểu thức số học

Nguyễn Thái Dư - AGU

45

## Kỹ thuật Vét cạn – Định vị cây biểu thức số học

- Giải thuật quay lui (vét cạn) cho việc định trị cây biểu thức số học

```
float Eval(node n) {  
    if (n là lá)  
        return (trị của toán hạng trong n);  
    else  
        return (Toán tử trong n (Eval (Con trái của n), Eval (Con phải của n)));  
}
```

- Muốn định trị cho cây biểu thức T, ta gọi Eval(ROOT(T)).

Nguyễn Thái Dư - AGU

46

## Kỹ thuật Vét cạn – Cây trò chơi – Mô tả

- Xét một trò chơi trong đó hai người thay phiên nhau đi nước của mình như cờ vua, cờ tướng, carô...
- Trò chơi có một trạng thái bắt đầu và mỗi nước đi sẽ biến đổi trạng thái hiện hành thành một trạng thái mới.
- Trò chơi sẽ kết thúc theo một quy định nào đó, theo đó thì cuộc chơi sẽ dẫn đến một trạng thái phản ánh:
  - có một người thắng cuộc
  - hoặc một trạng thái mà cả hai đấu thủ không thể phát triển được nước đi của mình, ta gọi nó là trạng thái hòa cờ.
- Ta tìm cách phân tích xem từ một trạng thái nào đó sẽ dẫn đến đấu thủ nào sẽ thắng với điều kiện cả hai đấu thủ đều có trình độ như nhau.

Nguyễn Thái Dư - AGU

47

## Kỹ thuật Vét cạn – Cây trò chơi – Biểu diễn

- Một trò chơi có thể được biểu diễn bởi một cây trò chơi.
- Mỗi một nút của cây biểu diễn cho một trạng thái.
- Nút gốc biểu diễn cho trạng thái bắt đầu của cuộc chơi.
- Mỗi nút lá biểu diễn cho một trạng thái kết thúc của trò chơi (trạng thái thắng, thua hoặc hòa).
- Nếu trạng thái x được biểu diễn bởi nút n thì các con của n biểu diễn cho tất cả các trạng thái kết quả của các nước đi có thể xuất phát từ trạng thái x.

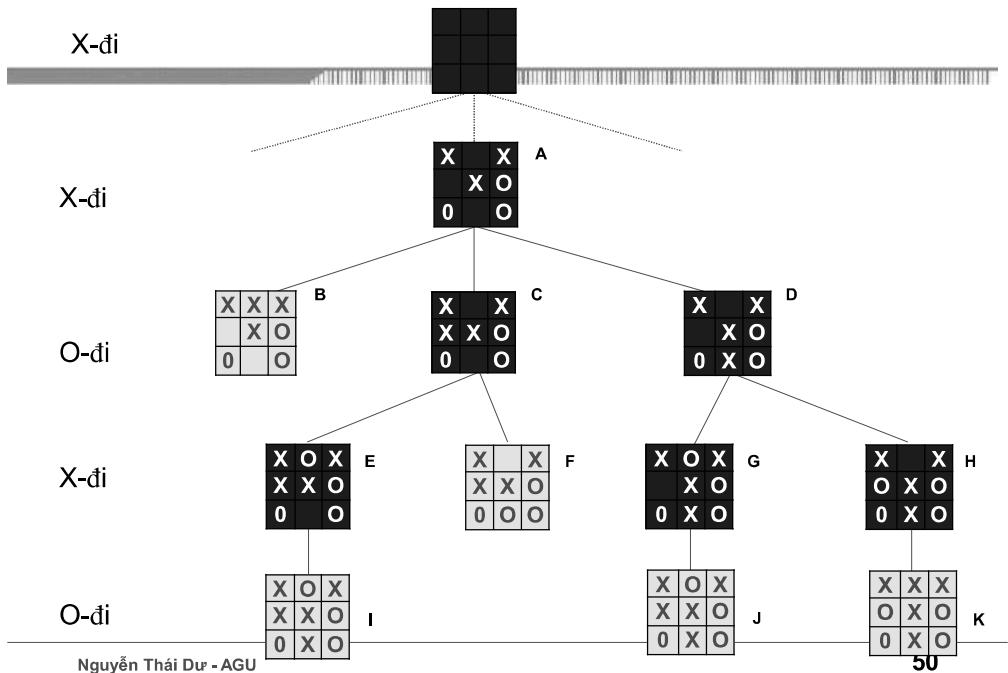
Nguyễn Thái Dư - AGU

48

## Kỹ thuật Vết cạn – Cây trò chơi

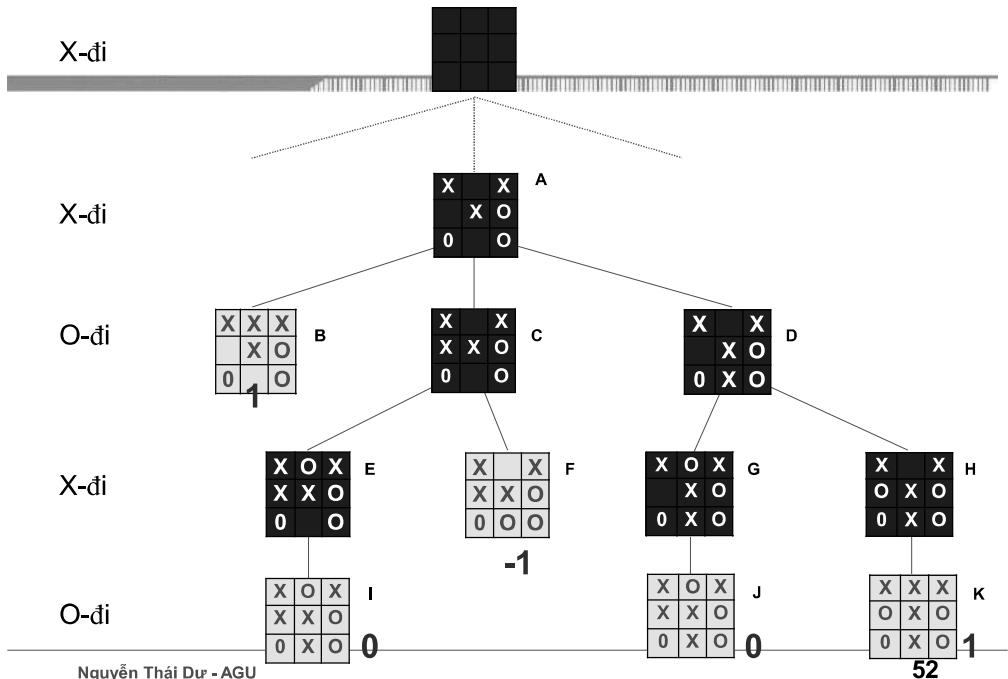
◆ **Ví dụ:** Xét trò chơi carô có 9 ô.

- Hai người thay phiên nhau đi X hoặc O.
- Người nào đi được 3 ô thẳng hàng (ngang, dọc, chéo) thì thắng cuộc.
- Nếu đã hết ô đi mà chưa phân thắng bại thì hai đấu thủ hòa nhau.
- Một phần của trò chơi này được biểu diễn bởi cây ở trang sau (*cây trò chơi carô 9 ô*)



## Kỹ thuật Vết cạn – Cây trò chơi

- ◆ Trong cây trò chơi, các nút lá được tô nền và viền khung đôi để dễ phân biệt với các nút khác.
- ◆ Ta gắn cho mỗi nút một chữ cái (A, B, C...) để tiện trong việc trình bày các giải thuật.
- ◆ Ta có thể gán cho mỗi nút lá một giá trị để phản ánh trạng thái thắng thua hay hòa của các đấu thủ. Chẳng hạn ta gán cho nút lá các giá trị như sau:
  - 1 nếu tại đó người đi X đã thắng,
  - -1 nếu tại đó người đi X đã thua và
  - 0 nếu hai đấu thủ đã hòa nhau.



## Kỹ thuật Vét cạn – Cây trò chơi

- ◆ Người đi X sẽ chọn cho mình một nước đi sao cho dẫn đến trạng thái có giá trị lớn nhất (trong trường hợp này là 1).
  - Ta nói X chọn nước đi MAX, nút mà từ đó X chọn nước đi của mình được gọi là nút MAX.
- ◆ Người đi O đến lượt mình sẽ chọn một nước đi sao cho dẫn đến trạng thái có giá trị nhỏ nhất (trong trường hợp này là -1, khi đó X sẽ thua và do đó O sẽ thắng).
  - Ta nói O chọn nước đi MIN, nút mà từ đó O chọn nước đi của mình được gọi là nút MIN.
- ◆ Do hai đấu thủ luân phiên nhau đi nước của mình nên các mức trên cây trò chơi cũng luân phiên nhau là MAX và MIN. Cây trò chơi vì thế còn có tên là cây MIN-MAX

## Kỹ thuật Vét cạn – Giải thuật định vị Cây trò chơi

```
float Search(NodeType n, ModeType mode) {  
    NodeType C; /*C là một nút con của nút n*/  
    float value;  
    if(is_leaf(n))  
        return Payoff(n);  
    if(mode == MAX) value = -∞;  
    else value = ∞;  
    for (với mỗi con C của n)  
        if(mode == MAX)  
            value = max(value, Search(C, MIN));  
        else  
            value = min(value, Search(C, MAX));  
    return value;  
}
```

/\*Khởi tạo giá trị tạm cho n  
Lúc đầu ta cho value một giá trị tạm,  
sau khi đã xét hết tất cả các con của nút  
n thì value là giá trị của nút n\*/

/\*Xét tất cả các con của n, mỗi lần xác định được  
giá trị của một nút con, ta  
phải đặt lại giá trị tạm  
value. Khi đã xét hết tất  
cả các con thì value là  
giá trị của n\*/

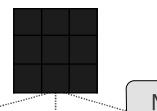
## Kỹ thuật Vét cạn – Cây trò chơi

- ◆ Ta có thể đưa ra một quy tắc định trị cho các nút trên cây để phản ánh tình trạng thắng thua hay hòa và khả năng thắng cuộc của hai đấu thủ.
- ◆ Nếu một nút là nút lá thì trị của nó là giá trị đã được gán cho nút đó.
- ◆ Ngược lại, nếu nút là nút MAX thì trị của nó bằng giá trị lớn nhất của tất cả các trị của các con của nó. Nếu nút là nút MIN thì trị của nó là giá trị nhỏ nhất của tất cả các trị của các con của nó.
- ◆ **Ví dụ:** Vận dụng quy tắc quay lui vét cạn để định trị cho nút A trong cây trò chơi trong ví dụ trước

## Kỹ thuật Vét cạn – Cây trò chơi

- ◆ Hàm Search nhận vào một nút n và kiểu mode của nút đó (MIN hay MAX) trả về giá trị của nút.
- ◆ Nếu nút n là nút lá thì trả về giá trị đã được gán cho nút lá.
- ◆ Ngược lại ta cho n một giá trị tạm value tương ứng như sau:
  - Nếu n là nút MAX thì gán value = -∞
  - Nếu n là nút MIN thì gán value = ∞
- ◆ Sau khi một con của n có giá trị V thì đặt lại value như sau:
  - Nếu n là nút MAX thì gán value = max(value, V)
  - Nếu n là nút MIN thì gán value = min(value, V)
- ◆ Khi tất cả các con của n đã được xét thì giá trị tạm value của n trở thành giá trị của nó

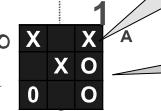
X-đi - Max



X thắng

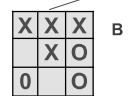
Max(0;1)=1

X-đi - Max

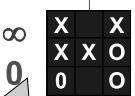


Max(-1;1)=1

O-đi - Min

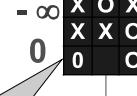


Max(-∞;1)=1

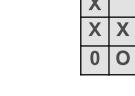


Min(0;1)=0

X-đi - Max



Max(-∞;0)=0



Min(-1;0;∞) = -1

O-đi - Min



Max(-∞;0)=0



Max(-∞;0)=0

Nguyễn Thái Dư - AGU



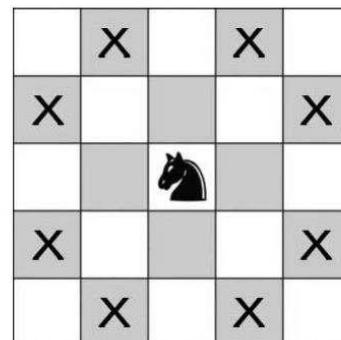
Max(-∞;0)=0



Max(-∞;0)=0

### Kỹ thuật vét cạn – Bài toán mã đi tuần

- Bàn cờ vua có kích thước 8x8
- In đường đi của mã trên khắp bàn cờ



### Kỹ thuật Vét cạn – Cây trò chơi

- ◆ Trong hình trên, các nút lá có giá trị gán ghi phía dưới mỗi nút.
- ◆ Đối với các nút trong, bên trái ghi các giá trị tạm theo thứ tự từ trên xuống, các giá trị thực được ghi bên phải hoặc phía trên bên phải

Nguyễn Thái Dư - AGU

58

### Kỹ thuật vét cạn – Bài toán mã đi tuần

- Bàn cờ vua có kích thước 8x8
- In 1 đường đi của mã trên khắp bàn cờ

1	48	31	50	33	16	63	18
30	51	46	3	62	19	14	35
47	2	49	32	15	34	17	64
52	29	4	45	20	61	36	13
5	44	25	56	9	40	21	60
28	53	8	41	24	57	12	37
43	6	55	26	39	10	59	22
54	27	42	7	58	23	38	11

Nguyễn Thái Dư - AGU

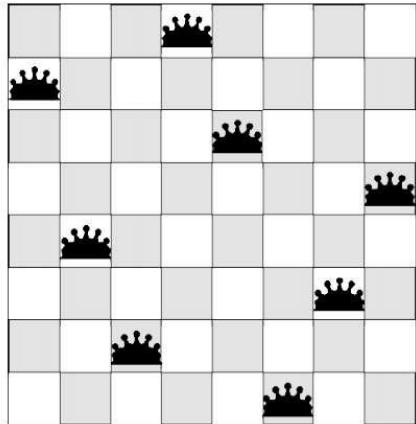
59

Nguyễn Thái Dư - AGU

60

## Kỹ thuật vét cạn – Bài toán 8 hậu

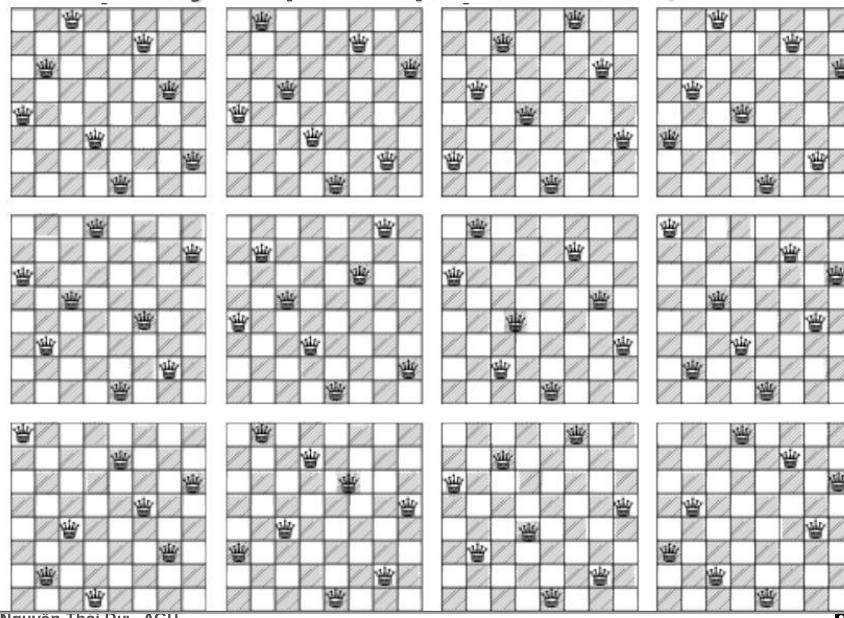
- Bàn cờ vua có kích thước 8x8
- Đặt 8 con hậu sao cho chúng không ăn nhau



