PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN (Design and Analysis of Algorithms)

L/O/G/O

GV: HUỲNH THỊ THANH THƯƠNG

Email: hh.thanhthuong@gmail.com

thuonghtt@uit.edu.vn

Nội dung

- Phương pháp chia để trị
- Phương pháp tham lam
- Phương pháp quay lui
- Phương pháp giảm để trị
- Phương pháp biến đổi để trị
- Phương pháp quy hoạch động

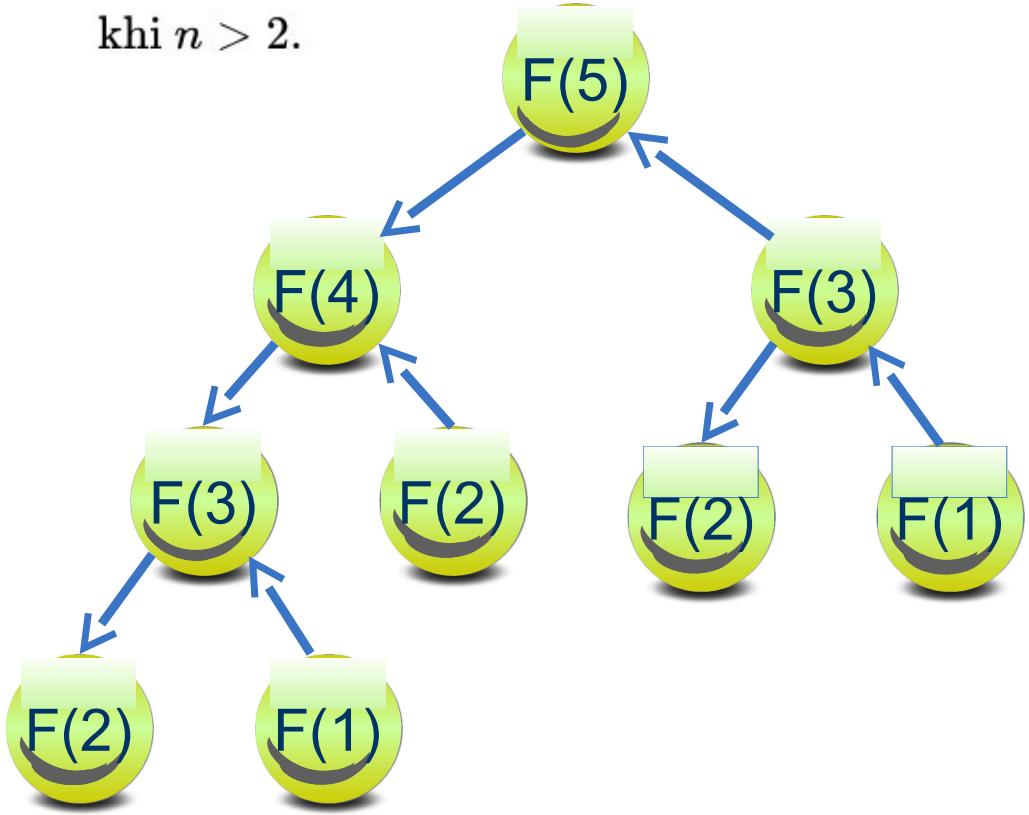
  (Dynamic programming)

Đặt vấn đề

- Xác định phần tử thứ n của dãy số Fibonacci

Công thức truy hồi của dãy:

$$F(n) := \begin{cases} 1, & \text{khi } n = 1; \\ 1, & \text{khi } n = 2; \\ F(n-1) + F(n-2) & \text{khi } n > 2. \end{cases}$$



