TRƯỜNG ĐẠI HỌC
VĂN LANG
Đạo đức - Ý chí - Sáng tạo
KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN KHOA HỌC DỮ LIỆU

NHẬP MÔN TRÍ TUỆ NHÂN TẠO

LAB 02 – ARTIFICIAL INTELLIGENCE

TS. TRẦN NGỌC VIỆT
THS. PHAN HỒ VIẾT TRƯỜNG

HỌC KỲ 3 – NĂM HỌC 2024-2025

KHÓA K29 & K28

#THUẬT TOÁN TÌM KIẾM MÙ

*Tìm kiếm không chính xác (tìm kiếm mù)

+Nó còn được gọi là chiến lược tìm kiếm mù hoặc kiểm soát mù, được đặt tên như vậy vì chỉ có thông tin về bài toán đặt ra và không có thông tin bổ sung nào khác về các trạng thái. Kỹ thuật tìm kiếm như thế này là tìm kiếm toàn bộ không gian trạng thái để tìm lời giải.

+Tìm kiếm chiều rộng (BFS) & tìm kiếm chiều sâu (DFS) là những ví dụ về tìm kiếm không chính xác.

#THUẬT TOÁN TÌM KIẾM MÙ

*Chiến lược tìm kiếm mù (Uninformed search strategies)

+Chiến lược tìm kiếm mù chỉ sử dụng thông tin có sẵn được định nghĩa trong bài toán.

+Các thuật toán tìm kiếm mù:

- Tìm kiếm theo chiều rộng (Breadth-first search)
- Tìm kiếm theo chiều sâu (Depth-first search)

#THỰC HÀNH 01

#Tìm kiếm theo chiều rộng

```
from collections import defaultdict
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self, u, v):
        self.graph[u].append(v)

    def BFS(self, u):
        visited = [False] * (len(self.graph))

        queue = []

        visited[u-1] = True
        queue.append(u)

        while queue:
            u = queue.pop(0)
            print(u, end=' ')

            for i in self.graph[u]:
                if not visited[i-1]:
                    queue.append(i)
                    visited[i-1] = True
```

```
if __name__ == '__main__':
    g = Graph()
    g.addEdge(0, 1)
    g.addEdge(1, 0)
    g.addEdge(0, 2)
    g.addEdge(2, 0)
    g.addEdge(2, 4)
    g.addEdge(4, 2)
    g.addEdge(1, 4)
    g.addEdge(4, 1)
    g.addEdge(1, 3)
    g.addEdge(3, 1)
    g.addEdge(3, 4)
    g.addEdge(4, 3)
    g.addEdge(3, 5)
    g.addEdge(5, 3)
    g.addEdge(5, 4)
    g.addEdge(4, 5)

    print("BFS - duyệt tìm kiếm chiều rộng bắt đầu từ đỉnh 0")
    g.BFS(0)
```

#THỰC HÀNH 01

#Tìm kiếm theo chiều rộng

```
BFS - duyệt tìm kiếm chiều rộng bắt đầu từ đỉnh 0
0 1 2 4 3 5
```

#Yêu cầu: mô tả BFS, duyệt tìm kiếm và vẽ đồ thị?

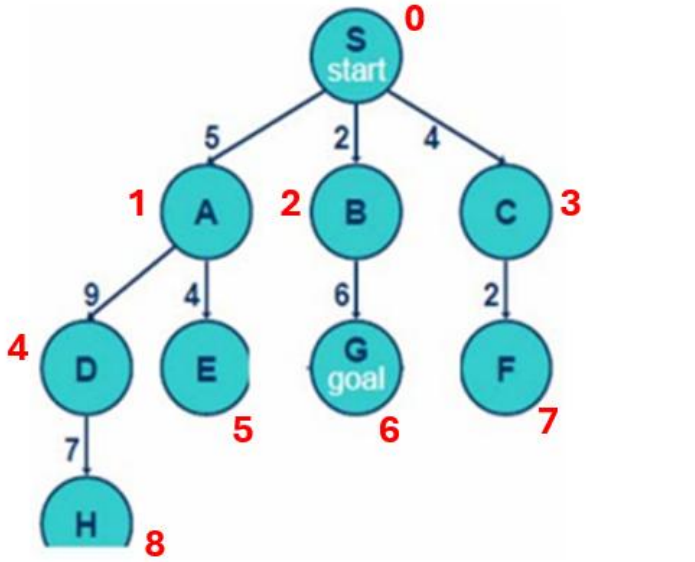
#THỰC HÀNH 02

#Tìm kiếm theo chiều rộng

```
BFS - duyệt tìm kiếm chiều rộng bắt đầu từ đỉnh 0
0 1 2 3 4 5 6 7 8
```

node	Queue	Father
	S	
S	A, B, C	Father[A,B,C]=S
A	B, C, D, E	Father[D,E]=A
B	C,D,E,G	Father[G]=B
C	D, E, G, F	Father[F]=C
D	E,G, F, H	Father[H]=D
E	G, F, H	
G	F, H	

Giá trị các biến trong giải thuật theo chiều rộng



Cây tìm kiếm của giải thuật theo chiều rộng

#Yêu cầu: mô tả BFS, duyệt tìm kiếm và vẽ đồ thị?

