TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

RECURSIVE ALGORITHM COMPLEXITY ANALYSIS

Môn học: Phân tích và thiết kế thuật toán

Nhóm 9

Sinh viên thực hiện:

Họ và tên	MSSV
Bùi Ngọc Thiên Thanh	23521436
Nguyễn Thái Sơn	23521356

Thành phố Hồ Chí Minh, 2024

Mục lục

1 Problem 1

1.1 Tóm tắt bài toán

- **Đầu vào**: Một chuỗi các chữ số.
- **Yêu cầu**: In ra tất cả các tổ hợp số có thể tạo được từ chuỗi đã cho nhưng giữ nguyên thứ tự các chữ số.

1.2 Ý tưởng giải

- Sử dụng **đệ quy** hoặc **backtracking** để tạo ra tất cả các tổ hợp.
- Với mỗi lần đệ quy, quyết định có chọn hoặc bỏ qua chữ số hiện tại.
- Lặp lại cho đến khi duyệt hết chuỗi.

1.3 Phân tích độ phức tạp

- Với mỗi chữ số, có hai lựa chọn: **chọn** hoặc **bỏ qua**.
- Tổng số khả năng là 2^n, với n là độ dài của chuỗi.

Kết luận: Độ phức tạp thời gian là $O(2^n)$.

1.4 Code

```python
def combinations(nums, start=0, current=""):
    if current:
        print(current)
    for i in range(start, len(nums)):
        combinations(nums, i + 1, current + nums[i])

# Example usage
combinations("123")
```

2 Problem 2

2.1 Tóm tắt bài toán

- **Đầu vào**: Một tập ký tự và một số nguyên dương k.
- **Yêu cầu**: In ra tất cả các chuỗi có độ dài k được tạo thành từ tập ký tự cho trước.

2.2 Ý tưởng giải

- Sử dụng **đệ quy** để tạo ra các chuỗi có thể có.
- Với mỗi vị trí trong chuỗi, thử đặt từng ký tự từ tập ký tự vào đó.
- Lặp lại cho đến khi chuỗi có đủ k ký tự.

2.3 Phân tích độ phức tạp

- Với tổng cộng k vị trí, và mỗi vị trí có n lựa chọn ký tự.
- Tổng số khả năng là n^k.

Kết luận: Độ phức tạp thời gian là $O(n^k)$.

2.4 Code

```python
def generate_strings(chars, k, current=""):
    if len(current) == k:
        print(current)
        return
    for char in chars:
        generate_strings(chars, k, current + char)

# Example usage
generate_strings(['a', 'b', 'c'], 2)
```

3 Problem 3

3.1 Tóm tắt bài toán

- **Đầu vào**: Một số nguyên n.
- **Yêu cầu**: In ra tất cả các tổ hợp thừa số của n.

3.2 Ý tưởng giải

- Sử dụng **đệ quy** để tìm các thừa số của n từ 2 đến n.
- Với mỗi thừa số, chia n cho thừa số đó và tiếp tục tìm các thừa số của phần còn lại.
- Dừng lại khi tích của các thừa số bằng n.

3.3 Phân tích độ phức tạp

- Tổng số tổ hợp phụ thuộc vào số lượng thừa số của n.
- Phức tạp gần như $O(2^{\log n})$.

Kết luận: Độ phức tạp thời gian là $O(2^{\log n})$.

3.4 Code

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

# Example usage
factor_combinations(12)
```

4 Problem 4

4.1 Tóm tắt bài toán

- **Đầu vào**: Hai số nguyên x và n.
- **Yêu cầu**: Tìm số cách biểu diễn x dưới dạng tổng các lũy thừa bậc n của các số tự nhiên.

4.2 Ý tưởng giải

- Sử dụng **đệ quy** để thử tất cả các số tự nhiên a sao cho $a^n \leq x$.
- Trừ phần đã xét đi và tiếp tục tìm tổng các lũy thừa còn lại.
- Dừng lại khi a^n vượt quá phần còn lại của x.

4.3 Phân tích độ phức tạp

- Số lũy thừa cần thử gần với $O(x^{1/n})$.

Kết luận: Độ phức tạp thời gian là $O(x^{1/n})$.

4.4 Code

```python
def sum_of_powers(x, n, num=1):
    power = num ** n
    if power > x:
        return 0
    if power == x:
        return 1
    return sum_of_powers(x - power, n, num + 1) +
        sum_of_powers(x, n, num + 1)

# Example usage
sum_of_powers(10, 2)
```

5 Problem 5

5.1 Tóm tắt bài toán

- **Đầu vào**: Số đĩa n và 3 cọc.
- **Yêu cầu**: Chuyển toàn bộ n đĩa từ cọc nguồn sang cọc đích theo quy tắc:
  - Chỉ được di chuyển một đĩa mỗi lần.
  - Không được đặt đĩa lớn lên trên đĩa nhỏ hơn.

5.2 Ý tưởng giải

- Sử dụng **đệ quy** để giải bài toán:
  1. Chuyển $n - 1$ đĩa từ cọc nguồn sang cọc trung gian.
  2. Chuyển đĩa lớn nhất sang cọc đích.
  3. Chuyển $n - 1$ đĩa từ cọc trung gian sang cọc đích.

5.3 Phân tích độ phức tạp

- Số bước di chuyển đĩa tuân theo công thức $2^n - 1$.

Kết luận: Độ phức tạp thời gian là $O(2^n)$.

5.4 Code

```python
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)

# Example usage
tower_of_hanoi(3, 'A', 'C', 'B')
```