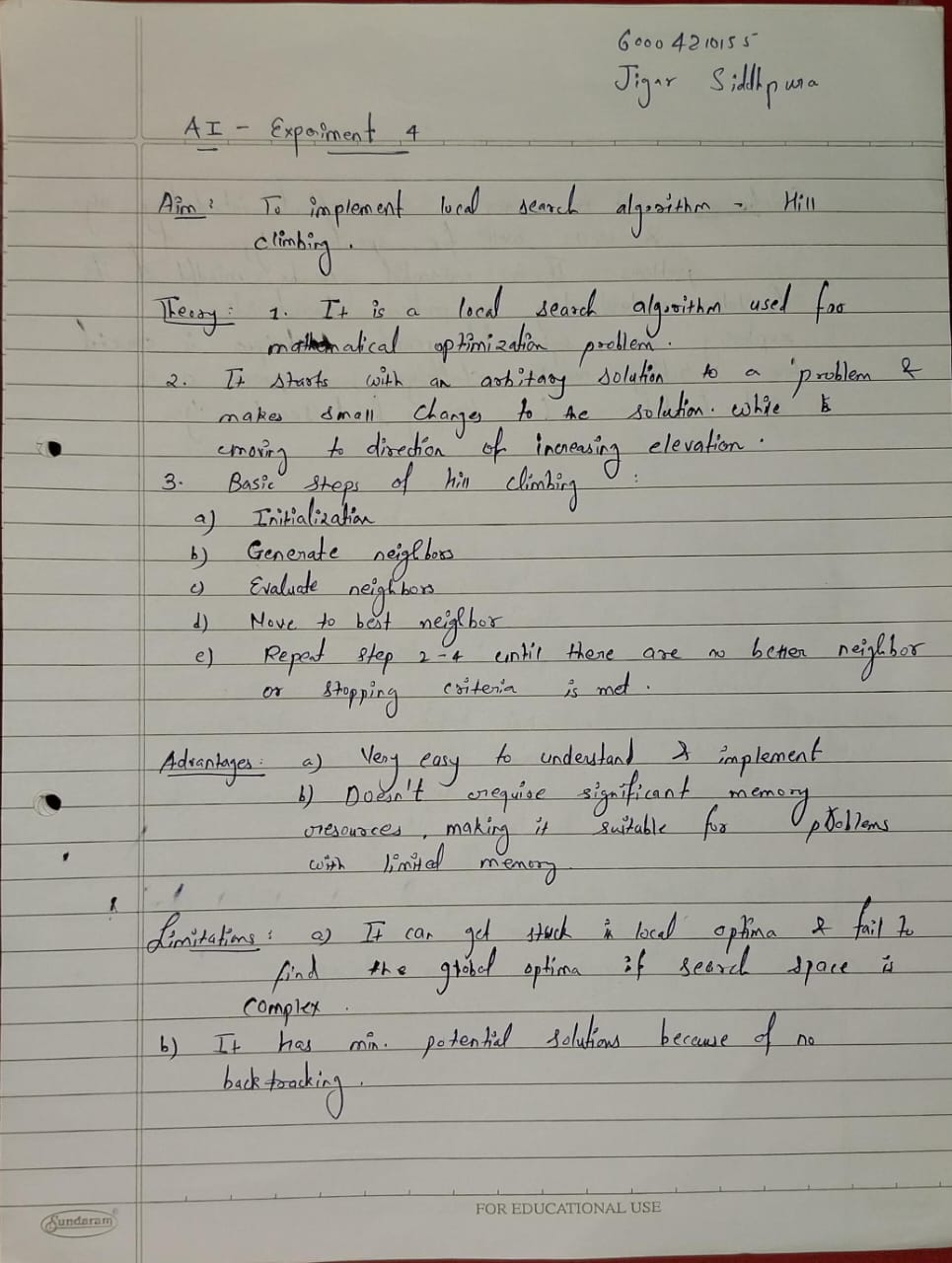
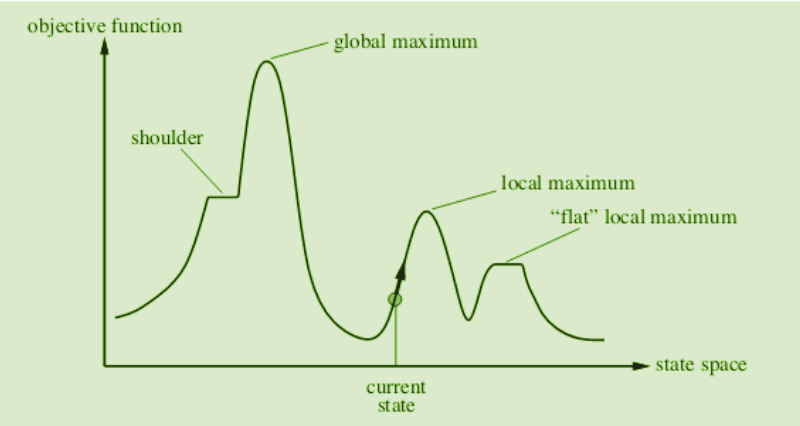