#THỰC HÀNH 03

#Tìm kiếm theo chiều sâu

```
from _collections import defaultdict
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self, u, v):
        self.graph[u].append(v)

    def DFSUtil(self, v, visited):
        visited[v] = True
        print(v, end=" ")

        for i in self.graph[v]:
            if visited[i] == False:
                self.DFSUtil(i, visited)

    def DFS(self, v):
        visited = [False] * (max(self.graph) + 1)

        self.DFSUtil(v, visited)
```

```
if __name__ == '__main__':
    g = Graph()
    g.addEdge(0, 1)
    g.addEdge(1, 0)
    g.addEdge(0, 2)
    g.addEdge(2, 0)
    g.addEdge(2, 4)
    g.addEdge(4, 2)
    g.addEdge(1, 4)
    g.addEdge(4, 1)
    g.addEdge(1, 3)
    g.addEdge(3, 1)
    g.addEdge(3, 4)
    g.addEdge(4, 3)
    g.addEdge(3, 5)
    g.addEdge(5, 3)
    g.addEdge(5, 4)
    g.addEdge(4, 5)
    print("DFS - Duyệt tìm kiếm từ vertex 0")
    g.DFS(0)
```

#THỰC HÀNH 03

#Tìm kiếm theo chiều sâu

```
DFS - Duyệt tìm kiếm từ vertex 0
0 1 4 2 3 5
```

#Yêu cầu: mô tả DFS, duyệt tìm kiếm và vẽ đồ thị?

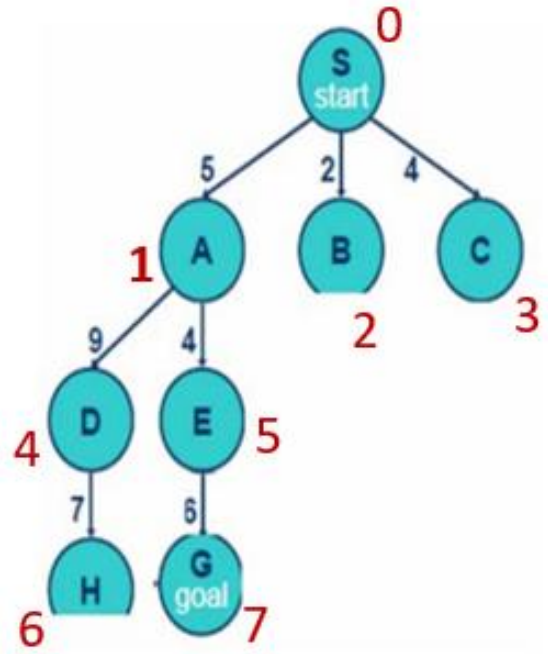
#THỰC HÀNH 04

#Tìm kiếm theo chiều sâu

```
DFS - Duyệt tìm kiếm từ vertex 0
0 1 4 6 5 7 2 3
```

node	Stack	father
	S	
S	A, B, C	Father[A,B,C]=S
A	D, E, B, C	Father[D,E]=A
D	H, E, B, C	Father[H]=D
H	E, B, C	
E	G, B, C	Father[G]=E
G		

Giá trị các biến trong giải thuật theo chiều sâu



Cây tìm kiếm của giải thuật theo chiều

#Yêu cầu: mô tả DFS, duyệt tìm kiếm và vẽ đồ thị?