Ví dụ: bài toán có công thức truy hồi

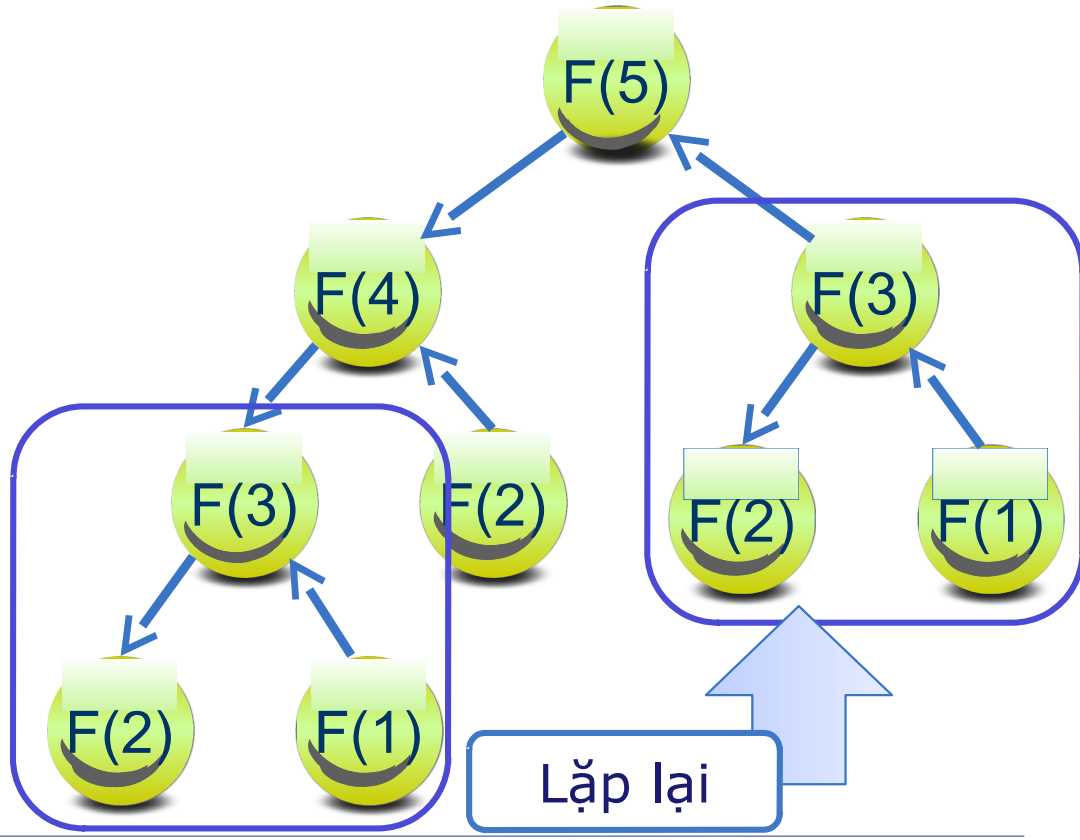
❖Ví dụ 1: Dãy số Fibonacci

$$F(n)=\begin{cases}1 & n\le 2\\ F(n-1)+F(n-2) & n>2\end{cases}$$

- Thuật toán chia để trị

```
long Fibo(int n)
{
   if (n == 1 || n == 2)
      return 1;
   return Fibo(n–1)+Fibo(n–2);
}
```

- Độ phức tạp: $O(2^n)$



Phương pháp quy hoạch động

- Chia để trị: Một số bài toán con nào đó có thể giải nhiều lần → chi phí cao, có thể $O(c^n)$
- Quy hoạch động: Như divide and conquer (kết hợp lời giải của các bài toán con) → nâng cấp, khắc phục nhược điểm của D&C
- Thường được vận dụng để giải bài toán tối ưu, bài toán có công thức truy hồi

Khi nào dùng quy hoạch động? Dấu hiệu

Bài toán tối ưu phải có 2 đặc trưng sau thì có thể áp dụng QHĐ

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

Khi nào dùng quy hoạch động? Dấu hiệu

Overlapping subproblems

A recursive solution contains a "small" number of distinct subproblems repeated many times.

Ý tưởng quy hoạch động

❖ **Khắc phục việc giải dư thừa 1 số bài toán con: lưu trữ kết quả + tra cứu**

- Tạo ra 1 bảng (hay vector) để lưu trữ kết quả của các bài toán con
- Sử dụng kết quả đã lưu trong bảng mà không cần giải lại bài toán con
- Giải các bài toán con theo hướng bottom-up (từ nhỏ đến lớn).

3-Steps of Developing Dynamic Programming Algorithm

❖ **Các bước thực hiện:**

- Bước 1: Phân tích đặc trưng của cấu trúc lời giải tối ưu + Xác định hàm (phương trình) quy hoạch động
- Bước 2: Tạo bảng: lấp đầy bảng theo 1 quy luật nào đó
  - Gán giá trị cho 1 số ô nào đó
  - Xác định giá trị của 1 số ô khác nhờ vào giá trị các ô trước đó
- Bước 3: Tra bảng và truy xuất/xây dựng lời giải của bài toán

Nhận xét Ưu nhược điểm

❖ Ưu điểm:

- Thực hiện nhanh ← không giải lại 1 số bài toán con
- Thường được vận dụng để giải các bài toán tối ưu, bài toán có công thức truy hồi

Nhận xét Ưu nhược điểm

❖ Nhược điểm:

- Không hiệu quả khi:
  - Không tìm được công thức truy hồi
  - Số lượng bài toán con và lưu trữ kết quả rất lớn
  - Sự kết hợp lời giải của các bài toán con chưa chắc cho lời giải của bài toán đầu

So sánh

- Greedy Method vs. Dynamic Programming
- Divide-and-Conquer vs. Dynamic Programming

