TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

fit@hcmus

VNUHCM - UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

BÁO CÁO

MÔN: CƠ SỞ TRÍ TUỆ NHÂN TẠO

Giảng viên: Lê Nhựt Nam

Sinh viên

23127116 – Nguyễn Quang Thái

23127254 – Nguyễn Thị Như Quỳnh

23127366 – Võ Lê Ngọc Hiếu

23127524 – Hình Diễm Xuân

Thành phố Hồ Chí Minh 2025 - 2026

Mục lục

1. Thông tin chung

1.1. Thông tin thành viên

MSSV	Họ tên	Email	Số điện thoại
23127116	Nguyễn Quang Thái	nqthai23@clc.fitus.edu.vn	0837848237
23127254	Nguyễn Thị Như Quỳnh	ntnquynh23@clc.fitus.edu.vn	0963745770
23127366	Võ Lê Ngọc Hiếu	vlnhieu23@clc.fitus.edu.vn	0792421305
23127524	Hình Diễm Xuân	hdxuan23@clc.fitus.edu.vn	0396723858

1.2. Bảng phân công công việc

MSSV	Công việc	Mức độ hoàn thành
23127116	-Backtracking algorithm -Tạo testcase -Tìm đảo lân cận -Phân tích thuật DPLL	Tốt
23127254	-Brute-force algorithm -Mô tả testcase -Điều kiện cầu chéo nhau -Nội dung Brute-force, Backtracking, A* -Nội dung hàm tổng thể -Phân tích thuật Brute-force -Report	Tốt
23127366	-Generate CNF -Điều kiện CNF -Tạo testcase -Solve Hashi use Pysat -Hàm biểu diễn ma trận dưới dạng ký tự -So sánh các thuật toán	Tốt
23127524	-A* algorithm -Nội dung 2.1 -Hàm lấy tọa độ các đảo -Nội dung hàm tổng thể -Phân tích thuật A*	Tốt

-Bảng trên liệt kê các thành viên chịu trách nhiệm chính cho từng nhiệm vụ cụ thể trong đồ án. Tuy nhiên trong quá trình làm việc, mọi người luôn sẵn sàng hỗ trợ lẫn nhau, cùng nhau thảo luận, đóng góp ý kiến và chia sẻ kinh nghiệm để giải quyết những vấn đề phát sinh cũng như hoàn thiện đồ án.

1.3. Tự đánh giá

Criteria	Scores
Solution description: Describe the correct logical principles for generating CNFs.	2
Generate CNFs automatically.	1
Use the PySAT library to solve CNFs correctly	1
Implement A* to solve CNFs without using a library	1
Implement additional algorithms for comparison: 1) Brute-force algorithm to compare with A* (speed); 2) Backtracking algorithm to compare with A* (speed).	1
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.	3

2. Nội dung chi tiết

2.1. Điều kiện bài toán Hashiwokakero

-Giữa 2 đảo có tối đa 2 cầu

-Tổng số cầu và số trên đảo phải bằng nhau

-Tránh các cầu cắt nhau

-Các đảo liên thông

Biểu diễn bằng CNF

x1: có ít nhất 1 cầu

x2: có đúng 2 cầu

- Giữa 2 đảo có tối đa 2 cầu

  -Nếu có 2 cầu thì có ít nhất 1 cầu, nên khi x2 đúng thì x1 cũng đúng

  x2=>x1 nên ta có

  ¬x2 V x1

  -Nếu không có ít nhất 1 cầu thì không có 2 cầu, nên khi x1 sai thì x2 cũng sai

  ¬x1 => ¬x2

  x1 V ¬x2

- Tổng số cầu và số trên đảo phải bằng nhau

  D_k: số trên đảo k

  i: các đảo kề, có thể liên kết với k

-Để đảm bảo tổng số cầu và số trên đảo bằng nhau, thì:

$\sum x1\_k\_i + x2\_k\_i = D_k$

❖ Tránh các cầu cắt nhau

x1_lj: có ít nhất 1 cầu giữa đảo i và đảo j

x1_kl : có ít nhất 1 cầu giữa đảo k và đảo l

-Để đảm bảo 2 cầu không cắt nhau thì chỉ có 1 trong 2 cầu được tồn tại

¬x1_lj V ¬x1_kl

❖ Các đảo liên thông

n: tổng số đảo

x1_ij: cầu giữa 2 đảo

-Đảm bảo trong n đảo phải có ít nhất n-1 cầu để đảm bảo các đảo liên thông nhau, nên ta cần ít nhất n-1 biến x1 có giá trị true

$\sum x1\_ij >= n-1$

2.2. Định dạng output

-"|" là cầu đơn dọc

-"$" là cầu đôi dọc

-"-" là cầu đơn ngang

-"=" là cầu đôi ngang

2.3. Hàm tổng thể

- **Class HashiGrid**

```
def __init__(self, filename)
```

-Khởi tạo các đối tượng (n_size, filename, n_islands, island_coords, digits)

+n_size: kích thước ma trận

+filename: tên tệp chứa dữ liệu đầu vào

+n_islands: số lượng đảo

+island_coords: danh sách tọa độ các đảo

+digits: danh sách số lượng cầu tối đa của mỗi đảo

```
def getInput(self)
```

→Đọc dữ liệu từ tệp đầu vào, xác định kích thước lưới, tìm tọa độ các hòn đảo, lưu số cầu tối đa của từng hòn đảo và tính tổng số hòn đảo trong lưới.

