

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

○○○○☆○○○○



**MÔN NHẬP MÔN TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI:**

**BÁO CÁO BÀI TẬP LỚN SỐ 1**

Áp dụng giải thuật tìm kiếm Blind search và Heuristic vào game  
Sudoku và game Kakurasu

**LỚP DT01 - HK 223**

**NGÀY NỘP 25/07/2023**

**Giảng viên hướng dẫn: VƯƠNG BÁ THỊNH**

Sinh viên thực hiện	Mã số sinh viên	Điểm số
Trương Văn Nhân	2220016	
Hoàng Việt Thông	1750060	
Nguyễn Hoàng Vinh	2233140	
Phạm Duy Thắng	2233076	

Thành phố Hồ Chí Minh – 2023

1.	GIỚI THIỆU SƠ LƯỢC:	3
1.1.	Lựa chọn đề tài game:	3
1.2.	Phân chia công việc thực hiện:	3
2.	GAME SUDOKU	4
2.1.	Giới thiệu game và luật chơi	4
2.2.	Biểu diễn game Sudoku:	5
2.2.1.	Không gian trạng thái:	5
2.2.2.	Trạng thái khởi đầu:	5
2.2.3.	Trạng thái mục tiêu:	5
2.2.4.	Luật chuyển trạng thái	6
2.3.	Xây dựng các hàm cơ bản:	6
2.4.	Xây dựng hàm giải bài toán bằng giải thuật Blind Search	7
2.5.	Xây dựng hàm giải bài toán bằng giải thuật Heuristic Search	9
2.6.	Kết quả chạy, tiêu tốn tài nguyên, so sánh và giải thích	11
2.6.1.	Phương pháp đo:	11
2.6.2.	Kết quả chạy thực nghiệm:	11
2.6.3.	Nhận xét và giải thích:	14
3.	GAME KAKURASU	16
3.1.	Giới thiệu game và cách chơi	16
3.2.	Biểu diễn game Kakurasu:	17
3.2.1.	Không gian trạng thái:	17
3.2.2.	Trạng thái khởi đầu:	17
3.2.3.	Trạng thái mục tiêu:	17
3.2.4.	Luật chuyển trạng thái	18
3.3.	Xây dựng các hàm cơ bản:	18
3.4.	Xây dựng hàm giải bài toán bằng giải thuật Blind Search	19
3.5.	Xây dựng hàm giải bài toán bằng giải thuật Heuristic Search	21
3.6.	Kết quả chạy, so sánh và giải thích	25
3.6.1.	Phương pháp đo:	25
3.6.2.	Kết quả chạy thực nghiệm:	25
3.6.3.	Nhận xét và giải thích:	29
4.	TÀI LIỆU THAM KHẢO	31

## 1. GIỚI THIỆU SƠ LƯỢC:

### 1.1. Lựa chọn đề tài game:

Dựa theo yêu cầu của BTL số 1 của môn học, sinh viên đã lựa chọn 02 game Logic Puzzles là game Kakurasu ( <https://www.puzzle-kakurasu.com/>) và game Sudoku (<https://www.puzzle-sudoku.com/>) để hiện thực các giải thuật tìm kiếm đã được tìm hiểu trong môn học.

Ngôn ngữ lập trình được sử dụng trong bài tập lớn này là python 3.9, chạy trên Pycharm với các thiết lập mặc định. Cấu hình máy tính dùng để chạy các bài testcase và thống kê time, memory: Laptop CPU AMD 5600u, 24Gb RAM 3200Mhz, SSD M2 Nvme.

Các mục tiếp theo đây sẽ trình bày về quá trình tìm hiểu, hiện thực và kết quả đạt được của sinh viên đối với 2 bài toán game logic trên, sử dụng giải thuật tìm kiếm Blind search và Heuristic.

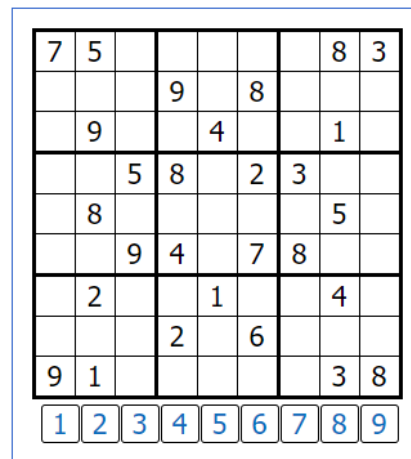
### 1.2. Phân chia công việc thực hiện:

Stt	Công việc	Sinh viên thực hiện chính
1	Giải bài toán Sudoku bằng Blind Search, viết báo cáo tương ứng.	Trương Văn Nhân - 2220016
2	Giải bài toán Sudoku bằng Heuristic Search, viết báo cáo tương ứng.	Trương Văn Nhân - 2220016
3	Giải bài toán Kakurasu bằng Blind Search, viết báo cáo tương ứng.	Trương Văn Nhân - 2220016
4	Giải bài toán Kakurasu bằng Heuristic Search, viết báo cáo tương ứng.	Trương Văn Nhân - 2220016
5	Video báo cáo Hỗ trợ báo cáo Slide trình bày	Team

Do 3 thành viên còn lại của nhóm chưa học qua môn cấu trúc dữ liệu và giải thuật, do vậy phần lớn nội dung bài tập lớn được thực hiện chính bởi 1 thành viên, các thành viên còn lại có trách nhiệm thảo luận, tìm kiếm testcase, hỗ trợ kiểm thử, thống kê và hỗ trợ công tác báo cáo.

## 2. GAME SUDOKU

### 2.1. Giới thiệu game và luật chơi



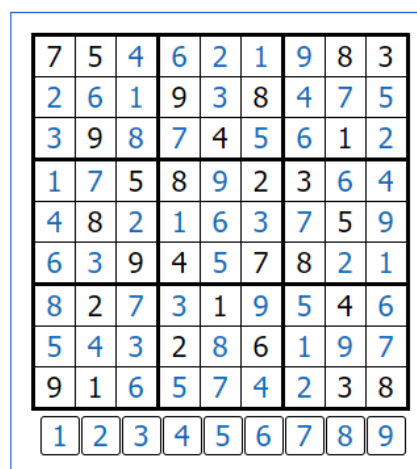
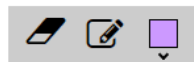
3x3 Basic Sudoku Puzzle ID: 57,452,606

Hình 2.1: Ví dụ một ván game Sudoku

- 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 là điền các con số có một chữ số vào vị trí còn trống trên lưới.
- Khi đã hoàn thành lời giải, lưới Sudoku sẽ trở thành một ma trận hình vuông, 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 3x3 nào của bảng trò chơi 9x9.

Congratulations! You have solved the puzzle in 03:49.97

Submit your score to the Hall of Fame



3x3 Basic Sudoku Puzzle ID: 57,452,606

Hình 2.2: Ví dụ một ván game Sudoku đã được giải thành công

- Link game: <https://www.puzzle-sudoku.com/>

## 2.2. Biểu diễn game Sudoku:

### 2.2.1. Không gian trạng thái:

- Ta sử dụng ma trận 9x9 (ma trận A) để biểu diễn trạng thái của game, giá trị tại vị trí  $A[m][n]$  (với  $0 \leq m, n \leq 8$ ) là giá trị tại ô được giao bởi hàng  $(m+1)$  với cột  $(n+1)$  trong game.
- Giá trị hợp lệ tại vị trí  $A[m][n]$  là  $k$  với  $k$  là số nguyên không âm  $0 \leq k \leq 9$ , trong đó khi  $k=0$  sẽ tương ứng với việc giá trị tại ô game hàng  $(m+1)$  giao cột  $(n+1)$  đang trống, chưa được điền giá trị.
- Trong python, ta có thể sử dụng cấu trúc dữ liệu là một list có 9 phần tử, mỗi phần tử của list là một list có 9 phần tử số nguyên để biểu diễn ma trận 9x9 này. Ngoài ra, do thư viện numpy hỗ trợ rất tốt các thao tác xử lý trên ma trận, nên sinh viên đã convert cấu trúc trên sang ma trận của numpy.

### 2.2.2. Trạng thái khởi đầu:

- Trạng thái khởi đầu game là input của game, là một không gian trạng thái với một số ô đã được điền trước giá trị hợp lệ, và các ô còn lại để trống chờ được người chơi điền (giải game).
- Trong chương trình mô phỏng và giải game, sinh viên đưa input đầu vào là cấu trúc list trong list, sau đó convert sang cấu trúc ma trận của numpy bằng lệnh `numpy.asarray(matrix)`
- Với  $A_0$  là không gian trạng thái biểu diễn trạng thái khởi đầu của game, tập các cặp giá trị  $(m,n)$  sao cho  $A_0[m,n] \neq 0$  được gọi là tập **O**, chứa vị trí các ô đã được cố định giá trị, không được thay đổi

```
game = [ [7, 5, 0, 0, 0, 0, 0, 8, 3],
          [0, 0, 0, 9, 0, 8, 0, 0, 0],
          [0, 9, 0, 0, 4, 0, 0, 1, 0],
          [0, 0, 5, 8, 0, 2, 3, 0, 0],
          [0, 8, 0, 0, 0, 0, 0, 0, 5],
          [0, 0, 9, 4, 0, 7, 8, 0, 0],
          [0, 2, 0, 0, 1, 0, 0, 4, 0],
          [0, 0, 0, 2, 0, 6, 0, 0, 0],
          [9, 1, 0, 0, 0, 0, 0, 3, 8]]

game = np.asarray(game)
print(game)
```

Hình 2.3: Minh họa biểu diễn trạng thái khởi đầu bằng cấu trúc list

```
[[7 5 0 0 0 0 0 8 3]
 [0 0 0 9 0 8 0 0 0]
 [0 9 0 0 4 0 0 1 0]
 [0 0 5 8 0 2 3 0 0]
 [0 8 0 0 0 0 0 0 5]
 [0 0 9 4 0 7 8 0 0]
 [0 2 0 0 1 0 0 4 0]
 [0 0 0 2 0 6 0 0 0]
 [9 1 0 0 0 0 0 3 8]]
```

Hình 2.4: Minh họa biểu diễn trạng thái khởi đầu dưới dạng ma trận trong numpy

### 2.2.3. Trạng thái mục tiêu:

- Trạng thái mục tiêu là trạng thái mà tất cả các giá trị của các ô trong ma trận 9x9 được điền (tức khác 0) và thỏa mãn điều kiện: **mỗi số nguyên duy nhất sẽ không**

xuất hiện hai lần trong cùng một hàng, cột hoặc bất kỳ một trong chín khối vuông  $3 \times 3$  nào của bảng trò chơi  $9 \times 9$  (\*).

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

Hình 2.5: Minh họa một trạng thái mục tiêu thỏa mãn

```
[[7 5 4 6 2 1 9 8 3]
 [2 6 1 9 3 8 4 7 5]
 [3 9 8 7 4 5 6 1 2]
 [1 7 5 8 9 2 3 6 4]
 [4 8 2 1 6 3 7 5 9]
 [6 3 9 4 5 7 8 2 1]
 [8 2 7 3 1 9 5 4 6]
 [5 4 3 2 8 6 1 9 7]
 [9 1 6 5 7 4 2 3 8]]
```

Hình 2.6: Minh họa biểu diễn một trạng thái mục tiêu dưới dạng ma trận trong numpy

#### 2.2.4. Luật chuyển trạng thái

- Các bước đi hợp lệ:

+ Chọn giá trị từ 1 đến 9 cho một vị trí ô còn trống (vị trí hiện đang có giá trị bằng 0 trong ma trận):  $A[m,n] = k$  ( $1 \leq k \leq 9$ ) với cặp giá trị  $(m,n)$  không nằm trong tập **O** và  $0 \leq m,n \leq 8$ .

+ Thay đổi giá trị của ô đã chọn giá trị  $A[m,n] = k$  thành  $A[m,n] = h$  với  $0 \leq k,h \leq 9$  và  $h \neq k$ ,  $(m,n)$  không nằm trong tập **O**.

