ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Khoa Khoa học máy tính

Môn học Phân Tích Và Thiết Kế Thuật Toán

Bài tập Quy hoạch động

Cao Lê Công Thành
MSSV: 23521437
Đặng Quang Vinh
MSSV: 23521786

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?

Không. vì Quy hoạch động sẽ không thể áp dụng được (hoặc nói đúng hơn là áp dụng cũng không có tác dụng gì) khi các bài toán con không gối nhau. Ví dụ với thuật toán tìm kiếm nhị phân, quy hoạch động cũng không thể tối ưu được gì cả, bởi vì mỗi khi chia nhỏ bài toán lớn thành các bài toán con, mỗi bài toán cũng chỉ cần giải một lần mà không bao giờ được gọi lại.Ngoài ra,khi chi phí lưu trữ và thời gian tính toán của phương pháp quy hoạch động lớn hơn lợi ích mang lại.

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

Tình huống thực tế

Một công ty vận tải cần lập kế hoạch quản lý tuyến xe buýt sao cho:

- Phục vụ nhiều hành khách nhất có thể, nhằm đáp ứng nhu cầu đi lại của người dân.
- Giảm thiểu chi phí vận hành, bao gồm chi phí nhiên liệu, lương tài xế, bảo dưỡng xe.
- Tối ưu hóa thời gian di chuyển để hành khách không phải chờ đợi lâu và giảm thiểu thời gian đi lại.

Bài toán này là sự kết hợp giữa tối ưu hóa chi phí và tối ưu hóa lợi ích cho cả công ty và hành khách.

1.2.1 Biểu diễn bài toán

Giả sử có N tuyến đường tiềm năng, mỗi tuyến đường i có:

- Chi phí vận hành C_i (đơn vị: triệu đồng).
- Thời gian di chuyển T_i (đơn vị: giờ).
- Số lượng hành khách phục vụ P_i.

Công ty có:

- Ngân sách tối đa là B (đơn vị: triệu đồng).
- Thời gian tối đa là $T_{\max}$ (đơn vị: giờ).

Mục tiêu là tìm tập hợp các tuyến đường sao cho **tối đa hóa số lượng hành khách phục vụ** trong giới hạn chi phí và thời gian cho phép.

Mô hình Quy hoạch Động

Để giải quyết bài toán, ta sẽ sử dụng quy hoạch động với bảng lưu trữ $dp[i][j]$, trong đó:

- i là số tuyến đường đang xét.
- j là ngân sách còn lại.
- $dp[i][j]$ là số lượng hành khách tối đa có thể phục vụ khi xét đến tuyến đường thứ i với ngân sách còn lại là j.

Công thức quy hoạch động:

$$dp[i][j] = \begin{cases} dp[i-1][j], & \text{nếu không chọn tuyến đường } i \\ \max(dp[i-1][j], dp[i-1][j-C_i] + P_i), & \text{nếu chọn tuyến đường } i \text{ và điều kiện } T_i \leq T \end{cases}$$

1.2.2 Thuật toán Quy hoạch Động

Thuật toán quy hoạch động sẽ được triển khai theo các bước sau:

1. Khởi tạo bảng dp với giá trị ban đầu là 0.
2. Lặp qua từng tuyến đường i và ngân sách j.
3. Cập nhật giá trị tối ưu tại mỗi bước dựa trên công thức quy hoạch động.

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?

Phương pháp Top-down (Đệ quy có nhớ)

Phương pháp này bắt đầu từ bài toán lớn, chia nó thành các bài toán nhỏ hơn và giải quyết chúng qua các cuộc gọi đệ quy. Tuy nhiên, để tránh tính toán lại các bài toán con đã giải quyết, ta sẽ lưu kết quả vào bảng (memoization).

Ưu điểm

- **Dễ hiểu và dễ triển khai:** Phương pháp này rất trực quan và dễ dàng để triển khai. Vì chúng ta thường nghĩ đến việc chia nhỏ bài toán lớn thành các phần nhỏ, nên cách làm này giống với cách mà chúng ta giải quyết vấn đề trong thực tế.
- **Tiết kiệm thời gian tính toán:** Khi bài toán con đã được tính toán, kết quả của nó sẽ được lưu lại, vì vậy ta không phải tính lại từ đầu mỗi lần gặp bài toán con giống nhau. Điều này làm giảm đáng kể thời gian chạy.

Nhược điểm

- **Sử dụng nhiều bộ nhớ:** Vì sử dụng đệ quy và phải lưu trữ các cuộc gọi trong stack, nếu bài toán quá phức tạp, việc này có thể dẫn đến việc tốn bộ nhớ hoặc tràn stack.
- **Chạy chậm hơn:** Việc duy trì đệ quy có thể khiến chương trình chạy chậm hơn so với Bottom-up, đặc biệt là trong các bài toán có số lượng lớn các trạng thái.

Phương pháp Bottom-up (Điền bảng phương án liên tục)

Phương pháp này làm việc ngược lại so với Top-down. Nó bắt đầu từ những bài toán con đơn giản nhất và dần dần xây dựng lên kết quả cho bài toán lớn. Việc tính toán được thực hiện qua các vòng lặp thay vì đệ quy.

Ưu điểm

- **Tốc độ nhanh hơn:** Không cần đệ quy, nên phương pháp này thường chạy nhanh hơn và tiết kiệm bộ nhớ hơn vì nó không phải giữ các cuộc gọi đệ quy trong bộ nhớ stack.
- **Khả năng tối ưu cao hơn:** Phương pháp này cho phép áp dụng các tối ưu hóa khác như sử dụng cấu trúc dữ liệu hiệu quả hoặc tính toán tuần tự, làm tăng hiệu quả và tốc độ xử lý.

