ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

BÀI TẬP VỀ NHÀ NHÓM 12

Môn học: Phân tích và thiết kế thuật toán

Sinh viên thực hiện:
Đặng Quốc Cường
Nguyễn Đình Thiên Quang
(Nhóm 1)

Giảng viên môn học:
Nguyễn Thanh Sơn

Ngày 6 tháng 12 năm 2024

Câu hỏi 1: Trình bày nguyên lý cơ bản của thuật toán quay lui. Tại sao thuật toán này thường được sử dụng để giải các bài toán tổ hợp?

Nguyên lý cơ bản của thuật toán quay lui (Backtracking):
Thuật toán quay lui là một phương pháp tìm kiếm thử và sai, trong đó thuật toán tìm kiếm lần lượt các lựa chọn khả dĩ trong một không gian tìm kiếm, và quay lại (backtrack) nếu phát hiện rằng một lựa chọn không mang lại kết quả mong muốn. Quá trình này tiếp tục cho đến khi tìm được một giải pháp hợp lệ hoặc tất cả các lựa chọn đều bị loại bỏ.
Các bước cơ bản của thuật toán quay lui là:

- **Tiến hành thử một lựa chọn:** Chọn một giá trị hoặc quyết định vào một bước tiếp theo.
- **Kiểm tra điều kiện hợp lệ:** Kiểm tra xem lựa chọn đó có hợp lệ không (ví dụ, không vi phạm ràng buộc).
- **Tiếp tục tìm kiếm:** Nếu hợp lệ, tiếp tục thử các lựa chọn tiếp theo. Nếu không hợp lệ, quay lại và thử các lựa chọn khác.

Tại sao thuật toán quay lui thường được sử dụng để giải các bài toán tổ hợp:
Thuật toán quay lui rất phù hợp cho các bài toán tổ hợp vì nó có thể hiệu quả trong việc tìm kiếm tất cả các tổ hợp hoặc phân hoạch của một tập hợp mà không phải thử tất cả các khả năng theo cách brute force. Bằng cách loại bỏ sớm các lựa chọn không hợp lệ hoặc không cần thiết, thuật toán quay lui giúp giảm độ phức tạp của bài toán tổ hợp, đặc biệt là trong các bài toán như:

- Tìm tất cả các cách chọn một số phần tử từ một tập hợp.
- Xếp hạng, phân loại.
- Các bài toán tìm kiếm đường đi trong đồ thị hoặc các bài toán có điều kiện ràng buộc.

Câu hỏi 2: So sánh điểm khác biệt chính giữa thuật toán nhánh cận và quay lui.

Sự khác biệt chính giữa thuật toán nhánh cận (Branch and Bound) và quay lui (Backtracking):

- **Cách tiếp cận:**
  - **Nhánh cận:** Sử dụng chiến lược tối ưu hóa để loại bỏ các nhánh không cần thiết từ không gian tìm kiếm. Nhánh cận sử dụng một hàm giới hạn (bound function) để đánh giá các nhánh và cắt bỏ nhánh nào mà không thể mang lại kết quả tốt hơn.
  - **Quay lui:** Quay lại khi gặp lựa chọn không hợp lệ hoặc không thể dẫn đến giải pháp mong muốn, không có bước cắt bỏ nhánh chủ động. Chỉ đơn giản là thử tất cả các khả năng mà không có sự tối ưu rõ ràng.
- **Độ phức tạp và tính hiệu quả:**

  - **Nhánh cận:** Được sử dụng khi bài toán có thể tối ưu hóa giải pháp (như bài toán quy hoạch động, bài toán tìm kiếm tối ưu). Nó có thể giảm đáng kể không gian tìm kiếm bằng cách loại bỏ các lựa chọn không khả thi từ trước.
  - **Quay lui:** Thường chỉ hữu ích trong các bài toán mà việc tìm kiếm tất cả các tổ hợp hoặc phân hoạch là cần thiết và không thể tối ưu hóa thêm.

- **Ứng dụng:**
  - **Nhánh cận:** Phù hợp với các bài toán tối ưu hóa (như bài toán bài toán bài toán đường đi ngắn nhất, bài toán lựa chọn tối ưu).
  - **Quay lui:** Thường dùng trong các bài toán tổ hợp hoặc các bài toán có nhiều điều kiện ràng buộc (như bài toán N-Queen, bài toán Sudoku).

Câu hỏi 3: Trình bày ưu điểm và nhược điểm của phương pháp Brute Force. Tại sao nó thường được xem là phương pháp kém hiệu quả trong các bài toán lớn?

Ưu điểm của phương pháp Brute Force:

- **Đơn giản và dễ hiểu:** Phương pháp Brute Force dễ triển khai và không yêu cầu các kỹ thuật phức tạp, vì chỉ cần thử tất cả các khả năng mà không phải tối ưu hóa gì.
- **Đảm bảo tính chính xác:** Nếu có giải pháp, Brute Force sẽ tìm ra nó, vì phương pháp này không bỏ sót bất kỳ khả năng nào. Đây là phương pháp "mạnh tay" trong các bài toán đơn giản.

