**![A white rectangular frame with black border

AI-generated content may be incorrect.]()BỘ GIÁO DỤC & ĐÀO TẠO**

TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ

THÀNH PHỐ HỒ CHÍ MINH

![A blue letter on a black background

AI-generated content may be incorrect.]()

**BÁO CÁO ĐỒ ÁN**

ĐỀ TÀI 5:

**Giải bài toán sudoku sử dụng heuristic**

Giảng viên hướng dẫn**: TS.Nguyễn Quan Bá Hồng**

Sinh viên thực hiện: Đặng Nguyễn Duy Trường -2302700093

TP.Hồ Chí Minh 7-2025

**1. Giới thiệu**

Sudoku là một trò chơi xếp số dựa trên logic phổ biến, với mục tiêu điền các chữ số từ 1 đến 9 vào một lưới 9x9 sao cho mỗi hàng, mỗi cột và mỗi vùng 3x3 chứa tất cả các chữ số đó mà không lặp lại. Bài toán Sudoku là một ví dụ điển hình cho các bài toán ràng buộc (Constraint Satisfaction Problems - CSP) và có thể được giải quyết hiệu quả bằng các thuật toán tìm kiếm như backtracking. Để tối ưu hóa tốc độ giải quyết, đồ án này tập trung vào việc triển khai thuật toán backtracking kết hợp với một heuristic mạnh mẽ: Minimum Remaining Values (MRV - Giá trị còn lại tối thiểu).

**2. Mô tả bài toán Sudoku**

Bài toán Sudoku yêu cầu điền các số vào một bảng 9x9 được chia thành chín vùng 3x3 nhỏ hơn. Một số ô đã được điền sẵn ban đầu. Mục tiêu là điền các chữ số từ 1 đến 9 vào các ô trống sao cho ba quy tắc sau được thỏa mãn:

* Mỗi hàng phải chứa tất cả các chữ số từ 1 đến 9 đúng một lần.
* Mỗi cột phải chứa tất cả các chữ số từ 1 đến 9 đúng một lần.
* Mỗi trong chín vùng con 3x3 phải chứa tất cả các chữ số từ 1 đến 9 đúng một lần.

**3. Các yêu cầu của đồ án**

Dựa trên mô tả ban đầu, các yêu cầu của đồ án được phân loại thành yêu cầu tối thiểu và yêu cầu nâng cao.

**3.1. Yêu cầu tối thiểu**

* **Nhập bảng Sudoku 9x9 chưa hoàn chỉnh:** Chương trình phải có khả năng đọc và xử lý một bảng Sudoku chưa hoàn chỉnh do người dùng cung cấp.
* **Giải Sudoku bằng thuật toán backtracking kết hợp heuristic:** Sử dụng thuật toán backtracking cơ bản và thêm heuristic chọn ô có ít số khả thi nhất để điền tiếp (Minimum Remaining Values - MRV).
* **Kiểm tra ràng buộc trong Sudoku:** Đảm bảo tính hợp lệ của việc điền số sau mỗi bước (kiểm tra hàng, cột và vùng 3x3).
* **Xuất lời giải hoặc báo không có lời giải:** Hiển thị bảng Sudoku đã giải hoặc thông báo nếu không tìm thấy lời giải cho bảng đã nhập.

**3.2. Yêu cầu nâng cao (tùy chọn)**

* **Hỗ trợ giải biến thể Sudoku:** Mở rộng để giải các biến thể như Sudoku 16x16 hoặc Sudoku X (có thêm ràng buộc đường chéo).
* **Xây dựng giao diện nhập liệu và hiển thị lời giải:** Phát triển giao diện người dùng (GUI) để nhập bảng và hiển thị kết quả một cách trực quan.
* **Hiển thị số lần thử, thời gian giải:** Cung cấp các chỉ số về hiệu suất của thuật toán.
* **Tự động phát hiện Sudoku không có lời giải:** Nhanh chóng xác định các bảng Sudoku không thể giải được.
* **Tạo đề tự động và đảm bảo có lời giải duy nhất:** Khả năng tạo ra các câu đố Sudoku mới, đảm bảo chúng có một lời giải duy nhất.

**4. Phân tích thiết kế hệ thống**

Hệ thống được thiết kế dựa trên thuật toán backtracking làm nền tảng, kết hợp với heuristic MRV để tối ưu hóa.

