

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



## NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

---

### TÌM KIẾM

---

Giảng viên: Vương Bá Thịnh  
Lớp: L01  
Nhóm: L01  
Sinh viên: Huỳnh Tấn Đạt - 1913026  
Phan Anh Tú - 1915822  
Lâm Thiện Toàn - 1915540

Thành phố Hồ Chí Minh, tháng 4/2022



## Mục lục

<b>1</b>	<b>Lựa chọn bài toán</b>	<b>2</b>
1.1	Mô tả bài toán . . . . .	2
1.1.1	Sudoku . . . . .	2
1.1.2	Shikaku . . . . .	2
<b>2</b>	<b>Hiện thực bài toán</b>	<b>2</b>
2.1	Sudoku . . . . .	2
2.1.1	Breath First Search . . . . .	2
2.1.2	Heuristic Search - Genetic Algorithm . . . . .	5
2.2	Shikaku: . . . . .	9
2.2.1	DFS (Blind Search): . . . . .	9
2.2.2	Heuristic Search - Genetic Algorithm . . . . .	13
<b>3</b>	<b>Kết quả</b>	<b>18</b>
3.1	Sudoku . . . . .	18
3.1.1	Breath First Search . . . . .	18
3.1.2	Heuristic Search - Genetic Algorithm . . . . .	23
3.2	Shikaku . . . . .	27
3.2.1	Depth First Search . . . . .	27
3.2.2	Heuristic Search - Genetic Algorithm . . . . .	36
	<b>Tài liệu tham khảo</b>	<b>39</b>

## 1 Lựa chọn bài toán

Về các bài toán có thể tham khảo qua link: <https://www.puzzle-pipes.com/>.

Hai bài toán nhóm lựa chọn để hiện thực trong Bài tập lớn này là:

- Sudoku (<https://www.puzzle-sudoku.com/>)
- Shikaku (<https://www.puzzle-shikaku.com/>)

### 1.1 Mô tả bài toán

#### 1.1.1 Sudoku

**State:** 1 ma trận dạng  $M \times M$ .

**Initial State:** Ma trận được khởi tạo với kích thước  $M \times M$ . Với một số vị trí đã được điền sẵn.

**Rules:**

- Điền các con số theo kích thước ma trận. Ví dụ  $9 \times 9$ , các chữ số có thể điền được từ 0 đến 9.
- Các con số theo hàng, theo cột và theo block ( $3 \times 3$ ) không được trùng nhau.

**Goal State:** Ma trận đã cho được điền đầy đủ bởi các con số và thỏa mãn luật chơi ở trên.

#### 1.1.2 Shikaku

**State:** 1 ma trận dạng  $N \times M$ , với giá trị tại điểm  $(x, y)$  là index của mảng hình vuông (hoặc chữ nhật) chứa nó hoặc là -1 nếu chưa thuộc mảng nào.

**Initial State:** ma trận dạng  $N \times N$ , các điểm  $(x, y)$  tương ứng với các ô chứa giá trị có giá trị thì 0 đến M.

**Rules:** Chia ma trận thành những mảng hình vuông và những hình chữ nhật sao cho mỗi mảng đó chỉ chứa một chữ số, và chữ số đó thể hiện diện tích của mảng đó.

**Goal State:** tất cả các điểm được đánh index và các vùng index có diện tích bằng với giá trị điểm thể hiện diện tích mà nó chứa.

Các giải thuật lần lượt được hiện thực bằng phương pháp Breath First Search, Depth First Search dựa trên ngôn ngữ Python Version 3. Ở phần 2, sẽ nói về cách hiện thực bài toán dựa trên từng phương pháp. Ở phần 3, sẽ giải thích về thời gian chạy và bộ nhớ của một số input.

## 2 Hiện thực bài toán

### 2.1 Sudoku

#### 2.1.1 Breath First Search

Hướng tiếp cận trong phần này tập trung vào cách giải quyết bài toán bằng Breath First Search (duyet theo tầng).

Hiện thực code được chia thành các module nhỏ được trình bày dưới đây.

#### Input

Input trong bài toán này để thể hiện bằng mảng hai chiều. Các số được cho trước sẽ điền vào

mảng với các giá trị nguyên dương và thỏa điều kiện mảng. Các chỗ trống chưa điền sẽ mang giá trị 0.

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

### **BFS\_solve**

Hàm này được gọi như entry vào của bài toán. Tham số của hàm này là một mảng hai chiều được khởi tạo ban đầu.

### **Class Problem**

Bước đầu tiên sau khi đi vào hàm **BFS\_solve**, ta sẽ khởi tạo một đối tượng có kiểu **Problem**, với hàm khởi tạo được thể hiện như sau:

```
class Problem(object):
```

```
    def __init__(self, initial):
        self.initial = initial
        self.size = len(initial) # Size of a grid
        self.height = int(self.size/3) # Size of a quadrant
```

Initial là mảng hai chiều, size là kích thước của mảng, height là kích thước của block.

Sau khởi tạo đối tượng, ta sẽ gọi hàm **BFS**, nhận vào một đối tượng kiểu **Problem**.

Ở đây, ta thực hiện giải thuật chính bằng cách khởi tạo các đối tượng **Node** với mục đích các **Node** sẽ nằm trong không gian trạng thái khi ta sử dụng hướng giải quyết.

Ban đầu, ta sẽ khởi tạo **Node** và kiểm tra xem trạng thái này đã hợp lệ hay chưa. Nếu trạng thái là hợp lệ thì chúng ta sẽ trả về trạng thái này.

Khởi tạo **Queue**, và đưa trạng thái khởi đầu vào queue nếu trạng thái này chưa hợp lệ. Với mỗi node ở đầu hàng đợi, chúng ta sẽ kiểm tra xem node này là hợp lệ hay chưa. Nếu chưa thì sẽ đẩy các node con vào trong hàng đợi. Cụ thể hàm **expand** được hiện thực như sau:

```
class Node:
```

```
    def __init__(self, state, action=None):
        self.state = state
        self.action = action

    # Use each action to create a new board state
    def expand(self, problem):
        return [self.child_node(problem, action)
                for action in problem.actions(self.state)]

    # Return node with new board state
    def child_node(self, problem, action):
        next = problem.result(self.state, action)
        return Node(next, action)
```

Theo như hiện thực ở trên, hàm `expand` sẽ gọi đến hàm `actions`. Hàm này sẽ nhận vào một `state` và sẽ sinh ra các `state` khác bằng cách trả về vị trí mà tại đó giá trị được điền vào là hợp lệ tính vào thời điểm đó. Hàm `actions` này sẽ lọc các con số theo hàng, theo cột, theo block và sau đó sẽ trả về những con số được lọc.

```
def result(self, state, action):  
  
    play = action[0]  
    row = action[1]  
    column = action[2]  
  
    # Add new valid value to board  
    new_state = copy.deepcopy(state)  
    new_state[row][column] = play  
  
    return new_state
```

Để tạo một `state` con từ `state` cha, ta cần trạng thái của cha và những `actions` có thể tạo ra. Khi đó hàm `child_node` sẽ hỗ trợ việc này. Cứ liên tục tạo ra các `state` con là đối tượng `Node` cho đến khi tìm thấy kết quả.

#### Hàm tạo ra một `state` với một `action`

```
def actions(self, state):  
    number_set = range(1, self.size+1) # Defines set of valid numbers  
    # that can be placed on board  
    in_column = [] # List of valid values in spot's column  
    in_block = [] # List of valid values in spot's quadrant  
  
    row, column = self.get_spot(self.size, state) # Get first empty  
    # spot on board  
  
    # Filter valid values based on row  
    in_row = [number for number in state[row] if (number != 0)]  
    options = self.filter_values(number_set, in_row)  
  
    # Filter valid values based on column  
    for column_index in range(self.size):  
        if state[column_index][column] != 0:  
            in_column.append(state[column_index][column])  
    options = self.filter_values(options, in_column)  
  
    # Filter with valid values based on quadrant  
    row_start = int(row/self.height)*self.height  
    column_start = int(column/3)*3  
  
    for block_row in range(0, self.height):  
        for block_column in range(0, 3):  
            in_block.append(state[row_start + block_row][column_start  
                                + block_column])  
    options = self.filter_values(options, in_block)
```

```
for number in options:
    yield number, row, column
```

### 2.1.2 Heuristic Search - Genetic Algorithm

Hướng tiếp cận trong phần này tập trung vào cách giải quyết bài toán bằng Heuristic Search - Genetic Algorithm kết hợp với phương pháp theo góc nhìn trực quan của con người.

#### Các bước di chuyển theo góc nhìn của con người

- Di chuyển theo hàng: chúng ta sẽ áp dụng bước di chuyển này lần lượt cho các hàng, cố gắng điền những số bị thiếu ở trong hàng. Chúng ta sẽ xét lần lượt các số từ 1-9. Với mỗi số chúng ta sẽ kiểm tra xem nếu chỉ có duy nhất 1 ô có thể điền số đó thì chúng ta sẽ điền số đang xét vào ô đó. Chúng ta áp dụng tương tự cho các cột và khối.

```
# def row_move(self):
def a(self):
    for n in range(9):
        for nums in range(1,10):
            fin = False
            cell = None
            if nums in self._row(n): continue
            for idx, cells in enumerate(self._row(n)):
                if cells == 0 and not fin:
                    if nums not in self._col(idx)
                    and nums not in self._block(n, idx):
                        if cell is None:
                            cell = idx
                        else:
                            fin = True

            if not fin and cell is not None:
                self.__matrix__[n][cell] = nums
```

- Di chuyển theo hàng - 3: chúng ta sẽ xét các hàng, nếu hàng đó có 3 ô chưa điền thì tương ứng sẽ có 3 số chưa điền. Nếu trong 3 số đó có 2 số không thể điền vào 1 ô thì chúng ta điền số còn lại vào ô đó. Chúng ta áp dụng tương tự cho các cột và khối.

```
# def row_move_3(self):
def d(self):
    for r in range(9):
        list_nums = []
        for nums in range(1, 10):
            if nums not in self._row(r): list_nums.append(nums)

        if len(list_nums) != 3: continue

        empty_cells = {}
        for idx, cells in enumerate(self._row(r)):
```

```
if cells == 0: empty_cells[idx] = []

for c in empty_cells.keys():
    for nums in list_nums:
        if nums in self._block(r,c) or nums in self._col(c):
            empty_cells[c] = empty_cells.get(c) + [nums]

for k, val in empty_cells.items():
    if len(val) == 2:
        _x = None
        for x in list_nums:
            if x not in val:
                _x = x
        self.__matrix__[r][k] = _x
```

- Di chuyển ngẫu nhiên theo hàng: chúng ta sẽ xét các hàng, nếu như hàng có 2 ô còn trống và cả 2 số chưa điền đều có thể điền vào 2 ô đó thì chúng ta sẽ điền ngẫu nhiên 2 số đó vào 2 ô. Chúng ta áp dụng tương tự cho các cột và khối. Với cách làm này thì chúng ta có thể vô tình điền sai dẫn đến không thể ra đáp án. Do đó đây là lý do chúng ta nên hiện thực giải thuật Heuristic - Genetic Algorithm

```
# def try_row_move(self):
def g(self):
    for r in range(9):
        dict_cell = {}
        for num in range(1,10):
            list_cell = []
            if num not in self._row(r):
                for c, cells in enumerate(self._row(r)):
                    if cells == 0 and num not in self._block(r,c)
                    and num not in self._col(c):
                        list_cell.append(c)
            dict_cell[num] = list_cell
        for key, val in dict_cell.items():
            if len(val) == 2:
                _c = val[random.randrange(2)]
                self.__matrix__[r][_c] = key
            break
```

### Các tiền xử lý trước khi hiện thực giải thuật Heuristic

- Bước 1: Khởi tạo đối tượng SudokuPuzzle trong đó có 2 field matrix and backup trong đó matrix sẽ lưu giá trị hiện tại của puzzle và backup sẽ dùng trong trường hợp ta muốn reset lại giá trị của puzzle sau khi thực hiện thuật toán Heuristic. Bên trong preprocess sẽ thực hiện các bước di chuyển theo hàng (cột, khối) và di chuyển theo hàng-3 (cột, khối) cho đến khi trạng thái của Puzzle không đổi. Sau đó chúng ta sẽ thực hiện giải thuật Heuristic - Genetic Algorithm.

```
class SudokuPuzzles():
    def __init__(self, array):
```

```
self.__matrix__ = copy.deepcopy(array)
self.__backup__ = copy.deepcopy(array)
self.preprocess()

def preprocess(self):
    process = 'abcdef'
    while True:
        prev_fitness = self.fitness()
        for i in process:
            exec(f"self.{i}()")
        new_fitness = self.fitness()
        if new_fitness == prev_fitness:
            break
    self.__backup__ = copy.deepcopy(self.__matrix__)
```

- Bước 2: Khởi tạo population trong đó population sẽ là list chuỗi string chứa các function cho việc thực thi theo góc nhìn của con người (những function đó sẽ đc đặt tên là a,b,c,d,e,f,g,h,i). Vì vậy nếu chuỗi là 'ad' thì sẽ thực hiện di chuyển theo hàng rồi sau đó là di chuyển theo hàng-3. Và khởi tạo hàm fitness cho giải thuật bằng công thức là tổng của các ô đã được đánh số. Vì vậy hàm mục tiêu của chúng ta sẽ cố gắng đạt được 81 (giá trị cho puzzle 9x9)

```
initial_population = [
    "".join([chr(random.randrange(97, 106))\
        for i in range(1,50)])\
        for i in range(10)
]
```

```
class SudokuPuzzles():
    def fitness(self):
        sum = 0
        for i in range(9):
            for j in range(9):
                if self.__matrix__[i][j] == 0:
                    sum += 1
        return sum
```

```
def run_and_fit(obj, cmds):
    for i in cmds:
        exec(f"obj.{i}()")
    fit = obj.fitness()
    obj.reset()
    return 9*9 - fit
```

- Bước 3: Chạy và đánh giá lại fitness với population hiện tại.

```
list_fitness = []
for gen in initial_population:
    list_fitness.append([gen, run_and_fit(puzzle, gen)])
list_fitness = sorted(list_fitness, key=lambda x: x[1],
```



reverse=True)

- Bước 4: Sử dụng giải thuật tournament selection để chọn ra các cha-mẹ mới cho việc tạo ra lại population mới. Chọn một cách ngẫu nhiên các giá trị theo xác suất ưu tiên các mã gen có giá trị fitness gần 1 nhất.

```
def tournament_selection(list_fitness):  
    sum = 0  
    for pop in list_fitness:  
        sum += pop[1]  
    length = len(list_fitness)  
    lst = []  
    for i in range(length):  
        ran = random.randrange(1, sum+1)  
        acc_sum = 0  
        for pop in list_fitness:  
            acc_sum += pop[1]  
            if acc_sum >= ran:  
                lst.append(pop)  
                break  
    lst = sorted(lst, key=lambda x: x[1], reverse=True)  
    return lst
```

- Bước 5: Sử dụng giải thuật Crossover để sinh ra các con từ các cha-mẹ để tạo ra 1 population mới. Trong đó, xác suất crossover là 0.3 và ưu tiên tạo ra những con có giá trị fitness lớn hơn hoặc bằng so với cha-mẹ

```
def crossover(list_fitness):  
    if random.randrange(10) > 6:  
        for i in range(0, len(list_fitness)):  
            prev_fit = list_fitness[i][1]  
            count = 0  
            while count < 100:  
                count += 1  
                crossover_point = random.randrange(len(list_fitness[i]))  
                crossover_with = random.randrange(len(list_fitness))  
                crossover_point1 = random.randrange(\  
                    len(list_fitness[crossover_with]))  
                list_fitness[i][0] = list_fitness[i][0][:crossover_point]\  
                    + list_fitness[crossover_with][0][crossover_point1:]  
                new_fit = run_and_fit(puzzle, list_fitness[i][0])  
                if new_fit >= prev_fit:  
                    list_fitness[i][1] = new_fit  
                    break  
    return list_fitness
```

- Bước 6: Sử dụng mutation để đột biến các con mới trong population với xác suất đột biến là 0.6 và chọn những con có giá trị fitness lớn hơn hoặc bằng so với cha-mẹ.

```
def mutation(list_fitness):
```

```
if random.randrange(10) > 3:
    for i in range(0, len(list_fitness)):
        prev_fit = list_fitness[i][1]
        count = 0
        while count < 100:
            count += 1
            mutation_point = random.randrange(len(list_fitness[i][0]))
            chosen_move = chr(random.randrange(97, 106))
            list_fitness[i][0] = list_fitness[i][0][:mutation_point]\
                + chosen_move + list_fitness[i][0][mutation_point+1:]
            new_fit = run_and_fit(puzzle, list_fitness[i][0])
            if new_fit >= prev_fit:
                list_fitness[i][1] = new_fit
                break
    return list_fitness
```

- Bước 7: Kiểm tra trong list fitness đã có giá trị 9\*9 là đáp án bài toán chưa, nếu chưa có ta quay lại bước 4.

```
def check_solve_puzzle(list_fitness):
    for e in list_fitness:
        if e[1] == 9*9:
            return e
    return False

while True:
    list_fitness = tournament_selection(list_fitness)
    list_fitness = crossover(list_fitness)
    if check_solve_puzzle(list_fitness): break
    list_fitness = mutation(list_fitness)
    if check_solve_puzzle(list_fitness): break
print(check_solve_puzzle(list_fitness))
```

## 2.2 Shikaku:

### 2.2.1 DFS (Blind Search):

- Problem: Ma trận NxN, có M ô chữ các số thể hiện diện tích cần bao phủ ( $M < N$ ).
- Solution:
  - Bước 1 (Khởi tạo):
    - \* Mảng chữ giá trị (x,y,value) với x, y là tọa độ điểm thể hiện diện tích, value là giá trị diện tích. Đặt tên cho mảng này là locationData.
    - \* Khởi tạo initial state. Lưu vào state này vào stack.
    - \* Từ từng giá trị của locationData, ta thực hiện các tìm các giá trị của chiều dài và chiều rộng mà mảng có thể có. Đặt tên mảng này là locationDataFactor.
  - Bước 2:

- \* Từ initial state, locationData, locationDataFactor, lần lượt điền giá trị của tất cả các ô theo rule tuy nhiên có thể điền vào các ô đã đánh index. Mục tiêu của việc này để tạo ra state có trạng thái gần giống với goal state. Gọi state này là lastCells.

– Bước 3:

- \* Thực hiện điền lần lượt đánh index vào các vùng theo dựa trên state, locationData, locationDataFactor. Check các ô sau mỗi lần đánh index để xem giá trị các ô này có ở vị trí đúng so với lastCell hay không. Việc so sánh này để giảm số lượt state vào trong stack, giảm thời gian duyệt DFS.
- \* Thực hiện việc duyệt DFS bằng cách lấy các trạng thái ở đầu stack và thực hiện lại bước trên cho đến khi tìm được goal state hoặc đánh index tất cả các điểm thể hiện diện tích.

- Implement:

- Khởi tạo locationData (danh sách chứa tọa độ và diện tích): list((x,y,value)).
- Khởi tạo locationDataFactor (danh sách chứa giá trị có thể của chiều dài và rộng của mảng): list(list(x,y)).
- Khởi tạo initial state.

```
def readPuzzle(inputFilename):
    global rows, cols, puzzle, locationData
    locationData = []
    with open(inputFilename, "r") as inputFile:
        rows = int(inputFile.readline())
        cols = int(inputFile.readline())
        puzzle = [cols * [" "] for i in range(rows)]
        for row, line in enumerate(inputFile):
            for col, symbol in enumerate(line.split()):
                if symbol == "-":
                    puzzle[row][col] = -1
                else:
                    puzzle[row][col] = int(symbol)
                    locationData.append((row, col, int(symbol)))

def factors():
    global locationData, locationDataFactor
    for anchor in locationData:
        value = anchor[2]
        current = []
        bound = round(sqrt(value))+1
        for i in range(1, int(bound)):
            if value % i == 0:
                if i == value//i:
                    current.append(i)
                else:
                    current.append(i)
                    current.append(value//i)
        locationDataFactor.append(current)
```

Vì các mảng có thể là hình vuông hoặc chữ nhật nên cạnh lớn có thể có giá trị bằng căn bậc hai của diện tích.

- **Tìm state lastCells:**

```
def initialization():
    global state, locationDataFactor, count, rows, cols
    global lastCells
    locationDataFactor = []
    factors()

    count = [0]*len(locationData)

    state = [[-1 for c in range(cols)] for r in range(rows)]
    for i in range(len(locationData)):
        state[locationData[i][0]][locationData[i][1]] = i

    for z in range(len(locationData)):
        eRow = locationData[z][0]
        eCol = locationData[z][1]
        eValue = locationData[z][2]
        facList = locationDataFactor[z]
        for fac in facList:
            for i in range(eValue//fac):
                for j in range(fac):
                    if checkValid(state, eRow-j, eRow+fac-1-j,
                                eCol+i-eValue//fac+1, eCol+i):
                        setValue(state, eRow-j, eRow+fac-1-j,
                                eCol+i-eValue//fac+1, eCol+i, z)
    lastCells = [[] for i in range(len(locationData))]
    for row in range(rows):
        for col in range(cols):
            value = state[row][col]
            lastCells[value].append([row, col])

    state = [[-1 for c in range(cols)] for r in range(rows)]
    for i in range(len(locationData)):
        state[locationData[i][0]][locationData[i][1]] = i
```

Ở for loop tìm state lastCells, eRow là hoành độ của điểm giá trị, eCol là tung độ của điểm giá trị, eValue là giá trị diện tích và facList là danh sách chứa các giá trị có thể của chiều dài và rộng của mảng.

- **DFS:**

```
if __name__ == "__main__":
    fileNames = sorted(glob.glob("input/*.txt"))
    for fileName in fileNames:
        readPuzzle(fileName)
        print(fileName)
    startTime = time.time()
```

```
initialization()
print(locationDataFactor)
DFS(0)
printGrid(state)
endTime = time.time()
if verifySolution():
    print("solved")
else:
    print("not_solved")
print(endTime - startTime)
print("")
```

Lấy state đầu tiên ở đầu state, kiểm tra state này có phải là solution hay không, Nếu có thì in ra kết quả và kết thúc thuật toán.

```
def verifySolution():
    global rows, cols, puzzle, locationData, state
    for i, (row, col, val) in enumerate(locationData):

        if state[row][col] != i:
            return False

        eWhere = [(r, c) for r in range(rows)
                    for c in range(cols) if state[r][c] == i]
        eNum = len(eWhere)
        if eNum != val:
            return False

        left = min(eWhere, key=lambda x: x[0])[0]
        right = max(eWhere, key=lambda x: x[0])[0]
        top = min(eWhere, key=lambda x: x[1])[1]
        bottom = max(eWhere, key=lambda x: x[1])[1]
        area = (right-left+1) * (bottom-top+1)
        if area != eNum:
            return False
    return True
```

Các bước kiểm tra state:

- Giá trị tại vị trí điểm thể hiện giá trị bằng với giá trị tại điểm đó ở initial state.
- Số lượng các ô đã đánh index bằng với giá trị diện tích ứng với index đó.
- Chiều dài \* chiều rộng của hình vuông ( hoặc chữ nhật) bằng diện tích ứng với index đó (Đảm bảo đây là hình vuông hoặc chữ nhật)

Nếu không, thực hiện DFS với state này

```
def DFS(nextIndex):
    global rows, cols, puzzle, locationData, state
    global count, locationDataFactor, lastCells

    if nextIndex > len(locationData)-1:
```

```
    return True

eRow = locationData[nextIndex][0]
eCol = locationData[nextIndex][1]
eValue = locationData[nextIndex][2]

while count[nextIndex] < len(locationDataFactor[nextIndex]):
    facList = locationDataFactor[nextIndex]
    fac = facList[count[nextIndex]]
    for i in range(eValue//fac):
        for j in range(fac):
            if checkValid(state, eRow-j, eRow+fac-1-j,
                           eCol+i-eValue//fac+1, eCol+i):
                if checkValidWithValue(state, eRow-j,
                                         eRow+fac-1-j, eCol+i-eValue//fac+1, eCol+i, -1, nextIndex):
                    setValue(state, eRow-j, eRow+fac-1-j,
                              eCol+i-eValue//fac+1, eCol+i, nextIndex)
                    notCover = False
                    for z in range(len(lastCells[nextIndex])):
                        r = lastCells[nextIndex][z][0]
                        c = lastCells[nextIndex][z][1]
                        if state[r][c] == -1:
                            notCover = True
                            break
                    if notCover == False:
                        if DFS(nextIndex+1) == True:
                            return True

                setValue(state, eRow-j, eRow+fac-1-j,
                          eCol+i-eValue//fac+1, eCol+i, -1)

                state[locationData[nextIndex][0]]
                [locationData[nextIndex][1]] = nextIndex
        count[nextIndex] += 1

if nextIndex > 0:
    count[nextIndex] = 0
```

Nếu state mới sau khi ghi thực hiện theo rule có các nơi đánh index trùng với lastCells chứng tỏ state này có trạng thái gần giống lastCells, lasCells cũng có trạng thái gần giống goal state => Lưu lại state này và tiếp tục DFS

### 2.2.2 Heuristic Search - Genetic Algorithm

Hướng tiếp cận trong phần này tập trung vào cách giải quyết bài toán bằng Heuristic Search - Genetic Algorithm. Vài thứ chúng ta cần quan tâm là cách thể hiện shikaku puzzle, tạo và lưu cái ô có thể của ô đã được đánh số trước, cách giải quyết kết hợp với heuristic search - genetic algorithm.

#### Các bước xử lý trước khi áp dụng giải thuật Heuristic - Genetic Algorithm

- Bước 1: Khởi tạo puzzle với input là 1 mảng 2 chiều:

```
class ShikakuPuzzle():
    def __init__(self, array):
        self.__matrix__ = copy.deepcopy(array)
        self.__locationData__ = []
        self.__locationDataFactor__ = []
        length_array = len(array)
        self.__partitions__ = []
        self.__population__ = []
        self.__puzzle__ = [
            length_array * [" "] for i in range(length_array)
        ]
        for n_row, row in enumerate(array):
            for n_col, cell in enumerate(row):
                if cell == 0:
                    self.__puzzle__[n_row][n_col] = -1
                else:
                    self.__locationData__.append((n_row, n_col, cell))
                    self.__partitions__.append("")
                    self.__puzzle__[n_row][n_col]
                        = len(self.__locationData__)-1
        self.__backup__ = copy.deepcopy(self.__puzzle__)
        self.__max_fit__ = [0,0]

        self.__factors()
        self.__generate_partitions()
        self.__preprocess()
        self.__gen_population()
```

- Bước 2: Khởi tạo list factors chứa giá trị cạnh có thể của ô đã được đánh số.

```
def __factors(self):
    for data in self.__locationData__:
        value = data[2]
        current = []
        bound = round(sqrt(value))+1
        for i in range(1, int(bound)):
            if value % i == 0:
                if i == value//i:
                    current.insert(0, i)
                else:
                    current.insert(0, i)
                    current.insert(0, value//i)
        self.__locationDataFactor__.append(current)
```

- Bước 3: Từ list các cách có thể ta sẽ tạo ra list các ô hình vuông hoặc hình chữ nhật tương ứng cho có ô vuông được đánh số.

```
def __generate_partitions(self):
```

```
for n_row, row in enumerate(self.__puzzle__):
    for n_col, cell in enumerate(row):
        if cell == -1: continue
row = col = len(self.__puzzle__) - 1
for idx, (r_idx, c_idx, value) in enumerate(self.__locationData__):
    list_rect = []
    for width in self.__locationDataFactor__[idx]:
        # Get valued squared and its factor
        height = value // width
        for i in range(r_idx-width+1, r_idx+1):
            if i < 0 or i+width-1 > col: continue
            for j in range(c_idx-height+1, c_idx+1):
                if j < 0 or j+height-1 > row: continue
                is_valid = True
                for x in range(i, i+width):
                    for y in range(j, j+height):
                        if self.__puzzle__[x][y] != -1\
                            and self.__backup__[x][y] != idx:
                            is_valid = False
                if is_valid:
                    list_rect.append((i, j, width, height, idx))
self.__partitions__[idx] = list_rect
```

- Bước 4: Tiếp đến là bước tiền xử lý để điền các ô mà có số lượng hình vuông hoặc hình chữ nhật là 1.

```
def __preprocess(self):
    while True:
        for idx, part in enumerate(self.__partitions__):
            if len(part) == 1:
                self.__draw_partition(part[0])
                self.__partitions__[idx] = []

        self.__generate_partitions()
        should_break = True
        for part in self.__partitions__:
            if len(part) == 1: should_break = False
        if should_break: break
```

- Bước 5: Tạo population phụ vụ cho giải thuật heuristic - genetic algorithm.

```
def __gen_population(self):
    n_pop = 20
    for i in range(n_pop):
        chromos = []
        for part in self.__partitions__:
            if len(part) > 0:
                chromos.append(str(random.randrange(0, len(part))))
            else: chromos.append("-")
        self.__population__.append(("".join(chromos), 0))
```



### Áp dụng giải thuật Heuriti - Genetic Algorithm

- Bước 1: Từ population hiện tại, ta lần lượt thực hiện các mã gen mà mình đã sinh ra và lưu lại các kết quả fitness. Sau đó kiểm tra xem đã có kết quả tối ưu hay chưa.

```
def fitness_of(self, puzzle):
    n = len(puzzle)
    sum = 0
    for row in puzzle:
        for cell in row:
            if cell != -1: sum += 1
    return sum / n**2

def __run(self, chromos):
    puzzle = copy.deepcopy(self.__puzzle__)
    for idx, x in enumerate(chromos[0]):
        if x == "-": continue
        part = self.__partitions__[idx][int(x)]
        (row, col, width, height, idx) = part
        for i in range(row, row+width):
            for j in range(col, col+height):
                puzzle[i][j] = idx
    return [chromos[0], self.fitness_of(puzzle)]

def __check_solve_puzzle(self):
    for p in self.__population__:
        if p[1] == 1.0:
            return p
    return False

def execute(self):
    for idx, chromos in enumerate(self.__population__):
        rt = self.__run(chromos)
        self.__population__[idx] = rt
    if self.__check_solve_puzzle():
        return self.__check_solve_puzzle()
```

- Bước 2: Thực hiện giải thuật tournament selection để từ kết quả có được ở bước 1 và chọn một cách ngẫu nhiên các giá trị theo xác suất ưu tiên các mã gen có giá trị fitness gần 1 nhất.

```
def __tournament_selection(self):
    sum = 0
    for pop in self.__population__:
        sum += pop[1]
    new_pop = []
    for _ in range(len(self.__population__)):
        point = random.uniform(0, sum)
        cum_sum = 0
        for part in self.__population__:
            cum_sum += part[1]
```

```
        if cum_sum > point:
            new_pop.append(part)
            break
    self.__population__ = new_pop
```

- Bước 3: Thực hiện crossover, chọn ngẫu nhiên 2 cha-mẹ và lai tạo tại một vị trí ngẫu nhiên với xác suất crossover là 0.3 và sau khi crossover thực hiện chạy lại để đánh giá các kết quả mới và ưu tiên chọn các con có giá trị tốt hơn.

```
def __crossover(self):
    if random.randrange(10) > 6:
        new_pop = []
        random.shuffle(self.__population__)
        for i in range(len(self.__population__)//2):
            crossover_point = random.randrange(
                len(self.__population__[i]))
            crossover_with = random.randrange(
                len(self.__population__))
            c1 = self.__population__[i][0][:crossover_point]
            + self.__population__[crossover_with][0][crossover_point:]
            c2 = self.__population__[crossover_with][0][:crossover_point]
            + self.__population__[i][0][crossover_point:]
            c1, c2 = (self.__run([c1, 0]), self.__run([c2, 0]))
            if c1[1] >= self.__population__[i][1]
            or c2[1] >= self.__population__[crossover_with][1]:
                self.__population__[i] = c1
                self.__population__[crossover_with] = c2
```

- Bước 4: Thực hiện mutation, chọn một vị trí ngẫu nhiên trên đoạn mã gen và thực hiện thay đổi 1 gen trên đoạn mã gen đó với xác suất mutation là 0.7 và thực hiện đánh giá lại mã gen đã được mutation.

```
def __mutation(self):
    for i in range(len(self.__population__)):
        if random.randrange(10) > 2:
            mutation_point = random.randrange(
                len(self.__population__[i][0]))
            )
            if self.__population__[i][0][mutation_point] == "-":
                continue
            new_muta = random.randrange(
                len(self.__partitions__[mutation_point]))
            )
            c = self.__population__[i][0][:mutation_point]
            + str(new_muta)
            + self.__population__[i][0][mutation_point+1:]
            c = self.__run([c, 0])
            if c[1] >= self.__population__[i][1]:
                self.__population__[i] = c
```

- Bước 5: Đánh giá lại các kết quả thu được, nếu chưa có kết quả thoả mãn thì quay lại bước 2.

```
def __check_solve_puzzle(self):  
    for p in self.__population__:  
        if p[1] == 1.0:  
            return p  
    return False  
  
def execute(self):  
    for idx, chromos in enumerate(self.__population__):  
        rt = self.__run(chromos)  
        self.__population__[idx] = rt  
        if self.__check_solve_puzzle():  
            return self.__check_solve_puzzle()  
    while True:  
        self.__tournament_selection()  
        self.__crossover()  
        if self.__check_solve_puzzle():  
            return self.__check_solve_puzzle()  
        self.__mutation()  
        if self.__check_solve_puzzle():  
            return self.__check_solve_puzzle()
```

## 3 Kết quả

### 3.1 Sudoku

#### 3.1.1 Breath First Search

Input 1:

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

Hình 1: Kiểm tra trên đầu vào để

Output 1:

**Giải thích:** Đầu tiên khi nhận được initial state này, giải thuật sẽ kiểm tra xem có hợp lệ goal state hay không. Vì trong ma trận còn chứa những giá trị 0 nên trạng thái này không hợp lệ. Giải thuật bắt đầu đưa những Node con vào hàng đợi.

Xét giá trị tại hàng 0 cột 0. Sau khi lọc theo hàng 0, cột 0 và block đầu tiên, các số có thể

```
Testing on easy 9x9 grid...
Problem:
[0, 0, 7, 2, 8, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 5, 0, 6]
[4, 1, 3, 0, 0, 6, 0, 8, 0]
[7, 2, 0, 3, 9, 0, 0, 0, 0]
[3, 4, 0, 0, 0, 0, 8, 1, 0]
[6, 8, 0, 1, 0, 7, 0, 0, 2]
[0, 0, 0, 6, 7, 4, 0, 2, 3]
[0, 0, 0, 0, 0, 5, 7, 0, 0]
[1, 0, 6, 0, 2, 3, 0, 4, 0]

Solving with BFS...
Found solution
[5, 6, 7, 2, 8, 9, 4, 3, 1]
[8, 9, 2, 4, 3, 1, 5, 7, 6]
[4, 1, 3, 7, 5, 6, 2, 8, 9]
[7, 2, 1, 3, 9, 8, 6, 5, 4]
[3, 4, 9, 5, 6, 2, 8, 1, 7]
[6, 8, 5, 1, 4, 7, 3, 9, 2]
[9, 5, 8, 6, 7, 4, 1, 2, 3]
[2, 3, 4, 9, 1, 5, 7, 6, 8]
[1, 7, 6, 8, 2, 3, 9, 4, 5]
Elapsed time: 0.03799843788146973 seconds
```

Hình 2: Kết quả

thỏa mãn là 5,9. Thêm hai state này vào hàng đợi. Tiếp tục lấy state(5) ra check, con số thỏa mãn ở hàng 0, cột 1 là 6,9 và ta sẽ thêm hai state mới này vào hàng đợi. Chúng ta sẽ giữ một bản copy trước khi thêm để có thể nhiều số vào một vị trí.

Về thời gian của test-case này là 0.038 giây.

#### Input 2:

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

Hình 3: Kiểm tra trên đầu vào không hợp lệ

#### Output 2:

**Giải thích:** Vì block bắt đầu tại index (3,3) chứa hai số 5 nên test-case này là không hợp lệ. Sau khi check mọi trường hợp khả hữu mà không có trạng thái nào thỏa mãn thì sẽ in ra là **"No possible solutions"**.

Cách kiểm tra hợp lệ của giải thuật. Kiểm tra trên hàng, cột và block. Nếu tổng của từng phần này khác tổng của goal state thì trả về false.

Về thời gian của test-case này là 0.890 giây.

```
Testing on invalid 9x9 grid...
Problem:
[0, 0, 9, 0, 7, 0, 0, 0, 5]
[0, 0, 2, 1, 0, 0, 9, 0, 0]
[1, 0, 0, 0, 2, 8, 0, 0, 0]
[0, 7, 0, 0, 0, 5, 0, 0, 1]
[0, 0, 8, 5, 1, 0, 0, 0, 0]
[0, 5, 0, 0, 0, 0, 3, 0, 0]
[0, 0, 0, 0, 0, 3, 0, 0, 6]
[8, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 1, 0, 0, 0, 0, 0, 8, 7]

Solving with BFS...
No possible solutions
Elapsed time: 0.889751672744751 seconds
```

Hình 4: Kết quả

### Input 3:

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

Hình 5: Kiểm tra trên đầu vào đã thỏa mãn

### Output 3:

**Giải thích:** Vì initial state cũng là goal state. Giải thuật này sẽ kiểm tra hợp lệ trước khi đưa vào hàng đợi. Vì hợp lệ nên giải thuật sẽ trả về trạng thái này.

Về thời gian của test-case này là 0.0 giây. Chỉ liên quan đến kiểm tra hợp lệ nên thời gian nhanh hơn hai trường hợp trước.

### Bảng số liệu theo chiến lược BFS

STT	Input	Output	Time	Space
1	grid = [[0,0,9,0,7,0,0,0,5], [0,0,2,1,0,0,9,0,0], [1,0,0,0,2,8,0,0,0], [0,7,0,0,0,5,0,0,1], [0,0,8,5,1,0,0,0,0], [0,5,0,0,0,0,3,0,0], [0,0,0,0,0,3,0,0,6], [8,0,0,0,0,0,0,0,0], [2,1,0,0,0,0,0,8,7]]	No possible solutions	1.287 seconds	23.992 MiB

2	grid [[0,0,7,2,8,0,0,0,0], [0,0,0,0,0,5,0,6], [4,1,3,0,0,6,0,8,0], [7,2,0,3,9,0,0,0,0], [3,4,0,0,0,8,1,0], [6,8,0,1,0,7,0,0,2], [0,0,0,6,7,4,0,2,3], [0,0,0,0,5,7,0,0], [1,0,6,0,2,3,0,4,0]] =	[5, 6, 7, 2, 8, 9, 4, 3, 1] [8, 9, 2, 4, 3, 1, 5, 7, 6] [4, 1, 3, 7, 5, 6, 2, 8, 9] [7, 2, 1, 3, 9, 8, 6, 5, 4] [3, 4, 9, 5, 6, 2, 8, 1, 7] [6, 8, 5, 1, 4, 7, 3, 9, 2] [9, 5, 8, 6, 7, 4, 1, 2, 3] [2, 3, 4, 9, 1, 5, 7, 6, 8] [1, 7, 6, 8, 2, 3, 9, 4, 5]	0.052 seconds	20.609 MiB
3	grid [[9,7,4,2,3,6,1,5,8], [6,3,8,5,9,1,7,4,2], [1,2,5,4,8,7,9,3,6], [3,1,6,7,5,4,2,8,9], [7,4,2,9,1,8,5,6,3], [5,8,9,3,6,2,4,1,7], [8,6,7,1,2,5,3,9,4], [2,5,3,6,4,9,8,7,1], [4,9,1,8,7,3,6,2,5]] =	[9, 7, 4, 2, 3, 6, 1, 5, 8] [6, 3, 8, 5, 9, 1, 7, 4, 2] [1, 2, 5, 4, 8, 7, 9, 3, 6] [3, 1, 6, 7, 5, 4, 2, 8, 9] [7, 4, 2, 9, 1, 8, 5, 6, 3] [5, 8, 9, 3, 6, 2, 4, 1, 7] [8, 6, 7, 1, 2, 5, 3, 9, 4] [2, 5, 3, 6, 4, 9, 8, 7, 1] [4, 9, 1, 8, 7, 3, 6, 2, 5]	0.0 seconds	20.453 MiB
4	grid [[0,0,0,0,5,0,9,7,6], [8,0,5,1,9,0,0,3,0], [3,7,0,0,4,0,0,8,0], [0,8,0,0,0,0,0,0,9], [0,2,0,0,0,0,4,0,7], [0,9,0,0,2,6,0,1,5], [0,0,0,0,8,1,6,0,0], [9,0,0,3,0,0,0,0,0], [2,0,0,4,0,9,0,0,0]] =	[1, 4, 2, 8, 5, 3, 9, 7, 6] [8, 6, 5, 1, 9, 7, 2, 3, 4] [3, 7, 9, 6, 4, 2, 5, 8, 1] [6, 8, 7, 5, 1, 4, 3, 2, 9] [5, 2, 1, 9, 3, 8, 4, 6, 7] [4, 9, 3, 7, 2, 6, 8, 1, 5] [7, 5, 4, 2, 8, 1, 6, 9, 3] [9, 1, 8, 3, 6, 5, 7, 4, 2] [2, 3, 6, 4, 7, 9, 1, 5, 8]	0.0475 sec- onds	20.621 MiB
5	grid [[0,3,9,0,0,0,1,2,0], [0,0,0,9,0,7,0,0,0], [8,0,0,4,0,1,0,0,6], [0,4,2,0,0,0,7,9,0], [0,0,0,0,0,0,0,0,0], [0,9,1,0,0,0,5,4,0], [5,0,0,1,0,9,0,0,3], [0,0,0,8,0,5,0,0,0], [0,1,4,0,0,0,8,7,0]] =	[4, 3, 9, 6, 5, 8, 1, 2, 7] [1, 5, 6, 9, 2, 7, 3, 8, 4] [8, 2, 7, 4, 3, 1, 9, 5, 6] [3, 4, 2, 5, 1, 6, 7, 9, 8] [7, 8, 5, 2, 9, 4, 6, 3, 1] [6, 9, 1, 7, 8, 3, 5, 4, 2] [5, 7, 8, 1, 4, 9, 2, 6, 3] [2, 6, 3, 8, 7, 5, 4, 1, 9] [9, 1, 4, 3, 6, 2, 8, 7, 5]	0.954 seconds	24.816 MiB

6	grid [[0,9,0,3,0,0,0,0,1], [0,0,0,0,8,0,0,4,6], [0,0,0,0,0,0,8,0,0], [4,0,5,0,6,0,0,3,0], [0,0,3,2,7,5,6,0,0], [0,6,0,0,1,0,9,0,4], [0,0,1,0,0,0,0,0,0], [5,8,0,0,2,0,0,0,0], [2,0,0,0,0,7,0,6,0]]	=	No possible solutions	0.924 seconds	22.352 MiB
7	grid [[0,3,0,0,0,1,5,0,0], [0,0,0,5,0,0,0,8,4], [0,0,5,0,0,7,0,6,0], [0,0,0,0,0,0,0,0,0], [0,8,0,2,0,0,0,7,0], [0,0,0,8,5,0,0,0,9], [0,0,3,0,9,4,0,0,7], [0,0,4,0,0,0,0,0,8], [5,0,6,0,1,0,0,0,0]]	=	[7, 3, 8, 4, 6, 1, 5, 9, 2] [6, 9, 1, 5, 3, 2, 7, 8, 4] [2, 4, 5, 9, 8, 7, 3, 6, 1] [4, 5, 2, 1, 7, 9, 8, 3, 6] [3, 8, 9, 2, 4, 6, 1, 7, 5] [1, 6, 7, 8, 5, 3, 4, 2, 9] [8, 1, 3, 6, 9, 4, 2, 5, 7] [9, 7, 4, 3, 2, 5, 6, 1, 8] [5, 2, 6, 7, 1, 8, 9, 4, 3]	39.873 sec- onds	96.652 MiB

**Kết luận:** Với hướng tiếp cận BFS, giải thuật trung bình chạy ổn với thời gian từ 0 giây đến 1s cho những trường hợp trung bình và 45 giây cho các ma trận khó. Giải thuật yêu cầu 20MiB đến 23MiB cho những trường hợp trung bình và 97MiB cho những ma trận phức tạp.

```
Testing on filled valid 9x9 grid...
Problem:
[9, 7, 4, 2, 3, 6, 1, 5, 8]
[6, 3, 8, 5, 9, 1, 7, 4, 2]
[1, 2, 5, 4, 8, 7, 9, 3, 6]
[3, 1, 6, 7, 5, 4, 2, 8, 9]
[7, 4, 2, 9, 1, 8, 5, 6, 3]
[5, 8, 9, 3, 6, 2, 4, 1, 7]
[8, 6, 7, 1, 2, 5, 3, 9, 4]
[2, 5, 3, 6, 4, 9, 8, 7, 1]
[4, 9, 1, 8, 7, 3, 6, 2, 5]

Solving with BFS...
Found solution
[9, 7, 4, 2, 3, 6, 1, 5, 8]
[6, 3, 8, 5, 9, 1, 7, 4, 2]
[1, 2, 5, 4, 8, 7, 9, 3, 6]
[3, 1, 6, 7, 5, 4, 2, 8, 9]
[7, 4, 2, 9, 1, 8, 5, 6, 3]
[5, 8, 9, 3, 6, 2, 4, 1, 7]
[8, 6, 7, 1, 2, 5, 3, 9, 4]
[2, 5, 3, 6, 4, 9, 8, 7, 1]
[4, 9, 1, 8, 7, 3, 6, 2, 5]
Elapsed time: 0.0 seconds
```

Hình 6: Kết quả

### 3.1.2 Heuristic Search - Genetic Algorithm

Input 1:

```
0 0 7 | 2 8 0 | 0 0 0
0 0 0 | 0 0 0 | 5 0 6
4 1 3 | 0 0 6 | 0 8 0
-----+-----+-----
7 2 0 | 3 9 0 | 0 0 0
3 4 0 | 0 0 0 | 8 1 0
6 8 0 | 1 0 7 | 0 0 2
-----+-----+-----
0 0 0 | 6 7 4 | 0 2 3
0 0 0 | 0 0 5 | 7 0 0
1 0 6 | 0 2 3 | 0 4 0
```

Hình 7: Input sudoku 1

Output 1:

**Giải thích input 1:** Với input đầu vào trên thì chúng ta có thể giải chúng sau khi chạy bước tiền xử lý mà không cần chạy giải thuật heuristic. Vì trong input trên chúng ta chỉ cần lần lượt áp dụng 6 bước di chuyển đầu theo hướng suy nghĩ của con người ra đã có thể đi đến kết quả bài toán. (di chuyển theo hàng-cột-khối, di chuyển theo hàng(cột-khối)-3)



```
After Preprocess
5 6 7 | 2 8 9 | 4 3 1
8 9 2 | 4 3 1 | 5 7 6
4 1 3 | 7 5 6 | 2 8 9
-----+-----+-----
7 2 1 | 3 9 8 | 6 5 4
3 4 9 | 5 6 2 | 8 1 7
6 8 5 | 1 4 7 | 3 9 2
-----+-----+-----
9 5 8 | 6 7 4 | 1 2 3
2 3 4 | 9 1 5 | 7 6 8
1 7 6 | 8 2 3 | 9 4 5
```

Hình 8: Output sudoku 1

Input 2:

```
4 0 0 | 0 7 8 | 0 0 3
0 0 0 | 2 0 0 | 0 0 0
0 0 9 | 0 0 0 | 0 1 0
-----+-----+-----
0 1 0 | 0 6 2 | 0 3 0
0 0 0 | 4 0 0 | 0 0 2
6 0 0 | 5 0 0 | 0 0 0
-----+-----+-----
0 4 0 | 0 0 0 | 7 0 0
7 0 0 | 0 3 6 | 0 0 8
0 0 0 | 0 5 0 | 0 0 0
```

Hình 9: Input sudoku 2

After Preprocess

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

Hình 10: Input sudoku 2 after preprocess

Output 2:

```

After CROSSOVER [['diidbdgdbbdcbigdeaaaaigagfееaaiddbfchbgbfhenfbbi', 80],
After MUTATION [['eagcbadboggachhebibhigbdahbfgdhebdaggheebdifdcaecac', 80],
After CROSSOVER [['eagcbadboggachhebibhigbdahbfgdhebdaggheebdifdcaecac', 80],
After MUTATION [['eagcbadboggachhebibhigbdahbfgdhebdaggheebdifdcaecac', 80],
After CROSSOVER [['eagcbadboggachhebibhigbdahbfgdhebdaggheebdifdcaecac', 80],
After MUTATION [['eagcbadboggachhebibhigbdahbfgdhebdaggheebdifdcaecac', 80],
After CROSSOVER [['eagcbadboggachhebibhigbdahbfgdhebdaggheebdifdcaecac', 80],
After MUTATION [['cgdggefchdiffgbebhcebcidiifhbehbchhidfafdbfbfdahih', 80],
After CROSSOVER [['cgdggefchdiffgbebhcebcidiifhbehbchhidfafdbfbfdahih', 80],
After MUTATION [['igehbaefdegfbfcgabahceecgeeeighfegghcidadghihhafeggf', 80],
After CROSSOVER [['igehbaefdegfbfcgabahceecgeeeighfegghcidadghihhafeggf', 80],
ccebgiabcbbeddehcbbaeffahgbefhihbgiaahaiaffgigahbahc
4 5 1 | 6 7 8 | 9 2 3
3 7 6 | 2 9 1 | 4 8 5
2 8 9 | 3 4 5 | 6 1 7
-----+-----+-----
5 1 7 | 9 6 2 | 8 3 4
9 3 8 | 4 1 7 | 5 6 2
6 2 4 | 5 8 3 | 1 7 9
-----+-----+-----
1 4 3 | 8 2 9 | 7 5 6
7 9 5 | 1 3 6 | 2 4 8
8 6 2 | 7 5 4 | 3 9 1

```

Hình 11: Output sudoku 2

**Giải thích input 2:** Với input đầu vào trên sau khi qua bước tiền xử lý thì chúng ta đã điền được một vài ô. Còn lại là những ô mà chúng ta không thể xác định chính xác liệu ô đó là số mấy thì chúng ta thực hiện giải thuật heuristic - genetic algorithm thì thu được kết quả sau "ccebgiabcbbeddehcbbaeffahgbefhihbgiaahaiaffgigahbahc" nếu thực hiện theo thứ tự của chuỗi trên thì bài toán sẽ được giải quyết. **Bảng số liệu theo chiến lược BFS**

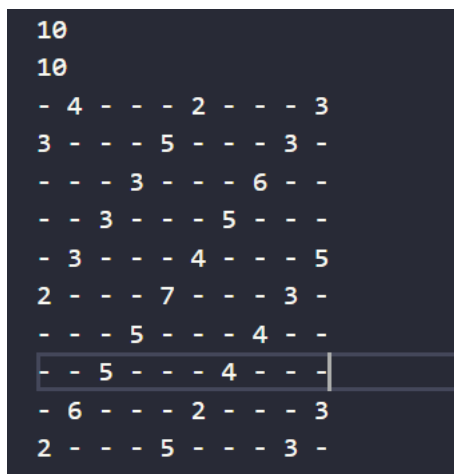
STT	Input	Output	Time	Space
1	grid [[0,0,7,2,8,0,0,0,0], [0,0,0,0,0,5,0,6], [4,1,3,0,0,6,0,8,0], [7,2,0,3,9,0,0,0,0], [3,4,0,0,0,8,1,0], [6,8,0,1,0,7,0,0,2], [0,0,0,6,7,4,0,2,3], [0,0,0,0,5,7,0,0], [1,0,6,0,2,3,0,4,0]] =	[5, 6, 7, 2, 8, 9, 4, 3, 1] [8, 9, 2, 4, 3, 1, 5, 7, 6] [4, 1, 3, 7, 5, 6, 2, 8, 9] [7, 2, 1, 3, 9, 8, 6, 5, 4] [3, 4, 9, 5, 6, 2, 8, 1, 7] [6, 8, 5, 1, 4, 7, 3, 9, 2] [9, 5, 8, 6, 7, 4, 1, 2, 3] [2, 3, 4, 9, 1, 5, 7, 6, 8] [1, 7, 6, 8, 2, 3, 9, 4, 5]	0.0037 sec- onds	15.8 MiB
2	grid [[9,7,4,2,3,6,1,5,8], [6,3,8,5,9,1,7,4,2], [1,2,5,4,8,7,9,3,6], [3,1,6,7,5,4,2,8,9], [7,4,2,9,1,8,5,6,3], [5,8,9,3,6,2,4,1,7], [8,6,7,1,2,5,3,9,4], [2,5,3,6,4,9,8,7,1], [4,9,1,8,7,3,6,2,5]] =	[9, 7, 4, 2, 3, 6, 1, 5, 8] [6, 3, 8, 5, 9, 1, 7, 4, 2] [1, 2, 5, 4, 8, 7, 9, 3, 6] [3, 1, 6, 7, 5, 4, 2, 8, 9] [7, 4, 2, 9, 1, 8, 5, 6, 3] [5, 8, 9, 3, 6, 2, 4, 1, 7] [8, 6, 7, 1, 2, 5, 3, 9, 4] [2, 5, 3, 6, 4, 9, 8, 7, 1] [4, 9, 1, 8, 7, 3, 6, 2, 5]	0.0028 sec- onds	17.4 MiB
3	grid [[0,0,0,0,5,0,9,7,6], [8,0,5,1,9,0,0,3,0], [3,7,0,0,4,0,0,8,0], [0,8,0,0,0,0,0,0,9], [0,2,0,0,0,0,4,0,7], [0,9,0,0,2,6,0,1,5], [0,0,0,0,8,1,6,0,0], [9,0,0,3,0,0,0,0,0], [2,0,0,4,0,9,0,0,0]] =	[9, 7, 4, 2, 3, 6, 1, 5, 8] [6, 3, 8, 5, 9, 1, 7, 4, 2] [1, 2, 5, 4, 8, 7, 9, 3, 6] [3, 1, 6, 7, 5, 4, 2, 8, 9] [7, 4, 2, 9, 1, 8, 5, 6, 3] [5, 8, 9, 3, 6, 2, 4, 1, 7] [8, 6, 7, 1, 2, 5, 3, 9, 4] [2, 5, 3, 6, 4, 9, 8, 7, 1] [4, 9, 1, 8, 7, 3, 6, 2, 5]	0.00433 seconds	17.6 MiB
4	grid [[0,0,0,0,5,0,9,7,6], [8,0,5,1,9,0,0,3,0], [3,7,0,0,4,0,0,8,0], [0,8,0,0,0,0,0,0,9], [0,2,0,0,0,0,4,0,7], [0,9,0,0,2,6,0,1,5], [0,0,0,0,8,1,6,0,0], [9,0,0,3,0,0,0,0,0], [2,0,0,4,0,9,0,0,0]] =	[4, 5, 1, 6, 7, 8, 9, 2, 3] [3, 7, 6, 2, 9, 1, 4, 8, 5] [2, 8, 9, 3, 4, 5, 6, 1, 7] [5, 1, 7, 9, 6, 2, 8, 3, 4] [9, 3, 8, 4, 1, 7, 5, 6, 2] [6, 2, 4, 5, 8, 3, 1, 7, 9] [1, 4, 3, 8, 2, 9, 7, 5, 6] [7, 9, 5, 1, 3, 6, 2, 4, 8] [8, 6, 2, 7, 5, 4, 3, 9, 1]	[4.18, 15.69, 10.75, 4.55, 1.423] seconds	18.7 MiB

**Kết luận:** Với hướng tiếp cận Heuristic - Genetic Algorithm, trong các trường hợp dễ giải thuật thường chạy với thời gian rất nhỏ. Tuy nhiên đối với các bài khó thì lại tốn nhiều thời gian hơn và thời gian cho chỗ lần giải có thể có sự khác biệt lớn như trong input số 4 kết quả 5 lần đo trải dài từ hơn 15 giây cho tới dưới 2 giây.

## 3.2 Shikaku

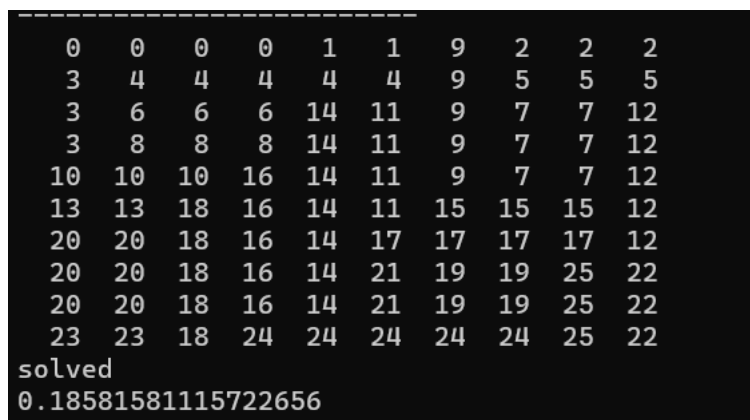
### 3.2.1 Depth First Search

Input 1:



Hình 12: Input shikaku 1

Output 1:



Hình 13: Kết quả

### Giải thích:

Trạng thái lastCell của input

-----STATE LAST CELL-----									
3	0	0	6	14	1	9	7	7	12
4	4	8	6	14	11	9	7	7	12
3	10	8	16	14	11	9	7	7	12
8	20	18	16	14	11	11	17	15	12
13	20	18	16	14	12	19	17	15	12
14	20	18	16	24	14	19	17	17	15
20	20	20	16	24	19	19	19	17	22
20	20	20	20	24	21	19	19	25	22
23	20	20	20	24	21	21	22	25	22
24	24	24	24	24	24	25	25	25	25

Lần lượt đánh index cho các ô theo thứ tự và kiểm tra với state lastCell, chẳng hạn đánh index cho ô thể hiện diện tích đầu tiên

-----STATE COVER-----									
0	0	0	0	-1	1	-1	-1	-1	2
3	-1	-1	-1	4	-1	-1	-1	5	-1
-1	-1	-1	6	-1	-1	-1	7	-1	-1
-1	-1	8	-1	-1	-1	9	-1	-1	-1
-1	10	-1	-1	-1	11	-1	-1	-1	12
13	-1	-1	-1	14	-1	-1	-1	15	-1
-1	-1	-1	16	-1	-1	-1	17	-1	-1
-1	-1	18	-1	-1	-1	19	-1	-1	-1
-1	20	-1	-1	-1	21	-1	-1	-1	22
23	-1	-1	-1	24	-1	-1	-1	25	-1

Hình 14: State hợp lệ

State lastCell, ta thấy có 2 vị trí được đánh index là 0: (0,1) và (0,2). Số sánh với state hiện tại, cả 2 vị trí này đều đã được đánh index với giá trị là 0 => state này là hợp lệ; sử dụng state này để tiếp tục đánh index.

Tiếp tục đánh index như vậy, cho đến khi đánh index lần đầu tiên cho ô thể hiện diện tích thứ 3, state lúc này sẽ như sau:

-----STATE NOT COVER-----									
0	0	0	0	1	1	-1	2	2	2
3	3	3	-1	4	-1	-1	-1	5	-1
-1	-1	-1	6	-1	-1	-1	7	-1	-1
-1	-1	8	-1	-1	-1	9	-1	-1	-1
-1	10	-1	-1	-1	11	-1	-1	-1	12
13	-1	-1	-1	14	-1	-1	-1	15	-1
-1	-1	-1	16	-1	-1	-1	17	-1	-1
-1	-1	18	-1	-1	-1	19	-1	-1	-1
-1	20	-1	-1	-1	21	-1	-1	-1	22
23	-1	-1	-1	24	-1	-1	-1	25	-1

Hình 15: State không hợp lệ

So sánh với state lastCell, ta thấy rằng ở vị trí (2,0) vẫn chưa được điền. Vì vậy state này ở việc đánh index cho ô thể hiện diện tích thứ 3 có trạng thái khác với lastCell nên sẽ không nhận trạng thái này. Quay lại trạng thái trước đó để tiếp tục đánh index cho ô thể hiện diện tích thứ 3 bằng cách khác.

-----STATE COVER-----									
0	0	0	0	1	1	-1	2	2	2
3	-1	-1	-1	4	-1	-1	-1	5	-1
-1	-1	-1	6	-1	-1	-1	7	-1	-1
-1	-1	8	-1	-1	-1	9	-1	-1	-1
-1	10	-1	-1	-1	11	-1	-1	-1	12
13	-1	-1	-1	14	-1	-1	-1	15	-1
-1	-1	-1	16	-1	-1	-1	17	-1	-1
-1	-1	18	-1	-1	-1	19	-1	-1	-1
-1	20	-1	-1	-1	21	-1	-1	-1	22
23	-1	-1	-1	24	-1	-1	-1	25	-1

Hình 16: State hợp lệ trước đó

Trạng thái tiếp theo sau khi đánh index:

-----STATE COVER-----									
0	0	0	0	1	1	-1	2	2	2
3	-1	-1	-1	4	-1	-1	-1	5	-1
3	-1	-1	6	-1	-1	-1	7	-1	-1
3	-1	8	-1	-1	-1	9	-1	-1	-1
-1	10	-1	-1	-1	11	-1	-1	-1	12
13	-1	-1	-1	14	-1	-1	-1	15	-1
-1	-1	-1	16	-1	-1	-1	17	-1	-1
-1	-1	18	-1	-1	-1	19	-1	-1	-1
-1	20	-1	-1	-1	21	-1	-1	-1	22
23	-1	-1	-1	24	-1	-1	-1	25	-1

Hình 17: State sau khi đánh index cho vị trí 3

State lastCell có 2 vị trí được đánh index là 3: (0,0) và (2,0). So sánh với state lastCell, ở trạng thái này vị trí (2,0) đã được đánh index là 3  $\Rightarrow$  hợp lệ và vị trí (0,0) đã được đánh index là 0 ( đã verify là valid khi đánh index cho ô thể hiện diện tích đầu tiên). Vì thế state này là hợp lệ và dùng state này để tiếp tục đánh index.

Lần lượt làm như vậy cho đến khi đánh index hết tất cả các thể hiện diện tích, nếu state cuối cùng này là solution thì bài toán được giải quyết.

```
0 0 0 0 1 1 9 2 2 2
3 4 4 4 4 4 9 5 5 5
3 6 6 6 14 11 9 7 7 12
3 8 8 8 14 11 9 7 7 12
10 10 10 16 14 11 9 7 7 12
13 13 18 16 14 11 15 15 15 12
20 20 18 16 14 17 17 17 17 12
20 20 18 16 14 21 19 19 25 22
20 20 18 16 14 21 19 19 25 22
23 23 18 24 24 24 24 24 25 22
solved
0.18581581115722656
```

Hình 18: Kết quả

Bài toán được giải quyết sau 0.18 giây, bộ nhớ tiêu tốn 20.9 Mi.

Input 2:

```

12
16
- - - - 4 - - - - - 7 - - - -
- - 6 - - - - 3 2 - - - - 4 - -
- 6 - - - 2 - - - - 3 - - - 3 -
- - 6 - - - 2 3 - - - 6 - - -
4 - - - - - 4 - - 5 - - - - 5
- - 4 - - 6 - - - - 4 - - 4 - -
- - 3 - - 3 - - - - 4 - - 4 - -
6 - - - - - 2 - - 6 - - - - 6
- - - 4 - - - 3 4 - - - 5 - - -
- 6 - - - 5 - - - - 6 - - - 4 -
- - 4 - - - - 3 4 - - - - 6 - -
- - - - 6 - - - - - 5 - - - -

```

Hình 19: Input shikaku 2

Output 2:

```

-----
 6  6  0  0  0  0  1  1  1  1  1  1  1  5  5 17
 6  6  2  2  2  3  3  3  4 16  8 13 13  5  5 17
 6  6  2  2  2  7  7 11  4 16  8 13 13 21  9 17
14 10 10 10 10 10 10 11 12 16  8 13 13 21  9 17
14 18 18 19 19 19 15 15 12 16 20 20 33 21  9 17
14 18 18 19 19 19 15 15 12 16 20 20 33 21 29 29
14 22 22 22 23 23 23 28 28 28 24 24 33 25 29 29
26 26 26 30 30 27 27 28 28 28 24 24 33 25 29 29
26 26 26 30 30 31 31 31 32 32 32 32 33 25 37 37
34 34 35 35 35 35 39 40 40 36 36 36 25 37 37
34 34 38 38 42 42 42 39 40 40 36 36 36 41 41 41
34 34 38 38 42 42 42 39 43 43 43 43 41 41 41
solvedd
2.817357063293457

```

Hình 20: Kết quả

Với input có kích thước 12x16, bài toán được giải quyết sau 2.81 giây, bộ nhớ tiêu tốn 20.9 MiB.



Input 3:

```

16
16
2 - - - - - - - - - - - - - - -
- - - - - 14 - - - - 16 - - - - -
- - - 6 - - - - - 10 - - - - -
- - - - - - - - - 14 - - - - -
- - - - - 4 - - - - - 8 -
- - - - - 4 - - - - 14 - - - - -
- - - - - - - - - 10 - - - - -
6 - 16 2 - 2 - 4 - - - - - -
- - - - - - - - - 16 - - - - -
- - - 6 - 3 - - - 2 - - 4 - -
2 - - - - - 9 - - - - 6 - - 4
- - - 3 - - - - - - 12 - - -
4 - - - 3 - - 4 - - - - - -
- - - - - - - - - 12 - - - - 2
- - 4 - - - - - - 8 - - - - -
- 2 - 2 - 8 - - - - 8 - - - - -

```

Hình 21: Input shikaku 3

Output 3:

```

-----
 0  1  1  1  1  1  1  1  2  2  2  2  2  2  2
 0  1  1  1  1  1  1  1  2  2  2  2  2  2  2
 3  3  3  3  3  4  4  4  4  4  4  4  4  4
 5  5  5  5  5  5  5  5  5  5  5  5  5  7  7
11 12 12 6  6  6  6  9  9  9  9  9  9  7  7
11 12 12 8  8  8  8  9  9  9  9  9  9  7  7
11 12 12 13 10 10 10 10 10 10 10 10 10 7  7
11 12 12 13 14 14 15 15 16 16 16 16 16 16 16
11 12 12 17 17 18 15 15 16 16 16 16 16 16 16
11 12 12 17 17 18 22 22 22 19 19 20 20 20 24
21 12 12 17 17 18 22 22 22 23 23 23 23 23 24
21 12 12 25 25 25 22 22 22 26 26 26 26 26 24
27 27 28 28 28 29 29 29 29 26 26 26 26 26 24
27 27 30 30 30 30 30 30 30 30 30 30 30 31 31
32 32 32 32 36 36 36 36 33 33 33 33 33 33 33
34 34 35 35 36 36 36 36 37 37 37 37 37 37 37
solved
1.688276767730713

```

Hình 22: Kết quả

Với input có kích thước 16x16, bài toán được giải quyết sau 1.68 giây, bộ nhớ tiêu tốn 21 MiB.



Bảng số liệu theo chiến lược DFS

STT	Input	Time	Space
1	<pre>11 13 - - 6 - - - - - 10 - - - - - 4 - - - 2 - - - - - 9 - - - 6 - - - 8 - - - - 3 - - - - 3 - - - - - - 4 - 3 - - - - 6 - 3 - - - - - 9 - 6 - - - - 4 - 2 - - - - - - - 3 - - - - 2 - - - 4 - - - 12 - - - 10 - - - - 6 - - - 6 - - - - - 8 - - - - - 4 - -</pre>	13.16 seconds	20.9 MiB

Output :

0	0	0	1	1	1	1	1	1	1	1	1	1
0	0	0	7	2	2	2	2	3	6	6	6	6
4	4	4	7	5	5	5	10	3	6	6	6	6
4	4	4	7	5	5	5	10	8	8	8	14	14
4	4	4	9	9	9	9	10	13	13	13	14	14
11	11	12	17	22	15	15	16	13	13	13	14	14
11	11	12	17	22	15	15	16	13	13	13	21	21
11	11	12	17	22	20	20	20	18	18	25	21	21
19	19	19	19	22	20	20	20	23	23	25	21	21
24	24	24	24	22	20	20	20	23	23	25	21	21
24	24	24	24	22	20	20	20	23	23	25	21	21

STT	Input	Time	Space
2	<pre>15 20 - - - 6 - - 7 - - - 9 - - - - - - 3 - - - - - - - 9 - - 4 - 6 - - - 2 - - 9 - 6 - - - - - - - 4 - - 5 - - - 4 - - - - 4 - 8 - 4 - - - - 6 - - - 4 - 15 - - - - - - - 10 - - - - - - - - - - - - - - - - - 8 - - - - - 12 - - 12 - - 8 - - - 3 - - 16 - - 2 - - - - - 6 - - - - - - - - - - - - - - - - - 6 - - - - - - - 6 - 3 - - - 4 - - - - 10 - 6 - 6 - - - - 4 - - - 4 - - 4 - - - - - - - - 3 - 2 - - 3 - - 6 - 9 - 10 - - - - - - - - 8 - - - - - - 6 - - - 4 - 4 - - - -</pre>	2.42 seconds	20.9 MiB

Output :



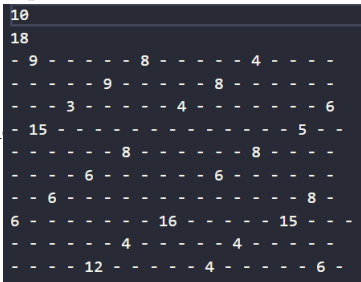
0	0	0	0	0	0	1	1	1	1	1	1	1	2	2	2	4	4	4	5
6	6	6	3	3	3	11	8	8	8	9	9	9	2	2	2	4	4	4	5
6	6	6	10	10	7	11	8	8	8	9	9	9	2	2	2	4	4	4	5
14	14	15	10	10	7	11	8	8	8	12	12	12	12	17	13	13	13	13	5
14	14	15	22	22	22	11	23	16	16	16	16	16	16	17	18	18	18	18	18
14	14	15	22	22	22	11	23	19	19	20	20	20	20	17	18	18	18	18	18
14	14	15	22	22	22	27	23	19	19	20	20	20	20	17	18	18	18	18	18
21	21	21	22	22	22	27	23	19	19	24	24	24	25	25	25	25	32	32	26
21	21	21	29	29	30	27	23	19	19	31	28	28	25	25	25	25	32	32	26
21	21	21	29	29	30	27	23	19	19	31	28	28	25	25	25	25	32	32	33
21	21	21	29	29	30	27	23	35	38	31	28	28	25	25	25	25	32	32	33
34	34	34	34	34	34	27	23	35	38	31	36	36	36	36	37	37	32	32	33
42	42	42	43	43	43	43	43	35	38	39	39	40	40	40	37	37	41	41	33
42	42	42	43	43	43	43	43	35	44	44	44	44	44	44	44	44	41	41	33
42	42	42	45	45	45	45	45	45	46	46	46	46	47	47	47	47	41	41	33

STT	Input	Time	Space
3	<pre>15 20 - - - - 4 - - - 5 6 - - - 4 4 - - - - 4 - - - 2 - 4 - - - - - - - - - - 4 - - - - 2 - 2 - - 4 4 - - - 4 2 - - - - 6 4 - - - - 4 - 4 - 3 6 - 4 2 4 - - - - 10 - - - 8 2 - 3 - - - 2 - - - - 2 - - 6 - - - - - - - - - - 6 6 - 4 - - - - 10 8 - - - 6 - 6 3 - - - - - - - - - - - 6 - - 4 - - - 2 - - - - 4 - 3 3 - - - 4 - - - - 6 4 4 - 3 2 - 3 - 3 - - - 6 4 - - - 6 3 - - - 6 4 - - 2 - 4 - - - 6 - - - - - - - - - - - 3 - 2 - - - 6 - - - - 3 4 - - - 4 6 - - - - 6 - - -</pre>	0.15 seconds	21 MiB

Output :

5	5	0	0	0	0	1	1	1	1	1	2	2	2	3	3	4	4	4	4
5	5	15	15	6	6	7	7	7	7	35	2	2	2	3	3	13	13	14	21
8	8	15	15	16	16	9	9	10	10	35	11	11	12	12	36	13	13	14	21
8	8	15	15	16	16	17	17	17	17	35	11	11	12	12	36	19	20	20	21
22	23	23	24	24	24	24	24	34	34	35	18	18	18	18	36	19	20	20	21
22	23	23	24	24	24	24	24	34	34	35	25	26	26	27	36	19	20	20	28
31	31	32	32	29	29	30	30	34	34	35	25	39	39	27	36	37	37	38	28
31	31	32	32	33	33	30	30	34	34	35	25	39	39	27	36	37	37	38	47
31	31	32	32	33	33	30	30	34	34	35	25	39	39	40	40	37	37	38	47
41	41	49	42	42	42	42	43	44	68	52	25	45	45	40	40	46	46	46	47
48	48	49	50	51	51	51	43	44	68	52	25	45	45	53	53	46	46	46	47
48	48	49	50	66	57	57	43	44	68	52	25	60	60	53	53	54	61	61	61
55	55	56	65	66	57	57	58	58	68	59	25	60	60	53	53	54	61	61	61
55	55	56	65	66	57	57	58	58	68	59	62	62	62	63	63	54	64	64	64
55	55	56	65	66	67	67	67	67	68	69	69	69	69	69	69	54	64	64	64



STT	Input	Time	Space
4		5.65 seconds	21 MiB

Output :

0	0	0	5	3	3	3	1	1	1	1	2	2	2	2	9	7	7
0	0	0	5	3	3	3	1	1	1	1	4	4	4	4	9	7	7
0	0	0	5	3	3	3	6	6	6	6	4	4	4	4	9	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	9	15	15
16	14	14	12	12	10	10	11	11	11	11	11	11	11	11	9	15	15
16	14	14	12	12	10	10	17	17	17	17	13	13	18	18	18	15	15
16	14	14	12	12	10	10	17	17	17	17	13	13	18	18	18	15	15
16	21	21	21	21	10	10	17	17	17	17	13	13	18	18	18	23	23
16	21	21	21	21	19	19	17	17	17	17	20	20	18	18	18	23	23
16	21	21	21	21	19	19	22	22	22	22	20	20	18	18	18	23	23

**Kết luận:** Với hướng tiếp cận DFS, giải thuật trung bình chạy ổn với thời gian từ 0 giây đến 6s cho những trường hợp trung bình và 13 giây cho các ma trận khó. Giải thuật yêu cầu 21MiB bộ nhớ cho đa số ma trận.

### 3.2.2 Heuristic Search - Genetic Algorithm

Input 1:

```
[0,0,0,4,2,2,2],
[0,2,2,0,0,0,0],
[0,0,7,0,0,0,0],
[0,4,0,0,0,2,0],
[2,0,0,0,3,0,0],
[0,4,4,0,0,0,0],
[0,0,0,2,0,3,4]
```

Hình 23: Input với số 0 là chưa tô màu

```
__locationData__ [(0, 3, 4), (0, 4, 2), (0, 5, 2), (0, 6, 2), (0, 7, 2)]  
__locationDataFactor__ [[2, 4, 1], [2, 1], [2, 1], [2, 1], [2, 1]]  
__puzzle__  
[0, 0, 0, 0, 1, 2, 3]  
[4, 4, 5, 5, 1, 2, 3]  
[6, 6, 6, 6, 6, 6, 6]  
[-1, 7, -1, -1, 8, 8, 15]  
[9, -1, -1, -1, 10, 14, 15]  
[11, 11, 12, -1, -1, 14, 15]  
[11, 11, -1, 13, -1, 14, 15]  
__partitions__ [[[]], [], [], [], [], [], [], [(3, 1, 2, 2, 7), (3, 1, 2, 2, 7)]]
```

Hình 24: Sau khi qua bước tiền xử lý

```
-----1-10-01--
[0, 0, 0, 0, 1, 2, 3]
[4, 4, 5, 5, 1, 2, 3]
[6, 6, 6, 6, 6, 6, 6]
[7, 7, 7, 7, 8, 8, 15]
[9, 9, 12, 12, 10, 14, 15]
[11, 11, 12, 12, 10, 14, 15]
[11, 11, 13, 13, 10, 14, 15]
Solution Found in 0.03900027275085449 seconds
```

Hình 25: Kết quả thu được

Input 2:

```
[0,6,0,0,0,3,0],  
[0,0,0,0,0,2,0],  
[0,2,0,3,0,2,0],  
[2,0,0,0,5,0,0],  
[0,0,6,0,0,0,4],  
[0,0,0,0,0,0,7],  
[0,3,0,0,4,0,0]
```

Hình 26: Input với số 0 là chưa tô màu

```
__locationData__ [(0, 1, 6), (0, 5, 3), (1, 5, 2), (2, 1, 2), (2, 3,  
__locationDataFactor__ [[3, 2, 6, 1], [3, 1], [2, 1], [2, 1], [3, 1],  
__puzzle__  
[-1, 0, -1, -1, -1, 1, -1]  
[-1, -1, -1, -1, -1, 2, -1]  
[-1, 3, -1, 4, -1, 5, -1]  
[6, -1, -1, -1, 7, -1, -1]  
[-1, -1, 8, -1, -1, -1, 9]  
[10, 10, 10, 10, 10, 10, 10]  
[-1, 11, -1, -1, 12, -1, -1]  
__partitions__ [[(0, 0, 2, 3, 0), (0, 1, 2, 3, 0)], [(0, 3, 1, 3, 1),
```

Hình 27: Sau khi qua bước tiền xử lý

```
0103010111-01  
[0, 0, 0, 4, 1, 1, 1]  
[0, 0, 0, 4, 2, 2, 9]  
[6, 3, 3, 4, 5, 5, 9]  
[6, 7, 7, 7, 7, 7, 9]  
[8, 8, 8, 8, 8, 8, 9]  
[10, 10, 10, 10, 10, 10, 10]  
[11, 11, 11, 12, 12, 12, 12]  
Solution Found in 0.34569811820983887 seconds
```

Hình 28: Kết quả thu được

Với input như trên sau khi trải qua các bước biến đổi dữ liệu, tạo ra bảng các factors(đồ dài

cạnh có thể có), tạo ra list các phân vùng và tô màu cho những số nào mà chỉ có 1 phân vùng vì đó chắc chắn là cách duy nhất để tô màu cho số đó. Sau bước tiền xử lý thì các số đều có nhiều hơn 1 phân vùng có thể tô được đó chúng ta áp dụng giải thuật Heuristic - Genetic Algorithm.

**Bảng số liệu theo chiến lược Heuristic - Genetic - Algorithm**

STT	Input	Output	Time	Space
1	grid = [[0,6,0,0,0,3,0], [0,0,0,0,0,2,0], [0,2,0,3,0,2,0], [2,0,0,0,5,0,0], [0,0,6,0,0,0,4], [0,0,0,0,0,0,7], [0,3,0,0,4,0,0]]	[0, 0, 0, 4, 1, 1, 1] [0, 0, 0, 4, 2, 2, 9] [6, 3, 3, 4, 5, 5, 9] [6, 7, 7, 7, 7, 7, 9] [8, 8, 8, 8, 8, 8, 9] [10, 10, 10, 10, 10, 10, 10] [11, 11, 11, 12, 12, 12, 12]	[0.3457, 0.852] sec- onds	33.9 MiB
2	grid = [[0,0,0,4,2,2,2], [0,2,2,0,0,0,0], [0,0,7,0,0,0,0], [0,4,0,0,0,2,0], [2,0,0,0,3,0,0], [0,4,4,0,0,0,0], [0,0,0,2,0,3,4]]	[0, 0, 0, 0, 1, 2, 3] [4, 4, 5, 5, 1, 2, 3] [6, 6, 6, 6, 6, 6, 6] [7, 7, 7, 7, 8, 8, 15] [9, 9, 12, 12, 10, 14, 15] [11, 11, 12, 12, 10, 14, 15] [11, 11, 13, 13, 10, 14, 15]	[0.0254, 0.009, 0.014, 0.006] sec- onds	40.8 MiB
3	grid = [[2,0,3,0,0], [3,0,0,0,0], [0,0,4,4,0], [0,0,2,2,0], [0,3,0,2,0]]	[0, 0, 1, 1, 1] [2, 3, 3, 4, 4] [2, 3, 3, 4, 4] [2, 5, 5, 6, 6] [7, 7, 7, 8, 8]	0.01841 sec- onds	42.4 MiB



## Tài liệu tham khảo

- [1] Puzzle logic. Link: <https://www.puzzle-pipes.com/>. Last accessed: 15/03/2022.
- [2] Finding Solutions to Sudoku Puzzles Using Human Intuitive Heuristics. Link: [https://www.researchgate.net/publication/256087959\\_Finding\\_Solutions\\_to\\_Sudoku\\_Puzzles\\_Using\\_Human\\_Intuitive\\_Heuristics](https://www.researchgate.net/publication/256087959_Finding_Solutions_to_Sudoku_Puzzles_Using_Human_Intuitive_Heuristics). Last accessed: 30/03/2022.
- [3] Implementation of Heuristic Technique and Genetic Algorithms in Shikaku Puzzle Problem. Link: [https://www.researchgate.net/publication/256087959\\_Finding\\_Solutions\\_to\\_Sudoku\\_Puzzles\\_Using\\_Human\\_Intuitive\\_Heuristics](https://www.researchgate.net/publication/256087959_Finding_Solutions_to_Sudoku_Puzzles_Using_Human_Intuitive_Heuristics). Last accessed: 30/03/2022.