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

BÀI TẬP MÔN HỌC
PHÂN TÍCH THIẾT KẾ THUẬT TOÁN

Sinh viên: Đỗ Phương Duy - 23520362

Sinh viên: Nguyễn Nguyên Khang - 22520623

Ngày 1 tháng 12 năm 2024

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?

1.1.0.1 Nguyên lý cơ bản của thuật toán Backtracking:

- **Ý tưởng chính:** Thuật toán quay lui là phương pháp thử và kiểm tra toàn bộ các khả năng của một bài toán bằng cách lần lượt xây dựng lời giải, sau đó quay lui nếu gặp ngõ cụt.
- **Cách hoạt động:**
  - Tiến hành xây dựng lời giải một cách từng bước.
  - Tại mỗi bước, kiểm tra tính hợp lệ của lời giải tạm thời.
  - Nếu lời giải tạm thời không dẫn đến kết quả cuối cùng, quay lại bước trước đó để thử phương án khác.
- **Cấu trúc:**
  - Được biểu diễn dưới dạng đệ quy, với mỗi bước đại diện cho một trạng thái của lời giải.
  - Gồm hai phần chính:
    - **Điều kiện dừng:** Xác định khi nào thuật toán đã đạt được lời giải mong muốn.
    - **Khám phá không gian lời giải:** Thử từng phương án có thể tại mỗi bước.
- **Ví dụ:** Các bài toán thường sử dụng Backtracking: Sudoku, N-Queens, tổ hợp và hoán vị, phân chia tập hợp, ...

1.1.0.2 Tại sao Backtracking thường được sử dụng trong bài toán tổ hợp?

- Phù hợp với bài toán tổ hợp vì:
  - Thử nghiệm tất cả các khả năng mà không lưu trữ toàn bộ không gian trạng thái.
  - Dễ dàng loại bỏ các trạng thái không hợp lệ, giúp giảm số lượng trạng thái cần duyệt.

—

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

Tiêu chí	**Quay lui (Backtracking)**	**Nhánh cận (Branch and Bound)**
Mục tiêu	Khám phá toàn bộ không gian lời giải.	Tìm lời giải tối ưu trong không gian lời giải.
Nguyên lý	Thử và kiểm tra, quay lui khi cần thiết.	Sử dụng ràng buộc và cận để giảm không gi
Loại bài toán	Bài toán tổ hợp, bài toán quyết định.	Bài toán tối ưu hóa.
Hiệu quả	Có thể duyệt toàn bộ nếu không cắt tỉa tốt.	Giảm đáng kể không gian tìm kiếm.
Ví dụ	Sudoku, N-Queens.	Traveling Salesman Problem, Knapsack Pro

Bảng 1: So sánh giữa Backtracking và Branch and Bound

—

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?

1.3.0.1 Ưu điểm của Brute Force:

- **Đơn giản:** Dễ hiểu và dễ triển khai vì không yêu cầu thuật toán phức tạp.
- **Tổng quát:** Có thể áp dụng cho hầu hết các bài toán.
- **Đảm bảo tìm được lời giải:** Nếu không gian trạng thái đủ nhỏ, Brute Force chắc chắn tìm được lời giải chính xác.

1.3.0.2 Nhược điểm của Brute Force:

- **Không hiệu quả:** Thử toàn bộ các khả năng dẫn đến chi phí tính toán lớn.
- **Không khả thi cho bài toán lớn:** Tốn thời gian và tài nguyên khi không gian trạng thái lớn.
- **Không tận dụng ràng buộc:** Không áp dụng các cải tiến để giảm không gian tìm kiếm.

1.3.0.3 Tại sao Brute Force kém hiệu quả trong các bài toán lớn?

- Khi kích thước bài toán tăng, số lượng trạng thái tăng theo cấp số nhân (tính chất tổ hợp), dẫn đến thời gian chạy tăng rất nhanh.
- Ví dụ:
  - Bài toán N-Queens: $N!$ trạng thái.
  - Bài toán TSP: $(N-1)!/2$ trạng thái.

2 Bài tập 2:

2.1 Source code

```python
from itertools import permutations, product

# Các phép toán
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0 or a % b != 0:
        return None  # Chỉ chấp nhận kết quả chia nguyên
```

```
    return a // b

# Tất cả các phép toán
operations = [add, subtract, multiply, divide]

# Hàm tính giá trị lớn nhất từ các thẻ bài
def max_value_under_24(cards):
    max_value = float('-inf')

    # Hoán vị thứ tự thẻ bài
    for card_order in permutations(cards):
        # Tổ hợp các phép toán
        for ops in product(operations, repeat=3):
            # Tính giá trị cho từng cách đặt dấu ngoặc
            # (a op1 b) op2 (c op3 d)
            try:
                result1 = ops[1](ops[0](card_order[0], card_order[1]), ops[2](card_order[2],
                if result1 is not None and result1 <= 24:
                    max_value = max(max_value, result1)
            except:
                pass

            # ((a op1 b) op2 c) op3 d
            try:
                result2 = ops[2](ops[1](ops[0](card_order[0], card_order[1]), card_order[2])
                if result2 is not None and result2 <= 24:
                    max_value = max(max_value, result2)
            except:
                pass

            # (a op1 (b op2 c)) op3 d
            try:
                result3 = ops[2](ops[0](card_order[0], ops[1](card_order[1], card_order[2]))
                if result3 is not None and result3 <= 24:
                    max_value = max(max_value, result3)
            except:
                pass

            # a op1 ((b op2 c) op3 d)
            try:
                result4 = ops[0](card_order[0], ops[2](ops[1](card_order[1], card_order[2]),
                if result4 is not None and result4 <= 24:
                    max_value = max(max_value, result4)
            except:
                pass

            # a op1 (b op2 (c op3 d))
            try:
```

```
                result5 = ops[0](card_order[0], ops[1](card_order[1], ops[2](card_order[2],
                if result5 is not None and result5 <= 24:
                    max_value = max(max_value, result5)
            except:
                pass

    return max_value if max_value != float('-inf') else "Impossible"

# Đọc đầu vào
def main():
    n = int(input("Nhập số lượng bộ bài: "))  # Số bộ bài
    results = []

    for _ in range(n):
        print("Nhập 4 giá trị của bộ bài:")
        cards = [int(input()) for _ in range(4)]
        results.append(max_value_under_24(cards))

    print("\nKết quả:")
    for res in results:
        print(res)

if __name__ == "__main__":
    main()
```

Tài liệu