

## BÀI 4

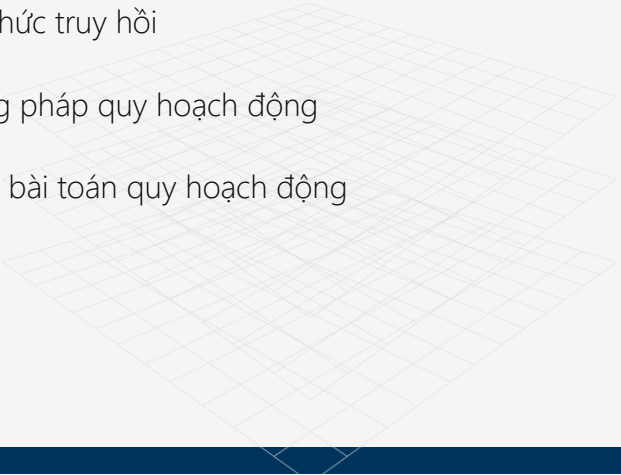
Email: anvanminh.hau@gmail.com

# Chiến lược quy hoạch động

1

## Chiến lược quy hoạch động – Dynamic programming

---

- 1 Giới thiệu quy hoạch động
  - 2 Công thức truy hồi
  - 3 Phương pháp quy hoạch động
  - 4 Một số bài toán quy hoạch động
- 

2

## Giới thiệu quy hoạch động

- Các thuật toán đệ quy có ưu điểm là “dễ cài đặt” nhưng đòi hỏi không gian nhớ và khối lượng tính toán lớn.
- Quy hoạch động là một kỹ thuật nhằm đơn giản hóa việc tính toán các công thức truy hồi bằng cách lưu trữ toàn bộ hay một phần kết quả tính toán tại mỗi bước với mục đích sử dụng lại.
- Bản chất của quy hoạch động là thay thế mô hình tính toán top – down bằng mô hình tính toán bottom – up.
- Từ programming không phải là lập trình mà là thuật ngữ các nhà toán học hay dùng để chỉ các bước chung trong việc giải quyết bài toán.

3

## Giới thiệu quy hoạch động

- Không có thuật toán quy hoạch động tổng quát.
- Quy hoạch động giúp giải quyết các bài toán tối ưu mang bản chất đệ quy.
- Đặc điểm chung của quy hoạch động:
  - Quy hoạch động bắt đầu từ việc giải tất cả các bài toán nhỏ nhất (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn cho tới khi giải được bài toán lớn nhất (bài toán ban đầu).
  - Quy hoạch động cần phải có bảng phương án.
  - Ý tưởng cơ bản của phương pháp quy hoạch động là tránh tính toán lại các bài toán con đã xét.

4

## Công thức truy hồi

Ví dụ: Phân tích số thành tổng

- Cho số tự nhiên  $n \leq 100$ . Có bao nhiêu cách phân tích số  $n$  thành tổng của dãy các số nguyên dương. Các cách phân tích là hoán vị của nhau thì chỉ tính là 1 cách.
- Chẳng hạn, với  $n = 5$  thì có 7 cách phân tích:

$$5 = 1 + 1 + 1 + 1 + 1$$

$$5 = 1 + 1 + 1 + 2$$

$$5 = 1 + 1 + 3$$

$$5 = 1 + 2 + 2$$

$$5 = 1 + 4$$

$$5 = 2 + 3$$

$$5 = 5$$

Sử dụng phương pháp sinh, hoặc quay lui?

5

## Phân tích số thành tổng

- Phân tích: Tìm công thức truy hồi
  - Gọi  $f(m, n)$  là số cách phân tích số  $n$  thành tổng các số nguyên dương  $\leq m$ .
  - Cách phân tích số  $n$  có thể chia làm 2 loại.
- Loại 1: Trong phân tích không có chứa số  $m$ 
  - Đây là số cách phân tích số  $n$  thành tổng các số nguyên dương  $< m$  hay chính là số cách phân tích số  $n$  thành tổng các số nguyên dương  $\leq m - 1$ , là  $f(m - 1, n)$ .
- Loại 2: Có ít nhất 1 số  $m$  trong phân tích
  - Nếu trong các cách phân tích ta bỏ đi số  $m$  thì được các cách phân tích số  $n - m$  thành tổng các số nguyên dương  $\leq m$ . Số các cách phân tích này là  $f(m, n - m)$ .

6

## Phân tích số thành tổng

- Trong trường hợp  $m > n$  thì chỉ có các phân tích loại 1. Còn khi  $m \leq n$  thì có cả phân tích loại 1 và loại 2.
- Ta có công thức tính  $f(m, n)$ :

$$f(m, n) = \begin{cases} f(m-1, n) & \text{khi } m > n \\ f(m-1, n) + f(m, n-m) & \text{khi } m \leq n \end{cases}$$

- Công thức này gọi là công thức truy hồi.
- Công thức này đưa việc tính  $f(m, n)$  về việc tính  $f(m', n')$  với dữ liệu nhỏ hơn.

7

## Phân tích số thành tổng

- Ví dụ: Với  $n = 5$  ta có
- $f(5, 5) = f(4, 5) + f(5, 0) = 7$
- Thứ tự tính: Trên xuống, trái sang phải.
- Khởi tạo dòng đầu với  $f(0, 0) = 1$ , các phần tử còn lại bằng 0.
- Các dòng còn lại sử dụng công thức truy hồi để tính.

f	0	1	2	3	4	5	n
0	1	0	0	0	0	0	
1	1	1	1	1	1	1	
2	1	1	2	2	3	3	
3	1	1	2	3	4	5	
4	1	1	2	3	5	6	
5	1	1	2	3	5	7	

m

8

## Phân tích số thành tổng

```
int analys01(){//Dùng mảng hai chiều
    for (int i = 1; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (j < i) {
                f[i][j] = f[i-1][j];
            }
            else {
                f[i][j] = f[i-1][j] + f[i][j-i];
            }
        }
    }
    return f[n][n];
}
```

9

## Phân tích số thành tổng

```
int analys02() { //Dùng 2 mảng một chiều
    for (int i = 1; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (j < i) {
                next[j] = cur[j];
            }
            else{
                next[j] = cur[j] + next[j-i];
            }
        }
        Copy: cur <- next;
    }
    return cur[n];
}
```

10

## Phân tích số thành tổng

```
int analys03() { //Dùng 1 mảng một chiều
    for (int i = 1; i <= m; i++){
        for (int j = i; j <= n; j++){
            if (j < i) {
                f[j] = f[j] + f[j - i];
            }
        }
    }
    return f[n];
}
```

11

## Phân tích số thành tổng

```
int f(int m, int n) { //Dùng kỹ thuật đệ quy (chậm)
    if (m == 0) {
        if (n == 0) {
            return 1;
        }
        else return 0;
    }
    else {
        if (m > n) return f(m - 1, n);
        else return f(m - 1, n) + f(m, n - m);
    }
}
```

12

## Phân tích số thành tổng

```
int f(int m, int n) { //Dùng kỹ thuật đệ quy có nhớ
    if (m == 0) {
        if (n == 0) {
            a[m][n] = 1;
        }
        else a[m][n] = 0;
    }
    else {
        if (m > n) a[m][n] = f(m - 1, n);
        else a[m][n] = f(m - 1, n) + f(m, n - m);
    }
    return a[m][n];
}
```

13

## Phương pháp quy hoạch động

- 1 Bài toán quy hoạch
- 2 Phương pháp quy hoạch động
- 3 Một số bài toán quy hoạch động

14

## Bài toán quy hoạch

- Bài toán quy hoạch là bài toán tối ưu gồm:
  - Một hàm  $f$  gọi là hàm mục tiêu (hàm đánh giá)
  - Các hàm  $g_1, g_2, \dots, g_n$  cho các giá trị logic gọi là hàm ràng buộc.
- Yêu cầu của bài toán là:
  - Tìm một cấu hình  $x$  thỏa mãn tất cả các ràng buộc:
 
$$g_i(x) = \text{true} \ (\forall i \in [1, n]).$$
  - Và  $x$  là cấu hình tốt nhất..
- Các bài toán quy hoạch động rất phong phú và ứng dụng nhiều trong thực tế.
- Tuy nhiên, có nhiều bài toán quy hoạch động là không giải được, hoặc chưa giải được.

