Bài 1.5: Quy hoạch động

- ✓ Định nghĩa
- ✓ Đặc điểm
- ✓ Cách tiếp cận
- ✓ Ví dụ minh họa
- ✓ Bài tập thực hành

Định nghĩa

- Quy hoạch động trong tiếng Anh là dynamic programming.
- Là một phương pháp giảm thời gian chạy của các thuật toán thể hiện các tính chất của các bài toán con gối nhau và cấu trúc con tối ưu.
- Được phát minh bởi Richard Bellman năm 1953.
- Bài toán con gối nhau(overlapping subproblems) là các bài toán có thể chia nhỏ sau đó được tái sử dụng về sau hoặc một thuật toán đệ quy cho các bài toán lặp đi lặp lại thay vì tạo ra bài toán mới.
- Ví dụ tìm số Fibonacci Fn: Fn = Fn -1 + Fn-2.
- Cấu trúc con tối ưu là các lời giải tối ưu cho các bài toán con có thể được sử dụng để tìm các lời giải tối ưu cho bài toán toàn cục.

Đặc điểm

Để giải một bài toán với cấu trúc con tối ưu, ta sử dụng quy trình ba bước:

- B1: chia bài toán thành các bài toán con nhỏ hơn.
- B2: giải các bài toán con này một cách tối ưu bằng cách sử dụng đệ quy quy trình 3 bước này.
- B3: sử dụng các kết quả tối ưu đó để xây dựng lời giải tối ưu cho bài toán ban đầu.

Các bài toán con được giải bằng cách chia chúng thành các bài toán nhỏ hơn cho tới khi gặp trường hợp đơn giản tìm được lời giải.

Ví dụ

Số Fibonacci là số thỏa mãn F0 = 0, F1 = 1, với mọi n >= 2: Fn = Fn-1 + Fn-2. Giả sử ta tính F5:

- F5 = F4 + F3 = (F3 + F2) + (F2 + F1);
- F5 = ((F2 + F1) + (F1 + F0)) + ((F1 + F0) + F1);
- F5 = (((F1 + F0) + F1) + (F1 + F0)) + ((F1 + F0) + F1);

Nhận thấy rằng việc tính F5, ta phải tính F2, F1, F0 nhiều lần.

Từ đó để tránh việc thực hiện lặp đi lặp lại một vấn đề(tính F2 chẳng hạn) ta lưu lời giải của các bài toán con đã giải. Sau này khi cần giải lại bài toán đó, ta chỉ cần lấy ra và sử dụng, không phải tính lại.

Hướng tiếp cận này gọi là lưu trữ(memorization).

Nếu chắc chắn rằng một lời giải nào đó không còn cần thiết nữa ta có thể xóa nó đi để tiết kiệm không gian bộ nhớ.

Đặc điểm

Tóm lại, quy hoạch động sử dụng:

- Các bài toán con gối nhau.
- Cấu trúc con tối ưu.
- Memoization.

Cách tiếp cận

- Có 2 cách tiếp cận với quy hoạch động:
- Top-down: từ trên xuống, bài toán được chia thành các bài toán con, các bài toán con được giải và ghi nhớ lại đề phòng trường hợp cần dùng lại. Đây là trường hợp đệ quy và lưu trữ được kết hợp lại với nhau.
- Bottom-up: từ dưới lên trên, tất cả các bài toán con có thể cần đến đều được giải trước. Sau đó được dùng để xây dựng lời giải cho các bài toán lớn hơn.
- Cách tiếp cận bottom-up tốt hơn về không gian bộ nhớ sử dụng so với top-down tuy nhiên nó lại khó thực hiện hơn do đòi hỏi nhiều kĩ năng hơn.

Ví dụ

```java
private static final int MAX = 90;
private static long[] fibo = new long[MAX];

// tìm số fibonacci quy hoạch động theo hướng top-down
// giả định n >= 0
private static long fibonacci(int n) {
    if (n < 2) {
        return fibo[n] = n; // trường hợp cơ sở
    }
    if (fibo[n] == 0) {
        fibo[n] = fibonacci( n: n - 1) + fibonacci( n: n - 2); // top-down
    }
    return fibo[n]; // trả về kết quả
}
```

Ví dụ

```java
// tìm số fibonacci quy hoạch động sử dụng chiến lược bottom-up
private static long fibonacciV2(int n) {
    long f0 = 0;
    long f1 = 1;
    long fn = n < 2 ? n : 0;
    for (int i = 2; i <= n; i++) {
        fn = f0 + f1; // tìm lời giải của bài toán nhỏ hơn
        f0 = f1;
        f1 = fn;
    }
    return fn; // trả về kết quả
}
```

Nội dung tiếp theo

Thuật toán quay lui