ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Recursive Algorithm Complexity Analysis

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 7 tháng 11 năm 2024

Mục lục

1 Bài toán 1: Tìm tất cả các tổ hợp của các số theo thứ tự

Mô tả bài toán: Cho một chuỗi ký tự gồm các chữ số. Yêu cầu là tìm tất cả các tổ hợp của các chữ số này theo đúng thứ tự trong chuỗi.

Giải pháp:

- **Ý tưởng giải**: Sử dụng đệ quy để xây dựng tất cả các tổ hợp của chuỗi. Tại mỗi vị trí, có hai lựa chọn: chọn hoặc bỏ qua ký tự đó. Với mỗi tổ hợp, bắt đầu từ vị trí đầu tiên và tiếp tục cho đến khi hết chuỗi.

Code:

```
def generate_combinations(s, index=0, current=""):
    if index == len(s):
        print(current)
        return
    # Không chọn ký tự hiện tại
    generate_combinations(s, index + 1, current)
    # Chọn ký tự hiện tại
    generate_combinations(s, index + 1, current + s[index])


# Ví dụ sử dụng
s = "123"
generate_combinations(s)
```

Độ phức tạp: Với mỗi ký tự, có hai lựa chọn nên tổng số tổ hợp là 2^n, với n là độ dài của chuỗi. Do đó, độ phức tạp là $O(2^n)$.

2 Bài toán 2: Tạo tất cả các chuỗi có độ dài k từ một tập hợp ký tự

Mô tả bài toán: Cho một tập hợp ký tự và một số nguyên k. Hãy tạo ra tất cả các chuỗi có độ dài k từ tập ký tự này.

Giải pháp:

- **Ý tưởng giải**: Sử dụng đệ quy để xây dựng chuỗi với độ dài k. Ở mỗi vị trí trong chuỗi, lần lượt chọn từng ký tự từ tập hợp.

Code:

```
def generate_strings(chars, k, current=""):
    if k == 0:
        print(current)
        return
    for char in chars:
        generate_strings(chars, k - 1, current + char)


# Ví dụ sử dụng
chars = ['a', 'b', 'c']
k = 2
generate_strings(chars, k)
```

Độ phức tạp: Tại mỗi vị trí có n lựa chọn, tổng số chuỗi tạo ra là n^k. Do đó, độ phức tạp là $O(n^k)$.

3 Bài toán 3: Tìm tổ hợp các ước số của một số n

Mô tả bài toán: Tìm tất cả các tổ hợp của các ước số của một số nguyên n.

Giải pháp:

- **Ý tưởng giải**: Sử dụng đệ quy để thử từng ước của n, sau đó đệ quy với phần còn lại sau khi chia. Khi giá trị còn lại là 1, thì lưu lại tổ hợp hiện tại.

Code:

```
def find_factors_combinations(n, start=2, current=[]):
    if n == 1:
        print(current)
        return
    for i in range(start, n + 1):
        if n % i == 0:
            find_factors_combinations(n // i, i, current + [i])


# Ví dụ sử dụng
n = 12
find_factors_combinations(n)
```

Độ phức tạp: Số lượng ước của n là $O(\sqrt{n})$, do phải duyệt qua các tổ hợp của chúng, độ phức tạp tăng lên $O(2^{\sqrt{n}})$.

4 Bài toán 4: Biểu diễn số x dưới dạng tổng các lũy thừa

Mô tả bài toán: Cho hai số x và n. Tìm cách biểu diễn x dưới dạng tổng của các lũy thừa n-th của các số tự nhiên khác nhau.

Giải pháp:

- **Ý tưởng giải**: Dùng đệ quy để thử từng số lũy thừa n từ 1 trở đi, trừ giá trị này từ x và tiếp tục đệ quy với phần còn lại. Khi phần còn lại là 0, đã tìm thấy một cách biểu diễn.

Code:

```
def count_ways(x, n, num=1):
```

```
    power = num ** n
    if power > x:
        return 0
    elif power == x:
        return 1
    # Đệ quy với trường hợp bao gồm và không bao gồm ‘num‘
    return count_ways(x - power, n, num + 1) + count_ways(x, n, num + 1)

# Ví dụ sử dụng
x = 10
n = 2
print(count_ways(x, n))
```

Độ phức tạp: Độ phức tạp thời gian là $O(2^x)$ vì có nhiều cách biểu diễn x bằng tổng các lũy thừa.

5 Bài toán 5: Tháp Hà Nội

Mô tả bài toán: Giải bài toán tháp Hà Nội với n đĩa và 3 cọc, yêu cầu chuyển toàn bộ đĩa từ cọc nguồn đến cọc đích.

Giải pháp:

- **Ý tưởng giải**: Chuyển $n-1$ đĩa từ cọc nguồn sang cọc trung gian, sau đó di chuyển đĩa lớn nhất sang cọc đích và tiếp tục di chuyển $n-1$ đĩa từ cọc trung gian sang cọc đích.

Code:

```
def tower_of_hanoi(n, source, target, auxiliary):
    if n == 1:
        print(f"Move disk 1 from {source} to {target}")
        return
    tower_of_hanoi(n - 1, source, auxiliary, target)
```

```
    print(f"Move disk {n} from {source} to {target}")
    tower_of_hanoi(n - 1, auxiliary, target, source)

# Ví dụ sử dụng
n = 3
tower_of_hanoi(n, 'A', 'C', 'B')
```

Độ phức tạp: Cần $2^n - 1$ bước để hoàn thành, do đó độ phức tạp là $O(2^n)$.