**4.1. Cấu trúc chương trình**

Chương trình được chia thành hai phần chính:

* **sudokutools.py:** Chứa các hàm cốt lõi cho việc giải Sudoku, bao gồm thuật toán backtracking, các hàm kiểm tra tính hợp lệ, và các hàm hỗ trợ tìm ô trống cũng như tạo bảng.
* **SudokuGUI.py:** Xây dựng giao diện người dùng đồ họa (GUI) sử dụng thư viện Pygame để tương tác với người dùng, hiển thị bảng, nhận input và trực quan hóa quá trình giải.

**4.2. Thuật toán Backtracking**

Thuật toán backtracking hoạt động theo nguyên tắc thử-và-sai:

1. **Tìm ô trống:** Tìm một ô trống trên bảng Sudoku.
2. **Thử giá trị:** Nếu tìm thấy ô trống, thử điền các số từ 1 đến 9 vào ô đó.
3. **Kiểm tra hợp lệ:** Đối với mỗi số được thử, kiểm tra xem việc điền số đó có hợp lệ theo các quy tắc Sudoku hay không.
4. **Đệ quy:** Nếu số hợp lệ, điền số vào ô và gọi đệ quy hàm giải cho bảng đã cập nhật.
5. **Quay lui (Backtrack):** Nếu lời gọi đệ quy không tìm được lời giải (trả về False), tức là số đã chọn không dẫn đến giải pháp, thì hủy bỏ việc điền số đó (đặt lại ô về 0) và thử số tiếp theo.
6. **Điều kiện dừng:** Nếu không còn ô trống nào, có nghĩa là bảng đã được giải thành công và hàm trả về True. Nếu đã thử tất cả các số cho một ô trống mà không có số nào hợp lệ, hàm trả về False.

**4.3. Heuristic Minimum Remaining Values (MRV)**

Heuristic MRV là một chiến lược lựa chọn ô trống để tối ưu hóa backtracking. Thay vì chọn ô trống bất kỳ, MRV ưu tiên chọn ô có số lượng giá trị hợp lệ khả thi ít nhất.

* **Ưu điểm:** Bằng cách tập trung vào các ô bị ràng buộc chặt chẽ nhất, thuật toán có thể phát hiện các mâu thuẫn sớm hơn trong quá trình tìm kiếm, giảm số lượng bước thử và quay lui, từ đó tăng tốc độ giải.
* **Cách triển khai:** Hàm find\_empty\_mrv sẽ duyệt qua tất cả các ô trống, tính toán số lượng giá trị hợp lệ cho mỗi ô, và trả về ô có số lượng giá trị hợp lệ nhỏ nhất. Nếu một ô có 0 giá trị hợp lệ, nó sẽ được trả về ngay lập tức để chỉ ra một bế tắc.

**4.4. Kiểm tra ràng buộc Sudoku**

Hàm valid(board, pos, num) được sử dụng để kiểm tra xem một số num có thể đặt vào vị trí pos trên board hay không. Hàm này thực hiện ba kiểm tra chính:

* **Kiểm tra hàng:** Đảm bảo num không xuất hiện trong cùng hàng với pos.
* **Kiểm tra cột:** Đảm bảo num không xuất hiện trong cùng cột với pos.
* **Kiểm tra vùng 3x3:** Đảm bảo num không xuất hiện trong vùng 3x3 chứa pos.

**4.5. Cơ chế nhập và xuất**

* **Nhập:** Chương trình được thiết kế để nhận bảng Sudoku dưới dạng mảng 2D (list of lists). Đối với giao diện GUI, bảng ban đầu được tạo ngẫu nhiên.
* **Xuất:** Lời giải được hiển thị trên console thông qua hàm print\_board hoặc trên giao diện đồ họa bởi Pygame.

**5. Phân tích chi tiết mã nguồn**

**5.1. Tệp sudokutools.py**

* **print\_board(board):** Hàm này chịu trách nhiệm in bảng Sudoku ra console một cách dễ đọc, có thêm các đường phân cách cho các khối 3x3.
* **find\_empty(board):** Hàm này tìm ô trống đầu tiên (giá trị 0) trong bảng theo thứ tự từ trái sang phải, từ trên xuống dưới.
* **find\_empty\_mrv(board):** Đây là hàm triển khai heuristic MRV. Nó duyệt qua tất cả các ô trống, tính số lượng giá trị có thể đặt vào mỗi ô (bằng cách gọi valid), và trả về ô trống có số lượng giá trị khả thi ít nhất. Nếu một ô có 0 giá trị khả thi, nó sẽ trả về ngay lập tức.
* **valid(board, pos, num):** Kiểm tra xem việc đặt num vào pos có hợp lệ theo các quy tắc hàng, cột và khối 3x3 hay không.
* **solve(board):** Hàm này triển khai thuật toán backtracking. Nó tìm một ô trống (hiện đang sử dụng find\_empty, **cần được thay đổi thành find\_empty\_mrv để tích hợp heuristic**), thử các số từ 1 đến 9. Nếu số hợp lệ, nó đặt số đó và gọi đệ quy. Nếu cuộc gọi đệ quy thành công, hàm trả về True. Ngược lại, nó quay lui bằng cách đặt ô về 0 và thử số tiếp theo.
* **generate\_board():** Tạo một bảng Sudoku ngẫu nhiên. Nó bắt đầu bằng cách điền ngẫu nhiên các khối 3x3 trên đường chéo, sau đó sử dụng backtracking để điền phần còn lại của bảng, và cuối cùng loại bỏ một số lượng ngẫu nhiên các ô để tạo thành một câu đố.

**5.2. Tệp SudokuGUI.py**

* **Board class:** Lớp này quản lý trạng thái của bảng Sudoku trong GUI. Nó khởi tạo một bảng mới bằng cách gọi generate\_board() và tạo một phiên bản đã giải của bảng đó (solvedBoard) để sử dụng cho gợi ý. Nó chứa một ma trận các đối tượng Tile.
  + **draw\_board():** Vẽ lưới Sudoku và các đường phân cách đậm cho các khối 3x3.
  + **deselect(tile):** Bỏ chọn tất cả các ô ngoại trừ ô hiện tại.
  + **redraw(keys, wrong, time):** Cập nhật hiển thị của bảng, tô sáng các ô được chọn, đúng, sai. Nó cũng hiển thị số lần sai và thời gian chơi.
  + **visualSolve(wrong, time):** Hàm này trực quan hóa quá trình giải Sudoku bằng backtracking. Nó hiển thị từng bước điền số, có độ trễ nhỏ để người dùng có thể theo dõi. **Tương tự như solve trong sudokutools.py, hàm này hiện đang sử dụng find\_empty, và cần được cập nhật để sử dụng heuristic MRV.**
  + **hint(keys):** Cung cấp một gợi ý bằng cách điền một ô trống ngẫu nhiên với số chính xác từ solvedBoard.
* **Tile class:** Đại diện cho một ô đơn lẻ trên bảng Sudoku.
  + **\_\_init\_\_:** Khởi tạo giá trị, cửa sổ Pygame, vị trí và trạng thái của ô (được chọn, đúng, sai).
  + **draw(color, thickness):** Vẽ khung cho ô.
  + **display(value, position, color):** Hiển thị giá trị số trong ô.
  + **clicked(mousePos):** Kiểm tra xem ô có được nhấp chuột hay không.
* **main():** Hàm chính của ứng dụng Pygame. Nó thiết lập cửa sổ, xử lý các sự kiện người dùng (nhấp chuột, nhấn phím), cập nhật trạng thái trò chơi và gọi redraw để cập nhật hiển thị. Nó cũng quản lý thời gian chơi và số lần sai.

**6. Đánh giá và hướng phát triển**

**6.1. Các yêu cầu đã đạt được**

* **Backtracking cơ bản:** Thuật toán backtracking đã được cài đặt đúng và hoạt động để giải các bảng Sudoku.
* **Kiểm tra ràng buộc:** Các hàm valid hoạt động chính xác để kiểm tra các ràng buộc hàng, cột và khối 3x3.
* **Hiển thị giải pháp:** Chương trình có thể in giải pháp ra console và hiển thị trực quan trên GUI.
* **Tạo bảng ngẫu nhiên:** Có khả năng tạo ra các bảng Sudoku ngẫu nhiên và đảm bảo chúng có thể giải được.
* **Giao diện người dùng cơ bản:** Đã xây dựng giao diện với Pygame, cho phép người dùng tương tác, điền số, và nhận gợi ý.
* **Trực quan hóa quá trình giải:** Chế độ visualSolve giúp người dùng thấy được cách thuật toán hoạt động.
* **Hiển thị thời gian và số lần sai:** GUI hiển thị thời gian đã trôi qua và số lần người dùng nhập sai.

**6.2. Hướng phát triển trong tương lai**

* **Triển khai các biến thể Sudoku:** Mở rộng hàm valid và logic giải để hỗ trợ Sudoku 16x16 hoặc Sudoku X.
* **Phân tích hiệu suất:** Bổ sung tính năng đếm số lần thử và quay lui để đánh giá hiệu suất của heuristic MRV so với backtracking cơ bản.
* **Tạo đề tự động với lời giải duy nhất:** Đây là một tính năng nâng cao đáng kể, đòi hỏi thuật toán phức tạp hơn để đảm bảo tính duy nhất của lời giải.