TRƯỜNG CAO ĐẲNG CNTT HỮU NGHỊ ViỆT - HÀN

KHOA KHOA HỌC MÁY TÍNH

----------***----------

THUẬT TOÁN

(Algorithms)

Nội Dung

C1 THUẬT TOÁN VÀ ĐỘ PHỨC TẠP

C2 CHIA ĐỂ TRỊ

C3 QUY HOẠCH ĐỘNG

C4 THUẬT TOÁN THAM LAM

C5 THUẬT TOÁN QUAY LUI

CHIA ĐỂ TRỊ

2.1 Thuật toán chia để trị tổng quát

- Giả sử rằng, thuật toán phân chia một bài toán cỡ n thành a bài toán nhỏ. Trong đó mỗi bài toán nhỏ có cỡ n/b.

  Cũng vậy, ta giả sử rằng tổng các phép toán thêm vào khi thực hiện phân chia và tổng hợp lời giải của bài toán là g(n).

  Khi đó nếu f(n) là số các phép toán cần thiết để giải bài toán đã cho, thì f thỏa mãn hệ thức truy hồi sau đây:

- F(n) = a.f(n/b) +g(n).

2.1 Thuật toán chia để trị tổng quát

Dưới đây là nội dung của thuật toán chia để trị:

Main D_and_C(n)

{

Nếu n <= n_0 thì (n_0 là kích thước đủ nhỏ *)*

Giải bài toán một cách trực tiếp

Ngược lại

i. Chia bài toán thành a bài toán con kích thước n/b

ii. Cho (Mỗi bài toán trong a bài toán con) thực Hiện D_and_C(n/b)

iii. Tổng hợp lời giải của a bài toán con để thu được lời giải của bài toán gốc

}

CHIA ĐỂ TRỊ

2.2 Một số thí dụ minh họa

- 2.2.1 *Bài toán tìm kiếm nhị phân*
- 2.2.2 *Bài toán phép nhân các số nguyên lớn*
- 2.2.3 *Bài toán nhân ma trận*
- 2.2.4 *Bài toán dãy con lớn nhất*
- 2.2.5 *Bài toán sắp xếp*
- 2.2.6 *Bài toán lũy thừa*

2.2.1 Bài toán tìm kiếm nhị phân

- **Bài toán:** Cho số x và mảng A[1..n] các số nguyên được sắp xếp theo thứ tự không giảm. Tìm i sao cho A[i] = x. (Giả thiết i tồn tại).

- Phân tích bài toán:

  Số x cho trước:

  + Hoặc là bằng phần tử nằm ở vị trí giữa mảng A
  + Hoặc là nằm ở nửa bên trái (x < phần tử ở giữa mảng A)
  + Hoặc là nằm ở nửa bên phải (x > phần tử ở giữa mảng A)

2.2.1 Bài toán tìm kiếm nhị phân

Từ nhận xét đó ta có giải thuật sau:

```
Index  location(index  low, index  hight)
{
  Index  mid;
  If (low > hight)   return 0;
  Else {
        mid =       (low    hight)/2
        If  (x == A[mid])  return mid
       Else
             If  (x < A[mid])  return  location(low,  mid - 1)
            Else
                  Return    location (mid + 1, hight);
    }
       }
```

2.2.1 Bài toán tìm kiếm nhị phân

Thí dụ:

Giả sử ta cần tìm x = 18 trong dãy

A = {10, 12, 13, 14, 18, 20, 25, 27, 30, 35, 40, 45, 47}.

ở đây n =13.

Ta thực hiện như sau:

Đầu tiên ta tính mid = (1 + 13)/2 = 7 => A[7] = 25

Vì x = 18 < 25 nên ta tìm trên dãy nhỏ A_1 = {10, 12, 13, 14, 18, 20}

Ta tìm số ở giữa mới đó là mid_1 = (1 + 6)/2 = 3 => A1[3] = 13

Vì x = 18 > 13 nên ta tìm trên dãy lớn A_{12} = {14, 18, 20}

Ta lại tiếp tục tìm phần tử giữa của dãy con mới

mid_2 = (4 + 6)/2 = 5 => A_{12}[5] = 18

Thông báo chỉ số i = 5 và dừng thuật toán.

2.2.1 Bài toán tìm kiếm nhị phân

❖ **Độ phức tạp của thuật toán**:

Gọi T(n) là thời gian cần thiết để thực hiện thuật toán. Khi đó:

$$T(n) \quad \begin{array}{ll} 1 & n \quad 1 \\ T(\frac{n}{2}) \quad 1 & n \quad 1 \end{array}$$

Theo định lý thợ (xem phụ lục A) ta có

$$T(n) \quad O(\log_2 n)$$

2.2 Một số thí dụ minh họa

- 2.2.1 *Bài toán tìm kiếm nhị phân*
- 2.2.2 *Bài toán phép nhân các số nguyên lớn*
- 2.2.3 *Bài toán nhân ma trận*
- 2.2.4 *Bài toán dãy con lớn nhất*
- 2.2.5 *Bài toán sắp xếp*
- 2.2.6 *Bài toán lũy thừa*

