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)

Ví dụ: bài toán có công thức truy hồi

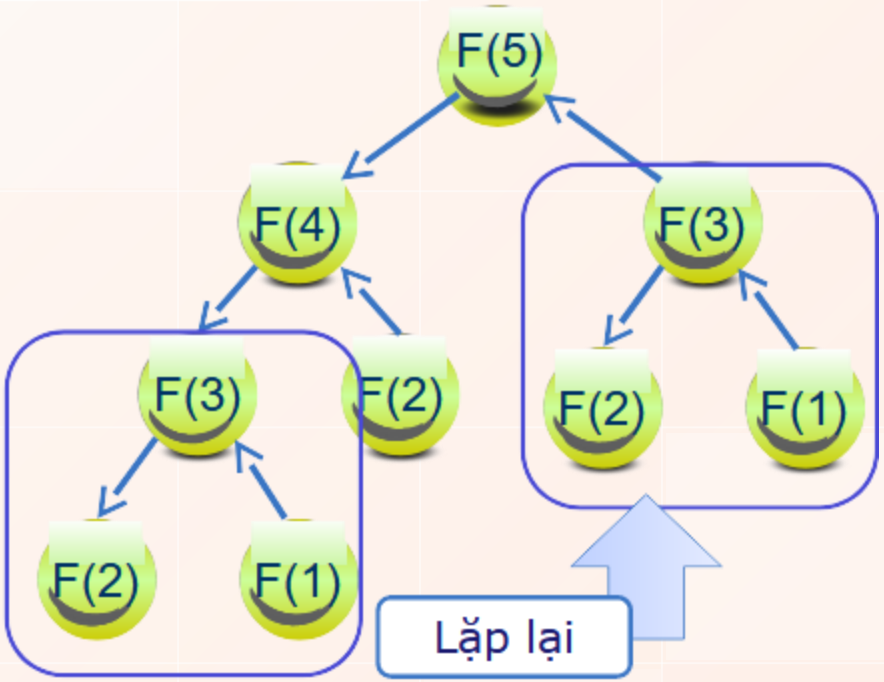
❖Ví dụ 1: Dãy số Fibonacci

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

- 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

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

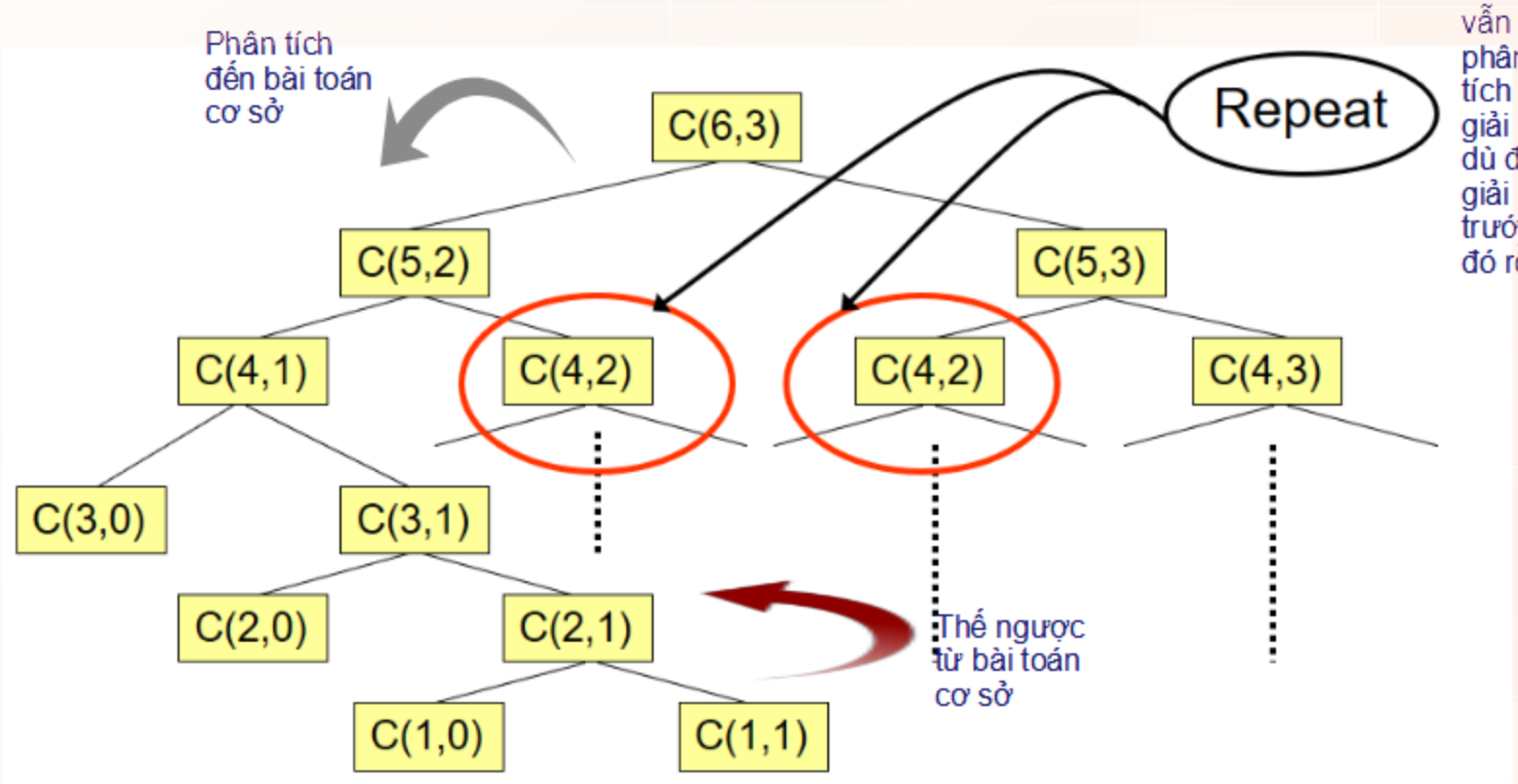
❖ Ví dụ 2: Tính tổ hợp chập k của n

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

