ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

Báo cáo bài tập lớn:

TÌM KIẾM

GVHD:	Vương Bá Thịnh	
Lớp:	L02	
Sinh viên thực hiện:	Nguyễn Ngọc Chiến Công	2210408
	Đào Ngọc Minh	2212023
	Nguyễn Việt Hùng	2011315
	Lê Văn Quang Vinh	2213965

TP. HỒ CHÍ MINH, 02/2025

MỤC LỤC

Danh sách thành viên và phân công công việc

Họ và tên	MSSV	Phân công	Mức độ hoàn thành
Nguyễn Ngọc Chiến Công	2210408	Sudoku - Quay video	100%
Đào Ngọc Minh	2212023	Sudoku - Báo cáo	100%
Nguyễn Việt Hùng	2011315	Chess Ranger - Quay video	100%
Lê Văn Quang Vinh	2213965	Chess Ranger - Báo cáo	100%

1 Sudoku

1.1 Giới thiệu đề bài toán

Đề bài đưa ra yêu cầu thiết kế và hiện thực giải thuật tìm kiếm để giải bài toán Sudoku and Chess Ranger, thực hiện bằng 2 giải thuật : Depth-first search và 1 giải thuật heuristic bất kì :

- Tạo ra input phù hợp.
- Hiện thực có tính năng demo step-by-step để có hiển thị trực quan (như game).
- Bắt buộc sử dụng Python 3.
- Viết lại báo cáo nêu rõ quá trình tìm hiểu và hiện thực bài toán này.

1.2 Mô tả bài toán

Sudoku là một trò chơi giải đố logic trên lưới ô vuông 9×9, trong đó người chơi cần điền các số từ 1 đến 9 vào các ô sao cho:

- Mỗi hàng chứa tất cả các số từ 1 đến 9 mà không bị trùng lặp.
- Mỗi cột chứa tất cả các số từ 1 đến 9 mà không bị trùng lặp.
- Mỗi vùng 3×3 (trong tổng số 9 vùng con của lưới 9×9) chứa tất cả các số từ 1 đến 9 mà không bị trùng lặp.

Một bảng Sudoku ban đầu sẽ có một số ô đã được điền sẵn số, và nhiệm vụ của người chơi là điền các số còn thiếu sao cho thỏa mãn ba điều kiện trên.

1.3 Giới thiệu sơ lược các giải thuật

1.3.1 Depth first search (DFS)

DFS là thuật toán tìm kiếm theo chiều sâu, hoạt động bằng cách thử tất cả các khả năng theo một nhánh trước khi quay lui (backtracking). Khi áp dụng vào bài toán Sudoku:

- Duyệt qua từng ô của bảng, tìm ô chưa được điền (giá trị 0).
- Thử tất cả các số từ 1 đến 9:
  - Nếu số đó hợp lệ (tức là không trùng trong hàng, cột, và ô 3x3), thì điền vào.
  - Gọi đệ quy để tiếp tục giải quyết ô tiếp theo.
  - Nếu một lựa chọn không dẫn đến kết quả hợp lệ, quay lui (backtrack) và thử số khác.
- Nếu đã điền xong tất cả các ô, thuật toán trả về True, nghĩa là đã giải thành công.

Những điểm quan trọng trong code:

- `is_valid(board, row, col, num)` kiểm tra xem một số có thể điền vào ô `(row, col)` không.
- Cập nhật GUI (giao diện đồ họa): `entries[row][col].delete(0, tk.END)`, `entries[row][col].insert(0, str(num))` giúp hiển thị tiến trình giải.
- Dùng `root.update_idletasks()` để cập nhật giao diện ngay lập tức.
- Backtracking (quay lui): Nếu thử hết 9 số mà không thành công, hàm trả về `False` và quay lại ô trước đó để thử số khác.

1.3.2 Heuristic search

Heuristic Search - cụ thể trong bài toán này là **Minimum Remaining Values (MRV)**, sử dụng một chiến lược thông minh hơn để chọn ô điền tiếp theo, thay vì duyệt toàn bộ bảng theo thứ tự từ trên xuống dưới. Trong bài toán Sudoku, một heuristic hữu ích là:

- Chọn ô có ít lựa chọn hợp lệ nhất trước, để giảm số lần thử sai.
- Dùng hàng đợi ưu tiên (heap) để sắp xếp các ô theo số lượng khả năng điền ít nhất.

