LAB : 6.

29/10/24

Date __/__/__
Page _____

Implementing of A* Algorithm (N-queens).

```
import heapq

class state
    def __init__(self, board, row):
        self.board = board
        self.row = row
        self.heuristics = self.calculate heuristic

    def calculate heuristic(self):
        attacks = 0
        for i in range(self.row):
            for j in range(i+1, self.row):
                if self.board[i] == self.board[j] or
                    abs(self.board[i] - self.board[j]) ==
                        abs(i)
        return attacks

    def is_goal(self):
        return self.row == 8.

    def generate_successors(self)
        successors = []
        for col in range(8):
            if col not in self.board:
                new_board = self.board[:]
                new_board.append(col)
                successors.append(state(new_board,
                    self.row + 1))
    return successors
```

Date __/__/__
Page _____

```
def a_star_8_queens():
    initial_state = state([], 0)
    open_set = []
    heapq.heappush(open_set, (initial_state.
                heuristic, initial_state))
    while open_set:
        current_state = heapq.heappop(open_set)

    if current_state.is_goal():
        print_solution(current_state, generate_
                successors():
            heapq.heappush(open_set, (successor_
            heuristic + len(successor_board),
                successors))
        print("NO solution exists")
        return False

if __name__ == "__main__"
    a_star_8_queens()
```

Implementing hill climbing method (8-queens)

code :-

```
import random

def heuristic(state):
    h = 0
    h = len(state)
    for i in range(n):      j in range(i+1, n):
        for state[i] == state[j] or
            if state[i] == state[j] or
                abs(state[i] - state[j]) == j - i
                h += 1
    return h
```

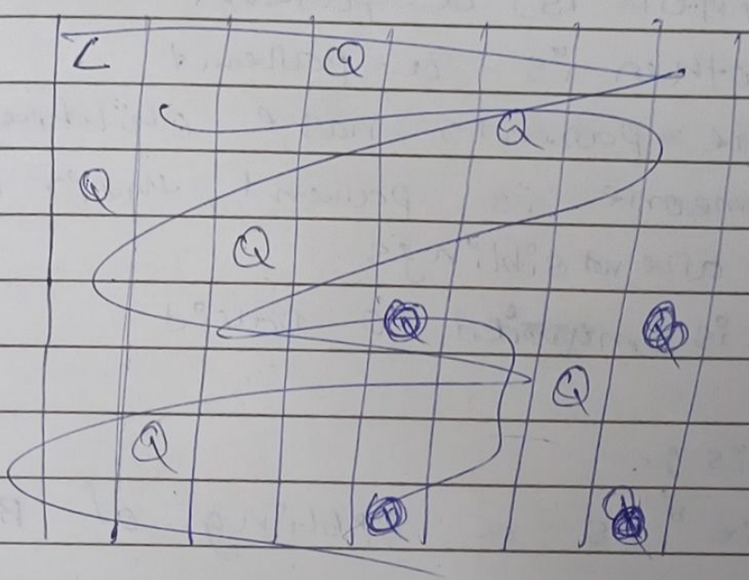
Date __/__/__
Page _____

```
def get_neighbors(state):
    neighbors = []
    n = len(state)
    for i in range(n):
        for j in range(n):
            if state[i] != j:
                new_state = list(state)
                new_state[i] = j
                neighbors.append(new_state)
    return neighbors

def hill_climbing(state):
    current = state
    while True:
        neighbors = get_neighbors(current)
        current_h = heuristic(current)
        best_h = current_h
        best_neighbor = current

        for neighbor in neighbors:
            h = heuristic(neighbor)
            best_h = current_h
            best_neighbor = current

        for neighbor in neighbors:
            h = heuristic(neighbor)
            if h < best_h:
                best_h = h;
                best_neighbor = neighbor
            if best_h >= current_h:
                break
        current = best_neighbors.
    return current, heuristic(current)
```

Date __/__/__
Page _____

```
initial_state = [random.randint(0,7)
    for in range(8)]

solution, final neurestic = hill_climbing
    initial_state)
Print("solution", solution).
```

Output :-



Hill climb :-
random placem
queen & finding
optimization.