- 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) )
}
```

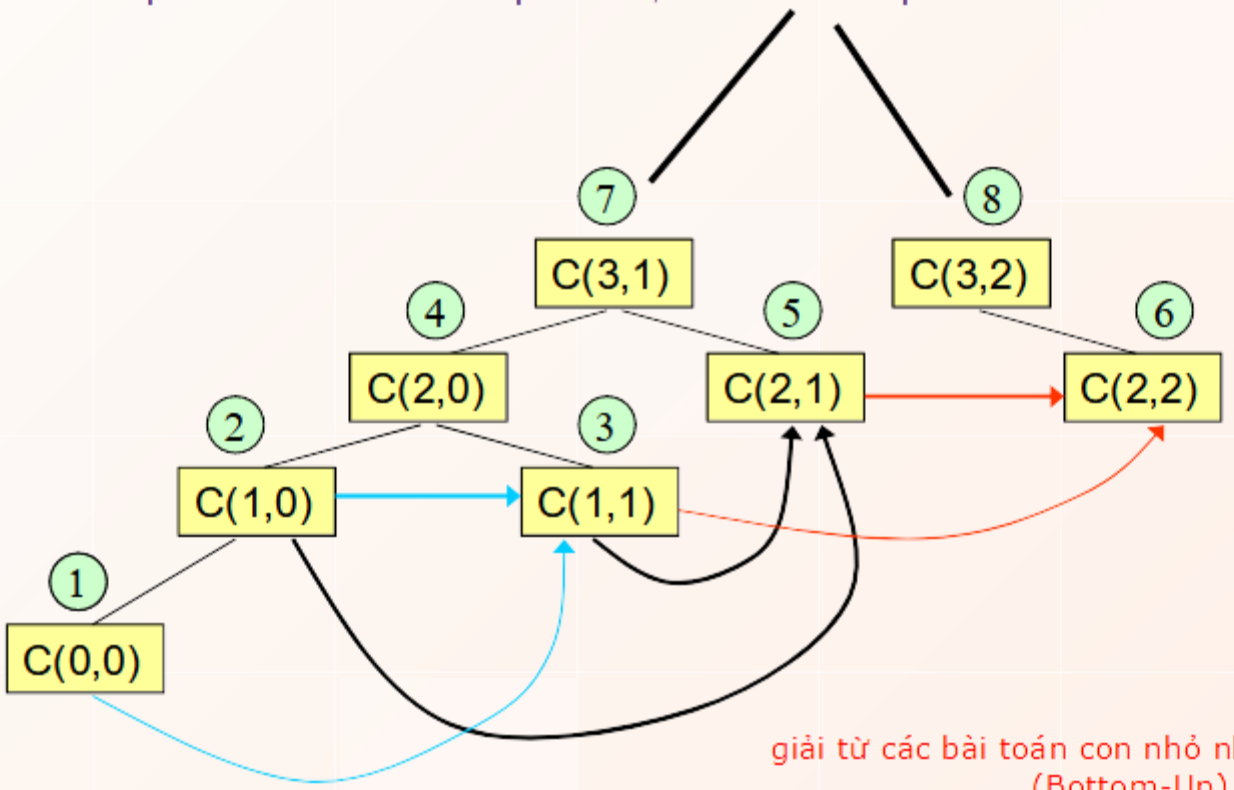
Top-Down Recursive



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

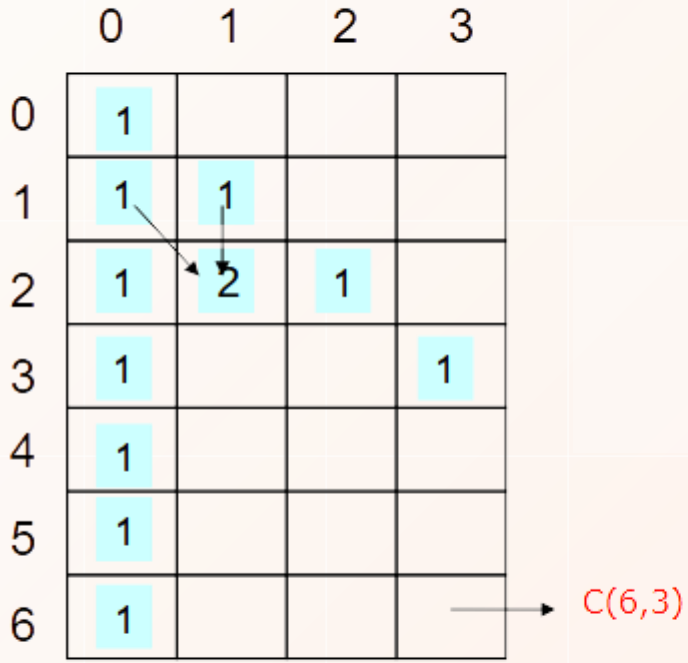
Dynamic Programming

các bài toán con không độc lập nhau, dùng chung những bài toán "cháu"
subproblems are not independent, share subsubproblems.



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

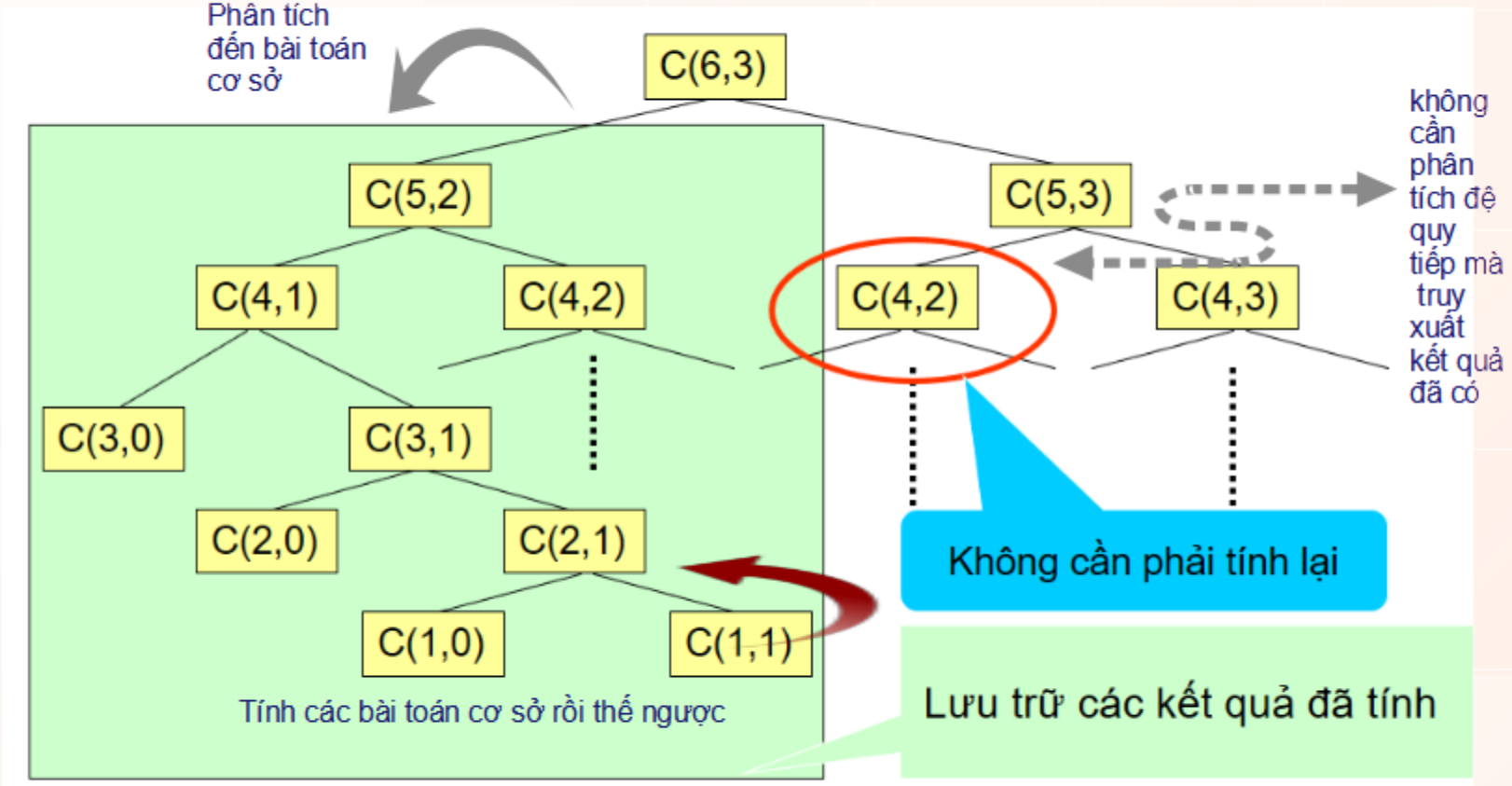
Bottom-Up Dynamic Programming



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

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

Top-Down Recursive + Memorization



Ví dụ: Bài toán tối ưu

❖Ví dụ 3: 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

$(A_1A_2)A_3$: 2205 phép nhân

$A_1(A_2A_3)$: 5000 phép nhân

Phương pháp quy hoạch động

❖Ý tưởng

- Bước 1.1: Phân tích đặc trưng "Overlapping subproblems"

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

  - Bài toán con có dạng như thế nào?