2.2.2 Bài toán phép nhân các số nguyên lớn

- Thuật toán cổ điển để nhân hai số nguyên có n chữ số là $O(n^2)$.
- Năm 1962 A.A. Karatsuba đã khám phá bằng cách rút gọn phép nhân hai số nguyên n chữ số, xuống thành bốn phép nhân hai số nguyên n/2 chữ số như sau:

2.2.2 Bài toán phép nhân các số nguyên lớn

- Input: $x \quad x_{n-1}x_{n-2}\ldots x_1x_0$ và $y \quad y_{n-1}y_{n-2}\ldots y_1y_0$
- Output: $z \quad x*y \quad z_{2n-1}z_{2n-2}\ldots z_1z_0$
- Ta biết rằng:

$$x \quad \sum_{i=0}^{n-1} x_i*10^i \quad x_{n-1}*10^{n-1} \quad x_{n-2}*10^{n-2} \quad \ldots \quad x_1*10^1 \quad x_0*10^0$$

$$y \quad \sum_{i=0}^{n-1} y_i*10^i \quad y_{n-1}*10^{n-1} \quad y_{n-2}*10^{n-2} \quad \ldots \quad y_1*10^1 \quad y_0*10^0$$

$$z \quad x*y \quad \sum_{i=0}^{2n-1} z_i*10^i \quad \left(\sum_{i=0}^{n-1} x_i*10^i\right)*\left(\sum_{i=0}^{n-1} y_i*10^i\right)$$

2.2.2 Bài toán phép nhân các số nguyên lớn

❖ Ví dụ:

$207 = 2*10^2 + 0*10^1 + 7*10^0$

$702 = 7*10^2 + 0*10^1 + 2*10^0$

$207*702 = 145314$

$= 1*10^5 + 4*10^4 + 5*10^3 + 3*10^2 + 1*10^1 + 4*10^0$

❖ Bây giờ giả sử ta đặt:

$a \quad x_{n\ 1}x_{n\ 2}\ldots x_{n/2} \qquad b \quad x_{(n/2)\ 1}x_{(n/2)\ 2}\ldots x_0$

$c \quad y_{n\ 1}y_{n\ 2}\ldots y_{n/2} \qquad d \quad y_{(n/2)\ 1}y_{(n/2)\ 2}\ldots y_0$

Khi đó: $x \quad a*10^{n/2} \quad b \qquad y \quad c*10^{n/2} \quad d$

$z \quad x*y \quad (a*10^{n/2} \quad b)(c*10^{n/2} \quad d) \quad (a*c)*10^n \quad (a*d \quad b*c)*10^{n/2} \quad (b*d)$

2.2.2 Bài toán phép nhân các số nguyên lớn

❖ Ví dụ:

với n = 4; x = 1026 và y = 3547

thì a = 10, b = 26, c = 35, d = 47

Khi đó

$x = 1026 = 10*10^2 + 26 = a*10^2 + b;$

$y = 3547 = 35*10^2 + 47 = c*10^2 + d$

và z = x*y

$= (10*35)*10^4 + (10*47+26*35)*10^2 + (26*47)$

$= 350*10^4 + 1380*10^2 + 1222$

= 3.639.222

2.2.2 Bài toán phép nhân các số nguyên lớn

- Khi đó ta có thời gian thực hiện thuật toán là:

$$T(n) \quad \begin{array}{ll} 1 & n \quad 1 \\ 4T(\frac{n}{2}) \quad cn & n \quad 1 \end{array}$$

- Theo định lý thợ ta có độ phức tạp của thuật toán là .

$$T(n) \quad O(n^2)$$

2.2.2 Bài toán phép nhân các số nguyên lớn

❖ Nhân 981 với 1234.

Tách từng toán hạng thành hai nửa:

0981 cho ra a = 09 và b = 81,

1234 thành c = 12 và d = 34.

Lưu ý rằng $981 = 10^2a + b$ và $1234 = 10^2c + d$.

Do đó, tích cần tìm có thể tính được là

$$981 \times 1234 = (10^2a + b)(10^2c + d)$$
$$= 10^4ac + 10^2(ad + bc) + bd$$
$$= 1080000 + 127800 + 2754 = 1210554$$

Thủ tục trên đến bốn phép nhân hai nửa: ac, ad, bc và bd.

2.2.2 Bài toán phép nhân các số nguyên lớn

- Hãy xét tích:
- r = (a + b)(c + d) = ac + (ad + bc) + bd

- p = ac = 09 * 12 = 108
- q = bd = 81 * 34 = 2754
- r = (a + b)(c+d) = 90 * 46 = 4140
- và cuối cùng
- 981 x 1234 = 10^4p + 10^2(r – p – q) + q
- = 1080000 + 127800 + 2754 = 1210554.
- Như vậy tích của 981 và 1234 có thể rút gọn về ba phép nhân của hai số có hai chữ số (09*12, 81*34 và 90*46) cùng với một số nào đó phép dịch chuyển (nhân với luỹ thừa của 10), phép cộng và phép trừ.

