TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHTN

ĐHQG-HCM

Project 02: Hashiwokakero

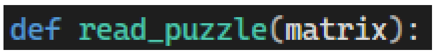
Thành viên:

21127006	-	Nguyễn Quốc Anh
20127347	-	Nguyễn Đình Thuận
20126018	-	Huỳnh Gia Khiêm
21126050	-	Nguyễn Hoàng Anh

I. Work assignment table & Self-evaluation

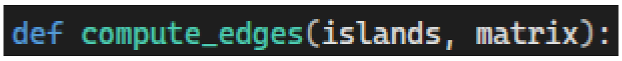
Criteria	Scores	Self-evaluation completion rate	Student ID
Solution description: Describe the correct logical principles for generating CNFs.	30%	100%	20127347
Generate CNFs automatically	10%	100%	20127347
Use the PySAT library to solve CNFs correctly.	10%	100%	21127006
Implement A* to solve CNFs without using a library.	10%	100%	20127347
Implement additional algorithms for comparison: 1) Brute-force algorithm to compare with A* (speed); 2) Backtracking algorithm to compare with A* (speed).	10%	100%	20127347
Documentation and analysis: 1) Write a detailed report (30%); 2) Thoroughness in analysis and experimentation; 3) Provide at least 10 test cases with different sizes (7× 7, 9×9, 11×11, 13×13, 17 ×17, 20×20) to verify your solution; 4) Compare results and performance.	30%	100%	21127006
Total	100%	100%	

II. Detailed explanation of each algorithm



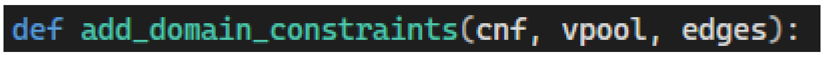
```
def read_puzzle(matrix):
```

- **read_puzzle()** reads the matrix and returns a list of dictionary of informations about each island: their position (row & col) and the amount of bridges that must be connected to them
- **Input**: matrix
- **Output**: dictionary of (row, col, number_of_bridges)



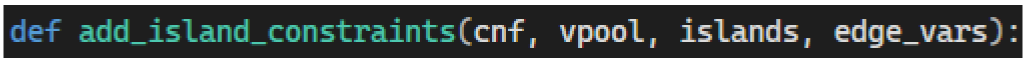
```
def compute_edges(islands, matrix):
```

- **compute_edges()** verifies the validity of the connections between each island
- Input:
  - list of islands and their data (row, col, number_of_bridges)
  - matrix
- Output:
  - final_edges: a list of tuples (id1, id2, extra). extra will be:
    - ('h', r, c_start, c_end) if the bridge is on the x-axis
    - ('v', c, r_start, r_end) if the bridge is on the y-axis
  - coord_to_id: converted value from (row, col) to island_id.



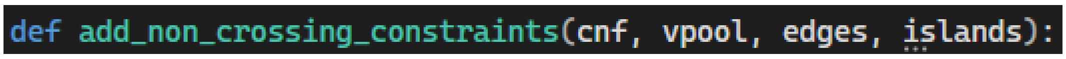
```
def add_domain_constraints(cnf, vpool, edges):
```

- **add_domain_constraints()** generates two outputs for every edge:
  + x_e1: is enabled when there is a minimum of one bridge
  + x_e2: is enabled when and only when x_e1 is enabled (which means there is at least one bridge)
- Input: CNF, vpool (from pysat library) and edges
- Output: edge_vars (a dictionary that maps each edge to a pair of bridge variables)



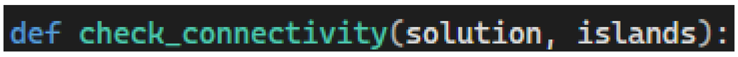
```
def add_island_constraints(cnf, vpool, islands, edge_vars):
```

- **add_island_constraints()** makes sure that the number of bridges connected to each island is equal to the predetermined value.
- For each edges connected to the island: value x_e1 enabled means +1 bridge and value x_e2 enabled means another +1 bridge



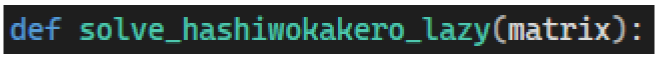
```
def add_non_crossing_constraints(cnf, vpool, edges, islands):
```

- **add_non_crossing_constraints()** ensures that no bridges cross over each other (value x_e1 of these bridges is not simultaneously enabled)



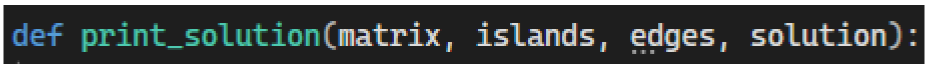
```
def check_connectivity(solution, islands):
```

- **check_connectivity()** checks whether the bridges connect the islands into a single connected group using DFS



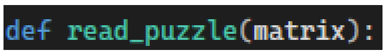
```
def solve_hashiwokakero_lazy(matrix):
```

- **solve_hashiwokakero_lazy()** generates CNFs automatically from the input matrix and uses the PySAT library to solve CNFs correctly
- The function then verifies the number of bridges and whether they crossed each other; if not, continue solving the puzzles.
- Finally, it returns a dictionary of the number of edges, the islands and the edges.