Ví dụ: Bài toán có công thức truy hồi

❖Dãy số Fibonacci

- Thuật toán quy hoạch động: lưu kết quả bài toán con đã tính để dùng lại

```
// tạo mảng F[n] để lưu giá trị

F[1] = 1; F[2] = 1;
for (i = 3; i<=n; i++)
    F[i] = F[i-1] + F[i-2];
Xuất F[n];
```

- Độ phức tạp: O(n)

Ví dụ: Bài toán có công thức truy hồi

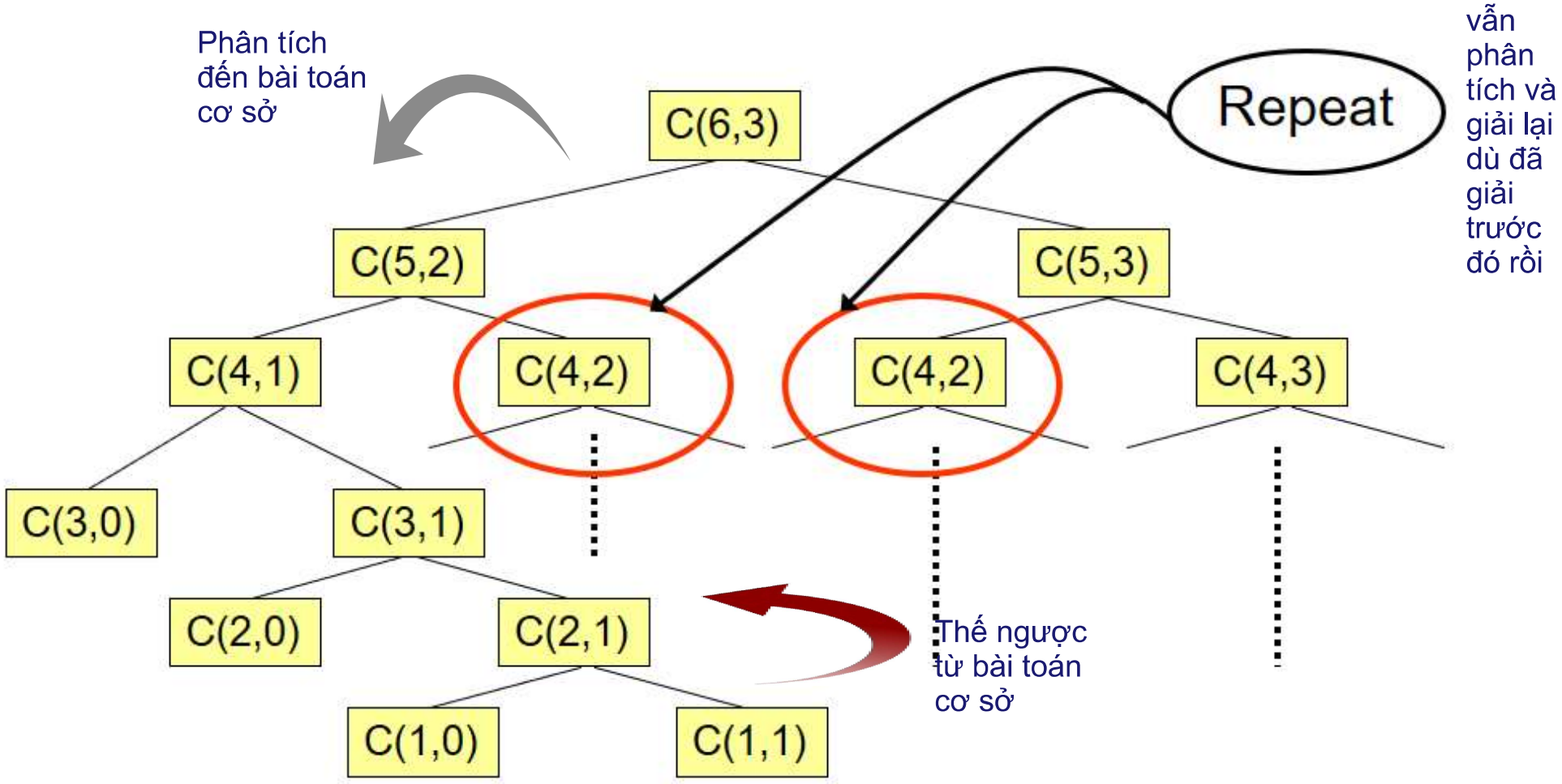
❖ Ví dụ 2: Tính tổ hợp chập k của n

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{if } 0<k<n \\ 1 & \text{if } k=0 \text{ or } k=n \\ 0 & \text{otherwise} \end{cases}$$