2.2.2 Bài toán phép nhân các số nguyên lớn

- Sau đây là mô hình cải tiến thuật toán nhân số nguyên lớn
- Cải tiến để còn lại 3 phép nhân :

$$\text{Đặt} \quad U = a \times c; V = b \times d; W = (a+b) \times (c+d)$$

$$\Rightarrow a \times d + b \times c = W - U - V$$

$$\Rightarrow Z = U \times 10^{n} + (W - U - V) \times 10^{n/2} + V$$

- Từ đó ta đưa ra thuật toán nhân số nguyên lớn là:

2.2.2 Bài toán phép nhân các số nguyên lớn

```
Large_integer Karatsuba(x, y, n);
{
   If  n == 1  Return x*y
  Else
  {
   a = x[n-1]. . . x[n/2]; b = x[n/2-1] . . . x[0];
   c = y[n-1]. . . y[n/2]; d = y[n/2-1] . . . y[0];
   U = Karatsuba(a, c, n/2);
   V = Karasuba(b, d, n/2);
   W = Karatsuba(a+b, c+d, n/2);
   Return U*10^n + (W-U-V)*10^(n/2) + V
  }
}
```

2.2.2 Bài toán phép nhân các số nguyên lớn

- ***Độ phức tạp thuật toán:***
- *Gọi T(n) là thời gian cần thiết để thực hiện thuật toán. Khi đó:*

$$T(n) \begin{cases} 1 & n \quad 1 \\ 3T(\frac{n}{2}) \quad cn & n \quad 1 \end{cases}$$

- *Theo định lý thợ $T(n) = O(n\log 3) \approx O(n^{1.58})$*

2.2 Một số thí dụ minh họa

- 2.2.1 *Bài toán tìm kiếm nhị phân*
- 2.2.2 *Bài toán phép nhân các số nguyên lớn*
- 2.2.3 *Bài toán nhân ma trận*
- 2.2.4 *Bài toán dãy con lớn nhất*
- 2.2.5 *Bài toán sắp xếp*
- 2.2.6 *Bài toán lũy thừa*

2.2.3 Bài toán nhân ma trận

❖ **Bài toán**:

Cho hai ma trận A, B với kích thước n*n, ma trận C là ma trận tích của hai ma trận A và B. Thuật toán nhân ma trận cổ điển như công thức dưới đây:

C = A*B hay

$$c_{ij} \quad {}_{n\ n} \qquad \overset{n}{\underset{k\ 1}{}} a_{ik} b_{kj}$$

Với thời gian thực hiện là $O(n^3)$ (n là kích thước của ma trận)

2.2.3 Bài toán nhân ma trận

❖ Xét trường hợp n = 2 thì

ta có: $A \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix}$, $B \begin{matrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{matrix}$ và $C \begin{matrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{matrix}$

Khi đó: $\begin{matrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{matrix} \quad \begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix} \quad \begin{matrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{matrix}$

Với

$$\begin{matrix} c_{11} & a_{11}b_{11} & a_{12}b_{21} \\ c_{12} & a_{11}b_{12} & a_{12}b_{22} \\ c_{21} & a_{21}b_{11} & a_{22}b_{21} \\ c_{22} & a_{21}b_{12} & a_{22}b_{22} \end{matrix}$$

Ta thấy thuật toán trên có ít nhất đến 8 phép nhân.

2.2.3 Bài toán nhân ma trận

- Strassen đã cải thiện lại thuật toán trên bằng cách đặt:

$$
\begin{aligned}
m_1 &\quad (a_{11} \quad a_{22})(b_{11} \quad b_{22}) \\
m_2 &\quad (a_{21} \quad a_{22})b_{11} \\
m_3 &\quad a_{11}(b_{12} \quad b_{22}) \\
m_4 &\quad a_{22}(b_{21} \quad b_{11}) \\
m_5 &\quad (a_{11} \quad a_{12})b_{22} \\
m_6 &\quad (a_{21} \quad a_{11})(b_{11} \quad b_{12}) \\
m_7 &\quad (a_{12} \quad a_{22})(b_{21} \quad b_{22})
\end{aligned}
$$

Thuật toán chỉ còn 7 phép nhân. Khi đó ma trận C là ma trận tích của hai ma trận A và B:

$$
C \quad \begin{matrix} m_1 & m_4 & m_5 & m_7 \\ & m_2 & m_4 & \end{matrix} \quad \begin{matrix} & & m_3 & m_5 \\ m_1 & m_2 & m_3 & m_6 \end{matrix}
$$

2.2.3 Bài toán nhân ma trận

❖ **Ví dụ: Cho 2 ma trận 2x2 như sau:**

$$A = \begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix} \qquad B = \begin{matrix} 1 & 2 \\ 3 & 2 \end{matrix}$$

Thực hiện nhân 2 ma trân trên:

