

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



BÀI TẬP LỚN 1
NHẬP MÔN TRÍ TUỆ NHÂN TẠO

**APPLY SEARCHING ALGORITHM
FOR LOGIC PUZZLES**

GVHD:	Vương Bá Thịnh		
Lớp:	L01		
SV thực hiện:	Nguyễn Phúc Thịnh	1910565	
	Nguyễn Thành Đạt	1910113	
	Trịnh Minh Phúc	1911878	
	Hồ Vũ Đại Hải	1913241	



Mục lục

1	Phân công công việc	2
2	Sudoku	3
2.1	Luật chơi	3
2.2	Giải thuật	3
2.2.1	Sử dụng giải thuật Depth First Search	3
2.2.1.a	Ý tưởng	3
2.2.1.b	Trạng thái đầu tiên	3
2.2.1.c	Các giá trị hợp lệ	4
2.2.1.d	Hướng di chuyển	4
2.2.1.e	Hiện thực	4
2.2.1.f	Kết quả giải thuật	4
2.2.1.g	Chạy chương trình	5
2.2.2	Sử dụng giải thuật Simulated Annealing	6
2.2.2.a	Ý tưởng	6
2.2.2.b	Trạng thái ban đầu	6
2.2.2.c	Giá trị nhiệt	6
2.2.2.d	Hiện thực	6
2.2.2.e	Kết quả giải thuật	7
2.2.2.f	Chạy chương trình	8
2.3	Kết luận	9
3	Hitori	10
3.1	Luật chơi	10
3.2	Giải thuật	10
3.2.1	Sử dụng giải thuật Depth-First Search	10
3.2.1.a	Ý tưởng	10
3.2.1.b	Trạng thái đầu tiên	10
3.2.1.c	Các giá trị hợp lệ	10
3.2.1.d	Hướng di chuyển	10
3.2.1.e	Hiện thực	11
3.2.1.f	Kết quả giải thuật	11
3.2.1.g	Chạy chương trình	12
3.2.2	Sử dụng giải thuật Simulated Annealing	12
3.2.2.a	Ý tưởng	12
3.2.2.b	Trạng thái ban đầu	13
3.2.2.c	Giá trị nhiệt	13
3.2.2.d	Hiện thực	13
3.2.2.e	Kết quả giải thuật	13
3.2.2.f	Chạy chương trình	14
3.3	Kết luận	15
4	Video thuyết trình	15



1 Phân công công việc

STT	MSSV	Họ và tên	Công việc	Đánh giá công việc
1	1913241	Hồ Vũ Đại Hải	Chỉnh sửa và test code sudoku	25%
2	1910565	Nguyễn Phúc Thịnh	Làm báo cáo, Hiện thực code Sudoku theo giải thuật DFS và giải thuật SA	25%
3	1910113	Nguyễn Thành Đạt	Làm báo cáo, hiện thực code Hitori theo giải thuật DFS	25%
4	1911878	Trịnh Minh Phúc	Làm báo cáo, hiện thực code Hitori theo giải thuật SA	25%

2 Sudoku

2.1 Luật chơi

Sudoku là một trò chơi câu đố sắp xếp chữ số dựa trên logic theo tổ hợp. Mục tiêu của trò chơi là điền các chữ số vào một lưới 9×9 sao cho mỗi cột, mỗi hàng, và mỗi phần trong số chín lưới con 3×3 cấu tạo nên lưới chính (cũng gọi là "hộp", "khối", hoặc "vùng") đều chứa tất cả các chữ số từ 1 tới 9. Câu đố đã được hoàn thành một phần, người chơi phải giải tiếp bằng việc điền số. Mỗi câu đố được thiết lập tốt có một cách làm duy nhất.

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

Hình 1: Hình ảnh ví dụ

2.2 Giải thuật

Như hình 1 đã minh họa, sudoku luôn có trạng thái ban đầu gồm 1 lượng các ô đã được điền cố định mà không được thay đổi trên ma trận 9×9 và các ô còn lại được để trống, như vậy luôn luôn thiết lập được trạng thái ban đầu các giải thuật.

2.2.1 Sử dụng giải thuật Depth First Search

2.2.1.a Ý tưởng

Đây là giải thuật đơn giản nhất cũng như tối ưu nhất trong việc giải quyết bài toán sudoku. Ý tưởng của giải thuật là thực hiện giải tuần tự từng ô trống trong ma trận của bài toán thông qua hình thức ghép thử các giá trị hợp lệ và di chuyển sang ô tiếp theo. Khi không còn giá trị hợp lệ tại ô trống, thực hiện back-tracking về ô trước đó để thay đổi giá trị. Giải thuật kết thúc khi không còn ô trống (không còn bài toán cần được giải quyết).

2.2.1.b Trạng thái đầu tiên

Như đã nêu ở trên, DFS sử dụng trạng thái ban đầu là hình dạng đầu tiên của ma trận. Cụ thể, các ô được điền sẵn được xem như những bài toán đã được giải quyết còn các ô trống là những

bài toán cần được giải quyết.

2.2.1.c Các giá trị hợp lệ

Tại từng ô trống, thực hiện chọn các giá trị hợp lệ cho ô. Các giá trị hợp lệ sẽ tuân thủ theo quy tắc của trò chơi đó là những giá trị trong khoảng từ 1 tới 9 và chưa xuất hiện trong hàng, cột và khối tương ứng. Như vậy tại một ô có thể sẽ chỉ có 1 giá trị hợp lệ hoặc 1 tập giá trị hợp lệ.

2.2.1.d Hướng di chuyển

Giải thuật giải quyết các ô trống trên hàng trước theo thứ tự từ chỉ mục thấp tới cao. Việc di chuyển này là không bắt buộc, do có thể di chuyển theo hướng chỉ mục cột hoặc chỉ mục trong khối. Tuy nhiên, hoàn toàn có thể di chuyển theo hướng bất kì các vị trí mà không cần có độ liên đới về vị trí, nhưng cách làm này có khả năng cao đó là làm tăng số bước cần kiểm tra, do khi lấp đầy dần 1 khối, cột hay hàng, bộ giá trị cần kiểm tra tại các ô liền cận giảm, giảm đi số giá trị hợp lệ cần đi qua.

2.2.1.e Hiện thực

Giải thuật trên được hiện thực trên python, với dữ liệu đầu vào là 1 chuỗi 81 số trong tầm từ 1 tới 9 trong 1 file text độc lập. Gồm 2 class đó là Node và Sudoku. Node được dùng để lưu trữ thông tin và thao tác trực tiếp trên ma trận. Sudoku được dùng để hiện thực di chuyển tìm kiếm DFS, kiểm tra tính hợp lệ của hướng đi và thực hiện backtracking khi hết nước đi hợp lệ.

2.2.1.f Kết quả giải thuật

Thời gian chạy giải thuật trên 10 testcase khác nhau (tính bằng giây)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	0.7133	0.009	0.0778	0.0341	0.0169	0.1393	1.1329	0.0339	0.1037	0.0259
Lần 2	0.665	0.007	0.0618	0.0259	0.0209	0.1895	0.9836	0.0359	0.0877	0.0229
Lần 3	0.7258	0.008	0.0688	0.0299	0.0269	0.1705	0.9077	0.0349	0.0847	0.0219
AVG	0.7014	0.008	0.0695	0.03	0.0216	0.1664	1.0081	0.0349	0.0920	0.0236

Chú ý: Vì việc ghi lại step-by-step các trạng thái vào file txt ảnh hưởng đến thời gian chạy giải thuật nên số liệu này được đo khi không ghi lại trạng thái vào file txt

Không gian bộ nhớ sử dụng khi chạy giải thuật trên 10 testcase khác nhau (tính bằng Byte)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	70783	70656	74351	74025	72695	70751	66303	74375	72663	72665
Lần 2	70783	70656	74351	74025	72695	70751	66303	74375	72663	72665
Lần 3	70783	70656	74351	74025	72695	70751	66303	74375	72663	72665
AVG	70783	70656	74351	74025	72695	70751	66303	74375	72663	72665

Hiện thực trên các testcase, thời gian thực thi của giải thuật DFS giao động trong tầm 0.3 tới 2 giây. với mức tổn trên bộ nhớ đạt tối đa trong tầm 60 tới 70 KiB. Điểm chú ý ở đây là do DFS là tìm kiếm mù nên thứ tự và điều kiện tìm kiếm của mỗi lần kiểm là như nha, dẫn đến sự nhất trí trong thời gian chạy và không gian bộ nhớ sử dụng. Đáng chú ý khác, trừ lần chạy đầu tiên, các lần chạy sau gần như luôn đạt giá trị thời gian tốt hơn.

2.2.1.g Chạy chương trình

Hướng dẫn chi tiết cùng source code đã được đăng lên Github ở link: https://github.com/Noknight41/NMAI_BTL1. Người dùng có thể tải về và chạy theo hướng dẫn được ghi ở **Readme.md**.