Nguyễn Thái Dư - AGU

61

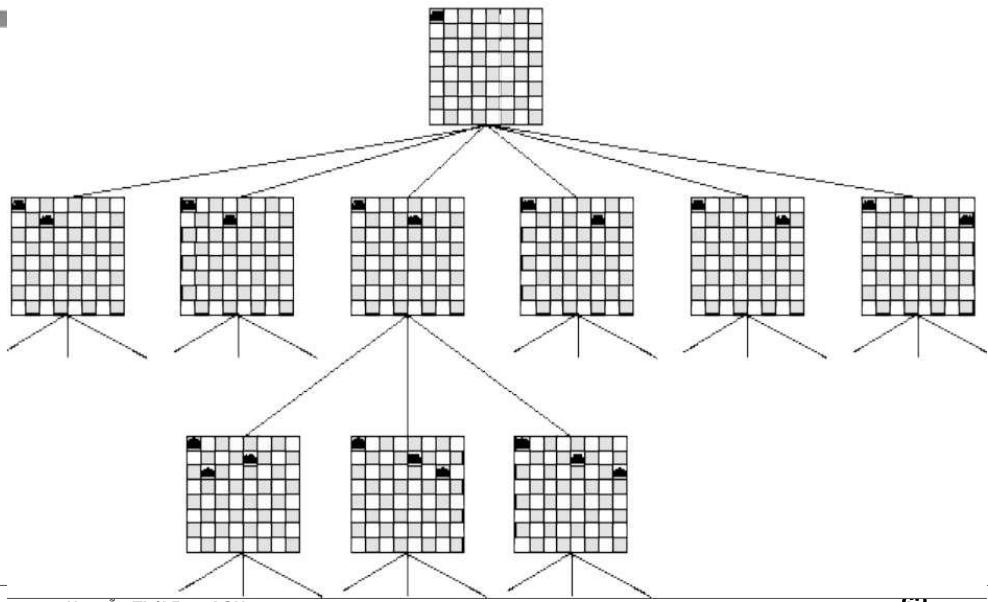
## Kỹ thuật vét cạn – Bài toán 8 hậu



Nguyễn Thái Dư - AGU

62

## Kỹ thuật vét cạn – Bài toán 8 hậu



Nguyễn Thái Dư - AGU

62

## Kỹ thuật cắt tỉa Alpha-Beta (Alpha-Beta Pruning)

- ◆ **Nhận xét:** trong giải thuật vét cạn ở trên:
  - Để định trị cho một nút nào đó,
    - ta phải định trị cho tất cả các nút con cháu của nó
    - và muốn định trị cho nút gốc ta phải định trị cho tất cả các nút trên cây.
  - Số lượng các nút trên cây trò chơi tuy hữu hạn nhưng không phải là ít.
    - Chẳng hạn trong cây trò chơi ca rô nói trên, nếu ta có bàn cờ bao gồm  $n$  ô thì có thể có tới  $n!$  nút trên cây (trong trường hợp trên là  $9!$ ).
  - Đối với các loại cờ khác như cờ vua chẳng hạn, thì số lượng các nút còn lớn hơn nhiều.
    - Ta gọi là một sự bùng nổ tổ hợp các nút.

Nguyễn Thái Dư - AGU

64

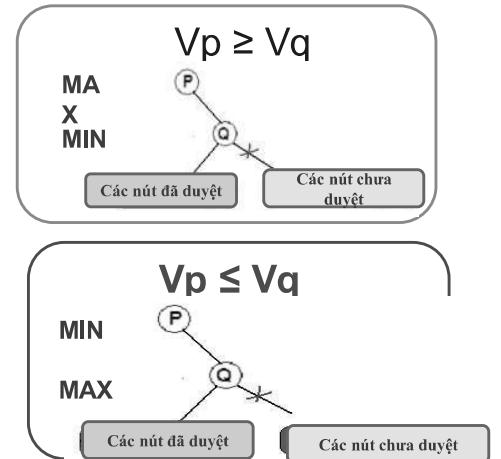
## Kỹ thuật cắt tỉa Alpha-Beta (Alpha-Beta Pruning)

**Ý tưởng cắt tỉa Alpha-Beta:** Khi định trị một nút thì không nhất thiết phải định trị cho tất cả các nút con cháu của nó.

◆ Nếu P là một nút MAX và ta đang xét một nút con Q của nó (đã nhiên Q là nút MIN).

- Giả sử  $V_p$  là một giá trị tạm của P,  $V_q$  là một giá trị tạm của Q và nếu ta có  $V_p \geq V_q$  thì ta không cần xét các con chưa xét của Q nữa.

◆ Nếu P là nút MIN (tất nhiên Q là nút MAX) và  $V_p \leq V_q$  thì ta cũng không cần xét đến các con chưa xét của Q nữa.



## Kỹ thuật cắt tỉa Alpha-Beta

◆ Quy tắc định trị cho một nút không phải là nút lá:

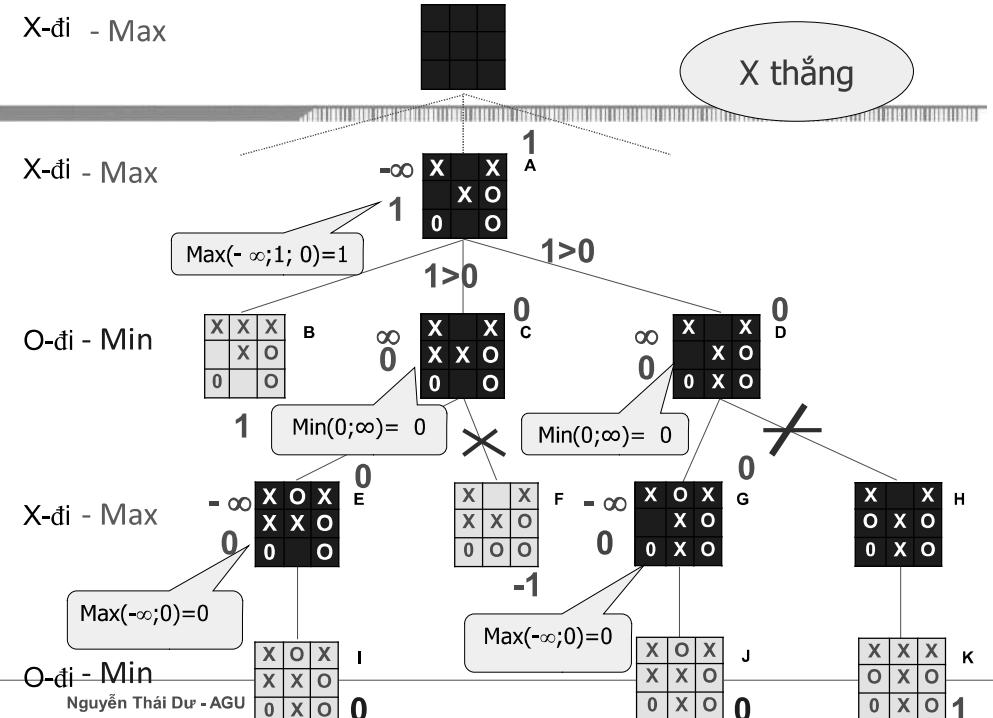
- Khởi đầu nút MAX có giá trị tạm là  $-\infty$  và nút MIN có giá trị tạm là  $\infty$ .
- Nếu tất cả các nút con của một nút đã được xét hoặc bị cắt tỉa thì giá trị tạm của nút đó trở thành giá trị của nó.
- Nếu một nút MAX n có giá trị tạm là  $V_1$  và một nút con của nó có giá trị là  $V_2$  thì đặt giá trị tạm mới của n là  $\max(V_1, V_2)$ . Nếu n là nút MIN thì đặt giá trị tạm mới của n là  $\min(V_1, V_2)$ .
- Vận dụng quy tắc cắt tỉa Alpha-Beta nói trên để hạn chế số lượng nút phải xét.

## Kỹ thuật cắt tỉa Alpha-Beta

● Giải thuật cắt tỉa Alpha-Beta định trị cây trò chơi

```
float cat_tia(NodeType Q, ModeType mode, float Vp) {
    NodeType C; /* C là một nút con của nút Q */
    float Vq;
    /* Vq là giá trị tạm của Q, sau khi tất cả các con của nút Q đã xét hoặc bị cắt tỉa thì Vq là giá trị của nút Q */
    if (is_leaf(Q)) return Payoff(Q);
    /* Khởi tạo giá trị tạm cho Q */
    if (mode == MAX) Vq = -∞;
    else Vq = ∞;
    /* Xét các con của Q, mỗi lần xác định được giá trị của một nút con của Q, ta phải đặt lại giá trị tạm Vq và so sánh với Vp để có thể cắt tỉa hay không */
}
```

```
/* Xét C là con trái nhất của Q; */
while (C là con của Q) {
    if (mode == MAX) {
        Vq = max(Vq, Cat_tia(C, MIN, Vq));
        if (Vp <= Vq) return Vq;
        /* cắt tỉa tất cả các con còn lại của Q */
    }
    else {
        Vq = min(Vq, Cat_tia(C, MAX, Vq));
        if (Vp >= Vq) return Vq;
    }
}
return Vq;
```



## Kỹ thuật nhánh cận – Bài toán cái Ba lô

- ◆ Nhánh cận là kỹ thuật xây dựng cây tìm kiếm phương án tối ưu, nhưng không xây dựng toàn bộ cây mà sử dụng giá trị cận để hạn chế bớt các nhánh.
- ◆ Với mỗi nút trên cây ta sẽ xác định một giá trị cận. Giá trị cận là một giá trị gần với giá của các phương án.
- ◆ Với bài toán tìm min ta sẽ xác định cận dưới còn với bài toán tìm max ta sẽ xác định cận trên.
  - Cận dưới là giá trị nhỏ hơn hoặc bằng giá của phương án,
  - ngược lại cận trên là giá trị lớn hơn hoặc bằng giá của phương án.

Nguyễn Thái Dư - AGU

69

## Kỹ thuật nhánh cận – Bài toán cái Ba lô

- ◆ Đây là bài toán tìm Max.
  - ◆ Danh sách các đồ vật được sắp xếp theo thứ tự giảm của đơn giá
1. Nút gốc biểu diễn cho trạng thái ban đầu của ba lô, ở đó ta chưa chọn một vật nào.
    - Tổng giá trị được chọn  $TGT = 0$ .
    - Cận trên của nút gốc  $CT = W * \text{Đơn giá lớn nhất}$ .

Nguyễn Thái Dư - AGU

70

## Kỹ thuật nhánh cận – Bài toán cái Ba lô

2. Nút gốc sẽ có các nút con tương ứng với các khả năng chọn đồ vật có đơn giá lớn nhất. Với mỗi nút con ta tính lại các thông số:
  - $TGT = TGT (\text{của nút cha}) + \text{số đồ vật được chọn} * \text{giá trị mỗi vật}$ .
  - $W = W (\text{của nút cha}) - \text{số đồ vật được chọn} * \text{trọng lượng mỗi vật}$ .
  - $CT = TGT + W * \text{Đơn giá của vật sẽ xét kế tiếp}$ .
3. Trong các nút con, ta sẽ ưu tiên phân nhánh cho nút con nào có cận trên lớn hơn trước. Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút ta lại phải xác định lại các thông số  $TGT$ ,  $W$ ,  $CT$  theo công thức đã nói trong bước 2.

Nguyễn Thái Dư - AGU

71

## Kỹ thuật nhánh cận – Bài toán cái Ba lô

4. Lặp lại bước 3 với chú ý: đối với những nút có cận trên nhỏ hơn hoặc bằng giá lớn nhất tạm thời của một phương án đã được tìm thấy thì ta không cần phân nhánh cho nút đó nữa (cắt bỏ).
5. Nếu tất cả các nút đều đã được phân nhánh hoặc bị cắt bỏ thì phương án có giá lớn nhất là phương án cần tìm.

Nguyễn Thái Dư - AGU

72

## Kỹ thuật nhánh cận – Bài toán cái Ba lô

- ◆ Ví dụ: Ta có một ba lô có trọng lượng là 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng bên dưới:

ĐV	TL	GT
A	15	30
B	10	25
C	2	2
D	4	6

ĐV: đồ vật

TL: Trọng lượng

ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

GT: Giá trị

ĐG: Đơn giá

Nguyễn Thái Dư - AGU

73

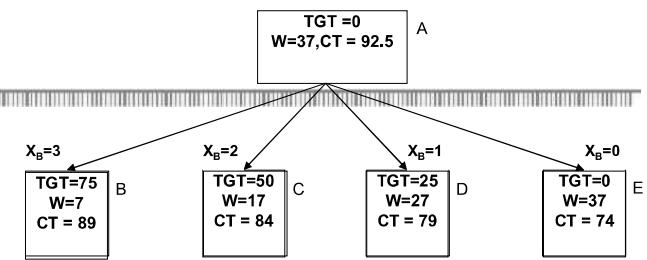
ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

\* Với nút B, ta có:

$$TGT = 0 + 3 * 25 = 75$$

$$W = 37 - 3 * 10 = 7$$

$$CT = 75 + 7 * 2 = 89$$



\* Với nút D, ta có:

$$TGT = 0 + 1 * 25 = 25$$

$$W = 37 - 1 * 10 = 27$$

$$CT = 25 + 27 * 2 = 79$$

\* Với nút E, ta có:

$$TGT = 0 + 0 * 25 = 0$$

$$W = 37 - 0 * 10 = 37$$

$$CT = 0 + 37 * 2 = 74$$

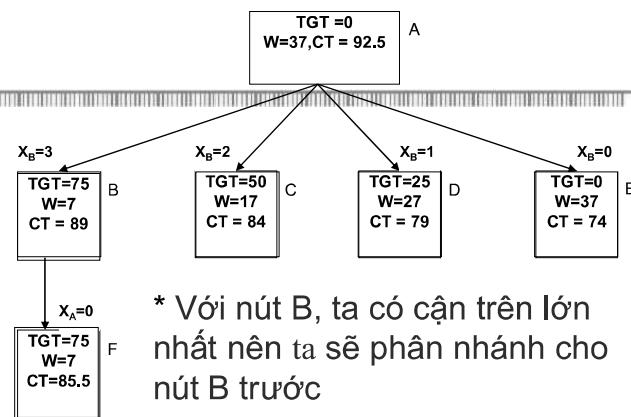
ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

\* Với nút F, ta có:

$$TGT = 75 + 0 * 30 = 75$$

$$W = 7 - 0 * 15 = 7$$

$$CT = 75 + 7 * 1.5 = 85,5$$



\* Với nút B, ta có cận trên lớn nhất nên ta sẽ phân nhánh cho nút B trước

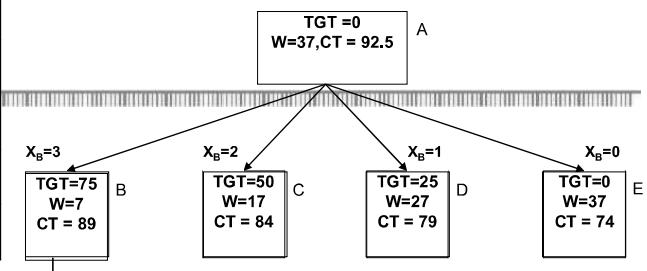
ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

\* Với nút G, ta có:

$$TGT = 75 + 1 * 6 = 81$$

$$W = 7 - 1 * 4 = 3$$

$$CT = 81 + 3 * 1 = 84$$



\* Với nút F, ta có cận trên lớn nhất nên ta tiếp tục phân nhánh cho nút F

\* Với nút H, ta có:

$$TGT = 75 + 0 * 6 = 75$$

$$W = 7 - 0 * 4 = 7$$

$$CT = 75 + 7 * 1 = 82$$

Nguyễn Thái Dư - AGU

75

76

ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

\* Với nút I, ta có:

$$TGT = 81 + 1 * 2 = 83$$

$$W = 3 - 1 * 2 = 1$$

\* Với nút J, ta có:

$$TGT = 81 + 0 * 2 = 81$$

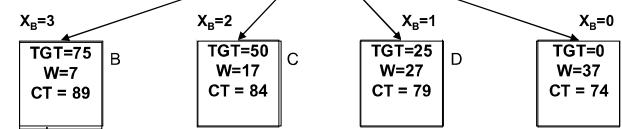
$$W = 3 - 0 * 2 = 3$$

Nguyễn Thái Dư - AGU

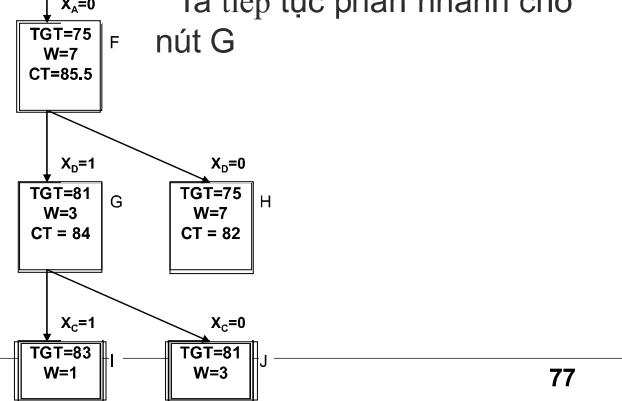
77

TGT = 0  
W=37, CT = 92.5

A



\* Ta tiếp tục phân nhánh cho nút G



Nút I và J là hai nút lá (biểu diễn cho phương án) vì với mỗi nút thì số đồ vật đã được chọn xong.

