Lab 6. 29/10/2024.

* 8 Queens using A star

```
class State:
    function __init__(self, queens)
        self.queens = queens
        self.g = len(queens)
        self.h = self.heuristic()

    function heuristic()
        collision = 0
        for i in range(len(self.queens)):
            for j in range(i+1, len(self.queens)):
                if self.queens[i] == self.queens[j]
                and abs(self.queens[i] - self.
                    queens[j] == abs(i-j):
                    collisions += 1

        return collisions.

    function f():
        return g + h

    function gen_children():
        children = []
        for col in range(8):
            if col not in queens:
                children.append(
                new_state = State(queens+
                    [col])
                children.append(new_state)

        return children.
```

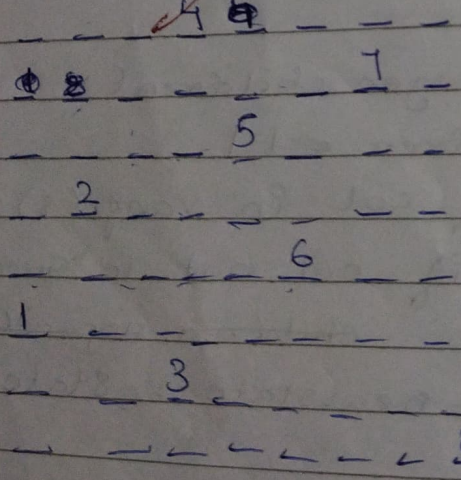
```
def a_star( ):
    in_state = State([])
    open = []
    heap.push(open, (in_state.h, in_state))
    while open:
        cur_state = heap.pop(open)
        if cur_state.g == 8 and
                cur_state.h == 0:
            return queens
        for child in generate children():
            heap.push(open, (child.h, child))
```

~~initial state~~ = ~~state(1)~~

```
sol = a_star()
if sol:
    print "Solution found"
else
    print "Not found".
```

4 7 5 2 6 1 3 8
0 1 3 4

Output:-



DATE: PAGE:

* 8 Queens using Hill climbing.

```
class State:
    def __init__(queen)
        initialize queens
        calculate h (no of collisions for the cur_state)

    def heuristic( ):
        collision = 0
        for i in range(len(queen)):
            for j in range(i+1, len(queens)):
                if queens[i] == queens[j]
                    and abs(queens[i] - queens[j])
                        == abs(i - j):
                    collisions. + = 1

        return collisions.

fun generate neighbour( )
    best_state = [ ]
    for i in range(8).
        for j in range(8):
            new_state = state[ :i] + j
                + state[i+1: ]
            if new_state.heuristic
                < Best_state.heuristic
                best_state = newstate.
```

DATE:

PAGE

```
def hill_climbing():
    initial_state = [random.randrange(8) for _ in range(8)]
    open = []
    for i in range(8):
        initial_state.append(random.randrange(8))
    heap_push(open, (initial_state.heuristic, initial_state))
    while True:
        cur_state = heap_pop(open)
        if cur_state.heuristics != 0:
            generate state
            push to heap
        elif cur_state.heuristics == 0:
            break
    print cur_state
```

```
a = State()
a.hill_climbing()
```

Proceed

DATE PAGE

output :-

[4, 7, 5, 2, 6, 1, 3, 8]

collicion = 0

22/10/24