ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Phương pháp thiết kế Divide, decrease and transform and conquer

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 Chia để trị (Divide and Conquer)

Khái niệm: Phương pháp chia để trị (Divide and Conquer) là chia nhỏ bài toán lớn thành các bài toán con có kích thước nhỏ hơn. Sau khi giải được các bài toán con, ta kết hợp kết quả để có lời giải cho bài toán lớn.

Ví dụ: Tìm kiếm nhị phân (Binary Search).

Giải thích cách áp dụng:

- Đầu tiên, chia mảng thành hai phần bằng cách chọn chỉ số giữa làm mốc.
- So sánh giá trị ở giữa với giá trị cần tìm:
  - Nếu bằng nhau, giá trị ở giữa chính là kết quả.
  - Nếu nhỏ hơn, tìm ở nửa bên trái.
  - Nếu lớn hơn, tìm ở nửa bên phải.
- Lặp lại cho đến khi tìm thấy hoặc kết thúc với một nửa không có giá trị cần tìm.

Mã giả:

```
def binary_search(arr, left, right, x):
    if left > right:
        return -1  # Không tìm thấy
    mid = (left + right) // 2
    if arr[mid] == x:
        return mid  # Tìm thấy vị trí
    elif arr[mid] > x:
        return binary_search(arr, left, mid - 1, x)
    else:
        return binary_search(arr, mid + 1, right, x)
```

Ứng dụng: Áp dụng để tìm kiếm nhanh trong các danh sách đã sắp xếp như từ điển, danh sách khách hàng, bảng điểm.

2 Giảm để trị (Decrease and Conquer)

Khái niệm: Kỹ thuật này giải bài toán bằng cách giảm kích thước của bài toán gốc một cách từ từ và xử lý các phần tử nhỏ hơn trước khi đến bài toán lớn.

Ví dụ: Sắp xếp chèn (Insertion Sort).

Giải thích cách áp dụng:

- Bắt đầu với giả định rằng phần đầu của mảng đã sắp xếp.
- Lấy phần tử tiếp theo và chèn nó vào vị trí thích hợp trong phần đã sắp xếp sao cho mảng vẫn được sắp xếp.
- Lặp lại cho đến khi tất cả các phần tử đều được sắp xếp.

Mã giả:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
```

Ứng dụng: Sắp xếp các danh sách nhỏ hoặc danh sách gần như sắp xếp.

3 Biến đổi để trị (Transform and Conquer)

Khái niệm: Biến đổi để trị là kỹ thuật biến đổi bài toán thành dạng khác giúp giải quyết bài toán dễ dàng hơn.

Ví dụ: Cây tìm kiếm nhị phân (Binary Search Tree - BST).

Giải thích cách áp dụng:

- Chuyển danh sách thành cây nhị phân tìm kiếm (BST).
- Duyệt qua cây để tìm kiếm phần tử một cách nhanh chóng.

Mã giả:

```
def transform_and_conquer_search(arr, x):
    bst = build_bst(arr)  # Xây dựng cây nhị phân tìm kiếm từ danh sách
    return search_bst(bst, x)  # Tìm kiếm phần tử x trong BST
```

Ứng dụng: Tối ưu hóa việc tìm kiếm và quản lý dữ liệu trong các hệ thống lớn.

4 Bài toán tổng lũy thừa

Đề bài

Cho hai số nguyên x và n (với $x \leq 10^{18}$ và $n \leq 10^{18}$), tính tổng:

$$S = x^0 + x^1 + x^2 + \cdots + x^n$$

Ví dụ: Với $x = 5$ và $n = 5$, ta có $S = 3906$.

Cách giải và mã giả

Cách 1 - Brute Force

- **Kỹ thuật**: Sử dụng phương pháp cộng trực tiếp từng lũy thừa của x từ x^0 đến x^n.

Mã giả:

```
def sum_of_powers(x, n):
    S = 0
    for i in range(n + 1):
        S += x ** i
    return S
```

Cách 2 - Lũy thừa nhanh (Fast Exponentiation)

- **Kỹ thuật**: Áp dụng chia để trị (Divide and Conquer) để tính nhanh từng lũy thừa x^i và cộng lại.

Mã giả:

```
def fast_power(x, i):
    if i == 0:
        return 1
    y = fast_power(x, i // 2)
    if i % 2 == 0:
        return y * y
    else:
        return x * y * y


def sum_of_powers_fast(x, n):
    S = 0
    for i in range(n + 1):
        S += fast_power(x, i)
    return S
```

Cách 3 - Công thức cấp số nhân (Geometric Series Formula)

- **Kỹ thuật**: Sử dụng công thức tổng cấp số nhân để tính nhanh tổng S mà không cần phải tính từng lũy thừa.
- **Công thức**:

$$S = \frac{x^{n+1} - 1}{x - 1} \quad \text{nếu } x \neq 1$$

  Với $x = 1$, tổng $S = n + 1$.

Mã giả:

```
def sum_of_powers_formula(x, n):
    if x == 1:
        return n + 1
    else:
        return (fast_power(x, n + 1) - 1) // (x - 1)
```