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

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN
CS112.P11.KHTN

BTVN NHÓM 9

Nhóm 7 :
Hoàng Đức Dũng - 23520328
Nguyễn Văn Hồng Thái - 23521418

Giảng viên :
Nguyễn Thanh Sơn

Ngày 5 tháng 12 năm 2024

Mục lục

1 Lí thuyết

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

- Không phải mọi bài toán đều được giải quyết bằng quy hoạch động.
- **Lý do:** Quy hoạch động chỉ áp dụng được cho các bài toán thỏa mãn:
  - **Tính chất con lặp lại (Overlapping Subproblems):** Bài toán có thể được chia nhỏ thành các bài toán con tương tự và các bài toán con này được giải đi giải lại.
  - **Tính tối ưu con (Optimal Substructure):** Nghiệm của bài toán lớn có thể xây dựng từ nghiệm của các bài toán con.
- Nếu bài toán không thỏa mãn một trong hai tính chất trên (ví dụ, các bài toán cần sự tìm kiếm toàn cục hoặc không thể chia nhỏ), thì quy hoạch động không thể áp dụng 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.

- Trong thực tế tôi đã gặp bài toán Balo (Knapsack)
- **Cách tiếp cận:**
  - **Tính chất con lặp lại:** Quyết định cho từng vật phụ thuộc vào trạng thái hiện tại.
  - **Tính tối ưu con:** Kết quả tối ưu của balo lớn có thể xây dựng từ các balo nhỏ hơn.
  - **Phương pháp:**
    - **Top-down:** Sử dụng đệ quy với lưu trữ kết quả (memoization).
    - **Bottom-up:** Sử dụng bảng để lưu kết quả từ nhỏ đến lớn.

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?

Top-down (Đệ quy + Memoization)

- **Ưu điểm:**
  - Dễ cài đặt và trực quan khi bài toán được mô tả theo đệ quy.
  - Hiệu quả khi chỉ cần tính toán một số trạng thái cụ thể.
- **Nhược điểm:**
  - Tiêu tốn bộ nhớ ngăn xếp, dễ gây lỗi tràn ngăn xếp với bài toán lớn.
  - Chậm hơn do chi phí quản lý ngăn xếp và kiểm tra bảng memoization.

Bottom-up (Bảng)

- **Ưu điểm:**
  - Không sử dụng đệ quy, tránh lỗi tràn ngăn xếp.
  - Hiệu quả hơn khi cần tính toán toàn bộ trạng thái.
- **Nhược điểm:**
  - Tính toàn bộ trạng thái, ngay cả khi một số trạng thái không cần thiết.
  - Khó cài đặt hơn cho bài toán phức tạp.

Ưu tiên phương pháp nào?

- Tôi ưu tiên **Bottom-up** vì:
  - An toàn hơn với bài toán lớn, tránh lỗi tràn ngăn xếp.
  - Thường nhanh hơn nhờ giảm chi phí quản lý đệ quy.

Tuy nhiên, với bài toán cụ thể hoặc khi dễ mô tả bằng đệ quy, tôi sẽ chọn **Top-down**.

2 Thực hành

2.1 Bài toán chú ếch

Ý tưởng giải quyết bài toán

Bài toán yêu cầu tối ưu hóa chi phí nhảy từ hòn đá đầu tiên đến hòn đá cuối cùng. Chúng ta có thể giải quyết bằng phương pháp **Quy hoạch động (Dynamic Programming)**.

*Phân tích bài toán

- Gọi $dp[i]$ là chi phí tối thiểu để nhảy đến hòn đá thứ i.
- Mỗi lần nhảy từ i đến j ($i < j \leq i + k$), chi phí là $|h[i] - h[j]|$.
- Bài toán yêu cầu tối thiểu hóa $dp[i]$ từ $dp[1]$ đến $dp[n]$.

*Công thức truy hồi

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

Trong đó:

- $i - k$: Hòn đá xa nhất mà ếch có thể nhảy đến.
- $dp[j] + |h[i] - h[j]|$: Chi phí khi nhảy từ j đến i.

*Khởi tạo

$$dp[1] = 0$$

Không có chi phí khi bắt đầu từ hòn đá đầu tiên.

*Kết quả Kết quả cần tìm là $dp[n]$, chi phí tối thiểu để nhảy đến hòn đá cuối cùng.

*Độ phức tạp

- **Thời gian:** Với mỗi hòn đá i, duyệt tối đa k hòn đá trước đó. Tổng độ phức tạp: $O(n \cdot k)$.
- **Không gian:** Bộ nhớ $O(n)$ để lưu mảng dp.

