TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BTVN NHÓM 12:
Completed search - Brute force, Backtracking, Branch and Bound

PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

Nhóm 9

Sinh viên thực hiện:

Họ và tên	MSSV
Bùi Ngọc Thiên Thanh	23521436
Nguyễn Thái Sơn	23521356

Thành phố Hồ Chí Minh, 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?
Nguyên lý cơ bản của thuật toán quay lui (Backtracking):

- Thuật toán quay lui là một phương pháp giải bài toán bằng cách thử các lựa chọn từng bước một, quay lại khi phát hiện rằng lựa chọn đó không đúng hoặc không đưa đến kết quả tối ưu. Quá trình này tiếp tục cho đến khi tìm được nghiệm đúng hoặc hoàn tất việc duyệt qua tất cả các lựa chọn có thể.

- Quay lui thường được áp dụng trong các bài toán tìm kiếm lời giải từ không gian giải pháp rất lớn và phức tạp, với việc sử dụng đệ quy và lược đồ loại trừ.

Ứng dụng trong bài toán tổ hợp:

- Thuật toán quay lui rất thích hợp để giải các bài toán tổ hợp vì các bài toán này thường yêu cầu duyệt qua tất cả các khả năng của một tập hợp, và quay lui giúp giảm bớt việc duyệt những khả năng không khả thi (ví dụ: khi một tập hợp con không thể trở thành một nghiệm hợp lệ).

- Quay lui giúp xây dựng các giải pháp từng phần và thử nghiệm chúng, loại bỏ các lựa chọn không hợp lý ngay từ đầu mà không cần phải thử tất cả các lựa chọn.

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 thuật toán nhánh cận (Branch and Bound) và quay lui (Backtracking):

1. Mục tiêu:

   - **Nhánh cận (Branch and Bound):** Tìm kiếm lời giải tối ưu cho bài toán tối ưu hóa. Thuật toán này sử dụng một phương pháp để "cắt tỉa" không gian tìm kiếm dựa trên các giới hạn (bounds) để loại bỏ các phần không thể có lời giải tối ưu.
   - **Quay lui (Backtracking):** Chủ yếu tìm kiếm nghiệm của bài toán và thường không đảm bảo tìm được nghiệm tối ưu mà chỉ tìm kiếm một nghiệm khả thi.

2. Cách tiếp cận:

   - **Nhánh cận (Branch and Bound):** Xây dựng cây tìm kiếm và mỗi nút trong cây được đánh giá bằng một giới hạn, giúp loại trừ những nhánh không thể dẫn đến nghiệm tối ưu.
   - **Quay lui (Backtracking):** Tiến hành thử từng khả năng một (thường là đệ quy), quay lại khi phát hiện nhánh không hợp lệ hoặc không thể tiến tới lời giải hợp lệ.

3. Tối ưu hóa:

   - **Nhánh cận (Branch and Bound):** Luôn cố gắng tối ưu hóa quá trình tìm kiếm, giảm thiểu số lượng các

phép thử cần thực hiện.

- **Quay lui (Backtracking):** Không hướng đến tối ưu hóa về mặt thời gian hay hiệu quả, mà chủ yếu là tìm nghiệm hợp lệ.

4. Hiệu quả:

- **Nhánh cận (Branch and Bound):** Thường hiệu quả hơn vì có khả năng loại trừ các nhánh không hợp lý dựa trên thông tin về giới hạn trên hoặc giới hạn dưới của nghiệm.
- **Quay lui (Backtracking):** Thường không hiệu quả trong việc tối ưu hóa, và cần phải thử qua nhiều lựa chọn có thể dẫn đến các nghiệm không hợp lệ.

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 của phương pháp Brute Force:

- **Đơn giản:** Phương pháp brute force dễ hiểu và dễ thực hiện, không yêu cầu các kiến thức phức tạp về thuật toán.
- **Tính chính xác:** Phương pháp này luôn tìm ra được nghiệm đúng vì nó kiểm tra tất cả các khả năng có thể, đảm bảo rằng không bỏ sót nghiệm nào.