Những điểm quan trọng trong code:

- Dùng hàng đợi ưu tiên `heapq` để chọn ô có ít lựa chọn nhất.
- Sử dụng `find_candidates()` để tìm số hợp lệ trước khi thử điền.
- Tương tự DFS, thuật toán cũng dùng đệ quy và backtracking nhưng thử các số theo thứ tự tối ưu trước, giúp giảm số lần backtrack.

1.4 Tìm hiểu và hiện thực bài toán

- Trạng thái:
  Bản đồ là một ma trận 9×9 đại diện cho bảng Sudoku, ví dụ:

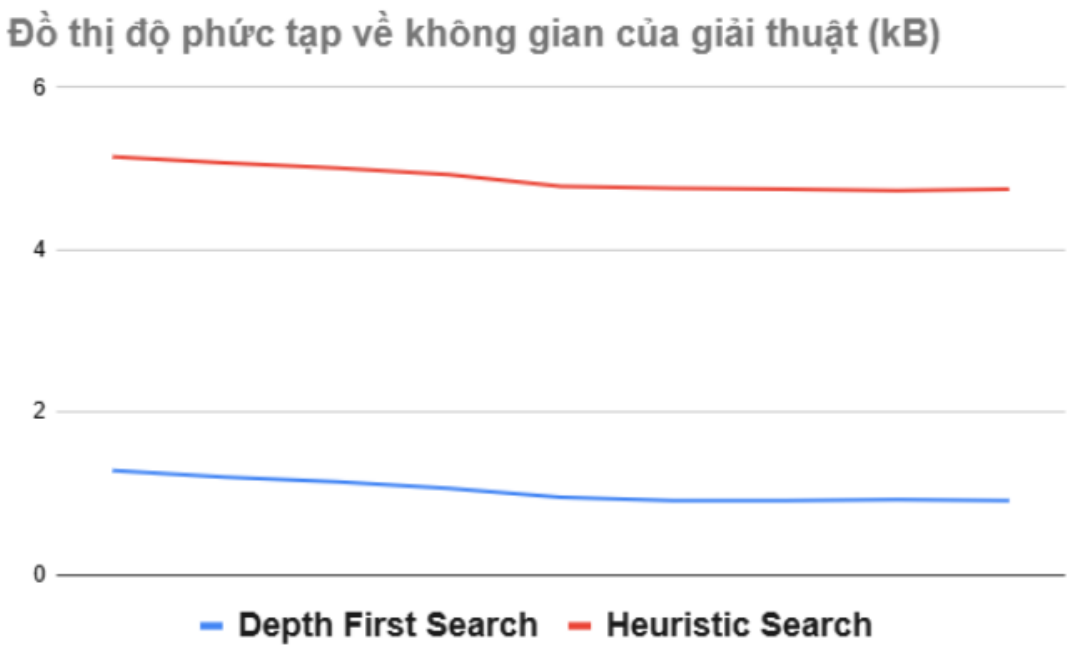
5	3	0	0	7	0	0	0	0
6	0	0	1	9	5	0	0	0
0	9	8	0	0	0	0	6	0
8	0	0	0	6	0	0	0	3
4	0	0	8	0	3	0	0	1
7	0	0	0	2	0	0	0	6
0	6	0	0	0	0	2	8	0
0	0	0	4	1	9	0	0	5
0	0	0	0	8	0	0	7	9

  - Số từ 1 đến 9 đại diện cho các ô đã điền.
  - Số 0 đại diện cho các ô chưa điền.
- Trạng thái khởi đầu:
  - Một bảng Sudoku chưa hoàn chỉnh với một số ô đã có số từ trước.
  - Các ô còn trống có giá trị 0, cần phải điền số phù hợp.
- Trạng thái đích:
  Một bảng Sudoku đã hoàn chỉnh, thỏa mãn điều kiện:
  - Mỗi hàng chứa đủ số từ 1 đến 9, không trùng lặp.
  - Mỗi cột chứa đủ số từ 1 đến 9, không trùng lặp.
  - Mỗi ô 3×3 chứa đủ số từ 1 đến 9, không trùng lặp.
- Bước đi hợp lệ:
  - Chỉ được điền vào ô có giá trị khởi đầu là `0`.
  - Điền các giá trị từ `1` tới `9`.
  - Chưa tồn tại trong hàng, cột và ô 3x3.

