ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BỘ MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Báo cáo

Đề tài: Divide/Decrease/Transform and Conquer

Môn học: Phân tích và thiết kế thuật toán

Sinh viên thực hiện:

Nguyen Minh Huy(23520634)

Do Quang Luc(23520902)

Giáo viên hướng dẫn:

Nguyen Thanh Son

Ngày 21 tháng 11 năm 2024

1. Divide and Conquer

Bài toán: Sắp xếp mảng bằng thuật toán Merge Sort

- **Mô tả bài toán**: Cho một mảng số nguyên, sắp xếp mảng theo thứ tự tăng dần.
- **Cách áp dụng**:
  - *Chia (Divide)*: Chia mảng ban đầu thành hai mảng con có kích thước xấp xỉ bằng nhau.
  - *Trị (Conquer)*: Đệ quy sắp xếp từng mảng con.
  - *Hợp (Combine)*: Gộp hai mảng con đã được sắp xếp thành một mảng hoàn chỉnh bằng cách so sánh và chọn phần tử nhỏ hơn từ mỗi mảng.
- **Ứng dụng thực tế**: Tối ưu hóa việc sắp xếp danh sách lớn trong cơ sở dữ liệu hoặc quản lý dữ liệu giao dịch tài chính.

2. Decrease and Conquer

Bài toán: Tìm kiếm nhị phân (Binary Search)

- **Mô tả bài toán**: Cho một mảng đã được sắp xếp và một số x, tìm chỉ số của x trong mảng (hoặc trả về -1 nếu không tồn tại).
- **Cách áp dụng**:
  - *Giảm bài toán (Decrease)*: Ở mỗi bước, so sánh x với phần tử giữa mảng:
    - Nếu x nhỏ hơn phần tử ở giữa, chỉ xét nửa mảng bên trái.
    - Nếu x lớn hơn phần tử ở giữa, chỉ xét nửa mảng bên phải.
  - Lặp lại quá trình cho đến khi tìm thấy x hoặc không còn mảng để xét.
- **Ứng dụng thực tế**: Tìm kiếm nhanh trong danh sách lớn, như tìm kiếm từ khóa trong từ điển hoặc cơ sở dữ liệu.

3. Transform and Conquer

Bài toán: Kiểm tra một số có phải là số nguyên tố

- **Mô tả bài toán**: Xác định xem số n có phải là số nguyên tố hay không.
- **Cách áp dụng**:
  - *Biến đổi bài toán (Transform)*: Thay vì kiểm tra tất cả các số từ 2 đến $n-1$, chỉ cần kiểm tra các ước từ 2 đến $\sqrt{n}$, vì mọi ước x lớn hơn $\sqrt{n}$ đều có một ước nhỏ hơn hoặc bằng $\sqrt{n}$.
  - *Giải bài toán đã được biến đổi*: Kiểm tra n có chia hết cho bất kỳ số nào từ 2 đến $\sqrt{n}$. Nếu không, n là số nguyên tố.
- **Ứng dụng thực tế**: Kiểm tra tính nguyên tố trong các thuật toán mã hóa RSA hoặc các hệ thống bảo mật dữ liệu.

Tóm tắt

- **Divide and Conquer**: Chia bài toán lớn thành các bài toán con, giải từng bài toán con và kết hợp kết quả (ví dụ: Merge Sort).
- **Decrease and Conquer**: Giảm kích thước bài toán qua từng bước và giải bài toán nhỏ hơn (ví dụ: Binary Search).
- **Transform and Conquer**: Biến đổi bài toán thành một dạng khác để giải quyết dễ hơn (ví dụ: Kiểm tra số nguyên tố).

Đề bài

Cho 2 số nguyên x và n ($x \leq 10^{18}, n \leq 10^{18}$). Tính tổng:

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

Ví dụ: với $x = 5$ và $n = 5$, thì $S = 3906$.