#THỰC HÀNH 05

#Heuristic algorithm - DFS

```
from collections import deque
class Graph:
    def __init__(self, adjac_lis):
        self.adjac_lis = adjac_lis

    def get_neighbors(self, v):
        return self.adjac_lis[v]

    #hàm heuristic có giá trị bằng nhau cho tất cả các nút
    def h(self, n):
        H = {
            'A': 1,
            'B': 1,
            'C': 1,
            'D': 1,
            'E': 1,
            'Z': 1
        }

        return H[n]
```

```
def heuristic_alg(self, start, stop):

    open_lst = set([start])
    closed_lst = set([])

    poo = {}
    poo[start] = 0

    par = {}
    par[start] = start

    while len(open_lst) > 0:
        n = None

        for v in open_lst:
            if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):
                n = v;

        if n == None:
            print('Path does not exist!')
            return None
```

#THỰC HÀNH 05

#Heuristic algorithm - DFS

```
            if n == stop:
                reconst_path = []
                while par[n] != n:
                    reconst_path.append(n)
                    n = par[n]
                reconst_path.append(start)
                reconst_path.reverse()
                print('Path found: {}'.format(reconst_path))
                return reconst_path

            for (m, weight) in self.get_neighbors(n):
                if m not in open_lst and m not in closed_lst:
                    open_lst.add(m)
                    par[m] = n
                    poo[m] = poo[n] + weight
                else:
                    if poo[m] > poo[n] + weight:
                        poo[m] = poo[n] + weight
                        par[m] = n
                        if m in closed_lst:
                            closed_lst.remove(m)
                            open_lst.add(m)

            open_lst.remove(n)
            closed_lst.add(n)
        print('Path does not exist!')
        return None
```

#THỰC HÀNH 05

#Heuristic algorithm - DFS

```
if __name__ == '__main__':
    adjac_lis = {
        'A': [('B', 4), ('C', 2)],
        'B': [('A', 4), ('D', 10), ('E', 12)],
        'C': [('A', 2), ('E', 7)],
        'D': [('B', 10), ('E', 6), ('Z', 15)],
        'E': [('B', 12), ('C', 7), ('D', 6), ('Z', 9)],
        'Z': [('D', 15), ('E', 9)]
    }
    g = Graph(adjac_lis)
    start_node = str(input("Enter the start node:"))
    stop_node = str(input("Enter the stop node:"))
    g.heuristic_alg(start_node, stop_node)

    #out:
```

```
Enter the start node:A
Enter the stop node:Z
Path found: ['A', 'C', 'E', 'Z']
```

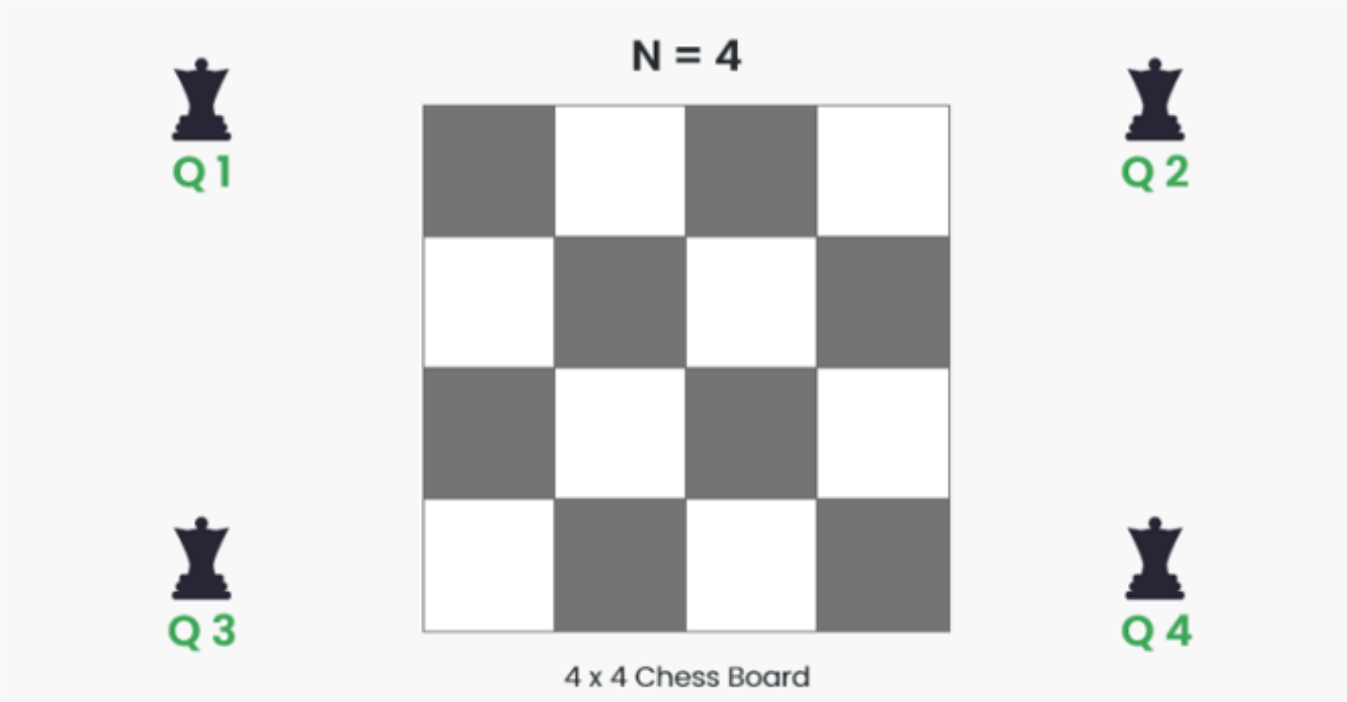
#Yêu cầu: mô tả DFS, duyệt tìm kiếm và vẽ đồ thị?

