# Phân tích và Thiết kế THUẬT TOÁN

Hà Đại Dương
duonghd@mta.edu.vn
Web: fit.mta.edu.vn/~duonghd

# Bài 5 - Chia để trị (tiếp)

PHÂN TÍCH VÀ THIẾT KẾ THUẬ TOÁN

## NỘI DUNG

I. Giới thiệu
II. Lược đồ chung
III. Bài toán áp dụng
IV. Bài tập

## III. Bài toán áp dụng

**5. Nhân số nguyên (lớn)**

- Bài toán: Nhân 2 số nguyên (lớn) x, y có n chữ số

$$x = x_{n-1}x_{n-2}...x_1x_0$$
$$y = y_{n-1}y_{n-2}...y_1y_0$$
$$z = x * y = z_{2n-1}z_{2n-2}...z_1z_0$$

Quá quen: Đến mức không cần phải thắc mắc về tính tối ưu của nó

➡ Cách thức vẫn làm (quá quen): Độ phức tạp $O(n^2)$

## III. Bài toán áp dụng

**5. Nhân số nguyên (lớn)**

- Ý tưởng: Chia để trị

**Đặt**

$$a = x_{n-1}x_{n-2}...x_{n/2} \qquad b = x_{(n/2)-1}x_{(n/2)-2}...x_0$$
$$c = y_{n-1}y_{n-2}...y_{n/2} \qquad d = y_{(n/2)-1}y_{(n/2)-2}...y_0$$

**Khi đó**

$$x = a*10^{n/2} + b \qquad y = c*10^{n/2} + d$$

**Và**

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

## III. Bài toán áp dụng

**5. Nhân số nguyên (lớn)**

- Ý tưởng: Chia để trị

x,y: có độ dài bằng nhau và độ dài có dạng $2^m$, nếu

- Có 1 chữ số: làm trực tiếp
- Có n chữ số: Tích của nó có thể biểu diễn qua tích của 4 số nguyên có độ dài n/2 chữ số

$$z = (a*c)*10^n + (a*d + b*c)*10^{n/2} + (b*d)$$

(và các phép cộng, dịch phải)

## III. Bài toán áp dụng

**5. Nhân số nguyên (lớn)**

- Ý tưởng: Chia để trị

$$z = (a*c)*10^{n} + (a*d + b*c)*10^{n/2} + (b*d)$$

Gọi T(n) là thời gian thực hiện phép nhân 2 số nguyên có độ dài n thì

T(n)=4T(n/2)+O(n)

(O(n) là thời gian thực hiện các phép cộng và dịch phải)

Giải công thức truy hồi trên ta được $T(n) = O(n^2)$

➡ **Chưa nhanh hơn nếu không chia để trị**

## III. Bài toán áp dụng

**5. Nhân số nguyên (lớn)**

- Ý tưởng: Năm 1962 nhà toán học người Nga **Anatoly Alexeevitch Karatsuba** (Karatsuba) đã tối ưu thời gian thực hiện phép nhân 2 số nguyên có n chữ số như sau:

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

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

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

Khi đó T(n) = 3T(n/2)+O(n)

Giải phương trình đệ qui ta được

$T(n) = O(n\log_2 3) \approx O(n^{1.585})$

## III. Bài toán áp dụng

**5. Nhân số nguyên (lớn)**

- Thuật toán: Karatsuba

```
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
  }
}
```

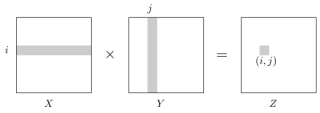
# III. Bài toán áp dụng

**6. Nhân ma trận**

The product of two $n \times n$ matrices $X$ and $Y$ is a third $n \times n$ matrix $Z = XY$, with $(i, j)$th entry

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

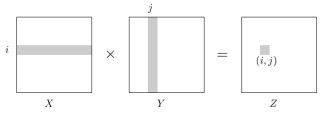
To make it more visual, $Z_{ij}$ is the dot product of the $i$th row of $X$ with the $j$th column of $Y$:




# III. Bài toán áp dụng

**6. Nhân ma trận**

$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$




In general, $XY$ is not the same as $YX$; matrix multiplication is not commutative.

The preceding formula implies an $O(n^3)$ algorithm for matrix multiplication: there are $n^2$ entries to be computed, and each takes $O(n)$ time. For quite a while, this was widely believed to be the best running time possible, and it was even proved that in certain models of computation no algorithm could do better. It was therefore a source of great excitement when in 1969, the German mathematician Volker Strassen announced a significantly more efficient algorithm, based upon divide-and-conquer.

# III. Bài toán áp dụng

**6. Nhân ma trận**

Matrix multiplication is particularly easy to break into subproblems, because it can be performed *blockwise*. To see what this means, carve $X$ into four $n/2 \times n/2$ blocks, and also $Y$:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

Then their product can be expressed in terms of these blocks and is exactly as if the blocks were single elements (Exercise 2.11).

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

We now have a divide-and-conquer strategy: to compute the size-$n$ product $XY$, recursively compute eight size-$n/2$ products $AE, BG, AF, BH, CE, DG, CF, DH$, and then do a few $O(n^2)$-time additions. The total running time is described by the recurrence relation

$$T(n) = 8T(n/2) + O(n^2).$$

# III. Bài toán áp dụng

**6. Nhân ma trận**

We now have a divide-and-conquer strategy: to compute the size-$n$ product $XY$, recursively compute eight size-$n/2$ products $AE, BG, AF, BH, CE, DG, CF, DH$, and then do a few $O(n^2)$-time additions. The total running time is described by the recurrence relation

$$T(n) = 8T(n/2) + O(n^2).$$

This comes out to an unimpressive $O(n^3)$, the same as for the default algorithm. But the efficiency *can* be further improved, and as with integer multiplication, the key is some clever algebra. It turns out $XY$ can be computed from just *seven* $n/2 \times n/2$ subproblems, via a decomposition so tricky and intricate that one wonders how Strassen was ever able to discover it!

# III. Bài toán áp dụng

**6. Nhân ma trận**

Strassen was ever able to discover it!

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

where

$$\begin{aligned} P_1 &= A(F - H) & P_5 &= (A + D)(E + H) \\ P_2 &= (A + B)H & P_6 &= (B - D)(G + H) \\ P_3 &= (C + D)E & P_7 &= (A - C)(E + F) \\ P_4 &= D(G - E) \end{aligned}$$

The new running time is

$$T(n) = 7T(n/2) + O(n^2),$$

which by the master theorem works out to $O(n^{\log_2 7}) \approx O(n^{2.81})$.

# III. Bài toán áp dụng

**7. 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 giá trị các phần tử.
  - Tìm mảng con có trọng lượng lớn nhất (1<= *p* <= *q* <= *n*)
  - 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

## III. Bài toán áp dụng

**7. Dãy con lớn nhất**

- Tiếp cận trực tiếp: 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
- Độ phức tạp: $O(n^3)$
- Tối ưu thuật toán: loại bỏ vòng lặp 3, độ phức tạp $O(n^2)$

```
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
      }
}
```

## III. Bài toán áp dụng

**7. Dãy con lớn nhất**

- **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

## III. Bài toán áp dụng

**7. Dãy con lớn nhất**

- **Cài đặt**

```
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 )
   }
}
```

# III. Bài toán áp dụng

**7. Dãy con lớn nhất**

- **Cài đặt**
  - Hàm *MaxLeftVector*

```
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 )
  }
}
```

```
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;
  }
```

# III. Bài toán áp dụng

**7. Dãy con lớn nhất**

- **Cài đặt**
  - Hàm *MaxLeftVector*
  - Hàm *MaxRightVector*

```
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 )
  }
}
```

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

# III. Bài toán áp dụng

**7. Dãy con lớn nhất**

- **Độ phức tạp: O(nlogn)**

```
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 )
  }
}
```

## III. Bài toán áp dụng

**8. Tính lũy thừa**

- Bài toán: Tính $a^n$ với a, n là các số nguyên và n không âm.
- Tiếp cận trực tiếp:
  - Thuật toán tính $a^n$ được thực hiện bằng phương pháp lặp như sau

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

## III. Bài toán áp dụng

**8. Tính lũy thừa**

- Bài toán: Tính $a^n$ với a, n là các số nguyên và n không âm.
- Tiếp cận trực tiếp:
  - 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;
}
```

  - Độ phức tạp: O(n)

## III. Bài toán áp dụng

**8. Tính lũy thừa**

- Tiếp cận chia để trị

$$a^n = \begin{cases} 1 & ,n = 0 \\ (a^2)^{\lfloor n/2 \rfloor} & ,n\%2 = 0 \\ a(a^2)^{\lfloor n/2 \rfloor} & ,n\%2 <> 0 \end{cases}$$

# III. Bài toán áp dụng

**8. Tính lũy thừa**

- Tiếp cận chia để trị

$$a^n = \begin{cases} 1 & ,n = 0 \\ (a^2)^{\lfloor n/2 \rfloor} & ,n\%2 = 0 \\ a(a^2)^{\lfloor n/2 \rfloor} & ,n\%2 <> 0 \end{cases}$$

- 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)  }
```

# III. Bài toán áp dụng

**8. Tính lũy thừa**

- Tiếp cận chia để trị
- Độ phức tạp: O(log n)