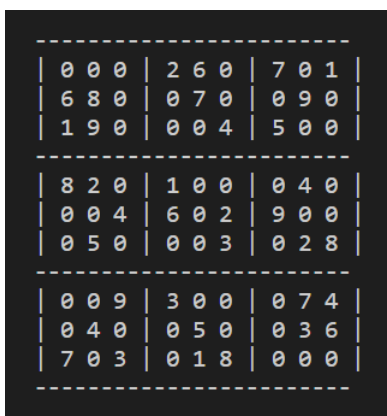
Hướng dẫn chạy chương trình python của giải thuật Depth-First Search bài toán Sudoku:

Bước 1: Mở 1 tab (hoặc cửa sổ) riêng biệt của Command Prompt/Powershell (Windows), hay Terminal (Linux và MacOS), di chuyển đến địa chỉ đặt các file .py của source code đã tải về. Di chuyển vào folder Sudoku: **cd Sudoku**

Bước 2: Chọn testcase và chạy chương trình python với cú pháp:
python sudoku_dfs.py [Tên file txt của input]
Ví dụ: **python sudoku_dfs.py test_1.txt**

Kết quả xuất ra của chương trình

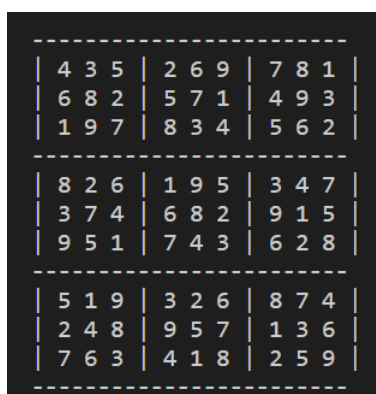
Chương trình xuất ra input bài toán Sudoku 9x9 (trạng thái đầu của bài toán)



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

Hình 2: Bài toán ban đầu, với 0 là ô trống

Chương trình xuất ra output lời giải cho bài toán Sudoku 9x9 (trạng thái đích của bài toán)



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

Hình 3: Đáp án bài toán sudoku

Bên dưới là thời gian chạy của giải thuật

--- 0.07876944541931152 seconds ---

Hình 4: Thời gian chạy giải thuật

Ngoài ra, qua mỗi bước, chương trình sẽ viết kết quả tạm thời ra file **demo_step_by_step_dfs.txt** để mô tả lại quá trình giải thuật tìm kiếm lời giải.

2.2.2 Sử dụng giải thuật Simulated Annealing

2.2.2.a Ý tưởng

Trước tiên, thực hiện lấp đầy các khối 3x3 theo qui tắc các giá trị trong khối luôn là duy nhất mà không quan tâm tới các ràng buộc trong cột và hàng.

Đặt 1 hàm giá trị heuristic cho trạng thái của ma trận với 1 giá trị mục tiêu đại diện cho trạng thái mục tiêu. khi chưa đạt được trạng thái tối ưu, sử dụng cách chọn bất kì 2 ô trong 1 khối và thực hiện giao hoán và kiểm tra kết quả giao hoán, nếu đạt giá trị trạng thái tốt hơn, ta chọn trạng thái đó làm trạng thái tiếp theo.

Nếu không, ta chọn trạng thái đó làm trạng thái tiếp theo ngẫu nhiên dựa trên xác suất của hàm nhiệt.

Hàm giá trị heuristic được tính dựa trên lấy tổng số ô trong ma trận trừ đi số ô độc nhất trong hàng và cột và lấy tổng 2 giá trị này với giá trị mục tiêu là 0.

2.2.2.b Trạng thái ban đầu

Tương tự như DFS, trạng thái đầu của giải thuật SA cũng là dạng đầu của bài giải. Tuy nhiên, bước đầu tiên không phải là tuần tự trên từng ô mà là lấp đầy ma trận bằng 1 đáp án thử. Có thể xem như mỗi lần thử cho giải thuật luyện kim và di chuyển từ đáp án thử trên thành đáp án thật

2.2.2.c Giá trị nhiệt

Giá trị nhiệt ban đầu sẽ được tính toán theo công thức:

$kT = D_x(E)$ với E là tập các giá trị trạng thái kết quả có thể có

Công thức xác định xác suất nhận giá trị dựa trên sai biệt nhiệt độ:

$\rho = e^{-\frac{\delta E}{kT}}$ với δE là sai biệt giữa hàm giá trị hiện tại và hàm giá trị được đề xuất thay thế.

Việc lựa chọn sẽ từ chọn random từ bộ số từ 0 tới 1, sau đó so sánh xem giá trị ρ có lớn, nếu có thì chọn trạng thái mới thay trạng thái cũ. Tương ứng với nó đồng nghĩa giá trị càng gần với giá trị mục tiêu càng có nhiều cơ hội được chọn và giá trị thấp hơn hiện tại sẽ không được chọn.

2.2.2.d Hiện thực

Việc hiện thực giải thuật trên được thể hiện thông qua hàm sudoku. Đáng lưu ý, hàm sudoku giờ đây lưu thêm 1 ma trận xác định các vị trí chứa giá trị cố định ở trạng thái ban đầu.