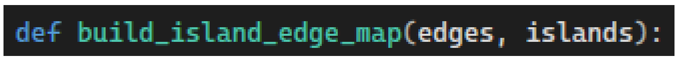
```
def print_solution(matrix, islands, edges, solution):
```

- **print_solution()** prints the output matrix to the debug screen, with zero as empty spaces and any other larger number representing islands and the number of bridges required to connect to them. For bridges:
    + "|" means one vertical bridge
    + "$" means two vertical bridges
    + "-" means one horizontal bridge
    + "=" means two horizontal bridges



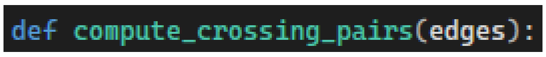
```
def read_puzzle(matrix):
```

- **read_puzzle()** reads the puzzle from a matrix and returns a dictionary of island IDs with their coordinates and number of required bridges.



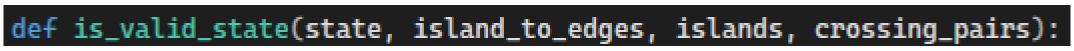
```
def build_island_edge_map(edges, islands):
```

- **build_island_edge_map()** builds a mapping from each island to the list of edges connected to it.



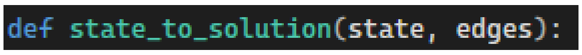
```
def compute_crossing_pairs(edges):
```

- **compute_crossing_pairs()** calculates whether pairs of edges cross each other (if yes, they cannot both have bridges) and returns a list of tuples representing the indices of crossing edges



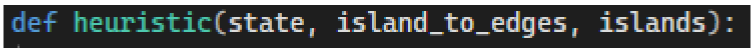
```
def is_valid_state(state, island_to_edges, islands, crossing_pairs):
```

- **is_valid_state()** checks if the list representing the number of bridges on each edge is valid (does not exceed the predetermined number and that no bridges cross each other).



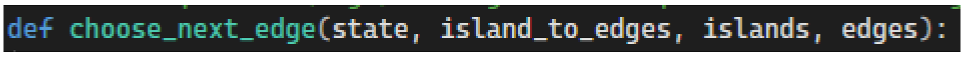
```
def state_to_solution(state, edges):
```

- **state_to_solution()** converts a state into a dictionary (maps tuples of island IDs to the number of bridges between them).



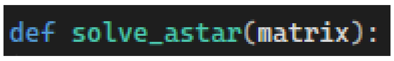
```
def heuristic(state, island_to_edges, islands):
```

- **heuristic()** calculates the number of bridges still needed to complete the map.



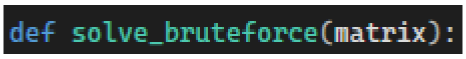
```
def choose_next_edge(state, island_to_edges, islands, edges):
```

- **choose_next_edge()** selects the next edge (with the smallest remaining required bridges) to assign a value to during the backtracking process.



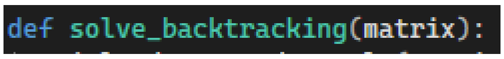
```
def solve_astar(matrix):
```

- **solve_astar()** implements the A* algorithm to solve the puzzle.



```
def solve_bruteforce(matrix):
```

