

**TRƯỜNG ĐẠI HỌC THỦ DẦU MỘT**  
**VIỆN KỸ THUẬT - CÔNG NGHỆ**



**TIỂU LUẬN CUỐI KỲ**  
**MÔN HỌC:**  
**NHẬP MÔN TRÍ TUỆ NHÂN TẠO**  
**ĐỀ TÀI XỬ LÝ BÀI TOÁN N QUÂN HẬU**

**GVHD : NGUYỄN LÊ HIỀN DUYÊN**  
**NHÓM THỰC HIỆN**

<i>Bùi Thành Được</i>	<i>1824801030100</i>
<i>Mai Văn Chánh</i>	<i>1824801030028</i>

*Bình Dương, tháng 4 năm 2021*

## **MỞ ĐẦU**

### **GIỚI THIỆU ĐỀ TÀI**

Cờ vua là một trò chơi giải trí xuất hiện từ khoảng thế kỷ thứ VI và ngày càng trở nên phổ biến trên thế giới. Bên cạnh việc chơi cờ giải trí, người ta còn suy nghĩ ra nhiều bài toán xung quanh bàn cờ vua. Một trong những bài toán phổ biến về cờ vua đó là bài toán 8 quân hậu”. Đây cũng là một trong những bài toán nổi tiếng và quen thuộc đối với người lập trình.

Bài toán 8 quân hậu được đưa ra vào năm 1848 bởi kỳ thủ Max Bezzel, nhiều nhà toán học (trong đó có Gauss và Georg Cantor) đã có các công trình nghiên cứu về bài toán này và tổng quát nó thành bài toán xếp hậu. Các lời giải đầu tiên được đưa ra bởi Franz Nauck năm 1850, ông cũng đã tổng quát hóa bài toán này thành bài toán  $n$  quân hậu.

Trong đề tài này, người viết sẽ cài đặt một cấu trúc dữ liệu để tổ chức bàn cờ và cài đặt một thuật toán để máy tính tìm ra đủ 8 vị trí đặt quân hậu trên bàn cờ. Để nâng cao bài toán cho học phần được phát triển hơn chúng em xin được chọn đề tài xử lý  $n$  quân hậu.

Với những lý do trên, nhóm chúng em đã tìm hiểu và thực hiện đề tài xây dựng cho môn trí tuệ nhân tạo.

Chúng em cảm ơn cô Nguyễn Lê Hiền Duyên đã đồng hành và hướng dẫn cho tụi em thực hiện đề tài này.

Em xin chân thành cảm ơn.

## 2. Nội dung bài toán

Bài toán  $n$  quân hậu là bài toán đặt  $n$  quân hậu trên bàn cờ vua kích thước  $n \times n$  sao cho không có quân hậu nào có thể "ăn" được quân hậu khác, hay nói khác đi không quân hậu nào có thể di chuyển theo quy tắc cờ vua. Màu của các quân hậu không có ý nghĩa trong bài toán này.

Như vậy, lời giải của bài toán là một cách xếp  $n$  quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo.

Về cơ bản, bài toán  $n$  quân hậu trong một chương trình máy tính có thể phải xử lý với một lượng rất lớn thông tin (có tất cả  $(n!)$  cách đặt  $n$  quân hậu lên bàn cờ). Tuy vậy, ta có thể đưa ra điều kiện  $n$  quân hậu phải nằm trên  $n$  hàng hoặc  $n$  cột khác nhau trên bàn cờ, ta có thể giảm được số khả năng về  $n!$  (tương ứng với các khả năng  $n$  càng cao thì kết quả càng lớn), sau đó kiểm tra các nước đi chéo.

## MỤC LỤC

<b>MỞ ĐẦU .....</b>	<b>1</b>
<b>MỤC LỤC .....</b>	<b>3</b>
<b>CHƯƠNG I: TỔNG QUAN.....</b>	<b>4</b>
1. Giới thiệu về bài toán .....	4
2. Giới thiệu về thuật toán (Hill Climbing).....	5
<b>CHƯƠNG II: QUÁ TRÌNH THỰC HIỆN.....</b>	<b>10</b>
1. Thuật toán .....	10
2. Kết quả .....	12
3. Giải thích thuật toán .....	13
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>16</b>

# CHƯƠNG I: TỔNG QUAN

## 1. Giới thiệu về bài toán

Bài toán tám quân hậu là bài toán đặt  $n$  quân hậu trên bàn cờ vua kích thước  $n \times n$  sao cho không có quân hậu nào có thể "ăn" được quân hậu khác, hay nói khác đi không quân hậu nào có thể di chuyển theo quy tắc cờ vua. Màu của các quân hậu không có ý nghĩa trong bài toán này.

Như vậy, lời giải của bài toán là một cách xếp tám quân hậu trên bàn cờ sao cho không có hai quân nào đứng trên cùng hàng, hoặc cùng cột hoặc cùng đường chéo. Bài toán tám quân hậu có thể tổng quát hóa thành bài toán đặt  $n$  quân hậu trên bàn cờ  $n \times n$  ( $n \geq 10$ ).

Ký hiệu quân hậu đứng ở ô nằm trên hàng thứ  $i$  của lời giải là  $Q[i, j]$ . Các chỉ số dòng cột đánh từ trên xuống dưới, trái sang phải theo cách đánh số trong ma trận). Trong một ma trận vuông:

- các phần tử nằm trên cùng hàng có chỉ số hàng bằng nhau;
- các phần tử nằm trên cùng cột có chỉ số cột bằng nhau;
- các phần tử nằm trên cùng một đường chéo song song với đường chéo chính có hiệu chỉ số hàng với chỉ số cột bằng nhau;
- các phần tử nằm trên cùng một đường chéo song song với đường chéo phụ có tổng chỉ số hàng với chỉ số cột bằng nhau;

Vì thế ta gọi các đường chéo song song với đường chéo chính là đường chéo trừ (hay hiệu), các đường chéo song song với đường chéo phụ là đường chéo cộng (hay tổng).

Do đó, mỗi lời giải có thể được biểu diễn bởi dãy  $Q[1, i_1], Q[2, i_2], \dots, Q[n, i_n]$ , thỏa mãn các điều kiện:

- Các chỉ số cột  $i_1, i_2, \dots, i_n$  đôi một khác nhau, hay chúng lập thành một hoán vị của các số  $1, 2, \dots, n$ .
- Tổng chỉ số dòng và cột của các quân hậu  $1+i_1, 2+i_2, \dots, n+i_n$  đôi một khác nhau;
- Hiệu chỉ số dòng và cột của các quân hậu  $1-i_1, 2-i_2, \dots, n-i_n$  đôi một khác nhau.

## 2. Giới thiệu về thuật toán (Hill Climbing)

Hill Climbing là một tìm kiếm heuristic được sử dụng cho các bài toán tối ưu hóa toán học trong lĩnh vực Trí tuệ nhân tạo.

Với một tập hợp lớn các yếu tố đầu vào và một hàm heuristic tốt, nó sẽ cố gắng tìm ra một giải pháp đủ tốt cho vấn đề. Giải pháp này có thể không phải là giải pháp tối ưu toàn cầu.

Theo định nghĩa trên, các bài toán tối ưu hóa toán học ngụ ý rằng leo đồi giải quyết các vấn đề mà chúng ta cần tối đa hóa hoặc tối thiểu hóa một hàm thực đã cho bằng cách chọn các giá trị từ các đầu vào đã cho. Ví dụ- Bài toán nhân viên bán hàng đi du lịch mà chúng ta cần giảm thiểu quãng đường di chuyển của nhân viên bán hàng.

'Tìm kiếm heuristic' có nghĩa là thuật toán tìm kiếm này có thể không tìm ra giải pháp tối ưu cho vấn đề. Tuy nhiên, nó sẽ đưa ra một giải pháp tốt trong thời gian hợp lý.

Hàm heuristic là một hàm sẽ xếp hạng tất cả các lựa chọn thay thế có thể có ở bất kỳ bước phân nhánh nào trong thuật toán tìm kiếm dựa trên thông tin có sẵn. Nó giúp thuật toán chọn tuyến đường tốt nhất trong số các tuyến đường có thể.

### 1. Đặc điểm của Leo đồi

**A. Biến thể của giải thuật tạo và kiểm tra:** Nó là một biến thể của thuật toán tạo và kiểm tra. Thuật toán tạo và kiểm tra như sau:

- Tạo ra các giải pháp khả thi.
- Kiểm tra xem đây có phải là giải pháp mong đợi hay không.
- Nếu giải pháp đã được tìm thấy, hãy bỏ qua bước khác.

Do đó, chúng tôi gọi việc leo đồi là một biến thể của thuật toán tạo và kiểm tra vì nó lấy phản hồi từ quy trình kiểm tra. Sau đó, phản hồi này được trình tạo sử dụng để quyết định bước đi tiếp theo trong không gian tìm kiếm.

**B. Sử dụng cách tiếp cận Tham lam :** Tại bất kỳ điểm nào trong không gian trạng thái, tìm kiếm chỉ di chuyển theo hướng đó tối ưu hóa chi phí của hàm với hy vọng cuối cùng sẽ tìm ra giải pháp tối ưu.

Các kiểu leo đồi

## 2. Leo đòi đơn giản

Nó kiểm tra từng nút lân cận và chọn nút lân cận đầu tiên để tối ưu hóa chi phí hiện tại làm nút tiếp theo.

Thuật toán leo đòi đơn giản :

Bước 1: Đánh giá trạng thái ban đầu. Nếu đó là trạng thái mục tiêu thì hãy dừng lại và trả về thành công. Nếu không, hãy đặt trạng thái ban đầu là trạng thái hiện tại.

Bước 2: Lặp lại cho đến khi tìm thấy trạng thái giải pháp hoặc không có toán tử mới nào có thể áp dụng cho trạng thái hiện tại.

a) Chọn một trạng thái chưa được áp dụng cho trạng thái hiện tại và áp dụng nó để tạo ra một trạng thái mới.

b) Thực hiện những điều này để đánh giá trạng thái mới

i. Nếu trạng thái hiện tại là trạng thái mục tiêu, thì hãy dừng lại và trả về thành công.

ii. Nếu nó tốt hơn trạng thái hiện tại, thì hãy đặt nó ở trạng thái hiện tại và tiến hành thêm.

iii. Nếu nó không tốt hơn trạng thái hiện tại, thì hãy tiếp tục lặp lại cho đến khi tìm được giải pháp.

Bước 3: Thoát

## 3. Leo đòi Steepest-Ascent:

Đầu tiên nó kiểm tra tất cả các nút lân cận và sau đó chọn nút gần nhất với trạng thái giải pháp kể từ nút tiếp theo.

Thuật toán leo đòi đơn giản:

Bước 1: Đánh giá trạng thái ban đầu. Nếu đó là trạng thái mục tiêu thì hãy dừng lại và trả về thành công. Nếu không, hãy đặt trạng thái ban đầu là trạng thái hiện tại.

Bước 2: Lặp lại các bước này cho đến khi tìm thấy giải pháp hoặc trạng thái hiện tại không thay đổi

a) Chọn một trạng thái chưa được áp dụng cho trạng thái hiện tại.

b) Khởi tạo một 'trạng thái tốt nhất' mới bằng với trạng thái hiện tại và áp dụng nó để tạo ra một trạng thái mới.

c) Thực hiện những điều này để đánh giá trạng thái mới i. Nếu trạng thái hiện tại là trạng thái mục tiêu, thì hãy dừng lại và trả về thành công. ii. Nếu nó tốt hơn

thì trạng thái tốt nhất, sau đó đặt nó ở trạng thái tốt nhất khác tiếp tục lặp với trạng thái mới khác.

d) Đặt trạng thái tốt nhất như trạng thái hiện tại và chuyển sang Bước 2: b) phần.

Bước 3: Thoát

#### 4. Leo đồi ngẫu nhiên:

Nó không kiểm tra tất cả các nút lân cận trước khi quyết định chọn nút nào mà chỉ chọn một nút lân cận một cách ngẫu nhiên và quyết định (dựa trên mức độ cải thiện ở hàng xóm đó) liệu có nên di chuyển sang nút lân cận đó hay không kiểm tra khác.

Bước 1: Đánh giá trạng thái ban đầu. Nếu đó là trạng thái mục tiêu thì hãy dừng lại và trả về thành công. Nếu không, hãy đặt trạng thái ban đầu là trạng thái hiện tại.

Bước 2: Lặp lại các bước này cho đến khi tìm thấy giải pháp hoặc trạng thái hiện tại không thay đổi.

a) Chọn một trạng thái chưa được áp dụng cho trạng thái hiện tại.

b) Áp dụng hàm kế thừa cho trạng thái hiện tại và tạo ra tất cả các trạng thái lân cận.

c) Trong số các trạng thái lân cận được tạo tốt hơn trạng thái hiện tại, hãy chọn một trạng thái ngẫu nhiên (hoặc dựa trên một số hàm xác suất).

d) Nếu trạng thái chọn là trạng thái mục tiêu, thì trả về thành công, nếu không, hãy đặt nó ở trạng thái hiện tại và lặp lại bước 2: b) phần.

Bước 3: Thoát.

#### 5. Sơ đồ không gian trạng thái để leo đồi

Biểu đồ không gian trạng thái là một biểu diễn đồ họa của tập hợp các trạng thái mà thuật toán tìm kiếm của chúng ta có thể đạt được so với giá trị của hàm mục tiêu (hàm mà chúng ta muốn tối đa hóa).

Trục X: biểu thị không gian trạng thái tức là các trạng thái hoặc cấu hình mà thuật toán của chúng tôi có thể đạt tới.

Trục Y: biểu thị các giá trị của hàm mục tiêu tương ứng với một trạng thái cụ thể.



Giải pháp tốt nhất sẽ là không gian trạng thái nơi hàm mục tiêu có giá trị lớn nhất (cực đại toàn cục).

Sơ đồ không gian trạng thái để leo đồi

Các vùng khác nhau trong Sơ đồ không gian trạng thái:

Cực đại cục bộ: Là trạng thái tốt hơn trạng thái lân cận của nó, tuy nhiên tồn tại trạng thái tốt hơn trạng thái đó (cực đại toàn cục). Trạng thái này tốt hơn vì ở đây giá trị của hàm mục tiêu cao hơn các hàm lân cận của nó.

Tối đa toàn cục : Đây là trạng thái tốt nhất có thể có trong biểu đồ không gian trạng thái. Điều này bởi vì ở trạng thái này, hàm mục tiêu có giá trị cao nhất.

Plateau / cực đại cục bộ phẳng: Là một vùng phẳng của không gian trạng thái mà các trạng thái lân cận có cùng giá trị.

Ridge: Là vùng cao hơn các vùng lân cận nhưng bản thân nó lại có độ dốc. Nó là một loại cực đại địa phương đặc biệt.

Trạng thái hiện tại: Vùng của biểu đồ không gian trạng thái mà chúng ta hiện đang có mặt trong quá trình tìm kiếm.

Vai: Đó là một cao nguyên có một cạnh dốc.