2.2.2.e Kết quả giải thuật

Thời gian chạy giải thuật trên 8 testcase khác nhau (tính bằng giây)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8
Lần 1	5.5363	0.1207	1.73996	15.1326	20.3635	38.4604	14.0628	36.2125
Lần 2	32.6249	0.1306	0.2452	48.2705	5.8268	105.7201	49.8438	2.2879
Lần 3	10.5672	0.1346	0.5438	74.0073	6.7362	61.9459	46.5503	2.2171
AVG	16.2428	0.1286	0.8430	45.8035	10.9755	68.7088	36.8190	13.5725

Chú ý: Vì việc ghi lại step-by-step các trạng thái vào file txt ảnh hưởng đến thời gian chạy giải thuật nên số liệu này được đo khi không ghi lại trạng thái vào file txt

Số trạng thái kiểm tra tương ứng với thời gian trên (lần)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8
Lần 1	76614	1930	28379	204306	255903	422371	154220	512515
Lần 2	506767	1988	3990	674943	92174	1498340	629933	34321
Lần 3	153641	2162	8688	1019008	102124	914719	652180	32565
AVG	245674	2026.667	13685.67	632752.3	150067	945143.3	478777.7	193133.7

Nhìn vào thời gian chạy của giải thuật Simulated Annealing, ta thấy mất đi tính nhất trí trong thời gian chạy, nguyên nhân do trạng thái ban đầu khởi tạo trong từng hợp là khác nhau và các hướng di chuyển là ngẫu nhiên, khiến thời gian chạy có thể từ 0.2 giây (hay xấp xỉ 2000 trạng thái) đến 100 giây (hay xấp xỉ 1 triệu trạng thái)

Không gian bộ nhớ sử dụng khi chạy giải thuật trên 8 testcase khác nhau (tính bằng Byte)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8
Lần 1	33792	34614	31864	35263	31523	33062	36637	36032
Lần 2	34918	36209	33887	34822	32897	29322	32567	35428
Lần 3	35439	32579	37023	33667	31192	35702	34933	33832
AVG	34716.33	34467.33	34258	34584	31870.67	32695.33	34712.33	35097.33

So với giải thuật DFS, giải thuật Simulated Annealing không sử dụng nhiều bộ nhớ (30 ~ 40 KiB), nguyên nhân là giải thuật chỉ lưu 1 trạng thái của Sudoku và thực hiện di chuyển trên trạng thái đó, thay vì lưu nhiều trạng thái cùng lúc nhằm thực hiện backtracking như DFS.

2.2.2.f Chạy chương trình

Hướng dẫn chi tiết cùng source code đã được đăng lên Github ở link: https://github.com/Noknight41/NMAI_BTL1. Người dùng có thể tải về và chạy theo hướng dẫn được ghi ở **Readme.md**.

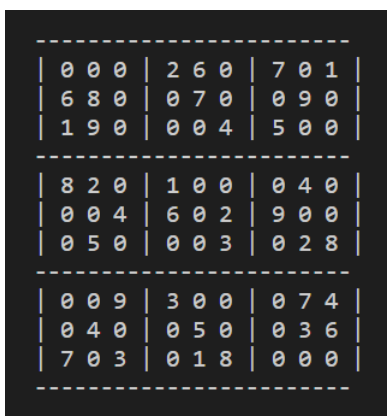
Hướng dẫn chạy chương trình python của giải thuật Simulated Annealing bài toán Sudoku:

Bước 1: Mở 1 tab (hoặc cửa sổ) riêng biệt của Command Prompt/Powershell (Windows), hay Terminal (Linux và MacOS), di chuyển đến địa chỉ đặt các file .py của source code đã tải về. Di chuyển vào folder Sudoku: **cd Sudoku**

Bước 2: Chọn testcase và chạy chương trình python với cú pháp:
python sudoku_sa.py [Tên file txt của input]
Ví dụ: **python sudoku_sa.py test_1.txt**

Kết quả xuất ra của chương trình

Chương trình xuất ra input bài toán Sudoku 9x9 (trạng thái đầu của bài toán)



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

Hình 5: Bài toán ban đầu, với 0 là ô trống

Chương trình xuất ra output lời giải cho bài toán Sudoku 9x9 (trạng thái đích của bài toán)