- **Hàm solutionSoTring()**

  -Tạo lưới rỗng (empty_grid) kích thước n_size × n_size, ban đầu chứa toàn "0"

  -Đặt giá trị cho các hòn đảo dựa trên danh sách tọa độ và số cầu tối đa.

  -Xác định và vẽ cầu giữa các đảo từ bridge_count:

  +Tìm các ô giữa hai đảo và hướng cầu bằng coordinates_between().

  +Đặt ký hiệu cầu: Ngang: "-" (1 cầu), "=" (2 cầu); Dọc: "|" (1 cầu), "$" (2 cầu).

  -Trả về ma trận kết quả, biểu diễn ma trận dưới dạng ký tự.

- **Hàm adjacent_islands()**

  -Lấy tọa độ hòn đảo cần xét từ danh sách island_coords.

  -Duyệt qua tất cả hòn đảo khác để tìm hòn đảo gần nhất theo bốn hướng:

  +Trên: cùng cột, nhỏ hơn theo hàng.

  +Dưới: cùng cột, lớn hơn theo hàng.

  +Trái: cùng hàng, nhỏ hơn theo cột.

  +Phải: cùng hàng, lớn hơn theo cột.

  -Lưu hòn đảo gần nhất mỗi hướng (nếu có).

  -Trả về danh sách chỉ mục của các đảo lân cận.

  → Tìm các hòn đảo lân cận của một hòn đảo trong ma trận, tức là những hòn đảo gần nhất theo bốn hướng chính (trên, dưới, trái, phải), không bị chặn bởi đảo khác.

- **Hàm coordinates_between()**

  -Kiểm tra nếu hai hòn đảo nằm trên cùng một hàng (cầu ngang):

  +Lấy hàng x của hai đảo.

  +Duyệt các cột giữa hai đảo để tạo danh sách tọa độ (coordinates).

  +Đặt is_horizontal = True.

  -Kiểm tra nếu hai hòn đảo nằm trên cùng một cột (cầu dọc):

  +Lấy cột y của hai đảo.

  +Duyệt các hàng giữa hai đảo để tạo danh sách tọa độ (coordinates).

  +Đặt is_horizontal = False.

  -Trả về danh sách tọa độ và kiểu cầu (ngang/dọc).

→ Xác định các ô nằm giữa hai hòn đảo trong ma trận để kiểm tra hoặc đặt cầu nối giữa chúng.

- **Hàm intersect()**

-Lấy tọa độ các ô giữa hai cầu bằng coordinates_between():

+coords_under_a: tọa độ giữa hòn đảo a và b.

+coords_under_b: tọa độ giữa hòn đảo b và c.

-Kiểm tra tính hợp lệ của các cầu: nếu một trong các danh sách tọa độ rỗng, trả về False.

-Kiểm tra loại cầu (ngang/dọc): nếu cầu có hướng khác nhau (một ngang, một dọc), kiểm tra xem chúng có giao nhau không bằng phép giao giữa hai danh sách tọa độ.

-Trả về:

+True nếu hai cầu giao nhau tại ít nhất một ô.

+False nếu không giao nhau hoặc không cùng loại cầu.

- **Hàm generate_cnf()**

-Tạo biến nối cầu:

+Duyệt qua từng cặp đảo liền kề trong h_grid.

+Tạo biến x1 cho ít nhất một cầu và x2 cho chính xác hai cầu.

+Thêm ràng buộc: Nếu có hai cầu (x2 đúng), thì ít nhất phải có một cầu (x1 đúng).

-Ràng buộc tổng số cầu:

+Tạo danh sách biến liên quan đến số cầu của mỗi đảo.

+Sử dụng CardEnc.equals để đảm bảo tổng số cầu đi từ đảo khớp với giá trị yêu cầu (h_grid.digits[i]).

-Tránh cầu giao nhau:

+Kiểm tra từng cặp cầu để xác định xem chúng có giao nhau không.

+Nếu có, thêm ràng buộc để không thể tồn tại cả hai cầu cùng lúc.

-Ràng buộc kết nối:

+Đảm bảo phần lớn các đảo được kết nối bằng cách yêu cầu ít nhất n_islands - 1 cầu tồn tại.

+Sử dụng CardEnc.atleast để mã hóa ràng buộc này.

- **Hàm PySAT_solver()**

  -Tạo một solver SAT sử dụng thư viện Glucose4 với cnf.clauses là tập hợp các mệnh đề CNF đã sinh ra từ generate_cnf().

  -Gọi Glucose4 để giải hệ công thức CNF đã sinh ra (qua generate_cnf). Trả về True nếu có nghiệm hoặc False nếu vô nghiệm. Nếu true thì:

  +Lấy nghiệm từ SAT solver (ví dụ giá trị x: x là true, -x là false).

  +In ra màn hình dòng “Solution found".

  +Khởi tạo một dictionary rỗng (bridge_count) để lưu số lượng cầu giữa từng cặp đảo.

  +Vòng lặp qua từng đảo. Với x_vars là một dictionary chứa ảnh xạ từ cặp đảo i, j. Biến

  -x1, x2 là biến SAT: x1 là có ít nhất 1 cầu, x2 là có đúng 2 cầu. Mỗi i, j sẽ duyệt qua một cặp biến x1, x2 tương ứng.

  +Nếu biến x2 trả giá trị true nghĩa là cầu đôi giữa cặp đảo i, j, gán giá trị cho bridge_count[(i,j)] = 2.

  +Nếu biến x1 trả về giá trị true, mà x2 trả về giá trị false (vì dùng else if nên x2 false mới chạy xuống điều kiện này) nghĩa là có cầu đơn, gán giá trị cho bridge_count[(i,j)] = 1.

  +Còn lại (x1, x2 trả giá trị false) thì không có cầu nối giữa 2 đảo.gán giá trị cho bridge_count[(i,j)] = 0.

  +Ta gọi hàm solutionToString để chuyển thành ma trận lời giải ra Output.

  +Trả về ma trận 2D vừa chuyển ở trên.

  -Còn nếu false thì:

  +In ra màn hình thông báo "No feasible solution found!"

  +Trả về None.

- **Hàm strategy_solver()**

  -model là một dictionary lấy giá trị từ solver_function(cnf), trả về giá trị (1 là đúng, 0 là sai, None).

  -Nếu không có lời giải thì:

  +In ra màn hình thông báo "No feasible solution found!".

  +Trả về giá trị None.

-Khởi tạo một dictionary rỗng (bridge_count) để lưu số lượng cầu giữa từng cặp đảo.

-Vòng lặp qua từng đảo. Với x_vars là một dictionary chứa ảnh xạ từ cặp đảo i, j. Biến x1, x2 là biến SAT: x1 là có ít nhất 1 cầu, x2 là có đúng 2 cầu. Mỗi i, j sẽ duyệt qua một cặp biến x1, x2 tương ứng.

+Nếu biến x2 trả giá trị true nghĩa là cầu đôi giữa cặp đảo i, j, gán giá trị cho bridge_count[(i,j)] = 2.

+Nếu biến x1 trả về giá trị true, mà x2 trả về giá trị false (vì dùng else if nên x2 false mới chạy xuống điều kiện này) nghĩa là có cầu đơn, gán giá trị cho bridge_count[(i,j)] = 1.

+Còn lại (x1, x2 trả giá trị false) thì không có cầu nối giữa 2 đảo.gán giá trị cho bridge_count[(i,j)] = 0.

-In ra màn hình thông báo "Solution found.".

-Ta gọi hàm solutionToString để chuyển thành ma trận lời giải ra Output.

Trả về ma trận 2D vừa chuyển ở trên.

2.4. A* algorithm

2.4.1 Khái niệm

-Thuật toán A* (A-star) là một thuật toán tìm kiếm và tối ưu hóa, thường được sử dụng để tìm kiếm đường đi ngắn nhất trong các đồ thị hoặc không gian tìm kiếm có trọng số. A* kết hợp giữa tìm kiếm theo chiều rộng (BFS) và tìm kiếm theo chiều sâu (DFS) để có được một phương pháp tìm kiếm hiệu quả và tối ưu.

2.4.2 Đặc điểm

-Tìm kiếm theo ưu tiên

-Kết hợp chi phí thực tế và chi phí dự đoán

-Đảm bảo được giải pháp tối ưu

2.4.3 Áp dụng giải CNF

class AStarNode: đại diện cho 1 node trong A*

def __init__(self, f_value, g_value, current_assignment): khởi tạo các giá trị cho node.

f_value: chi phí dự đoán tổng (fx).

g_value: chi phí thực tế từ đầu đến vị trí hiện tại (gx).

Current_assignment: biến ánh xạ.

def __lt__(self, other): hàm so sánh nhỏ hơn, dùng để sắp xếp heap.

def __eq__(self, other): so sánh bằng.

def __repr__(self): định nghĩa cách hiển thị đối tượng khi in của node.

def a_star_cnf(cnf):

Sắp xếp các biến trong CNF theo tần suất xuất hiện giảm dần, biến nào xuất hiện nhiều nhất được ưu tiên. Duyệt toàn bộ các mệnh đề (clause) và literal (lit) trong mỗi mệnh đề, loại bỏ dấu và các biến trùng lặp, để trả về biến, vd: CNF gồm [[1, -2], [-1, 3], [2, 3]] thì kết quả sẽ là {1, 2, 3}. sum(1 for clause in ... if abs(lit) == v) tính số lần biến v xuất hiện trong CNF (bất kể dấu).

