CSC14003 – Introduction to Artificial Intelligence

Coloring Puzzle

MSSV	Name	Role
19127346	Nguyễn Nhật Cường	Leader
19127382	Đinh Hải Giang	Member
19127567	Võ Trần Quang Thông	Member
19127517	Hồ Thiên Phước	Member

The completion for project: 92%

Video demo:

https://drive.google.com/file/d/1i3NQ2Q6cnZZq0zWAHkfyDkjwDN3blHGW/view?usp=sharing

Coloring puzzle

I. Introduction:

1. Problem:

Given a matrix of size $m \times n$, where each cell will be a non-negative integer or -1 (empty cell). Each cell is considered to be adjacent to itself and 8 surrounding cells.

The problem is needed to color all the cells of the matrix with either blue or red, so that the number inside each cell corresponds to the number of blue squares adjacent to that cell.

2. The overall plan:

No.	Task	Member	%
1	Generate CNFs automatically	Cường	100
2	Use pysat library to solve CNFs correctly	Phước	100
3	Apply A* to solve CNFs without a library	Cường	75
4	Program brute-force	Thông	100
5	Program backtracking	Giang	100

6	Graphic interface	Phước	50
7	Test with test cases	All	100
8	Analysis	Cường	100
9	Write report	Cường, Phước	100

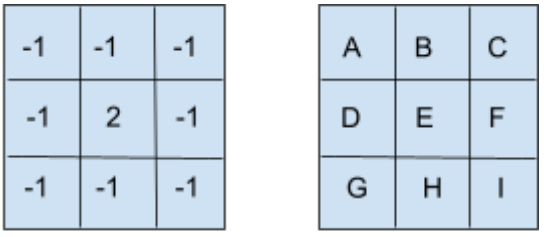
II. Approach:

1. Solution Description:

- Assign a logical variable to each cell in which the cell is green if the variable is True and red if False
- The initial part of solving a problem with pySAT is the encoding of solutions into CNF clauses corresponding with each cell. The CNF clauses should be simple enough so that operations can be executed easily and descriptive enough to characterize solutions. In this problem, cells should be assigned CNF clauses to characterize the solution and they should be shown as a list to easily check the constraints of the problem. The elements in the list will be assigned with a value 0/False/red or 1/True/green and must satisfy constraints.

2. Example:

a. Variable 2:



- There are 9C2 ways to assign logical values to the given cells. Assume that we assign values green which means True to A and B. So the rest need to be False.
- The clause for this case:

$$(A \land B \Rightarrow \neg C \land \neg D \land \neg E \land \neg F \land \neg G \land \neg H \land \neg I)$$

$$\wedge\ (\neg C \wedge \neg D \wedge \neg E \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I \Rightarrow A \wedge B)$$
$$\text{---------------------------------------}$$
$$A \wedge B \Leftrightarrow \neg C \wedge \neg D \wedge \neg E \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I$$

- Because the term "assigning True values to cells" depends on "assigning False values to cells" and otherwise. So we need to keep the constraints on both terms. So equals to:

$$(\neg A \vee \neg B \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg D) \wedge (\neg A \vee \neg B \vee \neg E) \wedge (\neg A \vee \neg B \vee \neg F) \wedge (\neg A \vee \neg B \vee \neg G) \wedge (\neg A \vee \neg B \vee \neg H) \wedge (\neg A \vee \neg B \vee \neg I) \wedge (A \vee C \vee D \vee E \vee F \vee G \vee H \vee I) \wedge (B \vee C \vee D \vee E \vee F \vee G \vee H \vee I)$$

b. Variable 3:

-1	-1	-1
-1	3	-1
-1	-1	-1

cell

A	B	C
D	E	F
G	H	I

variables

- There are 9C3 ways to assign logical values to the given cells. Assume that we assign values green which means True to A and B and E. So the rest need to be False.
- The clause for this case:

$$(A \wedge B \wedge E \Rightarrow \neg C \wedge \neg D \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I)$$
$$\wedge\ (\neg C \wedge \neg D \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I \Rightarrow A \wedge E \wedge B)$$
$$\text{---------------------------------------}$$
$$A \wedge B \wedge E \Leftrightarrow \neg C \wedge \neg D \wedge \neg F \wedge \neg G \wedge \neg H \wedge \neg I$$