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

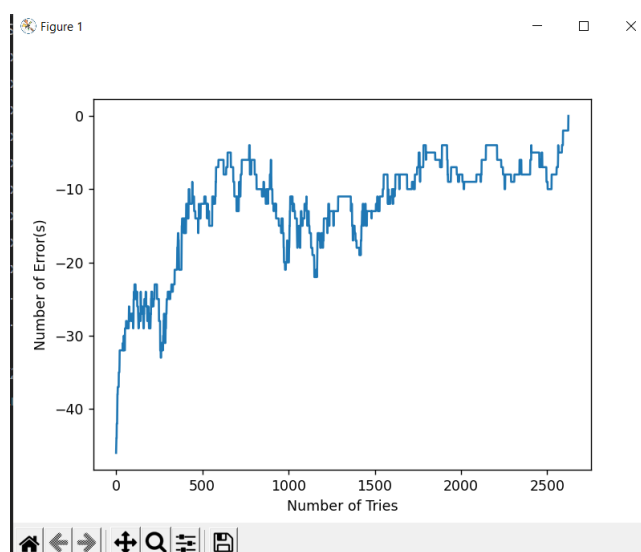
Hình 6: Đáp án bài toán sudoku

Bên dưới là thời gian chạy giải thuật

```
--- 0.19700288772583008 seconds ---
```

Hình 7: Thời gian chạy giải thuật

Bên cạnh đó, chương trình vẽ biểu đồ mô tả quá trình thay đổi giá trị heuristic từ trạng thái đầu đến trạng thái đích



Hình 8: Sơ đồ quá trình giá trị heuristic thay đổi

Ngoài ra, qua mỗi bước, chương trình sẽ viết kết quả tạm thời ra file **demo_step_by_step_sa.txt** để mô tả lại quá trình giải thuật tìm kiếm lời giải.

2.3 Kết luận

Giải thuật DFS có kết quả tối ưu hơn so với giải thuật Simulate Annealing trên phương diện thời gian, nhưng Simulate Annealing có ưu thế về không gian vùng nhớ.

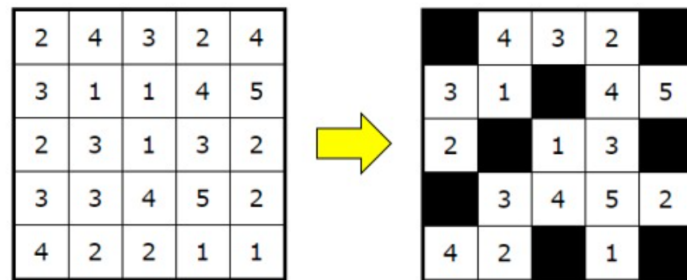
Điều này là do, DFS có chiến lược di chuyển rõ ràng và không phụ thuộc các giá trị ngẫu nhiên. Đối lại, DFS đòi hỏi việc duy trì trạng thái sống lâu hơn giải thuật Simulated Annealing, tạo ra một yêu cầu lớn về không gian bộ nhớ.

Hơn thế nữa các thay đổi của DFS mang tính nhỏ hơn về mức độ do chỉ yêu cầu điền vào ô thay vì là phép hoán đổi giá trị trên 1 ma trận. Hơn nữa việc tính toán trên DFS hầu như chỉ là các tính toán boolean, vì không sử dụng phương trình phức tạp nên giảm đi chi phí xử lý của hệ thống, về lâu dài giảm đáng kể thời gian chạy và vùng nhớ chiếm dụng của hệ thống.

3 Hitori

3.1 Luật chơi

Hitori là một trò chơi gồm một bảng cho sẵn tất cả các ô được điền số. Người chơi (bạn) phải tìm ra và tô đen những ô nào bị trùng trong cùng 1 hàng hoặc 1 cột (quy tắc là mỗi hàng và cột không được nhiều hơn một số giống nhau). Không có ô đen bên cạnh nhau, chỉ có đường chéo. Ô màu trắng còn lại phải liên tục (nghĩa là không có ô trắng nào bị cô lập bởi 4 ô đen trên - dưới - trái - phải).



Hình 9: Hình ảnh ví dụ

3.2 Giải thuật

3.2.1 Sử dụng giải thuật Depth-First Search

3.2.1.a Ý tưởng

Đây là giải thuật đơn giản để tìm ra kết quả trò chơi mà không cần phải tính toán. Mỗi ô có hai giá trị hoặc đen hoặc là trắng. Ý tưởng của giải thuật là thực hiện bôi trắng toàn bộ các ô trên bàn rồi kiểm tra xem trạng thái hiện tại có phải trạng thái kết thúc hay không, nếu không thì lần lượt đổi trạng thái cho từng ô, cho đến khi đạt được trạng thái kết thúc.

3.2.1.b Trạng thái đầu tiên

Như đã nêu ở trên, DFS sử dụng trạng thái ban đầu là trạng thái với tất cả các ô đã được bôi trắng.

