

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



NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

Báo cáo bài tập lớn:

Sử dụng giải thuật tìm kiếm để giải quyết Logic Puzzles

GVHD: Vương Bá Thịnh
SV thực hiện: Lương Hữu Phú Lợi – 1911545
Vũ Quang Huy – 1913578
Nguyễn Hoàng Thịnh – 1814172

Tp. Hồ Chí Minh, Ngày 26/03/2022



Mục lục

1	Giải thuật tìm kiếm	2
1.1	Depth-first search (DFS)	2
1.2	Heuristics Search - A *	3
2	Nonogram	4
2.1	Giới thiệu về trò chơi	4
2.2	Luật chơi	4
2.3	Phân tích trò chơi	6
2.4	Kết quả chạy chương trình	9
3	Tents	13
3.1	Giới thiệu về trò chơi	13
3.2	Luật chơi	13
3.3	Phân tích trò chơi - Solver đã hiện thực	14
3.3.1	Heuristic của A* cho bài Tents	16
3.3.2	Demo giải một bài tents 6x6	16
4	Đánh giá chung	19
4.1	Kết quả chạy được	19
4.1.1	Thông số máy	19
4.1.2	Excel:	19
4.2	Biểu đồ	22
4.3	Phân tích kết quả	31
4.3.1	Lý thuyết:	31
4.3.2	Thực nghiệm:	32
5	Mã nguồn và demo	33
6	Tài liệu tham khảo	33

1 Giải thuật tìm kiếm

1.1 Depth-first search (DFS)

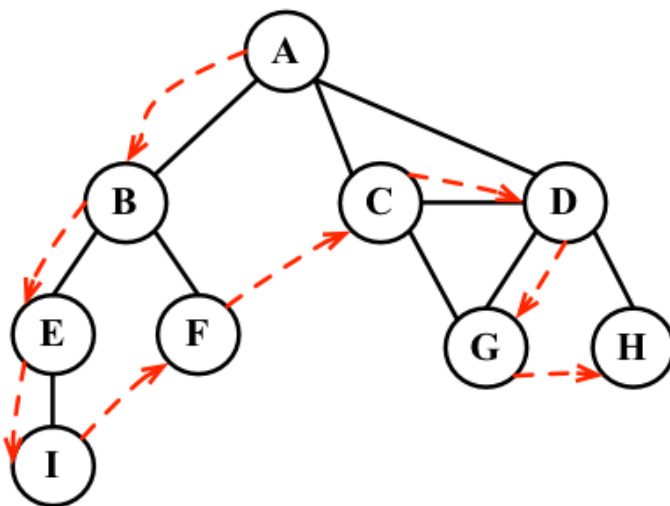
Đây là thuật toán tìm các đỉnh bằng cách duyệt theo chiều sâu.

Xuất phát từ 1 đỉnh và đi mãi cho đến khi không thể đi tiếp, sau đó đi về lại đỉnh đầu. Trong quá trình quay lại:

- Nếu gặp đường đi khác thì đi cho đến khi không đi tiếp được nữa
- Nếu không tìm ra đường đi nào khác thì ngừng việc tìm kiếm

Trong quá trình đi đến đỉnh khác, thuật toán sẽ lưu lại đỉnh cha vừa đi qua để khi đi ngược lại từ đỉnh Kết thúc đến đỉnh Xuất phát, ta có thể xem được đường đi từ đỉnh Kết thúc đến đỉnh Bắt Đầu.

Sở dĩ thuật toán này tìm được đường đi là nhờ vào cơ chế tô màu và lưu đỉnh cha. Quá trình tô màu khiến 1 đỉnh không thể xét 2 lần trở lên và có thể xem được đường đi từ đỉnh Kết Thúc đến đỉnh Xuất phát dựa vào việc lưu đỉnh cha.



Hình 1: DFS

Giải thích: Đỉnh A là đỉnh bắt đầu(gốc), bắt đầu đi về phía trái, tìm kiếm hết cây con bên trái mới chuyển sang tìm kiếm bên cây con bên phải. Quá trình tìm kiếm dừng lại khi đã duyệt hết toàn bộ đỉnh hoặc đến được đỉnh nghiệm.

Mã giả:

```
1 DFS(G,v)    (v is the vertex where the search starts)
2   Stack S := {};    (start with an empty stack)
3   for each vertex u, set visited[u] := false;
4   push S, v;
5   while (S is not empty) do
6     u := pop S;
7     if (not visited[u]) then
8       visited[u] := true;
9       for each unvisited neighbour w of u
10        push S, w;
11    end if
```

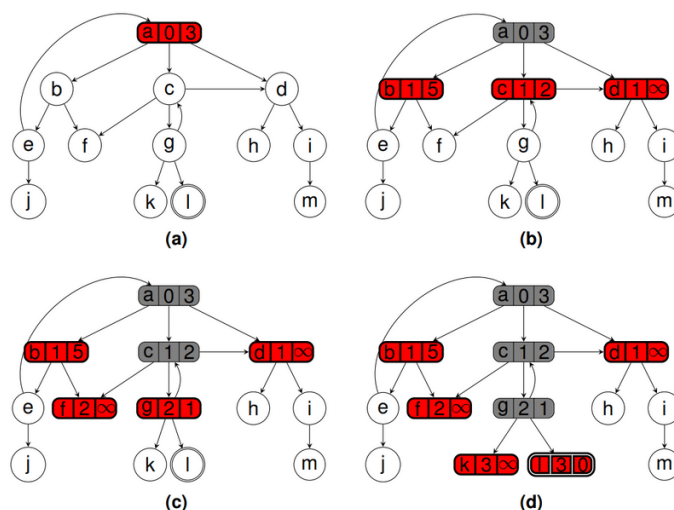
```
12   end while
13   END   DFS()
```

Vì tất cả các nút và đỉnh được truy cập, độ phức tạp thời gian trung bình cho DFS trên đồ thị là $O(V + E)$, trong đó V là số đỉnh và E là số cạnh. Độ phức tạp về không gian là $O(V)$.

1.2 Heuristics Search - A *

A* là một trong các thuật toán Informed Search (tìm kiếm theo kinh nghiệm). Nó xây dựng một cây gồm các đường dẫn từng phần bắt đầu từ đỉnh gốc cho đến khi một trong các đường dẫn kết thúc ở đỉnh nghiệm. Đỉnh cuối cùng của mỗi đường dẫn từng phần được chèn vào hàng đợi ưu tiên (priority queue) được sắp xếp theo chi phí của đường dẫn đến đỉnh đó cộng với khoảng cách còn lại tối nghiệm như ước tính của phương pháp heuristic:

$$f(n) = g(n) + h(n)$$



Hình 2: A-star

Giải thích: Đỉnh A là đỉnh bắt đầu(gốc), đỉnh I là đỉnh mục tiêu, các đỉnh màu trắng là các đỉnh chưa được duyệt, các đỉnh màu xám là các đỉnh đã được duyệt. Giá trị thứ 2 của đỉnh là khoảng cách từ đỉnh đó tới gốc, giá trị thứ 3 là giá trị ước tính của đỉnh tới đích được tính bằng hàm heuristic.

