NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

ARTIFICIAL INTELLIGENCE LAB

NAME	Ayesha Imran
Class	CS-A
Lab	06
Course	Artificial Intelligence
Date	22-October-25
Submitted To	Lec. Ijlal Haider

IN LAB TASKS

TASK 01:

Implement a Python program that uses the Steepest-Ascent Hill Climbing algorithm to rearrange the letters of a given word to form a target word. The program should allow the user to input both the initial arrangement of letters and the target word.

Initial word → EACHT

Target word →TEACH

CODE:

```
import random

# Generate a random arrangement of the letters (like generate_grid)
def generate_grid(word="EACHT"):
    letters = list(word)
    random.shuffle(letters)
    return letters

# Print the current arrangement (like print_grid)
def print_grid(grid):
    print("Current arrangement:", "".join(grid))
    print()

# Simple Hill Climbing algorithm (same structure as simple_hill_climbing_grid)
def simple_hill_climbing_grid(grid, target):
    current_word = grid[:]  # start with current arrangement
    path = ["".join(current_word)]
    current_score = sum(1 for a, b in zip(current_word, target) if a == b)

    print(f"Starting with word {''.join(current_word)} (Score: {current_score})")

    while True:
        move_made = False  # Track if we made an improvement
```

```
        # Try swapping letters to find a better arrangement
        for i in range(len(current_word)):
            for j in range(i + 1, len(current_word)):
                neighbor = current_word[:]
                neighbor[i], neighbor[j] = neighbor[j], neighbor[i]

                score = sum(1 for a, b in zip(neighbor, target) if a == b)

                # Move immediately if we find any better neighbor
                if score > current_score:
                    current_word = neighbor
                    current_score = score
                    move_made = True
                    path.append("".join(current_word))
                    print(f"Moved to {''.join(current_word)} (Score:
{current_score})")
                    break
            if move_made:
                break

        if not move_made:
            print("No better move found. Stopping.")
            break

        if "".join(current_word) == "".join(target):
            print("Target word reached!")
            break

    print(f"\nPath taken: {path}")
    print(f"Final Word: {''.join(current_word)}")
    print(f"Final Score: {current_score}")

# ------------------------------
# Run the algorithm
# ------------------------------
initial_word = "EACHT"
target_word = list("TEACH")

grid = generate_grid(initial_word)
print("Generated Arrangement:")
print_grid(grid)
simple_hill_climbing_grid(grid, target_word)
```

OUTPUT:

```
Generated Arrangement:
Current arrangement: CEATH

Starting with word CEATH (Score: 3)
Moved to TEACH (Score: 5)
Target word reached!

Path taken: ['CEATH', 'TEACH']
Final Word: TEACH
Final Score: 5
PS D:\Python practice>
```

TASK 02:

Implement above Python program using the Steepest-Ascent Hill Climbing algorithm.

CODE:

```
import random

# Generate a random arrangement of the letters (like the grid)
def generate_grid(word):
    letters = list(word)
    random.shuffle(letters)
    return letters

# Print the current arrangement
def print_grid(grid):
    print("Current arrangement:", "".join(grid))
    print()

def steepest_ascent_hill_climbing(grid, target):
    current_word = grid[:]  # copy
    current_score = sum(1 for a, b in zip(current_word, target) if a == b)
    path = ["".join(current_word)]

    print(f"Starting with word {''.join(current_word)} (Score: {current_score})")

    while True:
        next_moves = []  # all possible neighbor words
```

```
        # Create neighbors by swapping two letters
        for i in range(len(current_word)):
            for j in range(i + 1, len(current_word)):
                neighbor = current_word[:]
                neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
                score = sum(1 for a, b in zip(neighbor, target) if a == b)
                next_moves.append(("".join(neighbor), score))

        # Find neighbor with highest score (best improvement)
        next_position, next_cost = max(next_moves, key=lambda x: x[1])

        if next_cost > current_score:
            current_word = list(next_position)
            current_score = next_cost
            path.append(next_position)
            print(f"Moved to {next_position} (Score: {next_cost})")

            if next_position == "".join(target):
                print("Target word reached!")
                break
        else:
            print("No better move found. Stopping.")
            break

    print(f"\nPath taken: {path}")
    print(f"Final Word: {''.join(current_word)}")
    print(f"Final Score: {current_score}")

# -------------------------
# Run the algorithm
# -------------------------
initial_word = list("EACHT")
target_word = list("TEACH")

print("Initial Word:", "".join(initial_word))
print("Target Word:", "".join(target_word))
print()

grid = generate_grid(initial_word)   # like your random grid
print("Generated Arrangement:")
print_grid(grid)

steepest_ascent_hill_climbing(grid, target_word)
```

OUTPUT:

```
Initial Word: EACHT
Target Word: TEACH

Generated Arrangement:
Current arrangement: EHTCA

Starting with word EHTCA (Score: 1)
Moved to HETCA (Score: 2)
Moved to TEHCA (Score: 3)
Moved to TEACH (Score: 5)
Target word reached!

Path taken: ['EHTCA', 'HETCA', 'TEHCA', 'TEACH']
Final Word: TEACH
Final Score: 5
PS D:\Python practice>
```

POST LAB TASKS

TASK 01:

Implement a Simple Hill Climbing algorithm to optimize the selection of items for the knapsack. You are tasked with maximizing the total value of items that can be carried in a knapsack, given a set of items with specific weights and values. The knapsack has a defined weight capacity.

