ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KH VÀ KT MÁY TÍNH

ASSiGNMENT1 - INTRODUCTION TO AI: SEARCH

BÁO CÁO ASSIGNMENT1

GVHD: Vương Bá Thịnh
Lớp L01

SV thực hiện: Võ Quý Long 1914011
Hà Hải Thiên Sơn 1811193
Nguyễn Văn Khoa 1913817

Tp. Hồ Chí Minh, Tháng 3/2022

Contents

1 Lý thuyết giải thuật Search áp dụng để giải

1.1 Depth-first Search

Tìm kiếm theo chiều sâu (tiếng Anh: Depth-first search - DFS) là một thuật toán duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị. Thuật toán khởi đầu tại gốc (hoặc chọn một đỉnh nào đó coi như gốc) và phát triển xa nhất có thể theo mỗi nhánh.

Thông thường, DFS là một dạng tìm kiếm thông tin không đầy đủ mà quá trình tìm kiếm được phát triển tới đỉnh con đầu tiên của nút đang tìm kiếm cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước.

1.2 Hill Climbing

Trong khoa học máy tính, giải thuật Hill Climbing là một kỹ thuật tối ưu toán học thuộc họ tìm kiếm cục bộ. Nó thực hiện tìm một trạng thái tốt hơn trạng thái hiện tại để mở rộng. Để biết trạng thái tiếp theo nào là lớn hơn, nó dùng một hàm H để xác định trạng thái nào là tốt nhất. Mô tả thuật toán:

- B1: Xét trạng thái đầu: Nếu là đích => dừng

  Ngược lại, thiết lập trạng thái bắt đầu = trạng thái hiện tại.

- B2: Lựa một luật để áp dụng vào trạng thái hiện tại để sinh ra một trạng thái mới.

- B3: Xem xét trạng thái mới này:

  Nếu là đích => dừng.

  Nếu không phải là đích nhưng tốt hơn trạng thái hiện tại thì thiết lập trạng thái hiệu t là trạng thái mới.

  Nếu không tốt hơn thì đến trạng thái mới tiếp theo

- Lặp đến khi: gặp đích hoặt không còn luật nào nữa chưa được áp dụng vào trạng thái hiện tại.

2 Game 1: Sudoku

2.1 Giới thiệu về game và luật chơi

- Sudoku là câu đố trí tuệ có hình dạng lưới 9x9, gồm những ô vuông nhỏ tạo thành một ô vuông lớn, mỗi 9 ô vuông nhỏ sẽ tạo thành một khối vuông 3x3, lần lượt ta có 9 khối vuông tạo thành một ô vuông lớn hoàn chỉnh. Nhiệm vụ của người chơi vô cùng đơn giản, điền các con số có một chữ số vào vị trí còn trống trên lưới.
- Một lưới Sudoku sẽ được điền trước những con số ban đầu, là chìa khóa để người chơi điền thêm những chữ số tiếp theo. Khi đã hoàn thành lời giải, lưới Sudoku sẽ trở thành một ma trận hình vuông Latin, tuân theo quy luật: mỗi số nguyên duy nhất sẽ không xuất hiện hai lần trong cùng một hàng, cột hoặc bất kỳ một trong chín khối vuông 3×3 nào của bảng trò chơi 9x9.

2.2 Chiến thuật giải bài toán

2.2.1 Depth-first Search

2.2.1.a Cách giải trờ chơi

- Áp dụng lý thuyết của dfs ta duyệt từng hàng của ma trận 9x9. B1: Bắt đầu ở hàng đầu tiên tại các ô trống ta sẽ điền 1 giá trị hợp lệ từ 0-9 (không vi phạm hàng, cột , ô 3x3) vào đó và chuyển sang ô tiếp theo cùng hàng
- B2: Nếu các ô trong hàng vẫn điền được các số hợp lệ thì ta sẽ tiếp tục cho đến khi hàng không còn ô nào trống. Nếu không còn số hợp lệ, ta quay lại ô trước đó và điền lại giá trị hợp lệ khác. Nếu không còn thì cứ tiếp tục quay lại ô trước đó nữa
- B3: Sau khi điền xong 1 hàng ta lặp lại tương tự với các hàng khác cho đến khi điền hết.

2.2.1.b Các testcase mẫu

- **Testcase 1**

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

```
[Running] python -u "c:\Học Tập\HCMUT\212\AI\Assignment1\DFS Sudoku\Sudoku.py"
Testcase 1
Hang\Cot   1 2 3   4 5 6   7 8 9
 - - - - - - - - - - - - - - -
1        | 4 7 3 | 5 1 9 | 6 2 8
2        | 2 9 5 | 8 6 3 | 7 1 4
3        | 8 6 1 | 7 4 2 | 9 3 5
 - - - - - - - - - - - - - - -
4        | 9 4 7 | 1 3 8 | 2 5 6
5        | 5 1 2 | 6 9 7 | 8 4 3
6        | 6 3 8 | 4 2 5 | 1 9 7
 - - - - - - - - - - - - - - -
7        | 7 8 9 | 3 5 1 | 4 6 2
8        | 3 2 6 | 9 8 4 | 5 7 1
9        | 1 5 4 | 2 7 6 | 3 8 9
Mem memorry usage = 28

[Done] exited with code=0 in 1.172 seconds
```

- **Testcase 10**

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

```
[Running] python -u "c:\Học Tập\HCMUT\212\AI\Assignment1\DFS Sudoku\Sudoku.py"
Testcase 10
Hang\Cot   1 2 3   4 5 6   7 8 9
 - - - - - - - - - - - - - - -
1        | 7 1 9 | 3 4 6 | 2 8 5
2        | 3 4 6 | 2 5 8 | 1 9 7
3        | 5 2 8 | 9 1 7 | 4 3 6
 - - - - - - - - - - - - - - -
4        | 9 3 4 | 1 7 2 | 6 5 8
5        | 1 8 7 | 4 6 5 | 3 2 9
6        | 6 5 2 | 8 9 3 | 7 1 4
 - - - - - - - - - - - - - - -
7        | 4 6 3 | 5 2 9 | 8 7 1
8        | 8 9 1 | 7 3 4 | 5 6 2
9        | 2 7 5 | 6 8 1 | 9 4 3
Mem memorry usage = 28

[Done] exited with code=0 in 0.793 seconds
```

- **Testcase12**

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

```
[Running] python -u "c:\Học Tập\HCMUT\212\AI\Assignment1\DFS Sudoku\Sudoku.py"
Testcase 12
Hang\Cot   1 2 3   4 5 6   7 8 9
 - - - - - - - - - - - - -
1       | 2 8 6 | 3 9 1 | 5 4 7
2       | 7 3 1 | 5 6 4 | 8 9 2
3       | 5 4 9 | 2 7 8 | 3 1 6
 - - - - - - - - - - - - -
4       | 1 9 5 | 6 2 7 | 4 3 8
5       | 8 6 4 | 1 3 5 | 7 2 9
6       | 3 7 2 | 8 4 9 | 6 5 1
 - - - - - - - - - - - - -
7       | 4 2 3 | 7 1 6 | 9 8 5
8       | 9 5 7 | 4 8 2 | 1 6 3
9       | 6 1 8 | 9 5 3 | 2 7 4
Mem memorry usage = 28

[Done] exited with code=0 in 1.278 seconds
```

2.2.2 Hill Climbing

2.2.2.a Cách giải trò chơi

- B1: Bắt đầu ở hàng đầu tiên ta sẽ random những vị trí trống của hàng với những con số trống bất kỳ
- B2: Tại những vị trí ta đã random, theo thứ tự từ 0-9 ta sẽ lần lượt thay số đã random, nếu số lỗi (chạy hàm check lỗi của board 9x9) nhỏ hơn số lỗi của số random trước đó thì ta giữ nguyên số đã thay, nếu không ta sẽ điền lại số ramdon cũ
- B3: Lặp lại cho tới khi hết 1 hàng. Kiểm tra số lỗi nếu không bằng 0 (có vị trí điền sai) ta sẽ bắt đầu lại ở B1
- B4: Lặp lại các bước trên với các hàng còn lại. Nếu đã thực hiện việc random lại quá 200 lần, kết thúc giải thuật, kết quả: không tìm được đáp án

2.2.2.b Các testcase mẫu

- **Testcase 7**

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

```
[Running] python -u "c:\Học Tập\HCMUT\212\AI\Assignment1\Sudoku.py"
Testcase 7
The answer:
Hang\Cot   1 2 3   4 5 6   7 8 9
 - - - - - - - - - - - - - - -
1        | 5 2 3 | 7 1 4 | 8 6 9
2        | 6 9 7 | 8 2 5 | 1 3 4
3        | 1 8 4 | 3 9 6 | 2 7 5
 - - - - - - - - - - - - - - -
4        | 8 3 5 | 9 6 2 | 4 1 7
5        | 2 7 1 | 4 5 3 | 6 9 8
6        | 4 6 9 | 1 8 7 | 5 2 3
 - - - - - - - - - - - - - - -
7        | 3 1 8 | 6 4 9 | 7 5 2
8        | 7 5 6 | 2 3 8 | 9 4 1
9        | 9 4 2 | 5 7 1 | 3 8 6
Mem memorry usage = 16

[Done] exited with code=0 in 2.786 seconds
```

