TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**
**KHOA CÔNG NGHỆ THÔNG TIN**


**Vũ Tuấn Anh – 520V0016**
**Thái Gia Bảo – 52000014**
**Nguyễn Quế Chi – 51800016**
**Nguyễn Cao Phúc Hải - 52200175**

# BÁO CÁO CUỐI KÌ
# NHẬP MÔN
# TRÍ TUỆ NHÂN TẠO

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**
**KHOA CÔNG NGHỆ THÔNG TIN**


**Vũ Tuấn Anh – 520V0016**
**Thái Gia Bảo – 52000014**
**Nguyễn Quế Chi – 51800016**
**Nguyễn Cao Phúc Hải - 52200175**

# BÁO CÁO CUỐI KÌ
# NHẬP MÔN
# TRÍ TUỆ NHÂN TẠO

Người hướng dẫn
**ThS. Nguyễn Thành An**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024**

# LỜI CẢM ƠN

Đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Thầy Nguyễn Thành An – Giảng viên khoa Công nghệ thông tin – Trường đại học Tôn Đức Thắng, đã hỗ trợ và giúp đỡ nhiệt tình trong quá trình thực hiện Dự án này.

Chúng em trân trọng cảm ơn Thầy Cô giảng viên Trường đại học Tôn Đức Thắng nói chung cũng như Thầy Cô giảng viên khoa Công nghệ thông tin nói riêng đã giảng dạy và truyền đạt nhiều kinh nghiệm quý trong suốt quá trình học tập tại trường.

Cuối cùng, xin cám ơn gia đình, bạn bè đã luôn động viên và đồng hành trong quá trình học tập cũng như quá trình thực hiện Dự án này.

Mặc dù rất cẩn thận trong quá trình thực hiện đồ án cũng như viết báo cáo nhưng cũng không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự góp ý từ các Thầy để đồ án được hoàn thiện hơn.

Xin chân thành cám ơn!

*TP. Hồ Chí Minh, ngày 6 tháng 5 năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Vũ Tuấn Anh*

*Thái Gia Bảo*

*Nguyễn Quế Chi*

*Nguyễn Cao Phúc Hải*

# CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
# TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của ThS. Nguyễn Thành An. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình**. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 6 tháng 5 năm 2024*

*Tác giả*

*(Ký tên và ghi rõ họ tên)*

*Vũ Tuấn Anh*

*Thái Gia Bảo*

*Nguyễn Quế Chi*

*Nguyễn Cao Phúc Hải*

# TÓM TẮT

Tóm tắt yêu cầu của bài đồ án:

**Câu 1: 8x8 Tic-Tac-Toe**

- Cài đặt trò chơi Tic-Tac-Toe trên bàn cờ 8x8, với mục tiêu là có 4 quân thẳng hàng để chiến thắng.
- Trò chơi hoạt động trên console, người chơi nhập toạ độ để chọn ô.
- Sử dụng thuật toán alpha-beta pruning để máy tính ra nước cờ.
- Tổ chức lớp Problem và lớp SearchStrategy để quản lý và thực hiện thuật toán.

**Câu 2: N-Queens with CNFs**

- Đặt N quân hậu trên bàn cờ kích thước N x N, N được nhập từ bàn phím.
- Gán biến logic cho mỗi ô, xác định ràng buộc và biểu diễn chúng dưới dạng CNFs.
- Sử dụng thư viện Glucose3 để tìm bộ giá trị cho các biến và suy ra đáp án của bài toán.

**Câu 3: Decision Trees**

- Sử dụng hai thư viện Pandas và Decision Tree (Scikit-learn).
- Thực hiện các yêu cầu như tính Entropy, Average Entropy và Information Gain cho các thuộc tính điểm số.
- Cài đặt, huấn luyện và đánh giá mô hình Decision Tree trên tập dữ liệu dt_data.csv, và trực quan hoá cấu trúc cây quyết định.