- Because the term "assigning True values to cells" depends on "assigning False values to cells" and otherwise. So we need to keep the constraints on both terms. So equals to:

$$(\neg A \vee \neg B \vee \neg E \vee \neg C) \wedge (\neg A \vee \neg B \vee \neg E \vee \neg D) \wedge (\neg A \vee \neg B \vee \neg E \vee \neg F) \wedge (\neg A \vee \neg B \vee \neg E \vee \neg G) \wedge (\neg A \vee \neg B \vee \neg E \vee$$

¬H) ∧ (¬A ∨ ¬B ∨ ¬E ∨ ¬I) ∧ (A ∨ C ∨ D ∨ E ∨ F ∨ G ∨ H ∨ I) ∧ (B ∨ C ∨ D ∨ E ∨ F ∨ G ∨ H ∨ I) ∧ (A ∨ B ∨ C ∨ D ∨ F ∨ G ∨ H ∨ I)

- This is the CNF clause from this case. In generality:

$$\forall x, y \in P,\ x \neq y \text{ and } K = P \setminus Q$$

- Call p_i is clause of a case in 9C3 cases which x_i,y_i is assigned and the rest is False or $\forall k \in K_i$, k=False. So the CNF form for is:

$$[\bigwedge_{k \in K_i} (\neg x_i \vee \neg y_i \vee \neg k)] \wedge [[x_i \vee (\bigvee_{k \in K_i} k)] \wedge [y_i \vee (\bigvee_{k \in K_i} k))]]$$

3. pySAT approach:

- pySAT is a Python (2.7, 3.4+) toolkit, which aims at providing a simple and unified interface to a number of state-of-art Boolean satisfiability (SAT) solvers as well as to a variety of cardinality and pseudo-Boolean encodings.
- Boolean variables in PySAT are represented as natural identifiers,e.g. numbers from $\mathbb{N}>0$. A literal in PySAT is assumed to be an integer, e.g.-1 represents a literal $\neg x_1$ while 5 represents a literal x_5. A clause is a list of literals, e.g.[-3,-2] is a clause $\neg x_3 \vee \neg x_2$
- This process continues until some threshold value or stopping criterion is met. (no more cells can be red)

4. A* approach:

- Heuristic function
- Path cost from start node to current node
- Path cost from start node to current node + Heuristic cost
- Average of Path cost from start node to current node and Heuristic cost

III. Implementation:

1. Environments:

- Programming language: Python
- Version: 3.9.4
- IDE: Sublime Text, Jupyter Notebook

2. Data files:

- Input file: *input.txt*
  + First line: 2 integers as the matrix shape.
  + Another line: a matrix of non-negative numbers and -1.
- Output file: *output.txt*
  + An adjacency matrix (green=1/red=0 cells)

3. pySat:

A. Overview:

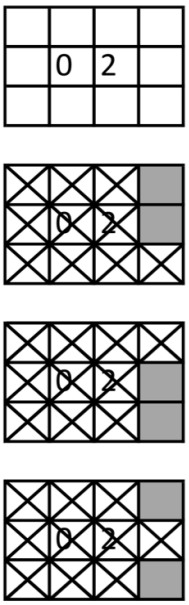
- Listed every possible solution.
- The toughness of the problem:
  + Don't know the worst case.
  + A similar situation occurs with *4* in a corner. Since a clue in the corner of a puzzle has only three neighbors, all four squares in the highlighted area – the one containing the *4* and the three around it - must be painted. The value of the corner is satisfied in range [0;4]
  + 6 or lower for the value along the edges.
  + There are no changes in each step.

B. Definition of objects:

[1]. *it*: represent all valid situations to solve one cell in input matrix
[2]. *lvars*: ascending list count from 1 to NxN
[3]. *adjacents*: neighbors
[4]. *clauses*: list of clauses for each cell.
[5]. *pre*: a CNF clause includes all clauses.

C. Combination Operation:

- Using *itertools.combinations()* in pySAT library.
- Purpose: get all possible ways/solutions to color a 'number' cell.



- With more complex boards of this game, it is not enough because it would have more than one possible solution or no solution.
- The only restriction is no data (Manhattan heuristic, Euclidean heuristic) to exclude cases.

D. Algorithm:

Step 1: Read file .txt

Transform .txt to 2d array of integer

Step 2: Get *lvars*

Step 3: Get all the clauses then add to *pre*

Check the cell's position in range or not?