Nhược điểm của phương pháp Brute Force:

- **Độ phức tạp cao:** Brute Force thường phải kiểm tra tất cả các khả năng có thể, dẫn đến thời gian chạy rất lâu. Ví dụ, với các bài toán có không gian tìm kiếm rất lớn, như bài toán tìm đường đi trong đồ thị, Brute Force có thể phải thử tất cả các tuyến đường có thể, dẫn đến độ phức tạp rất cao.
- **Không hiệu quả với các bài toán lớn:** Khi kích thước của bài toán tăng lên, số lượng khả năng phải kiểm tra sẽ tăng theo cấp số nhân hoặc theo một hàm đa thức rất lớn, điều này khiến Brute Force trở thành một phương pháp không khả thi cho các bài toán lớn trong một khoảng thời gian hợp lý.

Tại sao Brute Force thường được xem là phương pháp kém hiệu quả trong các bài toán lớn:

- **Không tối ưu:** Brute Force không có khả năng tối ưu hóa trong quá trình tìm kiếm, do đó nó không thể giảm số lượng trạng thái cần kiểm tra. Trong khi đó, các thuật toán như chia để trị, quy hoạch động hoặc nhánh cận có thể loại bỏ một phần lớn không gian tìm kiếm mà không cần phải thử mọi khả năng.
- **Thời gian chạy cao:** Khi kích thước bài toán tăng, thời gian chạy của Brute Force sẽ tăng lên rất nhanh và sẽ không thể giải quyết được các bài toán lớn trong một khoảng thời gian hợp lý.

Giải quyết bài toán 24 với C++

Ma C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <climits>

using namespace std;

// Ham tinh toan hai so voi phep toan
double calculate(double a, double b, char op) {
    if (op == '+') return a + b;
    if (op == '-') return a - b;
    if (op == '*') return a * b;
    if (op == '/') return (b != 0) ? a / b : INT_MIN;  // Tranh chia cho 0
    return INT_MIN;
}

// Ham kiem tra tat ca cac bieu thuc voi 4 the va tim gia tri lon nhat khong vuot qua 24
double find_max_value(vector<int>& nums) {
    double max_value = -INFINITY;  // Khoi tao gia tri lon nhat la -infinity
    vector<char> ops = {'+', '-', '*', '/'};

    // Duyet tat ca hoan vi cua 4 the
    do {
        // Duyet tat ca cac phep toan ket hop voi cac hoan vi
        for (char op1 : ops) {
            for (char op2 : ops) {
                for (char op3 : ops) {
                    // Cac cach nhom bieu thuc voi dau ngoac
                    double expr1 = calculate(calculate(calculate(nums[0], nums[1], op1), nu
                    double expr2 = calculate(calculate(nums[0], calculate(nums[1], nums[2],
                    double expr3 = calculate(nums[0], calculate(calculate(nums[1], nums[2],
                    double expr4 = calculate(nums[0], calculate(nums[1], calculate(nums[2],

                    // Cap nhat gia tri lon nhat neu nho hon hoac bang 24
                    if (expr1 <= 24 && expr1 > max_value) max_value = expr1;
                    if (expr2 <= 24 && expr2 > max_value) max_value = expr2;
                    if (expr3 <= 24 && expr3 > max_value) max_value = expr3;
                    if (expr4 <= 24 && expr4 > max_value) max_value = expr4;
                }
            }
        }
```

```
    } while (next_permutation(nums.begin(), nums.end()));  // Tien toi hoan vi tiep theo

    return (max_value == -INFINITY) ? 0 : max_value;  // Neu khong co bieu thuc hop le, tra
}

int main() {
    int n;
    cin >> n;  // Doc so bo bai

    while (n--) {
        vector<int> cards(4);
        for (int i = 0; i < 4; i++) {
            cin >> cards[i];  // Doc gia tri cua moi the
        }

        double result = find_max_value(cards);  // Tinh gia tri lon nhat tu cac the
        cout << result << endl;  // In ra ket qua
    }

    return 0;
}
```

Giai thich

- **Ham** `calculate`:
  - Thuc hien phep toan giua hai so `a` va `b` voi toan tu `op` (`+`, `-`, `*`, `/`).
  - Neu phep chia xay ra chia cho 0, ham tra ve `INT_MIN` de tranh loi chia cho 0.
- **Ham** `find_max_value`:
  - Thu tat ca cac hoan vi cua 4 the (su dung `next_permutation`).
  - Duyet tat ca cac phep toan `+`, `-`, `*`, `/` va cac cach nhom bieu thuc voi dau ngoac.
  - Tinh toan cac bieu thuc va cap nhat gia tri lon nhat thoa man dieu kien khong vuot qua 24.
- **Chuong trinh chinh** (`main`):
  - Doc so bo bai `n` va cac gia tri thẻ trong từng bộ bài.
  - Goi hàm `find_max_value` để tính toán và tìm giá trị lớn nhất có thể từ các thẻ.
  - In kết quả ra màn hình.

Vi du

Dau vao:

```
1
1
13
11
12
```

Dau ra:

```
24
```

Giai thich: Voi cac the {1, 13, 11, 12}, ta co the tao ra bieu thuc sau:

$$((1 \times 13) - 11) \times 12 = (13 - 11) \times 12 = 2 \times 12 = 24$$

Khi do, dap an chinh xac la **24**.