Nhược điểm

- **Khó triển khai hơn:** Đôi khi, việc chuyển bài toán thành mô hình Bottom-up đòi hỏi phải xây dựng bảng phương án một cách kỹ lưỡng, và quá trình này không dễ như việc cài đặt đệ quy Top-down.

- **Khó trực quan hóa:** Phương pháp này có thể không trực quan như Top-down, vì chúng ta không thể chia nhỏ bài toán ra một cách rõ ràng và tự nhiên như trong đệ quy.

Kết luận: Nên chọn phương pháp nào?

Cả hai phương pháp đều có điểm mạnh và điểm yếu. Tùy vào bài toán và yêu cầu của bạn mà lựa chọn phương pháp phù hợp. **Top-down** phù hợp khi bài toán có tính chất phân tách rõ ràng và bạn muốn một cách triển khai dễ hiểu, nhanh chóng. Tuy nhiên, nếu bài toán lớn và yêu cầu tối ưu về bộ nhớ và hiệu suất, **Bottom-up** sẽ là lựa chọn tốt hơn vì nó tiết kiệm bộ nhớ và tốc độ xử lý nhanh hơn.

Lý do chọn phương pháp Bottom-up

Lý do đơn giản là vì **hiệu suất**. Trong nhiều trường hợp thực tế, Bottom-up cho kết quả nhanh hơn và tiết kiệm tài nguyên hơn, đặc biệt khi bài toán có kích thước lớn. Việc không phải lo lắng về việc tràn bộ nhớ stack hay tính toán lại giúp bạn tập trung vào việc tối ưu các phần khác của giải thuật. Tuy nhiên, đôi khi **Top-down** vẫn là sự lựa chọn tốt trong những bài toán nhỏ hoặc khi cần triển khai nhanh chóng và trực quan.

2 Bài tập Thực hành

2.1 Bài toán: Chú ếch nhảy

2.1.1 Đề bài:

- Có n hòn đá được đánh số từ 1 đến n, với độ cao tương ứng là $h_1, h_2, \ldots, h_n$.
- Ban đầu, chú ếch ở hòn đá thứ nhất. Ếch có thể nhảy đến các hòn đá trong phạm vi $i+1$ đến $i+k$.
- Phí nhảy từ hòn đá i đến hòn đá j là $|h_i - h_j|$.
- Hãy tìm chi phí tối thiểu để chú ếch nhảy từ hòn đá thứ nhất đến hòn đá thứ n.

2.1.2 Ý tưởng

- Sử dụng phương pháp **quy hoạch động (dynamic programming)**.
- Gọi $dp[i]$ là chi phí tối thiểu để đến hòn đá thứ i.
- **Công thức chuyển trạng thái:**

$$dp[i] = \min_{j \in [i-k, i-1]} \big(dp[j] + |h_i - h_j|\big) \quad \text{với } j \geq 1.$$

- **Khởi tạo:**

$$dp[1] = 0, \quad dp[2..n] = +\infty.$$

- **Kết quả:**

$$dp[n] \quad \text{là chi phí tối thiểu cần tìm.}$$

2.1.3 Mã giả

```
Input:
    n: số lượng hòn đá
    k: khoảng cách tối đa chú ếch có thể nhảy
    h[1..n]: mảng độ cao của các hòn đá
Output:
    Chi phí tối thiểu để đến hòn đá cuối cùng (dp[n])

Thuật toán:
1. Khởi tạo mảng dp:
    dp[1] = 0   // Chi phí tại hòn đá đầu tiên là 0
    dp[2..n] = dương vô cùng // Chi phí ban đầu tại các hòn đá khác là vô cực
2. Duyệt qua từng hòn đá i từ 2 đến n:
    a. Duyệt qua các hòn đá j trong phạm vi tối đa k trước i:
        j thuộc [max(1, i-k), i-1]
        dp[i] = min(dp[i], dp[j] + |h[i] - h[j]|)
3. Kết quả cuối cùng là dp[n].
```

2.1.4 Tính toán độ phức tạp

- **Thời gian:**
  - Vòng lặp ngoài chạy từ 2 đến n: $O(n)$.
  - Vòng lặp trong chạy tối đa k lần: $O(k)$.
  - Tổng độ phức tạp: $O(n \cdot k)$.
- **Không gian:** $O(n)$ do sử dụng mảng dp.

2.1.5 Code Python

```python
def min_cost_to_last_stone(n, k, h):
    # Khoi tao mang dp
    dp = [float('inf')] * n
    dp[0] = 0  # Chi phi de o hon da dau tien la 0

    # Duyet tung hon da
    for i in range(1, n):
        # Xet cac hon da truoc do trong pham vi nhay duoc
        for j in range(max(0, i - k), i):
            dp[i] = min(dp[i], dp[j] + abs(h[i] - h[j]))

    # Ket qua la dp[n-1]
    return dp[-1]


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

# Xuat output
print(min_cost_to_last_stone(n, k, h))
```

2.1.6 Ví dụ minh họa

- **Input:**

$$n = 5, \quad k = 3, \quad h = [10, 30, 40, 50, 20]$$

- **Bảng dp:**

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

- **Output:** 30.

2.1.7 Kết luận

- Thuật toán quy hoạch động là giải pháp hiệu quả cho bài toán "Chú Ếch".
- Với giới hạn $n \leq 10^5$ và $k \leq 100$, thuật toán $O(n \cdot k)$ là đủ tốt.
- Kỹ thuật tối ưu hóa hàng đợi giảm dần có thể được áp dụng để nâng cao hiệu năng.