Append surrounding cells to *adjacents*

Color red by inverting to negative integer, apply for each *it*

Color green by inverting to positive integer, apply for each *it*

Append to *clauses*

Step 4: Solve CNF by using *get_model()* and *add_clause()* in pySAT library

Step 5: Write adjacency matrix to file .txt

4. A*:

A. Overview:

A* algorithm has 3 parameters:

- g: the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
- h: also known as the heuristic value, it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an overestimation of the cost.
- f: it is the sum of g and h. So, f = g + h

The way that the algorithm makes its decisions is by taking the f-value into account. The algorithm selects the smallest f-valued cell and moves to that cell. This process continues until the algorithm reaches its goal cell.

B. Definition of objects:

[1]. Class Number(ColoringPuzzle)
[2]. Class Box(ColoringPuzzle)
[3]. Class ColoringPuzzle

C. I/O of the algorithm:

- Input:

The program gets a table in CSV in the format that describes the desired Fill-A-Pix table for which a solution is desired. The CSV format is selected because Excel or other similar spreadsheets can create a feed. The CSV format table is easy to handle in the program.

- Output:

The outcome may be challenging to display the user at the terminal depending on the size of the feed, so the program finally stores the task in the CSV format. CSV includes numbers corresponding to colors 0 and 1 depending on the box red or green. The user may, if desired, look at the outcome in the spreadsheet program where the outcome is smarter.

D. Algorithm:

Step 1: Read file .txt

Transfer the .txt file into an array with elements that are string containing non-negative integers by a ';'.

Example: -1 -1 -1 -1 1
-1 9 -1 -1 -1
-1 8 8 -1 -1
-1 -1 -1 -1 4

4 -1 5 -1 2

⇒ [‘;;;;1’, ‘;9;;;’, ‘;8;8;;’, ‘;;;;4’, ‘4;;5;;2’]

Step 2: Create box and number values.

Step 3: Use the loop, generate the heuristic value at each iteration (Numbers to Do), check the conditions in the match box to assign the corresponding value (0 and 1)

Step 4: Check the validity of the modified matrix, if valid, write the result to the file.

IV. Experiment:

1. Overview:

a. Brute-force:

Running time is very slow. Because the combination of all cases may occur, it is possible to cause overloading in the test cases that have a lot of non-negative values, specifically from the 5x5 size matrix.

b. Backtracking:

Runtime faster than brute-force, does not cause overload, but running time is still quite slow. In small cases, the runtime can be up to tens of minutes.

c. pySat:

Because using the library available, the algorithm is not too complicated but gives very high efficiency. The runtime in large-sized cases just takes up a few seconds.

d. A*:

Do not use the available tools so the algorithm is quite complicated, but the effect is very large. Similar to pySat, A * can resolve large-sized cases in just a few seconds, even a few milliseconds.

2. Test cases:

a. 5x5:

Input:

```
5 5
-1 -1 -1 -1 1
-1 9 -1 -1 -1
-1 8 8 -1 -1
-1 -1 -1 -1 4
4 -1 5 -1 2
```

Output:

```
1 1 1 0 0
1 1 1 1 0
1 1 1 1 1
1 1 0 1 0
1 1 1 1 0
```

Runtime:

- Brute-force: 120.7s
- Backtracking: 815ms
- pySat: 802ms
- A*: 423ms

b. 7x7:

Input:

```
7 7
-1 -1 2 2 2 -1 -1
-1 -1 2 2 2 -1 -1
2 2 2 1 1 -1 -1
2 2 1 -1 -1 -1 -1
3 3 1 -1 1 2 2
1 1 -1 -1 2 3 3
1 1 -1 -1 2 3 3
```

Output:

```
0 0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 0 0 0 0 0
1 1 0 0 0 0 0
0 0 0 0 0 0 0
1 0 0 0 0 1 1
0 0 0 0 0 1 0
```

Runtime:

- Brute-force: overload.
- Backtracking: 900ms
- pySat: 474ms
- A*: 450ms

c. 10x10:

Input:

```
10 10
-1 2 3 -1 -1 0 -1 -1 -1 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6
-1 -1 5 -1 5 3 -1 5 7 4
-1 4 -1 5 -1 5 -1 6 -1 3
-1 -1 4 -1 5 -1 6 -1 -1 3
-1 -1 -1 2 -1 5 -1 -1 -1 -1
4 -1 1 -1 -1 -1 1 1 -1 -1
4 -1 1 -1 -1 -1 1 -1 4 -1
-1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 4 4 -1 -1 -1 -1 4 -1 -1
```