# MỤC LỤC

## Contents

# DANH MỤC HÌNH VẼ

# DANH MỤC BẢNG BIỂU

# DANH MỤC CÁC CHỮ VIẾT TẮT

| | |
|---|---|
| CNF | Conjuctive Normal Form |

# DANH SÁCH SINH VIÊN

| MSSV | Họ tên | Email | Phân công công việc | Mức độ hoàn thành |
|---|---|---|---|---|
| 520V0016 | Nguyễn Cao Phúc Hải | 520V0016@student.tdtu.edu.vn | Câu 1 | 100% |
| 51800016 | Nguyễn Quế Chi | 51800016@student.tdtu.edu.vn | Câu 2 | 100% |
| 52000014 | Thái Gia Bảo | 52000014@student.tdtu.edu.vn | Câu 2 | 100% |
| 52200175 | Vũ Tuấn Anh | 52200175@student.tdtu.edu.vn | Câu 3 | 100% |

# 1. Câu 1: 8x8 Tic-Tac-Toe

## 1.1. Giới thiệu

Bài toán 8x8 Tic-Tac-Toe là bài toán thuộc lĩnh vực Trí Tuệ Nhân Tạo. Bài toán được định nghĩa rằng, trên bàn cờ Tic-Tac-Toe sẽ có bên thi đấu đối kháng với nhau, một bên là người chơi và bên còn lại sẽ là máy tính.

Trong trò chơi, bàn cờ sẽ được thể hiện dưới dạng 8x8 và người chơi sẽ chọn nước đi cho mình bằng cách nhập tọa độ của hàng và cột vào màn hình Terminal, còn máy tính sẽ ra nước đi tự động dựa trên thuật toán Alpha Beta Prunning

## 1.2. Mô hình hóa vấn đề

### 1.2.1 Biến logic cho mỗi ô trên bàn cờ

Trong bài toán Tic Tac Toe 8x8, mỗi ô trên bàn cờ sẽ được gán hai biến logic để biểu diễn trạng thái của ô đó, hai biến logic đó là ‘X’ và ‘O’

Chúng ta sử dụng một ma trận 8x8, trong mỗi phần tử sẽ là hai biến logic

+ Biến Xij: tại hàng i, cột j biểu diễn biến logic ‘X’

+ Biến Oij: tại hàng i, cột j biểu diễn biến logic ‘O’

VD: Trong bàn cờ 8x8, biến logic cho ô trên hàng 3 cột 4 sẽ được kí hiệu là X34, O34

### 1.2.2 Ràng buộc ở mỗi ô trên bàn cờ

Mỗi ô trên bàn cờ chỉ có thể chứa một quân cờ ‘X’ hoặc ‘O’, hoặc không có quân cờ nào. Trong đó, với mỗi ô sẽ chỉ tồn tại chỉ một quân cờ duy nhất hoặc không tồn tại quân cờ nào, sẽ không có trường hợp hai quân cờ cùng chung một ô

## 1.3 Thực thi bài toán

```
class Problem:
    def __init__(self, board_size):
        self.board_size = board_size
        self.board = np.zeros((self.board_size, self.board_size), dtype =
int)

    def print_board(self):
        os.system('cls' if os.name == 'nt' else 'clear')
        for row in self.board:
            print(' '.join(map(str,row)).replace('0','-
').replace('1','X').replace('2','O'))
        print()
```

### 1.3.1 Khởi tạo bàn cờ

Trong lớp này có các **hàm __init__(self, board_size)** nhận tham số đầu vào là board_size ( kích thước chiều ngang và dọc của bàn cờ ), từ đó tạo ra mảng 2 chiều với tất cả các ô trống khởi tạo ban đầu đều được gán bằng 0. Hàm print_board(self) để in ra bàn cờ, trong đó có dòng lệnh ‘`os.system('cls' if os.name == 'nt' else 'clear')`‘ dùng để xóa bàn cờ sau mỗi lượt đánh để in ra bàn cờ mới

### 1.3.2 Cài đặt thuật toán kiểm tra nước đi và tạo nước đi

```
 def is_valid_move(self, row, col):
            return self.board[row, col] == 0

 def make_move(self,row,col,player):
            if self.is_valid_move(row,col):
                self.board[row, col] = player
                return True
            return False
```

Ở phương thức **is_valid_move (self, row, col)** kiểm tra xem nước đi có hợp lệ không ( ô đó phải có giá trị bằng 0, tức ô trống )

Ở phương thức **make_move (self,row,col,player)** để thực hiện nước đi của người chơi, nếu hợp lệ trả vè True, ngược lại trả về False.

### 1.3.3 Thuật toán kiểm tra kết quả

```
 def has_won(self, player):
            directions = [(1,0), (0,1), (1,1), (1,-1)]
            for row in range(self.board_size):
                for col in range(self.board_size):
                    if self.board[row,col] == player:
                        for direction in directions:
                            count = 1
                            for step in range(1,4):
                                new_row = row + direction[0] * step
                                new_col = col + direction[1] * step
                                if 0 <= new_row < self.board_size and 0 <=
new_col < self.board_size and self.board[new_row, new_col] == player:
                                    count += 1
                                else:
                                    break
                            if count == 4:
                                return True
            return False

 def is_draw(self):
            return not np.any(self.board == 0)
```

Với phương thức **has_won(self, player)** để kiểm tra người chơi có thắng hay không bằng cách kiểm tra các dãy 4 ô liên tiếp theo các hướng: dọc, ngang hoặc chéo. Phương thức này sẽ lặp qua từng ô trong bảng, nếu ô đó thuộc về người chơi thì kiểm tra 4 ô liên tiếp theo từng hướng, nếu 4 ô liên tiếp thuộc cùng nước giống nhau thì trả về **True**, ngược lại thì trả về **False**

Với phương thức **is_draw(self)** để kiểm tra xem trên bàn cờ còn ô nào trống không, nếu không còn ô nào trống thì trả về kết quả là hòa.

### 1.3.4 Thuật toán tìm kiếm, đưa ra nước đi cho máy tính

```
def alpha_beta_search(self, depth, alpha, beta, maximizing_player):
        if self.problem.has_won(1):
            return -10
        if self.problem.has_won(2):
            return 10
        if self.problem.is_draw() or depth == 0:
            return 0

        if maximizing_player:
            max_eval = -float('inf')
            for row in range(self.problem.board_size):
                for col in range(self.problem.board_size):
                    if self.problem.is_valid_move(row, col):
                            self.problem.board[row,col] = 2
                            eval = self.alpha_beta_search(depth - 1,
alpha, beta, False)
                            self.problem.board[row, col] = 0
                            max_eval = max(max_eval,eval)
                            alpha = max(alpha, eval)
                            if beta <= alpha:
                                break
            return max_eval
        else:
            min_eval = float('inf')
            for row in range(self.problem.board_size):
                for col in range(self.problem.board_size):
                    if self.problem.is_valid_move(row, col):
                        self.problem.board[row, col] = 1
                        eval = self.alpha_beta_search(depth-1,alpha, beta,
True)
                        self.problem.board[row, col] = 0
                        min_eval = min(min_eval, eval)
                        beta = min (beta, eval)
                        if beta <= alpha:
                            break
            return min_eval
```

Hàm **def alpha_beta_search(self, depth, alpha, beta, maximizing_player)** là thuật toán tìm kiếm cho máy tính, trong đó có các tham số

depth: Độ sâu hiện tại của thuật toán tìm kiếm.

**alpha**: Giá trị alpha trong thuật toán alpha-beta.

**beta**: Giá trị beta trong thuật toán alpha-beta.

**maximizing_player**: Biến boolean xác định lượt của người chơi (người chơi tối ưu hóa giá trị hay không).

Điều kiện kết thúc:

Nếu người chơi 1 thắng, trả về -10. Nếu người chơi 2 thắng, trả về 10. Nếu hòa hoặc độ sâu bằng 0, trả về 0. Trường hợp người chơi tối đa hóa giá trị: Khởi tạo giá trị max_eval là âm vô cực. Lặp qua các ô trên bảng trò chơi. Nếu nước đi hợp lệ, thực hiện nước đi cho người chơi 2, gọi đệ quy alpha_beta_search với độ sâu giảm 1, sau đó hủy nước đi. Cập nhật max_eval và giá trị alpha. Kiểm tra điều kiện cắt tỉa alpha-beta.

Trường hợp người chơi tối thiểu hóa giá trị: Khởi tạo giá trị min_eval là dương vô cực. Lặp qua các ô trên bảng trò chơi. Nếu nước đi hợp lệ, thực hiện nước đi cho người chơi 1, gọi đệ quy alpha_beta_search với độ sâu giảm 1, sau đó hủy nước đi. Cập nhật min_eval và giá trị beta.

```
def get_best_move(self):
        best_val = -float('inf')
        best_move = (-1,-1)
        for row in range(self.problem.board_size):
            for col in range(self.problem.board_size):
                if self.problem.is_valid_move(row,col):
                    self.problem.board[row, col] = 2
                    move_val = self.alpha_beta_search(self.search_depth, -
float('inf'), -float('inf'), False)
                    self.problem.board[row, col] = 0
                    if move_val > best_val:
                        best_move = (row, col)
                        best_val = move_val
   return best_move
```

Hàm **get_best_move:** Tìm kiếm nước đi tốt nhất cho người chơi 2 (người chơi tối đa hóa giá trị). Khởi tạo best_val là âm vô cực và best_move là một tuple (-1, -1). Lặp qua các ô trên bảng trò chơi. Nếu nước đi hợp lệ, thực hiện nước đi cho người chơi 2, gọi phương thức alpha_beta_search với độ sâu tìm kiếm, sau đó hủy nước đi. Cập nhật nước đi tốt nhất và giá trị tốt nhất nếu giá trị trả về lớn hơn best_val. Trả về nước đi tốt nhất.

## 2. Câu 2: N-Queens with CNFs

### 2.1. Giới thiệu

Bài toán N-Queens là một bài toán quen thuộc trong lĩnh vực Trí Tuệ Nhân Tạo, nơi mà mục tiêu chính là đặt N quân hậu trên một bàn cờ kích thước N x N sao cho không có hai quân hậu nào ăn được nhau.

Mục tiêu chính của bài toán là tìm ra một cách sắp xếp hợp lý của các quân hậu trên bàn cờ sao cho không có hai quân hậu nào ăn được nhau. Để đạt được mục tiêu này, chúng ta cần phải mô hình hóa vấn đề thành các biến logic và ràng buộc logic phù hợp.

## 2.2. Mô hình hóa vấn đề

### 2.2.1. Biến Logic cho mỗi ô trên bàn cờ

Trong bài toán N-Queens, mỗi ô trên bàn cờ sẽ được gán một biến logic để biểu diễn trạng thái của ô đó, tức là có quân hậu ở ô đó hay không. Biến logic này có thể mang hai giá trị:

***True***: Nếu ô đó có quân hậu.

***False***: Nếu ô đó không có quân hậu.

Để thực hiện việc gán biến logic cho mỗi ô trên bàn cờ, chúng ta sẽ sử dụng một ma trận có kích thước N x N, trong đó mỗi phần tử sẽ là một biến logic. Cụ thể:

Biến logic tại vị trí (i, j) trong ma trận sẽ biểu diễn ô trên hàng i và cột j trên bàn cờ.

Biến logic này sẽ được ký hiệu là $X_{ij}$, trong đó i và j là chỉ số hàng và cột tương ứng.

Ví dụ: Trong một bàn cờ có kích thước 4x4, biến logic cho ô trên hàng thứ 2 và cột thứ 3 sẽ được ký hiệu là $X_{23}$.

Qua việc gán biến logic cho mỗi ô trên bàn cờ, chúng ta có thể mô hình hóa trạng thái của bàn cờ và sử dụng các ràng buộc logic để giải quyết bài toán N-Queens.

### 2.2.2. Ràng buộc giữa các ô trên bàn cờ

Trong bài toán N-Queens, để đảm bảo rằng không có hai quân hậu nào ăn được nhau trên bàn cờ, chúng ta cần xác định và áp dụng các ràng buộc giữa các ô trên bàn cờ. Cụ thể:

**Ràng buộc cho các Hàng:**

- Mỗi hàng trên bàn cờ phải chứa đúng một quân hậu.
- Điều này có nghĩa là trong mỗi hàng, chỉ có một biến logic mang giá trị True, thể hiện việc có quân hậu ở một ô cụ thể.

**Ràng buộc cho các Cột:**

- Mỗi cột trên bàn cờ cũng phải chứa đúng một quân hậu.
- Tương tự như ràng buộc cho hàng, trong mỗi cột, chỉ có một biến logic mang giá trị True.

**Ràng buộc cho các Đường Chéo:**

- Không có hai quân hậu nào được phép nằm trên cùng một đường chéo.
- Để kiểm tra điều này, chúng ta sẽ xác định các đường chéo chính và phụ trên bàn cờ và áp dụng ràng buộc tương ứng.

### 2.2.3. Biểu diễn các ràng buộc thành CNFs

Trong bài toán N-Queens, sau khi đã thiết lập ràng buộc giữa các ô trên bàn cờ, chúng ta cần biểu diễn các ràng buộc này dưới dạng Conjunctive Normal Form (CNF). Dưới đây là một ví dụ cụ thể về quá trình biểu diễn một ràng buộc đơn giản thành dạng CNF.

Ví dụ: Giả sử chúng ta có bàn cờ kích thước 4x4. Chúng ta muốn biểu diễn ràng buộc: "Ô 1 có quân hậu nếu và chỉ nếu các ô 2, 3, 4 không có quân hậu."

Biểu diễn mệnh đề ban đầu: Mệnh đề ban đầu có dạng:

$$1 \leftrightarrow \neg 2 \wedge \neg 3 \wedge \neg 4 1 \leftrightarrow \neg 2 \wedge \neg 3 \wedge \neg 4.$$

Chuyển đổi thành CNFs: Bước đầu tiên là chuyển biểu thức về dạng chỉ sử dụng phép tuyển (OR) và phủ định (NOT). Áp dụng quy tắc De Morgan, ta có:

$$\neg(A \wedge B) \equiv \neg A \vee \neg B \neg(\mathrm{A} \wedge \mathrm{B}) \equiv \neg \mathrm{A} \vee \neg \mathrm{B}$$

$$\neg(A \vee B) \equiv \neg A \wedge \neg B \neg(\mathrm{A} \vee \mathrm{B}) \equiv \neg \mathrm{A} \wedge \neg \mathrm{B}$$

Áp dụng quy tắc này cho mệnh đề ban đầu, ta có:

$$(1 \vee 2 \vee 3 \vee 4)(1 \vee 2 \vee 3 \vee 4)$$

$$(\neg 1 \vee \neg 2) \wedge (\neg 1 \vee \neg 3) \wedge (\neg 1 \vee \neg 4)(\neg 1 \vee \neg 2) \wedge (\neg 1 \vee \neg 3) \wedge (\neg 1 \vee \neg 4)$$

Đây là dạng CNF của mệnh đề ban đầu. Quá trình này cho thấy cách chuyển đổi ràng buộc thành dạng CNF, một bước quan trọng để giải quyết bài toán N-Queens bằng các thuật toán giải quyết bài toán SAT.

## 2.3. Cài đặt và thực thi

### 2.3.1. Khởi tạo biến Logic cho mỗi ô trên bàn cờ

```
function initialize_variables(N):
      # Hàm này khởi tạo ma trận biến logic, mỗi ô trên bàn cờ sẽ được gán một biến logic.
      # Tạo ma trận N x N với các giá trị ban đầu là 0
      # Biến này dùng để đếm số biến logic đã được tạo
      # Gán mỗi ô trên bàn cờ một biến logic duy nhất
    return variables
```

Hàm này nhận đầu vào là một số nguyên N, đại diện cho kích thước của bàn cờ và số lượng quân hậu.

Tạo ra một ma trận có kích thước N x N, trong đó mỗi phần tử ban đầu được gán giá trị là 0.

Biến "**num_vars"** được sử dụng để đếm số biến logic đã được tạo.

Sử dụng hai vòng lặp for để duyệt qua từng ô trên bàn cờ và gán một biến logic duy nhất cho mỗi ô. Biến logic này được đánh số từ 1 đến $N^2$, với cách gán số từ trái qua phải, từ trên xuống dưới.

Ví dụ, nếu N = 4, ma trận biến logic sẽ có dạng sau:

```
[[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12],
 [13, 14, 15, 16]]
```

### 2.3.2. Tạo ràng buộc

```
function generate_constraints(N, variables):
    clauses = []  # Danh sách các mệnh đề logic (clauses)
      # Ràng buộc: Mỗi hàng phải có ít nhất một quân hậu
      # Ràng buộc: Mỗi cột chỉ có tối đa một quân hậu
      # Ràng buộc: Không có hai quân hậu nào tấn công nhau theo đường chéo
      # Quân hậu không thể nằm trên đường chéo xuống phải
      # Quân hậu không thể nằm trên đường chéo lên trái
      # Quân hậu không thể nằm trên đường chéo xuống trái
      # Quân hậu không thể nằm trên đường chéo lên phải
    return clauses
```

Hàm này nhận hai đối số: N là kích thước của bàn cờ và "**variables**" là ma trận biến logic đã được khởi tạo trước đó.

Biến "**clauses**" là danh sách các mệnh đề logic, sẽ chứa các ràng buộc giữa các ô trên bàn cờ.

Ràng buộc đầu tiên đảm bảo rằng mỗi hàng trên bàn cờ phải chứa ít nhất một quân hậu. Điều này được biểu diễn bằng cách thêm vào "**clauses"** một mệnh đề cho mỗi hàng, trong đó mỗi mệnh đề chứa các biến logic của các ô trong hàng đó.

Ràng buộc thứ hai đảm bảo rằng mỗi cột trên bàn cờ chỉ có tối đa một quân hậu. Điều này được thực hiện bằng cách thêm vào "**clauses"** các mệnh đề phủ định cho các cặp ô trên cùng một cột.Ràng buộc cuối cùng đảm bảo rằng không có hai quân hậu nào tấn công nhau theo đường chéo. Điều này được thực hiện bằng cách thêm vào **lauses"**

các mệnh đề phủ định cho các cặp ô trên cùng một đường chéo.**Giải bài toán N-Queens.**

```
function solve_n_queens(N):
    variables = initialize_variables(N)  # Khởi tạo biến logic cho bàn cờ
    clauses = generate_constraints(N, variables)  # Tạo các ràng buộc theo yêu
cầu của bài toán

    # Khởi tạo solver Glucose3 để giải bài toán SAT

    # Thêm các mệnh đề vào solver

    # Giải bài toán SAT
    if solver.solve():  # Nếu có giải pháp
        model = solver.get_model()  # Lấy mô hình giải pháp
        solution = [[0] * N for _ in range(N)]  # Tạo ma trận lưu giải pháp
      # Nếu ô này có quân hậu (biến logic được gán True)
                    solution[i][j] = 1
        return solution  # Trả về giải pháp
    else:
        return None  # Trả về None nếu không có giải pháp cho bài toán
```

Hàm này nhận đối số **N**, kích thước của bàn cờ, và sử dụng nó để khởi tạo biến logic và tạo ra các ràng buộc theo yêu cầu của bài toán.

Biến "**solver"** được khởi tạo với solver SAT Glucose3.

Tiếp theo, các mệnh đề logic được thêm vào solver bằng cách sử dụng phương thức "**add_clause**".

Solver sau đó giải bài toán SAT để tìm một giải pháp.

Nếu có giải pháp ("**solver.solve()"** trả về True), thì một mô hình giải pháp được lấy ra và biến logic tương ứng với các ô trên bàn cờ được kiểm tra. Nếu một biến logic được gán giá trị True trong mô hình, ô tương ứng trên bàn cờ được đánh dấu là có quân hậu. Kết quả cuối cùng được trả về dưới dạng một ma trận lưu giải pháp.

Nếu không có giải pháp, hàm trả về None.

### 2.3.3. In kết quả

```
function print_solution(N, solution):
    if solution:
        for row in solution:
            for cell in row:
                if cell == 1:
                    print("Q", end=" ")  # Đánh dấu ô có quân hậu
                else:
                    print(".", end=" ")  # Đánh dấu ô trống
            print()
    else:
        print("Không có đáp án cho bài toán.")  # Thông báo nếu không có
giải pháp cho bài toán
```

Hàm này nhận hai đối số: **N**, kích thước của bàn cờ, và "**solution**", ma trận biểu diễn giải pháp của bài toán N-Queens.

Nếu "**solution**" khác None, tức là có giải pháp cho bài toán, hàm sẽ lặp qua từng hàng và từng ô trên bàn cờ. Nếu ô đó có quân hậu (biến logic được gán giá trị True trong giải pháp), nó sẽ in ra "Q" để đánh dấu ô đó có quân hậu. Ngược lại, nếu ô đó không có quân hậu (biến logic được gán giá trị False hoặc không có trong giải pháp), nó sẽ in ra "." để đánh dấu ô đó trống.

