ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

CS112.P11.CTTN

Backtracking, Bruteforce

Nhóm 5 thực hiện

Môn học: CS112.P11.KHTN

Sinh viên thực hiện:

Nguyễn Thiên Bảo - 23520127

Trần Lê Minh Nhật - 2352

Giáo viên hướng dẫn:

ThS. Nguyễn Thanh Sơn

Ngày 7 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?

1.2 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.

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?

1.4 Lời giải Bài tập 1

Câu 1: *Nguyên lý cơ bản của Backtracking:* Thuật toán quay lui (Backtracking) xây dựng lời giải từng bước, nếu tại một bước nào đó không dẫn tới kết quả hợp lệ, ta sẽ "quay lui" để thử lựa chọn khác. Cụ thể:

1. Chọn một bước để mở rộng lời giải.
2. Kiểm tra tính hợp lệ (nếu vi phạm ràng buộc, loại bỏ và quay lui).
3. Nếu hợp lệ, tiếp tục sang bước tiếp theo.
4. Dừng khi tìm được lời giải hoàn chỉnh hoặc hết lựa chọn.

Tại sao Backtracking thường được dùng trong bài toán tổ hợp: Vì các bài toán tổ hợp thường có không gian nghiệm rất lớn. Backtracking giúp ta duyệt qua không gian này một cách có trật tự và

cho phép cắt bỏ sớm những nhánh không khả thi, từ đó tiết kiệm thời gian so với brute force đơn thuần.

Câu 2: *So sánh Backtracking và Branch and Bound:*

- **Mục tiêu:** Backtracking thường dùng để tìm lời giải hợp lệ, không nhất thiết tối ưu. Branch and Bound hướng đến tìm lời giải tối ưu (cực tiểu/cực đại).
- **Cắt tỉa:** Backtracking cắt tỉa dựa trên ràng buộc tính hợp lệ; Branch and Bound cắt tỉa dựa trên cận trên/dưới của hàm mục tiêu.
- **Hướng tìm kiếm:** Backtracking thử nghiệm các hướng đi có/không hợp lệ, còn Branch and Bound ưu tiên nhánh hứa hẹn hơn (dựa trên đánh giá cận).

Câu 3: *Ưu và nhược điểm của Brute Force:*

- **Ưu điểm:** Dễ hiểu, dễ cài đặt, đảm bảo không bỏ sót lời giải.
- **Nhược điểm:** Tốn rất nhiều thời gian và tài nguyên, đặc biệt khi không gian nghiệm lớn.

Tại sao kém hiệu quả cho bài toán lớn: Số lượng trường hợp cần xét thường tăng theo hàm mũ hoặc giai thừa, khiến thời gian chạy trở nên không khả thi.

2 Bài tập 2

Đề bài: Trò chơi 24 sử dụng bốn thẻ bài. Mỗi lượt, mỗi người chơi lật 1 thẻ. Mục tiêu là tìm biểu thức số học sử dụng giá trị 4 thẻ (A=1, J=11, Q=12, K=13) bằng các phép cộng, trừ, nhân, chia (chia phải ra số nguyên), sắp xếp và đặt ngoặc sao cho kết quả biểu thức bằng 24. Ví dụ:

$$((A \times K) - J) \times Q = ((1 \times 13) - 11) \times 12 = 24.$$

Nếu không tìm được biểu thức bằng 24, ta cần tìm biểu thức có giá trị lớn nhất không vượt quá 24. Nếu không có biểu thức nào ≤ 24, in ra giá trị lớn nhất có thể tạo được.

Định dạng đầu vào: Dòng đầu: số nguyên $N \leq 5$ là số lượng bộ bài. Mỗi bộ bài gồm 4 dòng, mỗi dòng chứa số nguyên $C \in [1, 13]$.

Định dạng đầu ra: Với mỗi bộ bài, in ra một dòng chứa số nguyên là giá trị tốt nhất ≤ 24 đạt được. Nếu không tồn tại giá trị ≤ 24, in ra giá trị lớn nhất có thể.

2.1 Lời giải Bài tập 2

Ý tưởng:

1. Hoán vị 4 thẻ để xét tất cả thứ tự.
2. Thử tất cả cách chèn 3 phép toán (+, -, $\times$, $\div$) giữa 4 số.
3. Xét tất cả cách đặt ngoặc (5 dạng đặt ngoặc khác nhau).
4. Kiểm tra tính hợp lệ của phép chia (phải là chia hết).
5. Ghi lại kết quả cuối cùng, tìm giá trị tốt nhất ≤ 24. Nếu không có, tìm giá trị lớn nhất.

