ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
Ngày 27, Tháng 11, Năm 2024

Bài tập về nhà nhóm 2:

Giáo viên hướng dẫn: ***Nguyễn Thanh Sơn***

Trần Vinh Khánh, Nguyễn Thiện Nhân

Mục lục

1 Bài 1: **2**
1.1 Câu hỏi 1: 2
1.2 Câu hỏi 2: 3
1.3 Câu hỏi 3: 4

2 Bài toán 24 - Trò chơi thẻ **5**

1. Bài 1:

1.1. Câu hỏi 1:

Trình bày nguyên lý cơ bản của thuật toán quay lui (Backtracking). Ti sao thut toán này thng c s dng gii các bài toán t hp?

Thuật toán **quay lui (Backtracking)** là một kỹ thuật tìm kiếm thử và sai để giải quyết các bài toán yêu cầu tìm kiếm trong không gian giải pháp. Nguyên lý cơ bản gồm các bước sau:

1. **Xây dựng không gian tìm kiếm:**
   - Xác định các bước hoặc trạng thái có thể dẫn đến một giải pháp.
   - Tạo ra cây trạng thái, trong đó mỗi nhánh thể hiện một lựa chọn tại một bước cụ thể.
2. **Thử lựa chọn tại một bước:**
   - Tại mỗi bước, thử một giá trị hoặc lựa chọn hợp lệ.
3. **Kiểm tra tính khả thi:**
   - Nếu lựa chọn đó không hợp lệ hoặc không thể dẫn đến một giải pháp, quay lui về bước trước và thử lựa chọn khác.
4. **Đi sâu hơn (đệ quy):**
   - Nếu lựa chọn khả thi, tiếp tục đi sâu vào bước tiếp theo với lựa chọn đó.
5. **Dừng khi đạt giải pháp hoặc hết lựa chọn.**

Thuật toán **quay lui (Backtracking)** thường được sử dụng để giải các bài toán tổ hợp vì những lý do sau:

1. **Tính chất thử và sai (Trial and Error):**
   - Các bài toán tổ hợp thường yêu cầu tìm tất cả các tổ hợp hoặc hoán vị thỏa mãn một điều kiện nhất định.
   - Quay lui cung cấp một cách tiếp cận có hệ thống để thử từng khả năng, đảm bảo không bỏ sót giải pháp nào.
2. **Cấu trúc cây tìm kiếm:**
   - Không gian tìm kiếm của các bài toán tổ hợp thường được biểu diễn dưới dạng một *cây trạng thái*.
   - Mỗi nhánh của cây thể hiện một lựa chọn (ví dụ: thêm phần tử vào tổ hợp, hoán vị, hoặc dãy).
   - Thuật toán quay lui duyệt qua từng nhánh của cây một cách hiệu quả và có kiểm soát.
3. **Cắt tỉa không gian tìm kiếm:**

- Quay lui giúp loại bỏ sớm các trường hợp không thể dẫn đến giải pháp hợp lệ (còn gọi là *cắt tỉa cây tìm kiếm*).
- Điều này làm giảm số lần tính toán không cần thiết, giúp thuật toán hoạt động hiệu quả hơn.

4. **Khả năng áp dụng rộng rãi:**

- Thuật toán quay lui có thể giải quyết nhiều loại bài toán tổ hợp khác nhau như:
  - Tìm tất cả các tổ hợp hoặc hoán vị.
  - Bài toán n-queens.
  - Tô màu đồ thị.
  - Bài toán phân hoạch số.

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.

Dưới đây là so sánh điểm khác biệt chính giữa hai thuật toán khi tìm kiếm lời giải tối ưu:

1. **Mục tiêu tìm kiếm:**

- **Quay lui (Backtracking):** Tập trung tìm tất cả các lời giải hợp lệ và thường sử dụng cho các bài toán tổ hợp hoặc bài toán thỏa mãn ràng buộc.
- **Nhánh cận (Branch and Bound):** Tập trung tìm kiếm lời giải **tối ưu nhất** bằng cách sử dụng các ràng buộc để loại bỏ các nhánh không cần thiết.

2. **Kỹ thuật cắt tỉa:**

- **Quay lui:** Cắt tỉa dựa trên việc loại bỏ các trạng thái không hợp lệ (các ràng buộc không được thỏa mãn).
- **Nhánh cận:** Cắt tỉa dựa trên **cận trên** và **cận dưới** của bài toán, loại bỏ các nhánh không thể dẫn đến lời giải tối ưu.

3. **Thứ tự duyệt:**

- **Quay lui:** Thường duyệt cây trạng thái theo thứ tự *Depth-First Search (DFS)*.
- **Nhánh cận:** Có thể duyệt theo *Breadth-First Search (BFS)*, *Best-First Search*, hoặc chiến lược khác dựa trên cận.

4. **Tính ứng dụng:**

- **Quay lui:** Hiệu quả cho các bài toán tổ hợp như bài toán n-queens, tô màu đồ thị, hoặc tìm tất cả các hoán vị.
- **Nhánh cận:** Hiệu quả hơn cho các bài toán tối ưu, ví dụ: bài toán balô (Knapsack Problem), bài toán người du lịch (TSP).

