

Quy hoạch động (6)



Khoa Khoa học máy tính

Quy hoạch động (dynamic programming)

- Nguyên tắc tương tự thuật toán chia để trị
 - Bài toán được chia thành nhiều bài toán con
 - Bài toán tiếp tục được chia thành các bài toán con khác, cho đến khi các bài toán con có thể giải quyết được dễ dàng
 - Kết hợp giải pháp của các bài toán con có được giải pháp của bài toán ban đầu

Quy hoạch động

- Sự khác nhau với thuật toán chia để trị
 - Quy hoạch động được áp dụng khi các bài toán con không độc lập
 - Các bài toán con chung
 - Khi các bài toán con không độc lập
 - Áp dụng thuật toán chia để trị
 - Thực hiện cùng công việc (giải quyết cùng một bài toán con) nhiều lần
 - Áp dụng thuật toán quy hoạch động
 - Mỗi bài toán con được giải quyết một lần và ghi kết quả vào một mảng, sau đó nếu gặp lại bài toán con đó chỉ lấy kết quả sử dụng
 - Giảm độ phức tạp

Quy hoạch động

- Quy hoạch động = Chia để trị + mảng
- Chia để trị : tiếp cận từ trên xuống
 - Giải quyết bài toán lớn trước sau đó giải quyết bài toán con sau
- Quy hoạch động : tiếp cận từ dưới lên
 - Giải quyết bài toán con trước sau đó dựa trên các bài toán con đã giải quyết, giải quyết bài toán lớn sau

Quy hoạch động

- Thuật toán quy hoạch động thường được áp dụng cho các **bài toán tối ưu**
 - Các bài toán này có thể có nhiều giải pháp, chúng ta muốn tìm giải pháp tối ưu theo một hàm mục tiêu
- Xây dựng thuật toán quy hoạch động thường trải qua các bước
 - Xác định các tính chất của cấu trúc của giải pháp tối ưu
 - Định nghĩa đệ quy giá trị của giải pháp tối ưu
 - Tính giá trị của giải pháp tối ưu thông qua các trường hợp đơn giản (trường hợp dừng của đệ quy) và lần lên cho đến khi giải quyết được bài toán ban đầu
 - Xây dựng giải pháp tối ưu đối với các thông tin vừa tính toán
 - Nếu cần cả giải pháp tối ưu chứ không chỉ là giá trị của giải pháp tối ưu

Một số ứng dụng

- Triển khai nhị thức $(a+b)^n$
- Nhân dãy ma trận
- Dãy con chung dài nhất
- Xếp ba lô

Triển khai nhị thức $(a+b)^n$

- Nhị thức $(a+b)^n$ được triển khai theo công thức sau

$$(a+b)^n = \sum_{k=0}^n C_n^k a^{n-k} b^k$$

- Với

$$C_n^k = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} = \frac{n!}{k!(n-k)!}$$

- Công thức cho phép tính tổ hợp chập k của n

$$C_n^k = \begin{cases} 1 & k=0, k=n \\ C_{n-1}^{k-1} + C_{n-1}^k & 1 \leq k \leq n-1 \end{cases}$$

Triển khai nhị thức $(a+b)^n$

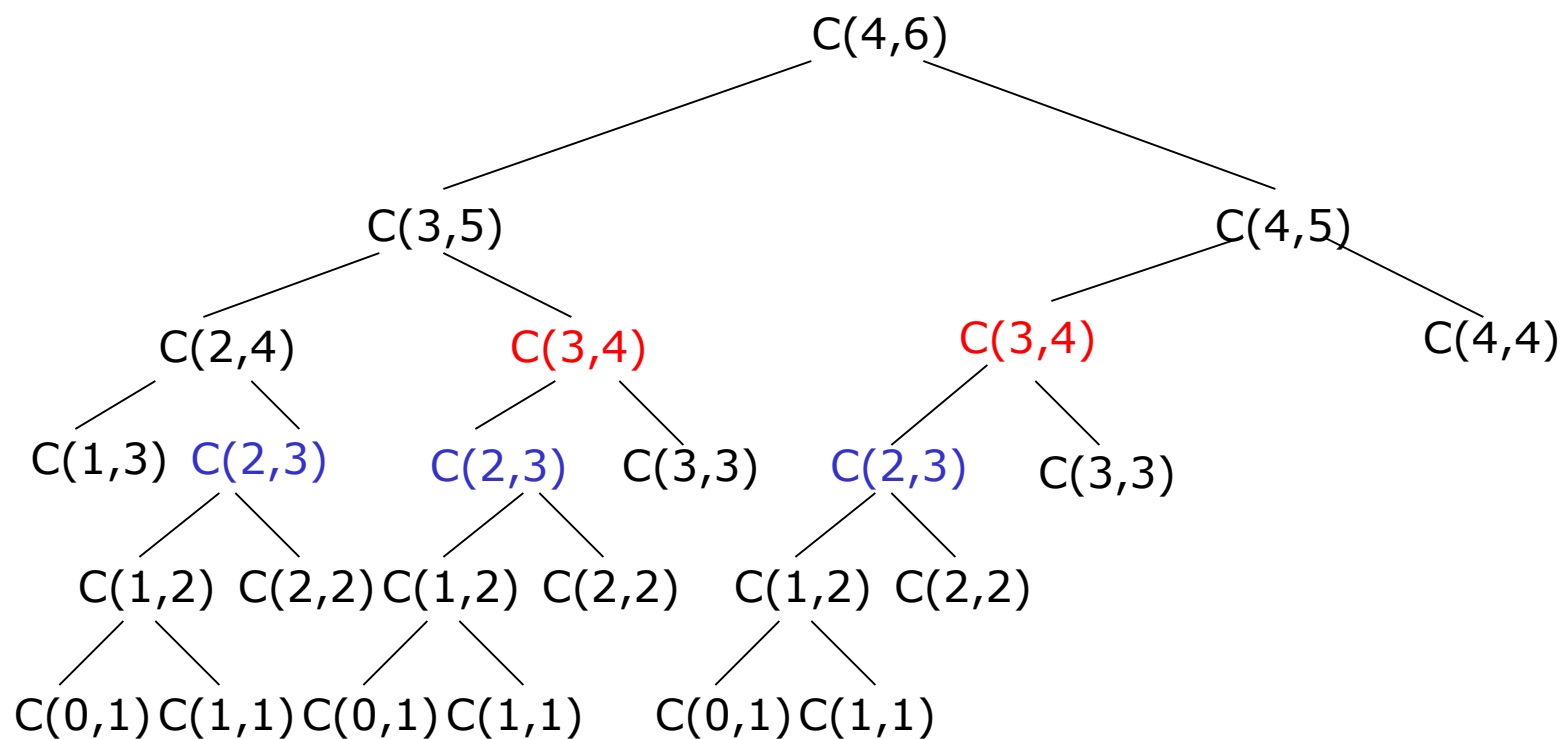
□ Thuật toán chia để trị tính C_n^k (1)

```
C(k, n)
begin
  if (k = 0 or k = n) then
    return (1)
  else
    return (C(k-1, n-1) + C(k, n-1))
end
```

- Có nhiều giá trị $C(i, j)$ với $i < k$ và $j < n$, được tính lặp nhiều lần
- Độ phức tạp lớn

Triển khai nhị thức $(a+b)^n$

□ Thuật toán chia để trị tính C_n^k (2)



Triển khai nhị thức $(a+b)^n$

□ Thuật toán chia để trị tính C_n^k (3)