1. Sử dụng thuật toán nhân cổ điển
2. Sử dụng thuật toán Strassen

2.2.3 Bài toán nhân ma trận

- Gọi T(n) là thời gian cần thiết để nhân hai ma trận kích thước n*n. Giả sử rằng n là luỹ thừa bậc 2.
- Thời gian để tính phép cộng, trừ ma trận là $O(n^2)$
- Do đó $T(n) = 7T(n/2) + dn^2$
- Áp dụng định lý thợ ta có $T(n) \quad O(n^{\log_2 7})$
- Đối với các trường hợp ma trận vuông nhưng kích thước không phải là luỹ thừa bậc 2 thì giải quyết vấn đề bằng cách thêm các dòng và các cột sao cho kích thước ma trận mới gấp đôi kích thước ma trận cũ và gán giá trị cho các phần tử mới thêm là 0.
- Do lg7<2,81 nên có thể thực hiện phép nhân hai ma trận kích thước n*n trong thời gian $O(n^{2.81})$

2.2 Một số thí dụ minh họa

- 2.2.1 *Bài toán tìm kiếm nhị phân*
- 2.2.2 *Bài toán phép nhân các số nguyên lớn*
- 2.2.3 *Bài toán nhân ma trận*
- 2.2.4 *Bài toán dãy con lớn nhất*
- 2.2.5 *Bài toán sắp xếp*
- 2.2.6 *Bài toán lũy thừa*

2.2.4 Bài toán dãy con lớn nhất

- **Bài toán:** Cho mảng A[1..n]. Mảng A[p..q] được gọi là mảng con của A. Trọng lượng mảng bằng tổng các phần tử. Tìm mảng con có trọng lượng lớn nhất (1<= *p* <= *q* <= *n*).

2.2.4 Bài toán dãy con lớn nhất

- **Thiết kế thuật toán**
- ***a/ Thuật toán đơn giản***
- Để đơn giản ta chỉ xét bài toán tìm trọng lượng của mảng con lớn nhất còn việc tìm vị trí thì chỉ là thêm vào bước lưu lại vị trí trong thuật toán.
- Ta có thể dễ dàng đưa ra thuật toán tìm kiếm trực tiếp bằng cách duyệt hết các dãy con có thể của mảng A như sau:

2.2.4 Bài toán dãy con lớn nhất

```
void    BruteForceNaice;
{
    Max1 =  -MaxInt;
    for (i = 1; i<= n; i++)              // i là điểm bắt đầu của dãy con
        for( j =i; j<= n; j++)          // j là điểm kết thúc của dãy con
            {
                s= 0;
              for ( k  = i; k<= j; k++)     // Tính trọng lượng của dãy
               s = s + A[k]
               if  (s > Max1)   Max1 = S
        }
}
```

2.2.4 Bài toán dãy con lớn nhất

- *Phân tích độ phức tạp của thuật toán trên:*
  Lấy s = s + A[k] làm câu lệnh đặc trưng, ta có số lần thực hiện câu lệnh đặc trưng là $\sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{j} k \quad O(n^3)$

- *Thời gian T(n) = O(n³)*
- *Nếu để ý, ta có thể giảm độ phức tạp của thuật toán bằng cách giảm bớt vòng lặp trong cùng (vòng lặp theo k).*

$$\sum_{k=1}^{j} a[k] \quad a[j] \qquad \sum_{k=1}^{j-1} a[k]$$

2.2.4 Bài toán dãy con lớn nhất

❖ *Khi đó thuật toán có thể được viết một cách tóm tắt như sau:*

```
for  ( i = 1; i<=  n; i++)
    for ( j = i; j<=  n; j++)
        {
            s  = s + A[j];        //Câu lệnh đặc trưng
           if   (s > max1)   max1  = s;
        }
```

Lấy s = s + A[j] làm câu lệnh đặc trưng thì ta có số lần thực hiện câu lệnh đặc trưng là

$$\sum_{i=1}^{n}\sum_{j=i}^{n} j \quad O(n^2)$$

=> Thời gian của thuật toán T(n) = O(n²)

2.2.4 Bài toán dãy con lớn nhất

- ***b/ Cách tiếp cận chia để trị***

- ***Chia:*** *Chia mảng A ra thành hai mảng con với chênh lệch độ dài ít nhất, kí hiệu là AL , AR.*

- ***Trị:*** *Tính mảng con lớn nhất của mỗi nửa mảng A một cách đệ quy. Gọi WL, WR là trọng lượng của mảng con lớn nhất trong AL, AR.*

- ***Tổng hợp:*** *Max (WL, WR).*
  WM = WML + WMR

2.2.4 Bài toán dãy con lớn nhất

❖ ***Cài đặt thuật toán:***

```
void    MaxSubVector(A, i, j);
{
    If ( i == j)   return   a[i]
   Else
     {
         m    = (i + j)/2;
         WL   = MaxSubVector(a, i, m);
         WR    = MaxSubVector(a, m+1, j);
         WM  = MaxLeftVetor(a, i, m) + MaxRightVector(a, m+1, j);
       Return   Max(WL, WR, WM )
   }
}
```