Nút I biểu diễn cho p.a

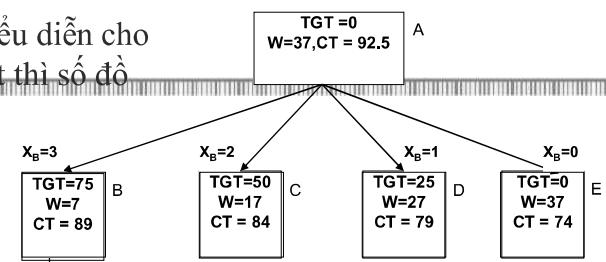
$$X_b=3, X_a=0, X_d=1, X_c=1$$

Giá: 83

Nút J biểu diễn cho p.a

$$X_b=3, X_a=0, X_d=1, X_c=0$$

Giá: 81T



Nguyễn Thái Dư - AGU

78

Như vậy giá trị lớn nhất tạm thời là 83

Quay lui lên nút H, ta thấy

cận trên của H là  $82 < 83$

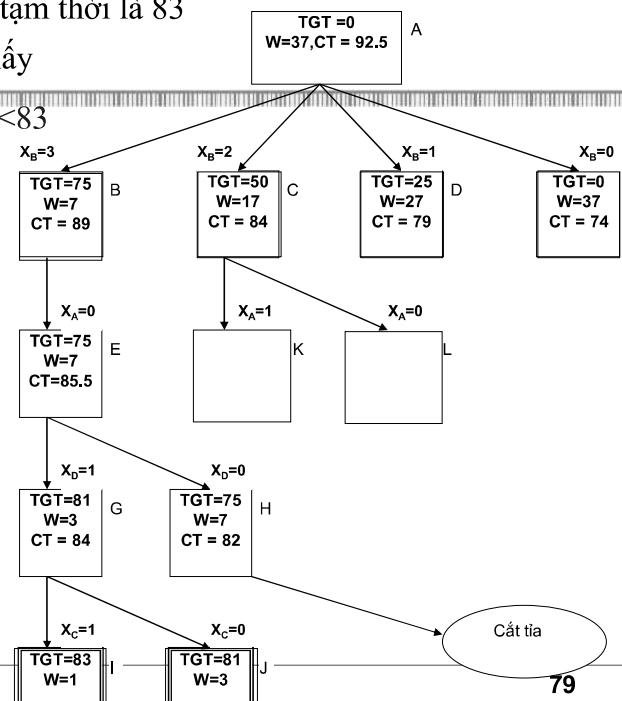
nên cắt tỉa nút H

Quay lui lên nút C

cận trên của C là  $84 > 83$

Tiếp tục phân nhánh

cho nút C



79

ĐV TL GT ĐG

B 10 25 2.5

A 15 30 2.0

D 4 6 1.5

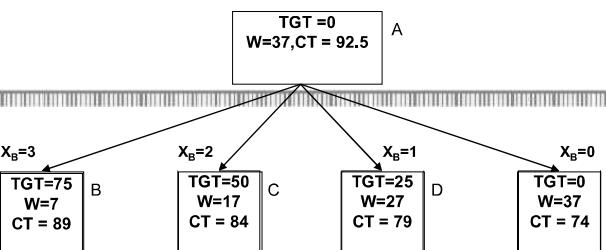
C 2 2 1.0

\* Với nút K, ta có:

$$TGT = 50 + 1 * 30 = 80$$

$$W = 17 - 1 * 15 = 2$$

$$CT = 80 + 2 * 1,5 = 83$$



Nguyễn Thái Dư - AGU

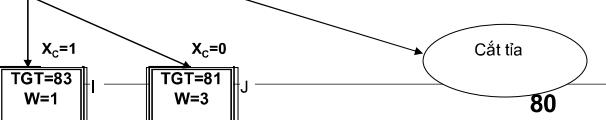
80

\* Với nút L, ta có:

$$TGT = 50 + 0 * 30 = 50$$

$$W = 17 - 0 * 15 = 17$$

$$CT = 50 + 17 * 1,5 = 75,5$$



Hai giá trị K, L đều không lớn hơn 83

Nên cả hai nút này đều bị cắt tỉa

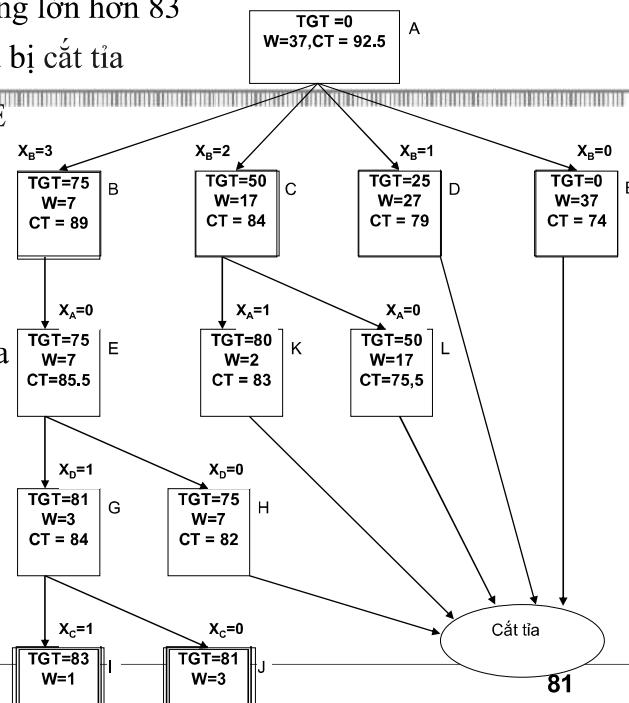
Tương tự các nút D và E

cũng bị cắt tỉa

Như vậy tất cả các nút

trên cây đều đã được

phân nhánh hoặc bị cắt tỉa



Nguyễn Thái Dư - AGU

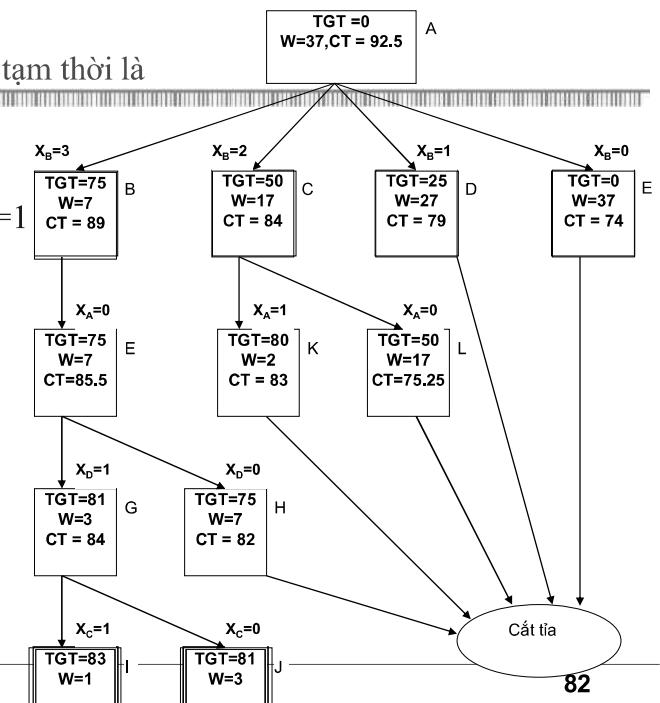
Vậy phương án tốt nhất tạm thời là  
phương án cần tìm

Theo đó ta cần chọn:

$X_A=3, X_B=0, X_D=1, X_C=1$

Tổng giá trị: 83

Tổng trọng lượng: 36



Nguyễn Thái Dư - AGU

## So sánh Vét cạn và Nhánh cạn

### Vét cạn

```
else {
    for (j ∈ LC của i) {
        C[i] = j;
        search(i + 1);
        C[i] = null;
    }
}
```

### Nhánh cạn

```
else {
    for (j ∈ LC của i)
        tính cận cho LC j
    S. xếp các LC theo cận
    for (j ∈ LC của i) {
        if (cận của j còn tốt) {
            C[i] = j;
            search (i + 1);
            C[i] = null;
        }
    }
}
```

## Quy hoạch động

### ◆ Mục đích:

- Cải tiến thuật toán chia để trị hoặc quay lui vét cạn để giảm thời gian thực hiện

### ◆ Ý tưởng:

- Lưu trữ các kết quả của các bài toán con trong BẢNG QUY HOẠCH (cơ chế caching)
- Đổi bộ nhớ lấy thời gian (trade memory for time)

## Quy hoạch động

### ◆ Thiết kế giải thuật bằng kỹ thuật QHD

- Phân tích bài toán dung kỹ thuật chia để trị/quay lui
  - Chia bài toán thành các bài toán con
  - Tìm mối quan hệ giữa kết quả của bài toán trước và kết quả của các bài toán con (*công thức truy hồi*)
- Lập bảng quy hoạch

## Quy hoạch động

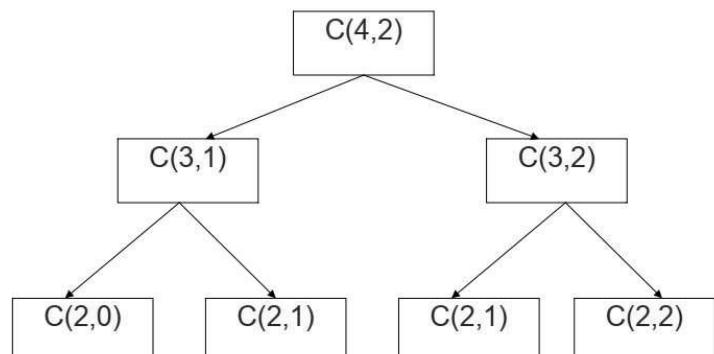
### ◆ Lập bảng quy hoạch

- Số chiều = số biến trong công thức truy hồi
- Thiết lập qui tắc đèn kết quả vào bảng quy hoạch
  - Điện các ô không phụ thuộc trước.
  - Điện các ô phụ thuộc sau.
- Tra bảng tìm kết quả (thường chỉ tìm được giá trị)
- Lần vết trên bảng để tìm lời giải tối ưu.

## Quy hoạch động

### ◆ Ví dụ: Tính tổ hợp

- $C(n,k) = 1$  nếu ( $n=k$ ) hoặc  $k=0$
- $C(n,k) = C(n-1, k-1) + C(n-1, k)$



## Quy hoạch động

### ◆ Tính tổ hợp

```
int comb(int n, int k) {  
    if((k == 0) || (k == n))  
        return 1;  
    else  
        return comb(n-1, k-1) + comb(n-1, k);  
}
```

## Quy hoạch động

### ◆ Độ phức tạp giải thuật đệ quy:

- T(n) là thời gian để tính số tổ hợp chập k của n, thi ta có phương trình đệ quy:

$$T(1) = C_1$$

$$T(n) = 2T(n-1) + C_2$$

=> Vậy độ phức tạp quá lớn:  $T(n) = O(2^n)$

## Quy hoạch động

### ◆ Tính tổ hợp

- Quy hoạch động:  $T(n) = O(n^2)$

	k					
n	0	1	2	3	4	
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	

## Quy hoạch động

```
int comb(int n, int k) {
    int c[maxlen][maxlen], i, j;
    c[0][0] = 1;
    for(i = 1; i<=n; i++) {
        c[i][0] = 1; c[i][i] = 1;
        for(j=1; j<i; j++)
            c[i][j] = c[i-1][j-1] + c[i-1][j];
    }
    return c[n][k];
}
```

## Quy hoạch động

```
int comb(int n, int k) {
    int c[maxlen], i, j, p1, p2;
    c[0] = 1; c[1] = 1;
    for(i = 2; i<=n; i++) {
        p1 = c[0];
        for(j=1; j<i; j++) {
            p2 = c[j];
            c[j] = p1 + p2;
            p1 = p2;
        }
        c[i] = 1;
    }
    return c[k];
}
```

## Chia để trị và Quy hoạch động

### Chia để trị

- Ý tưởng
  - Phân rã thành các bài toán con
  - Tổng hợp kết quả
- Giải thuật:
  - Độ quy từ trên xuống
  - Độ phức tạp thời gian lớn nếu có nhiều bài toán giống nhau
  - Không cần lưu trữ kết quả của tất cả các bài toán con

### Quy hoạch động

- Ý tưởng
  - Phân rã thành các bài toán con
  - Tìm mối quan hệ
- Giải thuật:
  - Lập bảng quy hoạch và giải từ dưới lên
  - Độ phức tạp thời gian nhỏ hơn nhờ sử dụng bảng quy hoạch
  - Cần bộ nhớ để lưu trữ bảng quy hoạch

## Kết luận

- ◆ Mỗi kỹ thuật chỉ phù hợp với 1 hoặc 1 số loại bài toán
- ◆ Mỗi kỹ thuật đều có ưu và khuyết điểm, không có kỹ thuật nào là “trị bá bệnh”
  - Kỹ thuật nhánh cành cần phải có cách ước lượng cận tốt mới mong cắt được nhiều nhánh
  - Kỹ thuật vét cạn có độ phức tạp thời gian quá cao (lũy thừa). Chỉ dùng khi n nhỏ hoặc khi không còn cách nào khác

## Kết hợp Quy hoạch động và đệ quy

- Sử dụng bảng quy hoạch để lưu kết quả bài toán con
- Không cần điền hết tất cả bảng quy hoạch
  - Điền bảng quy hoạch theo yêu cầu
    - Bắt đầu từ bài toán gốc
    - Nếu trong bảng quy hoạch chưa có KQ, gọi đệ quy để tìm kết quả và **lưu kết quả vào bảng quy hoạch**
    - Nếu KQ đã có trong bảng quy hoạch, sử dụng ngay kết quả này
  - Có thể sử dụng bảng băm để lưu trữ bảng quy hoạch

## Phân tích Bài toán Dãy Fibonaci

- ◆ Dãy số Fibonaci được cho bởi công thức:

$$\begin{cases} F_n = F_{n-1} + F_{n-2}, \forall n \geq 2 \\ F_0 = 0 \\ F_1 = 1 \end{cases}$$

- ◆ Xây dựng thuật toán tính Fn với n là 1 số nguyên nhập từ bàn phím

## Cách 1. giải bài toán Fibo bằng đệ quy

```
int fibonaci(int k)
{
    if(k>=2)
        return ( fibonaci(k-1) + fibonaci(k-2) );
    return 1;
}
```

Thuật toán trên hoàn toàn có thể cài đặt với 1 ngôn ngữ lập trình bất kỳ

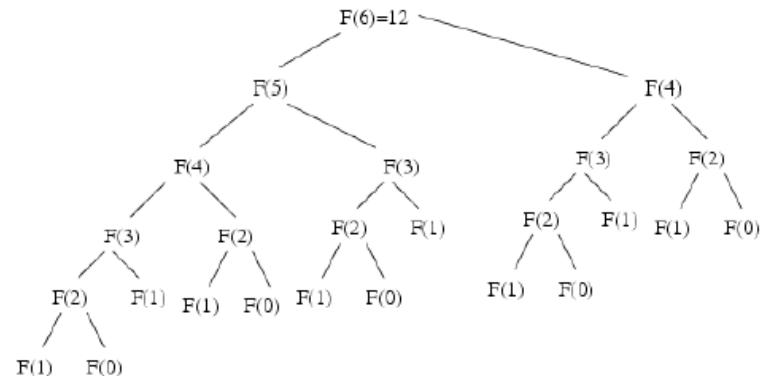
## Cách 1. giải bài toán Fibo bằng đệ quy

Ta có  $F_{n+1}/F_n \approx (1+\sqrt{5})/2 \approx 1.61803$  suy ra  $F^n \approx 1.61803^n$ .

