ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÀI TẬP BRUTE FORCE

Phan Nhật Tân - 23521405
Nguyễn Huy Phước - 23521234

Mục lục

1 Bài tập 1

1.1 Câu hỏi 1: Trình bày nguyên lý cơ bản của thuật toán quay lui (Backtracking). 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 thử sai có tổ chức nhằm tìm kiếm tất cả các lời giải khả thi cho một bài toán bằng cách:

- Xây dựng lời giải từng bước (từng phần của lời giải).
- Tại mỗi bước, kiểm tra tính khả thi (feasibility) của lời giải hiện tại:
  - Nếu lời giải tạm thời không thể dẫn đến lời giải đúng, **quay lui** (backtrack) về bước trước đó để thử một nhánh khác.
  - Nếu lời giải tạm thời là một phần của lời giải đúng, tiếp tục mở rộng (explore) đến bước tiếp theo.
- Tiếp tục đến khi tất cả các lựa chọn đã được thử.

Lý do sử dụng trong bài toán tổ hợp:

- **Tính hệ thống:** Quay lui tổ chức các bước tìm kiếm theo một cây trạng thái (state tree), dễ dàng duyệt qua tất cả các trường hợp.
- **Tối ưu hóa hiệu quả tìm kiếm:** Quay lui cho phép loại bỏ các nhánh không khả thi (pruning), giảm đáng kể số lượng trường hợp cần kiểm tra.
- **Tổng quát hóa:** Dễ dàng áp dụng cho nhiều bài toán tổ hợp như liệt kê hoán vị, tổ hợp, bài toán n-queens, sudoku, hoặc các bài toán lập lịch.

1.2 Câu hỏi 2: So sánh điểm khác biệt chính giữa thuật toán nhánh cận (Branch and Bound) và quay lui (Backtracking) khi tìm kiếm lời giải tối ưu

So sánh giữa Branch and Bound và Backtracking:

1.3 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:

- **Đơn giản và dễ triển khai:** Brute Force kiểm tra mọi trường hợp có thể, không đòi hỏi thuật toán phức tạp.
- **Tổng quát:** Phương pháp này có thể áp dụng cho bất kỳ bài toán nào có không gian tìm kiếm hữu hạn.
- **Tìm kiếm toàn diện:** Đảm bảo không bỏ sót bất kỳ lời giải nào.

Tiêu chí	**Backtracking (Quay lui)**	**Branch and Bound (Nhánh cận)**
Mục tiêu	Tìm kiếm tất cả các lời giải khả thi hoặc một lời giải đúng.	Tìm kiếm lời giải tối ưu cho bài toán tối ưu hóa (min/max).
Nguyên lý cắt tỉa	Loại bỏ các nhánh không khả thi dựa trên điều kiện ràng buộc.	Sử dụng cận trên và cận dưới để cắt bỏ các nhánh không khả thi.
Áp dụng	Bài toán tổ hợp, liệt kê (hoán vị, tổ hợp, n-queens).	Bài toán tối ưu như knapsack, TSP.
Hiệu quả cắt tỉa	Phụ thuộc vào kiểm tra tính khả thi của lời giải.	Hiệu quả hơn nhờ sử dụng thông tin cận trên/dưới.
Tính tổng quát	Dễ áp dụng cho nhiều loại bài toán.	Phù hợp hơn với bài toán tối ưu hóa có hàm mục tiêu.

Bảng 1: So sánh Branch and Bound và Backtracking

Nhược điểm:

- **Độ phức tạp tính toán cao:** Brute Force có độ phức tạp thường là hàm mũ hoặc giai thừa, dẫn đến thời gian chạy lâu khi không gian tìm kiếm lớn.
- **Không tối ưu:** Không sử dụng thông tin hoặc cấu trúc của bài toán để giảm số lượng trường hợp cần kiểm tra.

Lý do kém hiệu quả trong bài toán lớn:

- **Tăng trưởng theo cấp số nhân:** Khi kích thước bài toán tăng, số lượng trường hợp cần xét tăng rất nhanh, khiến phương pháp không khả thi.
- **Không cắt tỉa:** Phương pháp này không sử dụng chiến lược loại bỏ các trường hợp không khả thi, dẫn đến lãng phí tài nguyên tính toán.

2 Bài tập 2

2.1 Lời giải bài toán 24 - Trò chơi thẻ

Ý tưởng chính: Bài toán yêu cầu tìm giá trị lớn nhất không vượt quá 24 từ các phép toán cộng, trừ, nhân, chia và sử dụng tất cả bốn thẻ. Ta cần thử tất cả các cách sắp xếp và kết hợp các thẻ với các phép toán để kiểm tra xem kết quả nào phù hợp nhất.

Các bước giải:

1. **Sinh hoán vị các thẻ:** Với 4 thẻ, có tổng cộng $4! = 24$ cách sắp xếp các thẻ.
2. **Sinh các phép toán:** Với 3 phép toán giữa 4 thẻ, có $4^3 = 64$ tổ hợp phép toán (bao gồm +, -, *, /).
3. **Thử mọi cách đặt dấu ngoặc:** Có nhiều cách đặt dấu ngoặc để thay đổi thứ tự thực hiện các phép toán, ví dụ:

$$((a \circ b) \circ c) \circ d, \quad (a \circ (b \circ c)) \circ d, \quad a \circ ((b \circ c) \circ d), \ldots$$

4. **Kiểm tra tính hợp lệ:**
   - Kết quả trung gian và kết quả cuối cùng không được vượt quá 24.
   - Các phép chia phải có kết quả là số nguyên (bao gồm cả trung gian).
5. **Tìm giá trị lớn nhất:** Duyệt qua tất cả các biểu thức hợp lệ và lấy giá trị lớn nhất không vượt quá 24.

Giải thuật chi tiết:

Listing 1: Pseudo-code giải thuật

```
def max_expression(cards):
    max_val = float('-inf')
    for perm in permutations(cards):
        for ops in product(['+', '-', '*', '/'], repeat=3):
            for expr in generate_expressions(perm, ops):
                if is_valid(expr):
                    val = eval(expr)
                    if val <= 24:
                        max_val = max(max_val, val)
    return max_val if max_val != float('-inf') else -1
```

Triển khai từng bước:

- **Sinh hoán vị:** Sử dụng thuật toán sinh hoán vị (hoặc hàm có sẵn trong thư viện như ‘permutations‘ của Python).
- **Sinh tổ hợp phép toán:** Dùng hàm ‘product‘ với tập hợp các phép toán {+, -, *, /}.
- **Sinh biểu thức:** Sử dụng các mẫu biểu thức và thay thế các giá trị tương ứng.
- **Kiểm tra tính hợp lệ:** Đảm bảo phép chia có kết quả là số nguyên và không có lỗi (như chia cho 0).

Ví dụ minh họa:

- Với bộ bài {3, 3, 3, 1}:
$$((3 \times 3) + (3/1)) = 24$$
Giá trị lớn nhất không vượt quá 24 là 24.
- Với bộ bài {1, 1, 1, 1}:
$$(1 + 1) + (1 + 1) = 4$$
Giá trị lớn nhất không vượt quá 24 là 4.
- Với bộ bài {12, 5, 13, 1}:
$$((13 \times 1) + (12 - 5)) = 21$$
Giá trị lớn nhất không vượt quá 24 là 21.

Độ phức tạp:

- Số tổ hợp cần kiểm tra: $4! \times 4^3 \times k$, với k là số cách đặt dấu ngoặc.
- Mặc dù độ phức tạp cao, bài toán chỉ giới hạn với $N \leq 5$, nên thời gian xử lý chấp nhận được.