3.2.1.c Các giá trị hợp lệ

Mỗi ô trên bàn có hai trạng thái là đen và trắng. Một ô trắng hợp lệ là ô trắng không bị bao quanh 4 phía trên dưới trái phải bởi 4 ô đen và trên cùng 1 hàng hoặc một cột thì giá trị ô trắng phải là duy nhất. Một ô đen hợp lệ bốn phía trên dưới trái phải không nằm kề ô đen, chỉ cho phép nằm kề ô đen ở đường chéo.

3.2.1.d Hướng di chuyển

Giải thuật ở đây sẽ vét cạn tất cả các trường hợp. Sẽ thay đổi trạng thái từ ô cuối cùng trên bàn và lần lượt cho đến ô đầu tiên cho đến khi đạt được trạng thái kết thúc

3.2.1.e Hiện thực

Giải thuật trên được hiện thực trên python, với dữ liệu đầu vào gồm n hàng n cột có chứa giá trị. Dữ liệu được lưu vào một mảng. Chúng ta tiến hành đệ quy đi từ ô đầu tiên của bàn rồi lần lượt đến ô cuối cùng. Khi các ô đã được gán trạng thái đầy đủ ta sẽ tiến hành kiểm tra xem trạng thái đó có phải là trạng thái kết thúc hay không? Nếu kết thúc trả về true, kết thúc việc tìm kiếm. Ngược lại nếu trả về false thì bước phía trước sẽ thay đổi trạng thái. Tuy nhiên việc chạy thuần DFS khiến hiệu năng giải thuật rất thấp bởi vì vậy chúng ta cần cải thiện giải thuật bằng cách bôi trắng tất cả các ô mà giá trị của nó duy nhất trên mỗi hàng và duy nhất trên mỗi cột.

3.2.1.f Kết quả giải thuật

Thời gian chạy giải thuật giải 10 testcase khác nhau (tính bằng giây)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	0.0523	0.3856	0.0456	0.012	0.3142	0.2555	0.0299	0.0658	0.0658	0.1173
Lần 2	0.0559	0.3842	0.0419	0.0119	0.2768	0.2068	0.0259	0.0758	0.0618	0.1226
Lần 3	0.0658	0.374	0.0429	0.0109	0.2793	0.2074	0.0269	0.0658	0.0678	0.1691
AVG	0.058	0.3813	0.0435	0.0116	0.2901	0.2315	0.02757	0.0691	0.0651	0.1363

Chú ý: Vì việc ghi lại step-by-step các trạng thái vào file txt ảnh hưởng đến thời gian chạy giải thuật nên số liệu này được đo khi không ghi lại trạng thái vào file txt

Không gian bộ nhớ sử dụng khi chạy giải thuật trên 10 testcase khác nhau (tính bằng Byte)

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	46521	46786	46744	45275	46744	46786	46373	46373	46746	46459
Lần 2	46786	46786	46677	45275	46585	46723	46744	46744	46746	46936
Lần 3	46786	46786	46744	45275	46744	46786	46638	46744	46746	46618
AVG	46697.67	46786	46721.67	45275	46691	46765	46585	46620.33	46746	46671

Hiện thực trên các testcase puzzle Hitori 5x5, thời gian thực thi của giải thuật DFS giao động trong tầm 0.03 tới 0.3 giây, với khối lượng bộ nhớ sử dụng dao động từ 40 đến 50 KiB. Giống như giải thuật DFS bên Sudoku, các số liệu đều nhất trí, không chênh lệch nhau nhiều giữa các lần thử

3.2.1.g Chạy chương trình

Hướng dẫn chi tiết cùng source code đã được đăng lên Github ở link: https://github.com/Noknight41/NMAI_BTL1. Người dùng có thể tải về và chạy theo hướng dẫn được ghi ở **Readme.md**.

Hướng dẫn chạy chương trình python của giải thuật Depth-First Search bài toán Hitori:

Bước 1: Mở 1 tab (hoặc cửa sổ) riêng biệt của Command Prompt/Powershell (Windows), hay Terminal (Linux và MacOS), di chuyển đến địa chỉ đặt các file .py của source code đã tải về. Di chuyển vào folder Hitori: **cd Hitori**

Bước 2: Chọn testcase và chạy chương trình python với cú pháp:
python Hitori_DFS.py [Tên file txt của input]
Ví dụ: **python Hitori_DFS.py test_1.txt**

Kết quả xuất ra của chương trình

Chương trình xuất ra input bài toán Hitori 5x5 (trạng thái đầu của bài toán) với một số được chuyển thành số âm, thành output của bài toán cùng với visualization đơn giản của lời giải

```
-5  2 -4  3 -3  x o x o x
 2  1  3  4  5  o o o o o
-5  5 -4  2 -3  x o x o x
 1 -5  2  5  3  o x o o o
 5  3  4 -4  2  o o o x o
```