15

## Phương pháp quy hoạch động

- Dùng để giải các bài toán tối ưu có bản chất đệ quy.
- Chiến lược chia để trị đóng vai trò chủ đạo trong việc thiết kế giải thuật đệ quy.
- Phương pháp quy hoạch động cũng thể hiện rõ chiến lược này: Giải quyết tất cả các bài toán con và lưu trữ kết quả của chúng với mục đích sử dụng lại theo một sự phối hợp nào đó để giải quyết bài toán tổng quát hơn.
- Đệ quy: Bắt đầu từ bài toán tổng quát, phân rã thành các bài toán con, giải quyết các bài toán con, tiếp tục phân rã các bài toán con nếu chưa có lời giải trực tiếp.
- Quy hoạch động: Bắt đầu từ việc giải quyết các bài toán nhỏ nhất (có lời giải trực tiếp), từng bước giải quyết các bài toán lớn hơn, cho tới khi giải được bài toán ban đầu.

16



## Phương pháp quy hoạch động

- Ví dụ: Bài toán dãy số fibonacci.

```
//Cách 1: Kỹ thuật đệ quy
int f(int n) {
    if (n <= 2) {
        return 1;
    }
    else {
        return 0;
    }
}
```

```
//Cách 2: ?
void f(int n, int *a) {
    a[1] = a[2] = 1;
    for (int i = 3; i <= n; i++)
    {
        a[i] = a[i-1] + a[i-2];
    }
}
```

17

## Phương pháp quy hoạch động

- Để áp dụng quy hoạch động cần xem xét 2 yếu tố:
  - Bài toán lớn phải phân rã được thành các bài toán con, mà sự phối hợp lời giải của các bài toán con cho lời giải của bài toán lớn.
  - Có đủ không gian bộ nhớ để lưu trữ tất cả các lời giải để phối hợp chúng hay không?
- Các khái niệm
  - Bài toán giải theo phương pháp QHĐ gọi là bài toán QHĐ.
  - Công thức phối hợp nghiệm gọi là công thức truy hồi.
  - Tập các bài toán nhỏ nhất gọi là bài toán cơ sở.
  - Không gian lưu trữ lời giải của các bài toán gọi là bảng phương án.

18

## Phương pháp quy hoạch động

- Các bước để giải bài toán tối ưu bằng qui hoạch động:
  - Đưa ra cách tính nghiệm của các bài toán con đơn giản.
  - Tìm công thức xây dựng nghiệm của bài toán thông qua nghiệm của các bài toán con (công thức truy hồi).
  - Thiết kế bảng phương án để lưu nghiệm của các bài toán (tổ chức dữ liệu).
  - Tính nghiệm của các bài toán từ nhỏ đến lớn (xây dựng bảng phương án).
  - Truy vết tìm nghiệm của bài toán từ bảng phương án kết quả.