- **Testcase 13**

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

```
[Running] python -u "c:\Học Tập\HCMUT\212\AI\Assignment1\Sudoku.py"
Testcase 13
The answer:
Hang\Cot   1 2 3   4 5 6   7 8 9
 - - - - - - - - - - - - - - -
1       | 1 2 3 | 6 9 4 | 7 8 5
2       | 4 8 6 | 1 5 2 | 3 9 4
3       | 6 4 9 | 3 7 8 | 1 5 2
 - - - - - - - - - - - - - - -
4       | 3 7 4 | 5 1 9 | 8 1 2
5       | 2 1 5 | 7 8 3 | 6 3 9
6       | 6 3 8 | 4 1 6 | 5 7 3
 - - - - - - - - - - - - - - -
7       | 8 2 1 | 2 6 5 | 9 4 7
8       | 3 5 7 | 8 4 1 | 2 2 6
9       | 7 4 2 | 5 3 9 | 8 6 1
Mem memorry usage = 16

[Done] exited with code=0 in 35.167 seconds
```

- **Testcase 15**

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

```
[Running] python -u "c:\Học Tập\HCMUT\212\AI\Assignment1\Sudoku.py"
Testcase 15
The answer:
Hang\Cot   1 2 3   4 5 6   7 8 9
 - - - - - - - - - - - - - - -
1       | 9 8 2 | 5 3 4 | 1 6 7
2       | 6 1 7 | 8 2 9 | 5 4 3
3       | 4 3 5 | 6 1 7 | 8 9 2
 - - - - - - - - - - - - - - -
4       | 2 5 3 | 1 4 6 | 9 7 8
5       | 1 4 9 | 3 7 8 | 2 5 6
6       | 8 7 6 | 9 5 2 | 3 1 4
 - - - - - - - - - - - - - - -
7       | 5 9 4 | 7 8 3 | 6 2 1
8       | 3 2 1 | 4 6 5 | 7 8 9
9       | 7 6 8 | 2 9 1 | 4 3 5
Mem memorry usage = 16

[Done] exited with code=0 in 3.116 seconds
```

2.3 Đánh giá thời gian chạy và tiêu tốn bộ nhớ của 2 giải thuật

2.3.1 Tiêu tốn bộ nhớ sử

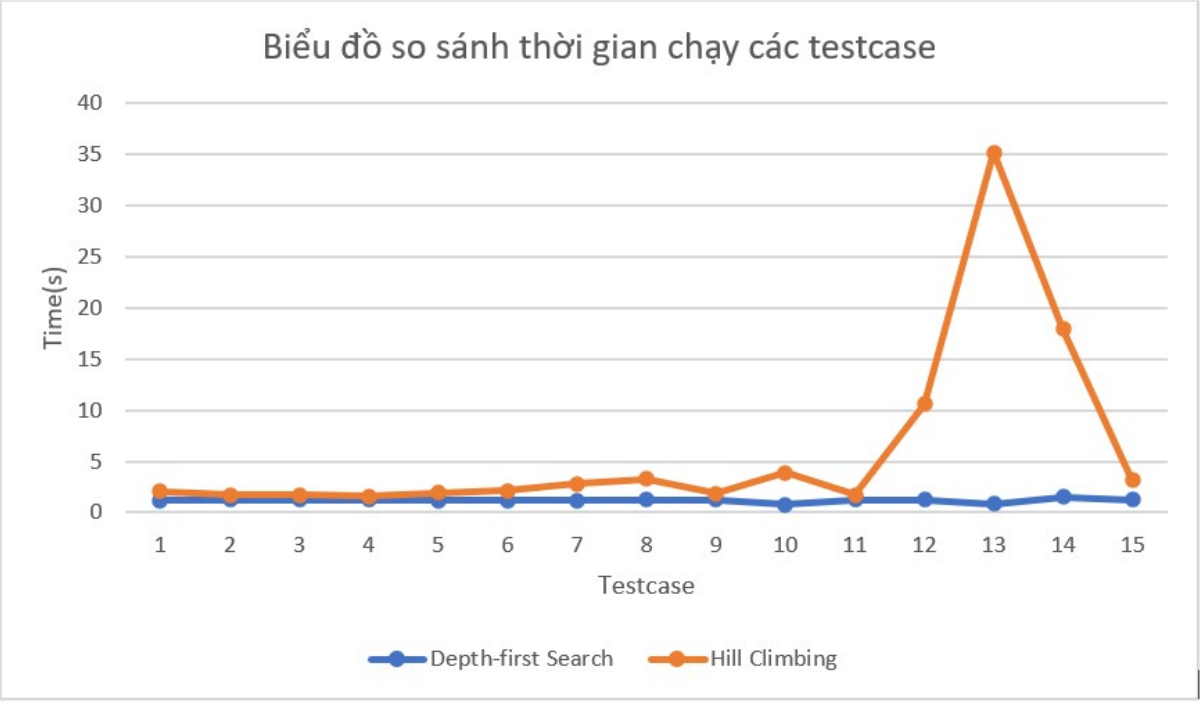
- Cả 2 hàm solve của 2 giải thuật đều truyền list của các list các hàng của ma trận 9x9 làm tham số. Nên các giữa các testcase của từng giải thuật tiêu tốn bộ nhớ sẽ giống nhau
- Đối với Hill Climbing: 16, DFS: 28
- Giải thuật DFS tiêu tốn bộ nhớ hơn giải thuật Hill Climbing là do khi chạy, DFS sẽ gọi đệ quy , điều này có nghĩa là sẽ cần không gian lưu trữ qua các lần đệ quy.

