



Phân Tích và Thiết Kế Thuật Toán


CS112.L21.KHTN

DYNAMIC PROGRAMMING

Nhóm 04:

19522291 - Lê Nguyễn Sĩ Thọ

19520874 – Dương Văn Nhật Quang





Understanding Dynamic Programming

TABLE OF CONTENTS

01

GIỚI THIỆU

Khái niệm và các bước thực hiện

02

VÍ DỤ

Giải và phân tích các bài toán
ví dụ

03

TỔNG KẾT

Tổng kết và bài tập

01

GIỚI THIỆU



Quy hoạch động là gì ?

Là một phương pháp thiết kế thuật toán.

Giải quyết bài toán bằng cách kết hợp lời giải của các bài toán con của nó.



Giống với Chia Để Trị
(Divide and Conquer)

Bài toán 1 : Dãy Fibonacci

Dãy số Fibonacci được cho bởi công thức đệ quy và các điều kiện ban đầu như sau:

$$\begin{cases} F_n = F_{n-1} + F_{n-2}, \forall n \geq 2 \\ F_0 = 0 \\ F_1 = 1 \end{cases}$$

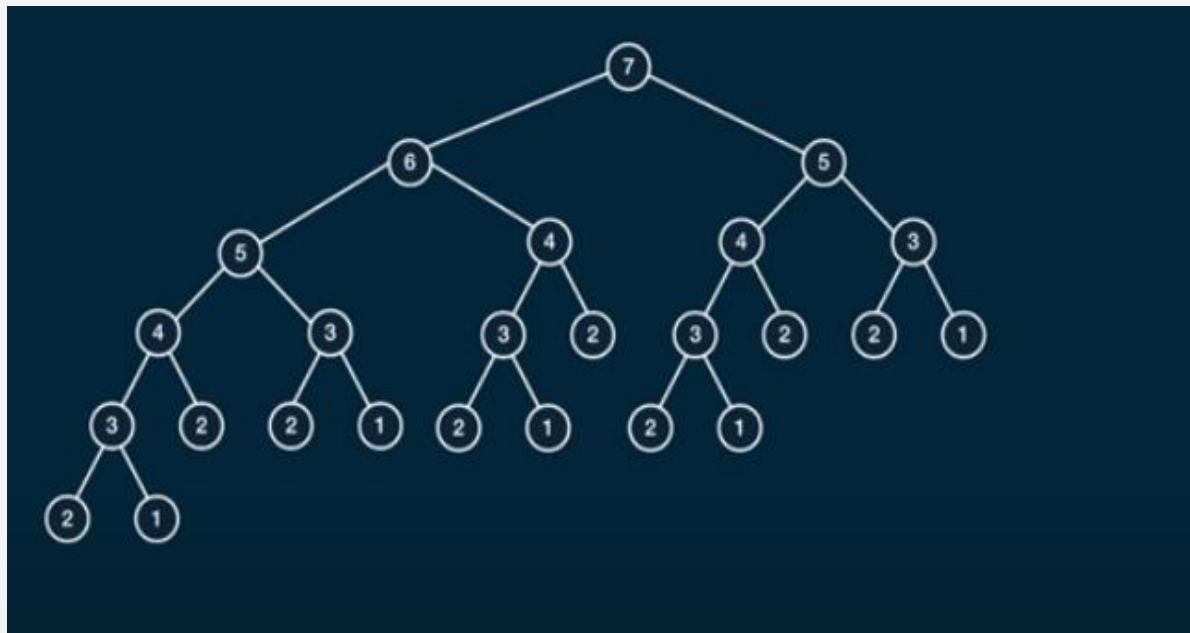
Hãy xây dựng thuật toán tính fib(n).



Có hai cách để làm:

- Memoization (Top down)
- Tabulation (Bottom up)

Dãy Fibonacci - Memoization



Biểu diễn đệ quy dưới dạng cây

Vấn đề xảy ra là gì ?

Độ phức tạp $O(2^n)$
Thời gian chạy lớn

Dãy Fibonacci - Memoization



Dạng tổng quát - Memoization

```
function (input, memo = {})  
  if input in memo -> return memo[input]  
  
  basecase  
  result_of_recursion = function(input_change, memo)  
  
  memo[input] = result_of_recursion  
  
  return memo[input]
```

Dãy Fibonacci - Tabulation

0	1	2	3	4	5	6
0	1	0	0	0	0	0

0	1	2	3	4	5	6
0	1	1	2	3	5	8

Biểu diễn Fibonacci dưới dạng bảng

Dạng tổng quát - Tabulation

```
# Tong quat - Tabulation
function (input)
    # Make an array depend on the input
    arr = []
    # Add base case
    arr[0] = somethings #this just an example
    # Make a loop start from the bottom up
    # Then add code which can solve the problem
    for i in range(bottom, len(arr))
        do_some_things -> result
    # Return the result
    return result
```

Các bước cơ bản

1

Xác định các trạng thái cơ bản của bài toán

2

Tìm ra mối quan hệ giữa các trạng thái đó

3

Xây dựng phép toán (giải thuật đệ quy)

4

Thêm biến nhớ, bằng phép toán, tính và lưu kết quả vào biến nhớ.

02

VÍ DỤ



Grid Traveller

Cho một bảng có n dòng và m cột. Có bao nhiêu cách để đi từ góc trái trên xuống góc phải dưới. Mỗi lần chỉ được di chuyển một ô qua phải hoặc xuống dưới.

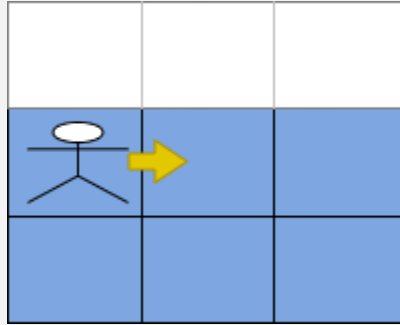
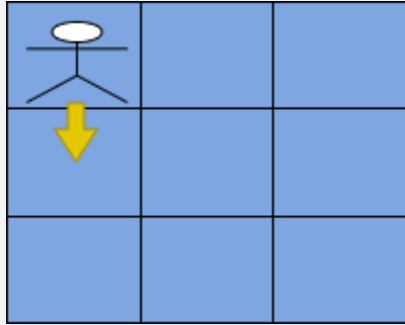
Hỏi có bao nhiêu cách để đi ?

s			s		
		E			E
s			s		
		E			E

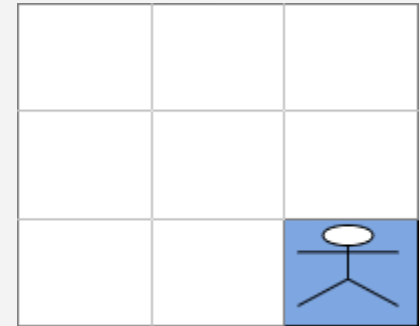
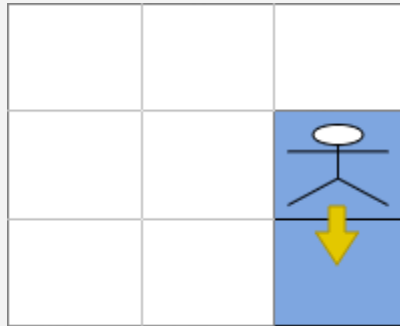
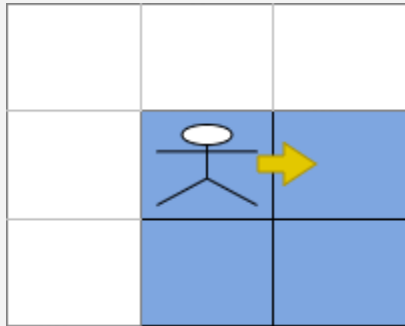
input: 2, 3

output: 3

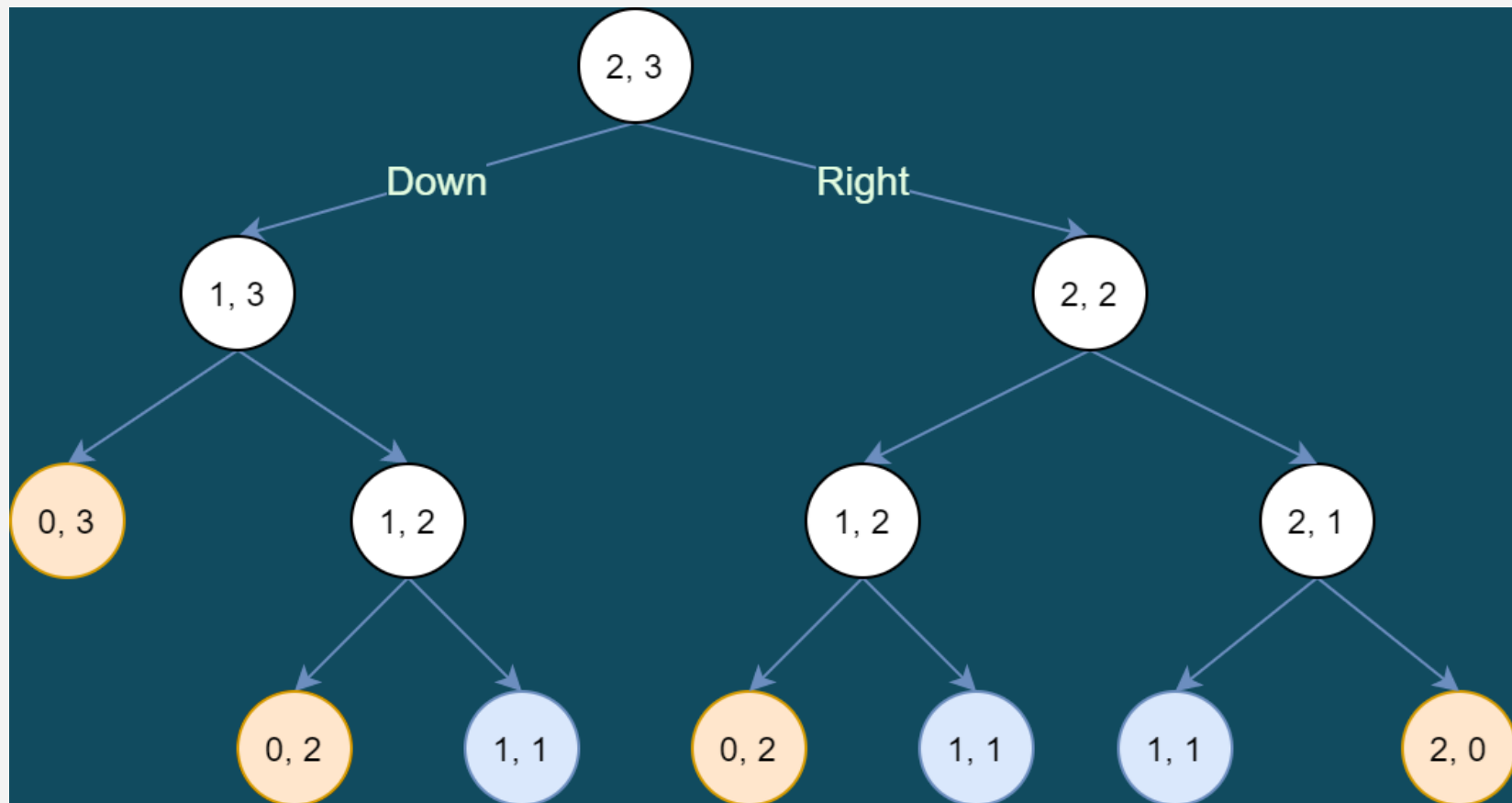
gridTraveler(3,3):



→ Solving GridTraveler(2,3)



Biểu diễn dưới dạng cây



Biểu diễn dưới dạng bảng

1	0	0
0	0	0
0	0	0

1	$1 (0 + 1)$	0
$1 (0 + 1)$	0	0
0	0	0

1	1	1
1	2	3
1	3	6

Gold mine problem

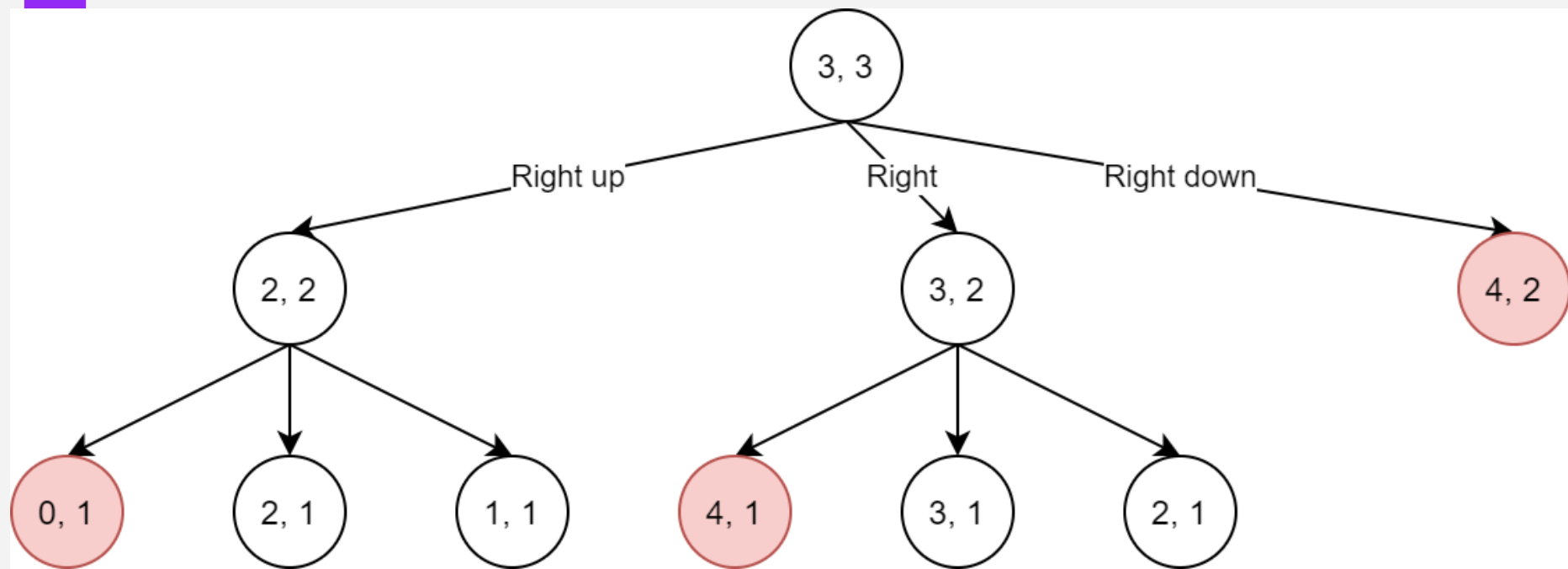
Cho một mảng gồm n dòng và m cột. Mỗi ô chứa một số nguyên ≥ 0 .

Mỗi lần chỉ được di chuyển 1 ô sang phải, phải trên, hoặc phải dưới.
Người đào mỏ có thể đào ở bất kì đâu trên cột đầu tiên.

Tìm ra cách đào mỏ có nhiều vàng nhất.



Biểu diễn dưới dạng cây



03

TỔNG KẾT





Quy hoạch động thường được áp dụng cho các
bài toán tối ưu.

Bài toán tối ưu: là bài toán tìm kiếm lời giải tốt nhất trong tất cả các lời giải khả thi.

Áp dụng nguyên lý tối ưu của Bellman: Trong một dãy tối ưu của các lựa chọn thì các dãy con của nó cũng tối ưu.