19

## Một số bài toán quy hoạch động

- Tìm số fibonacci thứ  $n$
- Dãy con đơn điệu tăng dài nhất
- Bài toán cái túi

20

## Tìm số fibonacci thứ n

- Cơ sở quy hoạch động
  - Các bài toán con nhỏ nhất là: Tính  $F[1] = 1$  và tính  $F[2] = 1$ .
- Công thức truy hồi
  - Tính  $F[i]$  là số fibonacci thứ  $i$  ( $i \geq 3$ ) sau khi đã tính được  $F[1]$ ,  $F[2]$ , ...,  $F[i-1]$ .
  - $F[i] = F[i-1] + F[i-2]$  với ( $i \geq 3$ )
  - $$F[n] = \begin{cases} 1 & \text{nếu } n < 3 \\ F[n-1] + F[n-2] & \text{nếu } n \geq 3 \end{cases}$$
- Bảng phương án
  - Mảng  $F[1...n]$  để lưu các số tìm được,  $F[n]$  là số fibonacci thứ  $n$  cần tìm.

21

## Tìm số fibonacci thứ n

- Thuật toán quy hoạch động

```
//Cách 2: ?
f(n) {
    F[1] = F[2] = 1;
    for (int i = 3; i <= n; i++)
    {
        F[i] = F[i-1] + F[i-2];
    }
    return F[n];
}
```

- Có thể sử dụng 3 biến thay cho mảng F

22

## Dãy con đơn điệu tăng dài nhất

- Cho một dãy gồm  $n$  số nguyên  $a = \{a[1], a[2], \dots, a[n]\}$ 
  - Một dãy con của dãy  $a$  là một cách chọn ra trong  $a$  một số phần tử giữ nguyên thứ tự (dãy  $a$  có 2 mũ  $n$  dãy con).
  - Yêu cầu: Tìm dãy con đơn điệu tăng của  $a$  có độ dài lớn nhất.
  - Ví dụ:  $a = \{1, 2, 3, 4, 9, 10, 5, 6, 7\}$   
Dãy con đơn điệu tăng dài nhất là dãy  $\{1, 2, 3, 4, 5, 6, 7\}$
- Phương pháp giải
  - Bổ sung thêm vào dãy  $a$  2 phần tử  $a[0] = \text{âm vô cùng}$  và  $a[n+1] = \text{dương vô cùng}$ .
  - Dãy con đơn điệu tăng dài nhất bắt đầu từ  $a[0]$  và kết thúc ở  $a[n+1]$ .
  - Với mọi  $i = 0, \dots, n + 1$ , tính  $L[i] = \text{độ dài dãy con đơn điệu tăng dài nhất bắt đầu tại } a[i]$

23

## Dãy con đơn điệu tăng dài nhất (dãy con)

- Cơ sở quy hoạch động
  - Bài toán con nhỏ nhất là tính  $L[n + 1]$  là độ dài dãy con bắt đầu tại  $a[n + 1] = \text{dương vô cùng}$ .
  - $L[n + 1] = 1$ .
- Công thức truy hồi
  - Tính  $L[i]$  là độ dài dãy con bắt đầu tại  $a[i]$ , khi đó đã tính được  $L[i+1], \dots, L[n+1]$ .
  - Dãy con bắt đầu tại  $a[i]$  được thành lập bằng cách ghép  $a[i]$  vào đầu 1 trong số những dãy con bắt đầu tại  $a[j]$  đứng sau  $a[i]$  mà  $a[j] > a[i]$  ( $j = i + 1, \dots, n + 1$ ).
  - Vậy  $L[i] = \max L[j] + 1$  với  $i < j \leq n + 1$  và  $a[i] < a[j]$ .
  - Với  $L[j]$  được chọn ta gọi  $j$  được chọn là  $j_{\max}$ .

24

## Dãy con đơn điệu tăng dài nhất (dãy con)

- Bảng phương án

- Sử dụng mảng  $T[0...n]$  để lưu lại các phương án tìm được ứng với mỗi  $L[i]$ .
- Tại mỗi bước tính  $L[i]$ , gán  $T[i] = j_{\max}$ , nghĩa là dãy con bắt đầu tại  $a[i]$  có phần tử thứ 2 là  $a[j_{\max}]$ .