Cách thực hiện: Dùng phương pháp đệ quy hoặc duyệt toàn bộ. Đối với mỗi cặp số, áp dụng một phép toán, thay hai số bằng kết quả, tiếp tục cho đến khi còn một số. Lặp lại cho mọi hoán vị và mọi phép toán.

Kết luận: Phương pháp này tương tự brute force, nhưng vẫn trong phạm vi tính toán được. Nó đảm bảo tìm ra giá trị tối ưu theo yêu cầu.

2.2 Mã giả

Hàm chính

```
    Đọc N
FOR i FROM 1 TO N:
    Đọc 4 giá trị thẻ: cards = [C1, C2, C3, C4]
    best_leq_24 = None
    best_any = None

    Tạo tất cả hoán vị của cards: perms = permutations(cards)

    FOR mỗi perm trong perms:
        # perm là một sắp xếp gồm 4 số, ví dụ [a, b, c, d]
        possible_results = TínhTấtCảKếtQuả(perm)

        FOR mỗi val trong possible_results:
            NẾU best_any = None HOẶC val > best_any:
                best_any = val
            NẾU val  24:
                NẾU best_leq_24 = None HOẶC val > best_leq_24:
                    best_leq_24 = val

    NẾU best_leq_24  None:
        In best_leq_24
    NGƯỢC LẠI:
        NẾU best_any = None:
            In 0  # Trường hợp không tạo ra được kết quả nào hợp lệ
        NGƯỢC LẠI:
            In best_any
```

Hàm TínhTấtCảKếtQuả(numbers):

```
Hàm TínhTấtCảKếtQuả(numbers):
NẾU độ dài(numbers) = 1:
    TRẢ VỀ tập kết quả {numbers[0]}

Tạo tập results = {}
FOR i FROM 0 TO length(numbers)-1:
    FOR j FROM i+1 TO length(numbers)-1:
        a = numbers[i]
        b = numbers[j]

        remain = tất cả phần tử trong numbers trừ a và b

        # Thử các phép toán cộng, trừ, nhân:
        results = results  TínhTấtCảKếtQuả(remain  {a+b})
        results = results  TínhTấtCảKếtQuả(remain  {a-b})
        results = results  TínhTấtCảKếtQuả(remain  {b-a})
        results = results  TínhTấtCảKếtQuả(remain  {a*b})

        # Thử phép chia nếu chia hết:
        NẾU b  0 và a % b = 0:
            results = results  TínhTấtCảKếtQuả(remain  {a/b})
        NẾU a  0 và b % a = 0:
            results = results  TínhTấtCảKếtQuả(remain  {b/a})

TRẢ VỀ results
```

2.3 code

```
import sys
input_data = sys.stdin.read().strip().split()
N = int(input_data[0])

def generate_results(numbers):
    # Khi chỉ còn một số, trả về tập kết quả gồm số đó
    if len(numbers) == 1:
        return {numbers[0]}

    results = set()
    # Thử chọn 2 số bất kỳ
    for i in range(len(numbers)):
        for j in range(i+1, len(numbers)):
            a = numbers[i]
            b = numbers[j]

            # Tập các số còn lại sau khi bỏ a, b
            remain = [numbers[k] for k in range(len(numbers)) if k != i and k != j]

            # Thử các phép toán:
            # a + b
            results.update(generate_results(remain + [a + b]))
            # a - b
            results.update(generate_results(remain + [a - b]))
            # b - a
            results.update(generate_results(remain + [b - a]))
            # a * b
```

```
            results.update(generate_results(remain + [a * b]))

            # a / b (nếu b != 0 và chia hết)
            if b != 0 and a % b == 0:
                results.update(generate_results(remain + [a // b]))
            # b / a
            if a != 0 and b % a == 0:
                results.update(generate_results(remain + [b // a]))

    return results

pos = 1
for _ in range(N):
    cards = [int(input_data[pos+i]) for i in range(4)]
    pos += 4

    from itertools import permutations
    best_leq_24 = None
    best_any = None

    for perm in permutations(cards):
        possible_results = generate_results(list(perm))
        for val in possible_results:
            # Cập nhật best_any
            if best_any is None or val > best_any:
                best_any = val
            # Cập nhật best_leq_24
            if val <= 24:
```

```
            if best_leq_24 is None or val > best_leq_24:
                best_leq_24 = val


if best_leq_24 is not None:
    print(best_leq_24)
else:
    # Nếu không có kết quả  24 thì in ra best_any
    if best_any is None:
        # Không tìm được kết quả nào
        print(0)
    else:
        print(best_any)
```