Khởi tạo dictionary cho các biến trong CNF.

def unit_propagation(assignment): đơn giản hóa bài toán

Khởi tạo biến changed = true.

Vòng lặp sẽ chạy nếu có thay đổi trong quá trình gán biến.

changed = False.

Duyệt qua từng mệnh đề trong công thức CNF.

unassigned: chứa các literal chưa được gán giá trị.

satisfied: đánh dấu nếu mệnh đề này đã được thỏa.

Duyệt qua từng literal trong mệnh đề hiện tại.

Lấy giá trị tuyệt đối của lit.

Nếu biến đã được gán bằng 1 hoặc -1 thì:

Nếu lit là giá trị dương thì biến được gán true, nếu âm thì biến được gán false.

Break.

Nếu biến chưa được gán thì đưa vào danh sách unassigned.

Nếu mệnh đề đã thỏa mãn thì bỏ qua mệnh đề này.

Nếu không còn literal nào chưa được gán giá trị trong một mệnh đề thì nghĩa là mệnh đề không thể thỏa mãn, ta trả về false để thông báo.

Nếu chỉ còn một literal chưa được gán giá trị trong mệnh đề, ta có thể gán giá trị cho literal đó mà không cần phải suy luận thêm:

Chọn literal chưa gán giá trị đầu tiên trong mệnh đề.

Lấy giá trị tuyệt đối của lit để xác định biến mà literal này đại diện.

Nếu lit là dương, ta gán giá trị true (1) cho biến. Nếu lit là âm, ta gán giá trị false (-1) cho biến.

Nếu biến assignment[var] chưa được gán giá ta gán giá trị đã tính được vào biến đó. Đánh dấu biến changed thành true.

Còn nếu biến đã được gán một giá trị khác với giá trị mới (new_val), tức là có mâu thuẫn, ta trả về false.

Return true.

def pure_literal_elimination(assignment): loại bỏ literal thuần túy.

Khởi tạo dictionary.

Duyệt qua tất cả các mệnh đề.

Duyệt qua tất cả các literal trong một mệnh đề.

Biến var là giá trị tuyệt đối của lit.

Nếu var xuất hiện trong literal_sign{} thì:

Nếu dấu của literal hiện tại khác với dấu trong literal_sign thì đánh dấu là None.

Còn lại thì: nếu lit > 0 thì gán bằng true, lit < 0 thì gán bằng false.

Duyệt qua các phần từ trong literal_sign, với mỗi phần tử là 1 cặp var, sign.

Nếu sign không là None và assignment hiện tại bằng 0:

Nếu sign là true thì gán assignment[var] = 1 nếu không gán assignment[var] = -1.

Trả về các biến đã được xác định.

def heuristic(assignment): tính giá trị heuristic.

Biến đếm số lượng các clause chưa được thoả mãn trong trạng thái hiện tại của assignment.

Duyệt qua tất cả các clause trong CNF.

kiểm tra xem có ít nhất một literal trong clause có giá trị đúng đối với biến của nó trong assignment.

Tăng số lượng clause chưa được thỏa mãn.

Trả về giá trị của unsatisfied.

def a_star():

Khởi tạo danh sách ưu tiên chứa các node trong quá trình tìm kiếm.

Khởi tạo node với giá trị (fx= gx + h, gx, current_assignment).

Thêm node mới tạo vào danh sách open_list được sắp xếp theo ưu tiên giá trị nhỏ nhất của fx.

Danh sách closed_list tập hợp các node đã được kiểm tra.

Duyệt cho đến kho open_list trống:

Khởi tạo biến lấy node của giá trị nhỏ nhất trong open_list.

Gọi hàm unit_propagation kiểm tra, nếu hàm trả về false thì bỏ qua node hiện tại.

Áp dụng loại bỏ các literal thuần túy.

Kiểm tra liệu tất cả các mệnh đề trong bài toán CNF có được thỏa mãn hay không, và nếu có, trả về lời giải hiện tại.

Duyệt các biến trong danh sách variables (các biến chưa được gán giá trị).

Nếu giá trị của biến hiện tại bằng 0 thì dừng việc tìm kiếm.

Nếu khác 0 thì tiếp tục tìm.

Tạo bản sao cho current_assignment (thử với trường hợp true).

Gán trạng thái mới bằng 1 (true).

Tính toán giá trị heuristic của trạng thái mới.

Kiểm tra xem trạng thái mới có nằm trong closed_list hay không, nếu không thì:

Tạo node mới với giá trị f_true, giá trị hiện tại, trạng thái mới (true).

Thêm node vào open_list.

Thêm trạng thái của biến mới vào closed_list để tránh duyệt lại.

Tạo bản sao cho current_assignment (thử với trường false).

Gán trạng thái mới bằng 1 (true).

Tính toán giá trị heuristic của trạng thái mới.

Kiểm tra xem trạng thái mới có nằm trong closed_list hay không, nếu không thì:

Tạo node mới với giá trị f_true, giá trị hiện tại, trạng thái mới (false).

