BÀI 4. CHIẾN LƯỢC QUY HOẠCH ĐỘNG

(Dynamic Programming Strategy)

1. Khái niệm

Chiến lược quy hoạch động là một kỹ thuật giải quyết vấn đề bằng cách chia nhỏ vấn đề gốc thành các bài toán nhỏ hơn, giải quyết các bài toán nhỏ này một cách độc lập và lưu trữ kết quả của các bài toán nhỏ đó để sử dụng lại trong tương lai. Kỹ thuật này thường được áp dụng cho các bài toán tối ưu hoá trong đó có sự tương quan giữa các bước con.

Chiến lược quy hoạch động thường được gọi là "đệ quy có nhớ" (memoization) bởi vì nó thường sử dụng kỹ thuật đệ quy để giải quyết các bài toán con nhỏ, và lưu trữ kết quả của các bài toán con đã được giải quyết để sử dụng lại trong tương lai.

Khi sử dụng kỹ thuật đệ quy, có thể xảy ra trường hợp một số bài toán con được tính toán nhiều lần. Điều này là không hiệu quả và có thể dẫn đến việc mất hiệu suất. Tuy nhiên, với kỹ thuật "đệ quy có nhớ", mỗi khi tính toán một bài toán con, chúng ta lưu trữ kết quả của nó. Khi cần tính toán lại bài toán con đó, chúng ta sẽ kiểm tra xem đã có kết quả được tính toán trước đó hay chưa. Nếu đã có, chúng ta sẽ lấy kết quả từ bộ nhớ thay vì tính toán lại. Điều này giúp tăng hiệu suất tính toán và giảm độ phức tạp của thuật toán.

Vì vậy, chiến lược quy hoạch động thường được gọi là "đệ quy có nhớ" do tính chất lưu trữ kết quả của các bài toán con đã được tính toán trước đó.

Ví dụ: xét bài toán tìm số Fibonacci thứ n: Dãy số Fibonacci là một dãy số vô hạn bắt đầu bằng hai phần tử đầu tiên là 1, các phần tử tiếp theo được tính bằng tổng của hai phần tử trước nó:

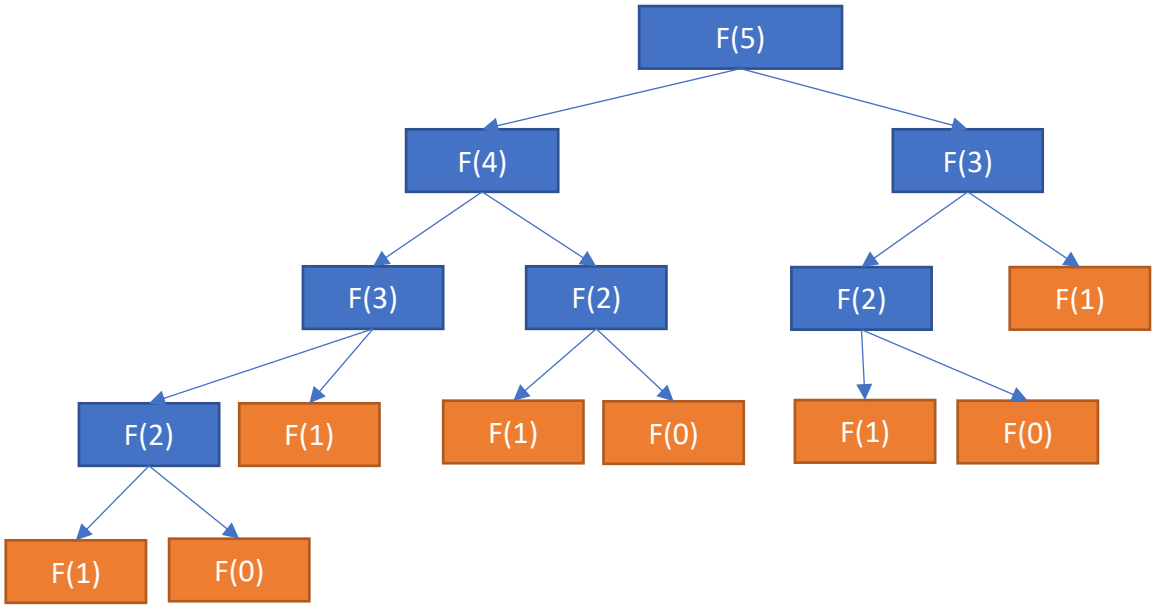
- $F(0) = F(1) = 1$;
- $F(n) = F(n-1) + F(n-2)$ với mọi $n \geq 2$

Viết chương trình để tính số Fibonacci thứ n (tức F(n)).

Cách tiếp cận đệ quy có thể viết như sau:

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

Giả sử ta cần tìm số Fibonacci thứ n = 5; các tiến trình đệ quy sinh ra có thể như sau:



Hình 4.1. Các tiến trình đệ quy sinh ra khi tính F(5).

Trong hình 4.1, có thể thấy rất nhiều tiến trình đệ quy giống nhau. Điều này có thể làm giảm hiệu suất của thuật toán.

Một ý tưởng tốt là có thể lưu trữ các giá trị F[i] đã tính được để sử dụng khi cần. Với quy trình trính F[5], ta chỉ cần tính và lưu trữ lần lượt F[0], F[1], F[2], F[3], F[4], F[5], thay vì phải tạo ra rất nhiều tiến trình như cách tiếp cận đệ quy. Ý tưởng này là cách tiếp cận quy hoạch động. Khi đó, giải thuật có thể viết lại như sau:

```
int fibonacci(int n)
{
    if (n <= 1)
          return 1;
    int fib[n + 1];  fib[0] =  fib[1] = 1;

    for (int i = 2; i <= n; i++)
        fib[i] = fib[i - 1] + fib[i - 2];

    return fib[n];
}
```

Cách làm này sẽ cho một giải thuật nhanh hơn. Tuy nhiên, nó cũng tốn một lượng bộ nhứ khi sử dụng mảng để lưu trữ các kết quả trung gian.z

2. Nguyên lý hoạt động

Nguyên lý hoạt động của quy hoạch động là phân tích vấn đề gốc thành các bài toán nhỏ hơn và giải quyết chúng một cách độc lập, sau đó sử dụng kết quả của các bài toán con đã giải quyết để xây dựng lời giải cho vấn đề gốc. Quy hoạch động thường được sử dụng để giải quyết các bài toán tối ưu hoá có cấu trúc con tối ưu.

Cụ thể, nguyên lý hoạt động của quy hoạch động bao gồm các bước sau:

[1]. Phân tích cấu trúc tối ưu của vấn đề: Xác định cách tối ưu hóa một vấn đề lớn thành các bài toán con nhỏ hơn, thường được biểu diễn bằng một cấu trúc đệ quy hoặc lặp.

[2]. Lưu trữ kết quả của các bài toán con: Thực hiện tính toán và lưu trữ kết quả của các bài toán con một cách độc lập, thường sử dụng một cấu trúc dữ liệu như một bảng hoặc một mảng.

[3]. Kết hợp các lời giải từ các bài toán con: Sử dụng kết quả từ các bài toán con để xây dựng lời giải cho vấn đề gốc.

[4]. Tính toán lời giải cuối cùng: Dựa vào các kết quả đã lưu trữ, xây dựng lời giải cho vấn đề gốc.

Quy hoạch động giải quyết vấn đề bằng cách giải quyết một loạt các bài toán con nhỏ hơn, lưu trữ kết quả của chúng và sử dụng lại kết quả đã tính toán trong quá trình giải quyết vấn đề gốc. Điều này giúp tránh trùng lặp tính toán và giảm độ phức tạp của thuật toán.

