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

BÀI TẬP MÔN HỌC
PHÂN TÍCH THIẾT KẾ THUẬT TOÁN

Sinh viên: Đỗ Phương Duy - 23520362

Sinh viên: Nguyễn Nguyên Khang - 22520623

Ngày 1 tháng 12 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?

- Không phải bài toán nào cũng có thể giải bằng quy hoạch động. Một bài toán muốn giải được bằng quy hoạch động phải đảm bảo có hai yếu tố: cấu trúc con tối ưu và các bài toán con gối nhau.
- Ví dụ: bài toán tìm phần tử lớn nhất trong mảng, sử dụng phép tìm kiếm tuyến tính, có thể thấy bài toán này không thể chia thành các bài toán con, mỗi lần duyệt qua một phần tử là mỗi lần duyệt độc lập, không có sự chồng gối giữa các lần duyệt hoặc kết quả của chúng.

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

- Bài toán tìm tuyến đường ngắn nhất
  - Chúng ta có một danh sách các địa điểm cần đi qua (tạp hóa, bưu điện, hiệu thuốc) và muốn tìm tối ưu hóa quãng đường di chuyển
  - Trước khi cố định duy nhất một tuyến đường, chúng ta sẽ trải nghiệm 1 vài tuyến đường khả thi trước khi đi đến kết luận.
  - Các bài toán con: Với mỗi điểm đến, chúng ta sẽ tính toán khoảng cách ngắn nhất để đến đó từ các địa điểm trước.
  - Tính chất gối nhau của các bài toán con: Nếu chúng ta đi qua cùng 1 nơi trên các lộ trình khác nhau, chúng ta có thể tái sử dụng khoảng cách ngắn nhất thay vì tính toán lại từ đầu.
  - Cấu trúc con tối ưu: tổng lộ trình ngắn nhất cuối cùng sẽ được xây dựng từ khoảng cách ngắn nhất giữa các điểm dừng với nhau.

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:**
- **Ưu điểm:**
  - Dễ triển khai hơn: chỉ cần thêm bộ nhớ để lưu trữ các bài toán con đã tính
  - Cấu trúc rõ ràng cho một số bài toán: các bài toán dựa trên cây khi đệ quy là lựa chọn tự nhiên thì Top-down có thể dễ dàng được triển khai và dễ hiểu hơn.
  - Chỉ tính toán các bài toán con cần thiết, có thể tiết kiệm thời gian tính toán nếu không cần tính tất cả các bài toán con
- **Nhược điểm:**
  - Chi phí bộ nhớ lớn: có thể gây tràn bộ nhớ nếu thuật toán sử dụng đệ quy rất sâu
  - Chạy chậm hơn: chi phí đệ quy và chi phí lưu trữ kết quả trung gian có thể chậm hơn so với cách tiếp cận thông thường

- **Phương pháp Bottom up:**
- **Ưu điểm:**
  - Tiết kiệm bộ nhớ hơn: không sử dụng ngăn xếp gọi hàm như trong đệ quy
  - Hiệu suất hơn hơn: Không lo ngại tràn bộ nhớ đệ quy, không phải quản lý đệ quy và ngăn xếp
- **Nhược điểm:**
  - Khó triển khai: phải xây dựng một cấu trúc dữ liệu tính toán tất cả các bài toán con trước khi giải quyết bài toán chính
  - Tốn chi phí bộ nhớ cho tất cả các bài toán con nếu số lượng bài toán con là quá lớn
- **Ưu tiên sử dụng:**
  - Khi cần tối ưu hiệu suất: Bottom-up
  - Khi cấu trúc đơn giản và rõ ràng: Bottom-up
  - Khi muốn triển khai nhanh và dễ dàng: Top-down
  - Khi không cần giải quyết tất cả bài toán con ngay lập tức hoặc số lượng bài toán con không quá lớn: Top-down

2 Bài tập thực hành:

2.1 Chú ếch

2.1.1 Ý tưởng

- Gọi f[i] là chi phí thấp nhất để nhảy từ đỉnh 1 đến đỉnh i.
- Công thức truy hồi:

$$f[i] = min(f[i], f[i-j] + abs(a[i] - a[i-j])$$

- Điều kiên ban đầu: $f[0] = 0$ vì con ếch ban đầu xuất phát ở ô đầu tiên nên không tốn chi phí.

2.1.2 Mã giả

```
function minCostJump(a: array of int)
    n = length(a)
    f = array of size n
    f[0] = 0
    INF = 1000000000000000000
    for i = 1 to n - 1:
        f[i] = INF
        for j = 1 to min(i, k):
            f[i] = min(f[i], f[i - j] + abs(a[i] - a[j])
    return f[n - 1]
```

2.1.3 Độ phức tạp

Thời gian: $O(n * k)$
Không gian: $O(n)$

2.1.4 Code

```
import math

n, k = [int (i) for i in input().split()]
a = [int (i) for i in input().split()]

f = []
INF = 1000000000000000000
f.append(0)
for i in range (1, n, 1):
    f.append(INF)
    x = min(k, i)
    for j in range(1, x + 1, 1):
        f[i] = min(f[i], f[i - j] + abs(a[i] - a[i - j]))
print(f[n - 1])
```

Tài liệu