Thêm node vào open_list.

Thêm trạng thái của biến mới vào closed_list để tránh duyệt lại.

Trả về None khi không thấy giải pháp.

Trả về kết quả của thuật toán A* từ hàm a_star().

2.5. Brute-force algorithm

2.5.1 Khái niệm

-Thuật toán brute-force (tấn công toàn bộ) là một phương pháp giải quyết vấn đề đơn giản bằng cách thử tất cả các khả năng có thể có và chọn ra giải pháp tốt nhất.

2.5.2 Đặc điểm

-Đơn giản và dễ hiểu

-Thử tất cả các khả năng

-Hiệu quả thấp, tốn thời gian

-Không cần tối ưu hóa

-Đảm bảo tìm ra giải pháp

-Dễ triển khai

2.5.3 Áp dụng giải CNF

-Các bước thực hiện:

+Thu thập tất cả biến xuất hiện trong CNF

- Các biến được trích xuất từ tất cả các mệnh đề trong công thức CNF.
- Hàm abs(lit) đảm bảo rằng mỗi biến chỉ xuất hiện một lần, bất kể dưới dạng dương hay phủ định.

+Sinh tất cả các phép gán

- Mỗi biến có 2 giá trị khả dĩ: 1 (True) hoặc -1 (False).
- Dùng itertools.product([1, -1], repeat=n) để sinh toàn bộ hoán vị các phép gán.

+Unit Propagation:

- Unit Clause: Là mệnh đề chỉ chứa một literal, ví dụ [3] hoặc [-2].
- Mục đích: Tự động gán giá trị phù hợp cho biến trong mệnh đề đơn vị để mệnh đề đó được thoả mãn.
- Hiệu quả: Giảm số biến chưa gán, phát hiện mâu thuẫn sớm.
- Nếu mâu thuẫn xảy ra: Bỏ qua phép gán hiện tại.

+Pure Literal Elimination:

- Literal thuần (Pure Literal): Là literal xuất hiện duy nhất dưới một dấu (toàn dương hoặc toàn âm) trong công thức CNF.
- Mục đích: Nếu một literal luôn xuất hiện dưới dạng x, ta có thể gán x=True để loại bỏ tất cả các mệnh đề chứa x.
- Ý nghĩa: Giúp rút gọn công thức và tránh xung đột không cần thiết.

+Duyệt từng phép gán

- Với mỗi phép gán, gán giá trị tương ứng cho từng biến.
- Áp dụng Unit Propagation để suy diễn thêm và phát hiện mâu thuẫn sớm.
- Nếu không mâu thuẫn, tiếp tục áp dụng Pure Literal Elimination để đơn giản hoá công thức.

+Kiểm tra tính thỏa mãn của CNF

- Với mỗi mệnh đề, kiểm tra xem có ít nhất một literal được thoả mãn trong phép gán hiện tại hay không.

- lit > 0: literal là biến dương, được thoả mãn nếu assignment[lit] == 1.
- lit < 0: literal là biến âm, được thoả mãn nếu assignment[abs(lit)] == -1.

- Nếu tất cả mệnh đề đều được thoả mãn, trả về phép gán đó là nghiệm.

+Kết luận: Nếu đã thử tất cả các phép gán mà không tìm được nghiệm, trả về None.

-Ưu và nhược điểm:

Ưu điểm	Nhược điểm
Đảm bảo tìm được nghiệm nếu tồn tại (tìm toàn bộ không gian)	Rất chậm với số biến lớn
Đơn giản, dễ cài đặt	Không phù hợp với bài toán CNF lớn
Kết hợp các heuristics giúp loại bỏ phép gán sai sớm	Không tận dụng đầy đủ tiềm năng của SAT solver hiện đại

2.6. Backtracking algorithm

2.6.1 Khái niệm

-Thuật toán backtracking là một phương pháp tìm kiếm và tối ưu hóa giải pháp trong không gian tìm kiếm, sử dụng nguyên lý "thử và sai". Thuật toán này liên tục thử nghiệm các lựa chọn, quay lại (backtrack) khi gặp phải một lựa chọn không hợp lý hoặc không thể tiếp tục được, và tiếp tục thử các lựa chọn khác cho đến khi tìm được giải pháp hoặc xác nhận rằng không có giải pháp nào tồn tại.

2.6.2 Đặc điểm

-Khám phá theo chiều sâu

-Chạy thử và quay lại

-Tối ưu hóa nhánh không khả thi

-Tìm kiếm giải pháp tất cả hoặc giải pháp tốt nhất

-Tối ưu cho bài toán có không gian giải pháp lớn

-Độ phức tạp thời gian cao

2.6.3 Áp dụng giải CNF

-Nếu ta giải bài toán SAT với thuật toán backtracking bình thường, thì ta sẽ lần lượt thử thế và kiểm tra từng giá trị, từng trường hợp của tất cả mệnh đề trong cnf

-Vấn đề xảy ra : Mệnh đề CNF của bài toán Hashiwokakero là quá lớn vì nó tăng lên theo cấp số nhân với kích thước và độ phức tạp của bài toàn cần giải.