—

Cách 1: Duyệt tuần tự (Naive Approach)

Ý tưởng:

- Tính từng lũy thừa x^k từ $k = 0$ đến $k = n$ và cộng dần vào tổng S.
- Đây là cách làm đơn giản nhưng không tối ưu.

Ưu điểm:

- Dễ triển khai.
- Phù hợp khi n nhỏ.

Nhược điểm:

- Không khả thi khi n lớn vì số phép tính tăng tuyến tính với n.

Mã giả C++:

```cpp
// C++ Implementation
long long naive_sum(long long x, long long n) {
    long long S = 0;
    long long current_power = 1; // x^0 = 1
    for (long long i = 0; i <= n; ++i) {
        S += current_power;
        current_power *= x;
    }
    return S;
}
```

Độ phức tạp: $O(n)$.

—

Cách 2: Sử dụng công thức cấp số nhân

Ý tưởng:

- Tổng S của cấp số nhân được tính bằng công thức:

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

- Nếu $x = 1$, thì $S = n + 1$.

Ưu điểm:

- Hiệu quả khi n lớn, không cần tính tất cả lũy thừa.
- Độ phức tạp thấp: $O(\log(n))$ nhờ sử dụng lũy thừa nhanh.

Nhược điểm:

- Cần xử lý số học lớn để tránh tràn số.

Mã giả C++:

```cpp
// C++ Implementation
long long geometric_sum(long long x, long long n) {
    if (x == 1) {
        return n + 1;
    } else {
        return (pow(x, n + 1) - 1) / (x - 1);
    }
}
```

Độ phức tạp: $O(\log(n))$.

—

Cách 3: Sử dụng Divide and Conquer

Ý tưởng:

- Chia bài toán thành hai phần:

$$S_1 = x^0 + x^1 + \cdots + x^{\text{mid}}, \quad S_2 = x^{\text{mid}+1} + \cdots + x^n,$$

  với $\text{mid} = \lfloor n/2 \rfloor$.

- Tính S_1 bằng đệ quy. Sau đó, tính S_2 bằng cách nhân S_1 với $x^{\text{mid}+1}$ (do S_2 là dịch chuyển lũy thừa của S_1).

Ưu điểm:

- Phù hợp khi n rất lớn.
- Tối ưu hóa tính toán bằng cách giảm số lần tính lũy thừa.

Nhược điểm:

- Phức tạp hơn khi triển khai.

Mã giả C++:

```cpp
// C++ Implementation
long long divide_and_conquer_sum(long long x, long long n) {
    if (n == 0) return 1; //  T r   Íng   h p   c    b n
    long long mid = n / 2;
    long long S1 = divide_and_conquer_sum(x, mid);
    long long S2 = S1 * pow(x, mid + 1);
    if (n % 2 == 0) {
        return S1 + S2;
    } else {
        return S1 + S2 + pow(x, n);
    }
}
```

Độ phức tạp: $O(\log^2(n))$.

—

So sánh 3 cách giải

Cách giải	Kỹ thuật áp dụng	Độ phức tạp	Ưu điểm / Nhược điểm
Cách 1: Duyệt tuần tự	Không tối ưu	$O(n)$	Dễ triển khai; không khả thi khi n
Cách 2: Công thức toán học	Transform and Conquer	$O(\log(n))$	Hiệu quả; cần xử lý số học lớn
Cách 3: Divide and Conquer	Divide and Conquer	$O(\log^2(n))$	Tối ưu hóa tốt cho n lớn

—

Kết luận

- Cách 1 chỉ phù hợp với n nhỏ, đơn giản nhưng không hiệu quả.
- Cách 2 là lựa chọn tốt nhất nếu cần hiệu suất cao và tính toán chính xác.
- Cách 3 là phương pháp hiệu quả và thực tế nhất khi n rất lớn, nhờ chia nhỏ bài toán và tối ưu lũy thừa.