ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÀI TẬP QUY HOẠCH ĐỘNG

Phan Nhật Tân - 23521405
Nguyễn Huy Phước - 23521234

Mục lục

1 Câu hỏi

1.1 Có phải mọi bài toán đều có thể giải quyết bằng quy hoạch động không? Tại sao?

Không, không phải mọi bài toán đều có thể giải quyết bằng **quy hoạch động (dynamic programming - DP)**. Điều này phụ thuộc vào hai điều kiện chính:

- **Cấu trúc con tối ưu (Optimal Substructure):** Một bài toán có cấu trúc con tối ưu nếu lời giải của bài toán lớn có thể được xây dựng từ lời giải của các bài toán con nhỏ hơn.
  - Ví dụ: Bài toán Fibonacci thỏa mãn điều kiện này, vì $F(n) = F(n-1) + F(n-2)$.
- **Tính chồng lặp của bài toán con (Overlapping Subproblems):** Một bài toán có tính chồng lặp khi các bài toán con giống nhau được tính lại nhiều lần trong quá trình giải quyết.
  - Ví dụ: Trong bài toán Fibonacci, nhiều lời gọi cho $F(n-2)$ và $F(n-3)$ trùng lặp.

Nếu một bài toán không thỏa mãn **cấu trúc con tối ưu** hoặc **tính chồng lặp**, nó không phù hợp với quy hoạch động.

- **Ví dụ:** Bài toán *tìm kiếm tuyến tính* trong mảng không có tính chất chồng lặp, vì mỗi phần tử chỉ cần xét đúng một lần.

1.2 Trong thực tế, bạn đã gặp bài toán nào có thể áp dụng quy hoạch động? Hãy chia sẻ cách tiếp cận.

Ví dụ: Bài toán lập lịch công việc (Job Scheduling Problem)

- **Mô tả bài toán:** Cho n công việc, mỗi công việc i có thời gian bắt đầu s_i, thời gian kết thúc f_i, và giá trị v_i. Chọn các công việc sao cho tổng giá trị là lớn nhất và không có hai công việc nào chồng lấn thời gian.
- **Ứng dụng thực tế:**
  - Lập lịch sản xuất trong nhà máy.
  - Phân bổ tài nguyên máy chủ (server scheduling).
- **Cách tiếp cận bằng quy hoạch động:**
  1. **Sắp xếp công việc:** Sắp xếp các công việc theo thứ tự thời gian kết thúc tăng dần $f_1, f_2, \ldots, f_n$.
  2. **Xác định trạng thái:** Gọi $dp[i]$ là tổng giá trị lớn nhất khi xét đến công việc thứ i.
  3. **Công thức chuyển:** Với mỗi công việc i, ta có hai lựa chọn:
     - Không chọn công việc i: $dp[i] = dp[i-1]$.
     - Chọn công việc i: $dp[i] = dp[j] + v_i$, với j là công việc cuối cùng trước i không chồng lấn (tức $f_j \leq s_i$).

     Khi đó:
     $$dp[i] = \max(dp[i-1], dp[j] + v_i)$$
  4. **Khởi tạo:** $dp[0] = 0$ (không có công việc nào được chọn).
  5. **Kết quả:** Giá trị lớn nhất của bài toán là $dp[n]$.

1.3 Hãy phân tích và làm rõ ưu, nhược điểm của 2 phương pháp Top-down và Bottom-up. Bạn sẽ ưu tiên phương pháp nào? Vì sao?

1. Phương pháp Top-down (Memoization):

- **Cách hoạt động:** Bắt đầu từ bài toán lớn nhất, gọi đệ quy để giải các bài toán con và lưu kết quả vào bảng nhớ (cache) để tránh tính lại.
- **Ưu điểm:**
  - Dễ hiện thực hơn nếu quen với đệ quy.
  - Tối ưu cho các bài toán có nhiều trạng thái không được sử dụng (tiết kiệm bộ nhớ).
- **Nhược điểm:**
  - Giới hạn bởi ngăn xếp đệ quy (stack) trong các bài toán lớn.
  - Có thể gây lãng phí thời gian nếu một bài toán nhỏ bị gọi nhiều lần không cần thiết.

2. Phương pháp Bottom-up (Tabulation):

- **Cách hoạt động:** Bắt đầu từ các bài toán con nhỏ nhất, tính toán theo thứ tự từ dưới lên và lưu kết quả trực tiếp vào bảng.
- **Ưu điểm:**
  - Không phụ thuộc vào ngăn xếp đệ quy, hiệu quả hơn về mặt bộ nhớ.
  - Tránh được các chi phí gọi hàm đệ quy.
- **Nhược điểm:**
  - Phải tính mọi trạng thái, ngay cả khi một số trạng thái không được sử dụng.
  - Khó hiện thực hơn nếu không quen với cách tiếp cận vòng lặp.

Lựa chọn ưu tiên: Tôi thường ưu tiên phương pháp **Bottom-up**, vì nó ổn định hơn trong các bài toán phức tạp và tối ưu hiệu năng khi số lượng trạng thái lớn.

2 Bài tập

2.1 Bài tập 1

Phân tích bài toán

- **Trạng thái quy hoạch động:** Gọi $f[i]$ là chi phí tối thiểu để đến hòn đá i.
- **Công thức chuyển trạng thái:**

$$f[i] = \min_{j=i-k}^{i-1} \{f[j] + |h[i] - h[j]|\}$$

  Với $j \geq 1$.
- **Khởi tạo:** $f[1] = 0$ (chi phí tại hòn đá đầu tiên là 0).
- **Kết quả:** Giá trị $f[n]$ là chi phí tối thiểu để đến hòn đá cuối cùng.

Ví dụ

- **Input:**

```
5 3
10 30 40 50 20
```

- **Output:**

```
30
```

- **Giải thích:**
  - $n = 5, k = 3$: Chó có thể nhảy tối đa 3 hòn đá.
  - Dãy chiều cao: $[10, 30, 40, 50, 20]$.
  - Chi phí tối thiểu:
    - Từ hòn 1 ($h[1] = 10$) đến hòn 2 ($h[2] = 30$): Chi phí $|30 - 10| = 20$.
    - Từ hòn 2 ($h[2] = 30$) đến hòn 5 ($h[5] = 20$): Chi phí $|20 - 30| = 10$.
  - Tổng chi phí: $20 + 10 = 30$.

Code giải bài toán Frog

File: code.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
int64_t n,a[100001],f[100001],k;
int main()
{
    ios_base:: sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    cin>>n>>k;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        f[i]=INT_MAX;
    }
    f[1]=0;
    for(int i=2;i<=n;i++)
    {
        for(int j=i-k;j<i;j++)
        {
            if(j>=1)
                f[i]=min(f[j]+abs(a[i]-a[j]),f[i]);
        }
    }
    cout<<f[n];
    return 0;
}
```

Độ phức tạp

- **Thời gian:** $O(n \cdot k)$, vì duyệt qua n hòn đá, với mỗi hòn đá tối đa k lần.
- **Không gian:** $O(n)$, chỉ sử dụng mảng a và f.

Liên kết đến GitHub

Xem mã nguồn tại link.