-Điều đó đồng nghĩa với việc tốn quá nhiều tài nguyên (thời gian, bộ nhớ) cho việc giải bài toán Hashiwokakero với thuật toán backtracking cho dù là ở các bản đồ nhỏ như 5x5 hay 7x7.

-Vậy việc dùng thuật toán Backtracking để giải bài toán này là không hợp lí.

-Vậy dùng cách nào để có thể giải được CNF của bài toán Hashiwokakero này.

-Qua thời gian tìm hiểu, tìm kiếm trên Internet cũng như hỏi qua chat box AI (Deepseek, Chat GPT)

-Chúng em đã tìm được 1 dạng thuật toán cải tiến của thuật toán backtracking.

-Davis-Putnam-Logemann-Loveland (DPLL). DPLL hoạt động dựa trên hai kỹ thuật chính:

1. Quyết định (Decision): Gán giá trị cho một biến chưa xác định.
2. Lan truyền đơn vị (Unit Propagation): Áp dụng các phép gán bắt buộc khi chỉ còn một lựa chọn.
3. Rút gọn công thức (Simplification): Loại bỏ các mệnh đề đã thỏa mãn hoặc không thể thỏa mãn.

-Nếu phát hiện mâu thuẫn (có mệnh đề trở thành false), DPLL sẽ quay lui (backtrack) và thử giá trị khác.

-Các bước thực hiện của DPLL:

1. Điều kiện dừng:

- Cũng như backtracking
- Nếu tất cả các mệnh đề đều true → Công thức thỏa mãn được (SAT).
- Nếu có một mệnh đề false → Không thỏa mãn (UNSAT), cần quay lui.

2. Lan truyền đơn vị (Unit Propagation)

- Nếu một mệnh đề chỉ còn 1 biến chưa gán giá trị thì mệnh đề sẽ gán giá trị cho biến còn lại để mệnh đề đúng. VD ($A \vee \neg B$ với A = False, thì phải gán B=False để mệnh đề đúng).

- Lặp lại bước này đến khi không còn mệnh đề đơn vị (mệnh đề còn 1 biến chưa gán giá trị)

3. Chọn biến để quyết định (Decision)

- Chọn 1 biến chưa được gán giá trị và thử gán 1 giá trị cho biến đó. VD biến A chưa được gán giá trị thì sẽ thử gán A=True hoặc A=False.

4. Đệ quy để thử các trường hợp (Recursion)

- Gọi đệ quy DPLL với công thức đã được gán các giá trị. Nếu 1 nhánh xảy ra mâu thuẫn (nhánh thất bại), quay lui và thử gán giá trị ngược lại.

5. Quay lui (Backtracking)

- Nếu 1 nhánh thất bại, hủy các giá trị đã được gán và thử với các giá trị khác.

-Quay lại bài toán Hashiwokakero của nhóm em.

-Nhóm em đã triển khai thuật toán DPLL gồm các phần 1 Khởi tạo

- Để cho có thể tối ưu hiệu suất của dpll. Nhóm em đã sắp xếp các biến theo thứ tự xuất hiện giảm dần trong CNF

- Lợi ích: Giảm kích thước bài toán nhanh chóng, thông qua việc gán 1 gái trị có thể tác động đến nhiều mệnh đề. Từ đó có thể sẽ nhận được kết quả sớm hơn nếu mệnh đề thỏa mãn, hoặc phát hiện xung động sớm hơn nếu bài toán xảy ra mâu thuẫn
- 2 Hàm DPLL chính với các thành phần

a. pure_literal_elimination

o Được đùng để tìm các literal chỉ xuất hiện dưới 1 dạng là dương (A) hoặc âm (¬A) trong toàn bộ CNF, mà không xuất hiện cả 2

o Sau đó, gán các giá trị phù hợp để thoải mãn tất cả các mệnh đề chứa chúng.

b. unit_propagation

o Tìm các mệnh đề đơn vị và buộc gán giá trị cho chúng

o Trả về None nếu xảy ra mâu thuẫn

c. Điều kiện dừng

o Kiểu tra tất cả mệnh đề đã thỏa mãn chưa

o Với điều kiện thoải mãn là:

♣ Literal dương và biến được gán giá trị True

♣ Literal âm và biến được gán giá trị False

d. Chọn biến tiếp theo

- Chọn biến chưa được gán giá trị để gán giá trị
- Nếu không còn biến nào chưa được gán nhưng bài toán vẫn chưa thỏa mãn trả về None

e. Thử gán giá trị

- Lần lượt gán True và False và nếu không giá trị nào thỏa màn điều kiện dừng -> Quay lui

3. Mô tả testcase

-Input 1:

+Kích thước: 7x7

+Số lượng đảo: 18

+Độ khó: trung bình

- Có nhiều đảo có giá trị cao (4 và 5), yêu cầu nhiều cầu nối
- Một số đảo nằm rìa, hạn chế hướng kết nối
- Có nhiều khoảng trống giữa các đảo, làm tăng độ phức tạp trong việc tìm đường đi hợp lý

