Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 1

AIM: To implement the A* (A-Star) search algorithm in Python.

REQUIREMENT:

- Python 3
- Basic understanding of graphs and heuristics

CODE:

```
import heapq


class Node:
    def __init__(self, position, parent=None, g=0, h=0):
        self.position = position
        self.parent = parent
        self.g = g  # Cost from start to current node
        self.h = h  # Heuristic cost from current node to goal
        self.f = g + h  # Total cost

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


def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])  # Manhattan distance


def astar(grid, start, goal):
    open_list = []
    closed_set = set()
    start_node = Node(start, None, 0, heuristic(start, goal))
    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)

        if current_node.position == goal:
            path = []
            while current_node:
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

```
            path.append(current_node.position)
            current_node = current_node.parent
        return path[::-1]

    closed_set.add(current_node.position)

    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        neighbor_pos = (current_node.position[0] + dx, current_node.position[1] + dy)

        if (0 <= neighbor_pos[0] < len(grid) and 0 <= neighbor_pos[1] < len(grid[0])
            and grid[neighbor_pos[0]][neighbor_pos[1]] == 0 and neighbor_pos not in closed_set):

            g_cost = current_node.g + 1
            h_cost = heuristic(neighbor_pos, goal)
            neighbor_node = Node(neighbor_pos, current_node, g_cost, h_cost)
            heapq.heappush(open_list, neighbor_node)

    return None  # No path found

# Example grid (0 = open space, 1 = obstacle)
grid = [
    [0, 0, 0, 0, 0],
    [1, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0]
]

start = (0, 0)
goal = (4, 4)
path = astar(grid, start, goal)
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

OUTPUT:

```
The destination cell is found
The Path is
-> (8, 0) -> (7, 0) -> (6, 0) -> (5, 0) -> (4, 1) -> (3, 2) -> (2,
1) -> (1, 0) -> (0, 0)
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 2

AIM: Write a program to implement Single Player Snakes and Ladder Game.

REQUIREMENT:

- Python 3

CODE:

```
class snakesandladder(object):
  def __init__(self, name, position):
    self.name = name
    self.position = position
    self.ladd = [4,24,48,67,86]
    self.lengthladd = [13,23,5,12,13]
    self.snake = [6,26,47,23,55,97]
    self.lengthsnake = [4,6,7,5,8,9]



  def dice(self):
    chances = 0
    print("----------------LeTs StArT ThE GaMe----------------\n")
    while self.position <= 104:
      roll = random.choice([1,2,3,4,5,6])
      print('roll value: ', roll)
      self.position = roll + self.position
      if self.position > 104:
        self.position = self.position - roll
      if self.position == 104:
        print('completed the game')
        break


      if self.position in self.ladd:
        for n in range(len(self.ladd)):
          if self.position == self.ladd[n]:
            self.position = self.position + self.lengthladd[n]
      if self.position in self.snake:
        for n in range(len(self.snake)):
```

```
            if self.position == self.snake[n]:
                self.position = self.position - self.lengthsnake[n]

        print('Current position of the player : ', self.position, '\n')
        chances += 4/4
    print('ToTal number oF chances : ', chances)

zack = snakesandladder('zack',0)
zack.dice()
```

OUTPUT:

```
roll value:  4
Current position of the player :  46

roll value:  2
Current position of the player :  53

roll value:  6
Current position of the player :  59

roll value:  4
Current position of the player :  63

roll value:  6
Current position of the player :  69

roll value:  1
Current position of the player :  70

roll value:  5
Current position of the player :  75

roll value:  5
Current position of the player :  80

roll value:  3
Current position of the player :  83

roll value:  2
Current position of the player :  85
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIEMENT 3

AIM: Write a program to implement Tic-Tac-Toe game problem

REQUIREMENT:

- Python 3
- Basic knowledge of loops and conditionals

CODE:

```
def print_board(board):
  for row in board:
    print(" | ".join(row))
    print("-" * 9)


def check_winner(board, player):
  for row in board:
    if all(cell == player for cell in row):
      return True


  for col in range(3):
    if all(board[row][col] == player for row in range(3)):
      return True


  if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
    return True


  return False


def is_full(board):
  return all(cell != " " for row in board for cell in row)


def tic_tac_toe():
  board = [[" " for _ in range(3)] for _ in range(3)]
  players = ["X", "O"]
  turn = 0


  while True:
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

```
        print_board(board)
        player = players[turn % 2]
        row, col = map(int, input(f"Player {player}, enter row and column (0-2): ").split())

        if 0 <= row < 3 and 0 <= col < 3 and board[row][col] == " ":
            board[row][col] = player
            if check_winner(board, player):
                print_board(board)
                print(f"Player {player} wins!")
                break
            if is_full(board):
                print_board(board)
                print("It's a tie!")
                break
            turn += 1
        else:
            print("Invalid move! Try again.")

tic_tac_toe()
```

OUTPUT:

```
 |  |
---------
 |  |
---------
 |  |
Player X, enter row and column (0-2): 0 0

X |  |
---------
 |  |
---------
 |  |
Player O, enter row and column (0-2): 1 1

X |  |
---------
 | O |
---------
 |  |
Player X, enter row and column (0-2): 0 1

X | X |
---------
 | O |
---------
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

```
 |  |
Player O, enter row and column (0-2): 2 2

X | X |
---------
 | O |
---------
 |  | O
Player X, enter row and column (0-2): 0 2

X | X | X
---------
 | O |
---------
 |  | O
Player X wins!
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 4

AIM: To implement a brute-force solution to the 0/1 Knapsack Problem using Python.

REQUIREMENT:

- Python 3
- Basic understanding of recursion and backtracking

CODE:

```
def knapsack_brute_force(weights, values, capacity, n):
    if n == 0 or capacity == 0:
        return 0

    if weights[n - 1] > capacity:
        return knapsack_brute_force(weights, values, capacity, n - 1)

    include_item = values[n - 1] + knapsack_brute_force(weights, values, capacity - weights[n - 1], n - 1)
    exclude_item = knapsack_brute_force(weights, values, capacity, n - 1)

    return max(include_item, exclude_item)

# Example usage
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
capacity = 5
n = len(weights)

max_value = knapsack_brute_force(weights, values, capacity, n)
print("Maximum value in knapsack:", max_value)
```

OUTPUT:

```
Maximum value in knapsack: 7
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 5

AIM: To implement the Graph Coloring Problem using Python

REQUIREMENT:

- Python 3
- Basic understanding of graphs and backtracking

CODE:

```
def is_safe(graph, colors, node, color):
  for neighbor in range(len(graph)):
    if graph[node][neighbor] == 1 and colors[neighbor] == color:
      return False
  return True

def graph_coloring(graph, m, colors, node=0):
  if node == len(graph):
    return True  # All nodes are successfully colored

  for color in range(1, m + 1):  # Try colors from 1 to m
    if is_safe(graph, colors, node, color):
      colors[node] = color
      if graph_coloring(graph, m, colors, node + 1):
        return True
      colors[node] = 0  # Backtrack if coloring fails

  return False  # No valid color found

def solve_graph_coloring(graph, m):
  colors = [0] * len(graph)  # Initialize all nodes with no color
  if graph_coloring(graph, m, colors):
    return colors
  else:
    return "No solution exists"

# Example Usage
graph = [
  [0, 1, 1, 1],
  [1, 0, 1, 0],
  [1, 1, 0, 1],
  [1, 0, 1, 0]
]
m = 3  # Number of colors

solution = solve_graph_coloring(graph, m)
print("Color assignment:", solution)
```

OUTPUT:

```
Color assignment: [1, 2, 3, 2]
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 6

AIM: To implement the Water Jug Problem using Breadth-First Search (BFS) in Python.

REQUIREMENT:

- Python 3
- Basic understanding of graphs and BFS

CODE:

```
from collections import deque

def water_jug_bfs(capacity_x, capacity_y, target):
  visited = set()  # Track visited states
  queue = deque()   # BFS queue

  # Initial state: (0, 0)
  queue.append((0, 0))

  while queue:
    x, y = queue.popleft()

    # If the target amount is reached in any jug
    if x == target or y == target:
      return f"Solution found: ({x}, {y})"

    # If state is already visited, skip it
    if (x, y) in visited:
      continue

    visited.add((x, y))

    # Possible operations
    possible_moves = [
      (capacity_x, y),    # Fill jug X
      (x, capacity_y),    # Fill jug Y
      (0, y),            # Empty jug X
      (x, 0),            # Empty jug Y
      (x - min(x, capacity_y - y), y + min(x, capacity_y - y)),  # Pour X → Y
      (x + min(y, capacity_x - x), y - min(y, capacity_x - x))   # Pour Y → X
    ]

    for move in possible_moves:
      if move not in visited:
        queue.append(move)

  return "No solution found"

# Example Usage
jug_x = 4  # Capacity of Jug X
jug_y = 3  # Capacity of Jug Y
target = 2  # Target amount

result = water_jug_bfs(jug_x, jug_y, target)
print(result)
```

OUTPUT:

```
Solution found: (2, 0)
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 7

AIM: To implement Depth-First Search (DFS) in Python for graph traversal.

REQUIREMENT:

- Python 3
- Basic understanding of graphs and DFS

CODE:

```
# DFS using recursion
def dfs_recursive(graph, node, visited):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for neighbor in graph.get(node, []):
            dfs_recursive(graph, neighbor, visited)

# DFS using stack (iterative)
def dfs_iterative(graph, start):
    visited = set()
    stack = [start]

    while stack:
        node = stack.pop()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            stack.extend(reversed(graph.get(node, [])))  # Reverse to maintain order

# Example Graph (Adjacency List)
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}

# Running DFS
print("DFS Recursive:")
dfs_recursive(graph, 'A', set())

print("\nDFS Iterative:")
dfs_iterative(graph, 'A')
```

OUTPUT:

```
DFS Recursive:
A B D E F C

DFS Iterative:
A B D E F C
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 8

AIM: To perform word and sentence tokenization using the NLTK (Natural Language Toolkit) package in Python.

REQUIREMENT:

- Python 3
- Install NLTK package:
- Import necessary modules from NLTK

CODE:

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

# Download necessary NLTK data
nltk.download('punkt')

# Sample text
text = "Hello! Welcome to the world of Natural Language Processing. Tokenization splits text into words and sentences."

# Sentence Tokenization
sentences = sent_tokenize(text)
print("Sentence Tokenization:")
print(sentences)

# Word Tokenization
words = word_tokenize(text)
print("\nWord Tokenization:")
print(words)
```

OUTPUT

```
Sentence Tokenization:
['Hello!', 'Welcome to the world of Natural Language Processing.', 'Tokenization splits text

Word Tokenization:
['Hello', '!', 'Welcome', 'to', 'the', 'world', 'of', 'Natural', 'Language', 'Processing', '.
```

Akshara Sharma
A023167022060
6 CSE – DS 1Y

EXPERIMENT 9

AIM: To design the XOR Truth Table using Python.

REQUIREMENT:

- Python 3
- Basic understanding of XOR operation

CODE:

```
# Function to print XOR Truth Table
def xor_truth_table():
    print("A | B | A ⊕ B")
    print("--|---|------")
    for a in [0, 1]:
        for b in [0, 1]:
            xor_result = a ^ b  # XOR operation
            print(f"{a} | {b} |  {xor_result}")

# Generate XOR Truth Table
xor_truth_table()
```

OUTPUT:

```
A | B | A ⊕ B
--|---|------
0 | 0 |  0
0 | 1 |  1
1 | 0 |  1
1 | 1 |  0
```