- Ví dụ:

i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	-	5	2	3	4	9	10	5	6	7	8	+
L[i]	9	5	8	7	6	3	2	5	4	3	2	1
T[i]	2	8	3	4	7	6	11	8	9	10	11	

25

## Dãy con đơn điệu tăng dài nhất (dãy con)

- Truy vết

- Bắt đầu từ  $T[0]$ ,  $k_1 = T[0] = 1$ .
- $a[k_1] = a[T[0]]$  là phần tử đầu tiên của dãy con,  $k_2 = T[k_1] = 2$ .
- $a[k_2] = a[T[k_1]]$  là phần tử thứ 2 của dãy con,  $k_3 = T[k_2] = 3$ .
- $a[k_3] = a[T[k_2]]$  là phần tử thứ 3 của dãy con,  $k_4 = T[k_3] = 4$ .
- ...
- Cho đến khi tìm được  $k_m = n + 1$ ,  $m$  là độ dài dãy con đơn điệu tăng dài nhất tìm được

- Ví dụ

- $a[T[0]] = a[2] = 2$ ;  $a[T[T[0]]] = a[T[2]] = a[3] = 3$ ;  $a[T[T[T[0]]]] = a[T[3]] = a[4] = 4$ ;
- $a[T[T[T[T[0]]]]] = a[T[4]] = a[7] = 5$ ; ...

26

## Dãy con đơn điệu tăng dài nhất (dãy con)

- Thuật toán quy hoạch động

```
algorithm() {
    a[0] = -2147483648; a[n + 1] = 2147483647;
    L[n + 1] = 1;
    for (int i = n; i >= 0; i--) {
        jmax = n + 1;
        for (int j = i + 1; j <= n + 1; j++)
            if (a[j] > a[i] && L[j] > L[jmax])
                jmax = j;
        L[i] = L[jmax] + 1;
        T[i] = jmax;
    }
}
```

27

## Dãy con đơn điệu tăng dài nhất (dãy con)

- Truy vết trên bảng phương án T tìm kết quả

```
result() {
    k = T[0];
    cout<<"Day con: ";
    while (k != n + 1) {
        cout<<"a["<<k<<"]: "<<a[k]<<" ";
        k = T[k];
    }
}
```

28

## Bài toán cái túi

- Bài toán

- Trong siêu thị có  $n$  gói hàng  $\{1, 2, \dots, n\}$ , gói hàng thứ  $i$  có trọng lượng là  $w[i]$  và giá trị  $v[i]$ .
- Ban đêm một tên trộm đột nhập vào siêu thị, cậu ấy mang theo một cái túi có thể mang được trọng lượng tối đa là  $m$ .
- Hỏi cậu ấy sẽ lấy đi những gói hàng nào để đạt được tổng giá trị lớn nhất.

- Phương pháp

- Nếu gọi  $f(i, j)$  là giá trị lớn nhất có thể nhận được bằng cách lấy trong số các gói  $\{1, 2, \dots, i\}$  với giới hạn trọng lượng là  $j$ .
- Giá trị lớn nhất khi lấy trong số các gói  $\{1, 2, \dots, n\}$  với giới hạn trọng lượng  $m$  sẽ là  $f(n, m)$ .

29

## Bài toán cái túi

- Công thức truy hồi tính  $f(i, j)$

- Với giới hạn trọng lượng  $j$ , việc lấy tối ưu trong số các gói  $\{1, 2, \dots, i-1, i\}$  để có giá trị lớn nhất có hai khả năng.
- KN1: Nếu không lấy gói thứ  $i$  thì  $f(i, j)$  là giá trị lớn nhất có thể có bằng cách lấy trong số các gói  $\{1, 2, \dots, i-1\}$ . Nghĩa là  $f(i, j) = f(i-1, j)$ .
- KN2: Nếu lấy gói thứ  $i$  ( $w[i] \leq j$ ) thì  $f(i, j) = v[i] + f(i-1, j-w[i])$ .

- Cơ sở quy hoạch động

- $f(0, j) = 0$  là giá trị lớn nhất có thể khi lấy trong số 0 gói hàng.