-Input 2:

+Kích thước: 7x7

+Số lượng đảo: 15

+Độ khó: trung bình

- Một số đảo có giá trị cao (3 và 4), yêu cầu nhiều cầu nối
- Các đảo phân bố khá đều, có nhiều khoảng trống giữa các đảo

-Input 3:

+Kích thước: 9x9

+Số lượng đảo: 20

+Độ khó: khó

- Nhiều đảo có giá trị cao (5 và 6), đòi hỏi nhiều cầu nối và tăng độ phức tạp
- Các đảo phân bố không đều, nhiều khoảng trống lớn ở trung tâm → tăng thách thức trong việc tìm đường đi hợp lý
- Một số đảo nhỏ (1 hoặc 2) có thể giúp tạo liên kết ban đầu, nhưng việc kết nối các đảo lớn sẽ đòi hỏi sự tính toán kỹ lưỡng

-Input 4:

+Kích thước: 9x9

+Số lượng đào: 19

+Độ khó: khó

- Nhiều đảo có giá trị cao (5 và 6), đòi hỏi nhiều cầu nối
- Một số đảo lớn nằm gần nhau → có thể gây xung đột trong việc bố trí cầu
- Một số đảo nhỏ (1 hoặc 2) có thể giúp xác định hướng cầu ban đầu, nhưng việc kết nối các đảo lớn sẽ đòi hỏi sự tính toán kỹ lưỡng

-Input 5:

+Kích thước: 11x11

+Số lượng đào: 23

+Độ khó: rất khó

- Nhiều đảo có giá trị cao (7 và 8), đòi hỏi nhiều cầu nối và có thể gây xung đột khi bố trí cầu
- Phân bố đảo khá dày đặc ở một số khu vực, nhưng cũng có những khoảng trống lớn → tăng thách thức trong việc kết nối hợp lý
- Một số đảo nhỏ (1 hoặc 2) có thể giúp xác định hướng cầu ban đầu, nhưng các đảo lớn cần được xử lý cẩn thận để tránh tạo xung đột hoặc đường cụt

-Input 6:

+Kích thước: 11x11

+Số lượng đào: 20

+Độ khó: khó

- Nhiều đảo có giá trị lớn như 5, yêu cầu nhiều cầu nối.
- Phân bố đảo không đều, tạo ra các khu vực khá dày đặc đảo và các khu vực còn lại có nhiều khoảng trống.
- Các đảo có giá trị cao gần nhau có thể gây xung đột khi cố gắng kết nối chúng, yêu cầu sự tính toán kỹ lưỡng.

→ Yêu cầu chiến lược rõ ràng và khả năng phân tích tình huống tốt để tìm ra đường đi hợp lý.

-Input 7:

+Kích thước: 13x13

+Số lượng đào: 28

+Độ khó: rất khó

- Các đảo có giá trị cao như 5, 6, và 7 đòi hỏi nhiều cầu nối, khiến việc kết nối các đảo trở nên phức tạp.
- Phân bố đảo khá dày đặc và có một số khu vực có rất ít khoảng trống, làm cho việc kết nối các đảo một cách hợp lý trở nên khó khăn.
- Các đảo giá trị nhỏ hơn (1 hoặc 2) có thể giúp xác định hướng kết nối ban đầu, nhưng việc kết nối chúng với các đảo lớn sẽ đòi hỏi chiến lược phức tạp.

➔ Yêu cầu khả năng tính toán và phân tích tốt để tìm ra các kết nối hợp lý mà không tạo ra xung đột.

-Input 8:

+Kích thước: 13x13

+Số lượng đào: 26

+Độ khó: rất khó

- Nhiều đảo có giá trị cao như 5, 6, đòi hỏi nhiều cầu nối và làm tăng độ phức tạp trong việc giải quyết.
- Phân bố đảo khá dày đặc, nhưng có một số khoảng trống lớn giữa các đảo, làm cho việc kết nối trở nên phức tạp.
- Các đảo giá trị nhỏ như 1 hoặc 2 có thể giúp xác định hướng kết nối ban đầu, nhưng các đảo lớn yêu cầu sự suy luận chi tiết hơn để tránh xung đột và đảm bảo tính hợp lệ.

➔ Đòi hỏi có chiến lược rõ ràng và khả năng phân tích tình huống tốt để xác định cách kết nối hợp lý giữa các đảo.

-Input 9:

+Kích thước: 17x17

+Số lượng đào: 37

+Độ khó: rất khó

- Nhiều đảo với giá trị cao như 5, 6, và 7, yêu cầu rất nhiều cầu nối, gây ra độ phức tạp cao.

- Các đảo được phân bố khá đồng đều trong lưới, nhưng có một số khu vực có nhiều đảo liên tiếp với các giá trị lớn, tạo nên nhiều khả năng kết nối nhưng cũng dễ dẫn đến xung đột.
- Các đảo nhỏ hơn (1 hoặc 2) có thể giúp xác định các hướng ban đầu, nhưng việc kết nối các đảo lớn sẽ đòi hỏi sự suy luận chi tiết và chiến lược hợp lý để tránh tạo đường cụt hoặc xung đột giữa các cầu.