Nhược điểm của phương pháp Brute Force:

- **Hiệu suất thấp:** Brute force yêu cầu duyệt qua toàn bộ không gian tìm kiếm, điều này làm cho thời gian thực hiện có thể rất lâu khi kích thước của không gian tìm kiếm tăng lên.

- **Kém hiệu quả trong các bài toán lớn:** Với các bài toán có không gian giải pháp rất lớn (như các bài toán tổ hợp hoặc tối ưu hóa), brute force trở nên kém hiệu quả vì thời gian xử lý tăng theo cấp số mũ. Ví dụ, trong bài toán tìm kiếm tất cả các hoán vị của một tập hợp n phần tử, số lượng hoán vị là n!, điều này dẫn đến thời gian tính toán rất lớn khi n tăng.

2 Bài tập 2:

Ý tưởng chính và các bước giải tóm gọn:

2.1 Ý tưởng chính:

- **Mục tiêu:** Tìm ra một biểu thức toán học hợp lệ sử dụng bốn thẻ số sao cho giá trị của biểu thức không vượt quá 24, và nếu có nhiều kết quả, chọn kết quả lớn nhất.
- **Biểu thức hợp lệ:** Dùng tất cả bốn số, kết hợp với các phép toán (+, -, *, /) và dấu ngoặc để tạo thành biểu thức hợp lệ.

2.2 Các bước giải:

1. **Hoán vị các số:** Sắp xếp lại tất cả các hoán vị của bốn số từ bộ bài (do phép toán không thay đổi thứ tự các số).
2. **Hoán vị các phép toán:** Dùng tất cả các hoán vị của ba phép toán (+, -, *, /) để thử mọi cách kết hợp các phép toán với các số.
3. **Kiểm tra tính hợp lệ:**

- Với mỗi cách kết hợp của các số và phép toán, tính giá trị của biểu thức.
- Nếu giá trị không phải là NaN (vì phép chia cho 0) và không vượt quá 24, kiểm tra và lưu lại kết quả lớn nhất.

4. **Tính toán và trả kết quả:** Sau khi thử tất cả các kết hợp, trả về giá trị lớn nhất hợp lệ mà không vượt quá 24. Nếu không có kết quả hợp lệ, trả về 0.

2.3 Mã giả:

```
// Ham tinh toan phep toan giua hai so
function calc(a, b, op):
if op == '+':
    return a + b
else if op == '-':
    return a - b
else if op == '*':
    return a * b
else if op == '/':
    if b == 0:
        return NAN // Tra tranh chia cho 0
    return a / b

// Ham kiem tra tat ca cac hoan vi cua so va phep toan de
   tim ra gia tri lon nhat <= 24
function max_value(cards):
max_val = -INF // Khoi tao gia tri lon nhat la -INF

// Cac phep toan co the su dung
operations = ['+', '-', '*', '/']

// Tao tat ca cac hoan vi cua cac the so
for each permutation of cards:
    // Tao cac hoan vi cua phep toan
    for each permutation of operations:
```

```
25          // Kiem tra cac cach sap xep so va phep toan
26          result1 = calc(calc(cards[0], cards[1],
               operations[0]), cards[2], operations[1])
27          if result1 is valid:
28              final_result = calc(result1, cards[3],
                   operations[2])
29              if final_result <= 24:
30                  max_val = max(max_val, final_result)
31
32          // Kiem tra cac nhom phep toan khac (vi du: (a * (b +
               c)))
33          result2 = calc(cards[0], calc(cards[1], cards[2],
               operations[0]), operations[1])
34          if result2 is valid:
35              final_result = calc(result2, cards[3],
                   operations[2])
36              if final_result <= 24:
37                  max_val = max(max_val, final_result)
38
39  // Neu khong tim duoc gia tri hop le, tra ve 0
40  if max_val == -INF:
41      return 0
42  return max_val
43
44  // Ham chinh de xu ly nhieu bo bai
45  function main():
46  N = input() // So luong bo bai
47  for i = 1 to N:
48      cards = input() // Doc 4 so tu bo bai
49      print(max_value(cards)) // In ra gia tri lon nhat <= 24
```