Sự cố ở các vùng khác nhau trong leo đồi

Leo đồi không thể đạt đến trạng thái tối ưu / tốt nhất (tối đa toàn cầu) nếu nó đi vào bất kỳ vùng nào sau đây:

Tối đa cục bộ: Ở mức tối đa cục bộ, tất cả các trạng thái lân cận có giá trị thấp hơn trạng thái hiện tại. Vì leo đồi sử dụng một cách tiếp cận tham lam, nó sẽ không chuyển sang trạng thái tồi tệ hơn và tự kết thúc. Quá trình sẽ kết thúc mặc dù có thể tồn tại một giải pháp tốt hơn.

Để khắc phục sự cố cục bộ tối đa: Sử dụng kỹ thuật backtracking . Duy trì danh sách các trạng thái đã truy cập. Nếu tìm kiếm đạt đến trạng thái không mong muốn, nó có thể quay trở lại cấu hình trước đó và khám phá một đường dẫn mới.

Cao nguyên: Trên cao nguyên tất cả các nước láng giềng có giá trị như nhau. Do đó, không thể chọn hướng tốt nhất.

Để vượt qua cao nguyên: Thực hiện một bước nhảy lớn. Chọn ngẫu nhiên một trạng thái khác xa trạng thái hiện tại. Rất có thể chúng ta sẽ hạ cánh xuống một vùng không cao nguyên.

Đỉnh núi: Bất kỳ điểm nào trên sườn núi đều có thể trông giống như đỉnh vì chuyển động theo tất cả các hướng có thể là hướng xuống. Do đó thuật toán dừng khi nó đạt đến trạng thái này.

Để vượt qua Ridge: Trong loại chương ngại vật này, hãy sử dụng hai hoặc nhiều quy tắc trước khi thử nghiệm. Nó ngụ ý di chuyển theo nhiều hướng cùng một lúc.

