**Practical No. 1**

**Title -: Implement DFS and BFS Algorithm. Use and Undirected Graph and develop a Recursive Algorithm for searching all the vertices of the graph or tree data structure.**

**Code -:**

***# Breadth First Search:***

**graph = {**

**'1' : ['2','5'],**

**'2' : ['3', '4'],**

**'5' : ['6'],**

**'3' : [],**

**'4' : ['6'],**

**'6' : []**

**}**

**visited = []**

**queue = []**

**def breadthFirstSearch(visited, graph, node):**

**visited.append(node)**

**queue.append(node)**

**while queue:**

**m = queue.pop(0)**

**print (m, end = " ")**

**for neighbour in graph[m]:**

**if neighbour not in visited:**

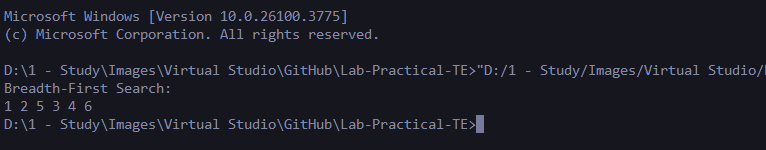
**visited.append(neighbour)**

**queue.append(neighbour)**

**print("Breadth-First Search: ")**

**breadthFirstSearch(visited, graph, '1')**

**Output -:**

****

***# Depth First Search:***

***graph = {***

***'1' : ['2','5'],***

***'2' : ['3', '4'],***

***'5' : ['6'],***

***'3' : [],***

***'4' : ['6'],***

***'6' : []***

***}***

***visited = set()***

***def depthFirstSearch(visited, graph, node):***

***if node not in visited:***

***print (node)***

***visited.add(node)***

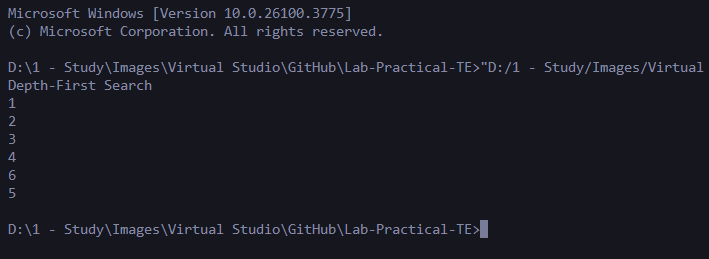
***for neighbour in graph[node]:***

***depthFirstSearch(visited, graph, neighbour)***

***print("Depth-First Search")***

***depthFirstSearch(visited, graph, '1')***

**Output -:**

****

**Practical No. 2**

**Title -: Implement A\* Algorithm for any game search problem**

**Code -:**

**import heapq**

**def a\_star(grid, start, goal):**

**def heuristic(a, b):**

**return abs(a[0] - b[0]) + abs(a[1] - b[1])**

**rows, cols = len(grid), len(grid[0])**

**open\_set = []**

**heapq.heappush(open\_set, (0, start))**

**came\_from = {}**

**g\_score = {start: 0}**

**while open\_set:**

**\_, current = heapq.heappop(open\_set)**

**if current == goal:**

**path = []**

**while current in came\_from:**

**path.append(current)**

**current = came\_from[current]**

**path.append(start)**

**return path[::-1]**

**for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:**

**neighbor = (current[0] + dx, current[1] + dy)**

**if 0 <= neighbor[0] < rows and 0 <= neighbor[1] < cols:**

**if grid[neighbor[0]][neighbor[1]] == 