2.3.2 Thời gian chạy

Bẳng số liệu sau khi chạy 15 testcase (Các testcase trong code)

A	B	C
	Depth-first Search	Hill Climbing
1	1.172	2.07
2	1.238	1.75
3	1.233	1.751
4	1.219	1.609
5	1.206	1.956
6	1.201	2.123
7	1.204	2.786
8	1.224	3.278
9	1.235	1.88
10	0.793	3.913
11	1.219	1.694
12	1.278	10.564
13	0.852	35.167
14	1.512	17.896
15	1.22	3.116

Đồ thị đường tương ứng:



Nhận xét

- Dựa theo đồ thị ta có thể thấy, giải thuật Hill Climbing có thời gian chạy các testcase đa số đều chậm hơn so với DFS
- Thời gian chạy của Hill Climbing phụ thuộc vào độ khó của testcase. Ta có thể thấy ở những testcase khó như 12, 13, 14 thì thời gian chạy có thể rất lớn và có thể không giải được đáp án của trờ chơi. Lý do: do tính chất của giải thuật và trò chơi, ta có thể thấy biến số của trò chơi khá lớn trên 1 ma trận 9x9 và có thể ta sẽ phải điền rất nhiều số với rất nhiều trường hợp khác nhau. Cộng với tính chất ta chỉ chuyển sang trạng thái kế tiếp tốt hơn mà không lưu lại những trạng thái đã đi qua, nên rất dễ gặp lại những trạng thái cũ và không tìm được lời giải(testcase 13 và 14)
- Còn thời gian chạy của DFS lại khá ổn định đối với các testcase dễ, vừa, khó và thời gian giải cũng rất nhanh. Với việc lưu lại các trạng thái đã đi nên chắc chắn ta sẽ đi được tất cả các trạng thái có thể và tìm được lời giải

2.4 Kết luận

Dựa vào những phân tích trên, ta có thể kết luận giải thuật DFS là giải thuật hiệu quả hơn trong việc giải trờ chơi Sudoku, đảm bảo được kết quả và thời gian giải. Tuy nhiên tiêu tốn bộ nhớ nhiều hơn so với Hill Climbing

3 Game 2: Word search game

3.1 Giới thiệu về game và luật chơi

- Boggle là một trò chơi chữ được phát minh bởi Allan Turoff và ban đầu được phân phối bởi Parker Brothers . Trò chơi được chơi bằng cách sử dụng một mạng lưới xúc xắc có chữ bằng nhựa , trong đó người chơi tìm kiếm các từ theo chuỗi các chữ cái liền kề.
- Mỗi người chơi tìm kiếm các từ có thể được xây dựng từ các chữ cái của các khối liền kề tuần tự, trong đó các khối "liền kề" là các khối lân cận theo chiều ngang, chiều dọc và đường chéo. Các từ phải dài ít nhất ba chữ cái, có thể bao gồm số ít và số nhiều (hoặc các dạng bắt nguồn khác) riêng biệt, nhưng không được sử dụng cùng một khối chữ cái nhiều hơn một lần cho mỗi từ.