Mã giả

```
INPUT: n, k
       h[1...n]

FUNCTION frog_jump(n, k, h):
    dp = array of size n, initialized with INF
    dp[1] = 0

    FOR i FROM 2 TO n:
        FOR j FROM max(1, i-k) TO i-1:
            dp[i] = min(dp[i], dp[j] + abs(h[i] - h[j]))

    RETURN dp[n]
```

Mã Python

```python
def frog_jump(n, k, h):
    dp = [float('inf')] * n
    dp[0] = 0

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

    return dp[-1]

n, k = map(int, input().split())
h = list(map(int, input().split()))

print(frog_jump(n, k, h))
```

Ví dụ minh họa

Input:

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

Output:

30

*Giải thích

- $dp[1] = 0$
- $dp[2] = dp[1] + |30 - 10| = 20$
- $dp[3] = \min(dp[1] + |40 - 10|, dp[2] + |40 - 30|) = \min(30, 30) = 30$
- $dp[4] = \min(dp[2] + |50 - 30|, dp[3] + |50 - 40|) = \min(40, 40) = 40$
- $dp[5] = \min(dp[2]+|20-30|, dp[3]+|20-40|, dp[4]+|20-50|) = \min(30, 50, 70) = 30$

Kết quả là $dp[5] = 30$.

2.2 Đại hội tin học UIT

Đặc điểm bài toán

- Mục tiêu: Đếm số cách gửi học sinh từ n trường thỏa mãn điều kiện:
  - Nếu trường i tham gia và gửi x học sinh thì $x > y$, với y là số học sinh của trường j (với $j < i$).
- Giới hạn số học sinh của mỗi trường i là từ a_i đến b_i học sinh.
- Nếu trường không tham gia, coi như trường đó không gửi học sinh.

Ý tưởng giải quyết

Bài toán này có thể giải quyết bằng phương pháp **Quy hoạch động (Dynamic Programming)**.

*Biến trạng thái Gọi $dp[i][x]$ là số cách phân bổ học sinh từ các trường $1, 2, \ldots, i$, trong đó:

- Trường i gửi x học sinh.
- x phải lớn hơn số lượng học sinh tối đa mà các trường $j < i$ gửi.

*Công thức truy hồi Để tính $dp[i][x]$, ta sử dụng công thức:

$$dp[i][x] = \sum_{y=a_{i-1}}^{x-1} dp[i-1][y]$$

Trong đó:

- a_{i-1}: Giới hạn tối thiểu số học sinh mà trường $i - 1$ có thể gửi.
- x: Số học sinh mà trường i gửi.
- $dp[i][x]$ là số cách gửi học sinh của trường i.

*Khởi tạo

$$dp[1] = 0$$

Không có chi phí khi bắt đầu từ hòn đá đầu tiên.

*Kết quả Kết quả cần tìm là tổng tất cả giá trị $dp[n][x]$ với $x \geq a_n$.

Độ phức tạp

- **Thời gian:** Với n trường và giới hạn $b[i] - a[i]$ cho mỗi trường, độ phức tạp là:

$$O(n \cdot \max(b[i] - a[i]))$$

- **Không gian:** $O(n \cdot \max_b)$ để lưu trữ mảng dp và $prefix$.

Mã giả

```
FUNCTION count_ways(n, a, b):
    MOD = 1,000,000,007
    dp = array of size [n+1][max_b+1], initialized to 0
    prefix = array of size [n+1][max_b+1], initialized to 0
    FOR x FROM a[1] TO b[1]:
        dp[1][x] = 1
        prefix[1][x] = prefix[1][x-1] + dp[1][x]
    FOR i FROM 2 TO n:
        FOR x FROM a[i] TO b[i]:
            dp[i][x] = (prefix[i-1][x-1] - prefix[i-1][a[i-1]-1])
               % MOD
            prefix[i][x] = (prefix[i][x-1] + dp[i][x]) % MOD
    result = 0
    FOR x FROM a[n] TO b[n]:
        result = (result + dp[n][x]) % MOD

    RETURN result
```

Mã Python

```python
def count_ways(n, intervals):
    MOD = 1000000007
    max_b = max(b for _, b in intervals)
    dp = [[0] * (max_b + 1) for _ in range(n + 1)]
    prefix = [[0] * (max_b + 1) for _ in range(n + 1)]
    a1, b1 = intervals[0]
    for x in range(a1, b1 + 1):
        dp[1][x] = 1
        prefix[1][x] = (prefix[1][x - 1] + dp[1][x]) % MOD
    for i in range(2, n + 1):
        ai, bi = intervals[i - 1]
        for x in range(ai, bi + 1):
            dp[i][x] = (prefix[i - 1][x - 1] - (prefix[i - 1][ai -
                1] if ai > 1 else 0)) % MOD
            prefix[i][x] = (prefix[i][x - 1] + dp[i][x]) % MOD
    an, bn = intervals[-1]
    result = sum(dp[n][x] for x in range(an, bn + 1)) % MOD

    return result
```

```
n = int(input())
intervals = [tuple(map(int, input().split())) for _ in range(n)]

print(count_ways(n, intervals))
```