Solve 8 queen's Problem.

29/10/24

A* algorithm.

```
import heapq

Class State:
    def __init__(queens, g, h)
        self.queens = queens
        self
```

```
def validatemove(queens, row, col):
    for r, c in enumerate(queens)
        if c == col or abs(c - col) == abs(r - row)
            return False
    return True.
```

```
def calculateattacks(queens):
    conflicts = 0
    for i in range(len(queens)):
        for j in range(i+1, len(queens)):
            if queens[i] == queens[j] or abs(queens[i] -
                queens[j]) == abs(j - i).
                conflicts += 1

    return conflicts.
```

```
def a*star algorithm(,):
    g=0, h=0
    f=0
    initial state = [[], 0]
    dist = [ ].
```

heapq.heappush(list, initialstate)

while list > 0:
current state = heapq.heappop(queens)
~~current cost = calculateattacks(current state)~~
next state = None
next cost = current cost

~~if~~ len(current state) ~~==~~ = n:
return current state.

~~row~~ row = len(current state)
for i in range(n):
if validate move(current state, row, i):
next state = current state + [i]
g = len(~~current~~ next state)
h = calculateattacks(current state)
f = g + h
heapq.heappush(list, next state)

return None.

Hill climbing

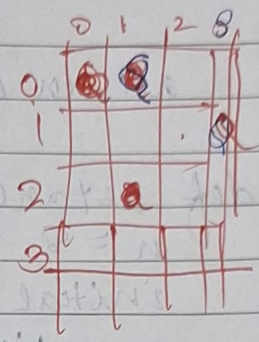
```
def initializeboard():
    return

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

def generateneighbours(state):
    for row in range(8):
        for col in range(8):
            if row != state[col]:
                neighbour = state[:] + col
    return neighbours

def hill_climbing():
    state = random.sample(range(0, 7) for in range(8)
    current_cost = calculate_attacks(state)
    new_state = None
    new_cost = current_cost

    while current_cost > 0:
        neighbours = generateneighbour(state)
        for neighbour in neighbours:
            cost = calculate_attacks(neighbour)
            if cost < new_cost:
                new_state = neighbour
                new_cost = cost
```

if new cost <= current cost:
    queens = new state.
    current cost = new cost.

if current cost == 0:
    return queens

return None.

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



A* algorithm

import heapq

class State:
    def __init__(self, queens, g, h):
        self.queens = queens
        self.g = g
        self.h = h
        self.f = g+h

    def __lt__(self, other):
        return self.f < other.f

def is_valid(queens, row, col):
    for r, c in enumerate(queens):
        if c == col or abs(c-col) == abs(r-row):
            return False
    return True

```
def calculate_conflicts(queens):
    conflicts = 0
    for i in range(len(queens)):
        for j in range(i+1, len(queens)):
            if queens[i] == queens[j] or
            abs(queens[i]-queens[j] ==
                abs(i-j):
                conflicts += 1
    return conflicts

def astar():
    n = 8
    initial_state = State([], 0, 0)
    open_list = []
    heapq.heappush(open_list, initial_state)

    while open_list:
        current_state = heapq.heappop(open_list

        if len(current_state.queens) == n:
            return current_state.queens

        row = len(current_state.queens)
        for col in range(n):
            if is_valid(current_state.queens, row,
                        col):
                new_queens = current_state.queens + [col]
                g = len(new_queens)
                h = calculate_conflicts(new_queens)
                new_state = State(new_queens, g, h)
                heapq.heappush(open_list, new_state)
    return None
```

solution = astar()

if solution:
    print(solution)
else:
    print("No solution")

Output
[1, 6, 2, 7, 0, 3, 5, 2]

~~Hill climbing~~
Hill climbing
Output: [5, 8, 6, 3, 0, 7, 1, 4]