CHAPTER 04

CHIẾN LƯỢC QUY HOẠCH ĐỘNG
(Dynamic Programming)

Design by Minh An

Email: anvanminh.haui@gmail.com

Chiến lược quy hoạch động

- **Giới thiệu phương pháp**
- **Công thức truy hồi**
- **Phương pháp quy hoạch động**
- **Một số bài toán quy hoạch động**

Giới thiệu phương pháp

- **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.**

Giới thiệu phương pháp

- **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.

Công thức truy hồi

- **Ví dụ: Phân tích số thành tổng**
  - Cho số tự nhiên n ≤ 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:
    1. 5 = 1 + 1 + 1 + 1 + 1
    2. 5 = 1 + 1 + 1 + 2
    3. 5 = 1 + 1 + 3
    4. 5 = 1 + 2 + 2
    5. 5 = 1 + 4
    6. 5 = 2 + 3
    7. 5 = 5
  - Sử dụng phương pháp sinh, hoặc quay lui?

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 ≤ 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 ≤ 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 ≤ m. Số các cách phân tích này là f(m, n – m).

Phân tích số thành tổng

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

$$f(m,n) = \begin{cases} f(m-1, \quad n)\ khi\ m > n \\ f(m-1, \quad n) + f(m, n-m)\ khi\ m \le n \end{cases}$$

- **Công thức trên đây 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.**

Phân tích số thành tổng

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

$$f(m,n) = \begin{cases} f(m-1, \quad n)\ khi\ m > n \\ f(m-1, \quad n) + f(m, n-m)\ khi\ m \le n \end{cases}$$

- **Công thức trên đây 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.**

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.**

f	0	1	2	3	4	5
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

n

m

- **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.**

Algorithm 01: Dùng mảng hai chiều

```
int analys01(){
    for (int i=1; i<=n; 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];
}
```

Tốn bộ nhớ vì dùng mảng hai chiều

Algorithm 02: Dùng 2 mảng một chiều

```
int analys02(){
    for (int i=1; i<=n; 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; //gán next cho cur
    }
    return cur[n];
}
```

Đỡ tốn bộ nhớ hơn nhưng mất thời gian copy

Algorithm 03: Dùng 1 mảng một chiều

```
int analys03(){
    for (int i=1; i<=n; i++){
        for (int j=i; j<=n; j++){
            f[j] = f[j] + f[j-i];
        }
    }
    return f[n];
}
```

Algorithm 04: Kỹ thuật đệ quy

```
int f(int m, int n){
    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);
    }
}
```

Phương pháp này chậm vì gọi đệ quy nhiều lần

Algorithm 05: Đệ quy có nhớ

```
int f(int m, int n){
    if (m == 0){
        if (n == 0) F[m][n] = 1;
        else F[m][n] = 0;
    }
    else{
        if (m > n)
            F[m][n] = f(m-1, n);
        else
            F[m][n] = f(m-1, n) + f(m, n-m);
    }
    return F[m][n];
}
```

Phương pháp quy hoạch động

- **Bài toán quy hoạch**
- **Phương pháp quy hoạch động**
- **Một số bài toán quy hoạch động**

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, …, 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.**

Phương pháp quy hoạch động

- **Dùng để giải các bài toán tối ưu có bản chất đệ quy.**
- **Chiến lược chia để trị thường 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.**

Phương pháp quy hoạch động

- Ví dụ: bài toán dãy số fibonaci

Cách 1: Đệ qui

```
int fibo(int n){
    if (n <= 2)
        return 1;
    else
        return fibo(n-1) + fibo(n-2);
}
```

Cách 2: ???

```
void fibo(int n, int *a){
    a[1] = a[2] = 1;
    for (int i=3; i<=n; i++)
        a[i] = a[i-1] + a[i-2];
}
```

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.

Phương pháp quy hoạch động

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

Một số bài toán quy hoạch động

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

Tìm số fibonaci 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 fibonaci thứ i (i ≥ 3) sau khi đã tính được các số F[1], ..., F[i-1].
- F[i] = F[i-1] + F[i-2] (i ≥ 3)

$$F[n] = \begin{cases} 1 \text{ nếu } n \leq 2 \\ F[n-1] + F[n-2] \text{ nếu } n > 2 \end{cases}$$

Bảng phương án:

- Mảng F[1...n] để lưu các số tìm được, F[n] là số fibonaci thứ n cần tìm.

Tìm số fibonaci thứ n

Thuật toán quy hoạch động

```
fibo(n){
        F[1] = 1;  F[2] = 1;
        for (i : 3 -> n){
                F[i] = F[i-1] + F[i-2];
        }
        return F[n];
}
```

Có thể sử dụng 3 biến "đuổi nhau" thay cho mảng F.