- Thuật toán chia để trị: Độ phức tạp: $O(2^n)$

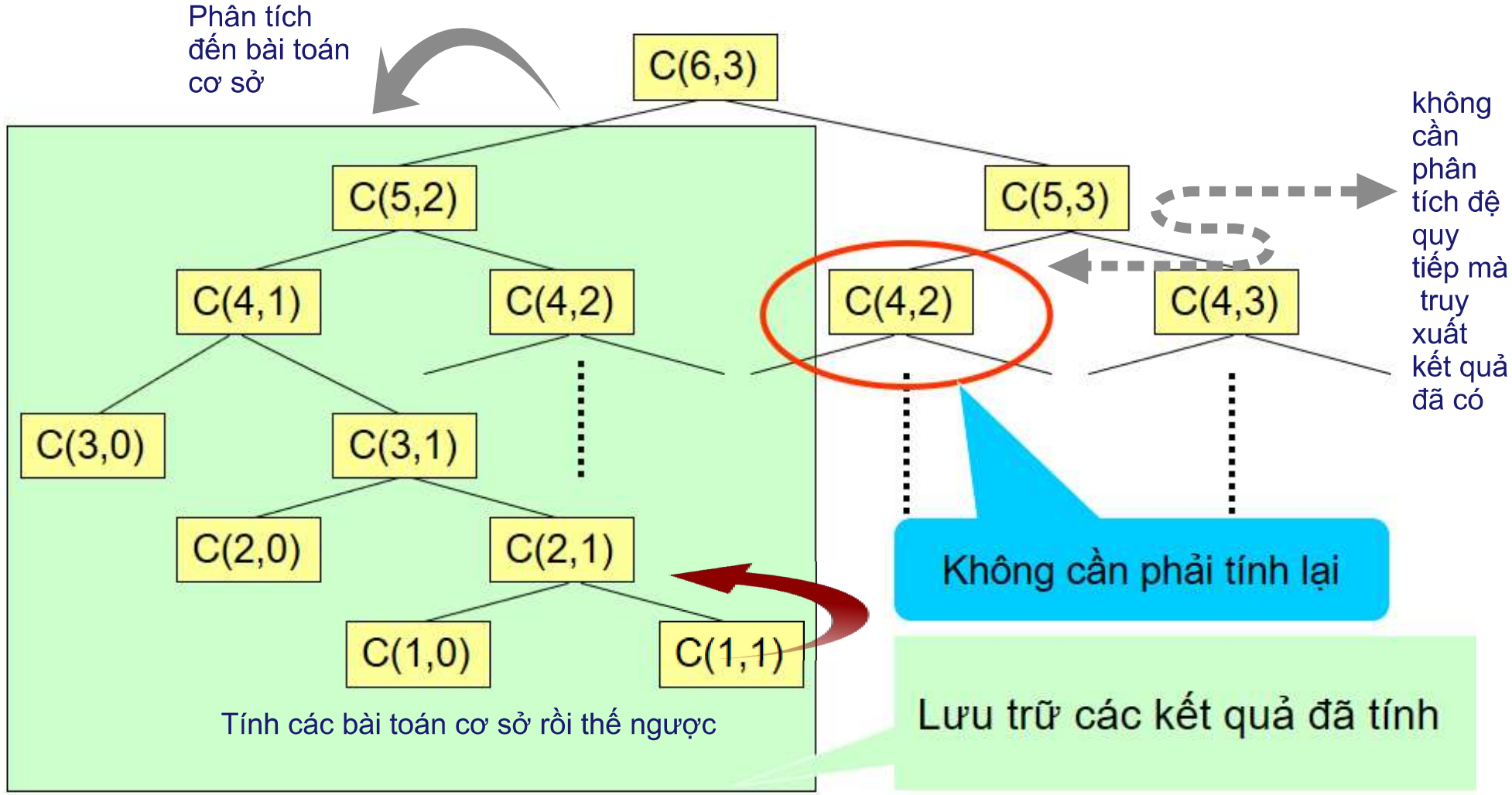
```
C(n,k)
{
    if k = 0 or k =n then return 1
    if k < 0 or k >n then return 0
    return( C(n-1,k-1) + C(n-1,k) )
}
```