#THỰC HÀNH 06

#4-Queens State

+Bài toán n = 4 Queen trên một bàn cờ vua có 4 x 4.

+Đặt ra không có hai Queen nào tấn công lẫn nhau. Ràng buộc điều kiện là không được phép đặt hai Queen trên cùng một hàng hoặc trên cùng một cột hoặc cùng một đường chéo.



#THỰC HÀNH 06

#4-Queens State

Step 1:

+Đặt quân Queen Q1 vào ô (0,0).

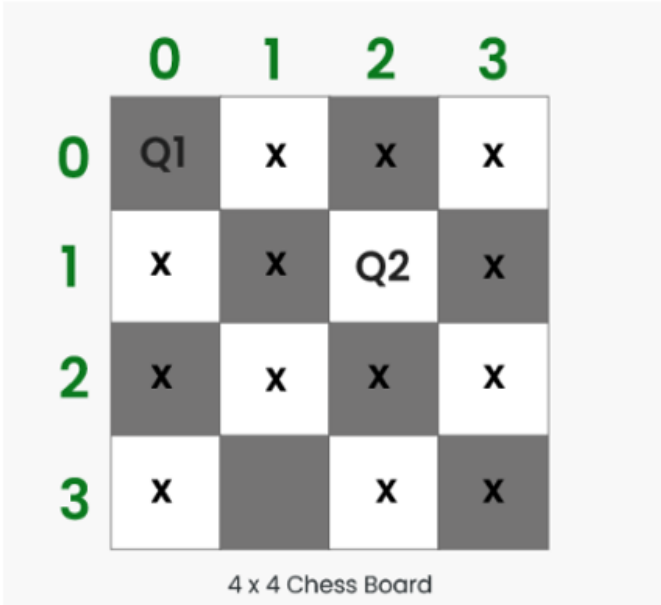
+'X' biểu thị các ô không an toàn tức là chúng đang bị Queen Q1 tấn công.

+Sau đó di chuyển đến hàng tiếp theo.

Step 2:

+Đặt quân Queen Q2 tiếp theo vào ô (1, 2).

+Di chuyển đến hàng tiếp theo.



#THỰC HÀNH 06

#4-Queens State

Step 3:

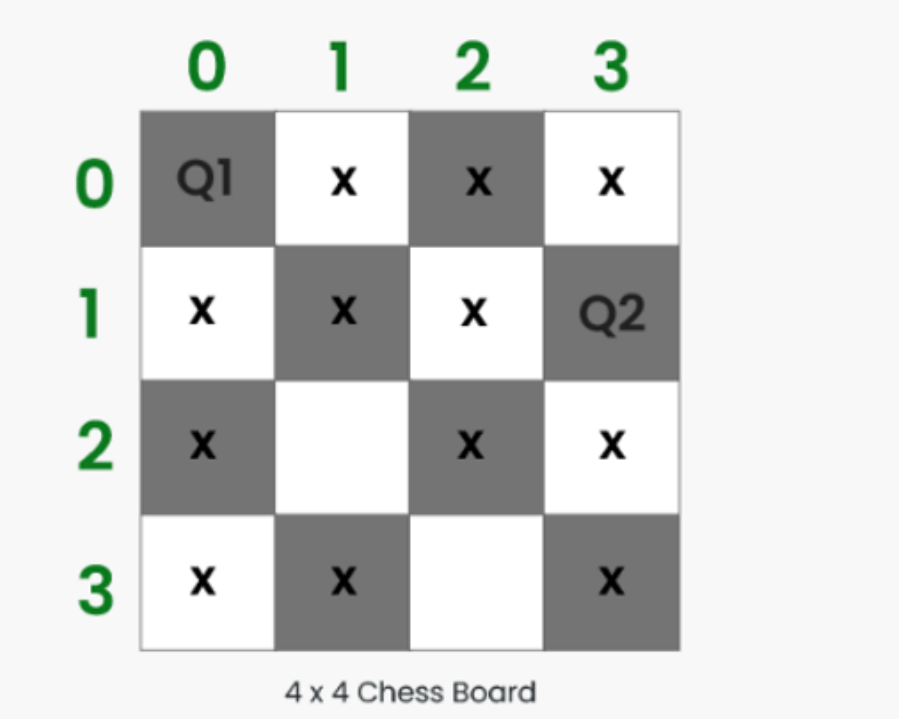
+Hàng 2 không có ô nào an toàn để đặt Queen Q3