Dãy con đơn điệu tăng dài nhất

Cho một dãy số gồm n số nguyên: a = {a[1], a[2], ..., 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^n dãy con}.
- Yêu cầu: Tìm dãy con đơn điệu tăng của a có độ dài lớn nhất.
- Ví dụ:

  Dãy a = {1, 2, 3, 4, 9, 10, 5, 6, 7}

  Dãy con đơn điệu tăng dài nhất là {1, 2, 3, 4, 5, 6, 7}

Phương pháp giải:

- Bổ sung thêm vào dãy a hai phần tử a[0] = $-\infty$ và a[n+1] = $+\infty$
- Khi đó dãy con đơn điệu tăng dài nhất sẽ bắt đầu từ a0 và kết thúc ở a[n+1].
- Với $\forall i \in [0, n+1]$, ta tính L[i] = độ dài dãy con đơn điệu tăng dài nhất bắt đầu tại a[i].

Dãy con đơn điệu tăng dài nhất

Từ đây gọi "dãy con đơn điệu tăng dài nhất" là "dãy con".

Cơ sở quy hoạch động

- Bài toán con nhỏ nhất là tính L[n+1]. Với L[n+1] là độ dài dãy con bắt đầu tại a[n+1] = +∞.
- L[n+1] = 1.

Tìm công thức truy hồi:

- Tính L[i] độ dài dãy con bắt đầu tại a[i].
- Khi đó đã tính được L[i+1], ..., 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, ..., n+1).
- Vậy: L[i] = max L[j] + 1 với i < j ≤ n+1 và a[i] < a[j]
- Với L[j] được chọn ta gọi j được chọn là jmax

Dãy con đơn điệu tăng dài nhất

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] = jmax, nghĩa là dãy con bắt đầu tại a[i] có phần tử thứ 2 là a[jmax].

Truy vết:

- Bắt đầu từ T[0], k1 = T[0] = 1
- a[k1] = a[T[0]] là phần tử đầu tiên của dãy con, k2 = T[k1] = 2
- a[k2] = a[T[k1]] là phần tử thứ 2, k3 = T[k2] = 3
- a[k3] = a[T[k2]] là phần tử thứ 3, k4 = T[k3] = 4
- ...
- Cho đến khi km = n + 1; (m là độ dài dãy con đơn điệu tăng dài nhất tìm được)

Dãy con đơn điệu tăng dài nhất

Ví dụ:

i	0	1	2	3	4	5	6	7	8	9	10	11
a[i]	$-\infty$	5	2	3	4	9	10	5	6	7	8	$+\infty$
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	

Dãy con:

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;

Dãy con đơn điệu tăng dài nhất

Thuật toán quy hoạch động

```
dp_algorithm(){
        a[0] = -2147483648; a[n+1] = 2147483647;
        L[n+1] = 1;
        for (i = n; i >= 0; i--){
                jmax = n + 1;
                for (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;
        }
}
```

Dãy con đơn điệu tăng dài nhất

Truy vết trên bảng phương án T tìm kết quả

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

Bài toán cái túi

Bài toán

- Trong siêu thị có n gói hàng {1, 2, ..., n}, gói thứ i có trọng lượng là w[i] và giá trị là v[i].
- Ban đêm, một tên trộm đột nhập vào siêu thị, hắn mang theo một cái túi có thể mang được trọng lượng tối đa là m.
- Hỏi tên trộm sẽ lấy đi những gói hàng nào để 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, ..., i} với giới hạn trọng lượng là j.
- Giá trị lớn nhất khi lấy trong số các gói {1, 2, ..., n} với giới hạn trọng lượng m sẽ là f(n, m).

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, ..., i – 1, i} để có giá trị lớn nhất có hai khả năng.
- 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, ..., i – 1}.

  Nghĩa là: f(i, j) = f(i – 1, j)
- Nếu lấy gói thứ i (w[i] ≤ 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.

Bài toán cái túi

- **Bảng phương án**
  - Bảng F gồm n + 1 dòng, m + 1 cột.
  - Dòng đầu F[0][j] = 0 (j = 0 -> m
  - Dùng dòng 0, tính dòng 1, dòng 1 tính dòng 2, v.v...

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

Bài toán cái túi

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

Bài toán cái túi

Thuật toán quy hoạch động

```
dp_algorithm(){
    for (j = 0; j <= m; j++) F[0][j] = 0;
    for (i = 1; i <= n; i++) {
        for (j = 0; j <= m; j++) {
            F[i][j] = F[i-1][j]; //Không lấy gói thứ i
            temp = F[i-1][j-w[i]] + v[i];
            if (w[i] <= j && F[i][j] < temp) //Lấy gói thứ i
                F[i][j] = temp;
        }
    }
}
```

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]; //Lấy gói thứ i
        }
        i--;
    }
}
```

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.**

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):
  - i = 0 hoặc j = 0 thì L(i, j) = 0
  - Nếu i = j. Chỉ có một cách cắm, do đó:

    L(i, i) = v[1, 1] + v[2, 2] + ... + v[i, i]
  - Nếu i > j: Không có cách cắm hợp lý L(i, j) = 0;
  - Nếu i < j: Có 2 trường hợp xảy ra:
    - 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 cộng với giá trị thẩm mỹ khi cắm bó hoa i vào lọ j)
    - Không cắm bó hoa i vào lọ j, giá trị thẩm mỹ của cách cắm là như cũ : L(i, j - 1)