CODE:

```python
import random

# Define items (value, weight)
items = [
    (10, 5),  # Item 0
    (7, 3),   # Item 1
    (12, 7),  # Item 2
    (8, 4),   # Item 3
```

```
    (6, 2)    # Item 4
]

capacity = 10  # Maximum knapsack weight

# Generate a random starting selection (like generate_grid)
def generate_grid(n):
    return [random.randint(0,1) for _ in range(n)]

# Print current selection
def print_grid(grid):
    print("Current selection:", grid)
    total_weight = sum(items[i][1] for i in range(len(grid)) if grid[i]==1)
    total_value = sum(items[i][0] for i in range(len(grid)) if grid[i]==1)
    print(f"Total weight = {total_weight}, Total value = {total_value}\n")

# Simple Hill Climbing
def simple_hill_climbing_grid(grid):
    n = len(grid)
    current_selection = grid[:]
    path = [current_selection[:]]
    total_value = sum(items[i][0] for i in range(n) if current_selection[i]==1)

    # Calculate initial weight and check capacity
    total_weight = sum(items[i][1] for i in range(n) if current_selection[i]==1)

    print("Starting selection:")
    print_grid(current_selection)

    while True:
        move_made = False

        # Try flipping each item
        for i in range(n):
            neighbor = current_selection[:]
            neighbor[i] = 1 - neighbor[i]  # Flip inclusion/exclusion
            neighbor_weight = sum(items[j][1] for j in range(n) if
neighbor[j]==1)
            neighbor_value = sum(items[j][0] for j in range(n) if neighbor[j]==1)

            # Move only if value improves and weight is within capacity
            if neighbor_value > total_value and neighbor_weight <= capacity:
                current_selection = neighbor
                total_value = neighbor_value
                total_weight = neighbor_weight
```

```
                path.append(current_selection[:])
                move_made = True
                print(f"Moved to better neighbor (flip item {i}):")
                print_grid(current_selection)
                break  # Simple hill climb: move immediately

        if not move_made:
            print("No better neighbor found. Stopping.\n")
            break

    print("Path taken (all selections):")
    for step in path:
        print(step)
    print(f"\nBest selection: {current_selection}")
    print(f"Total weight = {total_weight}, Maximum value = {total_value}")

# Run the algorithm
grid = generate_grid(len(items))
print("Generated Random Selection:")
print_grid(grid)
simple_hill_climbing_grid(grid)
```

OUTPUT:

```
Path taken (all selections):
[1, 1, 0, 1, 1]

Best selection: [1, 1, 0, 1, 1]
Total weight = 14, Maximum value = 31
PS D:\Python practice>
```

TASK 02:

Implement a Steepest-Ascent Hill Climbing algorithm to optimize the route for the Traveling Salesman Problem. Your task is to find the shortest possible route that visits each city exactly once and returns to the origin city using a list of cities and the distances between them.

CODE:

```
import random
import itertools

def calculate_total_distance(route, distance_matrix):
    total_distance = 0
    for i in range(len(route)):
        total_distance += distance_matrix[route[i]][route[(i + 1) % len(route)]]
    return total_distance

def get_neighbors(route):
    neighbors = []
    for i in range(len(route)):
        for j in range(i + 1, len(route)):
            neighbor = route[:]
            neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
            neighbors.append(neighbor)
    return neighbors

def steepest_ascent_hill_climbing(distance_matrix):
    num_cities = len(distance_matrix)
    current_route = list(range(num_cities))
    random.shuffle(current_route)
    current_distance = calculate_total_distance(current_route, distance_matrix)

    while True:
        neighbors = get_neighbors(current_route)
        best_neighbor = current_route
        best_distance = current_distance

        for neighbor in neighbors:
            distance = calculate_total_distance(neighbor, distance_matrix)
            if distance < best_distance:
                best_neighbor = neighbor
                best_distance = distance

        if best_distance < current_distance:
            current_route = best_neighbor
            current_distance = best_distance
        else:
            break

    return current_route, current_distance
```

```python
# Example usage
distance_matrix = [
    [0, 2, 9, 10],
    [1, 0, 6, 4],
    [15, 7, 0, 8],
    [6, 3, 12, 0]
]

route, distance = steepest_ascent_hill_climbing(distance_matrix)
print("Optimized Route:", route)
print("Total Distance:", distance)
```

OUTPUT:

```
Optimized Route: [2, 3, 1, 0]
Total Distance: 21
PS D:\Python practice>
```

END