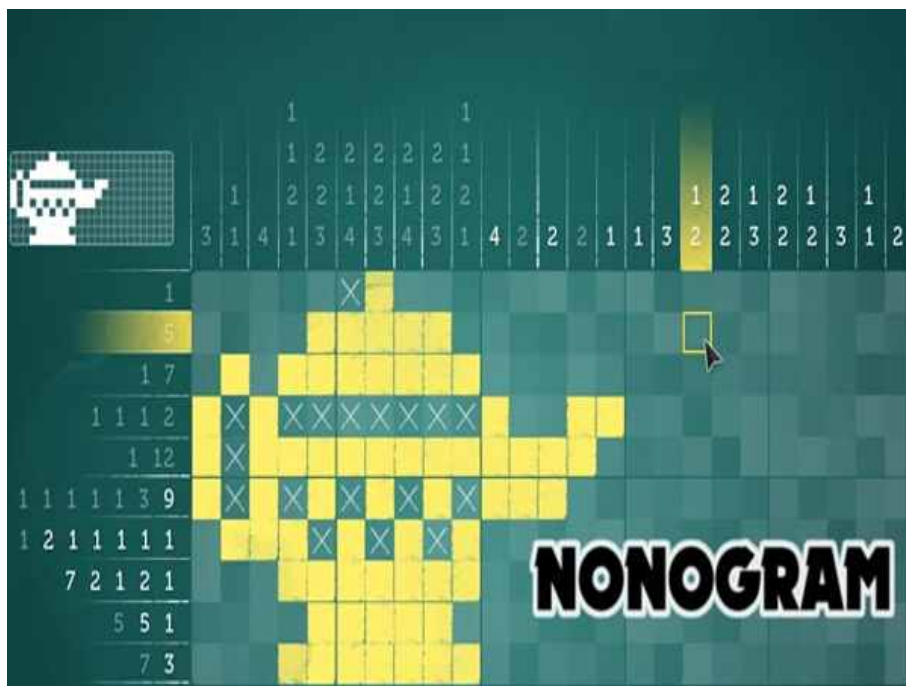
Mã giả:

Độ phức tạp thời gian của A* phụ thuộc vào đánh giá heuristic.

2 Nonogram

2.1 Giới thiệu về trò chơi

Nonogram là một trò chơi cổ điển truyền thống của nước Nhật Bản với những quy chuẩn nhất định. Game này còn được thế giới biết tới với những tên gọi khác nhau như Japanese Puzzles, Pic-a-Pix, Hanjie, Griddlers, Picross hay Logimage. . .

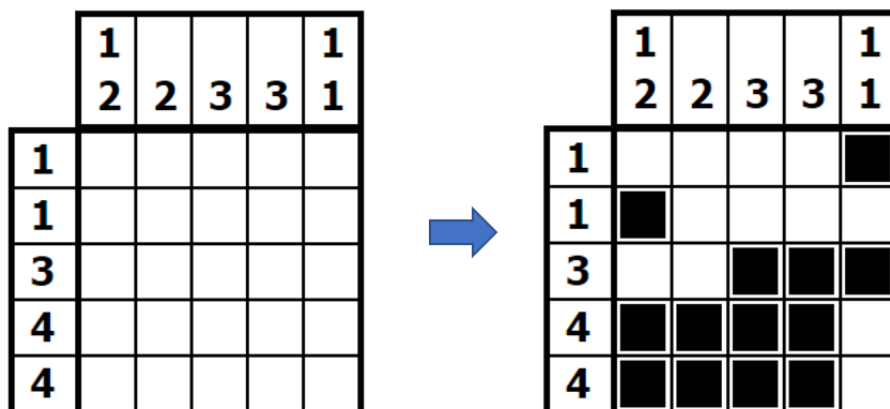


Hình 3: Ví dụ Nonogram

2.2 Luật chơi

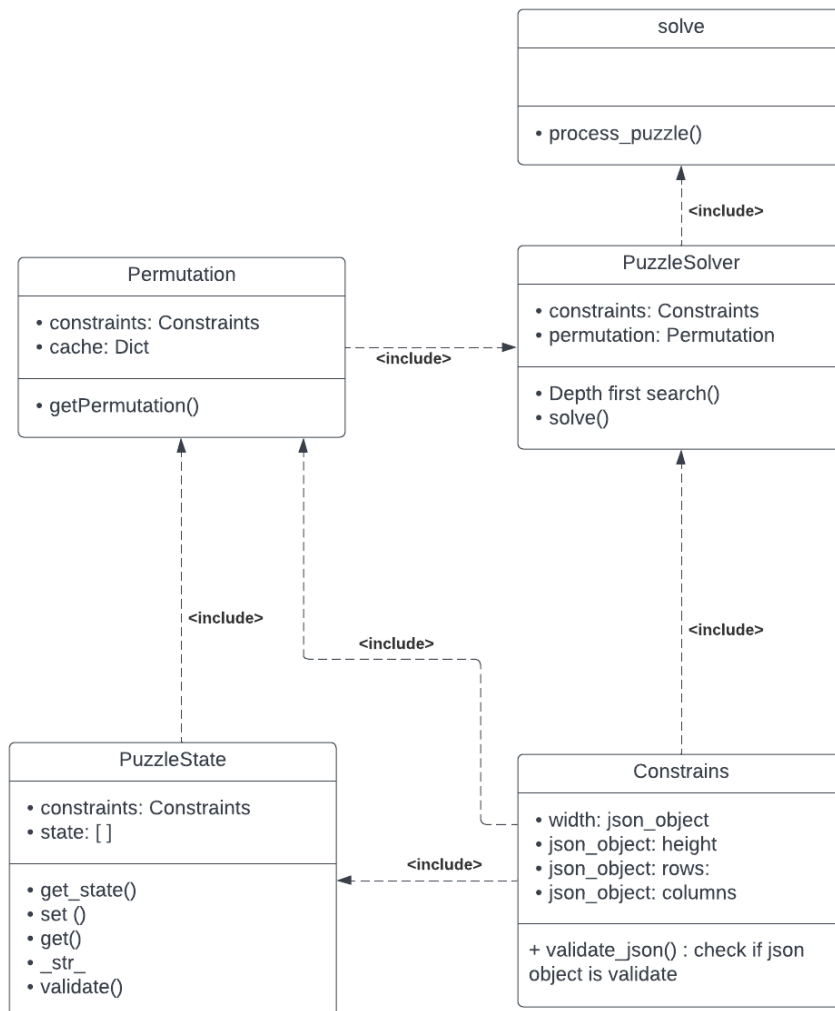
Nonogram là trò chơi mà bạn phải sử dụng tư duy logic để giải một hình ảnh được tạo nên bởi các ô được đánh dấu dựa theo số lượng các ô đã cho biết trước trên mỗi hàng hoặc cột. Lời giải cần phải thoả mãn những điều kiện sau:

- Mỗi hàng hoặc cột chỉ được có các ô đánh dấu bằng với các con số cho trước
- Các ô đánh dấu phải nối tiếp nhau thành một cụm các ô và các cụm ô phải cách nhau bởi ít nhất một ô trống



Hình 4: Ví dụ Nonogram

2.3 Phân tích trò chơi



Hình 5: Class diagram của Nonogram

Solver của Nonogram gồm các chức năng sau:

- **CONstraints**: Trong phần đầu vào của Nonogram, nhóm dùng cách đọc từ file json nên việc dữ liệu vào phải được kiểm tra từ bước đầu tiên.
- **PuzzleState**: xử lý, kiểm tra các state của Nonogram sau mỗi lần đi qua. Kiểm tra trạng thái hiện tại có thể tìm được hướng giải quyết nào không. Ngoài ra, tại đây cũng xử lý việc in ra trạng thái hiện tại của Nonogram.
- **Permutation**: đây là class quan trọng nhất vì nó thực hiện chủ yếu phần di chuyển giữa các ô. Các hoạt động của nó là kiểm tra từng hàng

- Solver: Tổng hợp các bước giải bài toán để in ra kết quả cuối cùng

Để giải bài toán, chương trình sẽ thực hiện theo các bước

- Thực hiện giải quyết một hàng rồi đánh dấu các ô khả thi theo các số của cột
- Sau đó kiểm tra xem hàng hiện tại đã thỏa mãn các điều kiện không, nếu có thì chuyển qua hàng tiếp theo, nếu không thì tìm cách giải khác bằng cách dịch qua phải một đơn vị của ô đánh dấu bên phải cùng
- Nếu hàng tiếp theo không thỏa mãn các điều kiện và không thể giải được nữa thì lùi lại một hàng và tìm cách giải khác.
- Nếu số hàng đã giải bằng với số hàng của đề thì dừng lại, cho ra kết quả tìm được.