  - Cách phân rã những bài toán con kiểu top-down?

Top-Down Recursive

Overlapping subproblems

Giải bằng quy hoạch động

❖Ý tưởng

- Bước 1: Phân tích đặc trưng "Optimal substructure" (cấu trúc lời giải tối ưu)
  - Lời giải có cấu trúc thế nào?
    - Ví dụ: Chuỗi ma trận $< A_1 , A_2 , A_3 >$

      có 2 lời giải: $(A_1A_2)(A_3)$ $(A_1)(A_2A_3)$

Optimal substructure?

❖ Ví dụ

Chuỗi ma trận $< A_1 , A_2 , A_3 , A_4 >$

có thể được đóng mở ngoặc theo 5 cách:

- $A_1 (A_2(A_3A_4))$ $A_1 ((A_2A_3)A_4)$ $(A_1 A_2)(A_3A_4)$
- $(A_1 (A_2A_3))A_4$ $((A_1 A_2)A_3)A_4$

Những lời giải này có đặc điểm chung là gì?

Tìm mối liên hệ giữa lời giải của bài toán cha và bài toán con

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 $A_1 \times A_2 \times \ldots \times A_n$ để cost nhỏ nhất.

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

$$1 \le k < n$$

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

- NX1: nhận xét gì ở đây?

Phương pháp quy hoạch động

Ví dụ: Nếu $\mathbf{(A_1 \times (A_2 \times A_3)) \times A_4}$ là tối ưu cho

$A_1 \times A_2 \times A_3 \times A_4$,

Thì .....

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

bài toán con 1 bài toán con 2

để đóng mở ngoặc tối ưu cho $A_1 \times A_2 \times \ldots \times A_n$ thì phải làm sao?

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: ……..

Phương pháp quy hoạch động

❖Ý tưởng

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

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

- NX3: cost $(A_1 \times \dots \times A_n)$ = ...

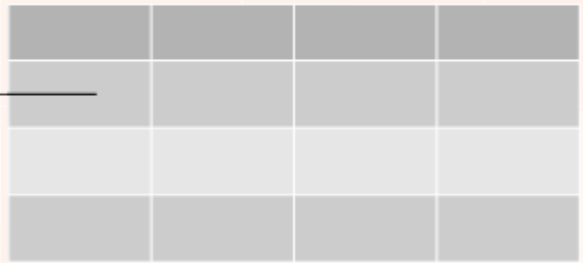
(cost = số phép nhân)

Phương pháp quy hoạch động

❖Ý tưởng:

- Bước 1.2: Xác định phương trình quy hoạch động
  - Bảng m có kích thước như thế nào?
  - m[i,j] = chứa thông tin gì?
  - Đáp án của bài toán chứa ở đâu trong bảng

+Mỗi ô phải chứa kết quả của 1 bài toán con nào đó
+ Các bài toán con có thể có là:
A1, A2, A3,A4
A1A2, A2A3, A3A4
A1A2A3, A2A3A4 ...-→ lưu tại ô nào?



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 ($A_1 \ldots A_n$)

Ví dụ

m[1,6] = ?

$(A_1 A_2 A_3 A_4 A_5 A_6)$

- $(A_1)(A_2 A_3 A_4 A_5 A_6)$ k = 1
- $(A_1 A_2)(A_3 A_4 A_5 A_6)$ k = 2
- $(A_1 A_2 A_3)(A_4 A_5 A_6)$
- $(A_1 A_2 A_3 A_4)(A_5 A_6)$
- $(A_1 A_2 A_3 A_4 A_5)(A_6)$ k = 5

có nhiều cách đặt ngoặc, vị trí ngoặc tương ứng giá trị k, vậy k nào sẽ cho lời giải tối ưu?