Top-Down Recursive

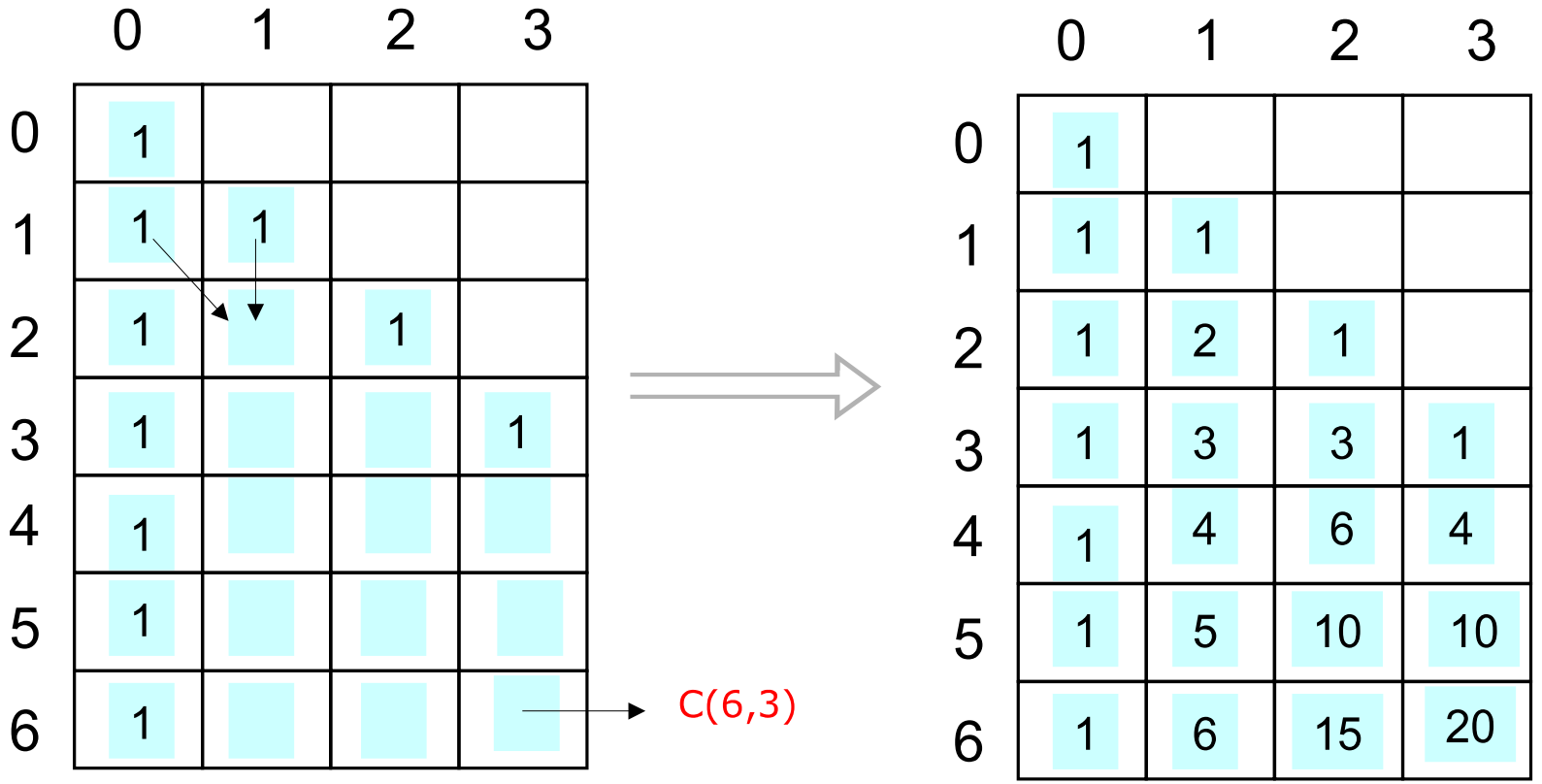


Phân hoạch và giải các bài toán con 1 cách độc lập (p
artition the problem into independent subproblems)

Top-Down Recursive + Memorization



Bottom-Up Dynamic Programming



giải từ các bài toán con nhỏ nhất đến lớn hơn
(Bottom-Up)

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{if } 0<k<n \\ 1 & \text{if } k=0 \text{ or } k=n \\ 0 & \text{otherwise} \end{cases}$$

viết mã giả ???

Ví dụ: Bài toán tối ưu

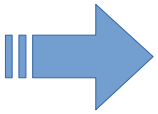
❖Bài tập 7: Bài toán nhân chuỗi/dãy ma trận (Matrix Chain Multiplication)

Input: Cho 1 chuỗi ma trận $<A_1 , A_2 , \ldots, A_n>$, với A_i có kích thước là $p_{i-1} \times p_i$

Output: Muốn tính tích của $A = A_1 \times A_2 \times \ldots \times A_n$. Yêu cầu: Xác định thứ tự nhân các ma trận sao cho số phép nhân được sử dụng là ít nhất.