Dưới đây là ví dụ đơn giản và ngắn về cách giải:

	1		
	1	5	1
1			
2			
1			
1			
3			

(a) Bước 1

	1		
	1	5	1
1			
2			
1			
1			
3			

(b) Bước 2

Hình 6: Điền ô khả thi tìm được

Đây là bước đầu tiên, chương trình sẽ tìm ở hàng đầu tiên và đánh dấu ô khả thi nhất có thể thỏa mãn điều kiện của hàng và cột.

	1		
	1	5	1
1			
2			
1			
1			
3			

(a) Bước 3

	1		
	1	5	1
1			
2			
1			
1			
3			

(b) Bước 4

Hình 7: Tìm lời giải thay thế khi không thỏa điều kiện

Tiếp theo, chương trình sẽ chuyển qua hàng tiếp theo và điền lời giải khả thi theo hàng trước. Nhưng khi kiểm tra với điều kiện của cột thì không thỏa mãn, chương trình phải dịch qua bên phải một ô.

	1		
	1	5	1
1			
2			
1			
1			
3			

(a) Bước 5

	1		
	1	5	1
1			
2			
1			
1			
3			

(b) Bước 6

Hình 8: Điền ô khả thi tìm được

Khi chuyển qua hàng tiếp theo, ô đầu tiên có thể thỏa mãn các điều kiện nên sẽ chuyển qua hàng thứ 4. Tại đây, sẽ tiếp tục một lần dịch sang phải để có thể thỏa mãn điều kiện.

2.4 Kết quả chạy chương trình

Sau khi thử nghiệm với Testcase đơn giản như hình sau:

		1	1		1	1
		3	1	1	1	1
		1	1	5	1	3
	1					
	1					
1	1					
	1					
1	3					
1	1					
	5					
1	1					
3	1					

Hình 9: Testcase TickNazi

Chương trình sẽ in ra các bước và kết quả sau:

```
PROBLEMS  OUTPUT  TERMINAL

Step: 90
┌───┐
|o| | | |
| | |o| |
|o| | |o|
| | | | |
└───┘

Step: 91
┌───┐
|o| | | |
| | |o| |
|o| | |o|
|o| |o|o|
└───┘

Step: 92
┌───┐
|o| | | |
| | |o| |
|o| | |o|
| | | | |
└───┘
```

(a) Các bước thực hiện

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS

|o| |o|o|o|
|o| |o| |
|o|o|o|o|o|
|o| | |o|
┌───┐

Step: 100
┌───┐
| | | | | |
| | |o| |
| | |o| |o|
|o| | |
|o| |o|o|o|
|o| |o| |
|o|o|o|o|o|
| | |o| |o|
└───┘

Step: 101
┌───┐
| | | | | |
| | |o| |
| | |o| |o|
|o| | |
|o| |o|o|o|
|o| |o| |
|o|o|o|o|o|
| | |o| |o|
└───┘

Step: 102
┌───┐
| | | | | |
|o| | | |
| | |o| |
| | |o| |o|
|o| |o|o|o|
|o| |o| |
|o|o|o|o|o|
| | |o| |o|
|o|o|o| |o|
└───┘

Done after 102 steps
```

(b) Kết quả

Hình 10: in ra các bước thực hiện và kết quả

Tuy nhiên vì phải in ra nhiều bước nên chương trình sẽ chạy lâu hơn và hao tốn hơn nên sau khi bỏ đi in ra các bước, chỉ in ra kết quả ta được