3.2 Chiến thuật giải bài toán

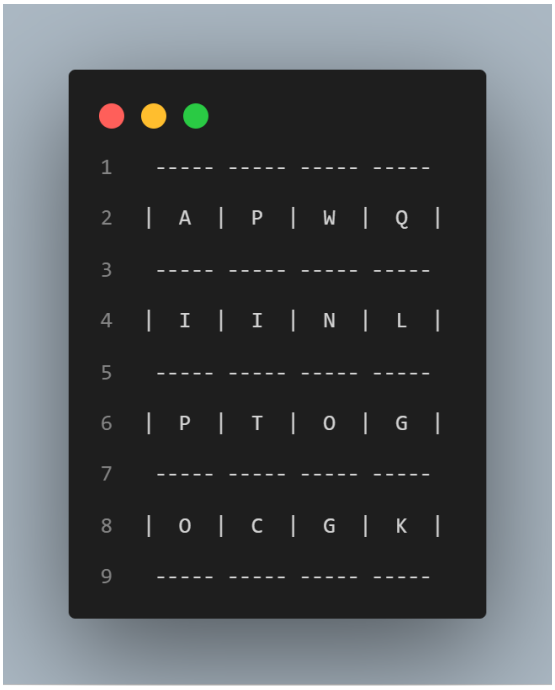
3.2.1 Depth-first Search

3.2.1.a Cách giải trờ chơi

- Áp dụng lý thuyết của DFS ta duyệt từng phần tử của ma trận game. B1: Ta khởi tạo một biến có kiểu là chuỗi và dùng một vòng lặp để duyệt trên từng phần tử của ma trận. Giải thuật DFS giúp ta duyệt theo các hướng xung quanh tại mỗi điểm cần duyệt, đồng thời biến kiểu chuỗi sẽ cộng dồn các giá trị là ký tự của từng ô
- B2: Nếu kiểm tra thấy chuỗi là một từ phù hợp có trong từ điển thì sẽ in ra kết quả ở console. Nếu trong quá trình duyệt mà ta kiểm tra thấy chuỗi là một từ vô nghĩa thì sẽ dừng lại, chuyển qua duyệt ô tiếp theo.
- B3: Cứ như thế sau khi duyệt hết ma trận chúng ta sẽ có mỗi danh sách chứa kết quả.

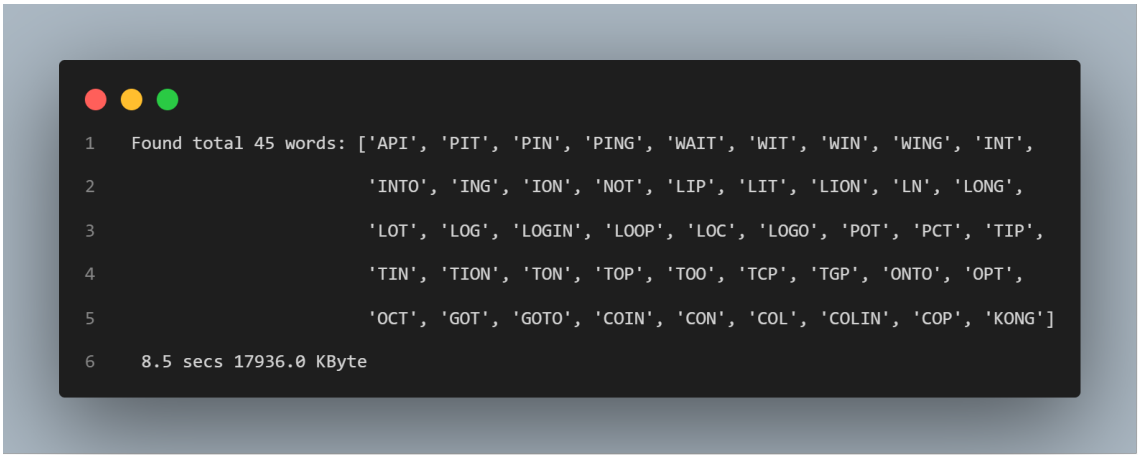
3.2.1.b Các testcase mẫu

- **Testcase 1**



```
1   ----- ----- ----- -----
2  | A | P | W | Q |
3   ----- ----- ----- -----
4  | I | I | N | L |
5   ----- ----- ----- -----
6  | P | T | O | G |
7   ----- ----- ----- -----
8  | O | C | G | K |
9   ----- ----- ----- -----
```

Solution:



```
1  Found total 45 words: ['API', 'PIT', 'PIN', 'PING', 'WAIT', 'WIT', 'WIN', 'WING', 'INT',
2                         'INTO', 'ING', 'ION', 'NOT', 'LIP', 'LIT', 'LION', 'LN', 'LONG',
3                         'LOT', 'LOG', 'LOGIN', 'LOOP', 'LOC', 'LOGO', 'POT', 'PCT', 'TIP',
4                         'TIN', 'TION', 'TON', 'TOP', 'TOO', 'TCP', 'TGP', 'ONTO', 'OPT',
5                         'OCT', 'GOT', 'GOTO', 'COIN', 'CON', 'COL', 'COLIN', 'COP', 'KONG']
6   8.5 secs 17936.0 KByte
```