#### 2.3. Xây dựng các hàm cơ bản:

- Xây dựng hàm *check\_solved(board)* để kiểm tra trạng thái hiện tại của **board** có phải là trạng thái mục tiêu hay chưa, nếu đúng **board** đã là trạng thái mục tiêu, hàm trả về **True**, ngược lại hàm trả về **False**. Cơ chế hiện thực của hàm là đưa danh sách các số của mỗi cột, hàng về dạng tập hợp *set()*; khi đó nếu số phần tử của tập hợp nhỏ hơn 9 tức là đang có ít nhất 2 số trùng nhau => **False**; hoặc nếu số phần tử của tập hợp bằng 9 mà có xuất hiện số 0 trong đó => có 1 ô chưa được điền giá trị => **False**; nếu tất cả các hàng, các cột không vi phạm => trả về **True**.

```
def check_solved(board):
    row = [set(i) for i in board]
    col = []
    for i in range(9):
        temp=[ board[j,i] for j in range(9)]
        col.append(set(temp))
    for i in range(9):
        if len(row[i]) !=9 or len (col[i]) !=9: return False
        if (0 in row[i]) or (0 in col[i]): return False
    return True
```

- Xây dựng hàm *find\_empty\_cell(board)* để tìm ô trống gần nhất xuất hiện trong ma trận **board** (vị trí hiện đang có giá trị bằng 0), thứ tự tìm kiếm từ trái sang phải, từ trên xuống dưới. Ngay khi tìm được vị trí đầu tiên có  $A[i,j] = 0$  hàm sẽ trả về cặp giá trị  $(i,j)$  là index của ô đó trong ma trận A, nếu tất cả các ô đã được điền thì hàm *find\_empty\_cell(board)* sẽ trả về giá trị *None*.  
 ⇒ Hàm *find\_empty\_cell* đảm bảo ta chỉ tác động đến các ô trống, không tác động đến các ô nằm trong tập **O** và các ô đã được điền.

```
def find_empty_cell(data):
    for i in range(len(data)):
        for j in range(len(data[0])):
            if data[i,j]==0:
                return (i,j)
    return None
```

- Xây dựng hàm *check\_valid\_num(board, row,col, number)* để kiểm tra việc điền giá trị number vào vị trí **board[row,col]** có là hợp lý hay không (có thể dẫn đến trạng thái mục tiêu, tức không vi phạm các điều kiện của trạng thái mục tiêu (\*)). Hàm *check\_valid\_num(board, row,col, number)* sẽ trả về giá trị là **True** nếu số **number** không vi phạm các điều kiện dẫn đến trạng thái mục tiêu, và sẽ trả về giá trị **False** nếu có ít nhất một điều kiện vi phạm (số **number** đã xuất hiện ở cột/hàng/box 3x3 mà ô **board[row,col]** thuộc về).  
 Các chỉ số **(row,col)** được lấy từ hàm *find\_empty\_cell(board)* để đảm bảo đó là một bước đi hợp lệ, không tác động đến các ô nằm trong tập **O**.

```
def check_valid_num(board, row,col, number): #kiểm tra nếu đặt number
tại vị trí (row,col) có hợp lệ không
    row_lst = board[row,:].flatten().tolist()
    col_lst = board[:,col].flatten().tolist()
    id_box_r = row-row%3
    id_box_c = col-col%3
    box =
board[id_box_r:id_box_r+3,id_box_c:id_box_c+3].flatten().tolist()
    if row_lst.count(number)==0 and col_lst.count(number)==0 and
box.count(number)==0: #kiểm tra row, col, box chưa xuất hiện number
        return True
    else:
        return False
```

## 2.4. Xây dựng hàm giải bài toán bằng giải thuật Blind Search

- Sử dụng thuật toán vét cạn theo chiều sâu DFS, thông qua việc xây dựng hàm đệ quy *solve\_sudoku(board)*.
- Đầu tiên, hàm sẽ kiểm tra trạng thái hiện tại của board có phải là trạng thái mục tiêu, nếu phải hàm sẽ trả về ma trận board và kết thúc, nếu chưa phải là trạng thái mục tiêu, hàm sẽ tìm ô trống gần nhất xuất hiện trong bảng board thông qua hàm *find\_empty\_cell(board)*:

- + Nếu hàm `find_empty_cell(board)` trả về `None` => hiện tại không còn ô trống nào cả => trạng thái hiện tại không là hướng đi có lời giải => hàm `solve_sudoku` trả về `None`
- + Nếu hiện tại còn ô trống, `(row,col)` của ô trống ( ô X) sẽ được ghi nhận, hàm `solve_sudoku` sẽ thử lần lượt các giá trị của ô này từ 1 đến 9 thông qua hàm `check_valid_num(board, row,col, number)` xem các giá trị nào là giá trị có thể dẫn đến trạng thái mục tiêu.
- + Với mỗi giá trị `number` có khả năng dẫn đến hàm mục tiêu, ta sẽ lập tức gán giá trị này cho ô X và gọi hàm đệ quy với bảng mới vừa được cập nhật giá trị cho ô X.
  - Nếu giá trị trả về của hàm đệ quy là `None`, nghĩa là việc chọn giá trị `number` trên cho ô X không dẫn tới trạng thái mục tiêu => gán lại giá trị 0 cho ô X và chuyển sang kiểm tra số `number` tiếp theo cho ô X.
  - Nếu giá trị trả về của hàm đệ quy là bảng `board`, nghĩa là bảng này là trạng thái mục tiêu (ngay từ đầu ta đã xác định hàm `solve_sudoku(board)` chỉ trả về 2 giá trị: `None` nếu không có lời giải, và bảng `board` nếu `board` là lời giải).
- Để phục vụ công tác hiển thị các step, ta bổ sung thêm các lệnh `print(board)` và lời nhắc cần thiết mỗi nhánh thay đổi giá trị `board`.

```
def solve_sudoku(board):
    empty_cell = find_empty_cell(board)
    if empty_cell is None:  # Tất cả các ô đã được fill chính xác
        if check_solved(board):
            print("SOLUTION FOUND: ")
            return board
        else:
            print("None")
            return None
    row, col = empty_cell
    for num in range(1, 10): #thử giá trị từ 1 đến 9
        if check_valid_num(board, row, col, num):
            board[row][col] = num
            print("Next step is: ")
            print(board)
            result = solve_sudoku(board)
            if result is not None:
                return result
            # Nếu không thể giải tiếp, quay lại giá trị 0 cho ô
            board[row][col] = 0
            print("IT'S NOT WAY TO SOLUTION- SET BOARD BACK: ")
            print(board)
```

- Như vậy hàm `solve_sudoku(board)` ở trên về bản chất chỉ đơn thuần là thử tất cả các giá trị có thể cho lần lượt từng ô trống, ưu tiên đi theo chiều sâu, khi hướng



chiều sâu đó không dẫn đến trạng thái mục tiêu, thì hàm mới rẽ sang nhánh kế bên theo đúng cơ chế DFS.

## 2.5. Xây dựng hàm giải bài toán bằng giải thuật Heuristic Search

- **Nhân xét:** các giá trị có thể nhận được của một ô trống phụ thuộc vào giá trị của các ô đã được điền nằm trong cùng hàng, cột và box 3x3 của ô trống đó, số lượng giá trị riêng biệt biết được của các ô được điền này càng lớn thì số giá trị có thể điền cho ô trống càng được thu hẹp => xác xuất tìm đúng giá trị cho ô trống càng cao.
- Từ cơ sở trên, sinh viên sử dụng thuật toán tìm kiếm leo đồi, thông qua việc xây dựng thêm hàm lượng giá *heuristic\_select\_cell(data)* để lựa chọn ô điền giá trị tiếp theo.
- Để phục vụ xây dựng hàm lượng giá, sinh viên xây dựng thêm một số hàm hỗ trợ gồm: hàm *list\_empty\_cell(board)* và hàm *list\_number\_for\_search(board, row, col)*.
  - + Hàm *list\_empty\_cell(board)* trả về một danh sách index các ô còn trống tại trạng thái board hiện tại, các cặp giá trị (i,j) thỏa *board[i,j]=0* sẽ được append vào list. Nếu không có ô nào trống hết ở trạng thái board hiện tại, hàm sẽ trả về **danh sách rỗng**.

```
def list_empty_cell(board):
    lst = []
    for i in range(len(board)):
        for j in range(len(board[0])):
            if board[i, j] == 0:
                lst.append((i, j))
    return lst
```

- + Hàm *list\_number\_for\_search(board, row, col)* trả về danh sách các **number** có thể điền vào vị trí ô *board[row,col]*, ta sẽ kiểm tra các giá trị **number** đi từ 1 đến 9, xem các giá trị nào chưa xuất hiện trong hàng, cột, box 3x3 mà ô trống đó thuộc về, các giá trị nào thỏa mãn sẽ được append vào list.

```
def list_number_for_search(board, row, col):
    lst = []
    row_lst = board[row, :].flatten().tolist()
    col_lst = board[:, col].flatten().tolist()
    row_block = row - row % 3
    col_block = col - col % 3
    block_lst = board[row_block:row_block + 3, col_block:col_block + 3].flatten().tolist()
    for number in range(1, 10):
        if (number not in row_lst) and (number not in col_lst) and (number not in block_lst):
            lst.append(number)
    return lst
```

- Hàm lượng giá *heuristic\_select\_cell(board)* tạo một danh sách các ô còn trống hiện tại thông qua hàm *list\_empty\_cell(board)*; với mỗi phần tử trong *list\_empty\_cell* này, ta xác định danh sách các số có thể điền tại vị trí đó thông qua hàm *list\_number\_for\_search(board, row, col)*
  - + Trong trường hợp *list\_empty\_cell* trả về danh sách rỗng, tức không còn ô trống nào, thì hàm *heuristic\_select\_cell* sẽ trả về None.
  - + Hàm *heuristic\_select\_cell(board)* so sánh và trả về index vị trí ô có số lượng số có thể điền là nhỏ nhất (kiểm tra chiều dài của mỗi phần tử trong *list\_empty\_cell*), trong trường hợp có nhiều vị trí có cùng số lượng nhỏ nhất, hàm sẽ chọn và trả về index ô xuất hiện gần nhất trong bảng.
  - + Nếu đã hết ô trống thông qua *heuristic\_select\_cell*, hàm lượng giá trả về None

```
def heuristic_select_cell(board):
    lst_empty_cell = list_empty_cell(board)
    if len(lst_empty_cell) == 1:
        return lst_empty_cell[0]
    if len(lst_empty_cell) == 0:
        return None
    counting_list = [0 for _ in range(len(lst_empty_cell))]
    for index in range(len(lst_empty_cell)):
        row = lst_empty_cell[index][0]
        col = lst_empty_cell[index][1]
        counting_list[index] = len(list_number_for_search(board, row,
col))
    min_valid = min(counting_list)
    id = counting_list.index(min_valid)
    return lst_empty_cell[id]
```

- Hàm đệ quy *solve\_sudoku(board)* hoàn toàn tương tự như hàm *solve\_sudoku* ở mục 2.5, tuy nhiên vị trí *empty\_cell* thay vì dùng hàm *find\_empty\_cell(board)* thì ta sử dụng hàm *heuristic\_select\_cell(board)* để lựa được vị trí ô nhanh dẫn đến trạng thái mục tiêu nhất.

```
def solve_sudoku(board):
    empty_cell = heuristic_select_cell(board)
    if empty_cell is None: # Tất cả các ô đã được fill chính xác
        if check_solved(board):
            print("SOLUTION FOUND: ")
            return board
        else:
            print("None")
            return None
    row, col = empty_cell
    for num in range(1, 10): #thử giá trị từ 1 đến 9
        if check_valid_num(board, row, col, num):
            board[row][col] = num
            print("Next step is: ")
            print(board)
```

```

    result = solve_sudoku(board)
    if result is not None:
        return result
    # Nếu không thể giải tiếp, quay lại giá trị 0 cho ô
    board[row][col] = 0
    print("IT'S NOT WAY TO SOLUTION- SET BOARD BACK: ")
    print(board)

```

- Hàm `solve_sudoku(board)` sử dụng hàm lượng giá `heuristic_select_cell(board)` để tìm được ô có xác suất chọn được giá trị đúng cao nhất, giúp quá trình tìm kiếm đến trạng thái mục tiêu nhanh hơn, đúng với mô tả của thuật toán tìm kiếm leo đồi.

## 2.6. Kết quả chạy, tiêu tốn tài nguyên, so sánh và giải thích

### 2.6.1. Phương pháp đo:

- Để đo thời gian chạy của chương trình, sinh viên sử dụng thư viện `time` của python, đo thời điểm bắt đầu và kết thúc của chương trình, từ đó tính hiệu của 2 mốc ta sẽ có được thời gian chương trình tiêu tốn.
- Để đo mức memory tối đa mà chương trình sử dụng, ta sử dụng thư viện `tracemalloc`, cập giá trị trả về sẽ gồm bộ nhớ tiêu tốn hiện tại và bộ nhớ tiêu tốn đỉnh của chương trình, ta chỉ quan tâm đến giá trị thứ 2.
- Đoạn code phục vụ kiểm tra tài nguyên và thời gian chạy sẽ có dạng:










```

start = time.time()
tracemalloc.clear_traces()
tracemalloc.start()
solve_sudoku(np.asarray(game))
print(f"Bộ nhớ tối đa sử dụng: {tracemalloc.get_traced_memory()[1]} bytes")
tracemalloc.stop()
print(f"Thời gian chạy là {time.time()-start} giây ")

```

### 2.6.2. Kết quả chạy thực nghiệm:

- Các testcase được lấy từ trang <https://www.puzzle-sudoku.com/>
- Có tất cả 6 mức độ cho game sudoku size 3x3; mỗi mức độ ta lấy 1 game, tổng là 6 game, mỗi game chạy lần lượt cả 2 thuật toán giải trên và ghi nhận kết quả.

STT	Tên	Mức độ	Input web	Solution Pass																																																																																																																																																																																				
1	game1	Basic	<div><table><tr><td></td><td>1</td><td></td><td></td><td>7</td><td>6</td><td>3</td><td></td><td></td></tr><tr><td>7</td><td></td><td>2</td><td></td><td>6</td><td></td><td></td><td>1</td><td>9</td></tr><tr><td></td><td>9</td><td></td><td></td><td></td><td></td><td></td><td>7</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>3</td></tr><tr><td></td><td>5</td><td></td><td></td><td>9</td><td>4</td><td></td><td></td><td>2</td></tr><tr><td>3</td><td></td><td></td><td>1</td><td></td><td></td><td></td><td>5</td><td></td></tr><tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td></tr><tr><td>9</td><td>3</td><td>7</td><td></td><td></td><td>6</td><td>5</td><td></td><td></td></tr><tr><td></td><td>8</td><td></td><td>1</td><td>7</td><td></td><td></td><td></td><td></td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table><div>3x3 Basic Sudoku Puzzle ID: 46,508,598</div></div>		1			7	6	3			7		2		6			1	9		9						7										3		5			9	4			2	3			1				5		2								8	9	3	7			6	5				8		1	7					1	2	3	4	5	6	7	8	9	<div><div>Congratulations! You have solved the puzzle in 05:20.35</div><div>Submit your score to the Hall of Fame</div><div></div><div><table><tr><td>8</td><td>1</td><td>5</td><td>9</td><td>2</td><td>7</td><td>6</td><td>3</td><td>4</td></tr><tr><td>7</td><td>4</td><td>2</td><td>5</td><td>6</td><td>3</td><td>8</td><td>1</td><td>9</td></tr><tr><td>6</td><td>9</td><td>3</td><td>8</td><td>4</td><td>1</td><td>2</td><td>7</td><td>5</td></tr><tr><td>4</td><td>2</td><td>6</td><td>7</td><td>5</td><td>8</td><td>1</td><td>9</td><td>3</td></tr><tr><td>1</td><td>5</td><td>8</td><td>3</td><td>9</td><td>4</td><td>7</td><td>6</td><td>2</td></tr><tr><td>3</td><td>7</td><td>9</td><td>6</td><td>1</td><td>2</td><td>4</td><td>5</td><td>8</td></tr><tr><td>2</td><td>6</td><td>1</td><td>4</td><td>3</td><td>5</td><td>9</td><td>8</td><td>7</td></tr><tr><td>9</td><td>3</td><td>7</td><td>2</td><td>8</td><td>6</td><td>5</td><td>4</td><td>1</td></tr><tr><td>5</td><td>8</td><td>4</td><td>1</td><td>7</td><td>9</td><td>3</td><td>2</td><td>6</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table><div>3x3 Basic Sudoku Puzzle ID: 46,508,598</div></div></div>	8	1	5	9	2	7	6	3	4	7	4	2	5	6	3	8	1	9	6	9	3	8	4	1	2	7	5	4	2	6	7	5	8	1	9	3	1	5	8	3	9	4	7	6	2	3	7	9	6	1	2	4	5	8	2	6	1	4	3	5	9	8	7	9	3	7	2	8	6	5	4	1	5	8	4	1	7	9	3	2	6	1	2	3	4	5	6	7	8	9
	1			7	6	3																																																																																																																																																																																		
7		2		6			1	9																																																																																																																																																																																
	9						7																																																																																																																																																																																	
								3																																																																																																																																																																																
	5			9	4			2																																																																																																																																																																																
3			1				5																																																																																																																																																																																	
2								8																																																																																																																																																																																
9	3	7			6	5																																																																																																																																																																																		
	8		1	7																																																																																																																																																																																				
1	2	3	4	5	6	7	8	9																																																																																																																																																																																
8	1	5	9	2	7	6	3	4																																																																																																																																																																																
7	4	2	5	6	3	8	1	9																																																																																																																																																																																
6	9	3	8	4	1	2	7	5																																																																																																																																																																																
4	2	6	7	5	8	1	9	3																																																																																																																																																																																
1	5	8	3	9	4	7	6	2																																																																																																																																																																																
3	7	9	6	1	2	4	5	8																																																																																																																																																																																
2	6	1	4	3	5	9	8	7																																																																																																																																																																																
9	3	7	2	8	6	5	4	1																																																																																																																																																																																
5	8	4	1	7	9	3	2	6																																																																																																																																																																																
1	2	3	4	5	6	7	8	9																																																																																																																																																																																
2	game2	Easy	<div><table><tr><td></td><td>7</td><td></td><td></td><td>1</td><td></td><td></td><td>2</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>2</td><td>6</td><td></td><td></td><td></td><td></td><td></td><td>1</td><td>4</td></tr><tr><td></td><td></td><td>2</td><td>1</td><td></td><td>4</td><td>6</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>8</td><td></td><td>3</td><td>2</td><td>9</td><td></td><td>5</td><td></td></tr><tr><td>3</td><td></td><td></td><td>4</td><td></td><td>2</td><td></td><td></td><td>5</td></tr><tr><td>6</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>8</td></tr><tr><td>7</td><td>5</td><td></td><td>8</td><td></td><td>3</td><td></td><td>4</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table><div>3x3 Easy Sudoku Puzzle ID: 35,786,095</div></div>		7			1			2											2	6						1	4			2	1		4	6													8		3	2	9		5		3			4		2			5	6								8	7	5		8		3		4	1	1	2	3	4	5	6	7	8	9	<div><div>Congratulations! You have solved the puzzle in 04:31.4€</div><div>Submit your score to the Hall of Fame</div><div></div><div><table><tr><td>4</td><td>7</td><td>3</td><td>5</td><td>1</td><td>8</td><td>9</td><td>2</td><td>6</td></tr><tr><td>8</td><td>9</td><td>1</td><td>2</td><td>4</td><td>6</td><td>5</td><td>7</td><td>3</td></tr><tr><td>2</td><td>6</td><td>5</td><td>9</td><td>3</td><td>7</td><td>8</td><td>1</td><td>4</td></tr><tr><td>5</td><td>3</td><td>2</td><td>1</td><td>7</td><td>4</td><td>6</td><td>8</td><td>9</td></tr><tr><td>9</td><td>4</td><td>7</td><td>6</td><td>8</td><td>5</td><td>1</td><td>3</td><td>2</td></tr><tr><td>1</td><td>8</td><td>6</td><td>3</td><td>2</td><td>9</td><td>4</td><td>5</td><td>7</td></tr><tr><td>3</td><td>1</td><td>8</td><td>4</td><td>9</td><td>2</td><td>7</td><td>6</td><td>5</td></tr><tr><td>6</td><td>2</td><td>4</td><td>7</td><td>5</td><td>1</td><td>3</td><td>9</td><td>8</td></tr><tr><td>7</td><td>5</td><td>9</td><td>8</td><td>6</td><td>3</td><td>2</td><td>4</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table><div>3x3 Easy Sudoku Puzzle ID: 35,786,095</div></div></div>	4	7	3	5	1	8	9	2	6	8	9	1	2	4	6	5	7	3	2	6	5	9	3	7	8	1	4	5	3	2	1	7	4	6	8	9	9	4	7	6	8	5	1	3	2	1	8	6	3	2	9	4	5	7	3	1	8	4	9	2	7	6	5	6	2	4	7	5	1	3	9	8	7	5	9	8	6	3	2	4	1	1	2	3	4	5	6	7	8	9
	7			1			2																																																																																																																																																																																	
2	6						1	4																																																																																																																																																																																
		2	1		4	6																																																																																																																																																																																		
	8		3	2	9		5																																																																																																																																																																																	
3			4		2			5																																																																																																																																																																																
6								8																																																																																																																																																																																
7	5		8		3		4	1																																																																																																																																																																																
1	2	3	4	5	6	7	8	9																																																																																																																																																																																
4	7	3	5	1	8	9	2	6																																																																																																																																																																																
8	9	1	2	4	6	5	7	3																																																																																																																																																																																
2	6	5	9	3	7	8	1	4																																																																																																																																																																																
5	3	2	1	7	4	6	8	9																																																																																																																																																																																
9	4	7	6	8	5	1	3	2																																																																																																																																																																																
1	8	6	3	2	9	4	5	7																																																																																																																																																																																
3	1	8	4	9	2	7	6	5																																																																																																																																																																																
6	2	4	7	5	1	3	9	8																																																																																																																																																																																
7	5	9	8	6	3	2	4	1																																																																																																																																																																																
1	2	3	4	5	6	7	8	9																																																																																																																																																																																
3	game3	Inter	<div><table><tr><td></td><td>5</td><td>6</td><td>1</td><td>7</td><td></td><td></td><td></td><td></td></tr><tr><td>9</td><td></td><td>1</td><td></td><td></td><td></td><td>7</td><td></td><td></td></tr><tr><td>7</td><td>4</td><td></td><td></td><td>9</td><td>6</td><td></td><td>8</td><td></td></tr><tr><td>5</td><td></td><td></td><td></td><td></td><td>1</td><td>3</td><td></td><td></td></tr><tr><td>6</td><td></td><td>8</td><td></td><td></td><td></td><td>5</td><td></td><td>9</td></tr><tr><td></td><td></td><td>9</td><td>5</td><td></td><td></td><td></td><td></td><td>1</td></tr><tr><td></td><td>9</td><td></td><td>6</td><td>5</td><td></td><td></td><td>1</td><td>7</td></tr><tr><td></td><td></td><td>5</td><td></td><td></td><td></td><td>2</td><td></td><td>6</td></tr><tr><td></td><td></td><td></td><td></td><td>1</td><td>2</td><td>9</td><td>5</td><td></td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table><div>3x3 Intermediate Sudoku Puzzle ID: 67,135,579</div></div>		5	6	1	7					9		1				7			7	4			9	6		8		5					1	3			6		8				5		9			9	5					1		9		6	5			1	7			5				2		6					1	2	9	5		1	2	3	4	5	6	7	8	9	<div><div>Congratulations! You have solved the puzzle in 04:07.40</div><div>Submit your score to the Hall of Fame</div><div></div><div><table><tr><td>3</td><td>5</td><td>6</td><td>1</td><td>7</td><td>8</td><td>4</td><td>9</td><td>2</td></tr><tr><td>9</td><td>8</td><td>1</td><td>4</td><td>2</td><td>5</td><td>7</td><td>6</td><td>3</td></tr><tr><td>7</td><td>4</td><td>2</td><td>3</td><td>9</td><td>6</td><td>1</td><td>8</td><td>5</td></tr><tr><td>5</td><td>2</td><td>7</td><td>9</td><td>6</td><td>1</td><td>3</td><td>4</td><td>8</td></tr><tr><td>6</td><td>1</td><td>8</td><td>2</td><td>3</td><td>4</td><td>5</td><td>7</td><td>9</td></tr><tr><td>4</td><td>3</td><td>9</td><td>5</td><td>8</td><td>7</td><td>6</td><td>2</td><td>1</td></tr><tr><td>2</td><td>9</td><td>4</td><td>6</td><td>5</td><td>3</td><td>8</td><td>1</td><td>7</td></tr><tr><td>1</td><td>7</td><td>5</td><td>8</td><td>4</td><td>9</td><td>2</td><td>3</td><td>6</td></tr><tr><td>8</td><td>6</td><td>3</td><td>7</td><td>1</td><td>2</td><td>9</td><td>5</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table><div>3x3 Intermediate Sudoku Puzzle ID: 67,135,579</div></div></div>	3	5	6	1	7	8	4	9	2	9	8	1	4	2	5	7	6	3	7	4	2	3	9	6	1	8	5	5	2	7	9	6	1	3	4	8	6	1	8	2	3	4	5	7	9	4	3	9	5	8	7	6	2	1	2	9	4	6	5	3	8	1	7	1	7	5	8	4	9	2	3	6	8	6	3	7	1	2	9	5	4	1	2	3	4	5	6	7	8	9
	5	6	1	7																																																																																																																																																																																				
9		1				7																																																																																																																																																																																		
7	4			9	6		8																																																																																																																																																																																	
5					1	3																																																																																																																																																																																		
6		8				5		9																																																																																																																																																																																
		9	5					1																																																																																																																																																																																
	9		6	5			1	7																																																																																																																																																																																
		5				2		6																																																																																																																																																																																
				1	2	9	5																																																																																																																																																																																	
1	2	3	4	5	6	7	8	9																																																																																																																																																																																
3	5	6	1	7	8	4	9	2																																																																																																																																																																																
9	8	1	4	2	5	7	6	3																																																																																																																																																																																
7	4	2	3	9	6	1	8	5																																																																																																																																																																																
5	2	7	9	6	1	3	4	8																																																																																																																																																																																
6	1	8	2	3	4	5	7	9																																																																																																																																																																																
4	3	9	5	8	7	6	2	1																																																																																																																																																																																
2	9	4	6	5	3	8	1	7																																																																																																																																																																																
1	7	5	8	4	9	2	3	6																																																																																																																																																																																
8	6	3	7	1	2	9	5	4																																																																																																																																																																																
1	2	3	4	5	6	7	8	9																																																																																																																																																																																

4	game4	Advance	<div><div></div><div><table><tr><td>3</td><td></td><td>7</td><td>4</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>8</td><td></td><td>1</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>4</td><td></td><td></td><td></td><td>9</td><td></td><td>6</td><td></td><td></td></tr><tr><td>5</td><td>1</td><td></td><td>6</td><td></td><td>8</td><td></td><td></td><td></td></tr><tr><td></td><td></td><td>4</td><td></td><td></td><td></td><td>8</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td>2</td><td></td><td>7</td><td></td><td>1</td><td>6</td></tr><tr><td></td><td></td><td>3</td><td></td><td>6</td><td></td><td></td><td></td><td>5</td></tr><tr><td></td><td></td><td></td><td></td><td>4</td><td></td><td>2</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>2</td><td>1</td><td></td><td>8</td><td></td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></div><div>3x3 Advanced Sudoku Puzzle ID: 60,763,368</div></div> <div><div>Congratulations! You have solved the puzzle in 03:37.11</div><div>Submit your score to the Hall of Fame</div><div><div></div><div></div><div></div></div><div><table><tr><td>3</td><td>5</td><td>7</td><td>4</td><td>8</td><td>6</td><td>2</td><td>9</td><td>1</td></tr><tr><td>9</td><td>8</td><td>6</td><td>1</td><td>2</td><td>3</td><td>5</td><td>7</td><td>4</td></tr><tr><td>4</td><td>2</td><td>1</td><td>7</td><td>9</td><td>5</td><td>6</td><td>8</td><td>3</td></tr><tr><td>5</td><td>1</td><td>2</td><td>6</td><td>4</td><td>8</td><td>9</td><td>3</td><td>7</td></tr><tr><td>6</td><td>7</td><td>4</td><td>3</td><td>1</td><td>9</td><td>8</td><td>5</td><td>2</td></tr><tr><td>8</td><td>3</td><td>9</td><td>2</td><td>5</td><td>7</td><td>4</td><td>1</td><td>6</td></tr><tr><td>2</td><td>9</td><td>3</td><td>8</td><td>6</td><td>1</td><td>7</td><td>4</td><td>5</td></tr><tr><td>1</td><td>6</td><td>8</td><td>5</td><td>7</td><td>4</td><td>3</td><td>2</td><td>9</td></tr><tr><td>7</td><td>4</td><td>5</td><td>9</td><td>3</td><td>2</td><td>1</td><td>6</td><td>8</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></div><div>3x3 Advanced Sudoku Puzzle ID: 60,763,368</div></div>	3		7	4							8		1						4				9		6			5	1		6		8						4				8						2		7		1	6			3		6				5					4		2							2	1		8		1	2	3	4	5	6	7	8	9	3	5	7	4	8	6	2	9	1	9	8	6	1	2	3	5	7	4	4	2	1	7	9	5	6	8	3	5	1	2	6	4	8	9	3	7	6	7	4	3	1	9	8	5	2	8	3	9	2	5	7	4	1	6	2	9	3	8	6	1	7	4	5	1	6	8	5	7	4	3	2	9	7	4	5	9	3	2	1	6	8	1	2	3	4	5	6	7	8	9
3		7	4																																																																																																																																																																																				
	8		1																																																																																																																																																																																				
4				9		6																																																																																																																																																																																	
5	1		6		8																																																																																																																																																																																		
		4				8																																																																																																																																																																																	
			2		7		1	6																																																																																																																																																																															
		3		6				5																																																																																																																																																																															
				4		2																																																																																																																																																																																	
				2	1		8																																																																																																																																																																																
1	2	3	4	5	6	7	8	9																																																																																																																																																																															
3	5	7	4	8	6	2	9	1																																																																																																																																																																															
9	8	6	1	2	3	5	7	4																																																																																																																																																																															
4	2	1	7	9	5	6	8	3																																																																																																																																																																															
5	1	2	6	4	8	9	3	7																																																																																																																																																																															
6	7	4	3	1	9	8	5	2																																																																																																																																																																															
8	3	9	2	5	7	4	1	6																																																																																																																																																																															
2	9	3	8	6	1	7	4	5																																																																																																																																																																															
1	6	8	5	7	4	3	2	9																																																																																																																																																																															
7	4	5	9	3	2	1	6	8																																																																																																																																																																															
1	2	3	4	5	6	7	8	9																																																																																																																																																																															
5	game5	Extreme	<div><div></div><div><table><tr><td>6</td><td></td><td></td><td>7</td><td></td><td>1</td><td></td><td></td><td>4</td></tr><tr><td></td><td>2</td><td></td><td>3</td><td></td><td>8</td><td></td><td>1</td><td></td></tr><tr><td></td><td></td><td></td><td>5</td><td>4</td><td>6</td><td></td><td></td><td></td></tr><tr><td>3</td><td>7</td><td>1</td><td></td><td></td><td></td><td>2</td><td>6</td><td>9</td></tr><tr><td></td><td></td><td>8</td><td></td><td></td><td></td><td>3</td><td></td><td></td></tr><tr><td>4</td><td>6</td><td>9</td><td></td><td></td><td></td><td>5</td><td>8</td><td>1</td></tr><tr><td></td><td></td><td></td><td>1</td><td>3</td><td>5</td><td></td><td></td><td></td></tr><tr><td></td><td>3</td><td></td><td>9</td><td></td><td>7</td><td></td><td>5</td><td></td></tr><tr><td>5</td><td></td><td></td><td>4</td><td></td><td>2</td><td></td><td></td><td>3</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></div><div>3x3 Extreme Sudoku Puzzle ID: 14,368,386</div></div> <div><div>Congratulations! You have solved the puzzle in 04:00.28</div><div>Submit your score to the Hall of Fame</div><div><div></div><div></div><div></div></div><div><table><tr><td>6</td><td>9</td><td>5</td><td>7</td><td>2</td><td>1</td><td>8</td><td>3</td><td>4</td></tr><tr><td>7</td><td>2</td><td>4</td><td>3</td><td>9</td><td>8</td><td>6</td><td>1</td><td>5</td></tr><tr><td>8</td><td>1</td><td>3</td><td>5</td><td>4</td><td>6</td><td>9</td><td>7</td><td>2</td></tr><tr><td>3</td><td>7</td><td>1</td><td>8</td><td>5</td><td>4</td><td>2</td><td>6</td><td>9</td></tr><tr><td>2</td><td>5</td><td>8</td><td>6</td><td>1</td><td>9</td><td>3</td><td>4</td><td>7</td></tr><tr><td>4</td><td>6</td><td>9</td><td>2</td><td>7</td><td>3</td><td>5</td><td>8</td><td>1</td></tr><tr><td>9</td><td>4</td><td>6</td><td>1</td><td>3</td><td>5</td><td>7</td><td>2</td><td>8</td></tr><tr><td>1</td><td>3</td><td>2</td><td>9</td><td>8</td><td>7</td><td>4</td><td>5</td><td>6</td></tr><tr><td>5</td><td>8</td><td>7</td><td>4</td><td>6</td><td>2</td><td>1</td><td>9</td><td>3</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></div><div>3x3 Extreme Sudoku Puzzle ID: 14,368,386</div></div>	6			7		1			4		2		3		8		1					5	4	6				3	7	1				2	6	9			8				3			4	6	9				5	8	1				1	3	5					3		9		7		5		5			4		2			3	1	2	3	4	5	6	7	8	9	6	9	5	7	2	1	8	3	4	7	2	4	3	9	8	6	1	5	8	1	3	5	4	6	9	7	2	3	7	1	8	5	4	2	6	9	2	5	8	6	1	9	3	4	7	4	6	9	2	7	3	5	8	1	9	4	6	1	3	5	7	2	8	1	3	2	9	8	7	4	5	6	5	8	7	4	6	2	1	9	3	1	2	3	4	5	6	7	8	9
6			7		1			4																																																																																																																																																																															
	2		3		8		1																																																																																																																																																																																
			5	4	6																																																																																																																																																																																		
3	7	1				2	6	9																																																																																																																																																																															
		8				3																																																																																																																																																																																	
4	6	9				5	8	1																																																																																																																																																																															
			1	3	5																																																																																																																																																																																		
	3		9		7		5																																																																																																																																																																																
5			4		2			3																																																																																																																																																																															
1	2	3	4	5	6	7	8	9																																																																																																																																																																															
6	9	5	7	2	1	8	3	4																																																																																																																																																																															
7	2	4	3	9	8	6	1	5																																																																																																																																																																															
8	1	3	5	4	6	9	7	2																																																																																																																																																																															
3	7	1	8	5	4	2	6	9																																																																																																																																																																															
2	5	8	6	1	9	3	4	7																																																																																																																																																																															
4	6	9	2	7	3	5	8	1																																																																																																																																																																															
9	4	6	1	3	5	7	2	8																																																																																																																																																																															
1	3	2	9	8	7	4	5	6																																																																																																																																																																															
5	8	7	4	6	2	1	9	3																																																																																																																																																																															
1	2	3	4	5	6	7	8	9																																																																																																																																																																															
6	game6	Evil	<div><div></div><div><table><tr><td></td><td>3</td><td></td><td>9</td><td></td><td></td><td>7</td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>6</td><td></td><td>5</td><td></td><td>3</td></tr><tr><td>5</td><td>6</td><td></td><td>3</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td>4</td><td></td><td>8</td><td></td><td>9</td></tr><tr><td></td><td>8</td><td></td><td>5</td><td></td><td>1</td><td></td><td>7</td><td></td></tr><tr><td>9</td><td></td><td>2</td><td></td><td>3</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td>3</td><td></td><td>1</td><td>7</td></tr><tr><td>7</td><td></td><td>4</td><td></td><td>1</td><td></td><td></td><td></td><td></td></tr><tr><td></td><td>3</td><td></td><td></td><td>7</td><td></td><td>5</td><td></td><td></td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></div><div>3x3 Evil Sudoku Puzzle ID: 4,893,670</div></div> <div><div>Congratulations! You have solved the puzzle in 03:04.78</div><div>Submit your score to the Hall of Fame</div><div><div></div><div></div><div></div></div><div><table><tr><td>2</td><td>3</td><td>1</td><td>9</td><td>8</td><td>5</td><td>7</td><td>4</td><td>6</td></tr><tr><td>4</td><td>9</td><td>7</td><td>1</td><td>6</td><td>2</td><td>5</td><td>8</td><td>3</td></tr><tr><td>5</td><td>6</td><td>8</td><td>3</td><td>7</td><td>4</td><td>2</td><td>9</td><td>1</td></tr><tr><td>1</td><td>7</td><td>5</td><td>2</td><td>4</td><td>6</td><td>8</td><td>3</td><td>9</td></tr><tr><td>3</td><td>8</td><td>6</td><td>5</td><td>9</td><td>1</td><td>4</td><td>7</td><td>2</td></tr><tr><td>9</td><td>4</td><td>2</td><td>7</td><td>3</td><td>8</td><td>1</td><td>6</td><td>5</td></tr><tr><td>8</td><td>2</td><td>9</td><td>4</td><td>5</td><td>3</td><td>6</td><td>1</td><td>7</td></tr><tr><td>7</td><td>5</td><td>4</td><td>6</td><td>1</td><td>9</td><td>3</td><td>2</td><td>8</td></tr><tr><td>6</td><td>1</td><td>3</td><td>8</td><td>2</td><td>7</td><td>9</td><td>5</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr></table></div><div>3x3 Evil Sudoku Puzzle ID: 4,893,670</div></div>		3		9			7							6		5		3	5	6		3										4		8		9		8		5		1		7		9		2		3										3		1	7	7		4		1						3			7		5			1	2	3	4	5	6	7	8	9	2	3	1	9	8	5	7	4	6	4	9	7	1	6	2	5	8	3	5	6	8	3	7	4	2	9	1	1	7	5	2	4	6	8	3	9	3	8	6	5	9	1	4	7	2	9	4	2	7	3	8	1	6	5	8	2	9	4	5	3	6	1	7	7	5	4	6	1	9	3	2	8	6	1	3	8	2	7	9	5	4	1	2	3	4	5	6	7	8	9
	3		9			7																																																																																																																																																																																	
				6		5		3																																																																																																																																																																															
5	6		3																																																																																																																																																																																				
				4		8		9																																																																																																																																																																															
	8		5		1		7																																																																																																																																																																																
9		2		3																																																																																																																																																																																			
					3		1	7																																																																																																																																																																															
7		4		1																																																																																																																																																																																			
	3			7		5																																																																																																																																																																																	
1	2	3	4	5	6	7	8	9																																																																																																																																																																															
2	3	1	9	8	5	7	4	6																																																																																																																																																																															
4	9	7	1	6	2	5	8	3																																																																																																																																																																															
5	6	8	3	7	4	2	9	1																																																																																																																																																																															
1	7	5	2	4	6	8	3	9																																																																																																																																																																															
3	8	6	5	9	1	4	7	2																																																																																																																																																																															
9	4	2	7	3	8	1	6	5																																																																																																																																																																															
8	2	9	4	5	3	6	1	7																																																																																																																																																																															
7	5	4	6	1	9	3	2	8																																																																																																																																																																															
6	1	3	8	2	7	9	5	4																																																																																																																																																																															
1	2	3	4	5	6	7	8	9																																																																																																																																																																															

- Kết quả chạy:

Game	Độ khó	Đúng đáp án		Thời gian chạy		Bộ nhớ tối đa	
		DFS (Y/N)	Climb (Y/N)	DFS (giây)	Climb (giây)	DFS (bytes)	Climb (bytes)
game1	Basic	Y	Y	3.380234	0.057464	81479	77785
game2	Easy	Y	Y	4.510625	0.044392	82215	78017
game3	Inter	Y	Y	0.890106	0.123616	77160	74105
game4	Advance	Y	Y	20.09316	0.193598	83687	79993
game5	Extreme	Y	Y	0.434091	0.069995	72764	72633
game6	Evil	Y	Y	6.366642	0.178845	82215	78521

Game	Độ khó	Độ vượt trội về thời gian chạy ( $T_{DFS}/T_{Climb}$ ) %	Độ vượt trội về bộ nhớ ( $M_{DFS}/M_{Climb}$ ) %
game1	Basic	5882%	105%
game2	Easy	10161%	105%
game3	Inter	720%	104%
game4	Advance	10379%	105%
game5	Extreme	620%	100%
game6	Evil	3560%	105%

### 2.6.3. Nhận xét và giải thích:

- **Nhận xét:**

- + Trong cùng một game, luôn luôn thuật toán leo đồi (Climb) sẽ cho thời gian chạy ra kết quả và bộ nhớ tối đa cần sử dụng ít hơn so với việc giải bằng thuật toán Blind Search (DFS).
- + Cả hai thuật toán có xu hướng khá giống nhau trong việc sử dụng tài nguyên bộ nhớ và thời gian để giải một bài toán (cùng tăng, cùng giảm để giải một game).
- + Về mặt thời gian chạy, giữa 2 thuật toán có độ chênh lệch khá lớn, thuật toán Climb chạy nhanh hơn từ 6-100 lần trong các bài thử nghiệm ở game Sudoku này.
- + Về mặt bộ nhớ, giữa 2 thuật toán có sự chênh lệch, nhưng hầu như không đáng kể.

- **Giải thích:**

- + Gọi x là số ô trống cần phải điền vào trong câu đố sudoku (mỗi ô trống sẽ cần xem xét 9 số có thể điền vào). Ta có bảng thống kê sau:

Game	Độ khó	Số ô trống
game1	Basic	52
game2	Easy	53
game3	Inter	47
game4	Advance	55
game5	Extreme	45
game6	Evil	53

- + Bài toán có maximum depth bằng  $x$  (khi tất cả  $x$  ô trống được điền) và solution depth bằng  $x$  (khi tất cả  $x$  ô trống được điền đúng).
  - + Bài toán có độ phức tạp thuật toán là  $9^x$
  - + Bài toán có độ phức tạp không gian cố định là  $9.x$ ; cụ thể bộ nhớ tối đa mà chương trình cần sử dụng là  $(9.x.A + b)$  trong đó  $x$  là số ô trống được điền,  $A$  là kích thước bộ nhớ cho một ma trận biểu diễn sudoku;  $b$  là các kích thước các biến phụ hoặc chạy trong các hàm phụ.
- ⇒ **Về mặt thời gian:** Hàm chạy thuật toán leo đồi được tối ưu để nhanh chóng tìm được lời giải hơn thông qua hàm lượng giá, giúp chọn được các ô dẫn đến trạng thái mục tiêu có xác suất chọn trúng số cao hơn, ít có nhánh phụ, do đó số nhánh, node mà hàm đi qua sẽ ít hơn, giúp chương trình chạy nhanh hơn, đặc biệt khi độ khó của game sudoku tăng lên, hoặc số ô trống càng nhiều (vì DFS sẽ dễ vướng bẫy, vướng nhiều nhánh cụt hơn).
- ⇒ **Về mặt bộ nhớ:** cả 2 hàm đều chiếm dụng không gian bộ nhớ tối đa là  $(9.x.A+b)$ ; có cùng thông số về  $x$ , và  $A$ ; độ chênh lệch do bộ nhớ dành cho các biến phụ không quá lớn, dẫn đến mức chênh lệch tổng thể bộ nhớ tối đa sử dụng của 2 thuật toán không đáng kể.
- + Trong một số trường hợp tùy vào giá trị của ô cần điền vào và vị trí của chúng mà thuật toán sẽ dẫn đến đáp nhanh hơn. Ví dụ với 2 ô game dưới đây, cùng có 4 ô chưa điền (là các ô đánh dấu màu vàng); thì việc giải ô game số 2 sẽ nhanh hơn game số 1 nhiều, vì ở game 1 các số cần điền có giá trị rất lớn, trong khi đó thuật toán sẽ thử dần từ các số nhỏ đến số lớn, xác suất gặp các nhánh cụt trước (không dẫn đến đáp án) là rất cao.

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

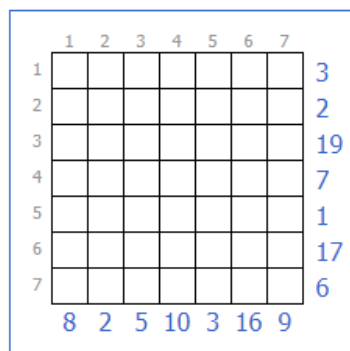
Game 1

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

Game 2

### 3. GAME KAKURASU

#### 3.1. Giới thiệu game và cách chơi



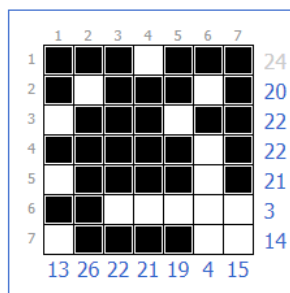
7x7 Easy Kakurasu Puzzle ID: 9,999,392

Hình 3.1: Ví dụ một ván game Kakurasu kích thước 7x7

- Kakurasu là một trò chơi logic thú vị của nhật bản, mục tiêu của trò chơi là điền các ô vuông trong một bảng chữ nhật sao cho tổng giá trị của các ô đã được điền theo hàng và cột tương ứng đúng bằng một số đã cho ở phía dưới và bên phải của bảng.
- Cấu trúc của bảng game thường là một lưới có kích thước  $n \times n$ , trong đó  $n$  là một số nguyên dương (thường gặp với  $4 \leq n \leq 9$ ). Ở phía dưới và bên phải của bảng, tương ứng sẽ có một hàng và một cột các số cho sẵn. Những con số này đại diện cho tổng giá trị của các ô vuông trong hàng và cột tương ứng mà người chơi cần phải điền.
- Cách chơi Kakurasu khá đơn giản: Khi một ô vuông được chọn, nó sẽ mang giá trị bằng thứ tự hàng của ô khi xét tổng các ô có cùng cột, và mang giá trị bằng thứ tự của cột khi xét tổng các ô có cùng hàng để thỏa mãn yêu cầu tổng theo cột và hàng tương ứng. Bài toán được giải khi ta chọn được bộ các ô thỏa mãn: **Tổng giá trị theo hàng và cột của các ô được chọn sẽ bằng cột bên phải và hàng phía dưới.**

Congratulations! You have solved the puzzle in 01:28.00

Submit your score to the Hall of Fame



7x7 Easy Kakurasu Puzzle ID: 7,597,779

Hình 3.2: Ví dụ một ván game Kakurasu kích thước 7x7 đã được giải thành công

- Link game: <https://www.puzzle-kakurasu.com/>



### 3.2. Biểu diễn game Kakurasu:

#### 3.2.1. Không gian trạng thái:

- Ta sử dụng ma trận  $(n+1) \times (n+1)$  (ma trận A/ ma trận board) để biểu diễn trạng thái của game có kích thước  $n \times n$ , với  $n$  là số nguyên dương.
- Các giá trị tại vị trí  $A[n][k]$  và  $A[k][n]$  (với  $0 \leq k \leq n-1$ ) là giá trị cho trước, các giá trị này phải là số nguyên dương và sẽ không thay đổi trong một ván game.
- Các ô còn lại nằm ở vị trí  $A[i][j]$  (với  $0 \leq i, j \leq n-1$ ) sẽ chỉ có 3 giá trị là 0 (chưa xét chọn), -1 (không được chọn) hoặc 1 (được chọn).
- Khi giá trị vị trí  $A[i][j] = 1$  (với  $0 \leq j \leq n-1$ ) thì  $B[i][j]$  sẽ có giá trị là  $(j+1)$  khi tính tổng theo hàng, và có giá trị là  $(i+1)$  khi tính tổng theo cột.
- Khi giá trị vị trí  $A[i][j] = 0$  hoặc  $A[i][j] = -1$  (với  $0 \leq j \leq n-1$ ) thì  $B[i][j]$  sẽ có giá trị bằng 0 khi tính tổng theo hàng và theo cột.
- Trong python, ta có thể sử dụng cấu trúc dữ liệu là một list có  $(n+1)$  phần tử, mỗi phần tử của list là một list có  $(n+1)$  phần tử số nguyên để biểu diễn ma trận  $(n+1) \times (n+1)$  này. Việc xử lý ở bài toán này được xử lý chính trên cấu trúc dữ liệu này, thư viện numpy được sử dụng chỉ để in ra màn hình cấu trúc ma trận của biến.

#### 3.2.2. Trạng thái khởi đầu:

- Trạng thái khởi đầu của một game có kích thước  $n$  là một ma trận  $(n+1) \times (n+1)$  với các ô tại cột thứ  $n+1$  và hàng thứ  $n+1$  đã được cho trước các giá trị nguyên dương -  $A[n][k]=x$  và  $A[k][n]=y$  (với  $0 \leq k \leq n-1$ ;  $0 \leq x, y$ ). Các ô còn lại được cho giá trị mặc định bằng 0 -  $A[i][j] = 0$  (với  $0 \leq i, j \leq n-1$ ).

	1	2	3	4	5	6	7	
1								13
2								27
3								16
4								17
5								21
6								25
7								13
	26	18	18	21	13	16	24	

Hình 3.3: Minh họa trạng thái khởi đầu game

```
game = [ [0, 0, 0, 0, 0, 0, 0, 13],
          [0, 0, 0, 0, 0, 0, 0, 27],
          [0, 0, 0, 0, 0, 0, 0, 16],
          [0, 0, 0, 0, 0, 0, 0, 17],
          [0, 0, 0, 0, 0, 0, 0, 21],
          [0, 0, 0, 0, 0, 0, 0, 25],
          [0, 0, 0, 0, 0, 0, 0, 13],
          [26, 18, 18, 21, 13, 16, 24, 0]]
```

Hình 3.4: Minh họa biểu diễn trạng thái khởi đầu của game

#### 3.2.3. Trạng thái mục tiêu:

- Trạng thái mục tiêu là trạng thái mà cả 2 điều kiện sau được thỏa mãn :  
 $\sum_{i=0}^{n-1} B[i][k] = A[n][k]$  và  $\sum_{i=0}^{n-1} B[k][i] = A[i][n]$  với  $(0 \leq k \leq n-1)$  (\*\*)

Trong đó :

+  $B[i][k] = i+1$  khi xét tổng cột và giá trị ô  $A[i][k] = 1$ ; nếu  $A[i][k]=0$  hoặc  $A[i][k]=-1$  thì  $B[i][k]=0$

+  $B[k][i] = i+1$  khi xét tổng hàng và giá trị ô  $A[k][i] = 1$ ; nếu  $A[k][i]=0$  hoặc  $A[i][k]=-1$  thì  $B[k][i]=0$

	1	2	3	4	5	6	7	
1	1	0	1	1	1	0	0	13
2	0	1	1	1	1	1	1	27
3	1	1	1	1	0	1	0	16
4	1	0	0	1	1	0	1	17
5	1	0	1	1	0	1	1	21
6	1	1	0	1	1	1	1	25
7	1	1	1	0	0	0	1	13
	26	18	18	21	13	16	24	

Hình 3.5: Minh họa một trạng thái mục tiêu thỏa mãn

[	1	0	1	1	1	0	0	13]
[	0	1	1	1	1	1	1	27]
[	1	1	1	1	0	1	0	16]
[	1	0	0	1	1	0	1	17]
[	1	0	1	1	0	1	1	21]
[	1	1	0	1	1	1	1	25]
[	1	1	1	0	0	0	1	13]
[26	18	18	21	13	16	24	0]	

Hình 3.6: Minh họa biểu diễn một trạng thái mục tiêu ở dạng ma trận

#### 3.2.4. Luật chuyển trạng thái

- Các bước đi hợp lệ: thay đổi giá trị của các ô  $A[i][j]$  (với  $0 \leq i, j \leq n-1$ ) từ 0 sang 1 và từ 1 sang 0.
- Do việc thay đổi cần đảm bảo tổng giá trị các ô theo hàng và theo cột bằng giá trị ngoài cùng bên phải và dưới cùng bên dưới, do đó để nhanh chóng tìm ra đáp án, ta sẽ thay đổi hàng loạt trạng thái các ô trong cùng một hàng hoặc một cột trong một lần thay đổi trạng thái.

### 3.3. Xây dựng các hàm cơ bản:

- Xây dựng hàm *get\_value\_row(board, row, col)* để lấy giá trị của ô  $A[\text{row}, \text{col}]$  khi xét tổng hàng.

```
def get_value_row(board, row, col):
    if board[row][col] == 1:
        return col + 1
    else:
        return 0
```

- Xây dựng hàm *get\_value\_col(board, row, col)* để lấy giá trị của ô  $A[\text{row}, \text{col}]$  khi xét tổng cột

```
def get_value_col(board, row, col):
    if board[row][col] == 1:
        return row + 1
    else:
        return 0
```

- Xây dựng hàm *check\_sum(target, set\_lst)* để kiểm tra tổng các số trong *lst* có bằng *target* hay không, nếu có trả về **True**, ngược lại trả về **False**.

```
def check_sum(target, set_lst):
    if set_lst is None: return False
    else:
        if sum(set_lst) == target: return True
        return False
```

- Xây dựng hàm *check\_final(board, size)* để kiểm tra trạng thái hiện tại có phải là trạng thái mục tiêu hay không. Trạng thái hiện tại là trạng thái mục tiêu khi và chỉ khi điều kiện (\*\*) được thỏa mãn.

```
def check_final(board, size):
    for row in range(0, size):
        sum_row = 0
        for col in range(0, size):
            sum_row += get_value_right(board, row, col)
        if sum_row != board[row][size]:
            return False
    for col in range(0, size):
        sum_col = 0
        for row in range(0, size):
            sum_col += get_value_bottom(board, row, col)
        if sum_col != board[size][col]:
            return False
    return True
```

### 3.4. Xây dựng hàm giải bài toán bằng giải thuật Blind Search

- **Nhận thấy:** để biết các tổ hợp số mà cộng lại bằng một giá trị cho trước, ta cần tạo ra tập hợp các tổ hợp số có thể có, và xét tổng của chúng so với giá trị cho trước. Cụ thể với game kích thước *n* thì giá trị của một ô khi được chọn sẽ là số nguyên *k* nằm trong khoảng  $1 \leq k \leq n$ , và tổ hợp sẽ có *m* phần tử với  $1 \leq m \leq n$ . Để nhanh chóng sinh ra đủ các tổ hợp số có *m* phần tử từ tập hợp có *k* số ( $C_k^m$ ), sinh viên đã dùng hàm **combinations** của thư viện **itertools**. Ví dụ:

```
1 from itertools import combinations
2 k = {1,2,3,4}
3 m = 2
4 lst = list(combinations(k,m))
5 print(lst)
```

[(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)]

- Xây dựng hàm *valid\_combination(target, size)* để lấy được tất cả tổ hợp số sao cho chúng có tổng bằng *target* (các số có giá trị từ 1 đến *size*, và chúng chỉ xuất hiện tối đa 1 lần trong 1 một tổ hợp số). Hàm sẽ trả về một danh sách, với mỗi phần tử của danh sách là một set các số mà tổng của chúng bằng *target*, trong trường hợp không có tổ hợp nào thỏa mãn, hàm sẽ trả về một danh sách rỗng.

```
def valid_combination(target, size):
    result = []
    lst = [i for i in range(1, size + 1)]    # tao list cac so de
    # phuc vu tao combination
    for i in range(1, size+1):
        temp_lst = list(combinations(lst, i))
        for j in temp_lst:
            if check_sum(target, j): result.append(j)
    return result
```

- Sử dụng thuật toán vét cạn theo chiều sâu DFS, thông qua việc xây dựng hàm đệ quy *solve(board, size, row)*.
  - + Đầu tiên hàm sẽ kiểm tra trạng thái hiện tại có phải phải là trạng thái mục tiêu hay chưa, nếu phải thì hàm sẽ trả về giá trị bảng board hiện tại.
  - + Giá trị row mặc định khi gọi hàm ta sẽ set giá trị bằng 0, để hàm có thể lần lượt duyệt qua và chọn ô ở tất cả các hàng của bảng. Hàm sẽ dừng và trả về None khi giá trị của row vượt quá giá trị size, tức đã duyệt hết các dòng, nhưng vẫn chưa tìm được đáp án.
  - + Hàm *solve* sẽ tạo danh sách các tổ hợp phù hợp mà tổng của chúng bằng giá trị cột ngoài cùng bên phải, sau đó chúng sẽ thử lần lượt chọn tổ hợp các ô trong các danh sách này. Nếu danh sách nhận được là danh sách rỗng, tức không có tổ hợp nào thỏa mãn, hàm sẽ trả về None
  - + Ứng với mỗi tổ hợp trong danh sách trên, sau khi chọn các ô tương ứng và cập nhật vào board, hàm *solve* sẽ gọi lại chính nó, nhưng với bảng vừa cập nhật và row tiếp theo (*row+1*).
  - + Nếu kết quả trả về của lời gọi này là *None* nghĩa là tổ hợp đã chọn không dẫn đến đáp án; ngược lại nếu lời gọi hàm này trả về một *board* thì đây là đáp án, và tổ hợp vừa chọn là tổ hợp dẫn đến đáp án.

```
def solve(board, size, row):
    if check_final(board, size):
        return board
    if row > size:
        return None
    valid_lst = valid_combination(board[row][size], size)
    if len(valid_lst) == 0: return None
    for lst in valid_lst:
        for value in lst:
            board[row][value-1] = 1
        print("Next step is")
        print(np.asarray(board))
        if solve(board, size, row+1) is None:
            for value in lst:
                board[row][value-1] = 0
            print("IT'S NOT WAY TO SOLUTION- SET BOARD BACK: ")
            print(np.asarray(board))
```

```
else:
    return solve(board, size, row+1)
```

- Ta thấy, hàm *solve(board, size, row)* sẽ chạy lần lượt hết các hàng của bảng, với mỗi hàng, hàm sẽ thử lần lượt tất cả các tổ hợp ô mà tổng của chúng bằng giá trị cột ngoài cùng bên phải kết hợp các tổ hợp ở các hàng tiếp theo, cho đến khi tìm được đáp án hoặc khi chạy hết tất cả tổ hợp và không có đáp án phù hợp. Như vậy cách chạy của hàm *solve* phù hợp với thuật toán tìm kiếm vét cạn theo chiều sâu DFS.

### 3.5. Xây dựng hàm giải bài toán bằng giải thuật Heuristic Search

- **Nhận thấy:** mỗi target sẽ có một số lượng tổ hợp số nhất định mà tổng của chúng bằng với target, như vậy cột/ hàng nào có target mà số tổ hợp phù hợp càng ít thì xác suất ta chọn được đúng tổ hợp số cho cột/ hàng đó càng cao.  
Sau khi chọn các ô theo phương hàng (dựa vào target ngoài cùng bên phải) hoặc chọn các ô theo phương thẳng đứng (dựa vào target dưới cùng); thì các ô còn lại của hàng/ cột đó sẽ không được chọn kể cả khi đang xét trong các hàng/cột khác.
- Từ cơ sở trên, sinh viên sử dụng thuật toán tìm kiếm leo đồi, thông qua việc xây dựng thêm hàm lượng giá *select\_where\_to\_action(board, size)* để lựa chọn hàng/ cột nào có ưu điểm nhất để chọn ô trong hàng/ cột đó tiếp theo.
- Để phục vụ xây dựng hàm lượng giá trên, sinh viên xây dựng lại một số hàm phụ trợ như sau:
  - + Xây dựng hàm *get\_list\_number\_available\_col(board, size, col)* và *get\_list\_number\_available\_row(board, size, row)*. Hai hàm này sẽ trả về gồm 2 danh sách: một danh sách là các ô (số) có thể chọn trong hàng/ cột này, và một danh sách là các ô (số) bắt buộc phải chọn trong hàng/ cột này.
    - Danh sách các ô có thể chọn trong hàng/ cột này là danh sách các ô chưa bị đánh dấu không được chọn (mark) tức giá trị  $B[i][j]$  có  $A[i][j] \neq -1$ .
    - Danh sách các ô số bắt buộc phải chọn trong hàng/ cột này là danh sách các ô **đã** được chọn thông qua các hàng/ cột đã xét trước đó, tức các giá trị  $B[i][j]$  có  $A[i][j] = 1$ .
    - Với một hàng/ cột đã hoàn thành chọn ô (tổng giá trị các ô trong hàng/ cột bằng target) thì ta sẽ trả về một list gồm 2 danh sách rỗng để phân biệt.

```
def get_list_number_available_row(board, size, row):
    lst = []
    picked_lst = []
    # check complete row
    sum_row = 0
    for j in range(size):
        if board[row][j] == 1: sum_row += j + 1
    if sum_row == board[row][size]: return [], []
    # not complete row
```

```

    for i in range(size):
        if board[row][i] != -1:
            lst.append(i+1)
    # lists number must have:
    for i in range(size):
        if board[row][i] == 1:
            picked_lst.append(i+1)
    return [lst, picked_lst]

def get_list_number_available_col(board, size, col):
    lst = []
    picked_lst = []
    # check complete col
    sum_col = 0
    for j in range(size):
        if board[j][col] == 1: sum_col += j+1
    if sum_col == board[size][col]: return []

    # not complete col
    for i in range(size):
        if board[i][col] != -1:
            lst.append(i+1)
    # lists number must have:
    for i in range(size):
        if board[i][col] == 1:
            picked_lst.append(i+1)
    return [lst, picked_lst]

```

+ Xây dựng hàm *valid\_combination(target, size, lst)* với *lst* là danh sách lấy từ hàm *get\_list\_number\_available\_col* và *get\_list\_number\_available\_row*. Hàm sẽ trả về một danh sách gồm tất cả tổ hợp số, mà các tổ hợp này tạo thành từ các số nằm trong danh sách *lst[0]*, có chứa đủ các số trong danh sách *lst[1]* và có tổng bằng *target*, mỗi số trong tổ hợp là riêng biệt (xuất hiện 1 lần trong 1 tổ hợp).

```

def valid_combination(target, size, lst):
    result = []
    if len(lst) == 1: return result
    valid_num = lst[0]
    picked_num = lst[1]
    if len(valid_num) == 0: return result
    for i in range(1, size):
        temp_lst = list(combinations(valid_num, i))
        for j in temp_lst:
            if not check_sum(target, j): continue
            get = True
            for number in picked_num:
                if number not in j:
                    get = False
                    break

```

```

        if get:
            result.append(j)
    return result

```

- Hàm lượng giá *select\_where\_to\_action(board, size)* sẽ xem xét tất cả các hàng và cột ở trạng thái hiện tại, với mỗi hàng và cột, hàm sẽ tìm danh sách các tổ hợp phù hợp tại hàng và cột đó thông qua hàm *valid\_combination* và *get\_list\_number\_available\_col / row*. Sau đó hàm sẽ chọn ra hàng/ cột có số tổ hợp là ít nhất để xác xuất ta chọn đúng tổ hợp dẫn đến trạng thái mục tiêu là cao nhất.

Hàm trả về thông tin dạng (*direct, indx, lst\_combination*) trong đó *direct* bằng row hoặc col; *indx* chỉ index của row/ col tương ứng, và *lst\_combination* trả về danh sách các tổ hợp phù hợp tại row/col đó. Nếu tất cả các hàng, cột đã được chọn hoặc số lượng tổ hợp nhỏ nhất là 0 (sau khi loại trừ các hàng, cột đã hoàn tất chọn ô), hàm sẽ trả về *None*.

```

def select_where_to_action(board, size):
    row_number = [get_list_number_available_row(board, size, row) for
row in range(size)]
    col_number = [get_list_number_available_col(board, size, col) for
col in range(size)]
    row_lst_combination = []
    col_lst_combination = []
    for i in range(size):
        temp_row_lst =
valid_combination(board[i][size], size, row_number[i])
        row_lst_combination.append(temp_row_lst)
        temp_col_lst =
valid_combination(board[size][i], size, col_number[i])
        col_lst_combination.append(temp_col_lst)
    row_count_combi = [len(i) for i in row_lst_combination]
    col_count_combi = [len(i) for i in col_lst_combination]
    for i in range(size):
        if row_count_combi[i]==0: row_count_combi[i]=size+1
        if col_count_combi[i]==0: col_count_combi[i]=size+1
    min_row = min(row_count_combi)
    min_col = min(col_count_combi)
    if min_col== size+1: return None
    if min_row <= min_col:
        indx = row_count_combi.index(min_row)
        return ('row', indx, row_lst_combination[indx])
    else:
        indx = col_count_combi.index(min_col)
        return ('col', indx, col_lst_combination[indx])

```

- Xây dựng hàm đệ quy *solve(board, size)* để giải bài toán:
  - + Đầu tiên hàm sẽ kiểm tra trạng thái hiện tại có phải phải là trạng thái mục tiêu hay chưa, nếu phải thì hàm sẽ trả về giá trị bảng board hiện tại. (7)



+ Hàm sẽ lựa chọn hàng/ cột tiếp theo để chọn ô dựa vào thông tin nhận được từ hàm lượng giá *select\_where\_to\_action*. Nếu giá trị từ hàm lượng giá là None thì hàm đệ quy sẽ trả về None. (8)

+ Sau khi nhận thông tin từ hàm lượng giá, ta sẽ lần lượt thử cập nhật từng tổ hợp số vào hàng/ cột tương ứng, trong **ma trận A** các ô không phải giá trị trong tổ hợp (không được chọn) trong hàng/ cột đó sẽ được gán giá trị bằng -1, và các ô được chọn sẽ được gán giá trị bằng 1. Trước khi gán giá trị của cột/ hàng tương ứng vào ma trận A, ta làm thêm một động tác là lưu lại danh sách các ô được chọn (*ori\_pick*) và danh sách các ô đánh dấu (*ori\_mark*) để có thể khôi phục lại trạng thái này sau này nếu cần.

+ Sau khi cập nhật ma trận A (board), hàm *solve* sẽ gọi lại chính nó, nếu kết quả trả về của lời gọi này là **None** nghĩa là tổ hợp vừa chọn không dẫn đến đáp án; hàm sẽ quay lại trạng thái ma trận trước khi chọn tổ hợp đó thông qua danh sách các ô *ori\_mark* và *ori\_pick*, ngược lại nếu lời gọi hàm này trả về một **board** thì đây là đáp án, và tổ hợp vừa chọn là tổ hợp dẫn đến đáp án.

```
def solve(board, size):
    if check_final(board, size):
        return board
    info = select_where_to_action(board, size)
    if info is None: return None
    print("CAC TO HOP CO THE CHON O ", info[0], " ", info[1], " LA
", info[2])
    if info[0] == 'row':
        row = info[1]
        lst = info[2]
        for element in lst:
            ori_mark = [i for i in range(size) if board[row][i] == -1]
            ori_pick = [i for i in range(size) if board[row][i] == 1]
            print("EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE action with
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE ", element)
            for i in range(size):
                if i+1 in element: board[row][i]=1
                else: board[row][i]=-1
            print("NEXT STEP IS")
            print(np.asarray(board))

    if solve(board, size) is None:
        print("IT'S NOT WAY TO SOLUTION- SET BOARD BACK: ")
        for i in range(size):
            if i in ori_mark:
                board[row][i]=-1
            else:
                if i not in ori_pick:
                    board[row][i]=0
        print(np.asarray(board))
    else: return solve(board, size)
```



```

else: # action by column
    col = info[1]
    lst = info[2]

    for element in lst:
        ori_mark = [i for i in range(size) if board[i][col] == -1]
        ori_pick = [i for i in range(size) if board[i][col] == 1]

        print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ action with
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ ", element)
        for i in range(size):
            if i+1 in element: board[i][col]=1
            else: board[i][col]=-1
        print("NEXT STEP IS")
        print(np.asarray(board))

        if solve(board,size) is None:
            print("IT'S NOT WAY TO SOLUTION- SET BOARD BACK: ")
            for i in range(size):
                if i in ori_mark:
                    board[i][col]=-1
                else:
                    if i not in ori_pick:
                        board[i][col]=0
            print(np.asarray(board))
        else: return solve(board,size)

```

### 3.6. Kết quả chạy, so sánh và giải thích

#### 3.6.1. Phương pháp đo:

- Để đo thời gian chạy của chương trình, sinh viên sử dụng thư viện time của python, đo thời điểm bắt đầu và kết thúc của chương trình, từ đó tính hiệu của 2 mốc ta sẽ có được thời gian chương trình tiêu tốn.
- Để đo mức memory tối đa mà chương trình sử dụng, ta sử dụng thư viện tracemalloc, cập giá trị trả về sẽ gồm bộ nhớ tiêu tốn hiện tại và bộ nhớ tiêu tốn đỉnh của chương trình, ta chỉ quan tâm đến giá trị thứ 2.
- Đoạn code phục vụ kiểm tra tài nguyên và thời gian chạy sẽ có dạng:

```

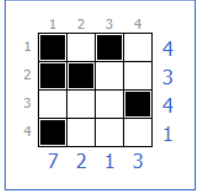
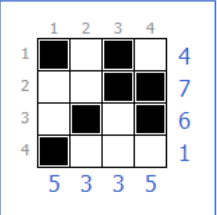
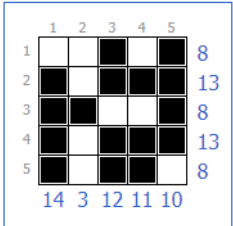
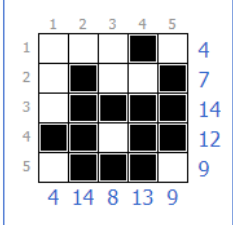
start = time.time()
tracemalloc.clear_traces()
tracemalloc.start()
solve(np.asarray(game))
print(f"Bộ nhớ tối đa sử dụng: {tracemalloc.get_traced_memory()[1]}
bytes")
tracemalloc.stop()
print(f"Thời gian chạy là {time.time()-start} giây ")

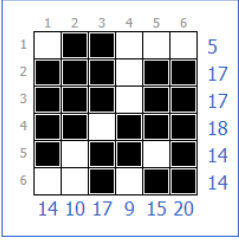
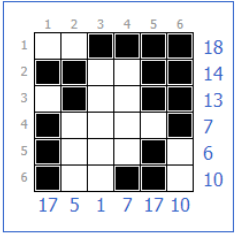
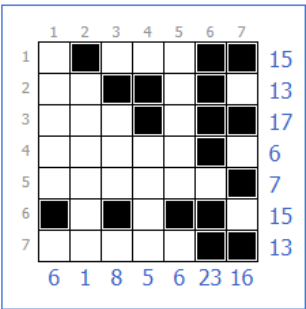
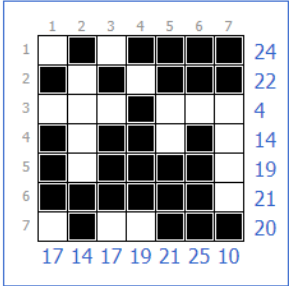
```

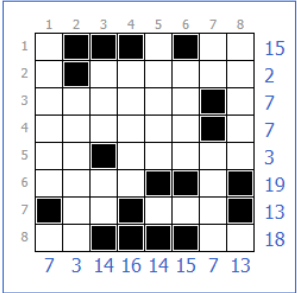
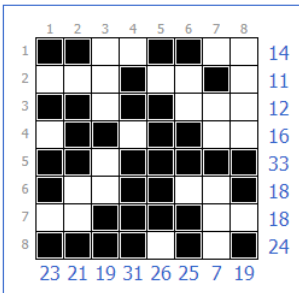
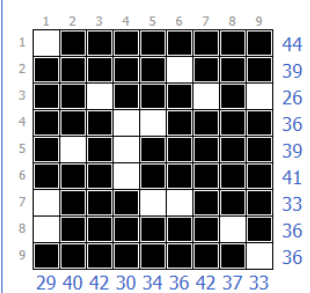
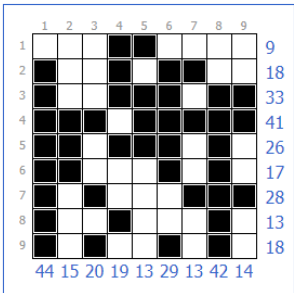
#### 3.6.2. Kết quả chạy thực nghiệm:

- Các testcase được lấy từ trang <https://www.puzzle-sudoku.com/>

- Có tất cả 6 kích thước phổ biến cho game Kakurasu trên trang web (từ 4x4 đến 9x9), ứng mỗi kích thước game có 2 mức độ là easy và hard; sinh viên lấy mỗi trường hợp một game và chạy lần lượt cả 2 thuật toán giải để giải, ghi nhận kết quả.