Trong bài toán tìm số Fibonacci thứ n, nguyên lý hoạt động của quy hoạch động thể hiện rõ ở các bước sau:

[1]. Phân tích cấu trúc tối ưu của vấn đề:

- Vấn đề gốc là tính toán số Fibonacci thứ n.
- Cấu trúc tối ưu của vấn đề này là tính toán số Fibonacci thứ n dựa trên hai số Fibonacci trước đó (n-1 và n-2).

[2]. Lưu trữ kết quả của các bài toán con:

- Sử dụng một mảng (hoặc vector) để lưu trữ kết quả của các số Fibonacci từ 0 đến n.
- Mỗi lần tính toán số Fibonacci mới, chúng ta lưu trữ kết quả vào mảng để sử dụng lại trong tương lai.

[3]. Kết hợp các lời giải từ các bài toán con:

- Mỗi lời giải cho số Fibonacci thứ n sẽ dựa trên kết quả đã tính toán của hai số Fibonacci trước đó (n-1 và n-2).

[4. Tính toán lời giải cuối cùng:

- Dựa vào các kết quả đã lưu trữ trong mảng, chúng ta xây dựng lời giải cho số Fibonacci thứ n.

3. Đặc điểm của chiến lược quy hoạch động

Các đặc điểm chính của quy hoạch động là:

[1]. Phân tách vấn đề thành các bài toán con nhỏ hơn:

- Quy hoạch động phân tách vấn đề gốc thành các bài toán con nhỏ hơn, thường được biểu diễn bằng một cấu trúc đệ quy hoặc lặp.

[2]. Lưu trữ kết quả của các bài toán con:

- Kỹ thuật này thường sử dụng một cấu trúc dữ liệu như một bảng hoặc một mảng để lưu trữ kết quả của các bài toán con một cách độc lập.

[3]. Tránh trùng lặp tính toán:

- Quy hoạch động tránh việc tính toán lại các kết quả đã biết bằng cách lưu trữ chúng trong một bảng hoặc một mảng và sử dụng lại kết quả đã tính toán trong quá trình giải quyết vấn đề gốc.

[4]. Thực hiện tính toán theo chiều dọc hoặc chiều ngang:

- Có hai cách tiếp cận chính trong quy hoạch động:

- Tiếp cận dọc (Top-down): Bắt đầu từ vấn đề lớn và giải quyết các bài toán con nhỏ hơn cho đến khi đạt đến trường hợp cơ sở.
- Tiếp cận ngang (Bottom-up): Bắt đầu từ các trường hợp cơ sở và tính toán dần dần lên đến vấn đề gốc.

[5]. Tăng hiệu suất tính toán:

- Quy hoạch động giúp tăng hiệu suất tính toán bằng cách tránh việc tính toán lại các kết quả đã biết, giảm độ phức tạp của thuật toán và tối ưu hóa thời gian chạy.

[6]. Áp dụng cho các bài toán tối ưu hoá có cấu trúc con tối ưu:

- Quy hoạch động thường được áp dụng cho các bài toán tối ưu hoá có cấu trúc con tối ưu chồng lên nhau, nơi mà các bài toán con có thể được giải quyết một cách độc lập.

[7]. Có thể sử dụng các kỹ thuật khác nhau:

- Quy hoạch động có thể được thực hiện bằng cách sử dụng một loạt các kỹ thuật như memoization, phân tích dọc và phân tích ngang. Điều này tùy thuộc vào bản chất của vấn đề cụ thể mà chúng ta đang giải quyết.

4. Các điều kiện để chiến lược quy hoạch động hiệu quả

- **Optimal Substructure**:

  Lời giải tối ưu của bài toán có thể được xây dựng từ lời giải tối ưu của các bài toán con nhỏ hơn.

- **Overlapping Subproblems:**

  Cùng một bài toán con xuất hiện nhiều lần trong quá trình giải bài toán lớn.