■ Tính độ phức tạp

- Gọi $C(n)$ là thời gian thực thi trong trường hợp xấu nhất tính $C(k, n)$ với mọi k
- Vậy từ thuật toán, ta có

$$C(n) = \begin{cases} c & \text{if } n = 1 \\ 2C(n-1) + d & \text{else} \end{cases}$$

với c và d là các hằng số

Triển khai nhị thức $(a+b)^n$

□ Thuật toán chia để trị tính C_n^k (4)

■ Tính độ phức tạp

$$C(n) = 2C(n-1) + d$$

$$= 2(C(n-2) + d) + d = 4C(n-2) + 2d + d$$

$$= 4(2C(n-3) + d) + 2d + d = 8C(n-3) + 4d + 2d + d$$

$$= 2^i C(n-i) + d \sum_{k=0}^{i-1} 2^k$$

$$= 2^{n-1} C(1) + d \sum_{k=0}^{n-2} 2^k = 2^{n-1} c + d \frac{2^{n-1} - 1}{2 - 1}$$

$$= 2^{n-1} (c + d) - d$$

■ Vậy: $C(n) = \Theta(2^n)$

Triển khai nhị thức $(a+b)^n$

- Thuật toán quy hoạch động tính C_n^k (1)
 - Sử dụng mảng $C[0..n, 0..k]$ lưu các kết quả trung gian
 - $C[i, j]$ chứa giá trị C_i^j
 - $C[i, j]$ được tính

$$C[i, j] = \begin{cases} 1 & j = 0 \text{ hay } j = i \\ C[i-1, j-1] + C[i-1, j] & \text{trường hợp còn lại} \end{cases}$$

- Tính các giá trị của tam giác Pascal

n/k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (2)

```
C1(k, n)
begin
    // sử dụng mảng hai chiều C[0..n,0..k]
    for i from 0 to n-k do C[i,0] = 1 endfor
    for i from 0 to k do C[i,i] = 1 endfor
    // tính từng cột
    for j from 1 to k do
        for i from j+1 to n-k+j do
            C[i,j] = C[i-1,j-1] + C[i-1,j]
        endfor
    endfor
    return (C[n,k])
end
```

Triển khai nhị thức $(a+b)^n$

□ Thuật toán quy hoạch động tính C_n^k (3)

■ Ví dụ: $n=8, k=5$

n/k	0	1	2	3	4	5
0	1					
1	1	1				
2	1		1			
3	1			1		
4					1	
5						1
6						
7						
8						



n/k	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4		4	6	4	1	
5			10	10	5	1
6				20	15	6
7					35	21
8						56

Chỉ cần tính các giá trị cần cho việc tính $C[8,5]$

Triển khai nhị thức $(a+b)^n$

□ Thuật toán quy hoạch động tính C_n^k (4)

■ Độ phức tạp

- Số các giá trị cần tính (giả sử phép cộng là phép toán cơ bản)

$$k(n-k) = nk - k^2 \leq nk$$

- Vậy độ phức tạp về thời gian $O(nk)$
- Sử dụng mảng có nk ô nhớ để lưu trữ các giá trị trung gian, hay độ phức tạp về mặt không gian $O(nk)$

- Có thể cải tiến thuật toán để giảm số ô nhớ sử dụng ?

Triển khai nhị thức $(a+b)^n$

▣ Thuật toán quy hoạch động tính C_n^k (5)

Chỉ sử dụng mảng
một chiều lưu trữ
dòng hiện thời của
tam giác Pascal

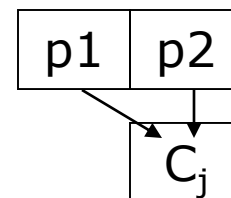
```
C2(k, n)
begin
    // sử dụng mảng một chiều C[0..n]
    C[0] = 1 // khởi gán hàng 1
    C[1] = 1
    // tính từng hàng
    for i from 2 to n do
        p1 = 1
        for j from 1 to i-1 do
            p2 = C[j]
            C[j] = p1 + p2
            p1 = p2
        endfor
        C[i] = 1 // phần tử trên đường chéo
    endfor
    return (C[k])
end
```


Triển khai nhị thức $(a+b)^n$

□ Thuật toán quy hoạch động tính C_n^k (6)

■ Ví dụ: $n=8, k=5$

n/k	0	1	2	3	4	5	6	7	8
0									
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1



Triển khai nhị thức $(a+b)^n$

□ Thuật toán quy hoạch động tính C_n^k (7)

■ Độ phức tạp

- Số các giá trị cần tính (giả sử phép cộng là phép toán cơ bản)

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

- Vậy độ phức tạp về thời gian $O(n^2)$
- Sử dụng mảng có n ô nhớ để lưu trữ các giá trị trung gian, hay độ phức tạp về mặt không gian $O(n)$

Thiết kế thuật toán quy hoạch động

□ Nhận dạng

- Xây dựng thuật toán chia để trị/thuật toán đơn giản
- Đánh giá độ phức tạp (hàm mũ)
- Cùng một bài toán con được giải quyết nhiều lần

□ Xây dựng

- Tách phần « trị » trong thuật toán chia để trị và thay thế các lời gọi đệ quy bằng việc tìm kiếm các giá trong một mảng
- Thay vì trả về giá trị, ghi giá trị vào mảng
- Sử dụng điều kiện dừng của thuật toán chia để trị để khởi tạo giá trị của mảng
- Tìm cách tính các giá trị của mảng
- Xây dựng vòng lặp để tính các giá trị của mảng

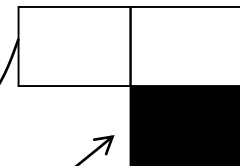
Thiết kế thuật toán quy hoạch động

```
C(k, n)
begin
  if (k = 0 or k = n) then return (1)
  else return C(k-1, n-1) + C(k, n-1)
end
```

Chia để trị

```
C1(k, n)
begin
  for i from 0 to n-k do C[i,0] = 1 endfor
  for i from 0 to k do C[i,i] = 1 endfor
  for j from 1 to k do
    for i from j+1 to n-k+j do
      C[i,j] = C[i-1,j-1] + C[i-1,j]
    endfor
  endfor
  return (C[n,k])
end
```

Quy hoạch động



Nhân dãy ma trận

□ Bài toán

- Có n ma trận M_1, M_2, \dots, M_n , cần tính tích $M_1.M_2.\dots.M_n$ sao cho thực hiện ít phép nhân nhất
- Phép nhân ma trận có tính kết hợp
 - Có thể thực hiện tích $M_1.M_2.\dots.M_n$ bởi nhiều thứ tự kết hợp khác nhau
- Giả sử dụng thuật toán đơn giản để nhân hai ma trận
 - Nhân hai ma trận có kích thước $p \times q$ và $q \times r$ cần thực hiện $p \times q \times r$ phép nhân

Nhân dãy ma trận

□ Ví dụ

■ Nhân dãy ma trận

$$M_1(10 \times 20) \cdot M_2(20 \times 50) \cdot M_3(50 \times 1) \cdot M_4(1 \times 100)$$

■ Có các 5 cách kết hợp

□ $(M_1 \cdot (M_2 \cdot (M_3 \cdot M_4)))$:
 $50 \times 1 \times 100 + 20 \times 50 \times 100 + 10 \times 20 \times 100 = 125000$ phép nhân

□ $(M_1 \cdot ((M_2 \cdot M_3) \cdot M_4))$: 72000

□ $((M_1 \cdot M_2) \cdot (M_3 \cdot M_4))$: 65000

□ $((M_1 \cdot (M_2 \cdot M_3)) \cdot M_4)$: 2200

□ $((((M_1 \cdot M_2) \cdot M_3) \cdot M_4)$: 60500

■ Vấn đề: cách kết hợp nào thực hiện ít phép nhân nhất?

