KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AI Lab Manual

Subject Code:KG23ACM317

Regulation: KG23

Academic Year: 2025-26

III B.TECH II SEMESTER

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated To JNTUH, Chilkur(V), Moinabad (M), Pincode:501504, R.R Dist, Telangana.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with government organizations and collaboration with industries
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development. .

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

- To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional, entrepreneurial and societal goals.

MISSION:

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societal issues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.

PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.

PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.

PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

• **PO I: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
• **PO II: Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
• **PO III: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
• **PO IV: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
• **PO V: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
• **PO VI: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
• **PO VII: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.

• **PO VIII: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
• **PO IX: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
• **PO X: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
• **PO XI: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
• **PO XII: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

List of Experiments

S.No	Experiment
1	Write a program to implement Breadth First Search (BFS) using python
2	Write a program to implement Depth First search algorithm
3	Write a program to implement Tic-Tac-Toe Game
4	Write a program to implement 8-Puzzle Problem
5	Write a program to implement Water Jug Problem
6	Write a program to implement Travelling Salesman Problem
7	Write a program to implement Tower of Hanoi.
8	Write a program to implement Monkey Banana Problem
9	Write a program to implement Alpha Beta Pruning
10	Write a program to implement 8 Queens Problem

1. Write a program to implement Breadth First Search (BFS) using python

CODE:

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)
   while queue:
        node = queue.popleft()
        print(node, end=" ")
  for neighbor in graph[node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': [],
    'F': []
}
bfs(graph, 'A')
```

OUTPUT: A B C D E F

2. Write a program to implement Depth First search algorithm

CODE:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
```

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': [],
    'F': []
}

dfs(graph, 'A')
```

OUTPUT: A B D E C F

3. Write a program to implement Tic-Tac-Toe Game

CODE:

```
board = [' '] * 9
def print_board():
    for i in range(0, 9, 3):
        print(board[i], '|', board[i+1], '|', board[i+2])
    print()

def check_win(p):
    wins = [
        (0,1,2),(3,4,5),(6,7,8),
        (0,3,6),(1,4,7),(2,5,8),
        (0,4,8),(2,4,6)
    ]
    return any(all(board[i] == p for i in w) for w in wins)

player = 'X'

for turn in range(9):
    print_board()
    move = int(input(f"Player {player}, enter position (0-8): "))

    if board[move] != ' ':
        print("Position already taken! Try again.")
        continue

    board[move] = player
```

```
    if check_win(player):
        print_board()
        print(f"🎉 Player {player} wins!")
        break

    player = 'O' if player == 'X' else 'X'
else:
    print_board()
    print("🤝 It's a draw!")
```

OUTPUT:

```
  |   |
  |   |
  |   |

Player X, enter position (0-8): 0
X |   |
  |   |
  |   |

Player O, enter position (0-8): 4
X |   |
  | O |
  |   |

Player X, enter position (0-8): 1
X | X |
  | O |
  |   |

Player O, enter position (0-8): 3
X | X |
O | O |
  |   |
```

4. Write a program to implement 8-Puzzle Problem

CODE:

```
from collections import deque

def bfs(start_state):
    target = [1, 2, 3, 4, 5, 6, 7, 8, 0]
    dq = deque([start_state])
    visited = {tuple(start_state): None}
```

```
    while dq:
        state = dq.popleft()

        if state == target:
            path = []
            while state is not None:
                path.append(state)
                state = visited[tuple(state)]
            return path[::-1]

        zero = state.index(0)
        row, col = divmod(zero, 3)

        for move in (-3, 3, -1, 1):
            new_pos = zero + move
            if 0 <= new_pos < 9:
                new_row, new_col = divmod(new_pos, 3)
                if abs(row - new_row) + abs(col - new_col) == 1:
                    neighbor = state[:]
                    neighbor[zero], neighbor[new_pos] = neighbor[new_pos], neighbor[zero]

                    if tuple(neighbor) not in visited:
                        visited[tuple(neighbor)] = state
                        dq.append(neighbor)

    return None

def printSolution(path):
    for state in path:
        for i in range(0, 9, 3):
            print(*state[i:i+3])
        print("-----")

# Example Usage
startState = [1, 3, 0, 6, 8, 4, 7, 5, 2]
solution = bfs(startState)

if solution:
    printSolution(solution)
    print(f"Solved in {len(solution) - 1} moves.")
else:
    print("No solution found.")
```

OUTPUT:

```
1 3 0
6 8 4
7 5 2
-----
1 3 4
6 8 0
7 5 2
-----
1 3 4
6 8 2
7 5 0
-----
1 3 4
6 8 2
7 0 5
-----
1 3 4
6 0 2
7 8 5
-----
1 3 4
0 6 2
7 8 5
-----
1 3 4
7 6 2
0 8 5
-----
1 3 4
7 6 2
8 0 5
-----
1 3 4
7 0 2
8 6 5
-----
1 3 4
7 2 0
8 6 5
-----
1 3 0
7 2 4
8 6 5
```

---
1 0 3
7 2 4
8 6 5
---
1 2 3
7 0 4
8 6 5
---
1 2 3
7 4 0
8 6 5
---
1 2 3
7 4 5
8 6 0
---
1 2 3
7 4 5
8 0 6
---
1 2 3
7 4 5
0 8 6
---
1 2 3
0 4 5
7 8 6
---
1 2 3
4 0 5
7 8 6
---
1 2 3
4 5 0
7 8 6
---
1 2 3
4 5 6
7 8 0

Solved in 20 moves.

5. Write a program to implement Water Jug Problem

CODE:

```
jug1 = 4
jug2 = 3
goal = 2

visited = [[0 for j in range(jug2 + 1)] for i in range(jug1 + 1)]

def waterJug(a, b):
    if a < 0 or b < 0 or a > jug1 or b > jug2:
        return False

    if (a == goal and b == 0) or (b == goal and a == 0):
        print(a, b)
        print("Solution Found")
        return True

    if visited[a][b] == 1:
        return False

    visited[a][b] = 1
    print(a, b)

    if waterJug(0, b):
        return True
    if waterJug(a, 0):
        return True
    if waterJug(jug1, b):
        return True
    if waterJug(a, jug2):
        return True

    x = min(b, jug1 - a)
    if waterJug(a + x, b - x):
        return True

    y = min(a, jug2 - b)
    if waterJug(a - y, b + y):
        return True
  return False
print("Steps:")
print("Jug1 Jug2")
```

```
waterJug(0, 0)
```

OUTPUT:

```
Steps:
Jug1 Jug2
0 0
4 0
4 3
0 3
3 0
3 3
4 2
0 2
Solution Found
```

6. Write a program to implement Travelling Salesman Problem

CODE:

```
from collections import deque

def tsp_bfs(graph):
    n = len(graph)
    startCity = 0
    min_cost = float('inf')
    opt_path = []

    dq = deque([([startCity], 0)])

    while dq:
        cur_path, cur_cost = dq.popleft()
        cur_city = cur_path[-1]

        if len(cur_path) == n:
            total_cost = cur_cost + graph[cur_city][startCity]
            if total_cost < min_cost:
                min_cost = total_cost
                opt_path = cur_path + [startCity]
            continue

        for next_city in range(n):
            if next_city not in cur_path:
                new_path = cur_path + [next_city]
                new_cost = cur_cost + graph[cur_city][next_city]
```

```
                dq.append((new_path, new_cost))
    return min_cost, opt_path
# Example graph
graph = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]

min_cost, opt_path = tsp_bfs(graph)
print("Optimal Solution:")
print("Minimum cost:", min_cost)
print("Optimal path:", opt_path)
```

OUTPUT:

```
Optimal Solution:
Minimum cost: 80
Optimal path: [0, 1, 3, 2, 0]
```

7. Write a program to implement Tower of Hanoi.

CODE:

```
def tower_of_hanoi(num, source, aux, target):
    """
    num (int): Number of disks
    source (str): Source tower
    aux (str): Auxiliary tower
    target (str): Target tower
    """
    if num == 1:
        print(f"Move disk 1 from {source} to {target}")
        return

    # Move num-1 disks from source to auxiliary
    tower_of_hanoi(num - 1, source, target, aux)

    print(f"Move disk {num} from {source} to {target}")

    # Move num-1 disks from auxiliary to target
    tower_of_hanoi(num - 1, aux, source, target)


# Example usage
```

```
num_disks = 3
tower_of_hanoi(num_disks, "A", "B", "C")
```

OUTPUT:

```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

8. Write a program to implement Monkey Banana Problem

CODE:

```
  def monkey_banana():
    monkey_pos = "A"
    box_pos = "A"
    banana_pos = "B"
    monkey_on_box = False
    banana_grabbed = False

    print("Initial State:")
    print("Monkey at", monkey_pos)
    print("Box at", box_pos)
    print("Banana at", banana_pos)
    print()
    print("Monkey moves to position B")
    monkey_pos = "B"
    print("Monkey pushes the box to position B")
    box_pos = "B"

    if monkey_pos == box_pos:
        print("Monkey climbs the box")
        monkey_on_box = True

    if monkey_on_box and box_pos == banana_pos:
        print("Monkey grabs the banana □")
        banana_grabbed = True
    print()
    if banana_grabbed:
        print("✓Success: Monkey got the banana!")
```

```
    else:
        print("❌Failure: Monkey could not get the banana.")
monkey_banana()
```

OUTPUT:

Initial State:
Monkey at A
Box at A
Banana at B

Monkey moves to position B
Monkey pushes the box to position B
Monkey climbs the box
Monkey grabs the banana

✅Success: Monkey got the banana!

9. Write a program to implement Alpha Beta Pruning

OUTPUT:

```
import math
def alphabeta(depth, nodeIndex, isMax, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]
 if isMax:
        best = -math.inf
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i, False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)
            if beta <= alpha:
                break
        return best
    else:
        best = math.inf
        for i in range(2):
            val = alphabeta(depth + 1, nodeIndex * 2 + i, True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)
            if beta <= alpha:
                break
        return best
values = [3, 5, 6, 9, 1, 2, 0, -1]
result = alphabeta(0, 0, True, values, -math.inf, math.inf)
print("The optimal value is:", result)
```

OUTPUT:
The optimal value is: 5

10. Write a program to implement 8 Queens Problem.

CODE:

```
N = 8
def is_safe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False
    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j] == 1:
            return False
        i -= 1
        j -= 1
    i, j = row, col
    while i < N and j >= 0:
        if board[i][j] == 1:
            return False
        i += 1
        j -= 1
    return True

def solve_queens(board, col):
    if col >= N:
        return True
    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_queens(board, col + 1):
                return True
            board[i][col] = 0
    return False

def print_board(board):
    for row in board:
        print(row)

board = [[0 for _ in range(N)] for _ in range(N)]

if solve_queens(board, 0):
    print("One solution to the 8-Queens problem:")
```

```
    print_board(board)
else:
    print("No solution exists")
```

OUTPUT:

```
One solution to the 8-Queens problem:
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
```