- **Testcase 2**

```
1   ----- ----- ----- -----
2  | K | S | R | Y |
3   ----- ----- ----- -----
4  | Q | R | Z | P |
5   ----- ----- ----- -----
6  | W | K | C | N |
7   ----- ----- ----- -----
8  | N | U | P | X |
9   ----- ----- ----- -----
```

Solution:

```
1   Found total 8 words: ['SRC', 'YRS', 'CU', 'CUP', 'CPU', 'UN', 'UPC', 'PUNK']
2
3   2.5 secs 17440.0 KByte
```

- **Testcase 3**

```
 ----- ----- ----- -----
| U | W | S | U |
 ----- ----- ----- -----
| B | K | S | H |
 ----- ----- ----- -----
| S | K | W | S |
 ----- ----- ----- -----
| O | B | U | Y |
 ----- ----- ----- -----
```

Solution:

```
Found total 10 words: ['SUB', 'SKU', 'USB', 'HWY', 'WY', 'BUS', 'BUSH', 'BUSY', 'BUY', 'BUYS']

3.7 secs 16200.0 KByte
```

3.2.2 Hill Climbing

3.2.2.a Cách giải trò chơi

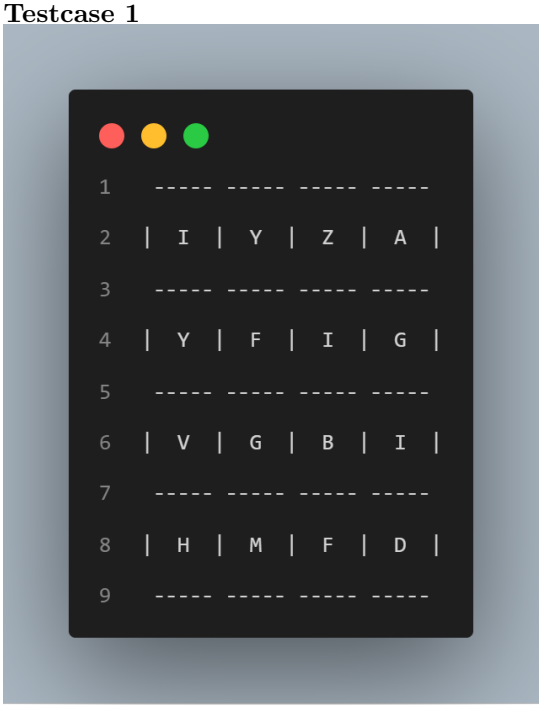
- B1: Ta sẽ dựa vào từ điển để tạo một cây Trie, đây là một cấu trúc dữ liệu sử dụng cây có thứ tự, dùng để lưu trữ một mảng liên kết của các xâu ký tự.
- B2: Ta sẽ tiến hành duyệt các phần tử trên ma trận game, một biến có kiểu dữ liệu là chuỗi được khởi tạo, khi duyệt qua từng phần tử, các kí tự trên phần tử đó sẽ được cộng dồn vào chuỗi. Dựa theo cây Trie đã được nạp từ điển, ta có thể biết được trạng thái hiện tại của chuỗi là tốt hay không. Cụ thể nếu trong quá trình duyệt, bên phía cây Trie không thể tìm được kí tự tương ứng với phần tử hiện tại trên ma trận thì sẽ hàm sẽ trả về trạng

thái trước đó. Khi duyệt đến nốt lá của cây mà vẫn hợp lệ thì ta sẽ lưu kí tự này vào một danh sách.

- B3: Lặp lại cho tới khi duyệt hết ma trận hoặc gặp điều kiện dừng. Ta sẽ đánh dấu vị trí đã duyệt vào một ma trận có kích thước tương ứng với ma trận game có kiểu dữ liệu là boolean. Quay lại B1 với ô liền kề trong hàng, nếu ô hiện tại là vị trí cuối của hàng thì sẽ quay lại B1 với ô đầu tiên hàng tiếp theo.
- B4: Lặp lại các bước trên với các ô còn lại.