30

## Bài toán cái túi

- Bảng phương án

- Bảng F gồm  $n + 1$  dòng,  $m + 1$  cột.
- Dòng đầu ( $i = 0$ ) với  $F[0][j] = 0$  ( $j = 0 \rightarrow m$ ).
- Dùng dòng  $i$ , tính dòng  $i + 1$ , với  $i = 0 \rightarrow n - 1$ .

F	0	1	2	...	m
0	0	0	0	0	0
1					
2					
...	...	...	...	...	...
n					result

31

## Bài toán cái túi

- Truy vết

- $F[n][m]$  là giá trị lớn nhất lấy được khi lấy trong  $n$  gói hàng với giới hạn trọng lượng là  $m$ .
- Nếu  $F[n][m] = F[n - 1][m] \rightarrow$  không lấy gói thứ  $n$ , truy tiếp  $F[n - 1][m]$ .
- Nếu  $F[n][m] \neq F[n - 1][m]$  có lấy gói thứ  $n$ , truy tiếp  $F[n - 1][m - w[n]]$ .
- Truy vết tiếp lên đến hàng 0.

32



## Bài toán cái túi

- Thuật toán

```
algorithm() {
    for (i = 0; i <= m; i++) { F[0][j] = 0; }
    for (i = 1; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            F[i][j] = F[i - 1][j]; //Không lay goi thu i
            temp = F[i - 1][j - w[i]] + v[i];
            if (w[i] <= j && F[i][j] < temp) //Lay goi thu i
                F[i][j] = temp;
        }
    }
}
```

33

## Bài toán cái túi

- Truy vết tìm kết quả

```
result() {
    cout<<"Max value: " << F[n][m] << endl;
    i = n; j = m;
    while (i != 0) {
        if (F[i][j] != F[i - 1][j]) {
            cout<<i<<" ";
            j = j - w[i]; //Lay goi thu i
        }
        i--;
    }
}
```

34

## Bài toán cắm hoa

- Có  $n$  lọ hoa sắp thành một hàng và  $k$  bó hoa được đánh số thứ tự từ nhỏ đến lớn.
- Cần cắm  $k$  bó hoa vào  $n$  lọ sao cho bó hoa có số thứ tự nhỏ phải đứng trước bó hoa có số thứ tự lớn.
- Giá trị thẩm mỹ tương ứng khi cắm bó hoa  $i$  vào lọ thứ  $j$  là  $v(i, j)$ .
- Hãy tìm 1 cách cắm  $k$  bó hoa sao cho tổng giá trị thẩm mỹ là lớn nhất.
- Chú ý: mỗi bó hoa chỉ được cắm vào 1 lọ và mỗi lọ cũng chỉ cắm được 1 bó hoa.

35

## Bài toán cắm hoa

- Phân tích
  - $L(i, j)$ : tổng giá trị thẩm mỹ lớn nhất khi xét đến bó hoa  $i$  và lọ  $j$ .
  - Tính  $L(i, j)$ :
  - Nếu  $i = 0$  hoặc  $j = 0$  thì  $L(i, j) = 0$ ;
  - Nếu  $i = j$  thì chỉ có 1 cách cắm, do đó:  $L(i, i) = v(1, 1) + v(2, 2) + \dots + v(i, i)$ .
  - Nếu  $i > j$ : Không có cách cắm hợp lý  $L(i, j) = 0$ ;
  - Nếu  $i < j$  thì có 2 trường hợp xảy ra:
    - TH1: Cắm bó hoa  $i$  vào lọ  $j$ , tổng giá trị thẩm mỹ là  $L(i - 1, j - 1) + v(i, j)$  – bằng tổng giá trị trước khi cắm và giá trị thẩm mỹ khi cắm bó  $i$  vào lọ  $j$ .
    - TH2: Không cắm bó hoa  $i$  vào lọ  $j$ , giá trị thẩm mỹ của cách cắm là như cũ:  $L(i, j - 1)$ .

36

