**EXPERIMENT NO : 04 DATE : 05/03/24**

**Aim**: To implement the 8-Puzzel problem using Hill Climb Search

**Theory**:

It has set off a 3x3 board having 9 block spaces out of which 8 blocks having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence for getting the goal state. The 8-puzzle problem is invented and popularized by Noyes Palmer Chapman in the year 1870. It is played on a 3-by-3 grid with 8 square blocks/tiles labelled 1 through 8 and a blank square. The goal of this 8-puzzle problem is to rearrange the given blocks in the correct order. The tiles can be shifted vertically or horizontally using the empty square. We also know the eight-puzzle problem by the name of N puzzle problem or sliding puzzle problem. N-puzzle that consists of N tiles (N+1 titles with an empty tile) where N can be 8, 15, 24 and so on. In our example N = 8. (that is, square root of (8+1) = 3 rows and 3 columns). In the same way, if we have N = 15, 24 in this way, then they have Row and columns as follow (square root of (N+1) rows and square root of (N+1) columns). That is if N=15 than number of rows and columns= 4, and if N= 24 number of rows and columns= 5. So, basically in these types of problems we have given an initial state or initial configuration (Start state) and a Goal state or Goal Configuration. Here We are solving a problem of 8 puzzle that is a 3x3 matrix.

**Algorithm:**

Hill Climb

node = start

newnode = head(sortn(movegen(node)))

while h(newnode) > h(node) do

node = newnode

newnode = head(sortn(movegen(node)))

//end of while

return newnode

**Example:**



**Code:**

#case that terminats hill climbing due to failed case

# start\_state = [[1, 2, 3],

# [4, 5, 6],

# [7, 8, 0]]

# goal\_state = [[1, 2, 3],

# [8, 0, 4],

# [7, 6, 5]]

#successfull case

start\_state = [[1, 2, 3],

[5, 6, 0],

[7, 8, 4]]

goal\_state = [[1, 2, 3],

[5, 8, 6],

[0, 7, 4]]

def print\_state(state):

for row in state:

print(row)

def get\_blank\_position(state):

for i in range(3):

for j in range(3):

if state[i][j] == 0:

return i, j

def gen\_moves(state):

moves = []

i, j = get\_blank\_position(state)

if i == 0:

if j == 0:

moves.append((i, j + 1))

moves.append((i + 1, 0))

elif j == 1:

moves.append((i, j - 1))

moves.append((i, j + 1))

moves.append((i + 1, j))

elif j == 2:

moves.append((i, j - 1))

moves.append((i + 1, j))

elif i == 1:

if j == 0:

moves.append((i, j + 1))

moves.append((i - 1, j))

moves.append((i + 1, j))

elif j == 1:

moves.append((i, j - 1))

moves.append((i, j + 1))

moves.append((i + 1, j))

moves.append((i - 1, j))

elif j == 2:

moves.append((i, j - 1))

moves.append((i + 1, j))

moves.append((i - 1, j))

elif i == 2:

if j == 0:

moves.append((i - 1, j))

moves.append((i, j + 1))

elif j == 1:

moves.append((i, j - 1))

moves.append((i, j + 1))

moves.append((i - 1, j))

elif j == 2:

moves.append((i, j - 1))

moves.append((i - 1, j))

return moves

def make\_move(state, moves):

i, j = get\_blank\_position(state)

states = []

for new\_i, new\_j in moves:

current = [row[:] for row in state]

current[i][j], current[new\_i][new\_j] = current[new\_i][new\_j], current[i][j]

states.append(current)

return states

def manhattan\_distance(state, goal\_state):

total\_distance = 0

# Create dictionaries to store the position of each number in the goal state

goal\_positions = {}

for i in range(3):

for j in range(3):

goal\_positions[goal\_state[i][j]] = (i, j)

