Lớp: CS112.P11.CTTN
Nhóm: 14
Sinh viên: Lê Nguyễn Anh Khoa. MSSV: 23520742
Sinh viên: Cáp Kim Hải Anh. MSSV: 23520036

BÀI TẬP
Phương pháp thiết kế thuật toán Dynamic programming

I. Bài tập lý thuyết

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

- Không phải mọi bài toán đều có thể giải quyết bằng quy hoạch động. Tuy quy hoạch động là một phương pháp mạnh nhưng không thể giải quyết với mọi bài toán
- Quy hoạch động chỉ phù hợp với các bài toán có những đặc điểm sau:
- Có cấu trúc con tối ưu (Optimal substructure): Nghiệm tối ưu của bài toán lớn có thể được xây dựng từ nghiệm tối ưu của các bài toán con.
- Có các bài toán con chồng chéo (Overlapping subproblems): Quá trình giải bài toán lớn sẽ gặp lại các bài toán con nhiều lần.
- Nhiều loại bài toán không thể giải bằng quy hoạch động, ví dụ:
- Các bài toán NP-khó như Traveling Salesman Problem
- Các bài toán không có cấu trúc đệ quy rõ rang
- Các bài toán đòi hỏi tìm kiếm toàn cục hoặc có nhiều điều kiện phức tạp

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

- Bài toán cắt thanh thép (Steel Cutting Problem).
- Bối cảnh:
- Một nhà máy có thanh thép dài N mét
- Có bảng giá các đoạn thép ngắn hơn với độ dài khác nhau
- Cần tìm cách cắt thanh thép để thu được lợi nhuận cao nhất
- Cách tiếp cận:
- Xác định công thức quy hoạch động:

  F(n) = max(price[i] + F(n-i)) với i từ 1 đến n

  Trong đó:

  + F(n) là lợi nhuận tối đa khi có thanh thép dài n

  + price[i] là giá của đoạn thép dài i.

- Code minh họa:

```python
def cut_rod(price, n):
    dp = [0] * (n + 1)

    for i in range(1, n + 1):
        max_val = -1
        for j in range(i):
            max_val = max(max_val, price[j] + dp[i-j-1])
        dp[i] = max_val

    return dp[n]
```

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 (Memoization):

- Cách hoạt động: Xuất phát từ bài toán lớn, gọi đệ quy để giải các bài toán con và lưu kết quả vào một cache để tránh lặp lại quá trình tính toán.

- Ưu điểm:

- Dễ implement vì theo tư duy tự nhiên của đệ quy, dễ hiểu
- Dễ cài đặt và linh động
- Chỉ tính toán các trạng thái thực sự cần thiết
- Dễ debug và theo dõi quá trình thực thi
- Phù hợp với bài toán có nhiều trạng thái nhưng không cần dùng hết

- Nhược điểm:

- Tốn bộ nhớ stack do đệ quy
- Có thể bị stack overflow với input lớn
- Overhead của việc gọi đệ quy nhiều lần
- Khó tối ưu
- Lãng phí thời gian nếu một bài toán nhỏ bị gọi nhiều lần không cần thiết.

Bottom-up (Tabulation):

- Cách hoạt động: Xuất phát từ bài toán con nhỏ nhất, việc tính toán thực hiện qua vòng lặp thay vì đệ quy theo thứ tự từ dưới lên và lưu kết quả trực tiếp vào bảng.

- Ưu điểm:

- Tốc độ thực thi nhanh hơn do không có overhead của đệ quy

- Không bị giới hạn bởi stack size
- Dễ tối ưu bộ nhớ (có thể chỉ cần lưu vài trạng thái gần nhất)

- Nhược điểm:

- Khó implement hơn, đặc biệt với bài toán phức tạp
- Khó trực quan hoá hơn Top-down
- Phải tính tất cả trạng thái, kể cả những trạng thái không cần thiết
- Code thường dài và khó đọc hơn
- Khó thực hiện hơn nếu không quen với cách tiếp cận vòng lặp

Mình thường ưu tiên Bottom-up vì:

- Hiệu năng ổn định và tốt hơn với input lớn
- Trong hầu hết trường hợp thực tế, bottom-up cho kết quả nhanh hơn và tiết kiệm tài nguyên hơn, đặc biệt với bài toán kích thước lớn
- Dễ tối ưu bộ nhớ
- An toàn hơn khi deploy (không lo stack overflow)

Tuy nhiên, với các bài toán:

- Cần prototype nhanh
- Logic phức tạp
- Không yêu cầu hiệu năng cao

  -> Mình sẽ chọn **Top-down** để code và debug dễ dàng hơn.

II. Bài tập thực hành

- **Ý tưởng: Sử dụng quy hoạch động:**

- dp[i]: chi phí ít nhất để nhảy đến hòn đá thứ i.
- Khởi tạo: dp[1] = 0 và dp[2..n] = oo
- Duyệt i từ 1 đến n
- Duyệt j từ i-1 đến i-k
- Công thức chuyển trạng thái:

  Cập nhật dp[i]: dp[i] = min(dp[i], dp[j] + |h(i) – h(j)|
- Đáp án là dp[n].

- **Mã giả:**

```
input: n, k, h[1..n]
initialize dp[1] = 0, dp[2..n] = +oo
for i = 1 to n-1:
     for j = i+1 to min(n, i+k):
          dp[j] = min(dp[j], dp[i] + |h[i] – h[j]|)
     output: dp[n]
```

- **Độ phức tạp:**

- Thời gian: O(n*k) (duyệt qua n hòn đá, với mỗi hòn đá xét tối đa k bước)
- Không gian: O(n) (sử dụng mảng dp)

- **Code python:**

```
n, k = map(int, input().split())
h = list(map(int, input().split()))
dp = [10000000000000000] * n
dp[0] = 0
for i in range(n):
    for j in range(i+1, min(n, i+k+1)):
        dp[j] = min(dp[j], dp[i] + abs(h[i] - h[j]))
print(dp[n-1])
```

- **Nhận xét:**

- Sử dụng thuật toán quy hoạch động là phù hợp
- Với độ phức tạp O(n*k), với $n \leq 10^5$ và $k \leq 100$ là phù hợp
- Để nâng cao hiệu năng có thể sử dụng kỹ năng tối ưu hoá hàng đợi giảm dần