```
phuloi@PhuLoi:/mnt/f/OneDrive - hcmut.edu.vn/Usi
ck\&Nazi.json
Processing puzzle 1 of 1
Done after 102 steps
.-----.
| | | | | |
|o| | | |
| | | |o|
| | |o|o|
| |o| | |
|o| |o|o|o|
|o| |o| | |
|o|o|o|o|o|
| | |o| |o|
|o|o|o| |o|
.-----.
102
0.0066678 secs 9.7560000 MB
```

Hình 11: Kết quả của TickNazi

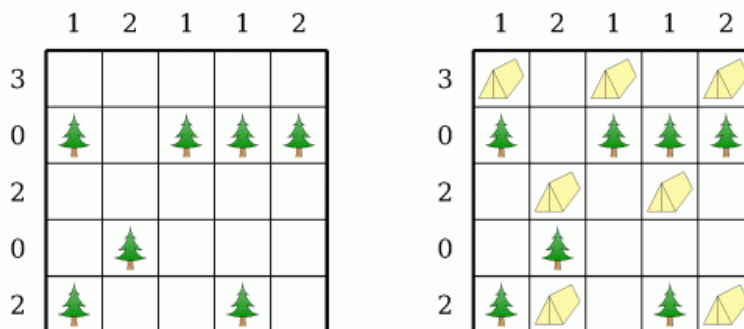
Thời gian chạy là 0.0066678 giây và hao tốn 9.75 Kb



$\backslash include\{AstarNono\}$

3 Tents

3.1 Giới thiệu về trò chơi



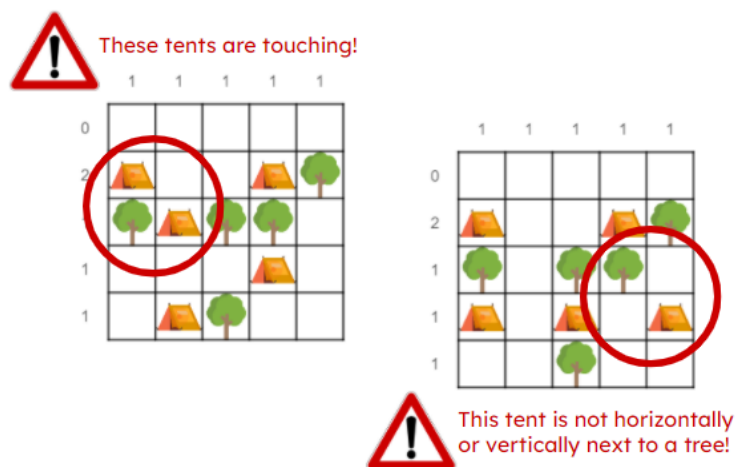
Hình 12: một màn đơn giản của Tents

Tents là một logic puzzle với luật chơi đơn giản nhưng lại có thể có độ khó khá cao tùy vào việc tạo màn và kích thước của câu đố mô phỏng việc những người đi cắm trại thường dựng lều nhưng nơi có bóng mát (cây).

3.2 Luật chơi

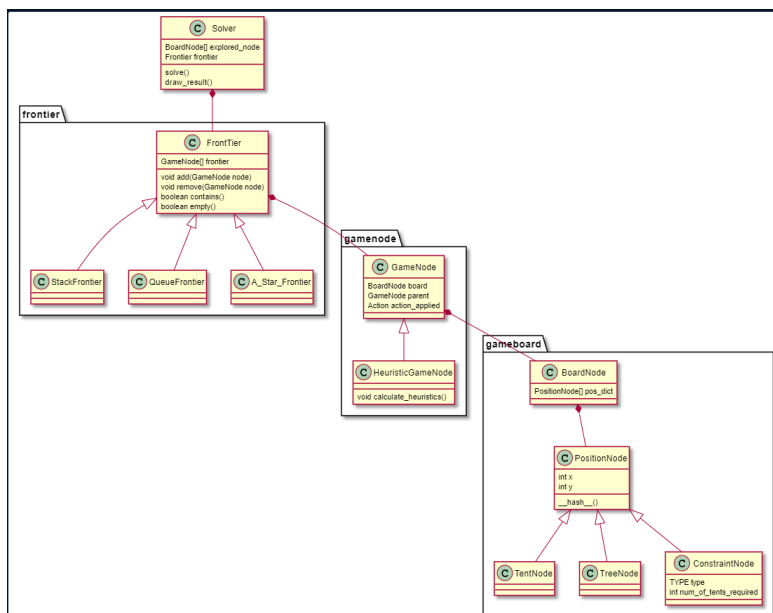
Để giải một màn Tents ta cần chọn vị trí của các lều đảm bảo các điều kiện sau:

- Binding: Mỗi lều bind vào một cây theo phương ngang hoặc phương dọc theo quan hệ 1-1
- Isolation: Không có 2 lều nào được đặt kề nhau (kể cả theo đường chéo)
- Quantity: Số lượng các lều trên mỗi hàng phải thỏa các số (Clues) ở đầu hàng và đầu cột tương ứng của câu đố



Hình 13: Luật chơi của Tents

3.3 Phân tích trò chơi - Solver đã hiện thực



Hình 14: Class diagram của solver cho game Tents

Solver của nhóm hiện thực giải thuật tìm kiếm đã mô hình lại một game tents như sau:

- Frontier: Lưu giữ các game node là candidates và quyết định candidate nào sẽ được lấy ra để thực hiện bước đi tiếp theo. Mỗi loại frontier sẽ quyết định chiến lược search được sử dụng (Stack - DFS, Queue - BFS, A Star với datastructure bên dưới là một heap - Heuristics A*)

- GameNode: Giữ 1 board với các trạng thái hiện tại của board nhằm hiện thực backtrack và tĩa nhánh khi revisit lại một node đã duyệt qua.
- PositionNode: Mô tả một vị trí trên board
- TentNode: Là lớp con của position node mà giải thuật đã đặt một lều ở đây.
- ConstraintNode: mô tả số lượng lều mà hàng / cột này cần phải đạt. Với mỗi lều được đặt lên, 2 constraint node ở hàng và cột tương ứng sẽ giảm đi 1 đơn vị
- BoardNode: giữ một dictionary các PositionNode, dictionary các TreeNode, dictionary các TentsNode, dictionary các ConstraintNode. Chọn cấu trúc dữ liệu là dictionary để thực hiện fast-access ($O(1)$ vì key của dictionary là một value đã được qua hàm băm thay vì $O(N)$ nếu chọn cấu trúc dữ liệu List)

Tuy không được đặt vào luật chính thức của trò chơi, nhưng từ các rule ở phần 3.2 ta có thể suy ra được một số quy luật khác để loại bớt các nhánh phải tìm cho giải thuật tìm kiếm như sau:

- 0-valued constraints: Các hàng 0 sẽ được loại bỏ khỏi các vị
- Các ô chéo so với một cây: do mỗi lều chỉ được bind vào cây theo phương ngang hoặc dọc, ta có thể loại bỏ các ô nằm chéo với một cây nhưng không phải là ô liền kề theo phương ngang hoặc dọc với một cây khác.
- Quantity: Dựa vào quantity constraint ta sẽ giảm đi một đơn vị cho 2 constrain (1 cho hàng và 1 cho cột) ứng với index của vị trí đã đặt lều.
- Definitely Binded Tree: các cây đứng một mình, khi bind một lều vào ta có thể loại bỏ tất cả vị trí xung quanh cây (theo phương ngang, dọc và chéo)



(a) Chéo của một cây nhưng không phải là thẳng với cây nào khác



(b) Một cây riêng lẻ và chắc chắn đã được bind - Definitely Binded Tree

Hình 15: Một số optimize để tĩa bớt nhánh của cây tìm kiếm (X là ô có thể lược bỏ)

3.3.1 Heuristic của A* cho bài Tents

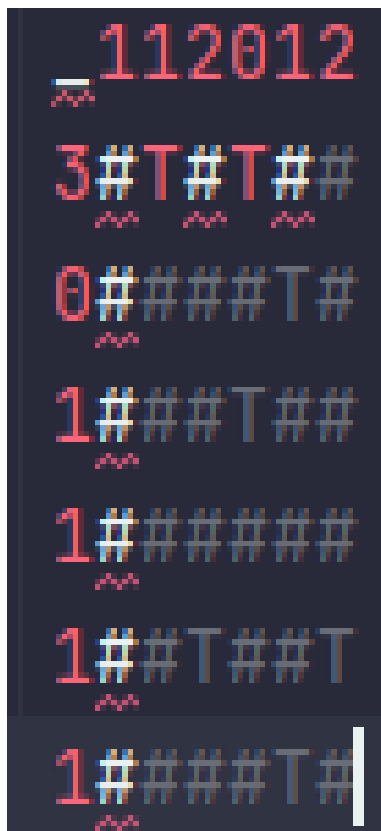
Để hiện thực giải thuật A*, Solver của nhóm đã tính hàm heuristic để chọn ra node tiếp theo như sau:

$$H(n) = \sum_{t=1}^N PossibleTentLocation$$

Đúng vậy, giải thuật A* sẽ ưu tiên các nốt mà số lượng vị trí còn lại có thể đặt lên sau khi bị loại bớt bởi các constraint và luật được suy ra ở phần 3.3 là nhỏ nhất.

3.3.2 Demo giải một bài tents 6x6

Với input như hình ?? (cách biểu diễn vui lòng tham khảo README.MD của github mã nguồn):



Hình 16: Biểu diễn dạng text của một game tents

Khi chạy solver và bật cờ để track từng put ta có thể theo dõi trên command line quá trình giải:

```
Tents initialized:
CONSTRAINTS:
C1:1 C2:1 C3:2 C4:0 C5:1 C6:2 R1:3 R2:0 R3:1 R4:1 R5:1 R6:1
BOARD:
#T#T##
####T#
###T##
#####
##T##T
####T#
Trying to solve the puzzle

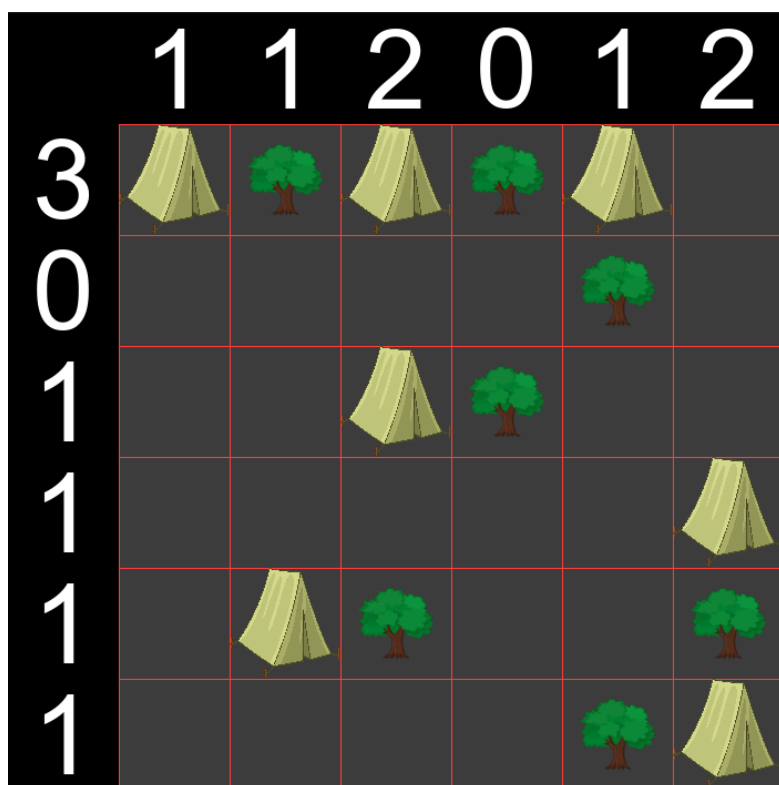
Node 1
#T#T##
####T#
###T##
#####
##T##T
####T#