STT	Tên	Mức độ	Input	Solution Pass
1	Size4x4	Easy	<pre> [0,0,0,0,4] [0,0,0,0,3] [0,0,0,0,4] [0,0,0,0,1] [7,2,1,3,0] </pre>	<p>Congratulations! You have solved the puzzle in 02</p> <p>Submit your score to the Hall of Fame</p>  <p>4x4 Easy Kakurasu Puzzle ID: 444,844</p>
2	Size4x4	Hard	<pre> [0,0,0,0,4] [0,0,0,0,7] [0,0,0,0,6] [0,0,0,0,1] [5,3,3,5,0] </pre>	<p>Congratulations! You have solved the puzzle in 00</p> <p>Submit your score to the Hall of Fame</p>  <p>4x4 Hard Kakurasu Puzzle ID: 8,936,569</p>
3	Size5x5	Easy	<pre> [ 0, 0, 0, 0, 0, 8 ], [ 0, 0, 0, 0, 0, 13 ] [ 0, 0, 0, 0, 0, 8 ], [ 0, 0, 0, 0, 0, 13 ] [ 0, 0, 0, 0, 0, 8 ], [ 14,3,12, 11,10,0]] </pre>	<p>Congratulations! You have solved the puzzle in 00</p> <p>Submit your score to the Hall of Fame</p>  <p>5x5 Easy Kakurasu Puzzle ID: 3,122,400</p>
4	Size5x5	Hard	<pre> [ 0, 0, 0, 0, 0, 4 ] [ 0, 0, 0, 0, 0, 7 ] [ 0, 0, 0, 0, 0,14 ] [ 0, 0, 0, 0, 0,12 ] [ 0, 0, 0, 0, 0, 9 ] [ 4,14, 8, 13, 9,0] </pre>	<p>Congratulations! You have solved the puzzle in 00</p> <p>Submit your score to the Hall of Fame</p>  <p>5x5 Hard Kakurasu Puzzle ID: 660,570</p>

5	Size6x6	Easy	<pre> [0 , 0, 0, 0, 0, 0, 5], [0 , 0, 0, 0, 0, 0, 17] [0 , 0, 0, 0, 0, 0, 17] [0 , 0, 0, 0, 0, 0, 18] [0 , 0, 0, 0, 0, 0,14], [0 , 0, 0, 0, 0, 0,14], [14 ,10,17, 9,15,20, 0] </pre>	<p>Congratulations! You have solved the puzzle in 01:07.</p> <p>Submit your score to the Hall of Fame</p>  <p>6x6 Easy Kakurasu Puzzle ID: 5,358,536</p>
6	Size6x6	Hard	<pre> [0 , 0, 0, 0, 0, 0, 18] [0 , 0, 0, 0, 0, 0, 14] [0 , 0, 0, 0, 0, 0, 13] [0 , 0, 0, 0, 0, 0, 7], [0 , 0, 0, 0, 0, 0, 6], [0 , 0, 0, 0, 0, 0,10], [17, 5, 1, 7,17,10, 0]] </pre>	<p>Congratulations! You have solved the puzzle in 01:42.35</p> <p>Submit your score to the Hall of Fame</p>  <p>6x6 Hard Kakurasu Puzzle ID: 7,917,066</p>
7	Size7x7	Easy	<pre> [0 , 0, 0, 0, 0, 0, 0, 15], [0 , 0, 0, 0, 0, 0, 0, 13], [0 , 0, 0, 0, 0, 0, 0, 17], [0 , 0, 0, 0, 0, 0, 0, 6], [0 , 0, 0, 0, 0, 0, 0,7], [0 , 0, 0, 0, 0, 0, 0,15], [0 , 0, 0, 0, 0, 0, 0,13], [6 , 1,8, 5,6,23, 16, 0]] </pre>	<p>Congratulations! You have solved the puzzle in 01:17</p> <p>Submit your score to the Hall of Fame</p>  <p>7x7 Easy Kakurasu Puzzle ID: 8,826,944</p>
8	Size7x7	Hard	<pre> [0 , 0, 0, 0, 0, 0, 0,24], [0 , 0, 0, 0, 0, 0, 0,22], [0 , 0, 0, 0, 0, 0, 0, 4], [0 , 0, 0, 0, 0, 0, 0, 14], [0 , 0, 0, 0, 0, 0, 0,19], [0 , 0, 0, 0, 0, 0, 0,21], [0 , 0, 0, 0, 0, 0, 0,20], [17 , 14,17,19,21,25,10,0]] </pre>	<p>Congratulations! You have solved the puzzle in 01:41.95</p> <p>Submit your score to the Hall of Fame</p>  <p>7x7 Hard Kakurasu Puzzle ID: 9,803,433</p>

9	Size8x8	Easy	<pre>[0 , 0, 0, 0, 0, 0, 0, 0, 15], [0 , 0, 0, 0, 0, 0, 0, 0, 2], [0 , 0, 0, 0, 0, 0, 0, 0, 7], [0 , 0, 0, 0, 0, 0, 0, 0, 7], [0 , 0, 0, 0, 0, 0, 0, 0, 3], [0 , 0, 0, 0, 0, 0, 0, 0, 19], [0 , 0, 0, 0, 0, 0, 0, 0, 13], [0 , 0, 0, 0, 0, 0, 0, 0, 18], [7 , 3, 14, 16, 14, 15, 7, 13, 0]]</pre>	<p><b>Congratulations! You have solved the puzzle in 01:46.4!</b></p> <p>Submit your score to the Hall of Fame</p>  <p>8x8 Easy Kakurasu Puzzle ID: 7,799,494</p>
10	Size8x8	Hard	<pre>[0 , 0, 0, 0, 0, 0, 0, 0, 14], [0 , 0, 0, 0, 0, 0, 0, 0, 11], [0 , 0, 0, 0, 0, 0, 0, 0, 12], [0 , 0, 0, 0, 0, 0, 0, 0, 16], [0 , 0, 0, 0, 0, 0, 0, 0, 33], [0 , 0, 0, 0, 0, 0, 0, 0, 18], [0 , 0, 0, 0, 0, 0, 0, 0, 18], [0 , 0, 0, 0, 0, 0, 0, 0, 24], [23, 21, 19, 31, 26, 25, 7, 19, 0]]</pre>	<p><b>Congratulations! You have solved the puzzle in 02:03.68</b></p> <p>Submit your score to the Hall of Fame</p>  <p>8x8 Hard Kakurasu Puzzle ID: 1,975,621</p>
11	Size9x9	Easy	<pre>[0 , 0, 0, 0, 0, 0, 0, 0, 0, 44], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 39], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 26], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 36], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 39], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 41], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 33], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 36], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 36], [29, 40, 42, 30, 34, 36, 42, 37, 33, 0]]</pre>	<p><b>Congratulations! You have solved the puzzle in 02:05.74</b></p> <p>Submit your score to the Hall of Fame</p>  <p>9x9 Easy Kakurasu Puzzle ID: 2,765,792</p>
12	Size9x9	Hard	<pre>[0 , 0, 0, 0, 0, 0, 0, 0, 0, 17], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 22], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 18], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 17], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 42], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 40], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 39], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 23], [0 , 0, 0, 0, 0, 0, 0, 0, 0, 10], [24, 27, 35, 29, 14, 24, 42, 23, 21, 0]]</pre>	<p><b>Congratulations! You have solved the puzzle in 01:59.16</b></p> <p>Submit your score to the Hall of Fame</p>  <p>9x9 Hard Kakurasu Puzzle ID: 4,258,681</p>

- Kết quả chạy:

Game	Độ khó	Đúng đáp án		Thời gian chạy		Bộ nhớ tối đa	
		DFS (Y/N)	Climb (Y/N)	DFS (giây)	Climb (giây)	DFS (bytes)	Climb (bytes)
Size4x4	Easy	Y	Y	0.004048	0.002508	29800	35242
Size4x4	Hard	Y	Y	0.005233	0.001998	29800	33834
Size5x5	Easy	Y	Y	0.005343	0.003504	30440	37884
Size5x5	Hard	Y	Y	0.004451	0.004009	29696	37820
Size6x6	Easy	Y	Y	0.306067	0.025599	45430	53742
Size6x6	Hard	Y	Y	0.267822	0.005998	42618	42364
Size7x7	Easy	Y	Y	32.540520	0.015932	53268	54462
Size7x7	Hard	Y	Y	3.493176	0.020508	52772	59238
Size8x8	Easy	Y	Y	1272.63135	0.015997	64444	67628
Size8x8	Hard	-	Y		0.075814	-	82078
Size9x9	Easy	-	Y		0.074890	-	98998
Size9x9	Hard	-	Y		0.064859	-	105342

Game	Độ khó	Độ vượt trội về thời gian chạy ( $T_{DFS}/T_{Climb}$ ) %	Độ vượt trội về bộ nhớ ( $M_{DFS}/M_{Climb}$ ) %
Size4x4	Easy	161%	85%
Size4x4	Hard	262%	88%
Size5x5	Easy	152%	80%
Size5x5	Hard	111%	79%
Size6x6	Easy	1196%	85%
Size6x6	Hard	4465%	101%
Size7x7	Easy	204246%	98%
Size7x7	Hard	17033%	89%
Size8x8	Easy	7955438%	95%
Size8x8	Hard	$+\infty$	-
Size9x9	Easy	$+\infty$	-
Size9x9	Hard	$+\infty$	-

### 3.6.3. Nhận xét và giải thích:

- **Nhận xét:**

+ Trong cùng một game, luôn luôn thuật toán leo đồi (Climb) sẽ cho thời gian chạy ra kết quả nhanh hơn rất nhiều so với thuật toán Blind Search (DFS), tuy nhiên về mặt bộ nhớ thuật toán leo đồi lại cần bộ nhớ nhiều hơn.

+ Về mặt thời gian chạy, giữa 2 thuật toán có độ chênh lệch rất rất lớn, thuật toán Climb luôn cho ra kết quả dưới 0.1 giây (với  $n \leq 9$ ), trong khi đó từ  $n=8$  là thuật toán Blind Search (DFS) đã tốn thời gian rất lớn, và với  $n \geq 9$  thì thời gian là vô cùng nhiều ( $>3h$ ).

+ Về mặt bộ nhớ, thuật toán leo đồi cần nhiều hơn khoảng 5-20% so với thuật toán Blind Search (DFS).

- **Giải thích:**

+ Gọi  $x_i$  là số tổ hợp các số, mà tổng các số trong mỗi tổ hợp sẽ có tổng bằng target của hàng i, gọi n là kích thước của game  $n \times n$ . Ta có

+ Bài toán có maximum depth bằng n khi tất cả n hàng (cột) đã chọn ô và solution depth bằng n (khi tất cả n hàng/ cột được điền đúng).

+ Bài toán có độ phức tạp thuật toán là  $x_1.x_2.x_3 \dots x_n$

+ Bài toán có độ phức tạp không gian thay đổi dựa kích thước n của game và số tổ hợp của từng hàng, cụ thể ví dụ tại hàng i có  $x_i$  tổ hợp có khả năng là đáp án, thì ta cần lưu danh sách có  $x_i$  tổ hợp này đến khi bài toán được giải hoặc các nhánh bên dưới nó không dẫn đến trạng thái mục tiêu.

Xét tổng quát, bộ nhớ tối đa mà chương trình cần sử dụng là  $(n.A + b)$  trong đó n kích thước của game, A là kích thước bộ nhớ cho một ma trận biểu diễn game; b là các kích thước các biến phụ, các biến lưu trữ danh sách các tổ hợp, các danh sách khác (nếu có) ở từng độ sâu mà nhánh phía dưới chưa được xét hết.

⇒ **Về mặt thời gian:** Hàm chạy thuật toán leo đồi được tối ưu để nhanh chóng tìm được lời giải hơn thông qua hàm lượng giá, giúp chọn được các hàng/cột có ít tổ hợp có thể xảy ra hơn, đồng thời với việc bổ sung thêm xem xét các ô đã được chọn trước đó của hàng/cột, ta đã loại đi rất nhiều tổ hợp thừa. Do đó càng đi về sau, sẽ càng có ít nhánh phụ => nhanh chóng phát hiện hướng đi sai/ dẫn đến kết quả hơn so với thuật toán vét cạn DFS => tốc độ chương trình chạy bằng thuật toán leo đồi chạy nhanh hơn so với Blind Search DFS, đặc biệt khi kích thước game tăng lên hoặc khi giá trị target nằm trong khoảng trung bình (là khoảng giá trị mà target có thể tạo ra số tổ hợp khả thi nhiều nhất).

⇒ **Về mặt bộ nhớ:** cả 2 hàm đều chiếm dụng không gian bộ nhớ tối đa là  $(9.x.A+b)$ ; tuy cùng các thông số về x, và A; tuy nhiên do chạy thuật toán leo đồi sẽ cần thêm các biến phụ, nhiều hàm phụ hơn, đặc biệt là lưu lại các danh sách ori\_mark, ori\_pick, valid\_list ở mỗi node cha, do đó thuật toán leo đồi sẽ chiếm không gian bộ nhớ lớn hơn so với thuật toán Blind search DFS, đây là mức đánh đổi khá nhỏ khi so với hiệu quả về mặt thời gian.

#### **4. TÀI LIỆU THAM KHẢO**

<https://www.geeksforgeeks.org/monitoring-memory-usage-of-a-running-python-program/>

<https://coderzcolumn.com/tutorials/python/tracemalloc-how-to-trace-memory-usage-in-python-code>

<https://docs.python.org/3/library/tracemalloc.html>