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

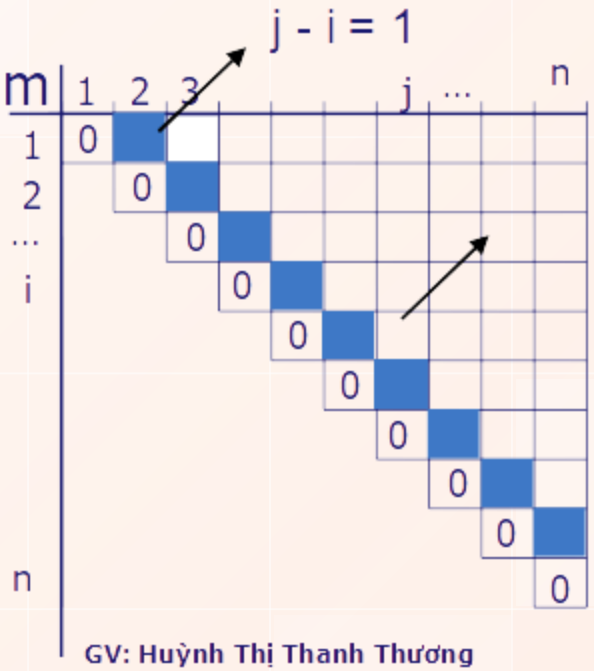
$$m[i,j]=\begin{cases} 0 & (\text{nếu } i=j) \\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}.p_k.p_j\} & (\text{nếu } i<j) \end{cases}$$

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)
  - Tiếp tục xét j - i = 1, j - i = 2, ...



Minh họa theo ví dụ

- A_1 : 10 × 20
- A_2: 20 × 50
- A_3: 50 × 1
- A_4 : 1 × 100

m	1	2	3	4
1	0	A_1A_2		
2	-	0	A_2A_3	
3	-	-	0	A_3A_4
4	-	-	-	0

Minh họa theo ví dụ

m	1	2	3	4
1	0	10000 A_1A_2	1200 $A_1(A_2A_3)$	2200 ???
2	-	0	1000 A_2A_3	3000 $(A_2A_3)(A_4)$
3	-	-	0	5000 A_3A_4
4	-	-	-	0

Trường hợp j-i = 3

$$m[1,4]\ (A_1A_2A_3A_4) = \min \begin{cases} m[1,1] + m[2,4] + 10 \times 20 \times 100 \\ m[1,2] + m[3,4] + 10 \times 50 \times 100 \\ m[1,3] + m[4,4] + 10 \times 1 \times 100 \end{cases}$$

$$= \min \{23000, 65000, 2200\} = 2200$$

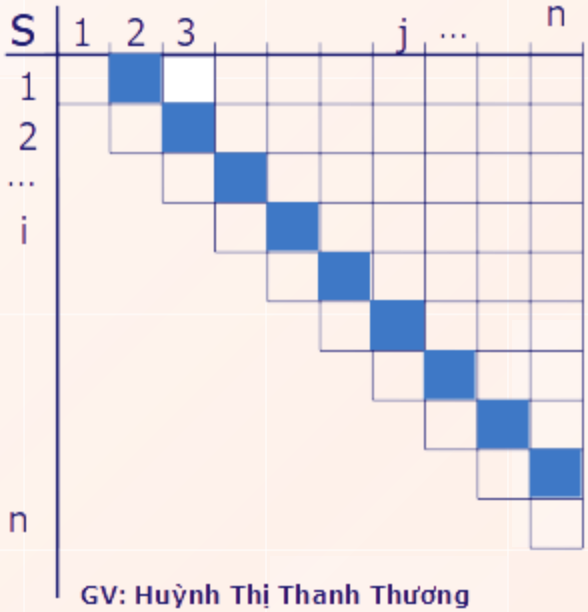
k=3

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

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

Đánh giá độ phức tạp

Chia để trị: Top Down Recursive

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

Bài tập Cộng điểm: từ VD.4→VD.12
Bài nào làm được thì làm

- **Yêu cầu:**

  Với mỗi bài tập: trình bày

  - Phân tích (Ý tưởng giải quyết)
    - ✓ Mô tả bài toán: chép lại đề cũng được, chọn lọc lại cho bớt thông tin ko cần thiết
    - ✓ Mô hình hóa bài toán: ghi rõ Input, Output, Các điều kiện ràng buộc nếu có
    - ✓ Phân tích giống như những VD mà GV hướng dẫn tại lớp, đặc biệt là QHĐ
  - Thiết kế thuật toán/thuật giải để giải các bài toán (trình bày mã giả có chú thích cho người đọc dễ hiểu, mô tả rõ ý nghĩa các CTDL (biến, mảng, …) dùng trong giải thuật).