➔ Do kích thước lưới lớn và số lượng đảo cao, testcase này đòi hỏi sự tính toán cẩn thận và khả năng tối ưu hóa trong việc nối các cầu một cách hợp lý

-Input 10:

+Kích thước: 17x17

+Số lượng đảo: 40

+Độ khó: rất khó

- Các đảo có giá trị cao như 5, 6 yêu cầu rất nhiều cầu nối và làm tăng độ phức tạp.
- Phân bố đảo khá đều trên toàn bộ lưới, nhưng có một số khu vực có nhiều đảo với giá trị lớn, đòi hỏi phải có chiến lược kết nối chính xác để tránh xung đột.
- Các đảo có giá trị nhỏ (1 và 2) có thể giúp làm điểm bắt đầu, nhưng việc kết nối các đảo lớn đòi hỏi một sự phân tích kỹ lưỡng.

➔ Số lượng đảo lớn và sự phân bố khá dày đặc, đặc biệt là ở các khu vực trung tâm và rìa, khiến cho testcase này có độ khó cao và yêu cầu một chiến lược tối ưu để giải quyết hiệu quả.

-Input 11:

+Kích thước: 20x20

+Số lượng đảo: 52

+Độ khó: rất khó

- Testcase này có kích thước lưới rất lớn và số lượng đảo khá cao, với nhiều đảo có giá trị lớn như 6, 7, 8, yêu cầu nhiều cầu nối và sự tính toán chính xác.
- Các đảo có giá trị nhỏ như 1, 2 có thể giúp tạo các mối nối ban đầu, nhưng việc kết nối các đảo lớn sẽ đòi hỏi rất nhiều suy luận và chiến lược tối ưu.
- Phân bố đảo trên lưới khá dày đặc và có một số khu vực không có đảo, tạo ra các khoảng trống cần phải kết nối hợp lý.

➔ Sự phân bố rộng lớn của các đảo và cầu nối đòi hỏi phải có một chiến lược giải quyết tốt để tránh lỗi và tìm ra cách nối tất cả các đảo một cách hợp lý mà không tạo ra xung đột.

-Input 12:

+Kích thước: 20x20

+Số lượng đảo: 58

+Độ khó: rất khó

- Kích thước lưới lớn với số lượng đảo khá nhiều, bao gồm các đảo có giá trị lớn như 4, 5 và 6, tạo ra nhiều sự kết nối và tính toán phức tạp.
- Các đảo có giá trị nhỏ như 1, 2 có thể giúp tạo các mối nối dễ dàng hơn, nhưng kết nối giữa các đảo lớn yêu cầu phải tìm ra các đường đi chính xác và tối ưu.
- Phân bố các đảo không quá dày đặc, nhưng có nhiều khu vực có thể gây khó khăn trong việc kết nối hợp lý, đặc biệt là các đảo lớn cần nhiều cầu nối và không có nhiều không gian xung quanh.

➔ Yêu cầu một chiến lược giải quyết tốt, vì số lượng cầu nối lớn và sự phân bố đảo phức tạp, điều này đòi hỏi sự tính toán cẩn thận để nối tất cả các đảo mà không vi phạm quy tắc.

4. So sánh các thuật toán

Bảng time: (s)

Thuật toán / Test	PySAT	A*	Backtrack (DPLL)	Brute-force
Test 1	0.0155	0.2875	0.2185	-
Test 2	0.0167	0.3631	0.3203	-
Test 3	0.0269	4.2843	3.1290	-
Test 4	0.0272	1.4391	1.2727	-
Test 5	0.0487	7.5947	6.7818	-
Test 6	0.0478	51.8486	46.5174	-
Test 7	0.0678	11.9052	10.8901	-
Test 8	0.0732	732.8758	604.5844	-
Test 9	0.1491	-	-	-
Test 10	0.1711	-	-	-
Test 11	0.2914	-	-	-
Test 12	0.3393	-	-	-
Test 1				

Bảng memory: (MB)

Thuật toán / Test	PySAT	A*	Backtrack (DPLL)	Brute-force
Test 1	0.70	0.75	0.28	-
Test 2	0.73	0.87	0.42	-
Test 3	1.09	15.77	2.88	-
Test 4	1.09	4.48	2.53	-
Test 5	1.73	17.27	6.60	-
Test 6	1.52	119.53	6.03	-
Test 7	2.09	27.74	10.12	-
Test 8	2.22	1358.08	24.16	-
Test 9	3.56	-	-	-
Test 10	3.70	-	-	-
Test 11	5.45	-	-	-
Test 12	5.45	-	-	-
Test 1				

5. Tham khảo

-Github: https://short.com.vn/FeUr

-Youtube, Chat GPT, DeepSeek

-https://arxiv.org/pdf/1905.00973