5. Các bước để áp dụng chiến lược quy hoạch động

Để áp dụng giải thuật quy hoạch động vào giải quyết một bài toán cụ thể, bạn có thể tuân theo các bước sau:

Bước 1:Xác định bài toán con

- Xác định các bài toán con cần được giải quyết để xây dựng lời giải cho bài toán lớn.

Bước 2: Xác định điều kiện cơ sở và khởi tạo giá trị biên

- Xác định và khởi tạo các giá trị ban đầu cho những trường hợp cơ bản nhất (giá trị biên), thường là những bài toán con nhỏ nhất mà ta có thể giải quyết ngay lập tức.

Bước 3: Xác định công thức truy hồi

- Xác định công thức hoặc quy tắc để chuyển đổi từ các bài toán con đã biết lời giải sang bài toán lớn hơn.
- Công thức này thường dựa trên việc sử dụng kết quả của các bài toán con đã được giải quyết trước đó.

Bước 4: Lưu trữ và truy xuất kết quả

- Lưu trữ kết quả của các bài toán con trong một cấu trúc dữ liệu như mảng hoặc bảng để sử dụng lại khi cần.
- Truy xuất kết quả từ cấu trúc dữ liệu này khi tính toán các bài toán con lớn hơn.

Bước 5: Kết hợp kết quả để tìm lời giải cuối cùng

- Sau khi đã tính toán và lưu trữ tất cả các bài toán con cần thiết, sử dụng chúng để xây dựng và trả về lời giải cho bài toán gốc.

Ví dụ: Với bài toán số Fibonacci, Chúng ta sẽ áp dụng các bước trên để giải quyết bài toán tính số Fibonacci thứ n.

Bước 1: Xác định bài toán con:

- Trạng thái: F(i) là số Fibonacci thứ i.
- Tính F(i) dựa trên F(i−1) và F(i−2).

Bước 2: Xác định điều kiện cơ sở và khởi tạo giá trị biên:

- F(0) = F(1) = 1

Bước 3: Xác định công thức truy hồi:

- F(i) = F(i−1) + F(i−2)

Bước 4: Lưu trữ và truy xuất kết quả:

- Kết quả các số Fibonacci từ 0 đến n được lưu trữ trong một mảng a.

Bước 5: Kết hợp kết quả để tìm lời giải cuối cùng:

- Trả về giá trị a[n], là số Fibonacci thứ n.

Ví dụ minh họa: Bài toán coin changing

Bài toán "Coin Changing" (đổi tiền) là một bài toán kinh điển trong lý thuyết thuật toán, nơi chúng ta cần tìm cách đổi một số tiền cụ thể bằng số lượng ít nhất các đồng xu có mệnh giá cho trước.

Ví dụ: giả sử bạn có các mệnh giá đồng xu là 1, 5, 10 và 25 (các đồng xu phổ biến ở Mỹ) và bạn cần đổi số tiền n = 63 đơn vị. Giả sử số đồng xu của mỗi mệnh giá là vô hạn. Mục tiêu của bài toán là lựa chọn số lượng ít nhất các đồng xu để tổng mệnh giá của chúng bằng 63.

25 10 5 1

Hình 4.1. Danh sách các mệnh giá có thể lựa chọn

Cách tiếp cận quy hoạch động

Bước 1: Xác định bài toán con:

- Trạng thái: gọi a[i] là số lượng ít nhất các đồng xu cần để đổi được số tiền i.
- Bài toán con: Đổi các số tiền từ 0 đến n.

Bước 2: Khởi tạo giá trị biên:

- a[0] = 0 (không cần đồng xu nào để đổi số tiền 0).
- Các giá trị còn lại được khởi tạo là INT_MAX (một số nguyên đủ lớn).

Bước 3: Xác định công thức truy hồi:

- a[i] = min (a[i], a[i−coin] + 1) với mỗi đồng xu có mệnh giá coin.

Bước 4: Lưu trữ và truy xuất kết quả:

- Kết quả cho các giá trị từ 0 đến n được lưu trữ trong mảng a có kích thước n + 1.