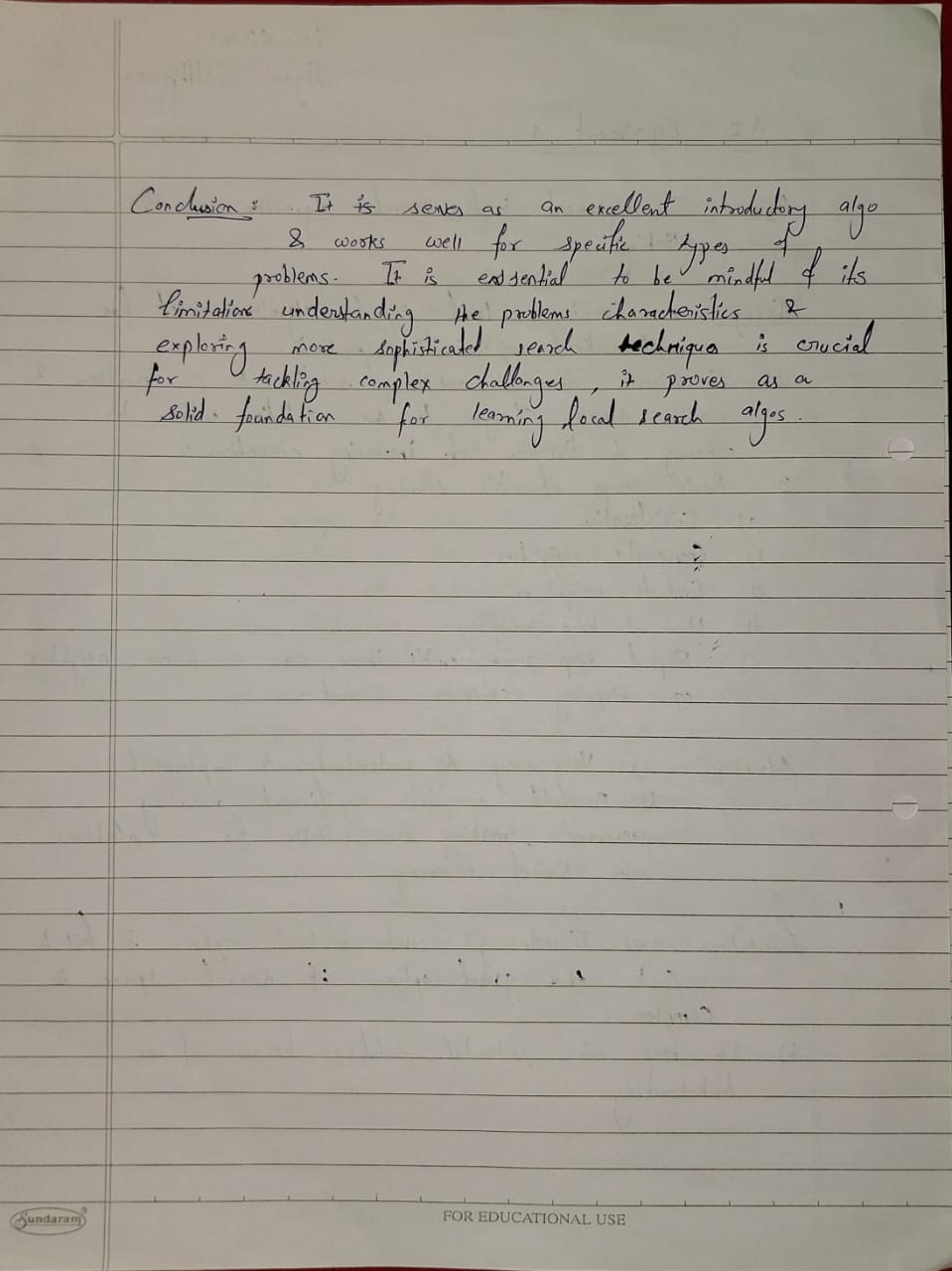
**Name: Jigar Siddhpura SAPID:** 60004200155