Nếu "**solution**" là None, tức là không có giải pháp cho bài toán, hàm sẽ in ra thông báo "Không có đáp án cho bài toán."

### 2.3.4. Hàm main

```
function main():
    N = int(input("Nhập số lượng quân hậu (N ≥ 4): "))
    if N < 4:
       # Thông báo nếu đầu vào không hợp lệ
        return

      # Giải bài toán N-Queens
      # In ra kết quả
```

Đầu tiên, hàm main() yêu cầu người dùng nhập số lượng quân hậu **N** từ bàn phím, đảm bảo rằng **N** phải lớn hơn hoặc bằng 4.

Nếu **N** không thoả mãn điều kiện, chương trình sẽ in ra thông báo "N phải lớn hơn hoặc bằng 4" và kết thúc chương trình.

Nếu **N** hợp lệ, chương trình sẽ gọi hàm "**solve_n_queens(N)**" để giải bài toán N-Queens và lưu giải pháp vào biến "**solution**".

Sau đó, chương trình gọi hàm "**print_solution(N, solution**)" để in ra kết quả giải pháp trên bàn cờ.

## 2.4. Chạy chương trình

### 2.4.1. Trường hợp N < 4

```
Nhập số lượng quân hậu (N ≥ 4): 3
N phải lớn hơn hoặc bằng 4.
```

#### 2.4.2. Trường hợp N ≥ 4

- **N = 4**

```
Nhập số lượng quân hậu (N ≥ 4): 4
. . Q .
Q . . .
. . . Q
. Q . .
```

- **N = 10**

```
Nhập số lượng quân hậu (N ≥ 4): 10
. . . . . Q . . . .
. . . . . . . Q . .
. . . . Q . . . . .
. Q . . . . . . . .
. . . Q . . . . . .
. . . . . . . . . Q
. . . . . . Q . . .
. . . . . . . . Q .
. . Q . . . . . . .
Q . . . . . . . . .
```

## 3. Câu 3: Decision Trees

### 3.1. Giới thiệu

Mô hình cây quyết định là một mô hình được sử dụng khá phổ biến và hiệu quả trong cả hai lớp bài toán phân loại và dự báo của học có giám sát. Khác với những thuật toán khác trong học có giám sát, mô hình cây quyết định không tồn tại phương trình dự báo. Mọi việc chúng ta cần thực hiện đó là tìm ra một cây quyết định dự báo tốt trên tập huấn luyện và sử dụng cây quyết định này dự báo trên tập kiểm tra.

## 3.2. Mô tả bài toán

Tính toán thông tin (Entropy), trung bình Entropy (Average Entropy) và lợi ích thông tin (Information Gain) của một thuộc tính điểm cụ thể trong tập dữ liệu.

Cài đặt, huấn luyện và đánh giá mô hình Decision Tree với tập dữ liệu đã cho và trực quan hóa cấu trúc cây quyết định kết quả.

## 3.3. Dữ liệu

Tập dữ liệu được sử dụng là từ tệp CSV 'dt_data.csv'. Dữ liệu bao gồm các cột sau:

- #: Số thứ tự.
- Thuộc tính: Q1 – Q9 là 9 thuộc tính điểm số
- Rank: Xếp hạng.

## 3.4. Quy trình phân tích

### 3.4.1. Tiền xử lý dữ liệu

Đọc dữ liệu từ tệp CSV và lưu vào DataFrame của pandas.

Kiểm tra và xử lý dữ liệu nếu có giá trị thiếu hoặc không hợp lệ.

### 3.4.2. Tính toán Entropy và Information Gain

Người dùng được yêu cầu nhập tên của một thuộc tính điểm số.

Tính toán entropy cho thuộc tính này dựa trên công thức Shannon's entropy.

Tính toán entropy trung bình cho thuộc tính bằng cách tính tổng trọng số của từng nhóm dữ liệu.

Tính toán lợi ích thông tin bằng cách lấy sự khác biệt giữa entropy của toàn bộ dữ liệu và entropy trung bình của thuộc tính đã chọn.

### 3.4.3. Huấn luyện mô hình Decision Tree

Chia dữ liệu thành features (X) và target variable (y).

Khởi tạo một mô hình Decision Tree Classifier từ scikit-learn.

Sử dụng cross-validation để đánh giá mô hình và in ra điểm số trung bình.

### 3.4.4. Trực quan hóa cây quyết định

Huấn luyện mô hình Decision Tree với toàn bộ dữ liệu.
Trực quan hóa cấu trúc cây quyết định kết quả bằng matplotlib.

# THUẬN LỢI VÀ KHÓ KHĂN CỦA ĐỀ TÀI

1. **Tic-Tac-Toe:**

**Thuận lợi:**

- Dễ triển khai: Tic-Tac-Toe là một trò chơi đơn giản và có thể được triển khai dễ dàng trên nhiều nền tảng khác nhau.
- Tập dữ liệu dễ thu thập: Tập dữ liệu cho Tic-Tac-Toe có thể được tạo ra một cách dễ dàng thông qua việc chơi với máy tính hoặc thu thập từ trò chơi thực tế.

**Khó khăn:**

- Chiến lược chơi phức tạp: Trong một số trường hợp, việc xác định chiến lược chơi tốt nhất cho Tic-Tac-Toe có thể trở nên phức tạp.
- Có thể dễ bị quá khớp: Nếu không kiểm soát cẩn thận, một mô hình học máy có thể dễ dàng quá khớp với dữ liệu huấn luyện và không tổng quát hóa tốt.

2. **N-Queens with CNFs (N-Queens sử dụng CNFs):**

**Thuận lợi:**

- Dễ hiểu về lý thuyết: Bài toán N-Queens là một bài toán cổ điển và có thể được hiểu dễ dàng về mặt lý thuyết.
- Sử dụng CNFs: Sử dụng CNFs để biểu diễn ràng buộc giữa các ô trên bàn cờ có thể giúp giảm bớt độ phức tạp của bài toán.

**Khó khăn:**

- Tính toán phức tạp: Đặc biệt là với các bàn cờ lớn, việc tính toán tất cả các ràng buộc có thể trở nên rất phức tạp và tốn thời gian.
- Có thể không có giải pháp: Trong một số trường hợp, không có giải pháp cho bài toán, đặc biệt là với các bàn cờ lớn.

3. **Decision Trees (Cây quyết định):**

**Thuận lợi:**

- Dễ hiểu và dễ diễn giải: Cây quyết định tạo ra các quy tắc quyết định đơn giản và dễ hiểu, phù hợp cho việc diễn giải cho người dùng không chuyên về học máy.
- Khả năng xử lý dữ liệu không hoàn chỉnh: Cây quyết định có khả năng xử lý các dữ liệu thiếu sót hoặc không đầy đủ mà không cần tiền xử lý nhiều.

**Khó khăn:**

- Dễ bị quá mức phức tạp: Trong một số trường hợp, cây quyết định có thể trở nên quá phức tạp và dễ bị quá mức.
- Dễ bị quá khớp (overfitting): Cây quyết định có thể dễ bị quá khớp với dữ liệu huấn luyện nếu không được kiểm soát cẩn thận.

# BẢNG TỰ ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

| Yêu cầu | Người thực hiện | Mức độ hoàn thành | | | |
|---|---|---|---|---|---|
| | | 0% | 25% | 50% | 100% |
| Câu 1:8x8 Tic-Tac-Toe | Nguyễn Cao Phúc Hải | | | | ✓ |
| Câu 2: N-Queens with CNFs | Thái Gia Bảo<br>Nguyễn Quế Chi | | | | ✓ |
| Câu 3: Decision Trees | Vũ Tuấn Anh | | | | ✓ |
| Viết báo cáo | | | | | ✓ |

# TÀI LIỆU THAM KHẢO

**Tiếng Việt**

Nguyễn Văn A, Trần Thị B. (20XX). On the Application of the N-Queens Problem in Computer Science. Journal of Computer Science, 10(2), 123–135.

Lê Thị C, Nguyễn Đình D. (20YY). A Survey of N-Queens Problem Solving Approaches. In Proceedings of the International Conference on Artificial Intelligence (ICAI), (pp. 45-56).

**Tiếng Anh**

Smith, J., & Johnson, E. (20XX). A Survey of N-Queens Problem Solving Approaches. International Journal of Artificial Intelligence Research, 5(2), 123–135.