2.2.4 Bài toán dãy con lớn nhất

❖ ***Các hàm MaxLeftVector, Max RightVector được cài đặt như sau:***

```
void      MaxLeftVector(a, i, j);
     {
                    MaxSum = -Maxint ; Sum = 0;
                    for( k = j;k>= i;k--)
                   {
                              Sum = Sum + A[k];
                              MaxSum = Max(Sum,MaxSum);
              }
              Return  MaxSum;
    }
```

2.2.4 Bài toán dãy con lớn nhất

❖ *Tương tự với hàm MaxRightVector là.*

```
for (k = i;k<= j;k++)
    {
        Sum = Sum + A[k];
        MaxSum = MaxSum(Sum, MaxSum);
    }
```

2.2.4 Bài toán dãy con lớn nhất

❖ ***Phân tích độ phức tạp***

Thời gian chạy thủ tục MaxLeftVector và MaxRightVector là O(m)

(m = j - i + 1)

Gọi T(n) là thời gian tính, giả thiết n = 2^k .

Ta có

n = 1 thì T(n) = 1

n > 1 thì việc tính WM đòi hỏi thời gian n/2 + n/2 = n

=> T(n) = 2T(n/2) + n $T(n) \quad O(\log n)$

Theo định lý thợ ta có

2.2 Một số thí dụ minh họa

- 2.2.1 *Bài toán tìm kiếm nhị phân*
- 2.2.2 *Bài toán phép nhân các số nguyên lớn*
- 2.2.3 *Bài toán nhân ma trận*
- 2.2.4 *Bài toán dãy con lớn nhất*
- 2.2.5 *Bài toán sắp xếp*
- 2.2.6 *Bài toán lũy thừa*

2.2.5 Bài toán sắp xếp

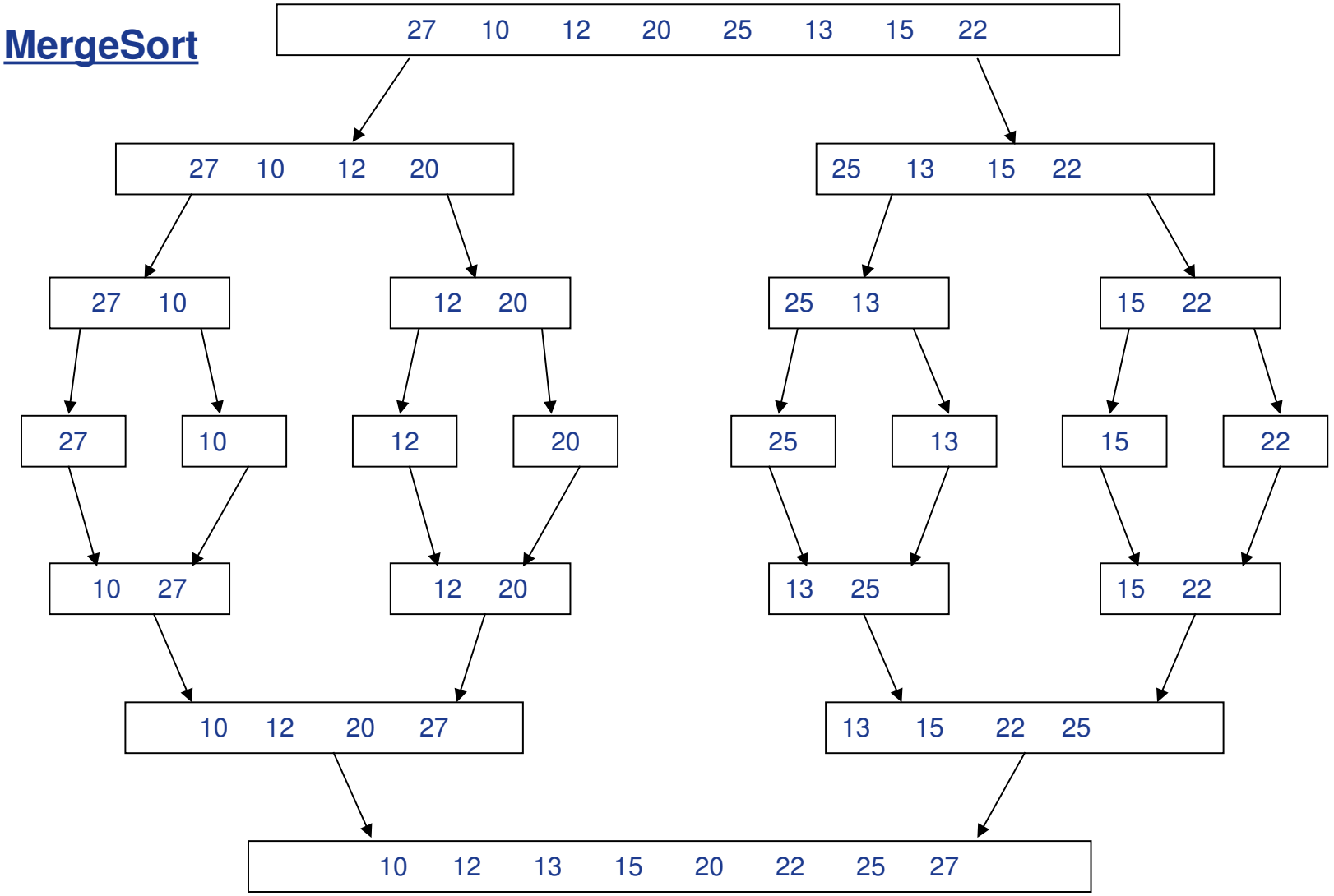
- Cho A[1..n] là một mảng n phần tử. Hãy sắp xếp các phần tử của A theo thứ tự không giảm.
- Như chúng ta đã biết thời gian dùng Selection Sort hay Insertion Sort để sắp xếp mảng A trong cả hai trường hợp: xấu nhất và trung bình đều vào cỡ n^2. Còn HeapSort vào khoảng n.logn.
- Có một số giải thuật đặc biệt cho bài toán này theo mô hình chia để trị đó là ***MergeSort*** và ***QuickSort,*** chúng ta sẽ lần lượt nghiên cứu chúng.