Output:

```
0 1 1 0 0 0 0 0 1 1
0 0 0 1 0 0 0 1 1 1
0 0 1 1 1 0 0 1 1 1
0 1 1 0 1 1 0 1 0 0
0 1 0 0 0 1 1 1 1 0
1 1 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 0 1
1 0 0 0 1 0 0 0 0 1
1 1 0 0 1 1 0 0 1 1
0 1 1 1 1 1 1 1 1 0
```

Runtime:

- Brute-force: overload.
- Backtracking: 1774.8s
- pySat: 633ms
- A*: 440ms

d. 21x21:

Input:

```
21 21
-1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6
-1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4
-1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3
```

```
-1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3
-1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1
4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1
-1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6
-1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4
-1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3
-1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3
-1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1
4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1
-1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1
```

Output:

```
0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1
0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1
0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1
0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0
0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0
1 1 0 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1
1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1
1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0
1 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 1 1 0 1 1
0 1 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1
0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1
0 0 1 1 1 0 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1
0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 0 0
0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0
1 1 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0 0 1
1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 1
1 1 0 0 1 1 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1
0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 0
```

Runtime:

- Brute-force: overload.
- Backtracking: time exceeded.
- pySat: 625ms
- A*: 500ms

e. 43x43:

Input:

```
43 43
-1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1 -1
-1 2 3 -1 -1 0 -1 -1 -1 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6
-1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1
5 3 -1 5 7 4
-1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1
5 -1 5 -1 6 -1 3
-1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1
-1 4 -1 5 -1 6 -1 -1 3
-1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
-1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1
1 -1 -1 -1 1 1 -1 -1
4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1
1 -1 -1 -1 1 -1 4 -1
-1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1 -1
-1 4 4 -1 -1 -1 -1 4 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1 -1
-1 2 3 -1 -1 0 -1 -1 -1 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6
-1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1
5 3 -1 5 7 4
-1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1
5 -1 5 -1 6 -1 3
-1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1
-1 4 -1 5 -1 6 -1 -1 3
-1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
-1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1
1 -1 -1 -1 1 1 -1 -1
```

```
4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1
1 -1 -1 -1 1 -1 4 -1
-1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1 -1
-1 4 4 -1 -1 -1 -1 4 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1 -1 -1 2 3 -1 -1 0 -1 -1 -1 -1 -1
-1 2 3 -1 -1 0 -1 -1 -1 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6 -1 -1 -1 -1 -1 3 -1 2 -1 -1 6 -1
-1 -1 -1 -1 3 -1 2 -1 -1 6
-1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1 5 3 -1 5 7 4 -1 -1 -1 5 -1
5 3 -1 5 7 4
-1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1 5 -1 5 -1 6 -1 3 -1 -1 4 -1
5 -1 5 -1 6 -1 3
-1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1 -1 4 -1 5 -1 6 -1 -1 3 -1 -1
-1 4 -1 5 -1 6 -1 -1 3
-1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1 -1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
-1 -1 -1 -1 2 -1 5 -1 -1 -1 -1
4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1 1 -1 -1 -1 1 1 -1 -1 -1 4 -1
1 -1 -1 -1 1 1 -1 -1
4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1 1 -1 -1 -1 1 -1 4 -1 -1 4 -1
1 -1 -1 -1 1 -1 4 -1
-1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4 -1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 -1 -1 -1 -1 6 -1 -1 -1 -1 4
-1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1 -1 -1 4 4 -1 -1 -1 -1 4 -1 -1 -1
-1 4 4 -1 -1 -1 -1 4 -1 -1
```