Ưu – Nhược Điểm



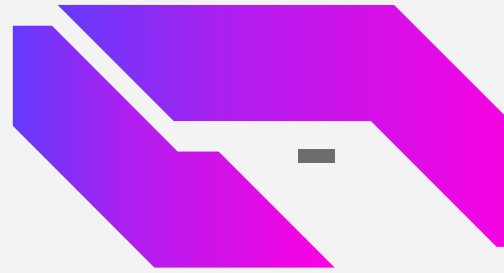
Ưu Điểm

- Tiết kiệm thời gian thực hiện, chi phí tính toán
- Thường áp dụng cho các bài toán tối ưu
- Giải quyết vấn đề một cách hệ thống
- Dễ dàng truy xuất kết quả bài toán con

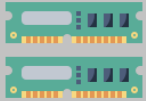
Nhược Điểm

- Tốn không gian bộ nhớ lưu trữ.
- Đòi hỏi sự phân tích tốt, dễ xảy ra sai sót.
- Khó khăn trong việc xây dựng công thức truy hồi.

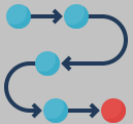
Điều kiện áp dụng quy hoạch động



Bài toán lớn **phân rã được thành nhiều bài toán con**, mà sự phối hợp lời giải của các bài toán con đó cho ta lời giải của bài toán lớn



Phải có đủ **không gian vật lí** lưu trữ lời giải (bộ nhớ, đĩa,...) để phối hợp lời giải của các bài toán con.



Quá trình từ bài toán cơ sở tìm ra lời giải bài toán ban đầu (qua các bài toán con) phải **qua hữu hạn bước**.

Divide and Conquer vs Dynamic Programming

Divide and Conquer

Các bài toán con độc lập với nhau

➡ Sẽ lặp lại việc giải quyết các bài toán con trùng nhau đó gây tiêu tốn không đáng có.

Dynamic Programming

Các bài toán con không độc lập (gối nhau nhau), có chung các bài toán con nhỏ hơn

➡ Sẽ giải quyết bài toán con 1 lần duy nhất, lưu kết quả lại để tránh việc phải lặp lại việc tính toán

TỔNG KẾT DYNAMIC PROGRAMMING



Sử dụng nguyên lí chia để trị



Các bài toán nhận diện:

- Các bài toán có được bằng việc tổ hợp các nghiệm của các bài toán con.
- Các bài toán tối ưu.



Là sự kết hợp của:

- Các bài toán con gộp nhau
- Cấu trúc con tối ưu

Bài Tập Về Nhà

BestSum

Cho một số tự nhiên K và một mảng `arr` có các phần tử dương.

Hãy sử dụng **Dynamic Programming (Quy hoạch động)** để tìm từ mảng `arr` các phần tử sao cho tổng của chúng bằng K và sử dụng ít phần tử nhất. Nếu có nhiều đáp án thì in ra bất kì đáp án thỏa điều kiện bài toán.

Ví dụ:

Input: `arr[] = {5, 10, 12, 13, 15, 18}`, $K = 30$

Output: `{12, 18}` Hoặc `{15, 15}`

The background of the image is a stylized world map divided into four quadrants by a vertical and a horizontal line. The top-left quadrant is red, the top-right is blue, the bottom-left is yellow, and the bottom-right is green. The word "Kahoot!" is written in a large, white, bold, sans-serif font across the center of the image, spanning across all four quadrants.

Kahoot!

Nguồn tham khảo

[https://edutechlearners.com/download/Introduction to algorithms -3rd%20Edition.pdf](https://edutechlearners.com/download/Introduction%20to%20algorithms%20-%203rd%20Edition.pdf)

<https://www.youtube.com/watch?v=oBt53YbR9Kk>

<https://www.slideserve.com/vinny/ch-ng-6-q-uy-ho-ch-ng>

<https://4fire.files.wordpress.com/2012/04/bai-giang-phan-tich-thiet-ke-va-danh-gia-thuat-toan.pdf>

<https://www.geeksforgeeks.org/fundamentals-of-algorithms/#AnalysisofAlgorithms>

The End

Cảm ơn thầy và các bạn đã lắng nghe!!!