Hình 10: Đáp án của bài toán Hitori

Giải thích: Ma trận bên trái là bài toán gốc có các số âm là các vị trí cần được tô đen trong bài giải. Ma trận bên phải là 1 visual đơn giản cho đáp án bài toán với x là ô cần tô đen, o là ô để tô trắng

Bên dưới là thời gian chạy giải thuật

```
--- 0.3437685966491699 seconds ---
```

Hình 11: Thời gian chạy giải thuật

Ngoài ra, qua mỗi bước, chương trình sẽ viết kết quả tạm thời ra file **demo_step_by_step_dfs.txt** để mô tả lại quá trình giải thuật tìm kiếm lời giải.

3.2.2 Sử dụng giải thuật Simulated Annealing

3.2.2.a Ý tưởng

Đầu tiên, hàm hiện thực việc kiểm tra hết bảng để tìm các ô có giá trị trùng nhau ngang hay dọc. Những ô có giá trị không bị trùng được đánh là ô trắng, còn lại sẽ cho vào một danh sách để thuật toán hoạt động. Hàm sẽ tiếp tục bằng việc tìm các trường hợp 3 ô trùng làm một góc vuông ở ngay góc rìa của bảng và trường hợp ba ô trùng theo hàng ngang hay dọc. Sau khi giải nhanh những trường hợp đặc biệt, một danh sách các ô còn chưa xử lý sẽ được giải quyết bằng giải thuật luyện kim.

3.2.2.b Trạng thái ban đầu

Tương tự như DFS, trạng thái đầu của giải thuật SA cũng là dạng đầu của bài giải. Khác biệt chính ngoài giải thuật còn thêm việc xử lý trường hợp đặc biệt để giảm số ô cần phải giải.

3.2.2.c Giá trị nhiệt

Giá trị nhiệt ban đầu sẽ được tính toán theo công thức:

$kT = D_x(E)$ với E là tập các giá trị trạng thái kết quả có thể có

Công thức xác định xác suất nhận giá trị dựa trên sai biệt nhiệt độ:

$\rho = e^{-\frac{\delta E}{kT}}$ với δE là sai biệt giữa hàm giá trị hiện tại và hàm giá trị được đề xuất thay thế.

Việc lựa chọn sẽ từ chọn random từ bộ số từ 0 tới 1, sau đó so sánh xem giá trị ρ có lớn, nếu có thì chọn trạng thái mới thay trạng thái cũ. Tương ứng với nó đồng nghĩa giá trị càng gần với giá trị mục tiêu càng có nhiều cơ hội được chọn và giá trị thấp hơn hiện tại sẽ không được chọn.

3.2.2.d Hiện thực

Khi hiện thực, hàm có lưu trữ thêm một mảng để hiện thực giải thuật kèm với một mảng lưu giữ bảng ban đầu.

3.2.2.e Kết quả giải thuật

Bảng 1: Bảng thời gian thực hiện giải thuật kèm với số trạng thái kiểm tra:

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	0.1102	1.4587	2.5571	0.8163	0.0844	0.3163	1.0119	2.5247	1.2536	1.5830
Lần 2	0.0806	2.4260	1.2315	3.9382	0.5748	3.2546	6.5349	1.6648	1.0020	18.3587
Lần 3	0.1732	0.3733	2.2542	3.1548	0.0424	6.7608	2.4431	2.7258	1.3381	4.8836
AVG	0.1214	1.4193	2.0143	2.6365	0.2338	3.4439	3.3300	2.3051	1.1979	8.2751

Chú ý: Vì việc ghi lại step-by-step các trạng thái vào file txt ảnh hưởng đến thời gian chạy giải thuật nên số liệu này được đo khi không ghi lại trạng thái vào file txt

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	1365	16758	28489	10077	965	3488	11347	26615	15147	19278
Lần 2	1031	26143	15164	44622	6548	40338	74412	20949	12979	228201
Lần 3	2267	4702	28047	40941	523	82586	32415	36168	17819	59862
AVG	1554.3	15867.7	23900	31880	2678.7	42137.3	39391.3	27910.7	15315	102447

Với giải thuật Simulated Annealing thì không có duy trì thời gian hiện thực đồng nhất, do tính ngẫu nhiên trong việc chọn và ra quyết định hướng đi của giải thuật.

Bảng 2: Không gian bộ nhớ sử dụng khi chạy giải thuật trên 8 testcase khác nhau (Byte)

So với giải thuật DFS, giải thuật Simulated Annealing không sử dụng nhiều bộ nhớ (khoảng 22 KiB), nguyên nhân là giải thuật chỉ lưu 1 trạng thái của Bảng và thực hiện di chuyển trên bảng, thay vì lưu nhiều trạng thái cùng lúc như DFS.



