ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

# Quy hoạch động

**Nhóm 5 thực hiện**

**Môn học: CS112.P11.KHTN**

*Sinh viên thực hiện:*

Nguyễn Thiên Bảo - 23520127

Trần Lê Minh Nhật - 23521098

*Giáo viên hướng dẫn:*

Nguyễn Thanh Sơn

ngày 4 tháng 11 năm 2024

# Mục lục

# 1 Bài Tập Lý Thuyết

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

**Trả lời:**

Không, không phải mọi bài toán đều có thể giải quyết bằng quy hoạch động. Quy hoạch động chỉ áp dụng hiệu quả cho những bài toán thỏa mãn hai tính chất quan trọng:

1. **Bài toán con lặp lại (Overlapping Subproblems):** Bài toán lớn có thể được chia thành các bài toán con nhỏ hơn, và các bài toán con này xuất hiện lặp đi lặp lại trong quá trình giải quyết.
2. **Cấu trúc con tối ưu (Optimal Substructure):** Giải pháp tối ưu của bài toán lớn được xây dựng từ các giải pháp tối ưu của các bài toán con.

Nếu một bài toán không thỏa mãn hai điều kiện trên, việc áp dụng quy hoạch động sẽ không mang lại hiệu quả, thậm chí có thể làm tăng độ phức tạp tính toán. Ví dụ, trong thuật toán tìm kiếm nhị phân, mỗi lần chia nhỏ chỉ giải quyết một bài toán con duy nhất, không có sự lặp lại của các bài toán con, do đó quy hoạch động không phù hợp.
Ngoài ra, nếu chi phí lưu trữ kết quả của các bài toán con hoặc thời gian tính toán trong quy hoạch động lớn hơn lợi ích mang lại, thì việc áp dụng quy hoạch động cũng không hiệu quả.

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

**Trả lời:**

**Bài Toán Lập Lịch Công Việc với Lợi Nhuận và Thời Hạn**

**Tình huống thực tế**

Một công ty có một danh sách các dự án cần hoàn thành. Mỗi dự án có:

- **Thời gian bắt đầu** ($s_i$) và **thời gian kết thúc** ($f_i$).
- **Lợi nhuận** ($p_i$) thu được nếu hoàn thành dự án.

Mục tiêu của công ty là chọn một tập hợp các dự án để thực hiện sao cho:

- **Tổng lợi nhuận thu được là tối đa**.
- **Không có sự chồng chéo về thời gian** giữa các dự án đã chọn.

## Cách tiếp cận bằng quy hoạch động

1. **Biểu diễn bài toán:**
   - **Sắp xếp các dự án** theo thứ tự thời gian kết thúc tăng dần.
   - Gọi $n$ là số lượng dự án.
   - Ký hiệu:
     - $jobs[i]$: dự án thứ $i$ với thời gian bắt đầu $s_i$, kết thúc $f_i$, và lợi nhuận $p_i$.
     - $dp[i]$: lợi nhuận tối đa có thể đạt được khi xem xét các dự án từ 1 đến $i$.
2. **Xây dựng công thức quy hoạch động:**
   - Đối với mỗi dự án $i$, có hai lựa chọn:
     - **Không chọn** dự án $i$: $dp[i] = dp[i-1]$.
     - **Chọn** dự án $i$: lợi nhuận là $p_i$ cộng với lợi nhuận tối đa từ các dự án không chồng chéo trước đó.
   - **Tìm chỉ số** $l$ là dự án cuối cùng trước $i$ mà không chồng chéo thời gian với $i$:

$$l = \max\{j \mid f_j \leq s_i\}$$

- **Công thức chuyển trạng thái:**

$$dp[i] = \max\left(dp[i-1], dp[l] + p_i\right)$$

với $dp[0] = 0$.

3. **Thuật toán:**
   (a) **Khởi tạo:** $dp[0] = 0$.
   (b) **Duyệt qua các dự án từ** $i = 1$ **đến** $n$**:**
   - **Tìm** $l$**:** Dùng tìm kiếm nhị phân để tìm dự án không chồng chéo cuối cùng trước $i$.
   - **Cập nhật** $dp[i]$**:**

$$dp[i] = \max\left(dp[i-1], dp[l] + p_i\right)$$

4. **Kết quả:**
   - **Tổng lợi nhuận tối đa** là $dp[n]$.