+Vì vậy, quay lại và loại bỏ Queen Q2 khỏi ô (1, 2)

Step 4:

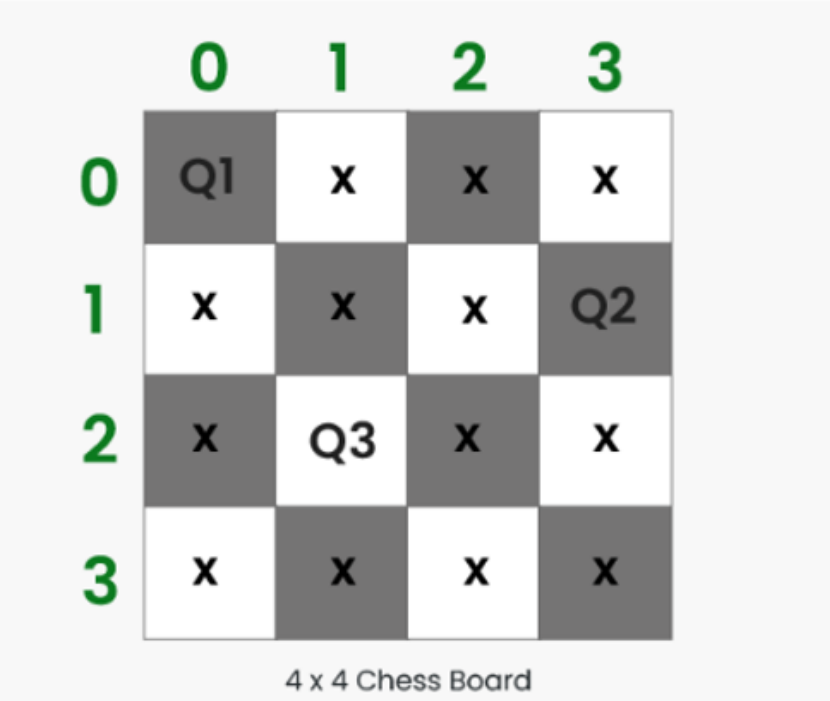
+Vẫn còn một ô an toàn ở hàng 1 tức là ô (1, 3).

+Đặt Queen Q2 vào ô (1, 3).



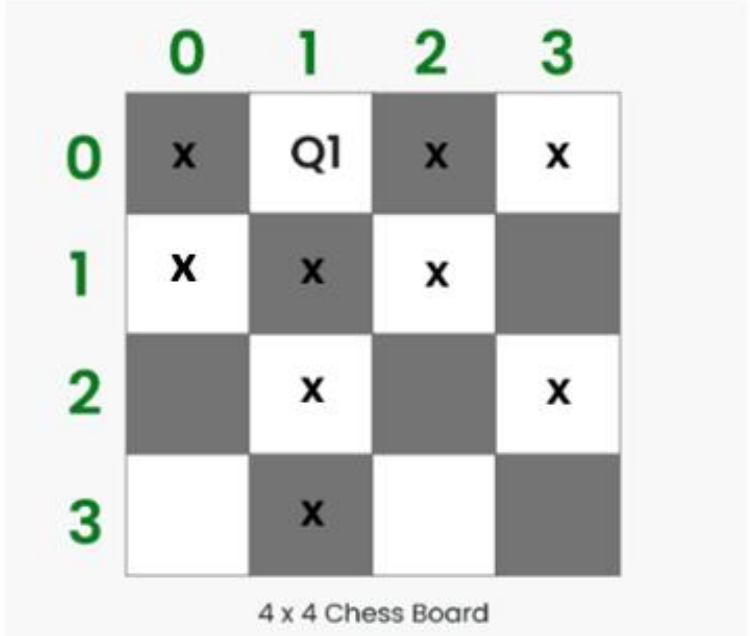
#THỰC HÀNH 06

Step 5:
+Đặt Queen Q3 vào ô (2, 1)



Step 6:
+Không có ô nào để đặt Queen Q4 vào hàng 3.
+Quay lại và loại Queen Q3 khỏi hàng 2.
+Tiếp tục không có ô an toàn nào khác ở hàng 2, vì vậy quay lại và loại Queen Q2 khỏi hàng 1.
+Queen Q1 sẽ bị loại khỏi ô (0,0) và di chuyển đến ô an toàn tiếp theo tức là (0, 1).