Nhân dãy ma trận

□ Thuật toán vét cạn (1)

- Thử tất cả các cách kết hợp có thể
- Tính số phép nhân cho mỗi cách
- Chọn cách tốt nhất

- Số tất cả các cách kết hợp là **hàm mũ**

- Gọi $P(n)$ là số cách kết hợp n ma trận

- Khi $n = 1$ thì $P(1) = 1$

- Khi $n \geq 2$ thì có thể chia tích $M_1.M_2.....M_n$ thành hai

$$M_1.M_2.....M_n = \underbrace{(M_1.M_2...M_k)}_{P(k) \text{ cách}} . \underbrace{(M_{k+1}...M_n)}_{P(n-k) \text{ cách}}$$

có $n-1$ cách chia ($k=1..n-1$)

Nhân dãy ma trận

□ Thuật toán vét cạn (2)

■ Vậy

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n \geq 2 \end{cases}$$

■ Có thể chỉ ra được

$$P(n) = \Omega(2^n)$$



Nhân dãy ma trận

□ Thuật toán chia để trị (1)

- Gọi $m_{i,j}$ là **số tối thiểu** các phép nhân khi thực hiện tích $M_i M_{i+1} \dots M_j$
 - Rõ ràng $m_{i,i} = 0$
- Giả sử kích thước các ma trận M_i, M_{i+1}, \dots, M_j lần lượt là $(d_{i-1}, d_i), (d_i, d_{i+1}), \dots, (d_{j-1}, d_j)$
- Giả sử rằng, chúng ta biết được cách kết hợp $M_i M_{i+1} \dots M_j$ với chi phí tối thiểu ($m_{i,j}$ nhỏ nhất) là $(M_i M_{i+1} \dots M_k) \cdot (M_{k+1} \dots M_j)$
 - nghĩa là k đã biết
 - khi đó:
$$m_{i,j} = m_{i,k} + m_{k+1,j} + \text{chi phí nhân hai ma trận } (d_{i-1}, d_k) \cdot (d_k, d_j)$$
$$m_{i,j} = m_{i,k} + m_{k+1,j} + d_{i-1} \cdot d_k \cdot d_j$$
- Tuy nhiên, k chưa xác định, $k \in [1..n-1]$, vậy:

$$m_{i,j} = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1} d_k d_j) & i < j \end{cases}$$

Nhân dãy ma trận

□ Thuật toán chia để trị (2)

- Từ đó xây dựng thuật toán chia để trị cho phép tính $m_{i,j}$ với tất cả các giá trị có thể của k

```
matrixChainRecur(i, j)
// sử dụng mảng m[i,j] lưu trữ giá trị  $m_{i,j}$ 
// ban đầu, m[i,j] được khởi gán  $+\infty$ 
begin
    if (i=j) then return (0)
    else for k from i to j-1 do
        r = matrixChainRecur(i,k) + matrixChainRecur(k+1,j) +  $d_{i-1}d_kd_j$ 
        if (r < m[i, j]) then
            r = m[i,j]
        endif
    endfor
    return (m[i,j])
end
```

Nghiệm của bài toán là $m[1,n]$

Nhân dãy ma trận

- ❑ Thuật toán chia để trị (3)

- Đánh giá độ phức tạp

- ❑ Gọi $C(n)$ là thời gian thực hiện chainMatrixRecur(i,j) khi $i=1, j=n$

- ❑ Ta có
$$C(n) = \begin{cases} c & n = 0 \\ \sum_{k=1}^{n-1} (C(k) + C(n-k) + d) & n > 0 \end{cases}$$

với c, d là các hằng số

- ❑ Hay

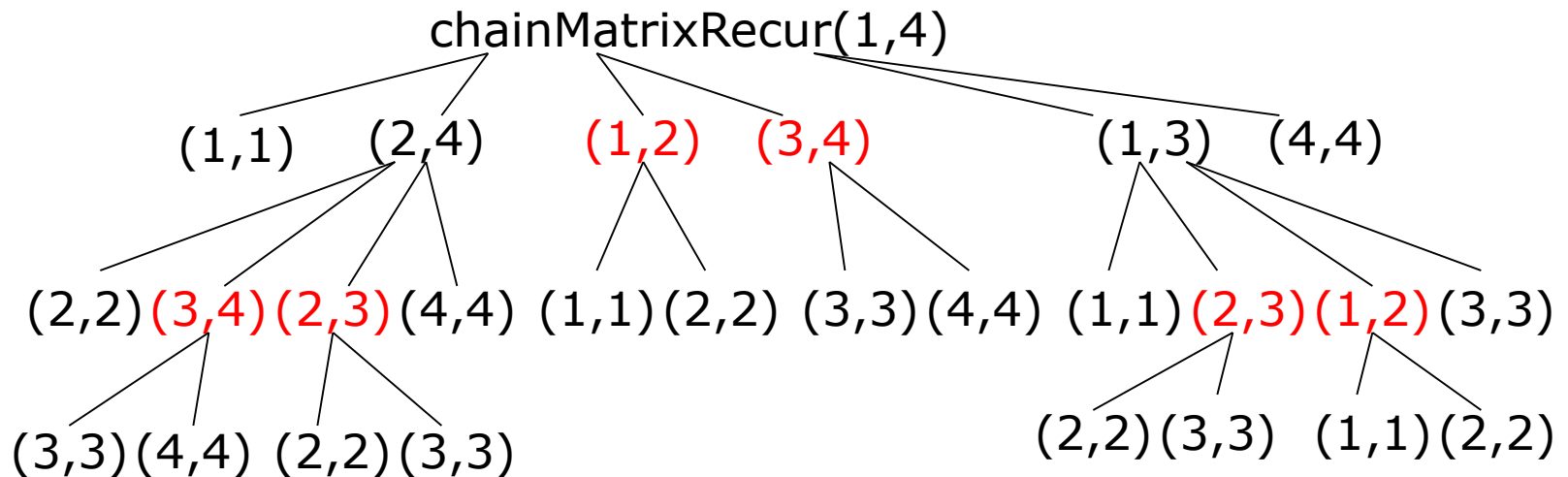
$$C(n) = 2 \sum_{k=1}^{n-1} C(k) + d(n-1)$$

- ❑ Vậy $C(n) \geq 2C(n-1)$, nghĩa là $C(n) = \Omega(2^n)$

- ❑ Nếu tính chính xác, thì $C(n) = \Theta(3^n)$

Nhân dãy ma trận

- Thuật toán chia để trị (4)
 - Ví dụ



Có nhiều bài toán con được thực hiện nhiều lần !

Nhân dãy ma trận

□ Thuật toán quy hoạch động (1)

- Chúng ta cần xây dựng mảng $m[1..n, 1..n]$
- Với mỗi $m[i, j]$, $i \leq j$, nên chỉ nửa trên đường chéo chính của mảng m được sử dụng
- Mỗi $m[i, j]$ được tính theo công thức

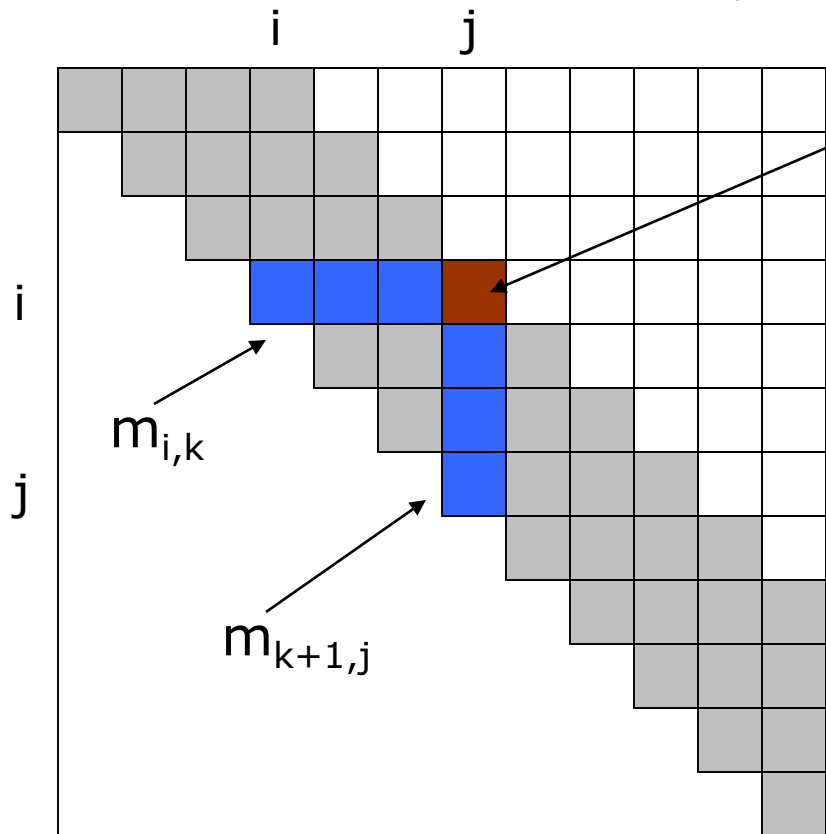
$$m_{i,j} = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1}d_kd_j) & i < j \end{cases}$$

- Vậy, muốn tính $m_{i,j}$ chúng ta phải có giá trị $m_{i,k}$ và $m_{k+1,j}$ với $i \leq k < j$

Nhân dãy ma trận

□ Thuật toán quy hoạch động (2)

■ Minh họa cách tính $m_{i,j}$



$$m_{i,j} = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,j} + d_{i-1}d_kd_j) & i < j \end{cases}$$

➡ Cần phải xây dựng dần bảng m theo các đường chéo

Nhân dãy ma trận

□ Thuật toán quy hoạch động (3)

- Mảng m sẽ được xây dựng dần theo từng đường chéo
- Đường chéo s sẽ gồm các phần tử $m[i,j]$ mà $j-i=s$
- Vậy

$$m_{i,j} = \begin{cases} 0 & s = 0 \\ \min_{i \leq k < j} (m_{i,k} + m_{k+1,i+s} + d_{i-1}d_kd_{i+s}) & 0 < s < n \end{cases}$$

- Xây dựng mảng m mới chỉ cho phép tính số phép nhân tối thiểu (giá trị) của giải pháp
- Cần phải xây dựng giải pháp (cách thực hiện các phép nhân ma trận)
 - Sử dụng mảng phụ $g[1..n,1..n]$ để ghi nhớ chỉ số k cho giải pháp tối ưu mỗi bước

Nhân dãy ma trận

□ Thuật toán quy hoạch động (4)

```
matrixChain(n)
begin
  for i from 0 to n do m[i,i] = 0 endfor
  for s from 1 to n-1 do // tính tất cả các đường chéo s
    for i from 1 to n-s do // tính một đường chéo s
      j = i + s
      // tính  $m_{i,j} = \min(m_{i,k} + m_{k+1,i+s} + d_{i-1}d_kd_{i+s})$ 
      m[i,j] =  $+\infty$ 
      for k from i to j-1 do
        r = m[i,k] + m[k+1,j] +  $d_{i-1}d_kd_j$ 
        if (r < m[i,j]) then
          r = m[i,j]
          g[i,j] = k // ghi nhớ k
        endif
      endfor
    endfor
  endfor
  return m và s
end
```


Nhân dãy ma trận

□ Thuật toán quy hoạch động (5)

■ Ví dụ

- $M_1(10 \times 20) \cdot M_2(20 \times 50) \cdot M_3(50 \times 1) \cdot M_4(1 \times 100)$
nghĩa là: $d_0=10, d_1=20, d_2=50, d_3=1, d_4=100$

□ Xây dựng đường chéo $s = 1$

- $m[1,2] = \min(m[1,k] + m[k+1,2] + d_0d_1d_2), \text{ với } 1 \leq k < 2$
 $= \min(m[1,1] + m[2,2] + d_0d_1d_2)$
 $= d_0d_1d_2 = 10 \times 20 \times 50 = 10000$

$$g[1,2] = k = 1$$

- $m[2,3] = \min(m[2,k] + m[k+1,3] + d_1d_2d_3), \text{ với } 2 \leq k < 3$
 $= d_1d_2d_3 = 20 \times 50 \times 1 = 1000$

$$g[2,3] = k = 2$$

- $m[3,4] = d_2d_3d_4 = 50 \times 1 \times 100 = 5000$

$$g[3,4] = k = 3$$

Nhân dãy ma trận

□ Thuật toán quy hoạch động (6)

■ Ví dụ

□ Xây dựng đường chéo $s = 2$

$$\begin{aligned} \blacksquare m[1,3] &= \min(m[1,k] + m[k+1,3] + d_0 d_k d_3), \text{ với } 1 \leq k < 3 \\ &= \min(m[1,1] + m[2,3] + d_0 d_1 d_3, \\ &\quad m[1,2] + m[3,3] + d_0 d_2 d_3) \\ &= \min(0 + 1000 + 200, 10000 + 0 + 500) \\ &= 1200 \end{aligned}$$

$$g[1,3] = k = 1$$

$$\begin{aligned} \blacksquare m[2,4] &= \min(m[2,k] + m[k+1,4] + d_1 d_k d_4), \text{ với } 2 \leq k < 4 \\ &= \min(m[2,2] + m[3,4] + d_1 d_2 d_4, \\ &\quad m[2,3] + m[4,4] + d_1 d_3 d_4) \\ &= \min(0 + 5000 + 100000, 1000 + 0 + 2000) \\ &= 3000 \end{aligned}$$

$$g[2,4] = k = 3$$

Nhân dãy ma trận

□ Thuật toán quy hoạch động (7)

■ Ví dụ

□ Xây dựng đường chéo $s = 3$

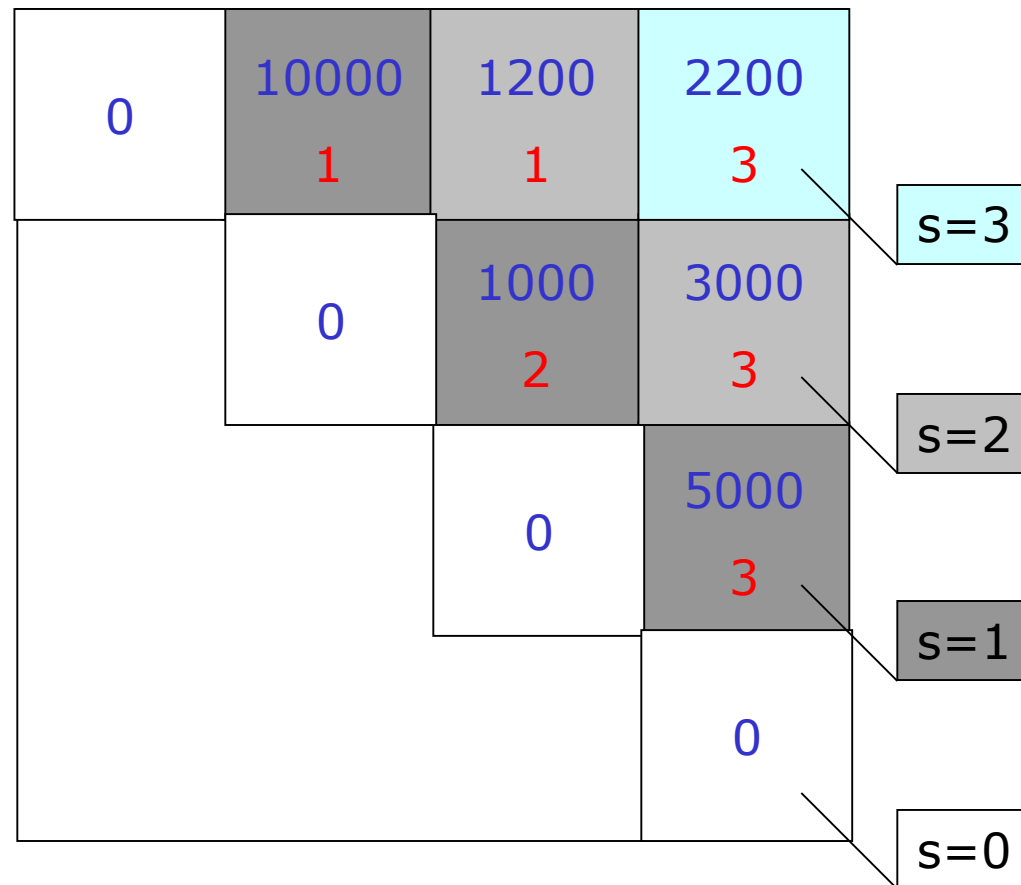
$$\begin{aligned} \blacksquare m[1,4] &= \min(m[1,k] + m[k+1,4] + d_0 d_k d_4), \text{ với } 1 \leq k < 4 \\ &= \min(m[1,1] + m[2,4] + d_0 d_1 d_4, \\ &\quad m[1,2] + m[3,4] + d_0 d_2 d_4, \\ &\quad m[1,3] + m[4,4] + d_0 d_3 d_4) \\ &= \min(0 + 3000 + 20000, \\ &\quad 10000 + 5000 + 50000, \\ &\quad 1200 + 0 + 1000) \\ &= 2200 \\ g[1,4] &= k = 3 \end{aligned}$$

□ $m[1,4]$ chính là kết quả cần tìm

Nhân dãy ma trận

▣ Thuật toán quy hoạch động (8)

- Ví dụ: các giá trị mảng **m** và **g**



Nhân dãy ma trận

- Thuật toán quy hoạch động (9)
 - Đánh giá độ phức tạp
 - Thuật toán gồm 3 vòng lặp lồng nhau
 - Mỗi vòng lặp đều không lặp quá n lần
 - Vậy độ phức tạp $O(n^3)$
 - Thuật toán quy hoạch động tốt hơn so với thuật toán đơn giản và thuật toán chia để trị

Nhân dãy ma trận

□ Thuật toán quy hoạch động (10)

■ Xây dựng giải pháp tối ưu

- thực hiện các phép nhân theo thứ tự tối ưu
- *chainmatrix* chỉ tính giá trị tối ưu của cách kết hợp các phép nhân, chứ không thực hiện phép nhân
- Sử dụng thông tin chứa trong mảng g
 - $g[i,j]$ chứa giá trị k , mà tích $M_i M_{i-1} \dots M_j$ được tách đôi giữa M_k và M_{k+1}
- Giả sử đã có thuật toán nhân hai ma trận X và Y :
`matrixProduct(X, Y)`

Nhân dãy ma trận

- ❑ Thuật toán quy hoạch động (11)
 - Xây dựng giải pháp tối ưu

```
chainMatrixProduct(M, g, i, j)
// dãy các ma trận
begin
  if (i < j) then
    X = chainMatrixProduct(M, g, i, s[i,j])
    Y = chainMatrixProduct(M, g, s[i,j]+1,j)
    return (matrixProduct(X,Y))
  else // i = j
    return (Mi)
  endif
end
```

- ❑ Khi gọi chainMatrixProduct(M, g, 1, 4) sẽ tính tích $M_1.M_2.M_3.M_4$ theo thứ tự: $((M_1.(M_2.M_3)).M_4)$
- ❑ Vì: $g[1,4] = 3, g[1,3] = 1$

Nhân dãy ma trận

□ Thuật toán quy hoạch động (12)

■ Bài tập

- Viết thuật toán in ra biểu thức kết hợp tối ưu thực hiện nhân dãy ma trận

- Ví dụ

Cho dãy ma trận: $M_1(10 \times 20) \cdot M_2(20 \times 50) \cdot M_3(50 \times 1) \cdot M_4(1 \times 100)$

Kết quả nhận được: $((M_1 \cdot (M_2 \cdot M_3)) \cdot M_4)$

Dãy con chung dài nhất

□ Bài toán

- Cho hai dãy kí hiệu X và Y, dãy con chung dài nhất (Longest Common Subsequence - LCS) của X và Y là dãy các kí hiệu nhận được từ X bằng cách xoá đi một số các phần tử và cũng nhận được từ Y bằng cách xoá đi một số phần tử

- Ví dụ: X = ABCBDAB và Y = BDCABA

X = A **B** **C** **B** D **A** B

Y = **B** D **C** A **B** **A**

Dãy con chung dài nhất: **BCBA**

- Ứng dụng so sánh « độ tương tự » hai chuỗi ADN

Dãy con chung dài nhất

□ Thuật toán vét cạn

- Giả sử $X = x_1x_2...x_n$ và $Y = y_1y_2...y_m$
- So sánh mỗi dãy kí hiệu con của của X với dãy kí hiệu Y
 - Có 2^n dãy con của X
 - Mỗi dãy con của X so sánh với Y sẽ thực hiện m phép so sánh
- Độ phức tạp sẽ là $O(m2^n)$
 - Hàm mũ !

Dãy con chung dài nhất

□ Thuật toán chia để trị (1)

- Có thể phân tích bài toán thành những bài toán con với kích thước nhỏ hơn
 - Bài toán con: tìm dãy con chung dài nhất của các cặp *tiền tố* của X và Y
 - Cho $X = x_1x_2\dots x_n$, $X_i = x_1x_2\dots x_i$ được gọi là tiền tố thứ i của X
- Gọi $Z = z_1z_2\dots z_k$ là dãy con chung dài nhất (LCS) của $X = x_1x_2\dots x_n$ và $Y = y_1y_2\dots y_m$
 - Nếu $x_n = y_m$ thì $z_k = x_n = y_m$ và $Z_{k-1} = \text{LCS}(X_{n-1}, Y_{m-1})$
 - Nếu $x_n \neq y_m$ và $z_k \neq x_n$ thì $Z = \text{LCS}(X_{n-1}, Y)$
 - Nếu $x_n \neq y_m$ và $z_k \neq y_m$ thì $Z = \text{LCS}(X, Y_{m-1})$

Dãy con chung dài nhất

□ Thuật toán chia để trị (2)

■ Giải pháp đệ quy

- Gọi $c[i,j]$ là độ dài dãy con chung dài nhất của X_i và Y_j
- Khi đó, độ dài dãy con chung dài nhất của X và Y sẽ là $c[n,m]$
- Độ dài dãy con chung dài nhất của một dãy rỗng và một dãy bất kỳ luôn bằng 0
 - $c[i,0] = 0$ và $c[0,j] = 0$ với mọi i, j
- Vậy, ta có

$$c[i, j] = \begin{cases} 0 & i = 0 \text{ hay } j = 0 \\ c[i-1, j-1] + 1 & i, j > 0, x_i = y_j \\ \max(c[i-1, j], c[i, j-1]) & i, j > 0, x_i \neq y_j \end{cases}$$

Dãy con chung dài nhất

▣ Thuật toán chia để trị (3)

```
LCS-length (i,j)
begin
  if (i = 0 or j = 0) then
    return (0)
  else
    if (xi = yj) then
      return (LCS-length(i-1,j-1) + 1)
    else
      return (max(LCS-length(i-1, j), LCS-length(i, j-1)))
    endif
  endif
end
```

Dãy con chung dài nhất

- Thuật toán chia để trị (4)
 - Đánh giá độ phức tạp
 - Giả sử $n \geq m$
 - Gọi $C(k)$ là thời gian thực thi trong **trường hợp xấu nhất**
 - Vậy: $C(k) \geq 2C(m-1) = 2^2C(m-2) = 2^mC(0) = 2^m$
 - Nghĩa là thuật toán có độ phức tạp $\Omega(2^m)$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (1)

- Lưu giá trị độ dài của các dãy con chung dài nhất các cặp tiền tố vào mảng $c[0..n, 0..m]$

```
LCS-length (X,Y)
begin
  n = length(X)
  m = length(Y)
  for i from 0 to n do c[i,0]=0 endfor
  for j from 0 to m do c[0,j]=0 endfor
  for i from 1 to n do
    for j from 1 to m do
      if ( $x_i = y_j$ ) then c[i,j] = c[i-1,j-1] + 1
      else
        c[i,j] = max(c[i-1,j], c[i,j-1])
      endif
    endfor
  endfor
  return c
end
```

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (2)

■ Ví dụ

		j	0	1	2	3	4	5
i		y_j	B	D	C	A	B	
0	x_i							
1	A							
2	B							
3	C							
4	B							

$X=ABCB, Y=BDCAB$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (3)

■ Ví dụ

		j	0	1	2	3	4	5
i		y_j	B	D	C	A	B	
0	x_i	0	0	0	0	0	0	
1	A	0						
2	B	0						
3	C	0						
4	B	0						

for i from 0 to n do $c[i,0]=0$
for j from 0 to m do $c[0,j]=0$

Dãy con chung dài nhất

Thuật toán quy hoạch động (4)

■ Ví dụ

		j	0	1	2	3	4	5
			y_j	B	D	C	A	B
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0				
2	B	0						
3	C	0						
4	B	0						

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

▣ Thuật toán quy hoạch động (5)

■ Ví dụ

		j	0	1	2	3	4	5
i		y_j	B	D	C	A	B	
0	x_i	0	0	0	0	0	0	
1	A	0	0	0	0			
2	B	0						
3	C	0						
4	B	0						

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (6)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y _j	B	D	C	A	B
0	x _i		0	0	0	0	0	0
1	A		0	0	0	0	1	
2	B		0					
3	C		0					
4	B		0					

if (x_i = y_j) then c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (7)

■ Ví dụ

		j	0	1	2	3	4	5
		y_j		B	D	C	A	B
i	x_i							
0			0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0					
3	C		0					
4	B		0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

▣ Thuật toán quy hoạch động (7)

■ Ví dụ

		j	0	1	2	3	4	5
		y_j		B	D	C	A	B
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0	0	1	1
2	B	0	1					
3	C	0						
4	B	0						

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

Thuật toán quy hoạch động (8)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y_j	B	D	C	A	B
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	
3	C		0					
4	B		0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (9)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y_j	B	D	C	A	B
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0					
4	B		0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (10)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y _j	B	D	C	A	B
0	x _i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1			
4	B		0					

if (x_i = y_j) then c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (11)

■ Ví dụ

		j	0	1	2	3	4	5
		y_j		B	D	C	A	B
i	x_i							
0			0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2		
4	B		0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (12)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y_j	B	D	C	A	B
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0					

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (13)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y_j	B	D	C	A	B
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1				

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (14)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y_j	B	D	C	A	B
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	

Arrows indicating the path for the longest common subsequence:

 From (4,4) to (3,4) to (2,4) to (2,3) to (1,3) to (1,2) to (0,2).

 Red numbers 1, 2, 2 are shown in the bottom row (i=4) under columns j=5, j=4, and j=3 respectively.

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
 else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (15)

■ Ví dụ

		j	0	1	2	3	4	5
		i	y_j	B	D	C	A	B
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

if ($x_i = y_j$) then $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

Dãy con chung dài nhất

- Thuật toán quy hoạch động (16)
 - Đánh giá độ phức tạp
 - Thuật toán nhằm tính mảng c có $n \times m$ phần tử
 - Tính mỗi phần tử cần $O(1)$ thời gian
 - Vậy độ phức tạp của thuật toán là $O(nm)$

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (17)

■ Xây dựng giải pháp tối ưu

- Thuật toán *LCS-length* chỉ tính độ dài của dãy con chung dài nhất
- Cần xác định dãy con chung dài nhất đó
- Tương tự như thuật toán nhân dãy ma trận, sử dụng mảng phụ $b[1..n, 1..m]$ ghi nhớ các phần tử của mảng c ứng với giải pháp tối ưu
 - Khi $c[i,j] = c[i-1,j-1] + 1$ thì ghi nhớ x_i , vì x_i thuộc dãy con chung dài nhất
- Xuất phát từ $c[m,n]$ lần ngược lên
 - Nếu $c[i,j] = c[i-1,j-1] + 1$ thì x_i thuộc dãy con chung dài nhất
 - Nếu $i = 0$ hoặc $j = 0$ thì dừng lại
- Đảo ngược dãy kí hiệu nhận được dãy con chung dài nhất

Dãy con chung dài nhất

Thuật toán quy hoạch động (18)

■ Ví dụ

		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
i	Xi							
0			0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1	1	1	1	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

Dãy con chung dài nhất (thứ tự ngược)

Dãy con chung dài nhất

B **C** **B**
B **C** **B**

Dãy con chung dài nhất

- Thuật toán quy hoạch động (20)
 - Chỉnh lại thuật toán *LCS-length*, ghi nhớ giải pháp tối ưu vào mảng *b*

```
LCS-length (X,Y)
begin
  n = length(X), m = length(Y)
  for i from 0 to n do c[i,0]=0 endfor
  for j from 0 to m do c[0,j]=0 endfor
  for i from 1 to n do
    for j from 1 to m do
      if ( $x_i = y_j$ ) then c[i,j] = c[i-1,j-1] + 1
                        b[i,j] = '\'
      else
        if (c[i-1,j] < c[i,j-1]) then c[i,j] = c[i,j-1]
                                      b[i,j] = '←'
        else c[i,j] = c[i-1,j]
            b[i,j] = '↑'
        endif endif endfor endfor
  return c
end
```

Dãy con chung dài nhất

□ Thuật toán quy hoạch động (19)