  - Cho VD minh họa từng bước thực hiện theo thuật toán đã thiết kế (để người đọc dễ hiểu)
  - Phân tích độ phức tạp (Nếu được, không bắt buộc, bonus)
  - Cài đặt chương trình chạy được, submit lên Wecode để kiểm tra
  - Soạn bằng word/Latex/powerpoint, rồi chuyển sang PDF→ nộp tất cả các file

Bài tập về nhà

- Ví dụ 4: Chuỗi con chung dài nhất (Longest common subsequence)

  Chuỗi con của một chuỗi x nhận được từ x bằng cách xóa đi 1 số phần tử

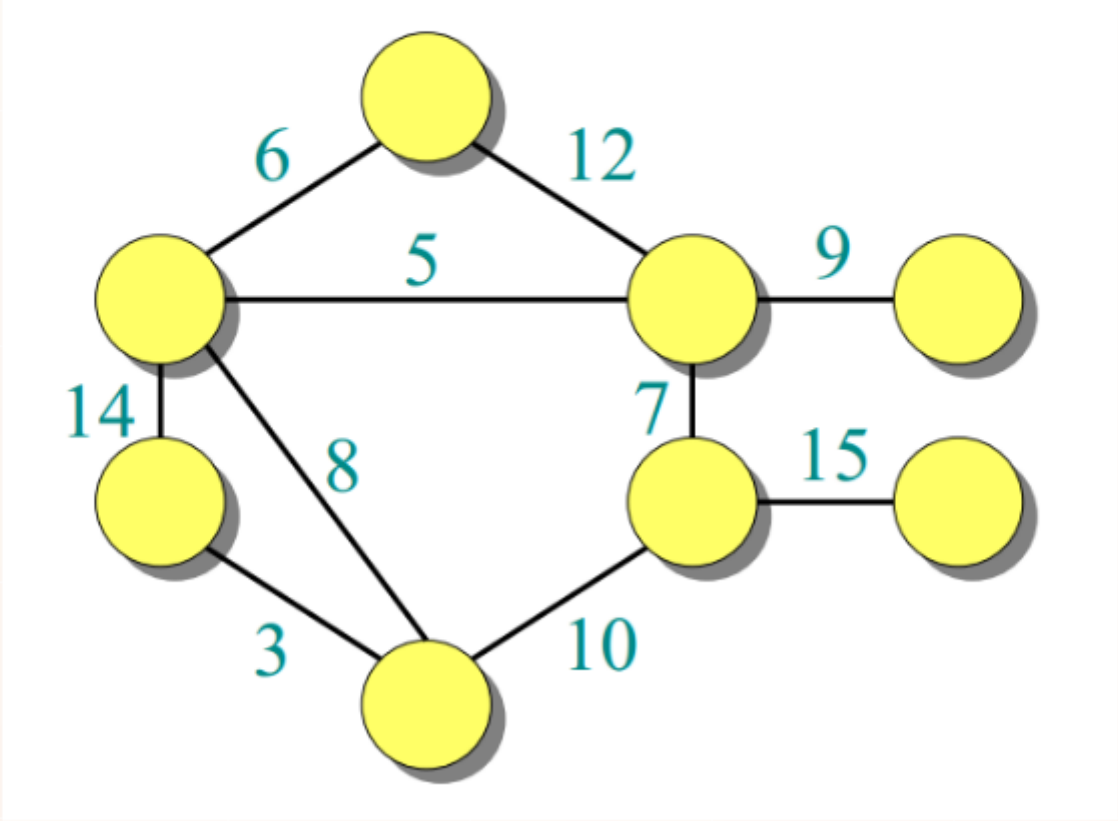
x:	A	B	C	B	D	A	B
y:	B	D	C	A	B	A	

Một số dãy con chung của x và y:

- A B A - B C A B - B D A B - B C B A

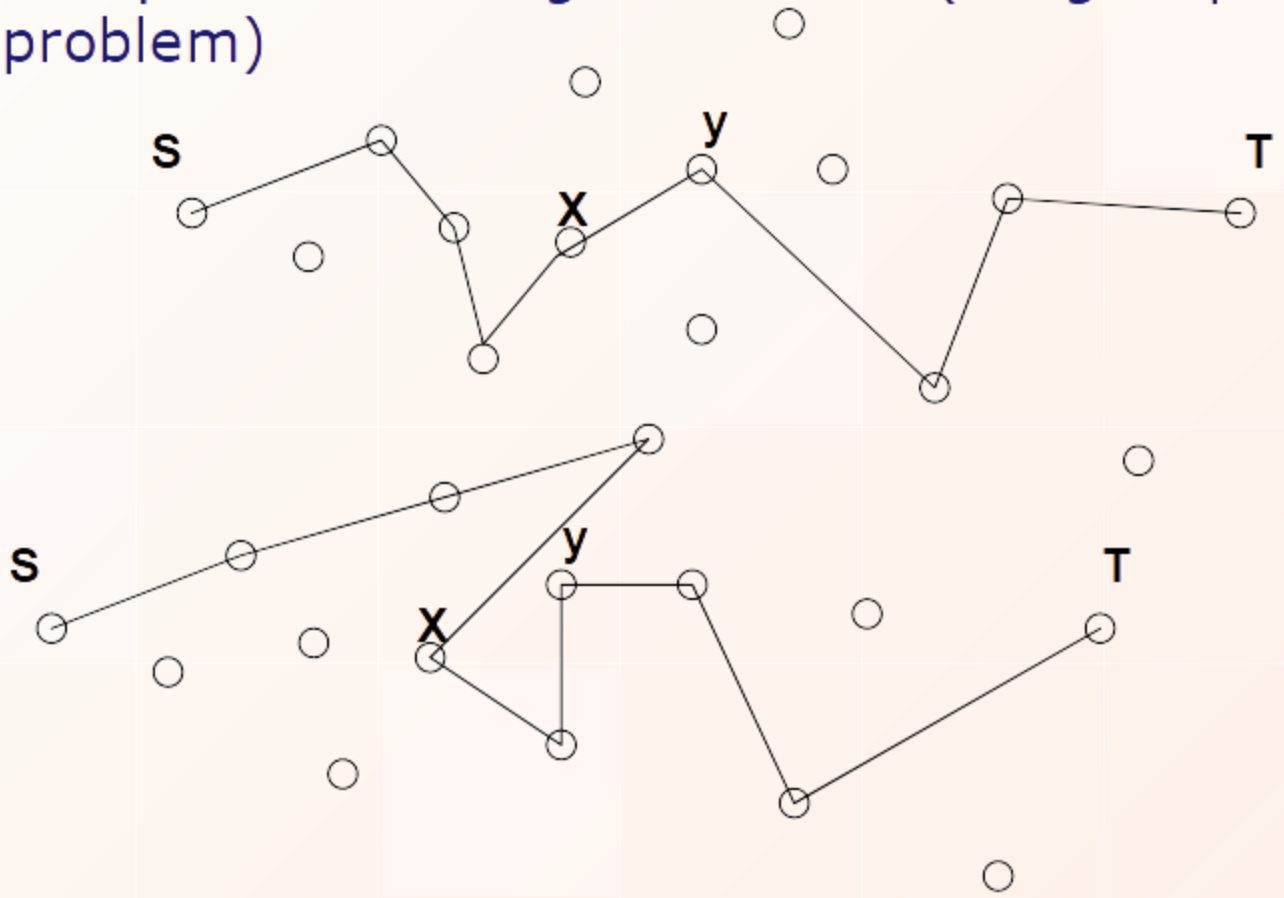
Bài tập về nhà

- Ví dụ 5: Tìm đường đi ngắn nhất (Shortest path problem) - Thuật toán Floyd



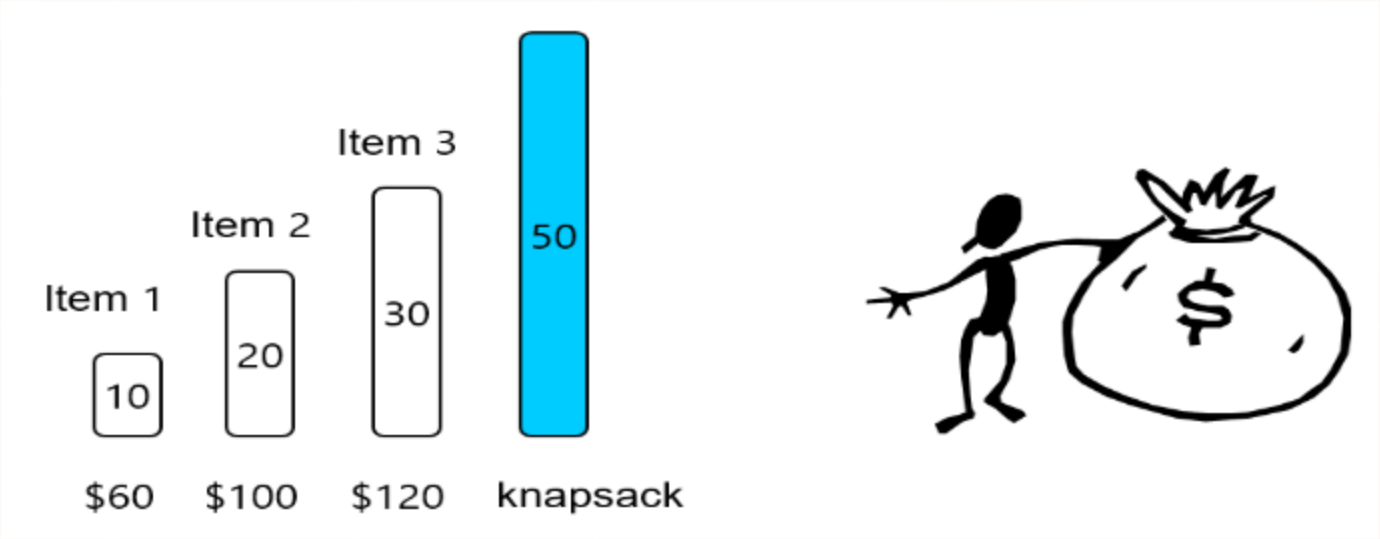
Bài tập về nhà

- Ví dụ 6: Tìm đường đi dài nhất (Longest path problem)



Bài tập về nhà

- Ví dụ 7: Bài toán balô/ăn trộm (Knapsack Problem - có 3 dạng)