1.5 Đánh giá hiệu năng các giải thuật

Testcase	Thời gian (s)		Bộ nhớ (KB)	
	DFS	HS	DFS	HS
1	0.1155	0.2203	6.51	9.70
2	0.0415	0.1119	6.51	13.59
3	4.3347	0.6081	6.51	9.55
4	0.2321	0.7020	6.51	9.68
5	0.3284	0.7648	6.51	9.60
6	2.9666	0.1637	6.51	9.57
7	0.4923	0.4735	6.51	9.66
8	2.8742	0.1571	6.51	9.60
9	0.0920	0.0790	6.51	9.73
10	0.0671	0.3021	6.51	9.66

Bảng 1: Bảng so sánh thời gian và bộ nhớ giữa DFS và HS



1.6 Kết luận và nhận xét

- Giải thuật DFS khi hiện thực, có lưu lại những node đã đi qua, nên giải thuật DFS sẽ luôn ra kết quả (nếu tồn tại), không bị lặp ở một nhánh. Mặt khác, Heuristic search sử dụng một chiến lược thông minh hơn để chọn ô điền tiếp theo nên khi ta sử dụng hàm Heuristic hợp lý, thời gian giải sẽ rút ngắn đi rất nhiều.
- Đối với những testcase càng ít gợi ý và số cho sẵn, thời gian cũng như bộ nhớ tiêu tốn càng lớn.

2 Chess Ranger

2.1 Giới thiệu đề bài toán

Đề bài đưa ra yêu cầu thiết kế và hiện thực giải thuật tìm kiếm để giải bài toán Chess Ranger, thực hiện bằng 2 giải thuật : Breadth-First Search (BFS) và 1 giải thuật heuristic bất kì :

- Tạo ra input phù hợp.
- Hiện thực có tính năng demo step-by-step để có hiển thị trực quan (như game).
- Bắt buộc sử dụng Python 3.
- Viết lại báo cáo nêu rõ quá trình tìm hiểu và hiện thực bài toán này.

2.2 Mô tả bài toán

Chess Ranger là một trò chơi đố logic với quy tắc đơn giản nhưng lời giải đầy thách thức.
Quy tắc:

- Bàn cờ có kích thước 8x8.
- Các quân cờ di chuyển như trong cờ vua tiêu chuẩn.
- Chỉ được thực hiện các nước đi ăn quân.
- Được phép ăn vua.
- Mục tiêu là tìm lộ trình ăn các quân sao cho chỉ còn lại duy nhất một quân trên bàn cờ.

2.3 Giới thiệu sơ lược các giải thuật

2.3.1 Breadth-First Search (BrFS)

Breadth-First Search (BrFS) là thuật toán duyệt theo chiều rộng, xem xét tất cả trạng thái cùng một mức trước khi tiếp tục xuống các trạng thái sau, luôn tìm ra lời giải tối ưu nếu tồn tại. Trong bài toán Chess Ranger, BFS hoạt động như sau:

1. Khởi tạo trạng thái ban đầu: Các quân cờ và vị trí ban đầu được lưu trữ trong ChessState.
2. Sử dụng hàng đợi deque: Các trạng thái được lưu vào hàng đợi để lần lượt xét duyệt.
3. Sinh trạng thái mới: Mỗi quân cờ tìm các nước đi hợp lệ (get_valid_moves) để ăn quân khác.
4. Duyệt theo chiều rộng: Các trạng thái mới được kiểm tra và thêm vào hàng đợi nếu chưa từng xuất hiện (visited).
5. Kết thúc khi đạt mục tiêu: Nếu trạng thái có duy nhất một quân cờ còn lại, BFS trả về danh sách các bước đi dẫn đến trạng thái đó.