3.2.2.b Các testcase mẫu

- **Testcase 1**



Solution:

```
Found "FIG" directions from (1,1)(F) go → →
Found "FIG" directions from (1,1)(F) go → ↙
Found "FBI" directions from (1,1)(F) go ↘ ↑
Found "FBI" directions from (1,1)(F) go ↘ →
Found "IBM" directions from (1,2)(I) go ↓ ↙
Found "GIF" directions from (1,3)(G) go ← ←
Found "GIG" directions from (1,3)(G) go ← ↙
Found "GIF" directions from (1,3)(G) go ↓ ↙
Found "GIF" directions from (2,1)(G) go ↗ ←
Found "GIG" directions from (2,1)(G) go ↗ →
Found "BIZ" directions from (2,2)(B) go ↑ ↑
Found "BIG" directions from (2,2)(B) go ↑ →
Found "BIG" directions from (2,2)(B) go ↑ ↙
Found "BIG" directions from (2,2)(B) go → ↑
Found "BID" directions from (2,2)(B) go → ↓
Found "BD" directions from (2,2)(B) go ↘
Found "IBM" directions from (2,3)(I) go ← ↙
Found "FBI" directions from (3,2)(F) go ↑ ↑
Found "FBI" directions from (3,2)(F) go ↑ →
Found "FIG" directions from (3,2)(F) go ↗ ↑
Found "DI" directions from (3,3)(D) go ↑
Found "DIG" directions from (3,3)(D) go ↑ ↑

0.1 secs 23060.0 KByte
```

- **Testcase 2**

Solution:

```
Found "WTO" directions from (0,1)(W) go  ↓ ↙
Found "WY" directions from (0,1)(W) go  ↘
Found "BAT" directions from (1,0)(B) go  ↑ ↘
Found "BOC" directions from (1,0)(B) go  ↓ →
Found "BOB" directions from (1,0)(B) go  ↓ ↓
Found "BOOB" directions from (1,0)(B) go  ↓ ↘ ←
Found "TAB" directions from (1,1)(T) go  ↖ ↓
Found "TBA" directions from (1,1)(T) go  ← ↑
Found "TOO" directions from (1,1)(T) go  ↙ ↘
Found "OCT" directions from (2,0)(O) go  → ↑
Found "CON" directions from (2,1)(C) go  ↓ ↗
Found "CONF" directions from (2,1)(C) go  ↓ ↗ ↗
Found "CORN" directions from (2,1)(C) go  ↓ → ↑
Found "NOR" directions from (2,2)(N) go  ↙ →
Found "BOB" directions from (3,0)(B) go  ↑ ↑
Found "BOC" directions from (3,0)(B) go  ↑ →
Found "BOOB" directions from (3,0)(B) go  → ↖ ↑
Found "BOOT" directions from (3,0)(B) go  → ↖ ↗
Found "BOOTY" directions from (3,0)(B) go  → ↖ ↗ →
Found "BOC" directions from (3,0)(B) go  → ↑
Found "BON" directions from (3,0)(B) go  → ↗
Found "BORN" directions from (3,0)(B) go  → → ↑
Found "OCT" directions from (3,1)(O) go  ↑ ↑
Found "ONTO" directions from (3,1)(O) go  ↗ ↖ ↙
Found "ROOT" directions from (3,2)(R) go  ← ↖ ↗
Found "RON" directions from (3,2)(R) go  ← ↗
Found "ROB" directions from (3,2)(R) go  ← ←
Found "ROBOT" directions from (3,2)(R) go  ← ← ↑ ↗

0.1 secs 23068.0 KByte
```

- **Testcase 3**

Solution:

3.3 Đánh giá thời gian chạy và tiêu tốn bộ nhớ của 2 giải thuật

3.3.1 Tiêu tốn bộ nhớ sử

- Cả 2 hàm findWords của 2 giải thuật đều truyền list của các list các hàng của ma trận 4x4 làm tham số. Nên các giữa các testcase của từng giải thuật sẽ tương giống nhau ngoại trừ cách in ra trên console.
- Đối với Hill Climbing: 19 Mb, DFS: 23 Mb