Matrix Chain Multiplication

- Mẫu 1: $< A_1 , A_2 , A_3 >$
  - $A_1 : 3 \times 100$
  - $A_2 : 100 \times 7$
  - $A_3 : 7 \times 5$



Thứ tự nhân các ma trận (cách đóng mở ngoặc) ảnh hưởng lớn đến chi phí tính toán

$(A_1A_2)A_3$: 3.100.7 + 3.7.5 = 2205 phép nhân

$A_1(A_2A_3)$: 3.100.5 + 100.7.5 = 5000 phép nhân

Phương pháp quy hoạch động

❖Ý tưởng

- Bước 1.1: Phân tích đặc trưng "Optimal substructure"

  Bài toán ban đầu: đóng mở ngoặc tối ưu cho $A_1 \times A_2 \times ... \times A_n$ để cost nhỏ nhất.

  – Lời giải tối ưu có lần nhân cuối cùng được biểu diễn ở dạng:

$$1 \le k < n$$

$$(A_1 \times \ldots \times A_k) \times (A_{k+1} \times \ldots \times A_n)$$

bài toán con 1 bài toán con 2

- NX1: ……

Phương pháp quy hoạch động

- Lời giải tối ưu tách chuỗi (ra làm 2 phần) ngay tại vị trí k, khi đó :

để đóng mở ngoặc tối ưu cho $A_1 \times A_2 \times \dots \times A_n$ **thì phải**

{ đóng mở ngoặc tối ưu cho $A_1 \times \dots \times A_k$
{ đóng mở ngoặc tối ưu cho $A_{k+1} \times \dots \times A_n$

→ NX2: ……..

Ví dụ: Nếu $(A_1 \times (A_2 \times A_3)) \times A_4$ là tối ưu cho $A_1 \times A_2 \times A_3 \times A_4$,
Thì .....

Phương pháp quy hoạch động

❖Ý tưởng

- Bước 1.1: Phân tích đặc trưng "Optimal substructure"

$$(\underbrace{A_1 \text{ x } \dots \text{ x } A_k}_{\text{bài toán con 1}}) \text{ x } (\underbrace{A_{k+1} \text{ x } \dots \text{ x } A_n}_{\text{bài toán con 2}})$$

- NX3: cost (A_1 x ...x A_n) = ……..

(cost = số phép nhân)

Phương pháp quy hoạch động

❖Ý tưởng

- Bước 1.1: Phân tích đặc trưng "Optimal substructure"

  - **Kỳ vọng** :