Node 2
#T#T##
####T#
###T##
#####
##T##T
####TL

Node 3
LTTL#
####T#
##LT##
####L
#LT##T
####TL
Num of state explored: 3
```

Hình 17: Quá trình giải với giải thuật A^*

Kết quả cuối cùng được visualize lại với thư viện pillow:



Hình 18: Kết quả cuối cùng

4 Đánh giá chung

4.1 Kết quả chạy được

4.1.1 Thông số máy

Máy tính chạy chương trình: Dell Inspiron 15 7000, Win 10, 4 cores, 8 Logical Processors, 16GB RAM.

Terminal chạy: WSL Ubuntu 20.01 LTS

4.1.2 Excel:

name	time_used	mem_used	steps
input/10x10/nono-1.js	0.0282224	10.176	1894
input/10x10/nono-10.js	0.0305883	10.18	2725
input/10x10/nono-2.js	0.0035355	10.18	105
input/10x10/nono-3.js	0.0018108	10.188	
input/10x10/nono-4.js	0.0280977	10.216	1825
input/10x10/nono-5.js	0.0125008	10.216	331
input/10x10/nono-6.js	0.0147469	10.216	717
input/10x10/nono-7.js	0.0189802	10.216	561
input/10x10/nono-9.js	0.0365754	10.216	1899
ut/Random/BKHorizon	0.0154457	10.216	154
input/10x10/nono-8.js	0.3798623	10.216	22189
put/Random/Dolphin.js	6.204844	10.644	129594
input/Random/Simple.js	0.0012156	10.644	7
input/Random/Smile.js	0.001206	10.644	19
ut/Random/Tick&Nazi	0.0025197	10.644	102

Hình 19: Kết quả các Test case cơ bản của Nonogram DFS

(a) Kết quả Memory là Kb

name	time_used	mem_used
input/8x8/4.	0.128320118	15304
input/8x8/5.	0.134988905	37292
input/8x8/1.	0.341269371	40376
input/8x8/3.	2.578877792	44936
input/8x8/2.	2.066751798	56804
t/6x6/puzzl	0.048628754	57320
t/6x6/puzzl	0.03528704	29320
t/6x6/puzzl	0.075104013	58320
t/6x6/puzzl	0.036554121	57320
t/6x6/puzzl	0.029290652	22320
it/6x6_hard	0.683306603	93320
it/6x6_hard	0.045356274	87220
it/6x6_hard	0.04100174	49320
it/6x6_hard	0.044403685	37320
it/6x6_hard	0.03793195	57320

Hình 21: Kết quả các Test case cơ bản của Nonogram A*