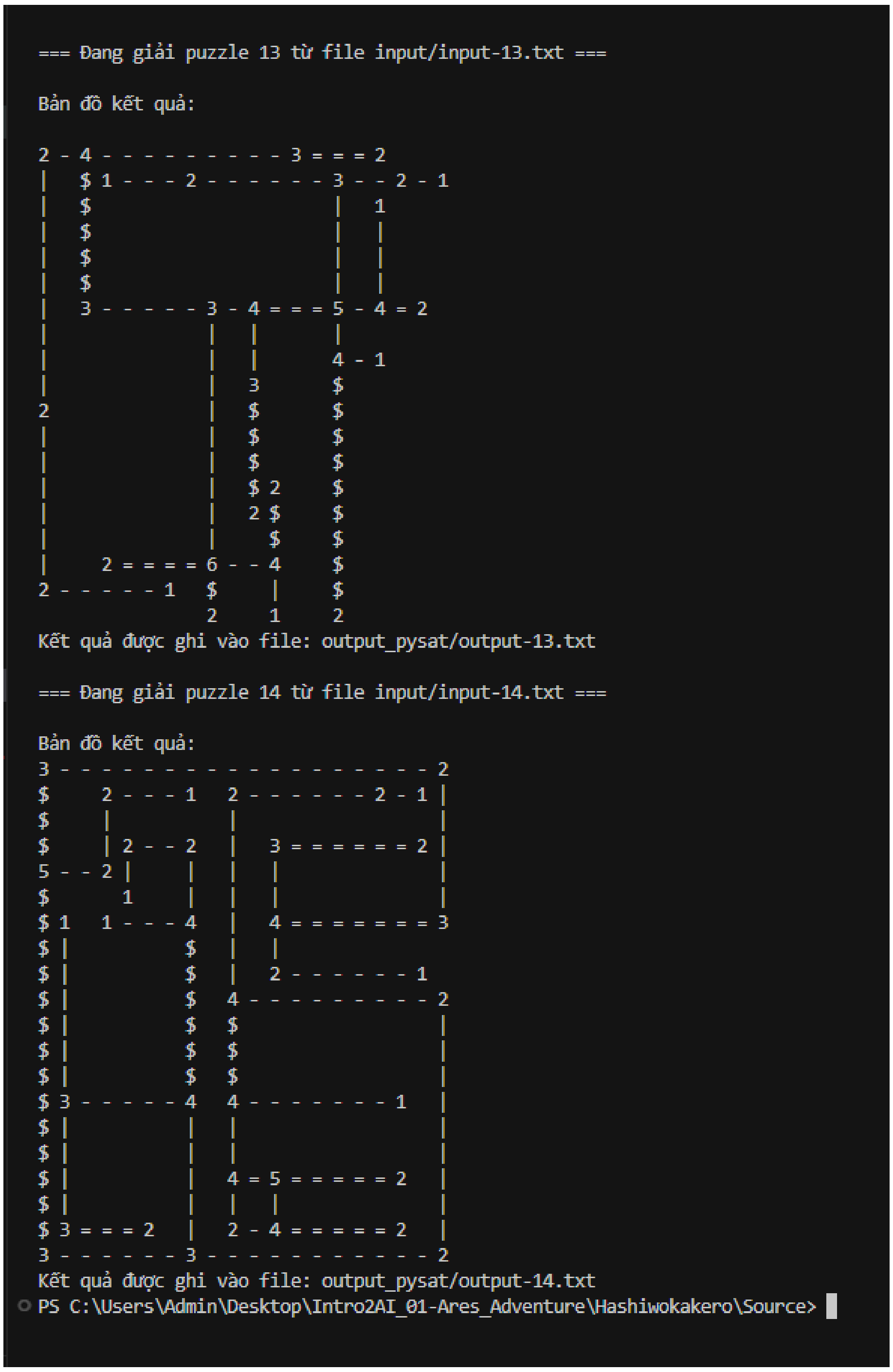
- **solve_bruteforce()** tries all possible combinations (brute-force) of bridges between islands to find a valid solution.
    + **recurse()** performs depth-first search to each edge and checks if the resulting state is valid. If a valid solution is found, it is stored in **solution_found**.



```
def solve_backtracking(matrix):
```

- **solve_backtracking()** implements a backtracking approach with a heuristic to solve the puzzle.

III. Description of the test cases and experiment results



```
=== Đang giải puzzle 13 từ file input/input-13.txt ===

Bản đồ kết quả:

2 - 4 - - - - - - - - - - 3 = = = 2
|   $ 1 - - - 2 - - - - - - - 3 - - 2 - 1
|   $                       |   1
|   $                       |   |
|   $                       |   |
|   $                       |   |
|   3 - - - - - 3 - 4 = = = 5 - 4 = 2
|               |   |       |
|               |   |       4 - 1
|               |   3       $
2               |   $       $
|               |   $       $
|               |   $       $
|               |   $ 2     $
|               |   2 $     $
|               |     $     $
|     2 = = = = 6 - - 4     $
2 - - - - - 1   $     |     $
                2     1     2
Kết quả được ghi vào file: output_pysat/output-13.txt

=== Đang giải puzzle 14 từ file input/input-14.txt ===

Bản đồ kết quả:
3 - - - - - - - - - - - - - - - - - - 2
$     2 - - - 1   2 - - - - - - 2 - 1 |
$     |           |                   |
$     | 2 - - 2   |   3 = = = = = = 2 |
5 - - 2 |     |   |   |               |
$       1     |   |   |               |
$ 1   1 - - - 4   |   4 = = = = = = = 3
$ |           $   |   |
$ |           $   |   2 - - - - - - - 1
$ |           $   4 - - - - - - - - - 2
$ |           $   $                   |
$ |           $   $                   |
$ |           $   $                   |
$ 3 - - - - - 4   4 - - - - - - - 1   |
$ |           |   |                   |
$ |           |   |                   |
$ |           |   4 = 5 = = = = = 2   |
$ |           |   |   |               |
$ 3 = = = 2   |   2 - 4 = = = = = 2   |
3 - - - - - - 3 - - - - - - - - - - - 2
Kết quả được ghi vào file: output_pysat/output-14.txt
PS C:\Users\Admin\Desktop\Intro2AI_01-Ares_Adventure\Hashiwokakero\Source>
```

IV. Demonstration video: test case video