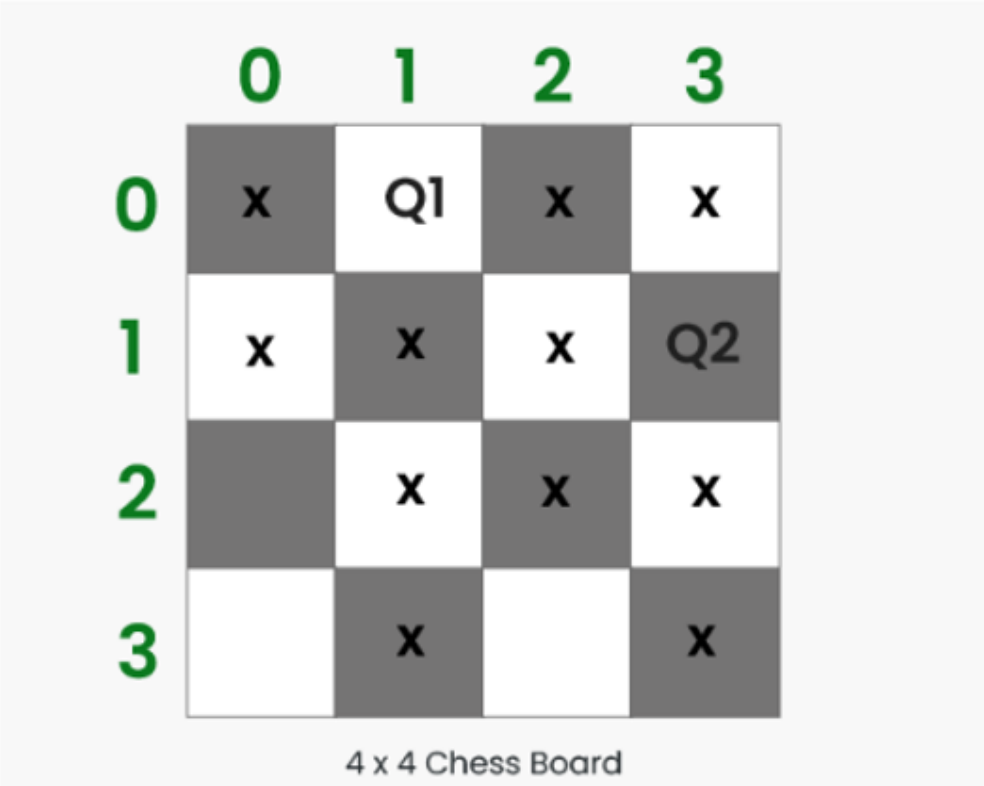
Step 7:
+Đặt Queen Q1 vào ô (0, 1) và di chuyển đến hàng tiếp theo.



#THỰC HÀNH 06

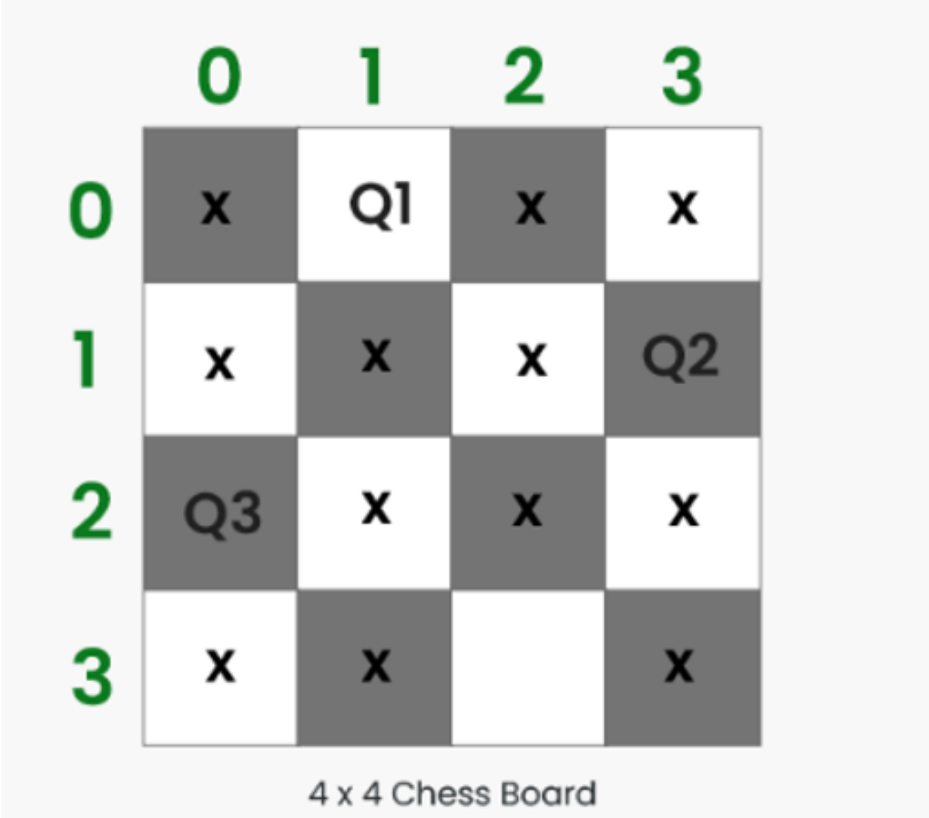
Step 8:

+Đặt Queen Q2 vào ô (1, 3) và di chuyển sang hàng tiếp theo.



Step 9:

+Đặt Queen Q3 vào ô (2, 0) và di chuyển sang hàng tiếp theo.

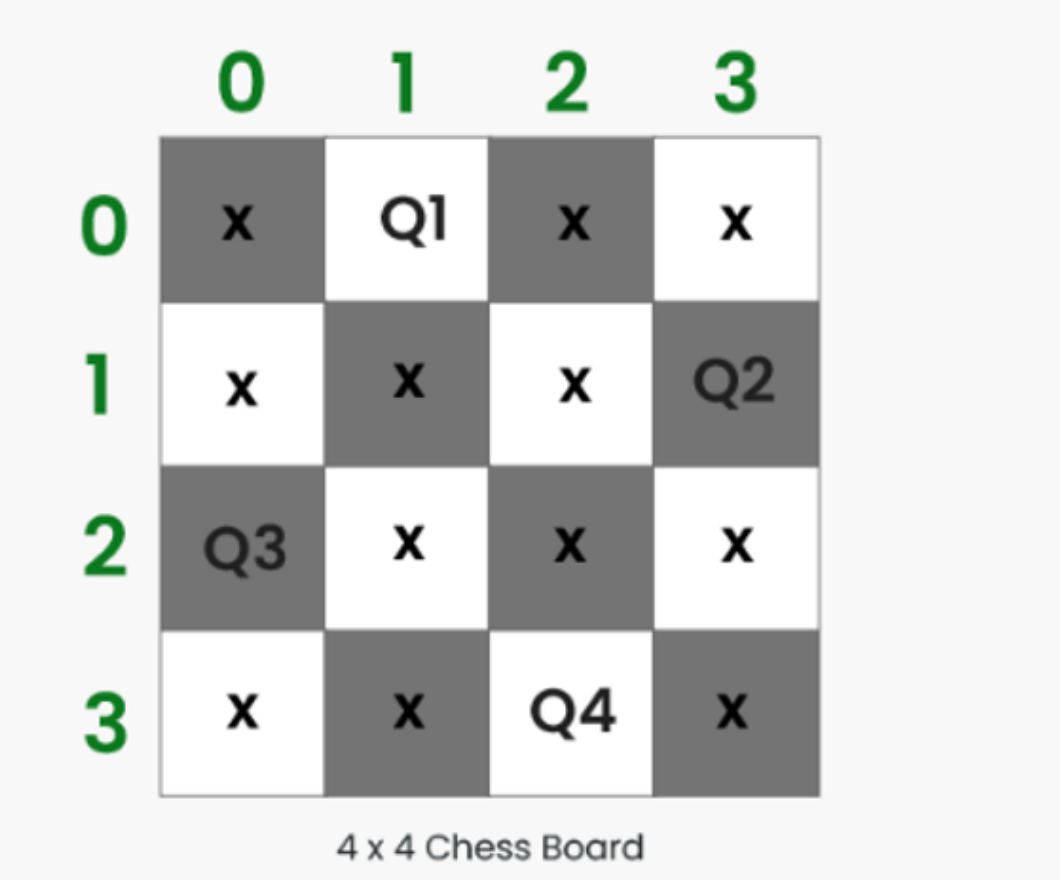


#THỰC HÀNH 06

Step 10:

+Đặt Queen Q4 vào ô (3, 2) và di chuyển đến hàng tiếp theo.