  cost (A_1 x ...x A_n) tối ưu = cost (A_1 x ...x A_k) tối ưu

  + cost (A_{k+1} x ...x A_n) tối ưu

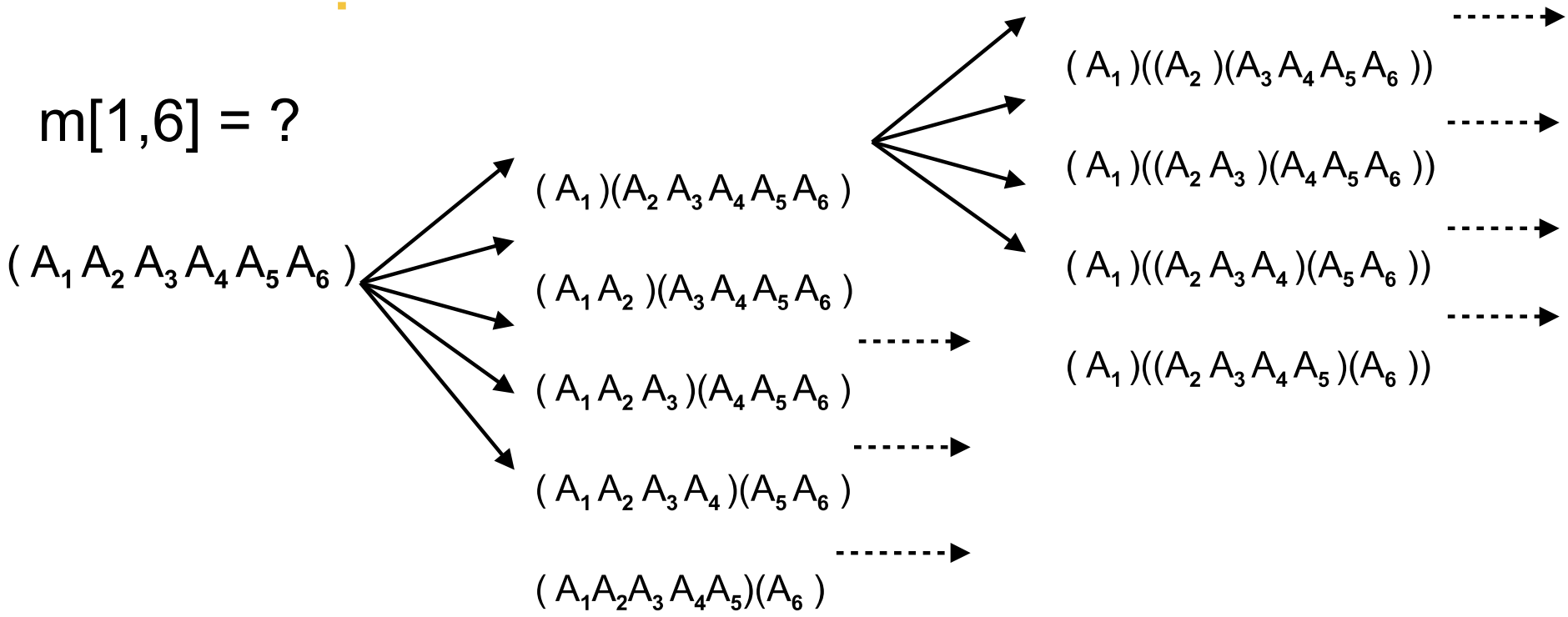
  + chi phí nhân 2 bài toán con lại

Quy ước

- A_1 có kích thước $p_0 \times p_1$
- A_2 : $p_1 \times p_2$
- A_i : $p_{i-1} \times p_i$
- $A_i \times \ldots \times A_j$: $p_{i-1} \times p_j$
- m[i, *j*] : minimum cost ($A_i \ldots A_j$)
- m[1 , *n*] : minimum cost ($A_1 \ldots A_n$)

Top-Down Recursive

Ví dụ



sự tối ưu còn phụ thuộc vào vị trí tách chuỗi (tức giá trị của k)

--> thử hết các giá trị có thể có của k, chọn giá trị cho cost nhỏ nhất

Ví dụ

$(A_1\ A_2\ A_3\ A_4\ A_5\ A_6)$ m[1 ,6] = ?

$(\ (A_1)\ (A_2\ A_3\ A_4\ A_5\ A_6)\ )$	$m[1,1] + m[2,6] + p_0\ p_1\ p_6$
$(\ (A_1\ A_2)\ (A_3\ A_4\ A_5\ A_6)\ )$	$m[1,2] + m[3,6] + p_0\ p_2\ p_6$
$(\ (A_1\ A_2\ A_3)\ (A_4\ A_5\ A_6)\ )$	$m[1,3] + m[4,6] + p_0\ p_3\ p_6$
$(\ (A_1\ A_2\ A_3\ A_4)\ (A_5\ A_6)\ )$	$m[1,4] + m[5,6] + p_0\ p_4\ p_6$
$(\ (A_1\ A_2\ A_3\ A_4\ A_5)\ (A_6)\ )$	$m[1,5] + m[6,6] + p_0\ p_5\ p_6$

m[1,6] = min of

A recursive formulation

$$m[i,j]=\begin{cases} 0 & (\text{nếu } i=j) \\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}.p_k.p_j\} & (\text{nếu } i<j) \end{cases}$$

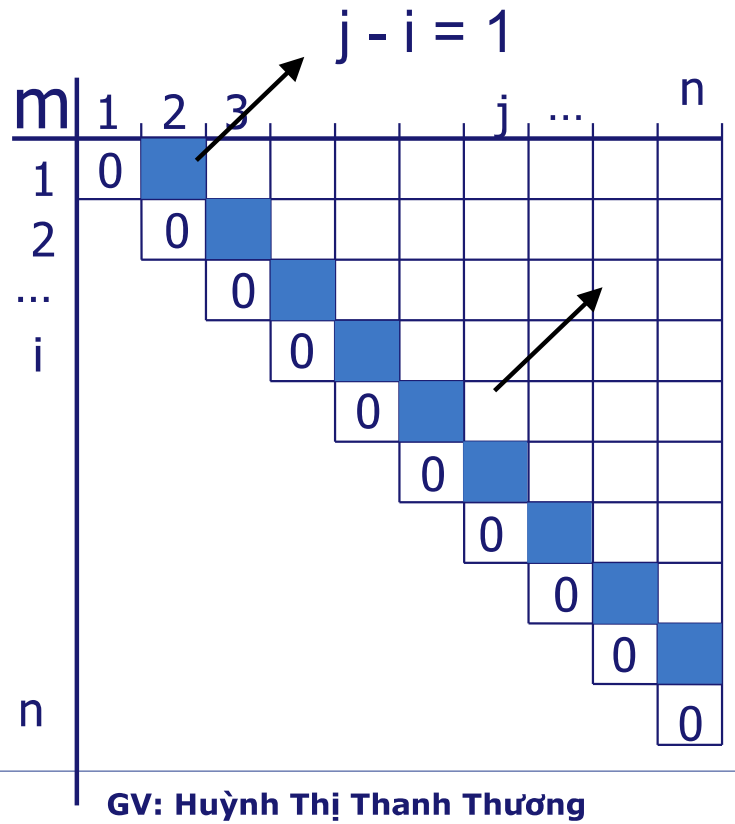
Giải bằng quy hoạch động

❖Ý tưởng

- **Bước 2: Tạo bảng và lưu giá trị**

- Lấp đầy bảng theo thứ tự tăng dần của (j – i)
  - Đầu tiên, gán các ô trên đường chéo chính = 0 (vì i=j)
  - Xét trường hợp j - i = 1

    (gán các ô dọc theo đường chéo kế trên đường chéo chính)
  - Tiếp tục xét j - i = 2, j - i = 3, ...