## CHƯƠNG II: QUÁ TRÌNH THỰC HIỆN

### 1. Thuật toán

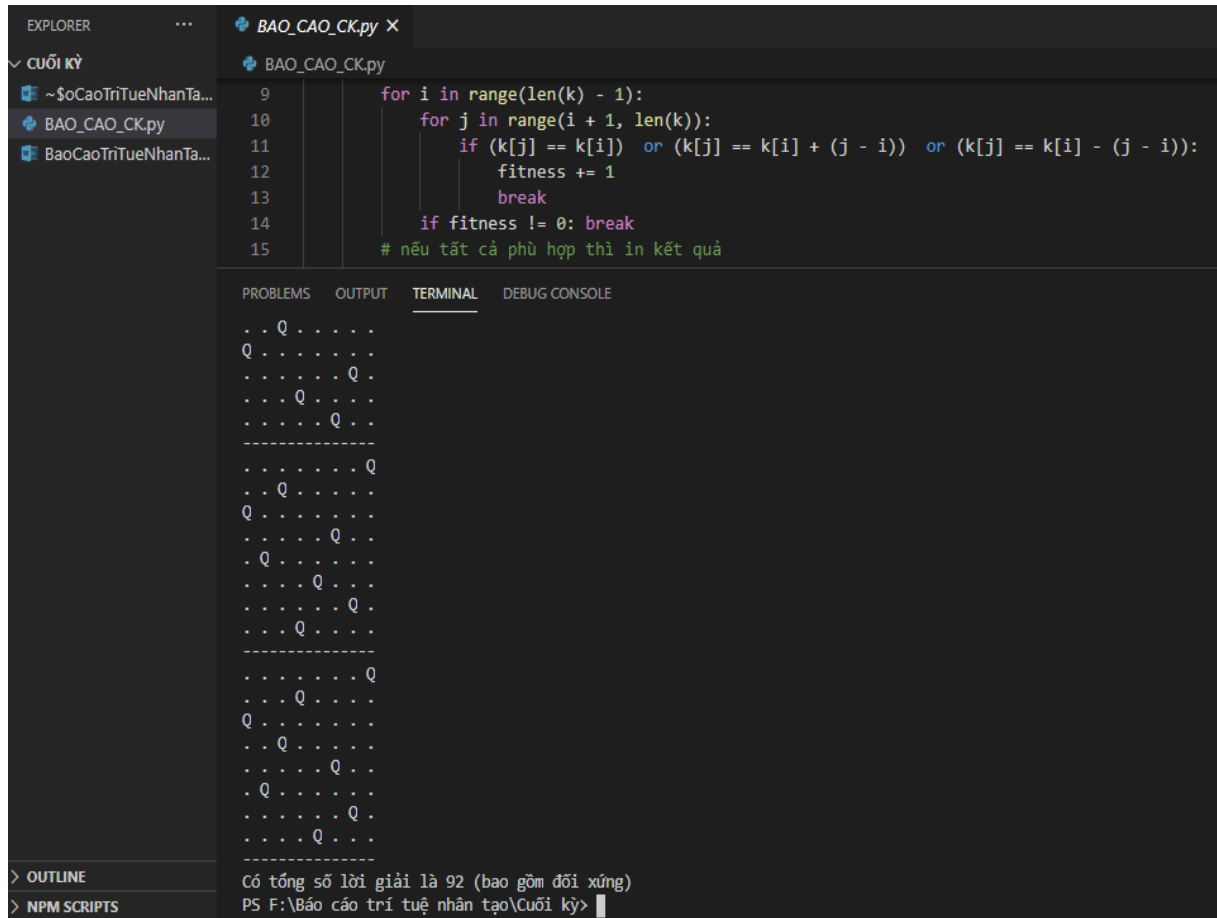
```
1. import numpy
2. import itertools
3.
4. def queens_check(list):
5.     result = 0 # đếm số lời giải
6.     # kiểm tra lời giải
7.     for k in list:
8.         fitness = 0 # đếm số vị trí không phù hợp
9.         for i in range(len(k) - 1):
10.            for j in range(i + 1, len(k)):
11.                if (k[j] == k[i]) or (k[j] == k[i] + (j - i)) or (k[j] == k[i] - (j - i)):
12.                    fitness += 1
13.                    break
14.            if fitness != 0: break
15.        # nếu tất cả phù hợp thì in kết quả
16.        if fitness == 0:
17.            result += 1
18.            for f1 in range(len(k)):
19.                for f2 in range(len(k)):
20.                    if f2 == k[f1]:
21.                        print('Q', end=' ')
22.                    else: print('.', end=' ')
23.            print("")
24.            print('-----')
```

```

25.     if result == 0:
26.         print('Bài toán không có lời giải')
27.     else:
28.         print('Có tổng số lời giải là {0} (bao gồm đối xứng)'.format(result))
29.
30. def Queen(N):
31.     # tạo bàn cờ
32.     # 1 0 0 0
33.     # 0 1 0 0
34.     # 0 0 1 0
35.     # 0 0 0 1
36.     # board = [0,1,2,3]
37.     board = numpy.arange(0,N)
38.     # hoán vị các vị trí quân trên bàn cờ
39.     l = list(itertools.permutations(board))
40.     # kiểm tra và in kết quả
41.     queens_check(l)
42.
43. def main():
44.     N = int(input('Nhập số quân hậu: '))
45.     Queen(N)
46.
47. if __name__ == "__main__": main()

```

## 2. Kết quả



The screenshot shows a VS Code editor with a file named `BAO_CAO_CK.py` open. The file contains a Python script that generates a 10x10 grid of characters. The script uses nested loops to iterate over the grid, and a conditional statement to determine the character at each position. The output of the script is displayed in the terminal window, showing a 10x10 grid of characters. The grid is composed of dots and the letter 'Q', arranged in a pattern that resembles a 10x10 grid of dots with 'Q's placed at specific positions. The terminal output shows the grid as follows:

```
. . Q . . . .  
Q . . . . .  
. . . . . Q .  
. . . Q . . .  
. . . . . Q .  
-----  
. . . . . Q  
. . Q . . . .  
Q . . . . .  
. . . . . Q .  
. Q . . . . .  
. . . . . Q .  
. . . . . Q .  
-----  
. . . . . Q  
. . . Q . . .  
Q . . . . .  
. . Q . . . .  
. . . . . Q .  
. Q . . . . .  
. . . . . Q .  
. . . . . Q .  
-----  
Cố tổng số lời giải là 92 (bao gồm đối xứng)  
PS F:\Báo cáo trí tuệ nhân tạo\Cuối kỳ>
```