- ◆ Do cây nhị phân tính Fn chỉ có 2 nút F0, F1 nên chúng ta sẽ có xấp xỉ  $1.61803^n$  lần gọi đến hàm Fibonaci trên (thực tế n=6 là 13 lần)
- ◆ Có thể nói độ phức tạp của thuật toán là hàm mũ.

## Cách 1. giải bài toán Fibo bằng đệ quy

- ◆ Tuy nhiên, đây là một thuật toán không hiệu quả, giả sử ta cần tính F6.



## Cách 2. giải bài toán Fibo bằng quy hoạch động

- ◆ Sử dụng một thuật toán có độ phức tạp tuyến tính để tính Fn bằng cách sử dụng một mảng để lưu các giá trị dùng để tính Fn
  - $F_0 = 0$
  - $F_1 = 1$
  - For  $i = 2$  to  $n$ 
    - $F_i = F_{i-1} + F_{i-2}$
- ◆ Giảm được thời gian, mất thêm bộ nhớ để chứa các kết quả trung gian trong quá trình tính toán.

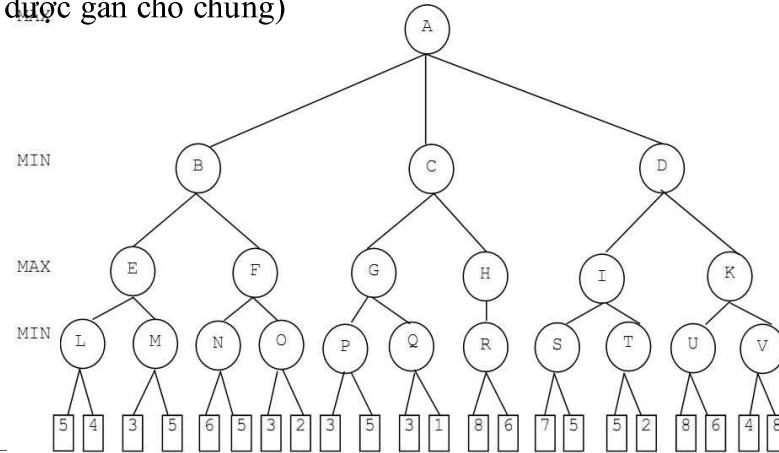
## Cách 2. giải bài toán Fibo bằng quy hoạch động

◆ Qua ví dụ trên rút ra nhận xét.

- Quy hoạch động là một kỹ thuật tính toán đệ quy hiệu quả bằng cách lưu trữ các kết quả cục bộ
- Trong quy hoạch động kết quả của các bài toán con thường được lưu vào một mảng

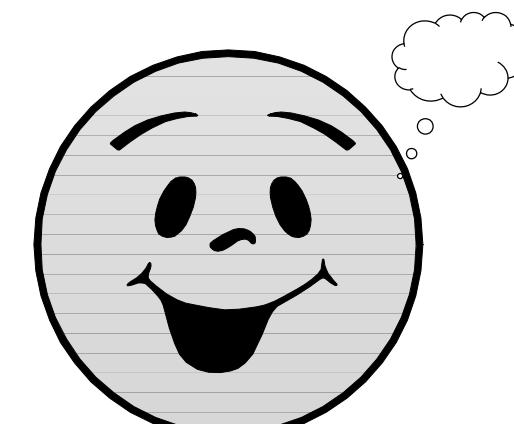
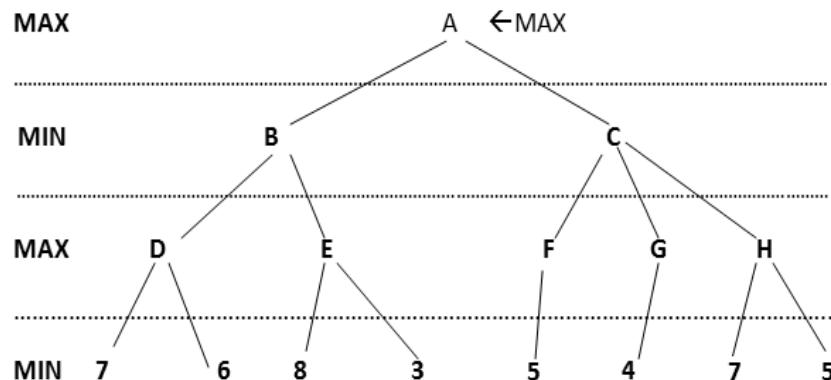
## Bài tập Kỹ thuật Alpha-Beta

◆ **Bài 3:** Dùng kĩ thuật cắt tỉa alpha-beta để định trị cho nút gốc của cây trò chơi sau (các số trong các nút lá là các giá trị đã được gán cho chúng)



## Bài tập Kỹ thuật Alpha-Beta

◆ **Bài 4:** Dùng kỹ thuật cắt tỉa alpha – beta để định trị cho nút gốc của cây trò chơi sau (các nút lá đã được gán trị):



Cảm ơn !

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:  
**NGUYỄN THÁI DƯ**

### Sắp xếp nội và sắp xếp ngoại

#### ◆ Sắp xếp nội:

- Sắp xếp dữ liệu trên RAM
- Tốc độ truy xuất ngẫu nhiên cao
- Dựa trên việc hoán đổi
- Lượng dữ liệu cần sx nhỏ (vừa với RAM)

#### ◆ Sắp xếp ngoại:

- Sắp xếp dữ liệu trên đĩa (Disk)
- Tốc độ truy xuất tuần tự cao (vẫn thấp hơn nhiều so với RAM)
- Dựa trên thao tác trộn
- Lượng dữ liệu cần sx lớn

## Chương 3 SẮP XẾP NGOẠI

- ◆ Sắp thứ tự ngoại là sắp thứ tự trên tập tin
- ◆ Vì sao phải sắp xếp trên tập tin?
- ◆ Các phương pháp SX ngoại
  - Phương pháp trộn Run
  - Phương pháp trộn tự nhiên
  - Phương pháp trộn đa lối cân bằng (Balanced multiway merging)

### 1. Phương pháp trộn Run

- ◆ Run là một dãy liên tiếp các phần tử đã có thứ tự
- ◆ Ví dụ về Run: 2 4 7 12 50 40 60
- ◆ Chiều dài của Run chính là số phần tử trong Run
- ◆ Trong ví dụ trên có 2 run có độ dài lần lượt là 5 và 2
- ◆ Mỗi phần tử của dãy chính là 1 run có độ dài bằng 1

## 1. Phương pháp trộn Run

- ◆ Việc tạo ra một run mới từ 2 run ban đầu gọi là trộn run (merge).
- ◆ Run được tạo từ hai run ban đầu là một dãy các phần tử đã được sắp thứ tự.

## 1. Phương pháp trộn Run

Mô tả bài toán

- Dữ liệu vào: tập tin f0 cần sắp xếp
- Dữ liệu ra: tập tin f0 đã được sắp xếp
- f1, f2 là hai tập tin phụ dùng để sắp xếp

## 1. Phương pháp trộn Run

- Giả sử các phần tử trên f0 là:

24 12 67 33 58 42 11 34 29 31

- Khởi tạo f1, f2 rỗng

### Bước 1:

- Thực hiện **phân bõ** m=1 phần tử lần lượt từ f0 vào f1 và f2:

f1: 24 67 58 11 29

f2: 12 33 42 34 31

- **Trộn** f1, f2 thành f0:

f0: 12 24 33 67 42 58 11 34 29 31

## 1. Phương pháp trộn Run

### Bước 2:

- **Phân bõ** m=2\*m=2 phần tử lần lượt từ f0 vào f1 và f2:

f0: 12 24 33 67 42 58 11 34 29 31

f1: 12 24 42 58 29 31

f2: 33 67 11 34

- **Trộn** f1, f2 thành f0:

f0: 12 24 33 67 11 34 42 58 29 31

f1: 12 24 42 58 29 31

f2: 33 67 11 34

## 1. Phương pháp trộn Run

### Bước 3:

- Tương tự bước 2, **phân bô**  $m=2*m=4$  phần tử lần lượt từ f0 vào f1 và f2, kết quả thu được như sau:

f0: 12 24 33 67 11 34 42 58 29 31

f1: 12 24 33 67 29 31

f2: 11 34 42 58

- **Trộn** f1, f2 thành f0:

f0: 11 12 24 33 34 42 58 67 29 31

## 1. Phương pháp trộn Run

### ◆ Thuật toán tổng quát

[B1]  $m = 1$

[B2] Chia xoay vòng dữ liệu của file f0 cho f1 và f2, mỗi lần m phần tử, cho đến khi file f0 hết

[B3] Trộn từng cặp m phần tử của f1 và f2 tạo thành dãy mới 2m phần tử (được sắp) trên f0

[B4]  $m = 2*m$

[B5] Nếu ( $m < N$ ) thì Quay lại bước [B2]

Ngược lại **Kết thúc thuật toán**

## 1. Phương pháp trộn Run

### Bước 4:

- **Phân bô**  $m=2*m=8$  phần tử lần lượt từ f0 vào f1 và f2:

f0: 11 12 24 33 34 42 58 67 29 31

f1: 11 12 24 33 34 42 58 67

f2: 29 31

- **Trộn** f1, f2 thành f0:

f0: 11 12 24 29 31 33 34 42 58 67

### Bước 5:

Lặp lại tương tự các bước trên, cho đến khi chiều dài m của run cần phân bổ lớn hơn chiều dài n của f0 thì dừng.

## Thuật toán trộn Run

```
m = 1
while (m < số phần tử của fo)
{
    Chia(Distribute) m phần tử của fo lần lượt
    cho f1, f2
    Trộn(Merge) f1, f2 lần lượt vào fo
    M = M * 2
}
```

## 2. Phương pháp trộn tự nhiên

- ◆ Trong phương pháp trộn đã trình bày ở trên, giải thuật chưa tận dụng được chiều dài cực đại của các run trước khi phân bổ; do vậy, việc tối ưu thuật toán chưa được tận dụng.
- ◆ Đặc điểm cơ bản của phương pháp trộn tự nhiên là tận dụng độ dài “tự nhiên” của các run ban đầu; nghĩa là, thực hiện việc trộn các run có độ dài cực đại với nhau cho đến khi dãy chỉ bao gồm một run -> dãy đã được sắp thứ tự.

## 2. Phương pháp trộn tự nhiên

### ◆ Giải thuật

While (Số Run của F0 >1)

{

    Phân bố F0 vào F1, F2 theo các Run tự nhiên.

    Trộn các Run của F1, F2 vào F0.

}

- [Distribute] Chia xoay vòng dữ liệu của F0 cho F1 và F2, mỗi lần 1 run cho đến khi file F0 hết.
- [Merger] Trộn từng cặp run của F1 và F2 tạo thành run mới trên F0.

## 2. Phương pháp trộn tự nhiên

**Lắp** Cho đến khi dãy cần sắp chỉ gồm duy nhất một run.

### Phân bố:

Phân bố F0 vào F1 và F2 theo các run tự nhiên

### Trộn:

Trộn các run của F1 và F2 vào F0

Quá trình này sẽ tiếp tục cho đến khi số run của F0 là 1 thì dừng

## 2. Phương pháp trộn tự nhiên

- ◆ F0: 1 2 9 8 7 6 5

### ◆ Bước 1:

- F1: 1 2 9 7 5
- F2: 8 6
- F0: 1 2 8 9 6 7 5

### ◆ Bước 2:

- F1: 1 2 8 9 5
- F2: 6 7
- F0: 1 2 6 7 8 9 5

### ◆ Bước 3:

- F1: 1 2 6 7 8 9
- F2: 5
- F0: 1 2 5 6 7 8 9

### ◆ Bước 4: Dừng vì F0 chỉ có 1 Run

### 3. Phương pháp trộn đa lối cân bằng

- ◆ Thuật toán sắp xếp ngoài cần 2 giai đoạn: Phân phối và trộn.
  - Giai đoạn nào làm thay đổi thứ tự?
  - Chi phí cho giai đoạn phân phối?
- ◆ Rút ra kết luận:
  - Thay vì thực hiện 2 giai đoạn, ta chỉ cần thực hiện 01 giai đoạn trộn.
    - Tiết kiệm  $\frac{1}{2}$  chi phí Copy.
    - Cần số lượng file trung gian gấp đôi.

### 3. Phương pháp trộn đa lối cân bằng

- ◆ B1: Gọi tập nguồn  $S = \{f_1, f_2, \dots, f_n\}$   
Gọi tập đích  $D = \{g_1, g_2, \dots, g_n\}$   
Chia xoay vòng dữ liệu của file  $F_0$  cho các file thuộc tập nguồn, mỗi lần 1 Run cho tới khi  $F_0$  hết.
  - ◆ B2: Trộn từng bộ Run của các file thuộc tập nguồn  $S$ , tạo thành Run mới, mỗi lần ghi lên các file thuộc tập đích  $D$ .
  - ◆ B3: Nếu ( $số Run trên các file của D > 1$ ) thì:
    - Hoán vị vai trò tập nguồn ( $S$ ) và tập đích ( $D$ ).
    - Quay lại B2
- Ngược lại kết thúc thuật toán.

### 3. Phương pháp trộn đa lối cân bằng

- Ví dụ: Cho dãy số sau

3 5 2 7 12 8 4 15 20 1 2 8 23 7 21 27

- Nhập :

f0 : 3 5 2 7 12 8 4 15 20 1 2 8 23 7 21 27

- Xuất :

f0: 1 2 2 3 4 5 7 7 8 8 12 15 20 21 23 27

### 3. Phương pháp trộn đa lối cân bằng

- Nhập :

f0 : 3 5 2 7 12 8 4 15 20 1 2 8 23 7 21 27

Bước 0: đặt  $m = 3$

Bước 1:

Phân phối các run luân phiên vào  $f[1], f[2], f[3]$

f1: 3 5 4 15 20

f2: 2 7 12 1 2 8 23

f3: 8 7 21 27

### 3. Phương pháp trộn đa lối cân bằng

Bước 2:

-Trộn các run của f[1], f[2], f[3] và luân phiên phân phôi vào các file g[1], g[2], g[3]

g1: 2 3 5 7 8 12

g2: 1 2 4 7 8 15 20 21 23 27

g3:

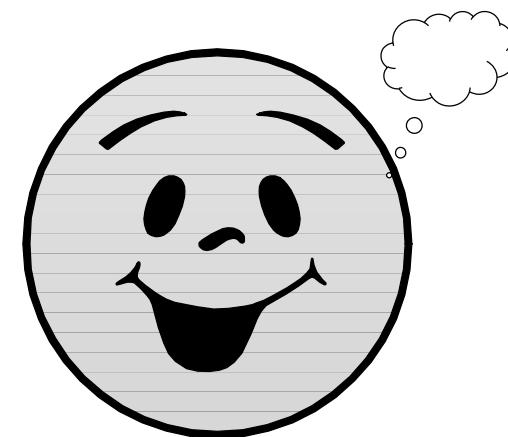
- Do số run sau khi trộn >1 nên tiếp tục trộn run từ g[1], g[2], g[3] vào ngược trở lại f[1], f[2], f[3]

f1: 1 2 2 3 4 5 7 7 8 8 12 15 20 21 23 27

f2:

f3:

- Do số run trộn = 1 nên kết thúc thuật toán



Cảm ơn !