2.2.5 Bài toán sắp xếp

- **a. MergeSort**
- Để sắp xếp mảng A[1..n] với n là kích thước của A.
- Thuật toán sắp xếp bằng phương pháp MergeSort được trình bày dựa trên ý tưởng của kỹ thuật **chi để trị** được mô tả theo 3 bước sau:
- Bước chia: nếu n =1 thì return A ngược lại chia A thành hai dãy con A1[1..n/2], A2[1+n/2..n].
- Bước đệ quy: gọi lại bước 1 cho A1, A2.
- Bước trị: kết hợp A1, A2 đã có thứ tự thành mảng A ban đầu.
- Thí dụ sau mô tả trực quan giải thuật trên bằng cây nhị phân dưới đây, với mỗi nút của cây là một hàm đệ quy của MergeSort.

2.2.5 Bài toán sắp xếp

MergeSort



2.2.5 Bài toán sắp xếp

- Để trộn hai mảng con A1, và A2 đã có thứ tự thành mảng A ban đầu cũng có thứ tự ta thực hiện theo các bước sau:
- Gọi h, m lần lượt là số phần tử của hai mảng con A1 và A2
- Bước 1: Gán i, j, k bằng 1
- Bước 2: Trong khi i<= h và j<= m so sánh A1[i] với A2[j]
- Bước 3: Nếu A1[i] < A2[j] thì đẩy phần tử thứ i của A1 vào A và tăng i một đơn vị ngược lại đẩy A2[j] vào A và tăng j một đơn vị.
- Bước 4: Tăng k một đơn vị
- Bước 5: Nếu h > m thì đẩy tất cả những phần tử còn lại của A1 vào A, ngược lại đẩy tất cả các phần tử còn lại của A2 vào A.

2.2.5 Bài toán sắp xếp

❖ Dưới đây là thủ tục của thuật toán trên:

```
void  Merge(h, m, A1, A2, A )   // h: số phần tử của A1; m = n – h: số phần tử của A2
{
    Index i, j, k;
    i = 1; j = 1; k = 1;
    while (i <= h && j <= m)
   {
      If (A1[i] < A2[j])
     {
         A[k] = A1[i];
          i++;
     }
    else {
         A[k] = A2[j];
         j++;
         }
      k++;
   }
 if(i > h)
   copy A2[j..m] to A[k..h + m];
 else
   copy A1[i..h] to A[k..h + m];
}
```

2.2.5 Bài toán sắp xếp

- Thuật toán sắp xếp trộn (MergeSort) sau đây có thể tốt hơn nếu các mảng A1 và A2 là các biến toàn cục và xem việc sắp xếp chèn Insert(A) như là giải thuật cơ bản

```
Void MergeSort(int n, keytype A[])
{
If (n >1) {
  Const int   , m = n – h;   /* trong đó   x   là phần nguyên lớn nhất không quá x
  Keytype A1[1..h], A2[1..m];
  Copy A[1..h] to A1[1..h];
  Copy A[h +1..n] to A2[1..m];
  MergeSort(h, A1);
  MergeSort(m, A2);
   Merge(h, m, A1, A2, A);
  }
}
```

2.2.5 Bài toán sắp xếp

- **Độ phức tạp của thuật toán**
- Giả sử T(n) là thời gian cần thiết để thuật toán này sắp xếp một mảng n phần tử. Việc tách A thành A1 và A2 là tuyến tính. Ta cũng dễ thấy Merge(A1, A2, A) cũng tuyến tính. Do vậy:
- $$T(n) \quad T(n/2) \quad T(n/2) \quad g(n)$$
- trong đó g(*n*) = O(*n*)
- hay T(*n*) = 2T(*n*/2) +g(*n*)
- Theo định lý thợ ta có: a = 2; b = 2
- Nên T(n) = O(n.logn).

2.2.5 Bài toán sắp xếp

- **b. <u>Quicksort</u>**
- Thuật toán này được phát minh bởi C.A.R Hoare vào năm 1960 và chính thức giới thiệu vào năm 1962, nó được hiểu như là tên gọi của nó – 'sắp xếp nhanh', hơn nữa nó cũng dựa theo nguyên tắc chia để trị.