Bước 5: Kết hợp kết quả để tìm lời giải cuối cùng:

- Trả về giá trị a[n], là số lượng ít nhất các đồng xu cần để đổi được số tiền n.

Hàm viết bằng C++:

```
int coinChange(int coins[], int m, int n)
{
    int a[n + 1];
    for (int i = 0; i <= n; ++i)  a[i] = INT_MAX;
    a[0] = 0;

    for (int i = 1; i <= n; ++i)
        for (int j = 0; j < m; ++j)
            if (coins[j]<=i && a[i-coins[j]] != INT_MAX)
                a[i] = min(a[i], a[i - coins[j]] + 1);
    if(a[n] == INT_MAX)
        return -1;
    return a[n];
}
```

Hàm main:

```
int main()
{
    int coins[] = {1, 5, 10, 25};
    int n = 63;
    int m = sizeof(coins) / sizeof(coins[0]);
    int result = coinChange(coins, m, n);
    if (result != -1)
        cout << "Solution: " << result << endl;
    else
        cout << "No solution !" << endl;
    return 0;
}
```

6. Ứng dụng

Chiến lược quy hoạch động (Dynamic Programming - DP) là một phương pháp giải bài toán phức tạp bằng cách chia nhỏ chúng thành các bài toán con nhỏ hơn và lưu trữ các kết quả của các bài toán con đó để tránh việc tính toán lại. Dưới đây là một số ứng dụng phổ biến của chiến lược quy hoạch động:

[1]. Tính toán dãy Fibonacci: Đây là ví dụ cơ bản nhất về chiến lược quy hoạch động. Sử dụng DP, ta có thể tính toán số Fibonacci thứ nnn một cách hiệu quả hơn so với phương pháp đệ quy thông thường.

[2]. Bài toán Knapsack (Túi đựng đồ): Trong bài toán này, bạn có một túi có trọng lượng giới hạn và một số lượng đồ vật với trọng lượng và giá trị khác nhau. Mục tiêu là chọn một tập hợp các đồ vật để đặt vào túi sao cho tổng giá trị của chúng là lớn nhất mà không vượt quá trọng lượng của túi. Chiến lược quy hoạch động rất hiệu quả trong việc giải quyết bài toán này.

[3]. Tìm dãy con dài nhất (Longest Common Subsequence): Trong bài toán này, ta được cung cấp hai dãy và mục tiêu là tìm dãy con dài nhất mà là một phần của cả hai dãy đó. DP thường được sử dụng để giải quyết bài toán này một cách hiệu quả.

[4]. Tìm dãy con dài nhất với tổng lớn nhất (Maximum Sum Increasing Subsequence): Bài toán này yêu cầu tìm dãy con có tổng lớn nhất trong dãy số đã cho, trong đó các phần tử của dãy con tăng dần. Chiến lược quy hoạch động là một phương pháp phổ biến để giải quyết bài toán này.

[5]. Tìm đường đi ngắn nhất trong đồ thị (Shortest Path in a Graph): Trong bài toán này, ta được cung cấp một đồ thị có trọng số và một điểm xuất phát. Mục tiêu là tìm đường đi ngắn nhất từ điểm xuất phát đến một điểm đích cụ thể. Thuật toán Dijkstra và thuật toán Bellman-Ford là hai ví dụ điển hình về việc sử dụng quy hoạch động để tìm đường đi ngắn nhất trong đồ thị.

[6]. Bài toán dãy số tăng dần dài nhất (Longest Increasing Subsequence): Trong bài toán này, mục tiêu là tìm dãy con dài nhất trong dãy số đã cho sao cho các phần tử trong dãy con tăng dần. DP là một cách hiệu quả để giải quyết bài toán này.

[7]. Tính toán lời giải cho bài toán xếp hạng (Ranking Problem): Trong nhiều bài toán xếp hạng như xếp hạng trường học, xếp hạng nhân viên, chiến lược quy hoạch động có thể được sử dụng để tính toán xếp hạng một cách chính xác và hiệu quả.