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:  
**NGUYỄN THÁI DƯ**

### ĐẶT VĂN ĐỀ

- ◆ Cho S là tập hợp n phần tử trong 1 cấu trúc dữ liệu được đặc trưng bởi 1 giá trị khóa
- ◆ Tìm 1 phần tử có hay không trong S
  - Tìm tuyến tính ( $O(n)$ ), chưa được sắp xếp
  - Tìm nhị phân ( $O(\log_2 n)$ ), đã được sắp xếp
- ◆ Có hay chăng 1 thuật toán tìm kiếm với  $O(1)$ 
  - Có, song ta phải tổ chức lại dữ liệu
  - Dữ liệu được tổ chức lại là Bảng băm

### Chương 4 BẢNG BĂM

- ◆ Giới thiệu bài toán
- ◆ Hàm băm
- ◆ Các phương pháp xử lý đụng độ

### Giới thiệu về Bảng Băm

- ◆ Là CTDL trong đó các phần tử của nó được lưu trữ sao cho việc tìm kiếm sẽ được thực hiện bằng cách truy xuất trực tiếp thông qua từ khóa.
- ◆ Bảng băm có M vị trí được đánh chỉ mục từ 0 đến M-1, M là kích thước của bảng băm.
- ◆ **Các phương pháp băm:**
  - PP kết nối trực tiếp
  - PP kết nối hợp nhất
  - PP dò tuyến tính
  - PP dò bậc 2
  - PP băm kép

## Hàm băm (Hash functions)

### ◆ Hàm băm: biến đổi khóa thành chỉ mục trên bảng băm

- Khóa có thể là dạng số hay dạng chuỗi
- Chỉ mục được tính từ 0..M-1, với M là số chỉ mục của bảng băm
- Hàm băm thường dùng: key % M, với M là độ lớn của bảng băm

### ◆ Hàm băm tốt phải thoả yêu cầu

- Giảm thiểu xung đột
- Phân bố đều trên M địa chỉ khác nhau của bảng băm

## Ưu điểm bảng băm

### ◆ Dung hòa tốt giữa thời gian truy xuất và dung lượng bộ nhớ

- Nếu không giới hạn bộ nhớ: one-to-one, truy xuất tức thì
- Nếu dung lượng bộ nhớ có giới hạn thì tổ chức một số khóa có cùng địa chỉ, lúc này thời gian truy xuất có bị suy giảm đôi chút.

### ◆ Bảng băm ứng dụng nhiều trong thực tế, thích hợp tổ chức dữ liệu có kích thước lớn và lưu trữ ngoài

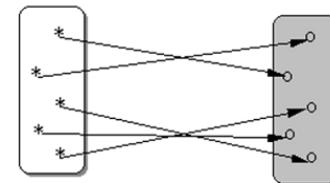
## Mô tả dữ liệu

◆ K: tập các khoá (set of keys)

◆ M: tập các địa chỉ (set of addresses).

◆ HF(k): hàm băm dùng để ánh xạ một khoá k từ tập các khoá K thành một địa chỉ tương ứng trong tập M.

Thông thường  $HF(k) = k \bmod M$



Tập khóa K

Hàm băm

Tập địa chỉ M

## Cách xây dựng bảng băm

◆ Dùng hàm băm để ánh xạ khóa K vào 1 vị trí trong bảng băm. Vị trí này như là 1 địa chỉ khi tìm kiếm.

◆ Bảng băm thường là mảng, danh sách liên kết, file(danh sách đặc)

## Ví dụ một bảng băm đơn giản

- ◆ Khóa k sẽ được lưu trữ tại vị trí  $k \bmod M$  ( $M$  kích thước mảng)
- ◆ Ví dụ: Thêm phần tử  $x = 95$  vào mảng  $M=10$

$$95 \bmod 10 = 5$$

0	1	2	3	4	5	6	7	8	9
					<b>95</b>				

## Ví dụ một bảng băm đơn giản

- ◆ Với các giá trị: 31, 10, 14, 93, 82, 95, 79, 18, 27, 46

0	1	2	3	4	5	6	7	8	9
10	31	82	93	14	95	46	27	18	79

## Tìm kiếm trên bảng băm

- ◆ Thao tác cơ bản nhất được cung cấp bởi Hashtable là “tìm kiếm”
- ◆ Chi phí tìm kiếm trung bình là  $O(1)$ , không phụ thuộc vào số lượng phần tử của mảng (Bảng).
- ◆ Chi phí tìm kiếm xấu nhất (ít gấp) có thể là  $O(n)$

## Các phép toán trên hàm băm

- ◆ Khởi tạo (Initialize)
- ◆ Kiểm tra rỗng (Empty)
- ◆ Lấy kích thước bảng băm (size)
- ◆ Tìm kiếm một phần tử trong bảng băm (Search)
- ◆ Thêm 1 phần tử vào bảng băm (Insert)
- ◆ Xóa 1 phần tử khỏi bảng băm (Remove)
- ◆ Duyệt (Traverse)

## Vấn đề nảy sinh

- ◆ Giả sử thêm 55 vào bảng băm sau:

0	1	2	3	4	5	6	7	8	9
		82			95		27		

- 55 phải lưu vào vị trí 5. Tuy nhiên vị trí này đã có chứa 95
  - $k_1 \neq k_2$  mà  $f(k_1) = f(k_2) \Rightarrow$  **Đụng độ**
- => **Cần giải quyết đụng độ (xung đột)**

## Làm giảm xung đột

- ◆ Hàm băm cần thỏa mãn các điều kiện:
  - Xác xuất phân bố khoá là đều nhau
  - Dễ dàng tính toán thao tác
  - Ít xảy ra đụng độ

## Vấn đề xung đột khi xử lý bảng băm

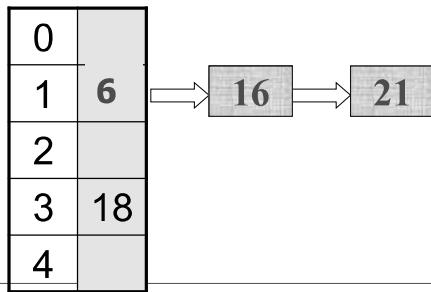
- ◆ Trong thực tế có nhiều trường hợp có nhiều hơn 2 phần tử sẽ được “băm” vào cùng 1 vị trí
- ◆ Hiển nhiên phần tử được “băm” đầu tiên sẽ chiếm lĩnh vị trí đó, các phần tử sau cần phải được lưu vào các vị trí trống khác sao cho vấn đề truy xuất và tìm kiếm phải dễ dàng

## Giải quyết xung đột

- ◆ Các phương pháp băm:
  - PP nối kết trực tiếp
  - PP nối kết hợp nhất
  - PP dò tuyến tính
  - PP dò bậc hai
  - PP băm kép

## Sử dụng DS liên kết (nối kết trực tiếp)

- ◆ Ý tưởng: “Các phần tử băm vào trùng vị trí k được nối vào danh sách nối kết” tại vị trí đó.
- ◆ Ví dụ: cho hàm băm  $F(k) = k \bmod 5$ ,
  - Thêm 6, 16 vào bảng băm sau:

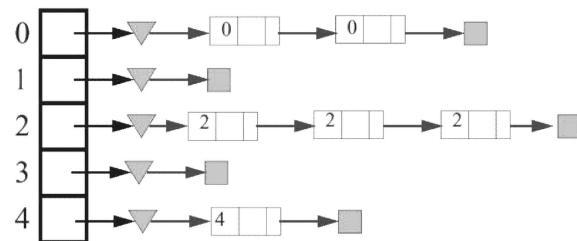


Nguyễn Thái Dư - AGU

17

## Sử dụng DS liên kết (nối kết trực tiếp)

- ◆ Các nút bị băm cùng địa chỉ (các nút bị xung đột) được gom thành một danh sách liên kết
- ◆ Các nút trên bảng băm được *băm* thành các danh sách liên kết. Các nút bị xung đột tại địa chỉ  $i$  được nối kết trực tiếp với nhau qua danh sách liên kết  $i$ .



Nguyễn Thái Dư - AGU

18

## \*Nhận xét

PP DSLK có nhiều khuyết điểm:

- Khi có quá nhiều khoá vào cùng vị trí, DSLK thì tại vị trí đó sẽ rất dài  
=> Tăng chi phí tìm kiếm
- Các ô trống còn dư nhiều => lãng phí về thời gian tìm kiếm và không lưu trữ

Nguyễn Thái Dư - AGU

19

## Sử dụng PP “nối kết hợp nhất”

- ◆ Ý tưởng: “Nếu có 1 khóa bị băm vào vị trí đã có phần tử thì nó sẽ được chèn vào ô trống phía cuối mảng”. (Dùng mảng có M phần tử)

Nguyễn Thái Dư - AGU

20

## Sử dụng PP “nối kết hợp nhất”

- ◆ Bảng băm trong trường hợp này được cài đặt bằng danh sách liên kết dùng mảng, có M nút. Các nút bị xung đột địa chỉ được nối kết nhau qua một danh sách liên kết.
- ◆ Mỗi nút của bảng băm là một mẫu tin có 2 trường:
  - Trường key: chứa các khóa node
  - Trường next: con trỏ chỉ node kế tiếp nếu có xung đột.
- ◆ Khi khởi động bảng băm thì tất cả trường key được gán NULL, tất cả trường next được gán -1.

Nguyễn Thái Dư - AGU

Key	next
NULL	-1
...	...
NULL	-1

21

## Ví dụ: nối kết hợp nhất

- ◆ Minh họa cho bảng băm có tập khóa là tập số tự nhiên, tập địa chỉ có 10 địa chỉ ( $M=10$ ) (từ địa chỉ 0 đến 9), chọn hàm băm  
 $f(key) = key \% 10$ .
- ◆ Thêm Key=10, 42, 20, 109

	Key	Next
0	10	9
1		-1
2	42	-1
...	null	-1
8	109	-1
9	20	8

Nguyễn Thái Dư - AGU

23

## Sử dụng PP “nối kết hợp nhất”

- ◆ Khi thêm một nút có khóa key vào bảng băm, hàm băm  $f(key)$  sẽ xác định địa chỉ i trong khoảng từ 0 đến  $M-1$ .
  - Nếu chưa bị xung đột thì thêm nút mới vào địa chỉ này.
  - Nếu bị xung đột thì nút mới được cấp phát là nút trống phía cuối mảng. Cập nhật liên kết next sao cho các nút bị xung đột hình thành một danh sách liên kết.
- ◆ Khi tìm một nút có khóa key trong bảng băm, hàm băm  $f(key)$  sẽ xác định địa chỉ i trong khoảng từ 0 đến  $M-1$ , tìm nút khóa key trong danh sách liên kết xuất phát từ địa chỉ i.

Nguyễn Thái Dư - AGU

22

## Sử dụng PP “Dò tuyển tính”

- ◆ Ý tưởng: “Nếu có 1 khóa bị băm vào vị trí đã có phần tử thì nó sẽ được chèn vào ô trống gần nhất” theo phía bên phải.

Nguyễn Thái Dư - AGU

24

## Sử dụng PP “Dò tuyển tính”

- ◆ Bảng băm trong trường hợp này được cài đặt bằng danh sách kề có M nút, mỗi nút của bảng băm là một mẩu tin có một trường key để chứa khoá của nút.

0	Null
1	Null
2	Null
3	Null
...	Null
M-1	Null

## Sử dụng PP “Dò tuyển tính”

- ◆ Hàm băm lại của phương pháp dò tuyển tính là truy xuất địa chỉ kế tiếp. Hàm băm lại lần i được biểu diễn bằng công thức sau:
- ◆  $f_i(key) = (f(key) + i) \% M$  với  $f(key)$  là hàm băm chính của bảng băm.
- ◆ Lưu ý địa chỉ dò tìm kế tiếp là địa chỉ 0 nếu đã dò đến cuối bảng

### Nhận xét:

- ◆ Chúng ta thấy bảng băm này chỉ tối ưu khi băm đều, tốc độ truy xuất lúc này có bậc 0(1).
- ◆ Trường hợp xấu nhất là băm không đều hoặc bảng băm đầy, lúc này hình thành một khối đặc có n nút trên tốc độ truy xuất lúc này có bậc 0(n).

## Sử dụng PP “Dò tuyển tính”

- ◆ Thêm các nút 32, 53, 22, 92, 17, 34, 24, 37, 56 vào bảng băm.

0	NULL	0	NULL	0	NULL	0	NULL	0	56
1	NULL								
2	32	2	32	2	32	2	32	2	32
3	53	3	53	3	53	3	53	3	53
4	NULL	4	22	4	22	4	22	4	22
5	NULL	5	92	5	92	5	92	5	92
6	NULL	6	NULL	6	34	6	34	6	34
7	NULL	7	NULL	7	17	7	17	7	17
8	NULL	8	NULL	8	NULL	8	24	8	24
9	NULL	9	NULL	9	NULL	9	37	9	37

## Sử dụng PP “Dò bậc hai”

- ◆ Bảng băm dùng phương pháp dò tuyển tính bị hạn chế do rải các nút không đều, bảng băm với phương pháp dò bậc hai rải các nút đều hơn.
- ◆ Hàm băm lại của phương pháp dò bậc hai là truy xuất các địa chỉ cách bậc 2.
- ◆ Hàm băm lại hàm  $f_i$  được biểu diễn bằng công thức sau:
  - $f_i(key) = (f(key) + i^2) \% M$
  - với  $f(key)$  là hàm băm chính của bảng băm.
- ◆ Nếu đã dò đến cuối bảng thì trở về dò lại từ đầu bảng.
- ◆ Bảng băm với phương pháp dò bậc hai nên chọn số địa chỉ M là số nguyên tố.

## Ví dụ:

- ◆ Cho hàm băm chính:  $F(k) = k \bmod 5$ , sử dụng PP “dò bậc hai”  $f_i(key) = (f(key) + i^2) \% M$
- ◆ Thêm 6, 16 vào bảng băm sau:

0	NONE	0	16
1	21	1	21
2	NONE	2	6
3	18	3	18
4	NONE	4	NONE

Thêm vào các khóa 10, 15, 16, 20, 30, 25, ,26, 36

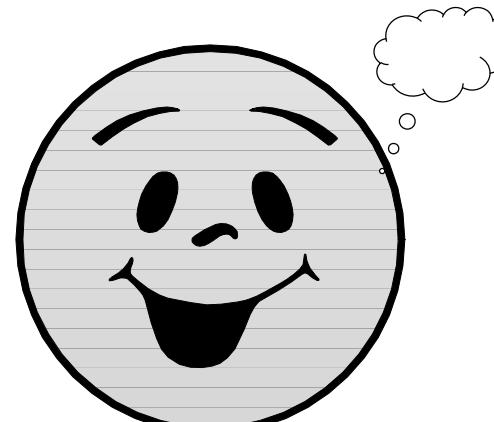
0	10	0	10	0	10	0	10	0	10
1	NONE	1	20	1	20	1	20	1	20
2	NONE	2	NONE	2	NONE	2	NONE	2	36
3	NONE								
4	NONE	4	NONE	4	30	4	30	4	30
5	15	5	15	5	15	5	15	5	15
6	16	6	16	6	16	6	16	6	16
7	NONE	7	NONE	7	NONE	7	26	7	26
8	NONE								
9	NONE	9	NONE	9	25	9	25	9	25

## Sử dụng PP “Băm kép”

- ◆ Ta sử dụng 2 hàm băm:

$$f_1(key) = key \% M$$

$$f_2(key) = (M-2) - key \% (M-2)$$



Cảm ơn !