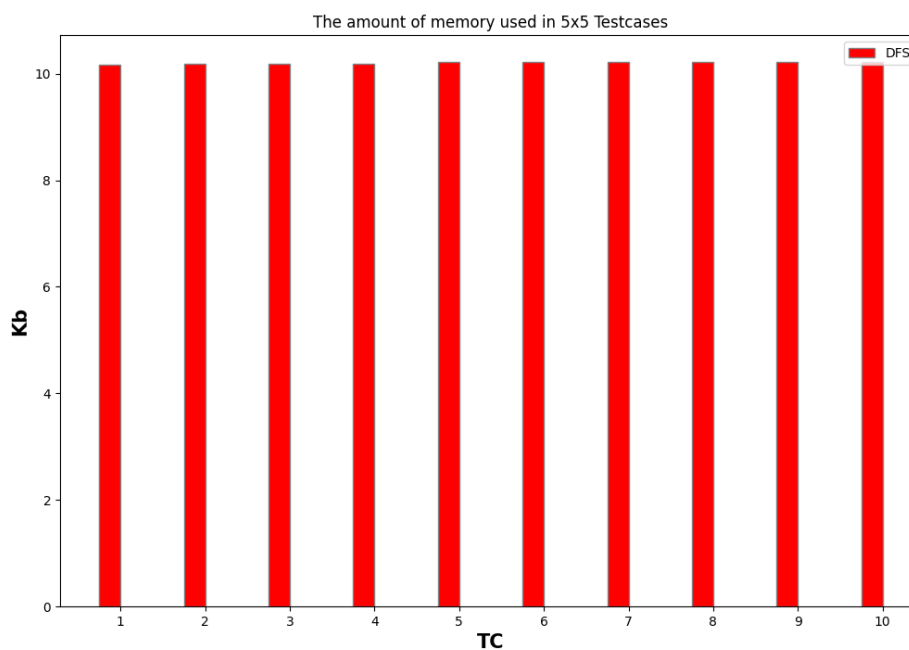
(a) Kết quả Memory là Byte

name	time_used	mem_used
input/8x8/4.	1.868857314	25412
input/8x8/5.	0.24132325	48076
input/8x8/1.	0.168837954	48584
input/8x8/3.	3.919173593	38584
input/8x8/2.	31.72144513	173332
t/6x6/puzzl	0.018236271	189040
t/6x6/puzzl	0.015502767	201040
t/6x6/puzzl	0.032227152	169040
t/6x6/puzzl	0.020055729	203000
t/6x6/puzzl	0.045842185	189040
it/6x6_hard	0.681165825	129040
it/6x6_hard	0.161581854	189040
it/6x6_hard	0.060733569	229040
it/6x6_hard	0.089349216	319040
it/6x6 hard	0.043530469	179040

Hình 23: Kết quả các Test case cơ bản của Nonogram DFS

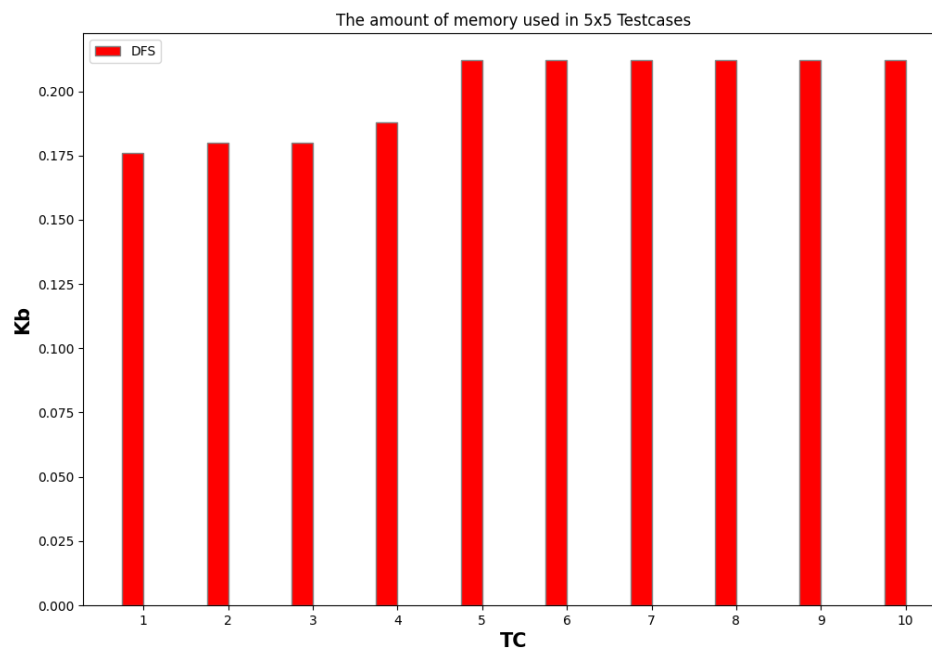
(a) Kết quả Memory là Byte

4.2 Biểu đồ

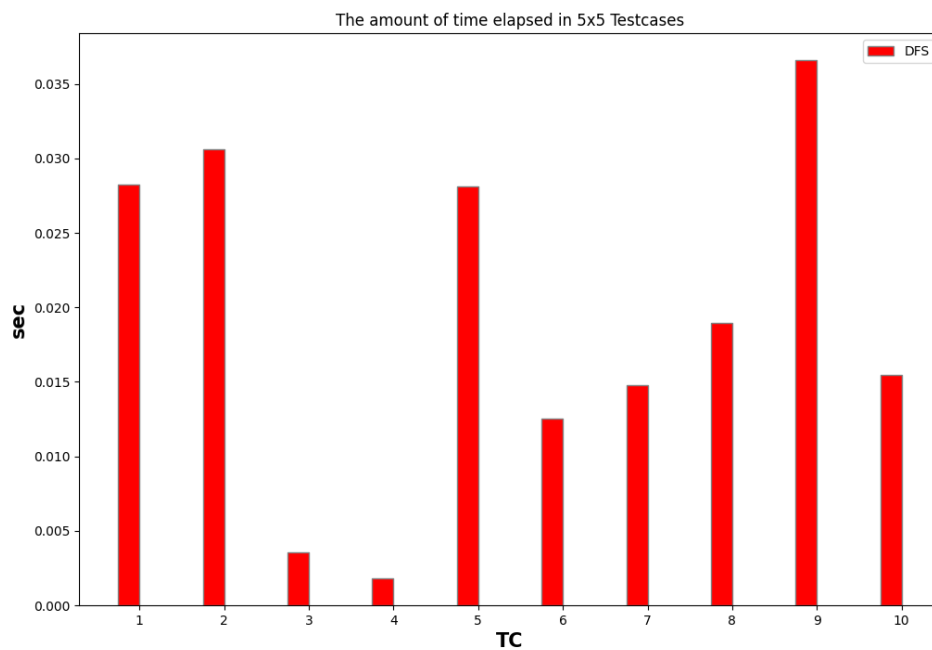


Hình 25: *Memory used in Nonogram DFS*

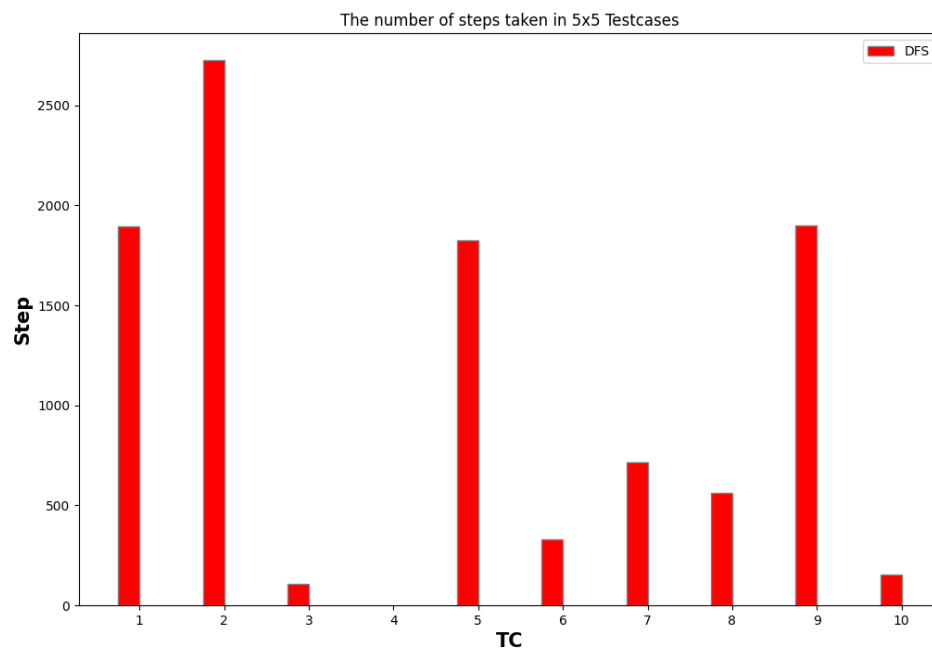
Vì các số là gần bằng nhau nên các cột không thể chỉ rõ độ chênh lệch. Vì thế nhóm lập bảng mới với tất cả đều **giảm đi 10** và được bảng như sau:



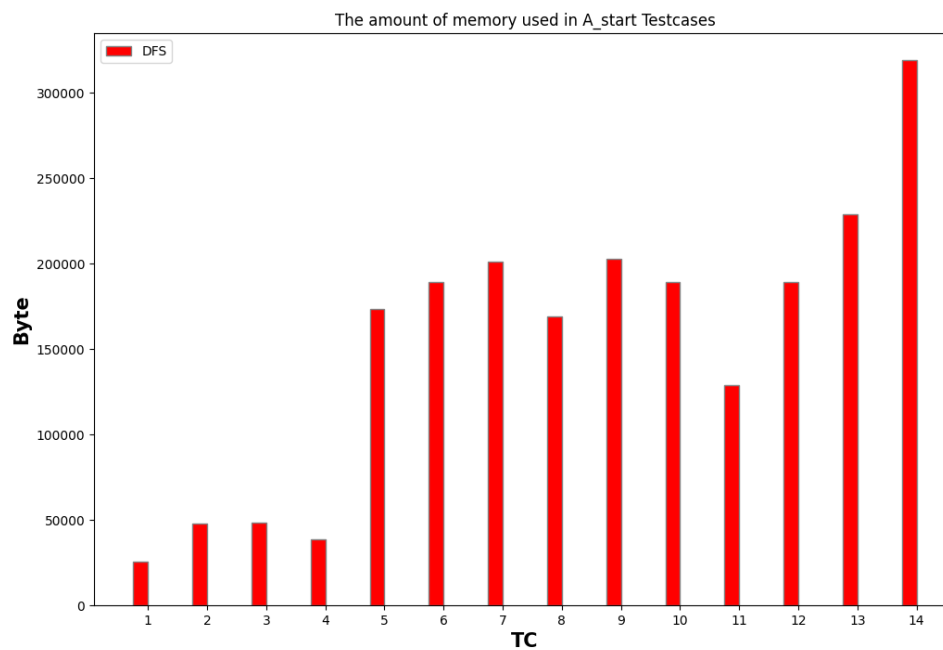
Hình 26: *Memory used in Nonogram DFS*



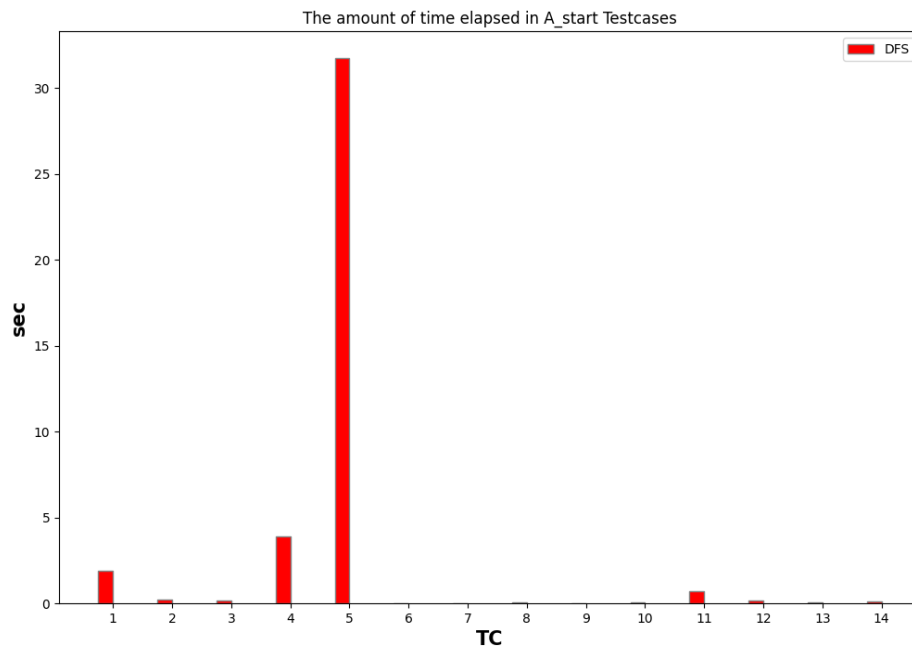
Hình 27: *Time used in Nonogram DFS*



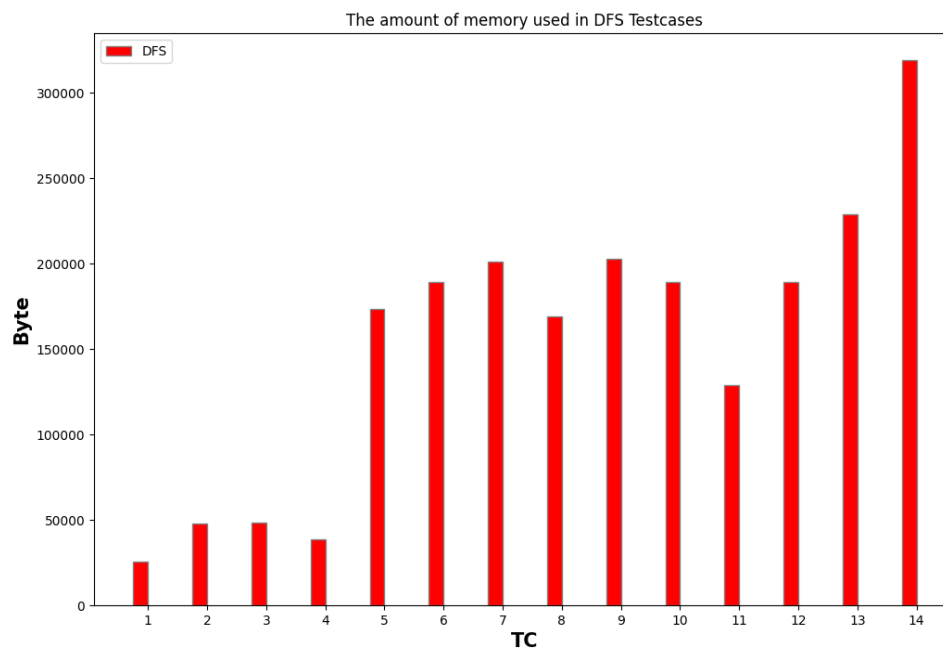
Hình 28: *Steps used in Nonogram DFS*



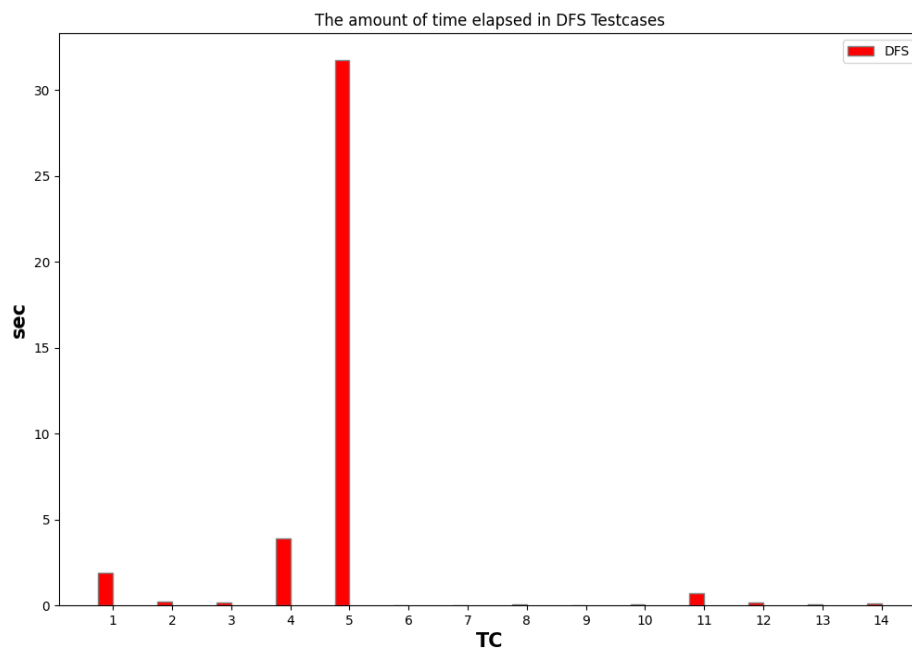
Hình 29: *Memory used in Tents A**



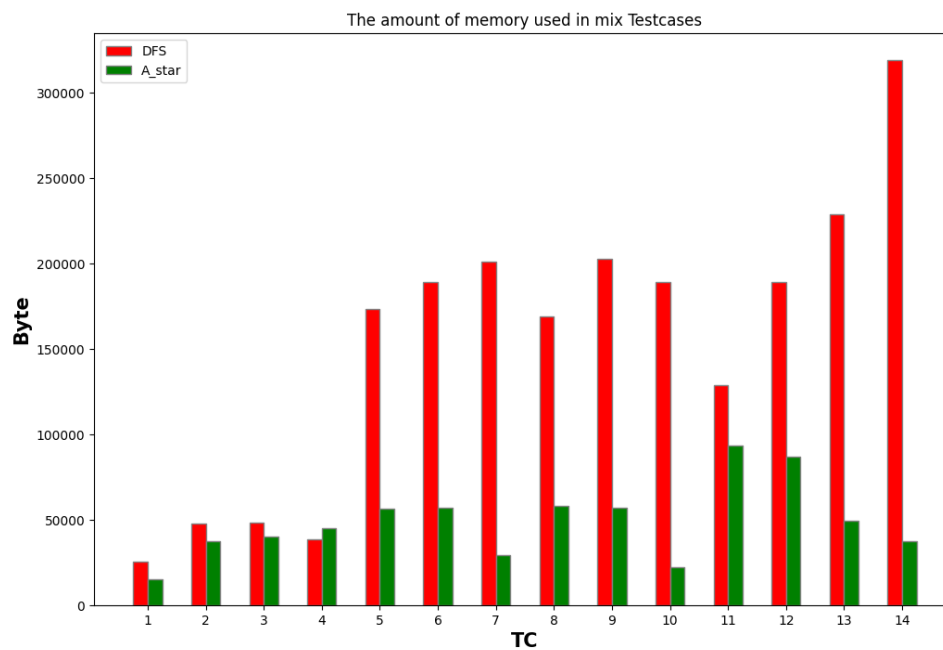
Hình 30: *Time used in Tents A**



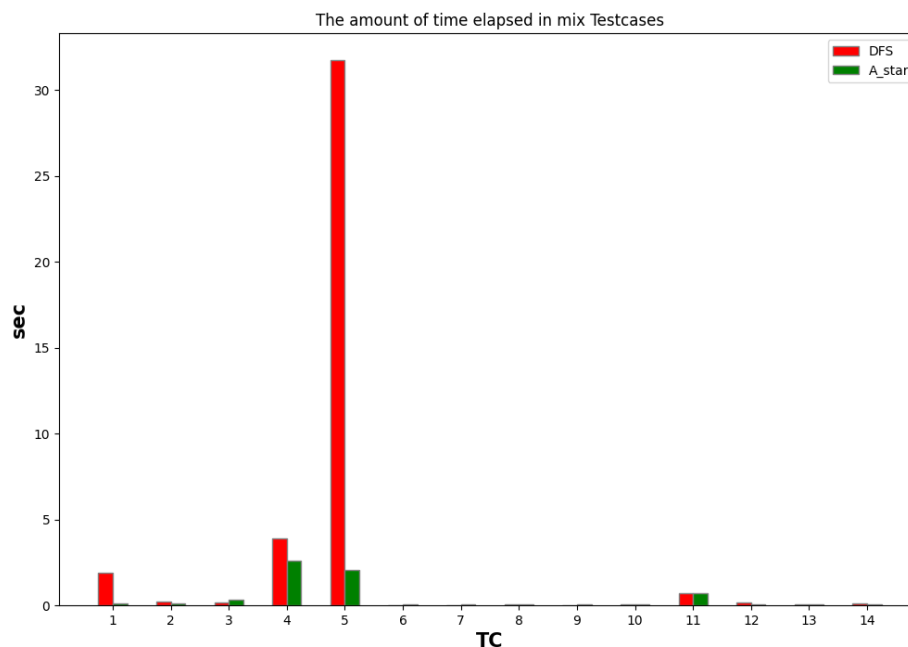
Hình 31: *Memory used in Tents DFS*



Hình 32: Time used in Tents DFS



Hình 33: *Memory used in Tents*



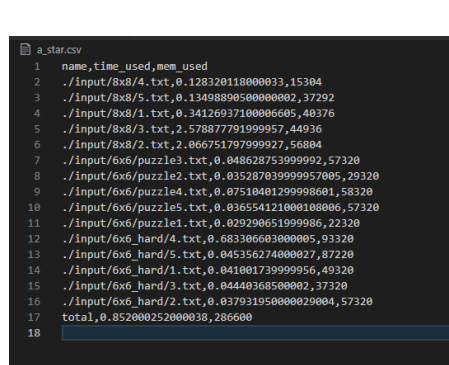
Hình 34: Time used in Tests

4.3 Phân tích kết quả

4.3.1 Lý thuyết:

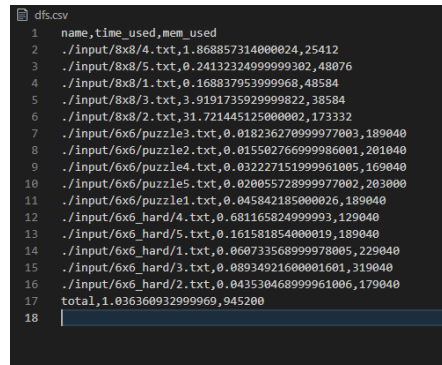
- DFS là thuật toán vét cạn theo chiều dọc. Điều này làm cho kết quả tìm kiếm được là đường đi dài để đến được đích nên có thể tiêu thụ nhiều thời gian thực hiện hơn.