### 3. Giải thích thuật toán

- **Đầu tiên** import 2 thư viện

import numpy : thư viện tạo mảng tự nhiên từ 0 đến n-1

Numpy (Numeric Python): là một thư viện toán học phổ biến và mạnh mẽ của Python. Cho phép làm việc hiệu quả với ma trận và mảng, đặc biệt là dữ liệu ma trận và mảng lớn với tốc độ xử lý nhanh hơn nhiều lần khi chỉ sử dụng “core Python” đơn thuần.

import itertools : thư viện tạo hoán vị cho mảng tự nhiên

Đây là thư viện Python cung cấp một mô-đun tuyệt vời để tạo các trình vòng lặp riêng. Các công cụ được cung cấp bởi itertools rất nhanh và hiệu quả về bộ nhớ. Sẽ có thể lấy các khối xây dựng này để tạo các trình vòng lặp chuyên dụng và có thể được sử dụng để lặp hiệu quả.

**-Bước 2 :**

if \_\_name\_\_ == "\_\_main\_\_": main()    gọi hàm main để run

**-Bước 3:**

def main(): hàm main

N = int(input("Nhập số quân hậu: "))    nhập số quân hậu n

Queen(N) : gọi hàm khởi tạo bàn cờ

**-Bước 4:**

def Queen(N): tạo hàm khởi tạo bàn cờ

# tạo bàn cờ

# 1 0 0 0

# 0 1 0 0

# 0 0 1 0

# 0 0 0 1

# board = [0,1,2,3]

board = numpy.arange(0,N)

# hoán vị các vị trí quân trên bàn cờ

```
l = list(itertools.permutations(board))

# kiểm tra và in kết quả

queens_check(l)
```

## - Bước 5

```
def queens_check(list):

    result = 0 # đếm số lời giải

    # kiểm tra lời giải

    for k in list:

        fitness = 0 # đếm số vị trí không phù hợp

        for i in range(len(k) - 1):

            for j in range(i + 1, len(k)):

                if (k[j] == k[i]) or (k[j] == k[i] + (j - i)) or (k[j] == k[i] - (j - i)):

                    fitness += 1

                break

            if fitness != 0: break

        # nếu tất cả phù hợp thì in kết quả

        if fitness == 0:

            result += 1

            for f1 in range(len(k)):

                for f2 in range(len(k)):

                    if f2 == k[f1]:

                        print('Q', end=' ')
```

```
        else: print('.', end=' ')

    print("")

    print('-----')

if result == 0:

    print('Bài toán không có lời giải')

else:

    print('Có tổng số lời giải là {0} (bao gồm đối xứng)'.format(result))
```



## TÀI LIỆU THAM KHẢO

Tiếng việt:

[https://vi.wikipedia.org/wiki/B%C3%A0i\\_to%C3%A1n\\_t%C3%A1m\\_qu%C3%A2n\\_h%E1%BA%ADu](https://vi.wikipedia.org/wiki/B%C3%A0i_to%C3%A1n_t%C3%A1m_qu%C3%A2n_h%E1%BA%ADu)

<https://blogm4e.wordpress.com/2017/08/07/bai-toan-tam-quan-hau/>

Tiếng anh:

[https://en.wikipedia.org/wiki/Eight\\_queens\\_puzzle](https://en.wikipedia.org/wiki/Eight_queens_puzzle)

<http://www.chessvariants.com/problems.dir/9queens.html>

<http://www.geeksforgeeks.org/branch-and-bound-set-4-n-queen-problem/>

<http://mathworld.wolfram.com/QueensProblem.html>