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:

**NGUYỄN THÁI DƯ**

### Giới thiệu

- ◆ Đối với cây nhị phân, mỗi nút chỉ có một mục dữ liệu và có thể có hai nút con.
- ◆ Nếu chúng ta cho phép một nút có nhiều mục dữ liệu và nhiều nút con thì kết quả là ta được cây nhiều nhánh



### Chương 5 CÂY 2-3-4

- ◆ Giới thiệu về cây 2-3-4
- ◆ Tổ chức
- ◆ Tìm kiếm
- ◆ Thêm vào

### Giới thiệu về cây 2-3-4

- ◆ Cây 2-3-4 là cây nhiều nhánh mà mỗi nút của nó có thể có đến bốn nút con và ba mục dữ liệu.
- ◆ Các số 2, 3 và 4 trong cụm từ cây 2-3-4 có ý nghĩa là khả năng có bao nhiêu liên kết đến các node con có thể có được trong một node cho trước.

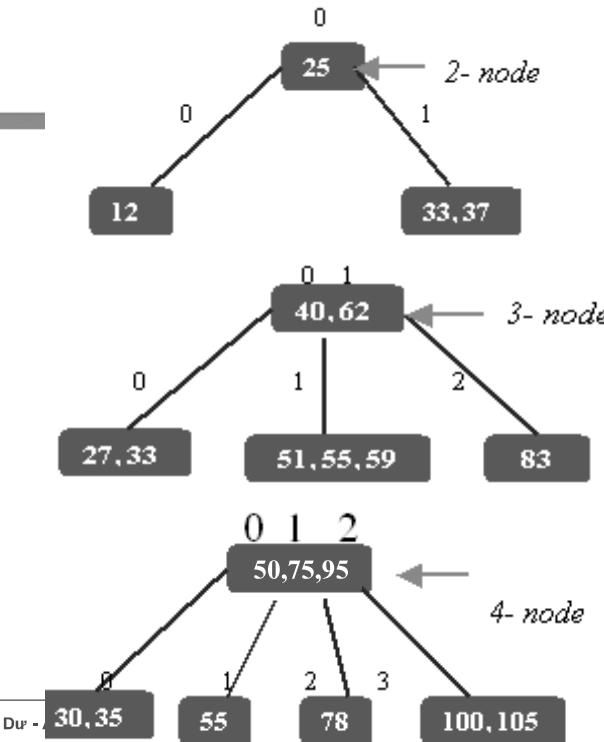
## Giới thiệu về cây 2-3-4 (tt)

◆ Đối với các node không phải là lá, có 3 cách sắp xếp sau:

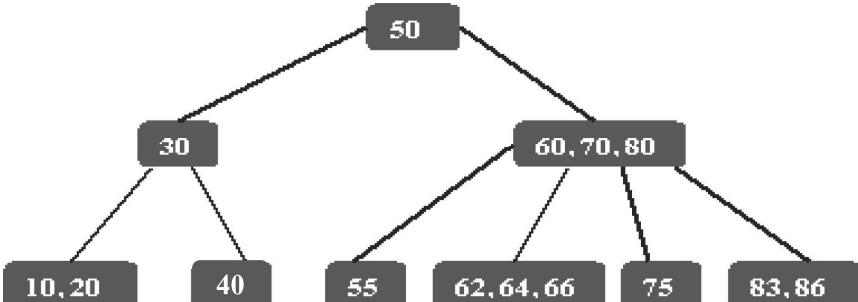
- Một node với một mục dữ liệu thì luôn luôn có 2 con.
- Một node với hai mục dữ liệu thì luôn luôn có 3 con.
- Một node với ba mục dữ liệu thì luôn luôn có 4 con.

◆ Nói cách khác, nếu số con là L và số mục dữ liệu là D, thì:

$$L = D + 1$$



## Giới thiệu về cây 2-3-4 (tt)



◆ Một node lá thì không có node con nhưng có thể chứa 1, 2 hoặc 3 mục dữ liệu.

◆ Trong **cây 2-3-4** không tồn tại node chỉ có liên kết đơn. Một node với 1 mục dữ liệu luôn luôn phải có 2 liên kết, trừ khi nó là node lá (node không có liên kết nào).

## Tổ chức cây 2-3-4

◆ Các mục dữ liệu trong mỗi node được sắp xếp theo thứ tự tăng dần từ trái sang phải (sắp xếp từ thấp đến cao).

◆ Trong cây tìm kiếm nhị phân: **NL < NRoot < NR** *khoá của nút cây con bên trái nhỏ hơn khoá của nút đang xét và khoá của nút đang xét nhỏ hơn khoá của nút cây con bên phải.*

◆ **Cây 2-3-4** cũng có tính chất như trên, nhưng có thêm đặc điểm sau:

- Tất cả các nút con của cây con có gốc thứ i thì có các giá trị khoá nhỏ hơn khoá i.

## Tổ chức cây 2-3-4



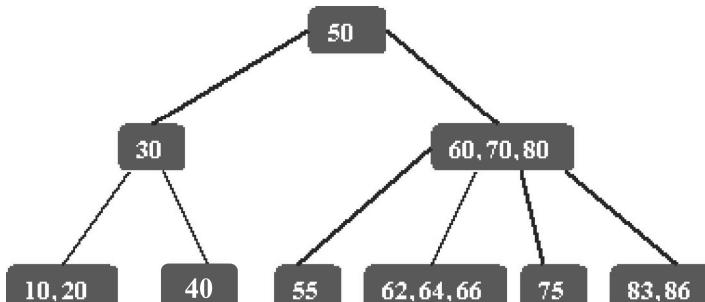
◆ Trong cây 2-3-4 thì nguyên tắc cũng giống như cây tìm kiếm nhị phân, nhưng có thêm một số điểm sau:

- Tất cả các node con của cây con có gốc tại node con thứ 0 thì có các giá trị khoá nhỏ hơn khoá 0.
- Tất cả các node con của cây con có gốc tại node con thứ 1 thì có các giá trị khoá lớn hơn khoá 0 và nhỏ hơn khoá 1.
- Tất cả các node con của cây con có gốc tại node con thứ 2 thì có các giá trị khoá lớn hơn khoá 1 và nhỏ hơn khoá 2.
- Tất cả các node con của cây con có gốc tại node con thứ 3 thì có các giá trị khoá lớn hơn khoá 2.

## Tìm kiếm

◆ Thao tác tìm kiếm trong cây 2-3-4 tương tự như thủ tục tìm kiếm trong cây nhị phân. Việc tìm kiếm bắt đầu từ node gốc và chọn liên kết dẫn đến cây con với phạm vi giá trị phù hợp.

◆ Ví dụ: tìm kiếm mục dữ liệu với khoá là 64

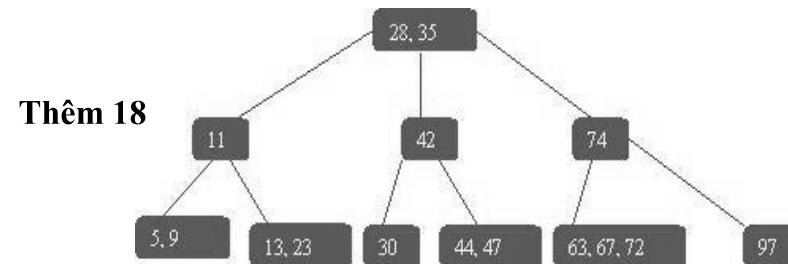


## Thêm vào

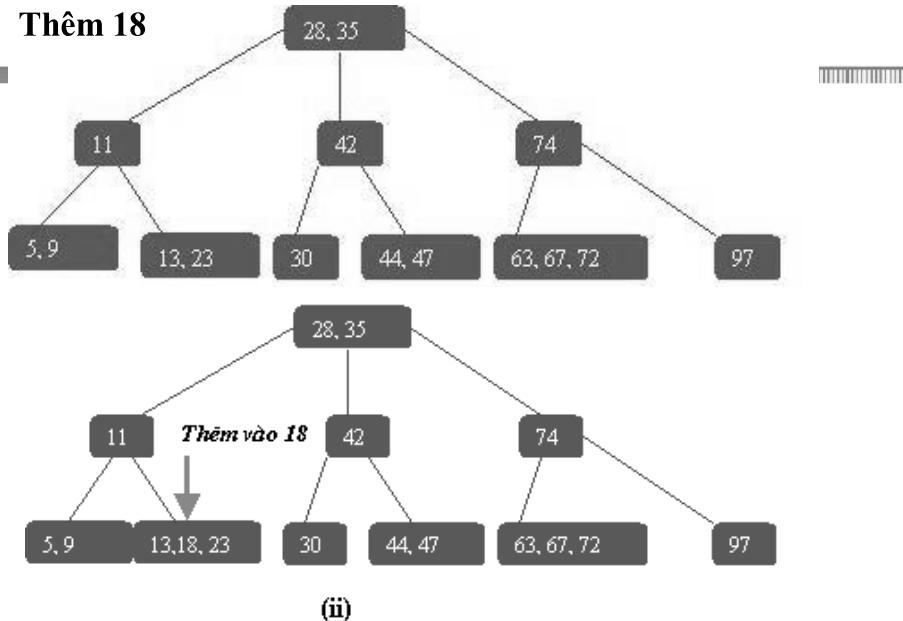
- ◆ Các mục dữ liệu mới luôn luôn được chèn vào tại các node lá.
- ◆ Việc thêm vào cây 2-3-4 trong bất cứ trường hợp nào thì quá trình cũng bắt đầu bằng cách tìm kiếm node lá phù hợp.

## Thêm vào

- ◆ Nếu không có nút đầy nào (nút có đủ 3 mục dữ liệu) được tìm thấy trong quá trình tìm kiếm, việc chèn vào khá là dễ dàng.
- ◆ Khi nút lá phù hợp được tìm thấy, mục dữ liệu mới đơn giản là thêm vào nó



### Thêm 18



### Thêm vào (tt)

- ◆ Việc thêm vào sẽ trở nên phức tạp hơn nếu gặp phải một node đầy (node có số mục dữ liệu đầy đủ) trên nhánh dẫn đến điểm thêm vào. Khi điều này xảy ra, node này cần thiết phải được tách ra.
- ◆ Quá trình tách nhằm giữ cho cây cân bằng.
- ◆ Loại cây 2-3-4 mà chúng ta đề cập ở đây thường được gọi là cây 2-3-4 top-down (các node được tách ra theo hướng đi xuống điểm chèn)

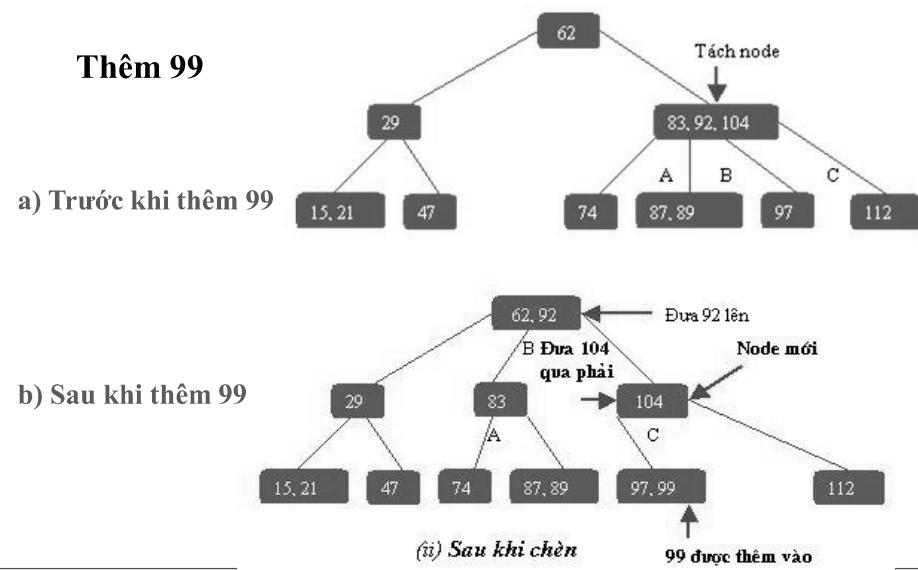
### Thêm vào (tt)

- ◆ Giả sử ta đặt tên các mục dữ liệu trên node bị phân chia là **A**, **B** và **C**. Sau đây là tiến trình tách (chúng ta giả sử rằng node bị tách không phải là node gốc; chúng ta sẽ kiểm tra việc tách node gốc sau này)

- Một node mới và rỗng được tạo. Nó là anh em với node sẽ được tách và được đưa vào bên phải của nó.
- Mục dữ liệu C được chuyển vào node mới.
- Mục dữ liệu B được chuyển vào node cha của node được tách.
- Mục dữ liệu A không thay đổi.
- Hai node con bên phải nhất bị hủy kết nối từ node được tách và kết nối đến node mới.

### Thêm vào (tt)

#### Thêm 99



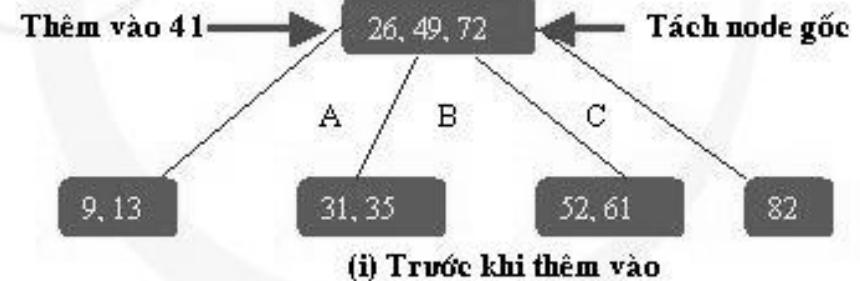
## Thêm vào (tt)

### ◆ Tách node gốc

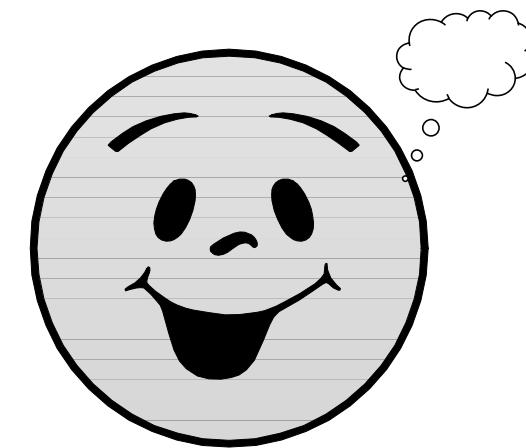
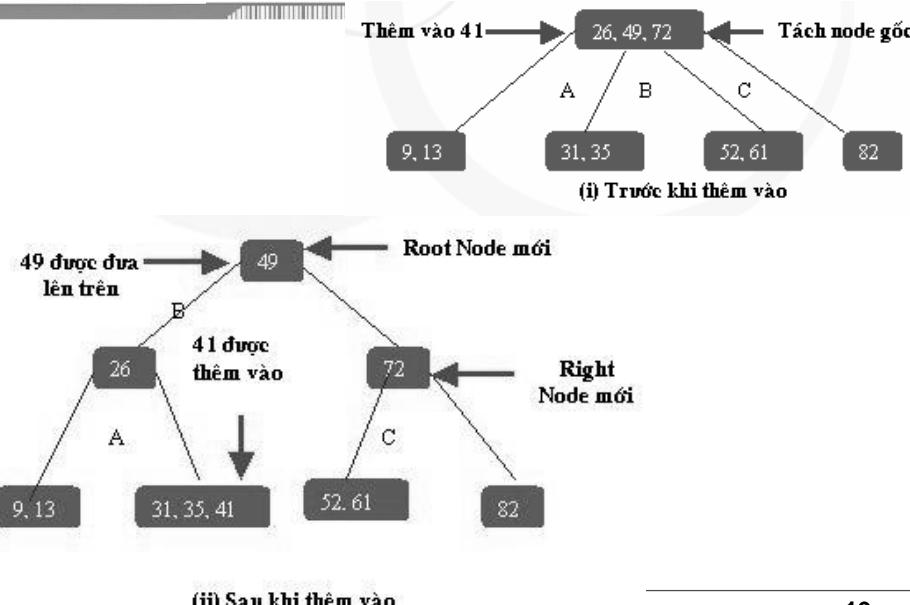
Khi gặp phải node gốc đầy tại thời điểm bắt đầu tìm kiếm điểm chèn, kết quả của việc tách thực hiện như sau:

- Node mới được tạo ra để trở thành gốc mới và là cha của node được tách.
- Node mới thứ hai được tạo ra để trở thành anh em với node được tách.
- Mục dữ liệu C được dịch chuyển sang node anh em mới.
- Mục dữ liệu B được dịch chuyển sang node gốc mới.
- Mục dữ liệu A vẫn không đổi.
- Hai node con bên phải nhất của node được phân chia bị hủy kết nối khỏi nó và kết nối đến node mới bên phải.