- Giải thuật A* tùy thuộc vào hàm heuristic nên số lượng Node sinh ra sẽ ít hơn nhưng kết quả tìm kiếm được là khá tốt không phải luôn luôn là tốt nhất

4.3.2 Thực nghiệm:



name	time_used	mem_used
./input/8x8/4.txt	0.128320118000033	15304
./input/8x8/5.txt	0.1248880050000002	37202
./input/8x8/1.txt	0.3412693710000605	40376
./input/8x8/3.txt	2.578877791999957	44936
./input/8x8/2.txt	2.066751797999927	56884
./input/6x6/puzzle3.txt	0.048628753999992	57320
./input/6x6/puzzle2.txt	0.035287039999957005	29320
./input/6x6/puzzle4.txt	0.07510401299998601	58320
./input/6x6/puzzle5.txt	0.036554121000180006	57320
./input/6x6/puzzle1.txt	0.029200651999986	22320
./input/6x6_hard/4.txt	0.683306603000005	93320
./input/6x6_hard/5.txt	0.045356274000027	87220
./input/6x6_hard/1.txt	0.041001739999956	49320
./input/6x6_hard/3.txt	0.04440368500002	37320
./input/6x6_hard/2.txt	0.037931950000029004	57320
total	0.852000252000038	286600

(a) Kết quả file csv A*



name	time_used	mem_used
./input/8x8/4.txt	1.868857314000024	25412
./input/8x8/5.txt	0.2413232499999302	48076
./input/8x8/1.txt	0.168837953999968	48584
./input/8x8/3.txt	3.9191735929999822	38584
./input/8x8/2.txt	31.721445125000002	173332
./input/6x6/puzzle3.txt	0.018236270999977003	189040
./input/6x6/puzzle2.txt	0.015502766999986001	201040
./input/6x6/puzzle4.txt	0.032227151999961005	169040
./input/6x6/puzzle5.txt	0.020055728999977002	203000
./input/6x6/puzzle1.txt	0.045842185000026	189040
./input/6x6_hard/4.txt	0.681165824999993	129040
./input/6x6_hard/5.txt	0.161581854000019	189040
./input/6x6_hard/1.txt	0.060733568999978005	229040
./input/6x6_hard/3.txt	0.08934921600001601	319040
./input/6x6_hard/2.txt	0.043530468999961006	179040