+Kết luận: đây là giải pháp được chấp nhận.



#THỰC HÀNH 06

#4-Queens State

```python
import numpy as np
#Kiểm tra trạng thái hiện tại là giải pháp hợp lệ hay k0
def is_valid_state(state, num_queens):
    return len(state) == num_queens

def get_candidates(state, num_queens):
    if not state: return range(num_queens)
    position = len(state)
    candidates = set(range(num_queens))
    for row, col in enumerate(state):
        candidates.discard(col)
        dist = position - row
        candidates.discard(col + dist)
        candidates.discard(col - dist)
    return candidates

#Đệ quy, tìm kiếm theo chiều sâu để tìm ra giải pháp hợp lệ
def search(state, solutions, num_queens):
    if is_valid_state(state, num_queens):
        solutions.append(state.copy())
        print(f"Valid State Found: {state}")

    for candidate in get_candidates(state, num_queens):
        state.append(candidate)
        search(state, solutions, num_queens)
        print(f"backtracking from: {state}")
        state.remove(candidate)
```

#THỰC HÀNH 06

#4-Queens State

```
def solve(num_queens):
    solutions = []
    state = []
    search(state, solutions, num_queens)
    return solutions

if __name__ == "__main__":
    num_queens = int(input("Enter number of queens n = "))
    solutions = solve(num_queens)
    for solution in solutions:
        board = np.full((num_queens, num_queens), "-")
        for row, col in enumerate(solution):
            board[row][col] = 'Q'
        print(f'\nSolution: {solution}')
        print(board)
```

#out:

```
Solution: [1, 3, 0, 2]
[['-' 'Q' '-' '-']
 ['-' '-' '-' 'Q']
 ['Q' '-' '-' '-']
 ['-' '-' 'Q' '-']]

Solution: [2, 0, 3, 1]
[['-' '-' 'Q' '-']
 ['Q' '-' '-' '-']
 ['-' '-' '-' 'Q']
 ['-' 'Q' '-' '-']]
```

#Yêu cầu: mô tả bài toán 4 – Queens & số solution?

#THỰC HÀNH 07

#8-Queens State

```python
import numpy as np
#Kiểm tra trạng thái hiện tại là giải pháp hợp lệ hay k0
def is_valid_state(state, num_queens):
    return len(state) == num_queens

def get_candidates(state, num_queens):
    if not state: return range(num_queens)
    position = len(state)
    candidates = set(range(num_queens))
    for row, col in enumerate(state):
        candidates.discard(col)
        dist = position - row
        candidates.discard(col + dist)
        candidates.discard(col - dist)
    return candidates

#Đệ quy, tìm kiếm theo chiều sâu để tìm ra giải pháp hợp lệ
def search(state, solutions, num_queens):
    if is_valid_state(state, num_queens):
        solutions.append(state.copy())
       # print(f"Valid State Found: {state}")

    for candidate in get_candidates(state, num_queens):
        state.append(candidate)
        search(state, solutions, num_queens)
        #print(f"backtracking from: {state}")
        state.remove(candidate)
```

#THỰC HÀNH 07

#8-Queens State

```python
def solve(num_queens):
    solutions = []
    state = []
    search(state, solutions, num_queens)
    return solutions

if __name__ == "__main__":
    num_queens = int(input("Enter number of queens n = "))
    solutions = solve(num_queens)
    i=0
    for solution in solutions:
        i+=1
        board = np.full((num_queens, num_queens), "-")
        for row, col in enumerate(solution):
            board[row][col] = 'Q'
        print(f'\nSolution: {solution}')
        print(board)
    print('Tổng số Giải pháp là:', len(solutions))
```

#out: . . .

```
Solution: [7, 3, 0, 2, 5, 1, 6, 4]
[['-' '-' '-' '-' '-' '-' '-' 'Q']
 ['-' '-' '-' 'Q' '-' '-' '-' '-']
 ['Q' '-' '-' '-' '-' '-' '-' '-']
 ['-' '-' 'Q' '-' '-' '-' '-' '-']
 ['-' '-' '-' '-' '-' 'Q' '-' '-']
 ['-' 'Q' '-' '-' '-' '-' '-' '-']
 ['-' '-' '-' '-' '-' '-' 'Q' '-']
 ['-' '-' '-' '-' 'Q' '-' '-' '-']]
Tổng số Giải pháp là: 92
```

#Yêu cầu: mô tả bài toán 8 – Queens & số solution?

TRƯỜNG ĐẠI HỌC
VĂN LANG
Đạo đức - Ý chí - Sáng tạo

KHOA CÔNG NGHỆ THÔNG TIN