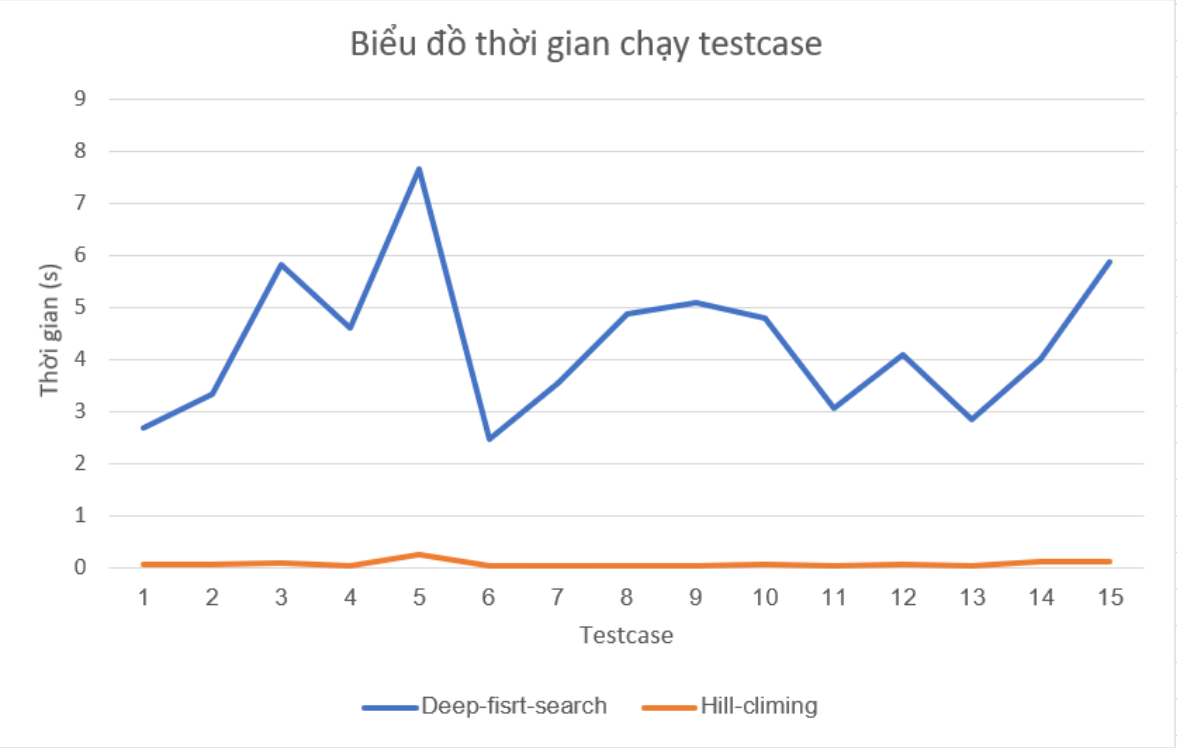
- Về lý thuyết giải thuật DFS tiêu tốn bộ nhớ hơn giải thuật Hill Climbing là do khi chạy, DFS sẽ gọi đệ quy nhiều hơn, điều này có nghĩa là sẽ cần không gian lưu trữ qua các lần đệ quy. Tuy nhiên trên thực tế giải thuật Hill Climing với cấu trúc cây Trie sẽ tốn thêm tài nguyên lưu trữ từ điển (Khoảng 10000 từ).

3.3.2 Thời gian chạy

Bẳng số liệu sau khi chạy 15 testcase (Các testcase trong code)

A	B	C
	Deep-fisrt-search	Hill-climing
1	2.6862	0.0657
2	3.3239	0.0741
3	5.8129	0.0832
4	4.6029	0.0501
5	7.6582	0.2627
6	2.4819	0.0457
7	3.5386	0.0326
8	4.8678	0.0463
9	5.0856	0.0408
10	4.7901	0.0592
11	3.0757	0.0509
12	4.0787	0.0553
13	2.8581	0.0462
14	4.0163	0.1191
15	5.8693	0.1164

Đồ thị đường tương ứng:



Nhận xét

- Dựa theo đồ thị ta có thể thấy, giải thuật Hill Climbing có thời gian chạy các testcase đa số nhanh hơn so với DFS
- Thời gian chạy của DFS phụ thuộc vào độ khó của testcase. Ta có thể thấy ở những testcase khó như 5, 3, 15 thì thời gian chạy có thể rất lớn và có thể gây tràn bộ nhớ stack. Lý do: do tính chất của giải thuật và trò chơi, ta có thể thấy biến số của trò chơi khá lớn trên 1 ma trận 4x4 và có thể ta sẽ phải duyệt qua rất nhiều rất nhiều vị trí với rất nhiều trường hợp khác nhau. Cộng với tính chất vét cạn (không chọn lọc) nên thời gian thực thi sẽ lâu hơn
- Còn thời gian chạy của Hill Climbing lại khá ổn định đối với các testcase dễ, vừa ,khó và thời gian giải cũng rất nhanh. Với việc duyệt theo từng phần tử và trượt trên cây Trie sẽ tìm được trạng thái tối ưu. Nếu không sẽ quay trở về trạng thái trước đó. Chắc chắn ta sẽ tìm được lời giải dựa trên từ điển được cho.

3.4 Kết luận

Dựa vào những phân tích trên, ta có thể kết luận giải thuật Hill Climb là giải thuật hiệu quả hơn trong việc giải trò chơi Boggle (Word search game), đảm bảo được kết quả và thời gian giải. Tuy nhiên tiêu tốn bộ nhớ nhiều hơn so với DFS do phải lưu trữ dữ liệu từ điển trên cấu trúc

cây Trie. Đối với những ma trận game lớn hơn nhiều so với mức 4x4 thì Hill Climb sẽ cho ra kết quả tuyệt đối về thời gian và bộ nhớ.