**DIV: C/C2** **Branch:** Computer Engineering

AI EXPERIMENT 4 - Hill Climbing





**Code :**

import copy

visited\_states: list = []

def generate\_child\_state(current\_state :list, prev\_heuristic :int, goal\_state :list) -> list|int :

"""Generates a child state by moving an element from one peg to another."""

*global* visited\_states

state\_copy = copy.deepcopy(current\_state)

    for i in range(len(state\_copy)):

        temp\_state = copy.deepcopy(state\_copy)

        if len(temp\_state[i]) > 0:

*# .pop remove last element from the list and returns it ⭐*

            element = temp\_state[i].pop()

            for j in range(len(temp\_state)):

                new\_state = copy.deepcopy(temp\_state)

                if j != i:

                    new\_state[j] = new\_state[j] + [element]

                    current\_heuristic = calculate\_heuristic(

                        new\_state, goal\_state)

                    if current\_heuristic > prev\_heuristic:

                        child\_state = copy.deepcopy(new\_state)

                        return child\_state

return 0

def calculate\_heuristic(current\_state :list, goal\_state :list) -> int:

"""Calculates the heuristic value for the given state compared to the goal state."""

    goal\_positions = goal\_state[3]

heuristic\_value = 0

    for i in range(len(current\_state)):

        check\_values :list = current\_state[i]

        if len(check\_values) > 0:

            for j in range(len(check\_values)):

                if check\_values[j] != goal\_positions[j]:

                    heuristic\_value -= j

                else:

                    heuristic\_value += j

    print(f"Heuristic value for {current\_state} is {heuristic\_value}")

return heuristic\_value

def solve\_puzzle(initial\_state, goal\_state):

*global* visited\_states

    if initial\_state == goal\_state:

        print(f"Solution found! {goal\_state}\n")

        Return

*# create deepcopy to prevent changes in the NESTED structure of the ORIGINAL list*

current\_state = copy.deepcopy(initial\_state)

    while True:

        visited\_states.append(copy.deepcopy(current\_state))

        print(f"Current State: {current\_state}")

        prev\_heuristic = calculate\_heuristic(current\_state, goal\_state)

        child = generate\_child\_state(current\_state, prev\_heuristic, goal\_state)

        if child == 0:

            print(

                f"No better heuristic value is obtained, declaring this as the goal state - {current\_state}\n"

            )

            Return

        print(f"Child chosen for exploration: {child}\n")

        current\_state = copy.deepcopy(child)

def main():

*global* visited\_states

    initial\_state = [[], [], [], ['B', 'C', 'D', 'A']]

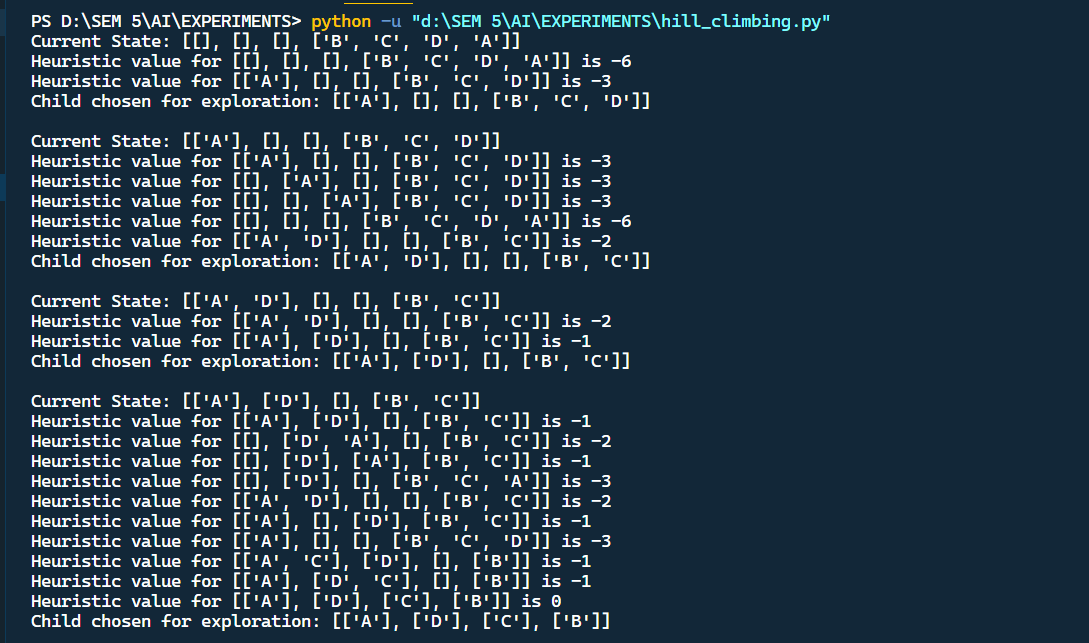
    goal\_state = [[], [], [], ['A', 'B', 'C', 'D']]