2.3.2 A-Star Search (A*)

A* kết hợp tìm kiếm theo chiều rộng với đánh giá heuristic nhằm tối ưu hóa thời gian tìm kiếm. Cách thức hoạt động trong bài toán Chess Ranger:

1. Sử dụng hàm heuristic (heuristic(state)): Xác định h(n) = số quân cờ còn lại - 1, giúp đánh giá trạng thái nào gần trạng thái đích hơn.
2. Hàm f(n) = g(n) + h(n): Trong đó:
   g(n): số bước đi từ trạng thái ban đầu đến trạng thái hiện tại.
   h(n): số quân cờ còn lại trừ đi 1.
3. Mở rộng trạng thái mới (get_next_states()): Tạo các trạng thái tiếp theo dựa trên các nước đi hợp lệ.
4. Tránh trùng lặp: Sử dụng tập visited để tránh lặp lại các trạng thái đã xét.
5. Kết thúc khi còn một quân cờ: Khi đạt được trạng thái có một quân cờ duy nhất, thuật toán trả về danh sách nước đi tối ưu.

2.4 Tìm hiểu và hiện thực bài toán

- Trạng thái: Bản đồ là ma trận 8x8 đại diện cho bàn cờ vua, ví dụ:

.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	B	.	N	.	.	.
.	.	B	.	N	.	.	.
.	.	P	.	.	.	.	.
.	.	.	R	R	P	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

  - R, N, B, K, P, Q đại diện cho quân cờ Rook (Xe), Knight (Mã), Bishop (Tượng), King (Vua), Pawn (Tốt), Queen (Hậu).
  - Dấu . đại diện cho ô trống.

- Trạng thái khởi đầu - Một bàn cờ có nhiều quân cờ được đặt vào các vị trí ban đầu.
  - Các quân cờ có thể di chuyển theo luật cờ vua và ăn quân khác nếu có thể.

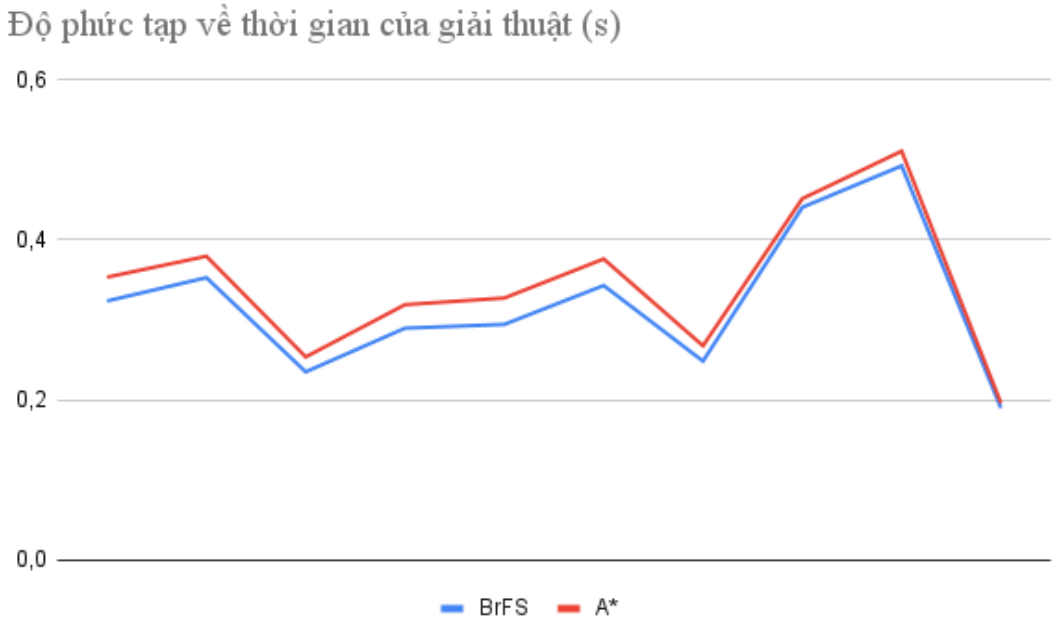
- Trạng thái đích - Khi trên bàn cờ chỉ còn lại duy nhất một quân cờ.