## Thêm vào (tt)



## Thêm vào (tt)



Cảm ơn !

## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:

NGUYỄN THÁI DƯ

## Chương 5 B-TREE

- ◆ Giới thiệu
- ◆ Định nghĩa B-Tree
- ◆ Các phép toán trên B-Tree

### Giới thiệu

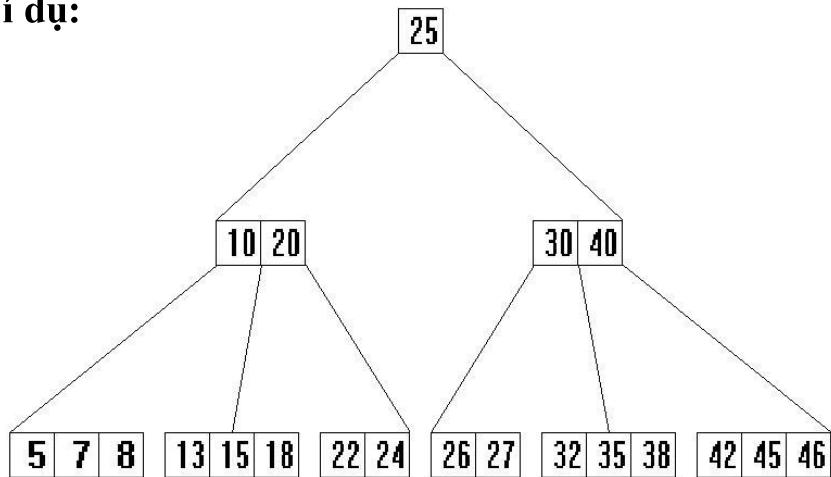
- ◆ Cây 2-3-4 là một ví dụ về cây nhiều nhánh, trong cây nhiều nhánh mỗi node sẽ có nhiều hơn hai node con và nhiều hơn một mục dữ liệu.
- ◆ Một loại khác của cây nhiều nhánh là B-tree, là cây rất hiệu quả khi dữ liệu nằm trong bộ nhớ ngoài.

### Định nghĩa B-Tree

- ◆ Một B-tree **bậc n** có các đặc tính sau:
  - i) Mỗi node có tối đa  $2^n$  khoá.
  - ii) Mỗi node (không là node gốc) có ít nhất là  $n$  khoá.
  - iii) Mỗi node hoặc là node lá hoặc có  $m+1$  node con ( $m$  là số khoá của trang này)
  - iv) Các khóa được sắp tăng dần từ trái sang phải
  - v) Các nút lá nằm cùng một mức

## Định nghĩa B-Tree

Ví dụ:



B-tree bậc 2 có 3 mức

Nguyễn Thái Dư - AGU

5

## Các phép toán trên B-Tree

- ◆ Tìm 1 phần tử có khóa bằng X trong cây
- ◆ Thêm 1 khoá vào vào B – Tree
- ◆ Xóa 1 khoá trong 1 nút

Nguyễn Thái Dư - AGU

7

## Ưu điểm B-Tree

- ◆ B-Tree là dạng cây cân bằng, phù hợp với việc lưu trữ trên đĩa
- ◆ B-Tree tiêu tốn số phép truy xuất đĩa tối thiểu cho các thao tác
- ◆ Có thể quản lý số phần tử rất lớn

Nguyễn Thái Dư - AGU

6

## Tìm kiếm phần tử có khóa X trên cây

- ◆ Khoá cần tìm là X. Với m đủ lớn ta sử dụng phương pháp tìm kiếm nhị phân, nếu m nhỏ ta sử dụng phương pháp tìm kiếm tuần tự. Nếu X không tìm thấy sẽ có 3 trường hợp sau xảy ra:
  - i)  $K_i < X < K_{i+1}$ . Tiếp tục tìm kiếm trên cây con  $C_i$
  - ii)  $K_m < X$ . Tiếp tục tìm kiếm trên  $C_m$
  - iii)  $X < K_1$ . Tiếp tục tìm kiếm trên  $C_0$

Quá trình này tiếp tục cho đến khi node đúng được tìm thấy. Nếu đã đi đến node lá mà vẫn không tìm thấy khoá, việc tìm kiếm là thất bại.

$C_0, K_1, C_1, K_2, \dots, C_{m-1}, K_m, C_m$

Nguyễn Thái Dư - AGU

8

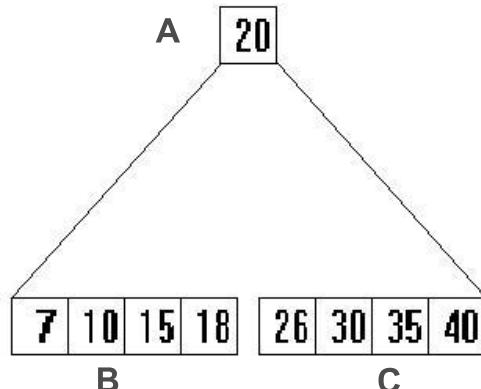
## Thêm 1 nút vào B-Tree

- ◆ Tính chất B-Tree: một node có ít nhất một nửa số khóa
- ◆ Thêm 1 nút có khóa X vào B-Tree
  - Thêm X vào 1 nút lá
  - Sau khi thêm, nếu nút lá đầy:
    - Tách nút lá ra làm đôi
    - Chuyển phần tử giữa lên nút cha và lan truyền ngược về gốc.
    - Nếu gốc bị tách, cây được đặt ở mức sâu hơn

## Thêm vào

### ◆ Ví dụ 1:

- Thêm  $x=22$  vào B-Tree bậc 2. Khóa 22 chưa có trong cây. Nhưng không thể thêm vào node C vì node C đã đầy.

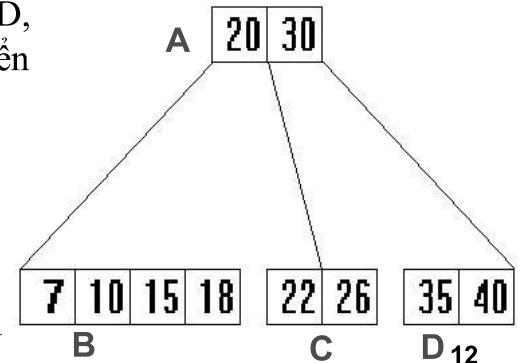
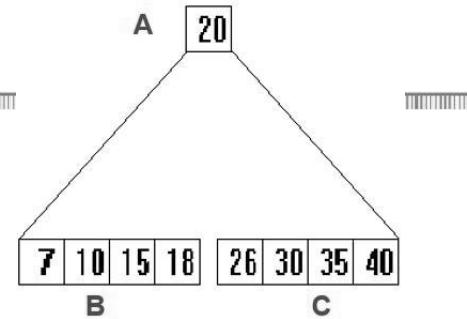


## Thêm 1 nút vào B-Tree

- ◆ Nếu số khóa lớn hơn  $2n$  thì tách node:
  - Đưa phần tử giữa lên node cha
  - Tạo thêm node mới
  - Chuyển dời một nửa phần tử sang node mới
  - Tiếp tục lan truyền ở node cha (nếu node cha sau khi thêm  $> 2n$  phần tử thì thực hiện tách node như trên).

## Thêm vào

Do đó tách node C thành hai node : node mới D được cấp phát và  $m+1$  khóa được chia đều cho 2 node C và D, và khóa ở giữa được chuyển lên node cha A

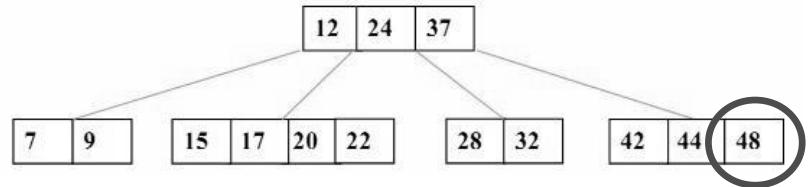


## Xóa 1 phần tử trên B-Cây bậc n

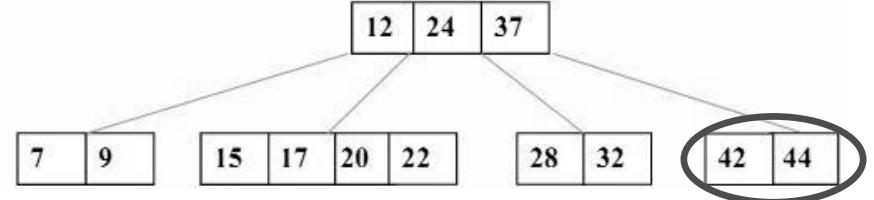
- ◆ Khóa cần xóa trên node lá -> Xóa bình thường.
- ◆ Khóa cần hủy không trên node lá:
  - Tìm phần tử thay thế: Trái nhất (hoặc phải nhất) trên hai cây con cần tìm
  - Thay thế cho nút cần xóa
- ◆ Sau khi xóa, node bị thiếu (vi phạm đk B-Tree):
  - Hoặc chuyển dời phần tử từ node thừa
  - Hoặc ghép với node bên cạnh (trái/phải)

## Ví dụ về xóa

- ◆ Giả sử đã xây dựng B-Tree như sau:

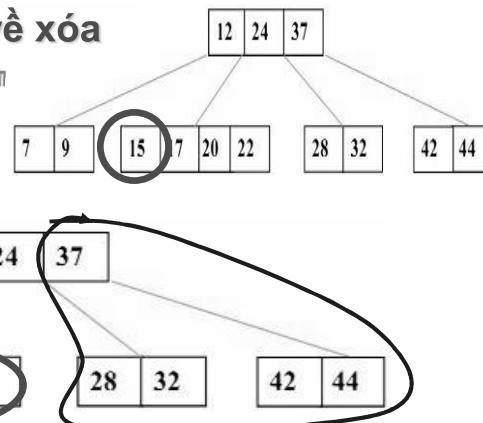


### Xóa 48

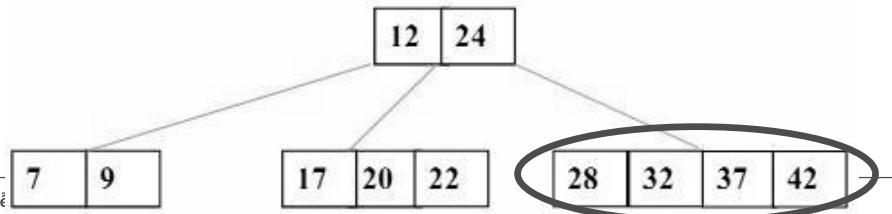


## Ví dụ về xóa

### Xóa 15:

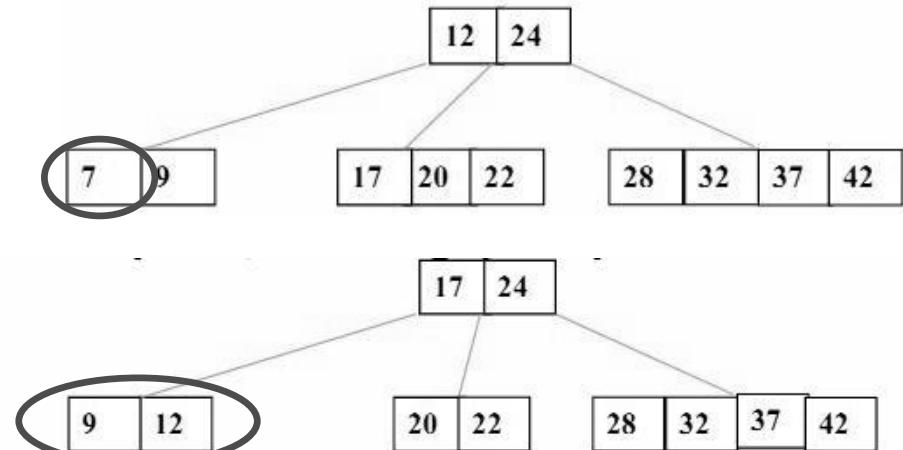


### Xóa 44 (ghép node)

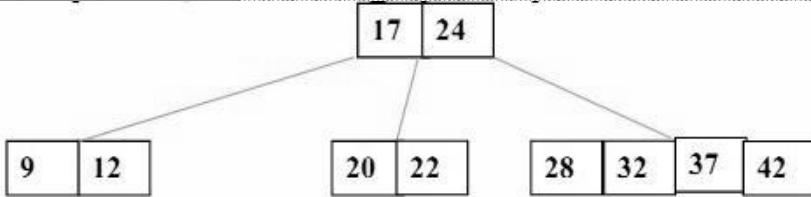


## Ví dụ về xóa

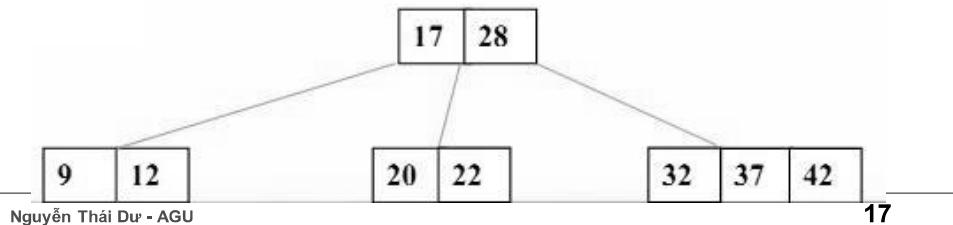
### Xóa 7 (mượn node phải):



## Ví dụ về xóa



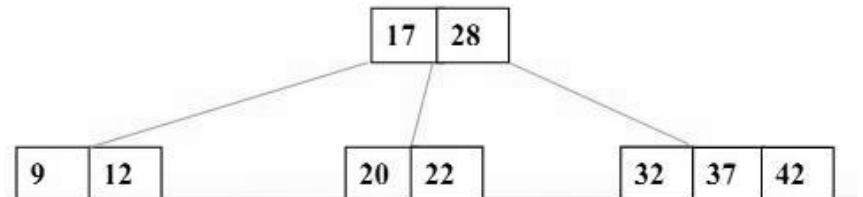
◆ Xóa 24, ta đem khóa 22 lên thay thế. Khi đó node 20,22 chỉ còn 20 (phạm), ta phải đem khóa 22 trở lại node 20,22. Mang 28 lên thêm



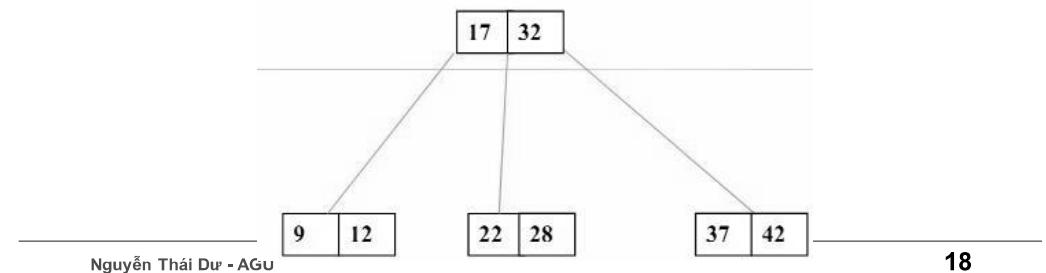
Nguyễn Thái Dư - AGU

19

## Ví dụ về xóa



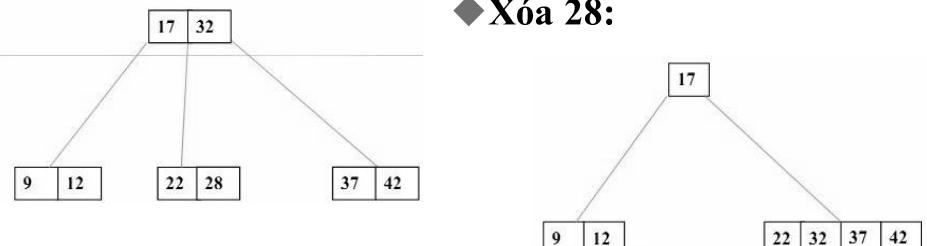
◆ Xóa 20: Mượn node phải 1 phần tử. Tức mang 32 lên cha, 28 xuống node có 1 khóa là 22