# Calculate Manhattan distance for each tile in the current state

for i in range(3):

for j in range(3):

if state[i][j] != 0: # Ignore the empty tile

num = state[i][j]

goal\_pos = goal\_positions[num]

distance = abs(i - goal\_pos[0]) + abs(j - goal\_pos[1])

total\_distance += distance

return total\_distance

def hill\_climbing(start):

current\_state = start

print("Initial State: ")

print\_state(current\_state)

if start\_state == goal\_state:

print("Goal reached")

prev\_heuristic = None

while True:

moves = gen\_moves(current\_state)

next\_states = make\_move(current\_state, moves)

min\_heuristic = float('inf')

next\_state = None

for state in next\_states:

heuristic = manhattan\_distance(state, goal\_state)

if heuristic < min\_heuristic:

min\_heuristic = heuristic

next\_state = state

if prev\_heuristic is not None and min\_heuristic >= prev\_heuristic:

print("No better state found. Terminating.")

break

prev\_heuristic=min\_heuristic

if min\_heuristic > 0:

current\_state = next\_state

print("Next State: ")

print\_state(current\_state)

print("Heuritic value = "+ str(min\_heuristic))

else:

print("Goal state found.")

print("Final state: ")

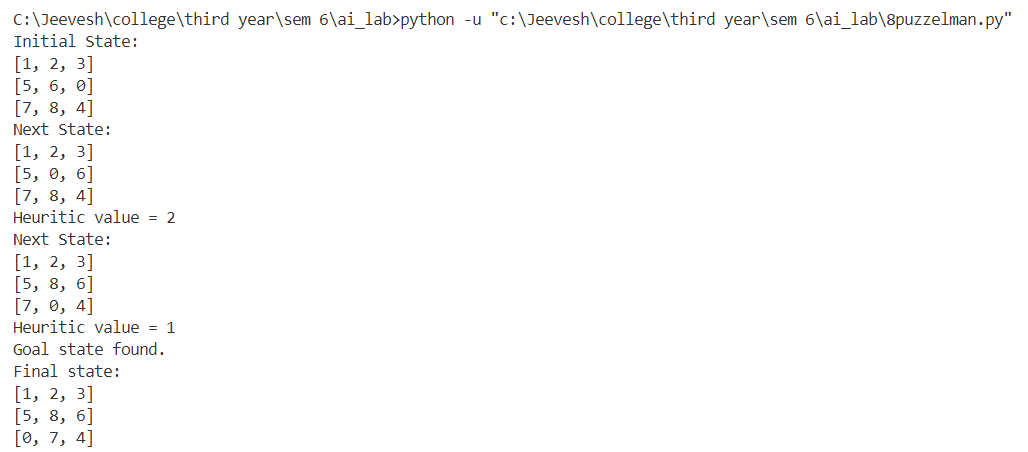
print\_state(goal\_state)

break

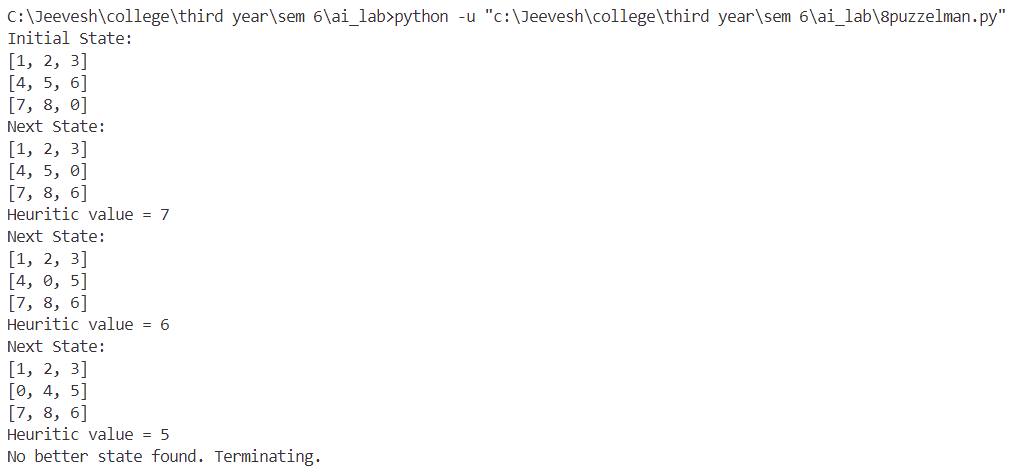
hill\_climbing(start\_state)

**Output:**

**Successful case**

****

**Failed case**

****

**Conclusion:**

Implementation of the 8 Puzzles Problem Using Hill Climb Search was carried out by tracing the algorithm and above output was obtained during the same.