## Giải thích chi tiết

- **Sắp xếp dự án:** Việc sắp xếp các dự án theo thời gian kết thúc giúp chúng ta dễ dàng tìm kiếm các dự án không chồng chéo bằng cách sử dụng tìm kiếm nhị phân.
- **Tìm dự án không chồng chéo** $l$**:** Để tính $dp[i]$, cần biết lợi nhuận tối đa $dp[l]$ của các dự án trước đó mà không chồng chéo với dự án $i$.
- **Quy hoạch động:** Công thức $dp[i] = \max\left(dp[i-1], dp[l] + p_i\right)$ thể hiện việc lựa chọn giữa:
  - **Bỏ qua dự án** $i$**:** Giữ nguyên lợi nhuận tối đa trước đó $dp[i-1]$.
  - **Chọn dự án** $i$**:** Cộng lợi nhuận $p_i$ với lợi nhuận tối đa của các dự án không chồng chéo trước đó $dp[l]$.
- **Tối ưu hóa bằng tìm kiếm nhị phân:** Do các dự án đã được sắp xếp theo thời gian kết thúc, ta có thể tìm $l$ một cách hiệu quả trong $O(\log n)$ thời gian.

## Ví dụ minh họa

Giả sử có 4 dự án với thông tin như sau:

| Dự án ($i$) | Thời gian bắt đầu ($s_i$) | Thời gian kết thúc ($f_i$) | Lợi nhuận ($p_i$) |
|---|---|---|---|
| 1 | 1 | 3 | 5 |
| 2 | 2 | 5 | 6 |
| 3 | 4 | 6 | 5 |
| 4 | 6 | 7 | 4 |

**Sắp xếp dự án theo $f_i$:** Dự án đã được sắp xếp.

**Khởi tạo:** $dp[0] = 0$.

**Tính $dp[i]$ cho từng $i$:**

- $i = 1$:
  - $l = 0$ (không có dự án nào trước đó).
  - $dp[1] = \max(dp[0], dp[0] + 5) = 5$.
- $i = 2$:
  - $l = 0$ (dự án 1 kết thúc tại $f_1 = 3 > s_2 = 2$).
  - $dp[2] = \max(dp[1], dp[0] + 6) = 6$.
- $i = 3$:
  - $l = 1$ (dự án 1 kết thúc tại $f_1 = 3 \leq s_3 = 4$).
  - $dp[3] = \max(dp[2], dp[1] + 5) = \max(6, 5 + 5) = 10$.
- $i = 4$:
  - $l = 3$ (dự án 3 kết thúc tại $f_3 = 6 \leq s_4 = 6$).
  - $dp[4] = \max(dp[3], dp[3] + 4) = \max(10, 10 + 4) = 14$.

**Kết quả:** Lợi nhuận tối đa là $dp[4] = 14$.

### Lợi ích của quy hoạch động trong bài toán này

- **Hiệu quả:** Giảm độ phức tạp thời gian từ thử tất cả các tập hợp dự án ($O(2^n)$) xuống $O(n \log n)$.
- **Tối ưu hóa:** Đảm bảo tìm được tổng lợi nhuận lớn nhất có thể mà không vi phạm các ràng buộc về thời gian.

### Kết luận

- Bài toán lập lịch công việc với lợi nhuận và thời hạn là một ví dụ thực tế điển hình có thể giải quyết hiệu quả bằng quy hoạch động.
- Cách tiếp cận này có thể mở rộng cho nhiều bài toán lập lịch và tối ưu hóa khác trong quản lý dự án và kinh doanh.

## 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?

**Trả lời:**

**Phương pháp Top-down (Đệ quy có nhớ - Memoization):**

- **Ưu điểm:**
  - **Dễ hiểu và dễ triển khai:** Bắt đầu từ bài toán lớn, chia nhỏ thành các bài toán con. Phù hợp với tư duy đệ quy.
  - **Chỉ tính toán những trạng thái cần thiết:** Tránh tính toán những trạng thái không được sử dụng trong kết quả cuối cùng.
- **Nhược điểm:**
  - **Tiêu tốn bộ nhớ ngăn xếp:** Sử dụng đệ quy có thể gây tràn stack khi số lượng trạng thái lớn.

  - **Hiệu suất chậm hơn:** Overhead của các cuộc gọi đệ quy có thể làm giảm hiệu suất so với phương pháp Bottom-up.

**Phương pháp Bottom-up (Điền bảng - Tabulation):**

- **Ưu điểm:**
  - **Hiệu suất cao hơn:** Không có overhead của đệ quy, thường chạy nhanh hơn.
  - **Kiểm soát bộ nhớ tốt hơn:** Tránh được việc tràn stack, có thể tối ưu bộ nhớ bằng cách chỉ lưu trữ các trạng thái cần thiết.
- **Nhược điểm:**
  - **Khó triển khai hơn trong một số trường hợp:** Cần xác định thứ tự tính toán và mối quan hệ giữa các trạng thái một cách tường minh.
  - **Có thể tính dư thừa một số trạng thái không cần thiết:** Không tối ưu nếu chỉ một phần nhỏ trạng thái được sử dụng.