[8]. Tìm kiếm dãy con chung dài nhất (Longest Common Substring): Trong bài toán này, ta cần tìm dãy con liên tục dài nhất xuất hiện trong cả hai chuỗi đã cho. Chiến lược quy hoạch động thường được sử dụng để giải quyết bài toán này.

[9]. Tìm kiếm dãy con Palindrome dài nhất (Longest Palindromic Subsequence): Trong bài toán này, ta cần tìm dãy con dài nhất trong dãy số đã cho mà là một chuỗi Palindrome. DP thường được sử dụng để giải quyết bài toán này.

[10]. Bài toán xếp thùng (Bin Packing Problem): Trong bài toán này, ta cần xếp một số lượng hữu hạn các đối tượng vào các thùng sao cho số lượng thùng được sử dụng là ít nhất. Chiến lược quy hoạch động là một trong những cách hiệu quả để giải quyết bài toán này.

[11]. Bài toán tối ưu hóa dãy con (Optimal Substructure Problem): Trong nhiều bài toán, ta cần tìm một dãy con có tổng hoặc giá trị tối ưu nhất. Chiến lược quy hoạch động thường được sử dụng để giải quyết các bài toán này, ví dụ như tìm dãy con có tổng lớn nhất hoặc nhỏ nhất, dãy con không chứa các phần tử liên tiếp nhau có tổng lớn nhất hoặc nhỏ nhất, và nhiều bài toán tối ưu hóa khác.

[12]. Bài toán phân loại (Classification Problem): Trong bài toán phân loại, ta cần phân loại các mẫu vào các nhóm khác nhau dựa trên các đặc trưng của chúng. Chiến lược quy hoạch động có thể được sử dụng để xây dựng mô hình phân loại hiệu quả trong nhiều trường hợp, bao gồm phân loại văn bản, phân loại hình ảnh, và phân loại dữ liệu về y tế.

[13]. Bài toán tìm đường đi ngắn nhất trong ma trận (Shortest Path Problem): Bài toán này yêu cầu tìm đường đi ngắn nhất từ một điểm xuất phát đến một điểm đích trong một ma trận, trong đó các ô của ma trận có giá trị là trọng

số của các cạnh. Thuật toán Dijkstra và thuật toán Floyd-Warshall là hai ví dụ về cách sử dụng quy hoạch động để giải quyết bài toán này.

[14]. Bài toán xếp lịch (Scheduling Problem): Trong bài toán xếp lịch, ta cần xếp lịch thời gian cho một loạt các sự kiện sao cho các ràng buộc nhất định được đáp ứng và mục tiêu được tối ưu hóa. Chiến lược quy hoạch động có thể được sử dụng để tối ưu hóa việc xếp lịch trong nhiều ngữ cảnh, bao gồm xếp lịch sản xuất, xếp lịch công việc, và xếp lịch giao hàng.

[15]. Bài toán tìm chuỗi con tương tự nhất (Longest Common Subsequence): Trong bài toán này, ta cần tìm chuỗi con dài nhất mà xuất hiện trong cả hai chuỗi đã cho. DP thường được sử dụng để giải quyết bài toán này, ví dụ như trong việc so sánh chuỗi văn bản, tìm kiếm chuỗi trong dữ liệu DNA, và các ứng dụng khác.

[16]. Bài toán cắt thanh sắt (Rod Cutting Problem): Trong bài toán này, ta cần cắt một thanh sắt thành các phần sao cho tổng giá trị của các phần cắt là lớn nhất. DP là một cách hiệu quả để giải quyết bài toán này, đặc biệt khi có một loạt các kích thước phần cắt và giá trị tương ứng.

Trên đây chỉ là một số ứng dụng phổ biến của chiến lược quy hoạch động, nhưng thực tế có rất nhiều bài toán có thể được giải quyết một cách hiệu quả bằng cách sử dụng phương pháp này.