1:**

**continue**

**tentative\_g = g\_score[current] + 1**

**if neighbor not in g\_score or tentative\_g < g\_score[neighbor]:**

**g\_score[neighbor] = tentative\_g**

**f\_score = tentative\_g + heuristic(neighbor, goal)**

**heapq.heappush(open\_set, (f\_score, neighbor))**

**came\_from[neighbor] = current**

**return None**

**grid = [**

**[0, 0, 0, 0],**

**[1, 1, 0, 1],**

**[0, 0, 0, 0],**

**[0, 1, 1, 0],**

**]**

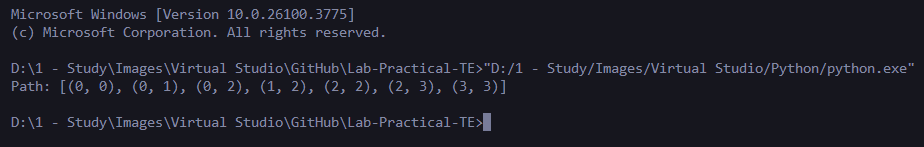
**start = (0, 0)**

**goal = (3, 3)**

**path = a\_star(grid, start, goal)**

**print("Path:", path)**

**Output -:**

****

**Practical No. 3**

**Title -: Implement Greedy search algorithm for Selection Sort.**

**Code -:**

**def selection\_sort\_greedy(arr):**

**n = len(arr)**

**print("\nList before Sorting: ", arr,"\n")**

**for i in range(n):**

**min\_idx = i**

**for j in range(i+1, n):**

**if arr[j] < arr[min\_idx]:**

**min\_idx = j**

**arr[i], arr[min\_idx] = arr[min\_idx], arr[i]**

**print("List After Pass ",i+1,": ",arr)**

**return arr**

**n=int(input("Length of List: "))**

**arr=[]**

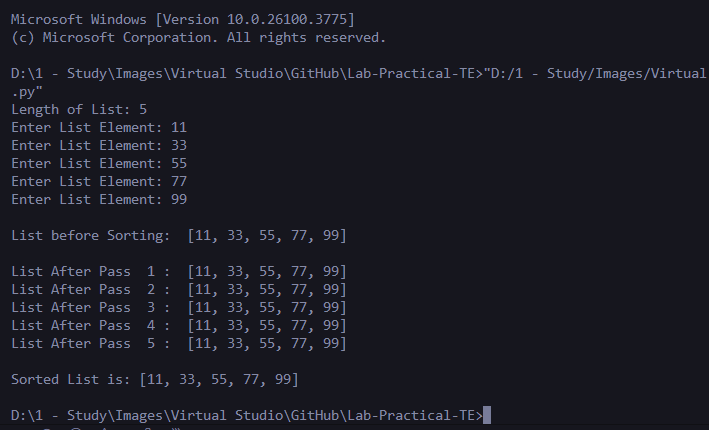
**for i in range(n):**

**element=int(input("Enter List Element: "))**

**arr.append(element)**

**print("\nSorted List is:", selection\_sort\_greedy(arr))**

**Output -:**

****

**Practical No. 4**

**Title -: Implement a solution for a Constraint Satisfaction Problem using Branch and Bound and Backtracking for n-queens problem or a graph coloring problem.**

**Code -:**

**def is\_safe(queens, row, col):**

**for r in range(row):**

**c = queens[r]**

**if c == col or abs(c - col) == abs(r - row):**

**return False**

**return True**

**def solve\_n\_queens(n):**

**solutions = []**

**def backtrack(row, queens):**

**if row == n:**

**solutions.append(queens[:])**

**return**

**for col in range(n):**

**if is\_safe(queens, row, col):**

**queens[row] = col**

**backtrack(row + 1, queens)**

**backtrack(0, [-1] \* n)**

**return solutions**

**def print\_board(solution):**

**for row in solution:**

**print(' '.join('Q' if i == row else '.' for i in range(len(solution))))**

**print()**

**n = 8**

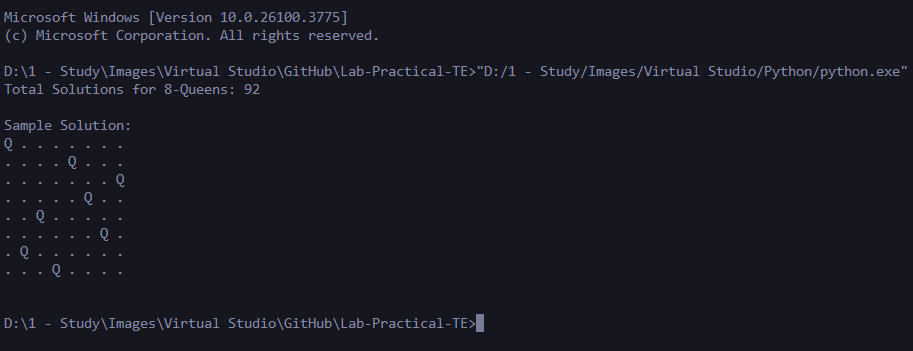
**all\_solutions = solve\_n\_queens(n)**