2.2.5 Bài toán sắp xếp

- Bước 1 chọn ngẫu nhiên một phần tử làm chốt (pivot) từ mảng A[i..j] cần sắp xếp.
- B2 ngăn mảng này ra 2 phần: các phần tử lớn hơn khóa chốt thì được chuyển về bên phải của nó (cuối dãy), ngược lại thì chuyển về bên trái (đầu dãy).
- Sau đó mỗi phần của mảng được sắp xếp độc lập bằng cách gọi đệ quy thuật toán này, và cuối cùng mảng sẽ được sắp xếp xong.

2.2.5 Bài toán sắp xếp

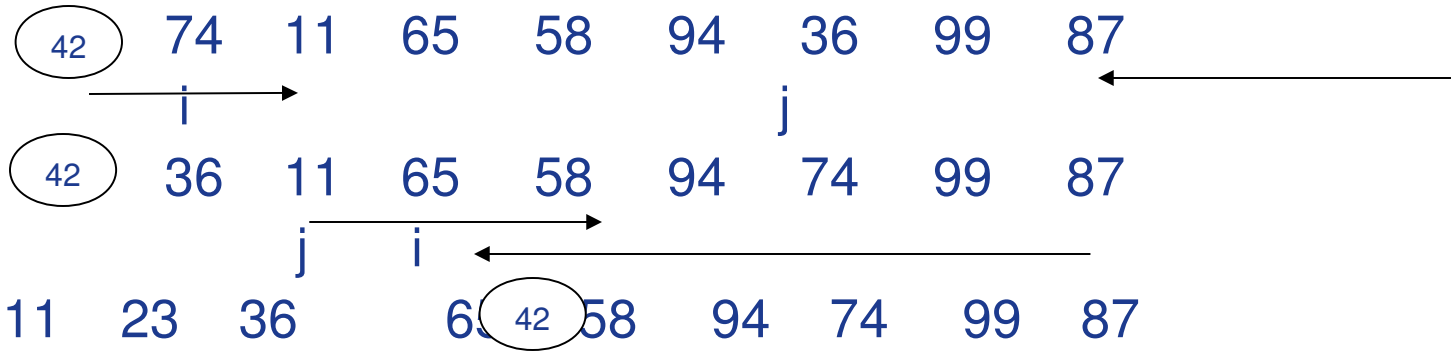
- Vấn đề đặt ra là làm thế nào để tìm chốt sao cho việc phân hoạch mảng đã cho thành 2 mảng con có kích thước cân bằng nhau.
- Để đơn giản, chúng ta chọn phần tử đầu tiên trong mảng làm chốt (tức là p = A[i]) *và hi vọng nó là tốt nhất có thể.*
- Tiếp đến, ta sử dụng hai biến, biến left bắt đầu từ i và chạy từ trái sang phải, biến k bắt đầu từ j+1 và chạy từ phải sang trái. Biến left tăng cho tới khi A[left] > p, còn biến k giảm cho tới khi A[k]<= p. nếu left < k thì hoán vị A[left] và A[k]. Lặp lại quá trình trên cho tới khi left > k. cuối cùng ta hoán vị A[i] và A[k] để đặt chốt vào đúng vị trí của nó.

2.2.5 Bài toán sắp xếp

❖ **Thí dụ**:

Giả sử ta cần sắp xếp dãy: 42 23 74 11 65 58 94 36 99 87.

Nếu ta chọn chốt là số đầu tiên thì p = 42. Ta tìm cách chuyển các số 11 23 36 về trước chốt. Nếu được vậy thì ta đã phân dãy ra thành hai dãy con như sau:



Hình 2.2 Mô phỏng sắp xếp bằng thuật toán Quicksort

2.2.5 Bài toán sắp xếp

❖ Dưới đây là thủ tục phân đoạn:

```
void  Partition(A[i..j], var k)
{
    P = A[i];
    left = i;
    k = j+ 1;
    do left = left + 1; while (A[left > p]) or (left >j) ;
    do k = k-1; while (A[k]<= p);
        While left < k do
        {
            Swap(A[left], A[k]) ;
            do left = left + 1; while (A[left > p]) or (left > j) ;
            do k = k-1; while (A[k]<= p);
        }
    Swap(A[i], A[k]) ;
}
```

2.2.5 Bài toán sắp xếp

- Sau đây là thuật toán sắp xếp với tên gọi là Quicksort dùng để sắp xếp mảng A[1..n]:

```
Void  Quicksort(A[1..n]);            // Sắp xếp theo thứ tự không giảm
{
    if n  đủ nhỏ  thì  Insert(A[1..n])
    else
    {
            Partition(A[1..n],k);
            quicksort(A[1..k-1]);
            quicksort(A[k+1,n]);
      }
}
```