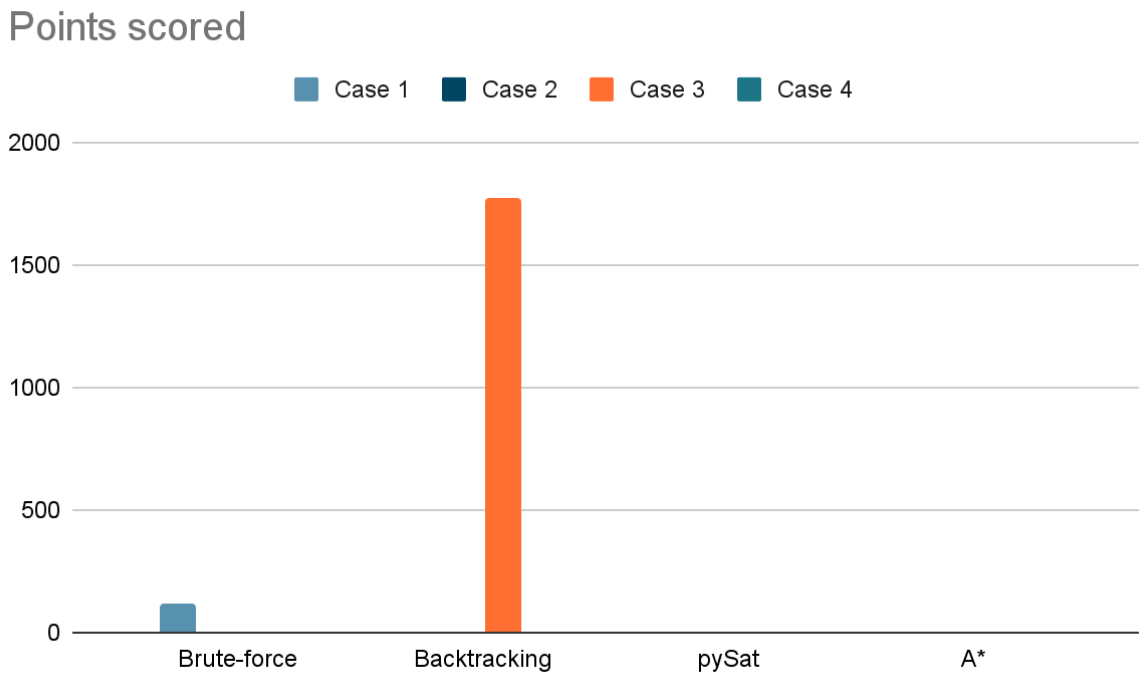
Output:

```
0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1
0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1
0 0 0 1 1 0 0 1 0 1 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 0 0 0 0 1 1 1 0 0 1 1 1
1 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 1 0 1 0 0
0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0
1 0 0 0 1 1 0 0 0 1 0 1 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1
1 0 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 1
0 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 1
0 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1
0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1
1 0 0 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1
0 1 1 1 1 0 1 1 0 1 0 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 0 0
1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0
1 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1
1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1
1 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0 0 1 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 1 0 0 1 1
0 0 0 1 1 1 0 0 1 0 0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 1 1 1 1 1 0
1 1 0 1 0 0 0 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1
0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1
1 1 0 1 1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0 0 0 1 1 1 0 0 1 1 1
0 0 1 1 1 0 1 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0
0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 1 1 0
1 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1
1 1 0 0 1 1 0 0 1 1 1 1 0 0 1 1 0 1 0 1 1 1 1 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 1 0 0 1 1
```

```
0 0 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0
0 1 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1
0 1 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 1 1
1 0 0 1 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 0 0 1 0 1 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1
0 0 1 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 0
1 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0
1 1 0 0 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 1 0 0 1 1 0 0 1 1
1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1
1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 1
0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0
```

Runtime:

- Brute-force: overload.
- Backtracking: time exceeded.
- pySat: 652ms
- A*: 700ms



- Through the test cases, it is easy to see the speed of pySat and A * is very fast and not too much difference. At the same time, their runtime through the Cases tests also does not have much differences despite the increased size of the Problem up quite large.
- In contrast, the running speed of Brute-Force and Backtracking is quite large, but in simple cases, backtracking resolves much faster.

V. Conclusion:

- Through this lab, we gained more knowledge about applying the lessons in theory class. We know one more SFML in graphics.
- There are many other projects around, so the time spent on this project is not enough to develop further. Within the scope of the project, we also tried the possible methods and referred to many other articles/documents.
- To complete this lab, thanks to the teachers with the enthusiastic and dedicated teaching assistants being so kind with us in this subject, we wish you health and joy in your work-teaching at HCMUS.

VI. Reference:

➢ https://www.geeksforgeeks.org/printing-solutions-n-queen-problem/
➢ https://www.freecodecamp.org/news/brute-force-algorithms-explained/
➢ https://www.geeksforgeeks.org/a-search-algorithm/
➢ https://stackoverflow.com/questions/11747254/python-brute-force-algorithm
➢ https://www.geeksforgeeks.org/a-search-algorithm/
➢ https://viblo.asia/p/a-search-algorithm-aWj53BN1l6m
➢ https://pysat.readthedocs.io/en/latest/installation.html
➢ https://pysathq.github.io/installation.html