**Ưu tiên lựa chọn:**

- **Nếu bài toán có không gian trạng thái nhỏ hoặc trung bình**, và cần triển khai nhanh chóng, **phương pháp Top-down** là lựa chọn phù hợp.
- **Nếu bài toán có không gian trạng thái lớn**, cần hiệu suất cao và kiểm soát bộ nhớ tốt, **phương pháp Bottom-up** nên được ưu tiên.

**Lựa chọn cá nhân:**
Tôi sẽ ưu tiên **phương pháp Bottom-up** vì:

- **Hiệu suất cao:** Giảm thiểu overhead, tối ưu thời gian chạy.
- **Kiểm soát bộ nhớ tốt:** Tránh nguy cơ tràn stack, đặc biệt quan trọng trong các bài toán lớn.
- **Dễ tối ưu hóa:** Có thể áp dụng các kỹ thuật tối ưu bộ nhớ, như sử dụng mảng một chiều hoặc chỉ lưu trữ các trạng thái cần thiết.

# 2 Bài Tập Thực Hành

## 2.1 Bài toán: Chú Ếch Nhảy

### 2.1.1 Đề bài

Cho $n$ hòn đá được đánh số từ 1 đến $n$, với độ cao tương ứng $h_1, h_2, \ldots, h_n$.

- Ban đầu, chú ếch ở hòn đá thứ 1.
- Mỗi lần, chú ếch có thể nhảy từ hòn đá thứ $i$ đến một hòn đá thứ $j$ với điều kiện $i < j \leq i+k$.
- Chi phí để nhảy từ hòn đá $i$ đến hòn đá $j$ là $|h_i - h_j|$.

Yêu cầu: Tìm chi phí tối thiểu để chú ếch nhảy từ hòn đá thứ 1 đến hòn đá thứ $n$.

### 2.1.2 Ý tưởng giải quyết

Ta sử dụng phương pháp **Dynamic Programming (Quy hoạch động)** để giải bài toán. Định nghĩa:

$$dp[i] = \text{chi phí tối thiểu để đến hòn đá } i$$

Với công thức truy hồi:

$$dp[i] = \min_{j=1\ldots k} (dp[i-j] + |h[i] - h[i-j]|)$$

$$dp[1] = 0$$

**Tính tối ưu**:

- Quy hoạch động giúp giảm số lượng phép tính bằng cách tái sử dụng kết quả từ các bài toán con đã giải.
- Trạng thái $dp[i]$ chứa thông tin chi phí tối thiểu để đạt đến hòn đá $i$, đảm bảo không lặp lại các phép tính không cần thiết.

**Phù hợp với bài toán dạng tuần tự**:

- Hành động nhảy của chú ếch phụ thuộc vào các hòn đá trước đó $(i-1, i-2, \ldots, i-k)$.
- Quy hoạch động tự nhiên phản ánh cấu trúc bài toán khi trạng thái $dp[i]$ chỉ phụ thuộc vào các trạng thái trước đó.

### 2.1.3 Mã giả

**Algorithm 1** Tìm chi phí tối thiểu

**Require:** $n$, $k$, $h[1 \ldots n]$
**Ensure:** Chi phí tối thiểu để nhảy đến hòn đá $n$

Khởi tạo $dp[1] = 0$ và $dp[i] = \infty$ với mọi $i \geq 2$
**for** $i = 2$ **to** $n$ **do**
  **for** $j = 1$ **to** $\min(k, i-1)$ **do**
    $dp[i] = \min(dp[i], dp[i-j] + |h[i] - h[i-j]|)$
  **end for**
**end for**
**return** $dp[n]$ =0

### 2.1.4 Code Python

- Link code tham khảo của bài toán: code.
- Dưới đây là hàm được dùng trong code trên:

```
def min_cost_to_jump(n, k, heights):
    dp = [float('inf')] * n
    dp[0] = 0

    for i in range(1, n):
        for j in range(1, k + 1):
            if i - j >= 0:
                dp[i] = min(dp[i], dp[i - j] + abs(heights[i] - heights[i - j]))

```

```
    return dp[n - 1]

n, k = map(int, input("Nhap so luong hon da va gioi han nhay: ").split())
heights = list(map(int, input("Nhap do cao cac hon da: ").split()))

result = min_cost_to_jump(n, k, heights)
print("Chi phi toi thieu de nhay den hon da cuoi cung la:", result)
```

Listing 1: Python Code: Minimum Cost to Jump Stones