5. **Đánh giá và loại bỏ nhánh:**

- **Quay lui:** Không sử dụng các phương pháp đánh giá giá trị tối ưu tại mỗi nhánh.
- **Nhánh cận:** Sử dụng **hàm đánh giá** để ước lượng và loại bỏ nhánh dựa trên tiềm năng 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. Ưu điểm

- **Đơn giản và dễ hiểu:**
  - Phương pháp Brute Force không yêu cầu các kỹ thuật phức tạp hoặc kiến thức chuyên sâu.
  - Có thể áp dụng cho hầu hết mọi bài toán bằng cách thử tất cả các khả năng.
- **Tính toàn diện:**
  - Đảm bảo tìm được lời giải chính xác nếu có, vì mọi khả năng đều được kiểm tra.
- **Tổng quát:**
  - Có thể áp dụng cho nhiều loại bài toán khác nhau mà không cần thay đổi cấu trúc cơ bản.

2. Nhược điểm

- **Kém hiệu quả:**
  - Thời gian chạy tăng nhanh theo kích thước đầu vào, thường có độ phức tạp *exponential* hoặc *factorial*.
  - Không phù hợp với các bài toán lớn do số lượng trường hợp cần thử quá nhiều.
- **Tốn tài nguyên:**
  - Yêu cầu bộ nhớ và thời gian tính toán lớn, dẫn đến hiệu suất thấp.
- **Không tối ưu:**
  - Không tận dụng được các tính chất đặc biệt hoặc ràng buộc của bài toán để giảm không gian tìm kiếm.

3. Tại sao phương pháp Brute Force kém hiệu quả trong các bài toán lớn?

- **Tăng trưởng theo cấp số nhân:**
  - Với các bài toán tổ hợp hoặc tối ưu, số lượng trường hợp cần kiểm tra tăng theo cấp số nhân hoặc giai thừa khi kích thước đầu vào tăng.

- Ví dụ: Đối với bài toán người du lịch (*Traveling Salesman Problem*), số lộ trình cần kiểm tra là $n!$, với n là số thành phố.

- **Không tận dụng tính chất bài toán:**
  - Phương pháp Brute Force không sử dụng các chiến lược cắt tỉa hoặc tối ưu hóa để giảm số lượng phép thử.

- **Thời gian tính toán lớn:**
  - Thời gian cần thiết để thử mọi khả năng có thể trở nên không khả thi đối với các bài toán lớn, khiến phương pháp này không thực tế.

2. Bài toán 24 - Trò chơi thẻ

Mã nguồn C++ và giải thích

Mã nguồn

```cpp
// Th   vin   cn   thit
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;

const double EPS = 1e-6;
int bestValue = 0;

// Hm   kim   tra hai  s   c   gn   bng  nhau
bool isEqual(double a, double b) {
    return fabs(a - b) < EPS;
}

// Hm  đ   quy đ    th   mi   php   ton   gia   cc   s
void solve(vector<double> nums) {
    if (nums.size() == 1) { // Nu  ch   cn   mt   s
        if (nums[0] <= 24 && nums[0] > bestValue) { // Gi tr khng   vt   qu  24
            bestValue = static_cast<int>(nums[0]);
        }
        return;
    }
    //  Th   mi   cp   s   v   mi   php   ton
    for (int i = 0; i < nums.size(); ++i) {
        for (int j = 0; j < nums.size(); ++j) {
            if (i == j) continue; // Khng  s   dng   li   s   đ   chn
            vector<double> nextNums;
            for (int k = 0; k < nums.size(); ++k) {
                if (k != i && k != j) nextNums.push_back(nums[k]);
            }
            //  Thc   hin   cc  php   ton   c   bn
            nextNums.push_back(nums[i] + nums[j]);
            solve(nextNums);
            nextNums.pop_back();

```

```
            nextNums.push_back(nums[i] - nums[j]);
            solve(nextNums);
            nextNums.pop_back();

            nextNums.push_back(nums[i] * nums[j]);
            solve(nextNums);
            nextNums.pop_back();

            if (nums[j] != 0 && isEqual(nums[i] / nums[j], floor(nums[i] / nums[j]))) {
                nextNums.push_back(nums[i] / nums[j]);
                solve(nextNums);
                nextNums.pop_back();
            }
        }
    }
}

int main() {
    int N;
    cin >> N;
    while (N--) {
        vector<int> cards(4);
        for (int i = 0; i < 4; ++i) {
            cin >> cards[i];
        }
        bestValue = 0;
        vector<double> nums(cards.begin(), cards.end());
        solve(nums);
        cout << bestValue << endl;
    }
    return 0;
}
```

Giải thích cách làm

- **Đọc đầu vào:** Chương trình đọc số lượng bộ bài N và giá trị của 4 thẻ trong mỗi bộ bài.
- **Thử tất cả các phép toán:**
  - Sử dụng một hàm đệ quy `solve` để thử mọi cặp số trong mảng.
  - Thực hiện các phép toán cộng, trừ, nhân, và chia (nếu phép chia không tạo ra số thập phân).
- **Điều kiện dừng:** Khi mảng chỉ còn một số, kiểm tra xem giá trị của số này có gần bằng hoặc nhỏ hơn 24 và cập nhật kết quả tốt nhất.
- **Kiểm tra phép chia:** Phép chia chỉ được thực hiện nếu kết quả là số nguyên (được kiểm tra bằng hàm `isEqual`).
- **Kết quả:** In ra giá trị lớn nhất không vượt quá 24.

Ví dụ hoạt động

Đầu vào:

```
1
1 13 11 12
```

Đầu ra:

```
24
```

Giải thích: Biểu thức $((1 \times 13) - 11) \times 12 = 24$ được tìm thấy trong quá trình thử tất cả các tổ hợp và phép toán.