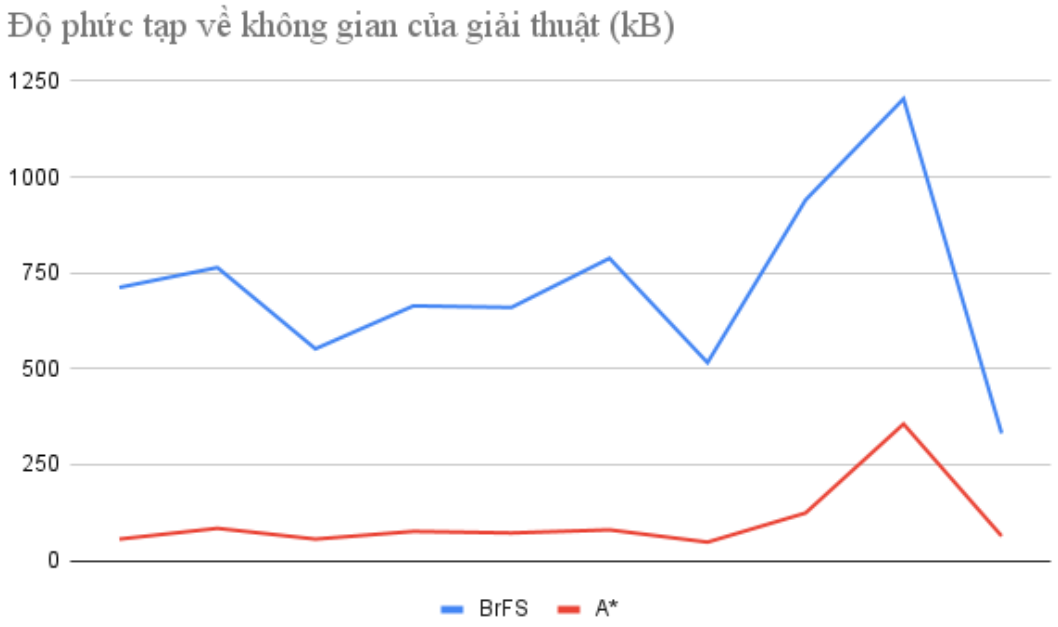
- Bước đi hợp lệ: Mỗi quân cờ di chuyển theo quy tắc của cờ vua: - Tốt (Pawn): Chỉ có thể ăn chéo.
  - Xe (Rook): Đi ngang hoặc dọc đến khi gặp quân cờ để ăn.
  - Tượng (Bishop): Đi chéo đến khi gặp quân cờ.
  - Mã (Knight): Nhảy theo hình chữ L.
  - Hậu (Queen): Kết hợp nước đi của Xe và Tượng.
  - Vua (King): Đi từng ô xung quanh.
  Nếu một quân có thể ăn quân khác, trạng thái mới được tạo ra.

2.5 Đánh giá hiệu năng các giải thuật

Testcase	Thời gian (s)		Bộ nhớ (KB)	
	BrFS	A*	BrFS	A*
1	0.3236	0.3534	712.0	56.00
2	0.3528	0.3796	764.0	84.00
3	0.2349	0.2536	552.0	56.00
4	0.2896	0.3190	664.0	76.00
5	0.2943	0.3273	660.0	72.00
6	0.3430	0.3762	788.0	80.00
7	0.2483	0.2675	516.0	48.00
8	0.4408	0.4517	940.0	124.0
9	0.4927	0.5111	1204	356.0
10	0.1896	0.1958	332.0	64.00

Bảng 2: Bảng so sánh thời gian và bộ nhớ giữa BrFS và A*





2.6 Kết luận và nhận xét

- Giải thuật BFS và giải thuật A* đều cho thời gian thực thi gần ngang nhau vì tính chất của câu đố này, số nước đi khá là hạn chế và ít, nên khó có sự chênh lệch về thời gian. Và do cách hoạt động của giải thuật A* nên đối với những câu đố ít nước đi này, thời gian xử lí sẽ thường nhiều hơn giải thuật BFS một ít.
- Về phần bộ nhớ, bởi vì giải thuật BFS lưu lại toàn bộ các nước đi có thể, nên lượng bộ nhớ tiêu tốn sẽ nhiều hơn. Còn về giải thuật A*, vì nó ưu tiên tìm nước đi có khả năng đến đích hơn, nên số nước đi có khả năng đến đích nó lưu lại cũng sẽ ít hơn, dẫn tới việc tiêu tốn bộ nhớ ít hơn so với BFS.

3 Báo cáo

Video thuyết trình:

[HK242-L02] Introduction-to-Artificial-Intelligence_Video

Link Github:

[HK242-L02] Introduction-to-Artificial-Intelligence_Code

4 Tài liệu tham khảo

- Tài liệu học tập của môn học
- https://www.puzzle-pipes.com/
- Minimum Remaining Values - Alooba. Tham khảo từ Minimum Remaining Values