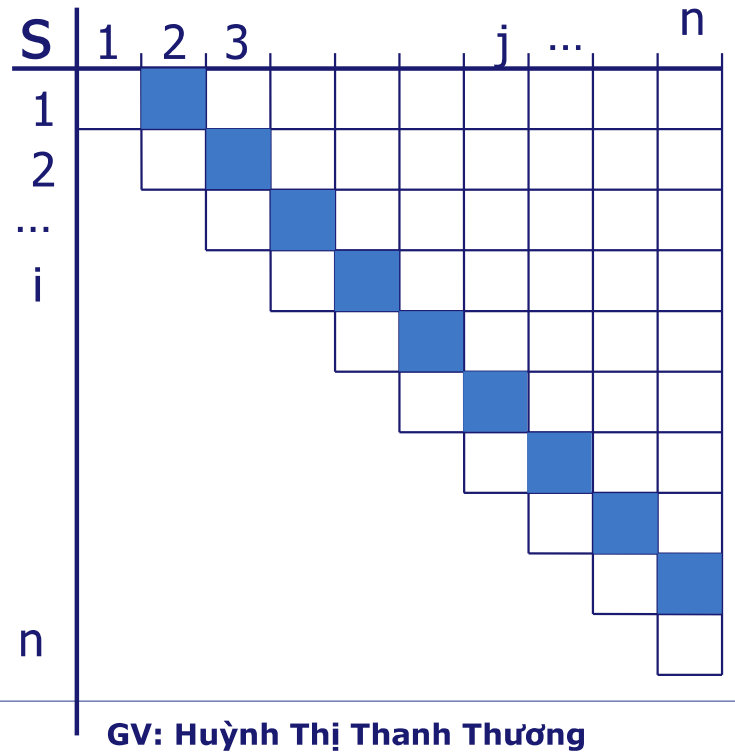


Giải bằng quy hoạch động

❖Ý tưởng

- Bước 3: Xây dựng lời giải của bài toán ban đầu

- Dùng 1 bảng khác lưu lại giá trị k mà tại đó thực hiện tách chuỗi
- s[i,j] = k, vị trí tối ưu để tách chuỗi (A_i x … x A_j)



Giải bằng quy hoạch động

❖Bài tập trên lớp:

- Viết mã giả giải bài toán bằng QHĐ

- Sau khi nắm được quy tắc tạo bảng, việc viết mã giả cho bài toán chỉ đơn giản là viết lại cách tạo bảng và return kết quả, cần 2 hàm:
  - Hàm tạo bảng (2 bảng lưu m[i,j] và s[i,j]): matrix (n) → hàm return về m[1,n] là cost tối ưu
  - Hàm truy tìm lời giải của bài toán, tức là trả về cách đặt các dấu ngoặc thể hiện thứ tự nhân các ma trận

Giải bằng quy hoạch động

❖Bài tập Bonus:

- Cài đặt hoàn chỉnh bài toán NHÂN MA TRẬN

- Nộp source code + các testcase (ít nhất 10 test) + video ghi lại quá trình chạy thử nghiệm (phòng trường hợp cô chạy ko được source code đã nộp)
- File word gồm mô tả chi tiết yêu cầu của bài toán, những quy ước đặc biệt nếu có, mã giả (có chú thích đầy đủ) cho người đọc dễ hiểu.

Bài tập trên lớp

- Ví dụ: ABCD
  - A : 10 × 5
  - B : 5 × 10
  - C : 10 × 5
  - D : 5 × 10

Bài tập trên lớp

- Ví dụ: ABCD
  - A : 10 × 5
  - B : 5 × 10
  - C : 10 × 5
  - D : 5 × 10

(giải hoàn chỉnh trên lớp)

m	1	2	3	4
1	**0** A	**500** AB	**500** A(BC)	**1000** Có 2 LG
2		**0** B	**250** BC	**500** (BC)D
3			**0** C	**500** CD
4				**0** D

Đánh giá độ phức tạp

Chia để trị: Top Down Recursive

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1))$$

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

Không chấp nhận được

Do “Overlapping subproblems”

Đánh giá độ phức tạp

Quy hoạch động: Bottom up

$$m[i,j] = \min_{i \le k < j} \{ m[i,k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$$T(n) = ???$$

SV tự chứng minh, nếu có thi GV sẽ hỏi á