Đọc tài liệu [5]. Anany Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2014. (Tài liệu chính)

Bài tập về nhà

- Ví dụ 8: Bài toán người giao hàng (Traveling Salesman Problem - TSP)

  (cách khác: tham lam, nhánh cận)

Bài tập về nhà

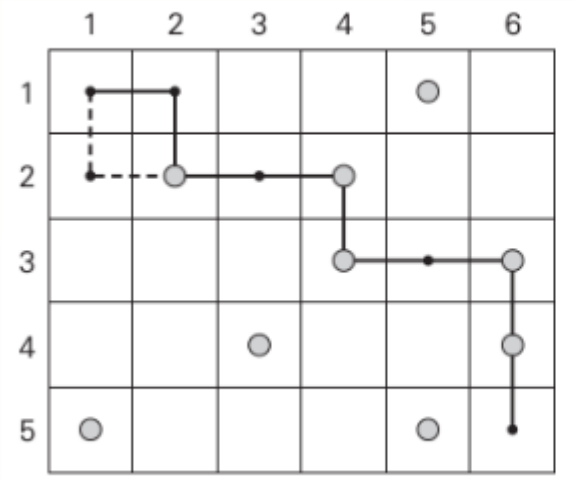
❖Ví dụ 9: Coin-row problem

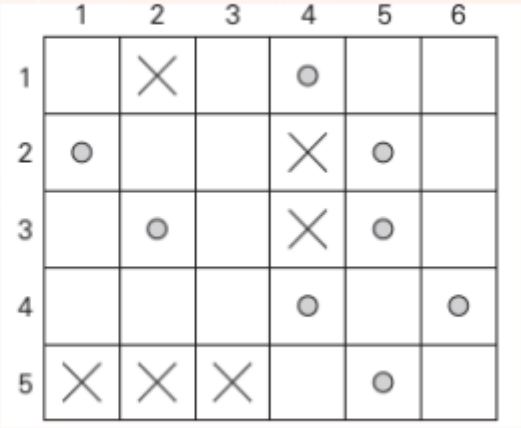
There is a row of n coins whose values are some positive integers (c_1 , c_2 , . . . , c_n), not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

Bài tập về nhà

❖Ví dụ 10: Coin-collecting problem

Several coins are placed in cells of an n × m board, no more than one coin per cell. A robot, located in the upper left cell of the board, needs to collect as many of the coins as possible and bring them to the bottom right cell. On each step, the robot can move either one cell to the right or one cell down from its current location. When the robot visits a cell with a coin, it always picks up that coin. Design an algorithm to find the maximum number of coins the robot can collect and a path it needs to follow to do this.





Bài tập về nhà

❖Ví dụ 11: Change-making problem

Give change for amount *n* using the minimum number of coins of denominations

$d_1 < d_2 < \ldots < d_m$

❖ Cách khác: giải bằng phương pháp "Tham lam'

Bài tập về nhà

- Ví dụ 12: Có n người xếp hàng mua vé.Thời gian bán vé cho người thứ i là t_i. Mỗi người mua tối thiểu 1 vé và được tối đa 2 vé, có thể mua hộ cho người đứng sau mình. Người thứ i mua vé hộ cho người thứ i+1 thì thời gian mua vé cho 2 người là p_i. Xác định phương án sao cho n người đều có vé với thời gian ít nhất.
- Khác: bài tập trong giáo trình của môn học

[5]. Anany Levitin, Introduction to the Design and Analysis of Algorithms, 3rd Edition, 2014. (Tài liệu chính)