	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Lần 1	19778	22061	22509	22517	21917	22646	21989	22573	22085	22601
Lần 2	19755	22557	22013	22021	20928	22709	21926	22447	22022	22518
Lần 3	20852	22494	21981	21095	20991	22646	22485	22573	22518	22601
AVG	20128.3	22370.7	22167.7	21877.7	21278.7	22667.0	22133.3	22531.0	22208.3	22573.3

3.2.2.f Chạy chương trình

Hướng dẫn chi tiết cùng source code đã được đăng lên Github ở link: https://github.com/Noknight41/NMAI_BTL1. Người dùng có thể tải về và chạy theo hướng dẫn được ghi ở **Readme.md**.

Hướng dẫn chạy chương trình python của giải thuật Simulated Annealing bài toán Hitori:

Bước 1: Mở 1 tab (hoặc cửa sổ) riêng biệt của Command Prompt/Powershell (Windows), hay Terminal (Linux và MacOS), di chuyển đến địa chỉ đặt các file .py của source code đã tải về. Di chuyển vào folder Hitori: **cd Hitori**

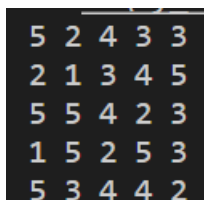
Bước 2: Chọn testcase và chạy chương trình python với cú pháp:

python Hitori_SA.py [Tên file txt của input]

Ví dụ: **python Hitori_SA.py test_1.txt**

Kết quả xuất ra của chương trình

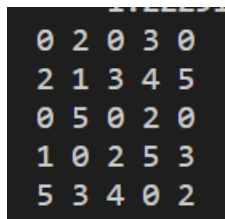
Chương trình xuất ra input bài toán Hitori (trạng thái đầu của bài toán)



5	2	4	3	3
2	1	3	4	5
5	5	4	2	3
1	5	2	5	3
5	3	4	4	2

Hình 12: Bài toán ban đầu

Chương trình xuất ra output lời giải cho bài toán Hitori (trạng thái đích của bài toán)



0	2	0	3	0
2	1	3	4	5
0	5	0	2	0
1	0	2	5	3
5	3	4	0	2

Hình 13: Đáp án bài toán Hitori

Bên dưới là thời gian chạy giải thuật

--- 1.2229104042053223 seconds ---

Hình 14: Thời gian chạy giải thuật

Ngoài ra, qua mỗi bước, chương trình sẽ viết kết quả tạm thời ra file **demo_step_by_step_sa.txt** để mô tả lại quá trình giải thuật tìm kiếm lời giải.

3.3 Kết luận

So với giải thuật Simulate Annealing, DFS mang tính ổn định hơn về thời gian xử lý và thuật toán để xử lý do phương thức di chuyển rõ ràng và đơn giản. Bù lại, DFS phải duy trì vùng nhớ lớn hơn chứa nhiều trạng thái cũ so với SA chỉ duy trì và thực hiện trên cùng trạng thái.

Kết quả thể hiện vậy do DFS trong bài toán không cần xử lý phức tạp để phát hiện đỉnh giả, làm mất đi ưu điểm mạnh trong giải thuật SA. Kèm với việc xử lý để ra quyết định kế tiếp, Simulate Annealing hoạt động kém hiệu quả cho bài toán dạng này so với DFS.

Đây là các tiêu biểu của giải thuật SA so với DFS, cho thấy vấn đề về không gian và thời gian là tương tự như với bài toán Sudoku.

4 Video thuyết trình

Link Video thuyết trình : <https://www.youtube.com/watch?v=6byr8VwscgU>

Tài liệu tham khảo

- [1] Stack Overflow - *Questions* [Online] Truy cập lần cuối 3/4/2022 : <https://stackoverflow.com/questions/>
- [2] Wikipedia - *Depth-first search* [Online] Truy cập lần cuối 3/4/2022: https://en.wikipedia.org/wiki/Depth-first_search
- [3] Wikipedia - *Hill Climbing* [Online] Truy cập lần cuối 3/4/2022: https://en.wikipedia.org/wiki/Hill_climbing
- [4] Wikipedia - *Simulated Annealing* [Online] Truy cập lần cuối 3/4/2022: https://en.wikipedia.org/wiki/Simulated_annealing
- [5] Puzzle.com - *Hitori* [Online] Truy cập lần cuối 3/4/2022: <https://www.puzzle-hitori.com/>
- [6] Puzzle.com - *Sudoku* [Online] Truy cập lần cuối 3/4/2022: <https://www.puzzle-sudoku.com/>