- 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)  }
```

# III. Bài toán áp dụng

**9. Hoán đổi phần tử của mảng**

- Bài toán:
  - Cho mảng gồm n phần tử A[1..n].
  - Hãy chuyển m phần tử đầu của mảng về cuối mảng.

a[8] = (1, 2, 3, 4, 5, 6, 7, 8)  n = 8

Nếu m = 3, thì kết quả là : ( 4, 5, 6, 7, 8, 1, 2, 3)
Nếu m = 5, thì kết quả là : ( 6, 7, 8, 1, 2, 3, 4, 5)
Nếu m = 4, thì kết quả là : ( 5, 6, 7, 8, 1, 2, 3, 4)

- Không dùng mảng phụ

## III. Bài toán áp dụng

**9. Hoán đổi phần tử của mảng**

- Ý tưởng:

Nếu m = n – m: Hoán đổi các phần tử của 2 nửa mảng có độ dài bằng nhau

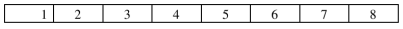
Nếu m < n – m : hoán đổi m phần tử đầu với m phân tử cuối của phần còn lại. Sau đó trong mảng a[1..n-m] ta chỉ cần hoán đổi m phần tử đầu với phần còn lại.

Nếu m > n – m : hoán đổi n-m phần tử đầu tiên với n-m phần tử của phần sau. Sau đó trong mảng a[n-m+1 . . n] ta hoán đổi n-m phần tử cuối mảng với các phần tử của phần đầu.

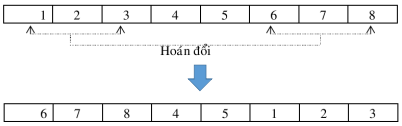
## III. Bài toán áp dụng

**9. Hoán đổi phần tử của mảng**

- Ví dụ: n=8, A=




- Hoán đổi với m=3, (n-m = 8-3 = 5),
  vì m = 3 < n-m = 5 -> Hoán đổi m(3) pt đầu với cuối



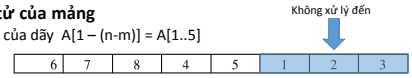

Được




## III. Bài toán áp dụng

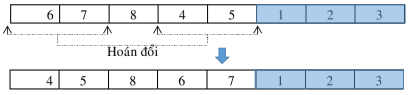
**9. Hoán đổi phần tử của mảng**

- Hoán đổi m (=3) của dãy A[1 – (n-m)] = A[1..5]



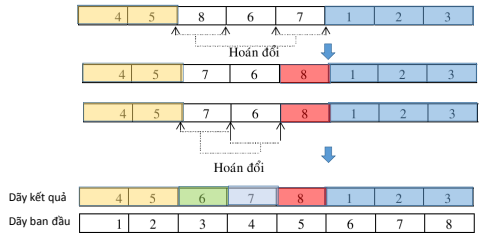

Bài toán trở thành: Hoán đổi m= 3 phần tử của dãy n=5 phần tử.

- Vì m = 3 > n-m = 2 nên



# III. Bài toán áp dụng

**9. Hoán đổi phần tử của mảng**




# III. Bài toán áp dụng

**9. Hoán đổi phần tử của mảng**

- Thuật toán

Input : a[1..n], m. (m ≤n)

Output : a[1..n] với tính chất m phần tử đầu mảng a ( mảng nhập ) nằm cuối

```
Exchange(a,i,j,m) ≡
        Với mọi p = 0 → m-1
                Đổichỗ (a[i+p],
```

```
Transpose(a,n,m) ≡
        i = m; j = n-m; m = m+1;
        Khi ( i ≠ j)
                Nếu ( i > j)
                {
                        Exchange(a,m-i,m,j);
                        i = i – j;
                }
                Ngược lại
                {
                        j = j – i;
                        Exchange(a,m-i,m+j,i);
                }
        Exchange(a,m-i,m,i);
```

# IV. Bài tập

Cho dãy A={-98,54,67, 65,-879,78,65,21,-6,67}

1. Hãy tìm dãy con có trọng số lớn nhất

Cho mảng A={3, 5, 8, 9, 4, 2, 7, 5, 3,9,8}

2. Hãy hoán đổi 3 phần tử của dãy về cuối
3. Hãy hoán đổi 4 phần tử của dãy về cuối
4. Hãy hoán đổi 5 phần tử của dãy về cuối
5. Sửa lại thuật toán tìm dãy con lớn nhất để cho phép lưu lại chỉ số đầu, cuối của dãy con lớn nhất.

## IV. Bài tập

2. Cài đặt thuật toán nhân 2 số nguyên có n (chẵn) chữ số. Đánh giá độ phức tạp bằng thực nghiệm và so sánh với lý thuyết.
3. Cài đặt thuật toán nhân ma trận theo chiến lược chia để trị của Strassen. Đánh giá độ phức tạp bằng thực nghiệm và so sánh với lý thuyết
4. Cài đặt thuật toán tìm dãy con lớn nhất. Đánh giá độ phức tạp bằng thực nghiệm và so sánh với lý thuyết.
5. Cài đặt thuật toán tính lũy thừa. Đánh giá độ phức tạp bằng thực nghiệm và so sánh với lý thuyết.
6. Cài đặt thuật toán hoán đổi vị trí phần tử mảng. Đánh giá độ phức tạp bằng thực nghiệm và so sánh với lý thuyết.

## NỘI DUNG BÀI HỌC

I. Giới thiệu
II. Lược đồ chung
III. Bài toán áp dụng
IV. Bài tập