total	1.036360932999969	945200

(b) Kết quả file csv DFS

Hình 35: So sánh vài kết quả của Tents

Từ việc kết quả đo đạc có thể thấy, giải thuật A* chạy tốn ít bộ nhớ hơn cho một bài toán cùng độ khó so với giải thuật DFS vì việc chạy giải thuật DFS phải chạy theo chiều dọc nên phải sinh ra nhiều bước hơn để có thể đi đến đích.

Tuy nhiên ở những bài toán dễ và không gian tìm nhỏ thì DFS lại thực hiện tốt hơn về phần thời gian chạy vì A* phải liên tục tính toán để lựa chọn bước tiếp theo tại mỗi node đi nên sẽ tốn thời gian hơn. Nhưng khi đến những bài toán với không gian tìm kiếm lớn hơn và phức tạp hơn thì A* lại thực hiện tốt hơn vì số lượng node sinh ra ở DFS lớn hơn nên thời gian để di chuyển theo từng nhánh và tìm node đáp án là rất lớn

Nhìn chung: Giải thuật DFS tốt hơn A* khi không gian tìm kiếm nhỏ và độ phức tạp bài toán là nhỏ. Giải thuật A* sẽ tốt hơn DFS ở những bài toán có không gian tìm kiếm lớn và độ phức tạp cao hơn.

Không gian tìm kiếm	Độ phức tạp bài toán	Giải thuật nên dùng
Nhỏ	Dễ	DFS
Nhỏ	Khó	A*
Lớn	Dễ	A*
Lớn	Khó	A*

Bảng 1: Lựa chọn dựa theo tiêu chí thời gian

5 Mã nguồn và demo

- Repos chứa toàn bộ mã nguồn hiện thực của nhóm:
<https://github.com/PhuLoi-1911545/solve-PuzzleGame-by-AI>
- Video demo:
https://drive.google.com/drive/folders/1gJYnTB_I8RD2IrvZCB8TNmHA1buh944J?usp=sharing

6 Tài liệu tham khảo

- Tents game, luật chơi và input:
<https://www.puzzle-tents.com/>
- Nonogram cách giải quyết DFS:
 - <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/Makalah/Makalah-Stima-2019-001.pdf>
 - https://escholarship.org/content/qt13t021xk/qt13t021xk_noSplash_b88e92ee4e5876462355685c6e452891.pdf?t=op2j5h
- Nonogram Heuristic:
https://www.researchgate.net/publication/221157413_Solving_Japanese_Puzzles_with_Heuristics