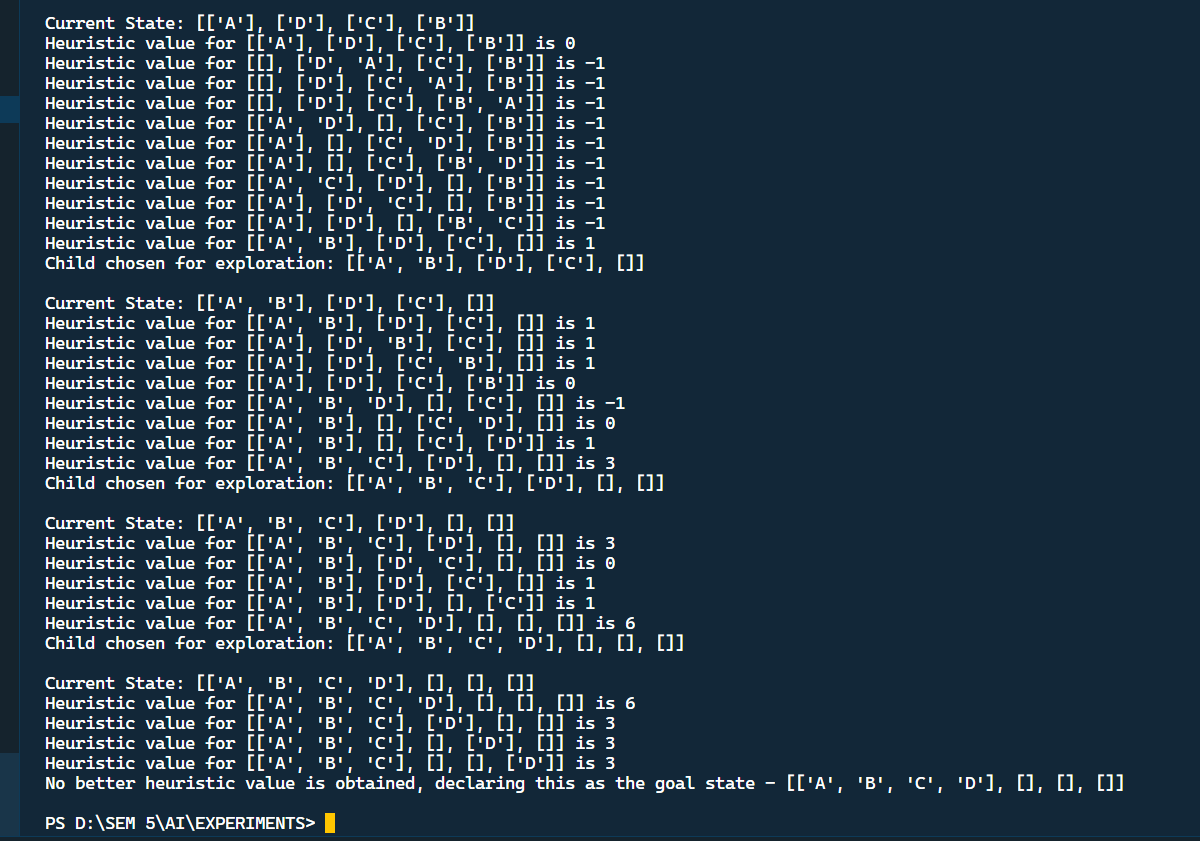
solve\_puzzle(initial\_state, goal\_state)

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Output :**





**Conclusion :**

Thus we successfully studied and implemented Hill-Climbing Search, and solved the block world problem with this algorithm.