■ Xây dựng giải pháp tối ưu






```
print-LCS(b, X, i, j)
begin
  if (i = 0 or j = 0) then
    return
  else
    if (b[i, j] = '\') then // xi thuộc dãy con chung dài nhất
      print-LCS(b,X,i-1,j-1)
      print xi
    else
      if (b[i,j]='←') then
        print-LCS(b,X,i,j-1)
      else // (b[i,j]='↑')
        print-LCS(b,X,i-1,j)
      endif
    endif
  endif
end
```

Xếp ba lô

□ Bài toán

- Có một ba lô có thể chứa tối đa trọng lượng W và có n đồ vật, mỗi đồ vật có trọng lượng w_i và giá trị b_i
 - W , w_i và b_i là các số nguyên
- Hãy xếp các đồ vật vào ba lô để tổng giá trị của ba lô là lớn nhất

■ Minh họa

	Đồ vật	Trọng lượng - w_i	Giá trị - b_i
<div><div>W = 12</div><div>Ba lô</div></div>		2	8
		5	10
		3	6
		9	2
		4	3

Xếp ba lô

□ Thuật toán vét cạn

- Có n đồ vật, nên có 2^n tập con các đồ vật của tập n đồ vật
- Duyệt tất cả 2^n tập con các đồ vật và chọn tập có tổng giá trị lớn nhất mà tổng trọng lượng nhỏ hơn W
- Độ phức tạp thuật toán là $O(2^n)$
 - Hàm mũ !

Xếp ba lô

□ Thuật toán chia để trị (1)

- Bài toán có thể chia thành các bài toán con
 - Thay vì xét k đồ vật, xét $k-1$ đồ vật ...
- Gọi $v_{k,w}$ là tổng giá trị lớn nhất của ba lô mà trọng lượng không vượt quá w khi chỉ sử dụng các đồ vật $1...k$
- Đối với mỗi đồ vật k cần trả lời câu hỏi
 - Ba lô hiện tại có thể chứa thêm đồ vật k hay không ?
- Trả lời
 - Nếu trọng lượng còn lại hiện tại w của ba lô nhỏ hơn w_k thì ba lô không thể chứa đồ vật k
 - Ngược lại, $w \geq w_k$ thì có hai trường hợp xảy ra:
 - Không thêm đồ vật k vào ba lô, chỉ sử dụng các đồ vật $1...k-1$
 - Thêm đồ vật k vào ba lô, khi đó giá trị của ba lô tăng lên b_k nhưng trọng lượng còn lại của ba lô giảm đi w_k (tức là bằng $w - w_k$)
 - Trường hợp nào cho tổng giá trị của ba lô lớn nhất sẽ được chọn

Xếp ba lô

□ Thuật toán chia để trị (2)

- Vậy, $v_{k,w}$ được tính như sau

$$v_{k,w} = \begin{cases} v_{k-1,w} & \text{if } w_k > w \\ \max\{v_{k-1,w}, v_{k-1,w-w_k} + b_k\} & \text{else} \end{cases}$$

■ Giải thích

- Nếu $w_k > w$, không thể thêm đồ vật k vào ba lô, chỉ xét các đồ vật $1...k-1$ (gồm $k-1$ đồ vật)
- Nếu $w_k \leq w$,
 - Nếu chỉ cần sử dụng các đồ vật $1...k-1$ mà cho tổng giá trị ba lô lớn nhất, thì không sử dụng đồ vật k
 - Sử dụng đồ vật k nếu cho tổng giá trị ba lô lớn nhất, khi đó xét các đồ vật $1...k-1$ với trọng lượng còn lại của ba lô là $w - w_k$

Xếp ba lô

□ Thuật toán chia để trị (3)

- Lưu ý rằng, $v_{k,w} = 0$ nếu $k = 0$
- Vậy công thức đầy đủ tính $v_{k,w}$ là

$$v_{k,w} = \begin{cases} 0 & k = 0 \\ v_{k-1,w} & w_k > w \\ \max(v_{k-1,w}, v_{k-1,w-w_k} + b_k) & w_k \leq w \end{cases}$$

- Giải pháp tối ưu của bài toán là giá trị $v_{n,w}$

Xếp ba lô

▣ Thuật toán chia để trị (4)

```
balo (k, w)
begin
  if (k = 0) then return 0
  else
    if ( $w_k > w$ ) then
      return balo(k-1,w) // không sử dụng đồ vật k
    else
      x = balo(k-1,w)
      y = balo(k-1,w- $w_k$ )
      return (max(x, y+ $b_k$ ))
    endif
  endif
end
```

Sử dụng: balo(n, W)

Xếp ba lô

□ Thuật toán chia để trị (5)

■ Đánh giá độ phức tạp

- Gọi $C(n)$ là độ phức tạp trong trường hợp xấu nhất
- Từ thuật toán, ta có phương trình truy hồi

$$C(n) = \begin{cases} c & n = 0 \\ \max(C(n-1), 2C(n-1)) + d & n > 0 \end{cases}$$
$$= \begin{cases} c & n = 0 \\ 2C(n-1) + d & n > 0 \end{cases}$$

với c và d là các hằng số

- Vậy độ phức tạp của thuật toán $O(2^n)$
 - Hàm mũ !

Xếp ba lô

□ Thuật toán quy hoạch động (1)

- Sử dụng mảng $v[0..n, 0..W]$ để lưu lại các giải pháp của các bài toán con
- $v[k, w]$ là tổng giá trị lớn nhất của ba lô mà trọng lượng không vượt quá w khi chỉ sử dụng các đồ vật $1...k$
- Ban đầu

$$v[0, w] = 0 \text{ với mọi } w$$

$$v[k, w] = 0 \text{ với mọi } k$$

- Sau đó, $v[k, w]$ sẽ được tính theo $v[k-1, w]$ hoặc $v[k-1, w-w_k]$

Xếp ba lô

▣ Thuật toán quy hoạch động (2)

```
balo (n, W)
begin
  for k from 0 to n do v[k,0] = 0 endfor
  for w from 0 to W do v[0,w] = 0 endfor
  for k from 1 to n do
    for w from 1 to W do
      if ( $w_k \leq w$ ) then // có thể sử dụng đồ vật k
         $v[k,w] = \max(v[k-1,w], v[k-1,w-w_k] + b_k)$ 
      else //  $w_k > w$ 
         $v[k,w] = v[k-1,w]$  // không sử dụng đồ vật k
      endif
    endfor
  endfor
end
```

Xếp ba lô

□ Thuật toán quy hoạch động (3)

```
balo (n, W)
begin
  for k from 0 to n do v[k,0] = 0 endfor
  for w from 0 to W do v[0,w] = 0 endfor
  for k from 1 to n do
    for w from 1 to W do
      if ( $w_k \leq w$ ) then // có thể sử dụng đồ vật k
        if ( $b_i + v[i-1, w-w_i] > v[i-1, w]$ ) then
           $v[i, w] = b_i + v[i-1, w-w_i]$  // sử dụng đồ vật k
        else
           $v[i, w] = v[i-1, w]$  // không sử dụng đồ vật k
        endif
      else //  $w_k > w$ 
         $v[k, w] = v[k-1, w]$  // không sử dụng đồ vật k
      endif
    endfor
  endfor
end
```

Xếp ba lô

□ Thuật toán quy hoạch động (4)

■ Đánh giá độ phức tạp

- Thuật toán tính mảng $n \times W$ phần tử bởi hai vòng lặp lồng nhau
- Độ phức tạp $O(nW)$

■ Ví dụ minh họa

$W = 5$ (trọng lượng tối đa ba lô có thể chứa)

$n = 4$ (4 đồ vật)

Các đồ vật lần lượt có (trọng lượng, giá trị):
(2,3), (3,4), (4,5), (5,6)

Xếp ba lô

Ví dụ (2)

	k	0	1	2	3	4
W						
0		0				
1		0				
2		0				
3		0				
4		0				
5		0				

for w = 0 to W do v[0,w] = 0

Xếp ba lô

Ví dụ (3)

	k	0	1	2	3	4
w	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					
5	0					

for k = 0 to n do v[k,0] = 0

Xếp ba lô

Ví dụ (4)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	→ 0			
2		0				
3		0				
4		0				
5		0				

$k=1$

$b_k=3$

$w_k=2$

$w=1$

$w-w_k=-1$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (5)

	k	0	1	2	3	4
w						
0		0	0	0	0	0
1		0	0			
2		0	3			
3		0				
4		0				
5		0				

$k=1$

$b_k=3$

$w_k=2$

$w=2$

$w-w_k=0$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else


$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (6)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0			
2		0	3			
3		0	3			
4		0				
5		0				



$k=1$

$b_k=3$

$w_k=2$

$w=3$

$w-w_k=1$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (7)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0			
2		0	3			
3		0	3			
4		0	3			
5		0				

$k=1$

$b_k=3$

$w_k=2$

$w=4$

$w-w_k=2$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (8)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0			
2		0	3			
3		0	3			
4		0	3			
5		0	3			

$k=1$

$b_k=3$

$w_k=2$

$w=5$

$w-w_k=2$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (9)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	→ 0		
2		0	3			
3		0	3			
4		0	3			
5		0	3			

$k=2$

$b_k=4$

$w_k=3$

$w=1$

$w-w_k=-2$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (10)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0		
2		0	3	→ 3		
3		0	3			
4		0	3			
5		0	3			

$k=2$

$b_k=4$

$w_k=3$

$w=2$

$w-w_k=-1$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (11)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0		
2		0	3	3		
3		0	3	4		
4		0	3			
5		0	3			

$k=2$

$b_k=4$

$w_k=3$

$w=3$

$w-w_k=0$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (12)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0		
2		0	3	3		
3		0	3	4		
4		0	3	4		
5		0	3			

$k=2$

$b_k=4$

$w_k=3$

$w=4$

$w-w_k=1$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (13)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0		
2		0	3	3		
3		0	3	4		
4		0	3	4		
5		0	3	7		

$k=2$

$b_k=4$

$w_k=3$

$w=5$

$w-w_k=2$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (14)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0 → 0		
2		0	3	3 → 3		
3		0	3	4 → 4		
4		0	3	4		
5		0	3	7		

$k=3$

$b_k=5$

$w_k=4$

$w=1..3$

$w - w_k < 0$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w - w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (15)

	k	0	1	2	3	4
W						
0	0	0	0	0	0	0
1	0	0	0	0	0	
2	0	3	3	3		
3	0	3	4	4		
4	0	3	4	5		
5	0	3	7			

$k=3$

$b_k=5$

$w_k=4$

$w=4$

$w - w_k = 0$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (16)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0	0	
2		0	3	3	3	
3		0	3	4	4	
4		0	3	4	5	
5		0	3	7	→ 7	

$k=3$

$b_k=5$

$w_k=4$

$w=5$

$w-w_k=1$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (17)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0	0 →	0
2		0	3	3	3 →	3
3		0	3	4	4 →	4
4		0	3	4	5 →	5
5		0	3	7	7	

$k=4$

$b_k=6$

$w_k=5$

$w=1..4$

$w-w_k < 0$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

Ví dụ (18)

	k	0	1	2	3	4
W						
0		0	0	0	0	0
1		0	0	0	0	0
2		0	3	3	3	3
3		0	3	4	4	4
4		0	3	4	5	5
5		0	3	7	7	7

$k=4$

$b_k=6$

$w_k=5$

$w=5$

$w-w_k=0$

Đồ vật

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

if $w_k \leq w$ then // đồ vật k có thể được sử dụng

if $b_k + v[k-1, w-w_k] > v[k-1, w]$ then

$v[k, w] = b_k + v[k-1, w-w_k]$

else

$v[k, w] = v[k-1, w]$

else $v[k, w] = v[k-1, w]$ // $w_k > w$, đồ vật k không được sử dụng

Xếp ba lô

- Thuật toán *balô* chỉ mới tìm được tổng giá trị lớn nhất mà ba lô có thể chứa
- Bài tập
 - Xây dựng giải pháp tối ưu
 - Các vật được chứa trong ba lô
 - Minh họa

Các đồ vật
sử dụng: 1, 2



	k	0	1	2	3	4
W	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	3	3	3	3	3
3	0	3	4	4	4	4
4	0	3	4	5	5	5
5	0	3	7	7	7	7

Bài tập

□ Bài 1

- Số Fibonacci được định nghĩa

$$F_0 = 1, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$

Xây dựng thuật toán quy hoạch động tính F_n

□ Bài 2

- Số Catalan được định nghĩa

$$T(n) = \begin{cases} 1 & n = 1 \\ \sum_{i=1}^{n-1} T(i).T(n-i) & n > 1 \end{cases}$$

Xây dựng thuật toán quy hoạch động tính T_n

Bài tập

□ Bài 3

Một băng cát-xét gồm 2 mặt, mỗi mặt có thể ghi được d phút (chẳng hạn $d=30$). Một đĩa CD chứa n bài hát có tổng thời lượng m phút ($m>d$, chẳng hạn $m=78$). Bài hát i có thời lượng d_i phút, $d_i>0$. Cần chọn các bài hát từ đĩa CD lên các mặt băng cát-xét sao cho tổng thời lượng là lớn nhất. Một bài hát được ghi lên một mặt đĩa hoặc không được ghi.

Gọi $time(i, t_1, t_2)$ là thời lượng lớn nhất có thể ghi lên băng cát-xét, trong đó t_1 (t_2) là thời lượng lớn nhất có thể ghi lên mặt thứ nhất (mặt thứ hai), và chỉ sử dụng i bài hát đầu tiên. Khi đó $time(n, d, d)$ là giải pháp của bài toán.

Hãy thực hiện:

1. Xây dựng hệ thức truy hồi tính $time(i, t_1, t_2)$.
2. Xây dựng thuật toán đệ quy tính hệ thức truy hồi trên.
3. Đánh giá độ phức tạp thuật toán đệ quy tính hệ thức truy hồi trên (giả thiết $d_i = 1$ với mọi i).
4. Xây dựng thuật toán quy hoạch động.
5. Đánh giá độ phức tạp thuật toán quy hoạch động.

Giải thích 1

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n \geq 2 \end{cases}$$

□ Chứng minh $P(n) = \Omega(2^n)$

■ Chứng minh bằng quy nạp $P(n) \geq 2^{n-2}$

□ $P(n)$ đúng với $n \leq 4$

□ Với $n \geq 5$

$$\begin{aligned} P(n) &= \sum_{k=1}^{n-1} P(k)P(n-k) \\ &\geq \sum_{k=1}^{n-1} 2^{k-2} 2^{n-k-2} = \sum_{k=1}^{n-1} 2^{n-4} = (n-1)2^{n-4} \\ &\geq 2^{n-2} \quad (\text{do } n \geq 5) \end{aligned}$$

■ Vậy $P(n) = \Omega(2^n)$