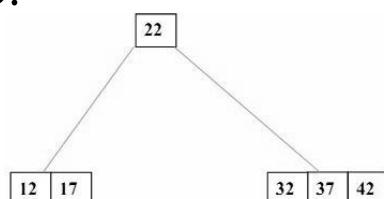
18

## Ví dụ về xóa

◆ Xóa 28:



◆ Xóa 9:

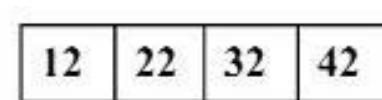


## Ví dụ về xóa

◆ Xóa 37:



◆ Xóa 17:



Nguyễn Thái Dư - AGU

Nguyễn Thái Dư - AGU

20

## B-tree: Cân bằng lại cây sau khi xóa

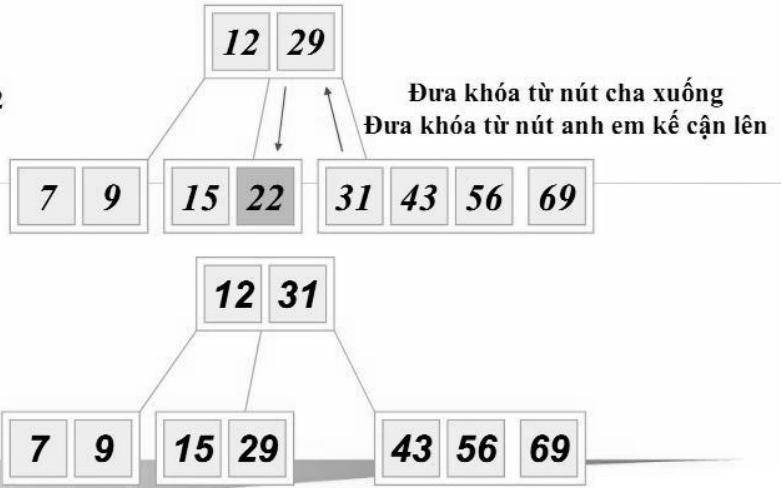
- ◆ Nếu một trong các nút anh em kế cận nút đang xét có số lượng khóa nhiều hơn số lượng tối thiểu
  - Đưa một khóa của nút anh em lên nút cha.
  - Đưa một khóa ở nút cha xuống nút đang xét.
- ◆ Nếu tất cả các nút anh em kế cận nút đang xét đều có số lượng khóa vừa đủ số lượng tối thiểu
  - Chọn một nút anh em kế cận và hợp nhất nút anh em này với nút đang xét và với khóa tương ứng ở nút cha.
  - Nếu nút cha trở nên thiếu khóa, lặp lại quá trình này.

Nguyễn Thái Dư - AGU

21

## Trường hợp:

- ◆ Nút anh em kế cận còn đủ khóa để bổ sung

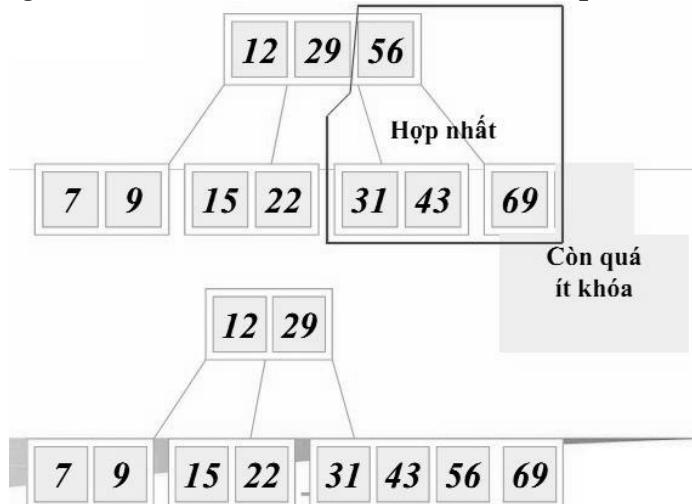


22

Nguyễn Thái Dư - AGU

## Trường hợp:

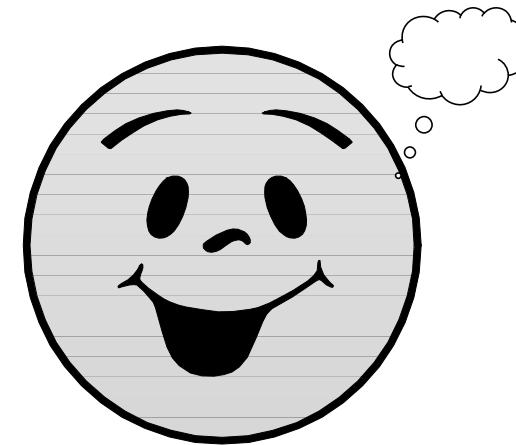
- ◆ Nút đang xét và nút anh em kế cận đều còn quá ít khóa



Nguyễn Thái Dư - AGU

23

Cảm ơn !



## PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:  
**NGUYỄN THÁI DƯ**

### Bổ túc kiến thức toán

Dãy số tự nhiên 1, 2, 3 ,4, 5.....,n

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

◆ Tính tổng:  $[(n+1) * n]/2$  ( số đầu cộng số cuối nhân số  
số hạng chia hai )

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

### Bổ túc kiến thức toán

◆ Viết chương trình tính tổng các số tự nhiên từ 1 đến N.  
Với N nhập vào từ bàn phím.

Dãy số tự nhiên 1, 2, 3 ,4, 5.....,n

$$S_n = 1+2+3+\dots+n$$

### III. Dạng toán vận dụng công thức tính tổng các số hạng của dãy số cách đều.

Đối với dạng này ở bậc học cao hơn như THPT các em sẽ có công thức tính theo cấp số cộng hoặc cấp số nhân, còn với lớp 6 các em dựa vào cơ sở lý thuyết sau:

- Để đếm được số hạng của 1 dãy số mà 2 số hạng liên tiếp cách đều nhau 1 số đơn vị ta dùng công thức:

$$\text{Số số hạng} = [(số cuối - số đầu):(khoảng cách)] + 1$$

- Để tính Tổng các số hạng của một dãy mà 2 số hạng liên tiếp cách đều nhau 1 số đơn vị ta dùng công thức:

$$\text{Tổng} = [(số đầu + số cuối).(số số hạng)] : 2$$

\* Ví dụ 1: Tính tổng:  $S = 1 + 3 + 5 + 7 + \dots + 39$

◦ Hướng dẫn:

- Số số hạng của S là:  $(39-1):2+1 = 19+1 = 20$ .

$$S = [20.(39+1)] : 2 = 10.40 = 400.$$

\* Ví dụ 2: Tính tổng:  $S = 2 + 5 + 8 + \dots + 59$

◦ Hướng dẫn:

- Số số hạng của S là:  $(59-2):3+1 = 19+1 = 20$ .

$$S = [20.(59+2)] : 2 = 10.61 = 610.$$

## Chuỗi thông dụng

$$S = 1 + 2 + 3 + \dots + n = n(n+1)/2 \approx n^2/2$$

$$S = 1 + 2^2 + 3^2 + \dots + n^2 = n(n+1)(2n+1)/6 \approx n^3/3$$

$$S = 1 + a + a^2 + a^3 + \dots + a^n = (a^{n+1} - 1)/(a-1)$$

Nếu  $0 < a < 1$  thì

$$S \leq 1/(1-a)$$

và khi  $n \rightarrow \infty$  thì

$$S \text{ tiến về } 1/(1-a)$$

$$S = 1 + 1/2 + 1/3 + \dots + 1/n = \ln(n) + \gamma$$

Hằng số Euler  $\gamma \approx 0.577215665$

$$S = 1 + 1/2 + 1/4 + 1/8 + \dots + 1/2^n + \dots \approx 2$$

Nguyễn Thái Dư - AGU

5

9. Lũy thừa :  $a, b > 0$

$a^\alpha \cdot a^\beta \cdot a^\gamma = a^{\alpha + \beta + \gamma}$	$\frac{a^\alpha}{b^\alpha} = \left(\frac{a}{b}\right)^\alpha$
$(a^\alpha)^\beta = a^{\alpha\beta}$	$a^\alpha \cdot b^\alpha = (a \cdot b)^\alpha$
$\sqrt[n]{a^k} = a^{\frac{k}{n}}$	$\sqrt[m]{\sqrt[n]{a^k}} = \sqrt[m \cdot n]{a^k} = a^{\frac{k}{m \cdot n}}$

10. Logarit :  $0 < N_1, N_2, N$  và  $0 < a, b \neq 1$  ta có

$\log_a N = M \Leftrightarrow N = a^M$	$\log_a \left(\frac{N_1}{N_2}\right) = \log_a N_1 - \log_a N_2$
$\log_a a^M = M$	$\log_a N^\alpha = \alpha \log_a N$
$a^{\log_a N} = N$	$\log_a N = \frac{1}{\alpha} \log_a N$
$N_1^{\log_a N_2} = N_2^{\log_a N_1}$	$\log_a N = \frac{\log_b N}{\log_b a}$
$\log_a(N_1 \cdot N_2) = \log_a N_1 + \log_a N_2$	$\log_a b = \frac{1}{\log_b a}$

## Một số tính chất của hàm Log

◆ Cho  $a$  dương và  $a$  khác 1,  $b$  dương và số thực  $\alpha$  thì :

$$\log_a b = \alpha \Leftrightarrow a^\alpha = b$$

$$\text{Do đó: } \log_a 1 = 0; \log_a a = 1$$

$$\log_a a^b = b, \forall b \in \mathbb{R}$$

$$a^{\log_a b} = b, \forall b > 0$$

Nguyễn Thái Dư - AGU

6

$$(a)^0 = 1; (a \neq 0)$$

$$(a)^1 = a$$

$$(a)^{-\alpha} = \frac{1}{a^\alpha}$$

$$(a)^\alpha \cdot (a)^\beta = (a)^{\alpha+\beta}$$

$$\frac{(a)^\alpha}{(a)^\beta} = (a)^{\alpha-\beta}$$

$$(a)^\alpha \cdot (b)^\alpha = (ab)^\alpha$$

$$\frac{(a)^\alpha}{(b)^\alpha} = \left(\frac{a}{b}\right)^\alpha; (b \neq 0)$$

$$(a)^{\frac{\alpha}{\beta}} = \sqrt[\beta]{(a)^\alpha} (\beta \in \mathbb{N}^*)$$

$$(a)^\alpha = b$$

$$\Rightarrow \alpha = \log_a b$$

$$(a^\alpha)^\beta = a^{\alpha \cdot \beta}$$

$$\log_a 1 = 0 (0 < a \neq 1)$$

$$\log_a a = 1 (0 < a \neq 1)$$

$$\log_a a^\alpha = \alpha (0 < a \neq 1)$$

$$\log_{a^\alpha} a = \frac{1}{\alpha} (0 < a \neq 1)$$

$$\log_a b^\alpha = \alpha \log_a b;$$

$$(a, b > 0, a \neq 1)$$

$$\log_{a^\beta} b = \frac{1}{\beta} \log_a b$$

$$\log_{a^\beta} b^\alpha = \frac{\alpha}{\beta} \log_a b$$

$$\log_a b + \log_a c$$

$$= \log_a (b \cdot c)$$

$$\log_a b - \log_a c$$

$$= \log_a \left(\frac{b}{c}\right)$$

$$\log_a b = \frac{1}{\log_b a}$$

7

Nguyễn Thái Dư - AGU

8

## Tổng của chuỗi số

### ◆ 1.2 Định nghĩa 2:

- ◆ Tổng n hữu hạn số hạng đầu của chuỗi gọi là tổng riêng phần thứ n của chuỗi (*sequence of partial sum*)

$$S_n = u_1 + u_2 + u_3 + \dots + u_n = \sum_{i=1}^n u_i$$

◆ Nếu  $\lim_{n \rightarrow \infty} S_n = S$  hữu hạn thì ta nói chuỗi hội tụ (*convergent*).

◆ Nếu  $\lim_{n \rightarrow \infty} S_n = \pm\infty$  hoặc không tồn tại ta nói chuỗi phân kỳ (*divergent*)

Thí dụ 1.2.1:

Nếu  $|q| < 1$  thì  $S_n \xrightarrow{n \rightarrow \infty} \frac{1}{1-q}$ , do đó chuỗi hội tụ và có tổng bằng  $\frac{1}{1-q}$

Xét chuỗi cấp số nhân:  $\sum_{n=0}^{\infty} q^n$  (*geometric series*)

Nếu  $q > 1$  thì  $S_n$  không có giới hạn hữu hạn, do đó chuỗi phân kỳ.

Ta có:  $S_n = 1 + q + \dots + q^n$

Nếu  $q = -1$  thì  $S_n = 1 - 1 + 1 - 1 + \dots$  do đó  $S_n = \begin{cases} 0 \\ 1 \end{cases}$

Nếu  $q = 1$  ta có:  $S_n = n \Rightarrow \lim_{n \rightarrow \infty} S_n = +\infty$

$$S_n = \sum_{k=0}^{n-1} q^k = \frac{q^n}{q-1} - \frac{1}{q-1}$$

Vậy  $S_n$  không có giới hạn và chuỗi đã cho phân kỳ.

Vậy chuỗi phân kỳ.  
Nguyễn Thái Dư - AGU

9

## Tổng của chuỗi số

Thí dụ 1.2.1:

Xét chuỗi cấp số nhân:  $\sum_{n=0}^{\infty} q^n$  (*geometric series*)

Ta có:  $S_n = 1 + q + \dots + q^n$

Nếu  $q = 1$  ta có:  $S_n = n \Rightarrow \lim_{n \rightarrow \infty} S_n = +\infty$

Vậy chuỗi phân kỳ.  $S_n = \sum_{k=0}^{n-1} q^k = \frac{q^n}{q-1} - \frac{1}{q-1}$

Nguyễn Thái Dư - AGU

10

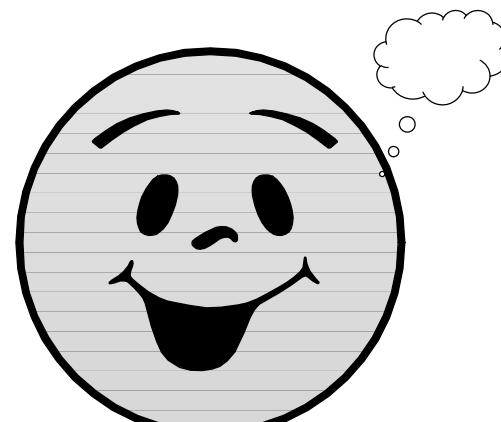
## Tổng của chuỗi số

Nếu  $|q| < 1$  thì  $S_n \xrightarrow{n \rightarrow \infty} \frac{1}{1-q}$ , do đó chuỗi hội tụ và có tổng bằng  $\frac{1}{1-q}$

Nếu  $q > 1$  thì  $S_n$  không có giới hạn hữu hạn, do đó chuỗi phân kỳ.

Nếu  $q = -1$  thì  $S_n = 1 - 1 + 1 - 1 + \dots$  do đó  $S_n = \begin{cases} 0 \\ 1 \end{cases}$

Vậy  $S_n$  không có giới hạn và chuỗi đã cho phân kỳ.



Cảm ơn !