2.2.5 Bài toán sắp xếp

2.1 Mô tả quá trình làm việc của pivot (chốt) và Quicksort

A[i] Lượt	**42**	23	74	11	65	58	94	36	99	87
1	**(11**	23	36)	42	(65	58	94	74	99	87)
2	11	**(23**	36)	42	(65	58	94	74	99	87)
3	11	23	**(36)**	42	**(65**	58	94	74	99	87)
4	11	23	36	42	**(58)**	65	(94	74	99	87)
5	11	23	36	42	58	65	**(94**	74	99	87)
6	11	23	36	42	58	65	**(87**	74)	94	(99)
7	11	23	36	42	58	65	**(74)**	87	94	(99)
8	11	23	36	42	58	65	74	87	94	**(99)**
9	11	23	36	42	58	65	74	87	94	99

2.2.5 Bài toán sắp xếp

- ***Độ phức tạp của thuật toán:***
- *để phân hoạch mảng n phần tử ta cần thời gian O(n).*
- *gọi T(n) là thời gian thực hiện QuickSort, chúng ta có quan hệ đệ quy sau:*

$$T(n) \begin{cases} O(1) & ,n \quad 1 \\ O(n) \quad T(n \quad 1) & ,n \quad 1 \end{cases}$$

2.2.5 Bài toán sắp xếp

- *Vậy ta có:*

T(n) = O(n) + T(n-1)

= O(n) + O(n-1) + T(n-2)

.....

$$\sum_{i=1}^{n} O(i) \quad O(n^2).$$

Như vậy, trong trường hợp xấu nhất QuickSort đòi hỏi thời gian $O(n^2)$.

- QuickSort có thể sắp xếp 1 mảng n phần tử khác nhau trong trường hợp trung bình là O(*n.logn*).

2.2 Một số thí dụ minh họa

- 2.2.1 *Bài toán tìm kiếm nhị phân*
- 2.2.2 *Bài toán phép nhân các số nguyên lớn*
- 2.2.3 *Bài toán nhân ma trận*
- 2.2.4 *Bài toán dãy con lớn nhất*
- 2.2.5 *Bài toán sắp xếp*
- 2.2.6 *Bài toán lũy thừa*

2.2.6 Bài toán lũy thừa

- Xét bài toán a^n với a, n là các số nguyên và n không âm. Thuật toán tính a^n được thực hiện bằng phương pháp lặp như sau:

```
int expose(a,n)
{ int result = 1;
      for (int i = 0; i <= n; ++i)
   result *= a;
 }
```

2.2.6 Bài toán lũy thừa

- Thuật toán này đòi hỏi thời gian tính cỡ O(*n*)
- lệnh *result = a.result* được thực hiện đúng *n* lần, với điều kiện phép nhân được tính là phép toán cơ bản.
- Tuy nhiên trong hầu hết các máy tính thậm chí với những giá trị nhỏ của *n* và *a* đã dẫn tới tràn bộ nhớ. Ví dụ 15^{17} đã không thể biểu diễn được trong 64-bit số nguyên.

2.2.6 Bài toán lũy thừa

- Một giải pháp để tăng hiệu quả của hàm *expose* là chia a^n thành $(a^{n/2})^2$ khi n chẵn.
- Đây là điều rất đáng chú ý vì $a^{n/2}$ có thể tính được nhanh gấp 4 lần so với a^n và với một phép bình phương đơn giản ta thu được kết quả từ $a^{n/2}$.

$$a^n = \begin{cases} 1 & n = 0 \\ (a^2)^{n/2} & \text{,n chẵn} \\ a(a^2)^{n/2} & \text{,n lẻ} \end{cases}$$

2.2.6 Bài toán lũy thừa

- Thí dụ: $a^{32} = ((((a^2)^2)^2)^2)^2$ chỉ bao hàm 5 phép nhân.
- $a^{31} = ((((a^2)a)^2a)^2a)^2a$ chỉ bao hàm 8 phép nhân.
- Từ phân tích trên đưa ra ý tưởng cho thuật toán sau:

```
(1)   int power(int a, int n)
(2)   { if (n = 0)
(3)       return 1;
(4)       else if (n %2 == 0)
(5)            return power(a*a,n/2)  // n chẵn
(6)           else
(7)           return a*power(a*a,n/2) //n lẽ
(8)       }
```

2.2.6 Bài toán lũy thừa

- Phân tích thời gian:
- gọi T(n) là thời gian thực hiện thuật toán. Khi đó ta có:

$$T(n) = \begin{cases} a & ,n = 0 \\ b + T(n/2) & ,n \text{ chẵn} \\ b + T(n/2) & ,n \text{ lẻ} \end{cases}$$

với a.b là những hằng số nào đó.

- Áp dụng định lý thợ ta có T(n) = O(logn).

Tổng kết chương

1. *Bài toán tìm kiếm nhị phân*
2. *Bài toán phép nhân các số nguyên lớn*
3. *Bài toán nhân ma trận*
4. *Bài toán dãy con lớn nhất*
5. *Bài toán sắp xếp*
6. *Bài toán lũy thừa*

Bài tập

Cài đặt các bài toán đã học trong chương 2

Nội dung nghiên cứu trước

Nghiên cứu trước chương 3

TRƯỜNG CAO ĐẲNG CNTT HỮU NGHỊ ViỆT - HÀN

KHOA KHOA HỌC MÁY TÍNH

----------***----------

Thank You !