**print(f"Total Solutions for {n}-Queens: {len(all\_solutions)}\n")**

**print("Sample Solution:")**

**print\_board(all\_solutions[0])**

**Output -:**

****

**Practical No. 5**

**Title -: Develop an elementary chatbot for any suitable customer interaction application.**

**Code -:**

**def pizza\_bot():**

**print("🍕 Welcome to PizzaBot!")**

**print("How can I help you today? (Type 'quit' to exit)\n")**

**while True:**

**user\_input = input("You: ").lower()**

**if "quit" in user\_input or "bye" in user\_input:**

**print("PizzaBot: Thanks for chatting! Have a cheesy day!")**

**break**

**elif "menu" in user\_input or "show" in user\_input:**

**print("PizzaBot: Here's our menu:\n- Margherita\n- Pepperoni\n- Veggie\n- BBQ Chicken")**

**elif "order" in user\_input or "want" in user\_input:**

**print("PizzaBot: Great! What pizza would you like to order?")**

**elif "margherita" in user\_input:**

**print("PizzaBot: Margherita pizza added to your order!")**

**elif "pepperoni" in user\_input:**

**print("PizzaBot: Pepperoni pizza added to your order!")**

**elif "veggie" in user\_input:**

**print("PizzaBot: Veggie pizza added to your order!")**

**elif "bbq" in user\_input or "chicken" in user\_input:**

**print("PizzaBot: BBQ Chicken pizza added to your order!")**

**elif "price" in user\_input or "cost" in user\_input:**

**print("PizzaBot: All pizzas are ₹299 each.")**

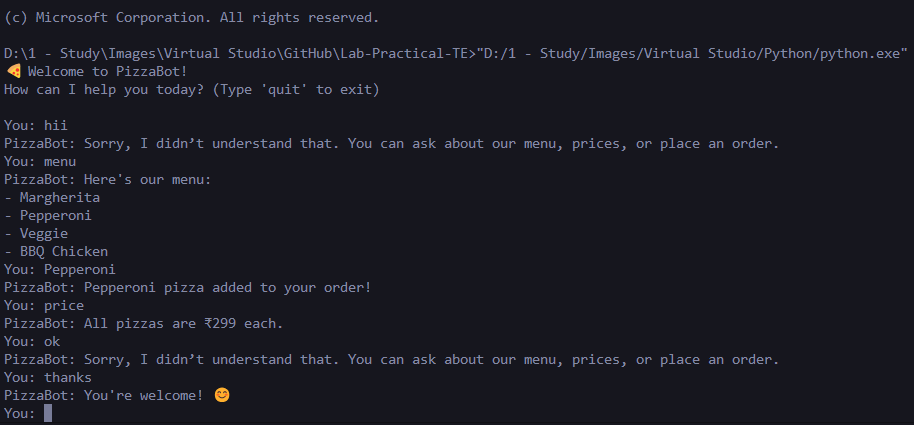
**elif "thanks" in user\_input or "thank you" in user\_input:**

**print("PizzaBot: You're welcome! 😊")**

**else:**

**print("PizzaBot: Sorry, I didn’t understand that. You can ask about our menu, prices, or place an order.")**

**pizza\_bot()**

**Output -:**

**Practical No. 6**

**Title -: Implement any one of the following Expert System**

1. **Information management**
2. **Hospitals and medical facilities**
3. **Help desks management**
4. **Employee performance evaluation**
5. **Stock market trading**
6. **Airline scheduling and cargo schedules.**

**Code -:**

**def evaluate\_performance(attendance, projects\_completed, teamwork\_score):**

***# Rule-based evaluation***

**if attendance >= 90 and projects\_completed >= 5 and teamwork\_score >= 8:**

**return "Excellent"**

**elif attendance >= 80 and projects\_completed >= 3 and teamwork\_score >= 6:**

**return "Good"**

**elif attendance >= 70 and projects\_completed >= 2 and teamwork\_score >= 5:**

**return "Average"**

**else:**

**return "Needs Improvement"**

**print("Employee Performance Evaluation System")**

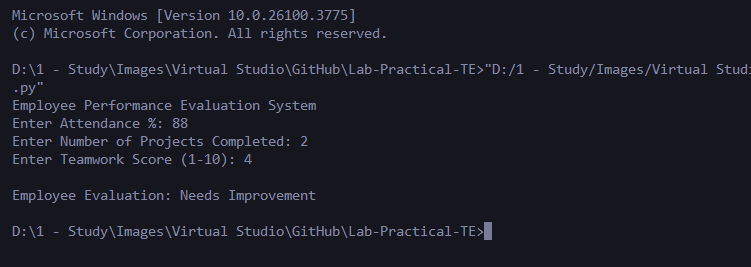
**attendance = int(input("Enter Attendance %: "))**

**projects = int(input("Enter Number of Projects Completed: "))**

**teamwork = int(input("Enter Teamwork Score (1-10): "))**

**result = evaluate\_performance(attendance, projects, teamwork)**

**print("\nEmployee Evaluation:", result)**

**Output -:**