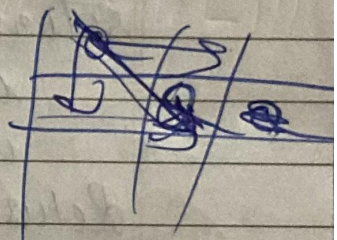
LAB: 06

29/10/24.

Hill climbing to implement 8-queens.

step 1: Initialize a random configured 8 x 8 chessboard with ~~each~~ ~~none~~ a queen placed in each row - call it start state.

step 2: calculate all possible conflicts of all the respective states. i.e no ~~the~~ two queens should be in same row & same diagonal.



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

step 3: Initialize current state
→ set current_state to initial_state.
→ calculate current_attacks by counting conflicts in current_state.

```
Function hill_climbing ():
current_state = randomly place queens in each row
current_attacks = calculate_attacks (current_
                                     state).
```

step4: -Iterate to find improved state.

```
For _ in range (100):
    neighbors = []
    for row in range(8):
        for col in range(8):
            if state[row] != col:
                neighbor = state[:]
                neighbor[row] = col.
                neighbors.append(neighbor)
```

step 5: choose the best neighbor. and return result.

```
current_state = next_state
current_attacks = next_attacks
Return current_state, current_attack
```

output:

Best solution found (with 0 attacks)

. . . Q . . . .
Q . . . .
. . . . Q . . .
. . . . Q
. . . . Q . . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .

⇒ A-star algorithm to implement 8 queens

step 1: Initialize a starting state where 8's board is empty.
→ Define it where each row is set to -1 i.e., no queens are placed
→ Add this as starting node in a priority queue i.e, priority_set with $g = 0$ and h = number of conflicts.

step 2: Define heuristic function which gives the computation of attacking pairs for given state.

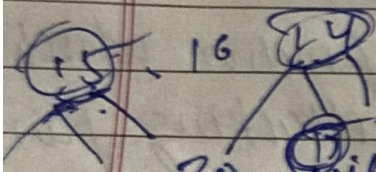
```
def heuristic(state): →
    attacks = 0
    for i in range(len(state)):
        for j in range(i+1, len(state)):
            if state[i] == state[j] or
abs(state[i] - state[j] == j - i)
                attacks += 1
    return attacks.
```

step 3: search for solution by removing node with lowest f (heuristic function)

```
if h == 0:
    return solution.
```

Function a_star
initial_state = [-1,-1,-1,-1,-1,-1,-1,-1]
priority_set = []
visited = empty set
while priority_set is not empty:
current_node = heapq.heappop(priority_set)
current_state = current_node.state

If heuristic(current_state) == 0
return current_state
If current_state is in visited:
continue
Add current_state to visited



next_row = index of first empty row in current state

if next_row < 8:
for column in range(8):
new_state = list(current_state)
new_state[next_row] = col
new_state = tuple(new_state)
if new_state is not in visited:
h = heuristic(new_state)
return heapq.heappush(priority_set, Node(new_state, h))

classmate
Date
Page

output
A* solution: