DFS/BFS

```
#include <bits/stdc++.h>
using namespace std;
//adj_mat,n,0,visited
void dfs(int adj_mat[10][10],int n,int node,int visited[10])
{
	cout<<"DFS"<<endl;
//	return;
	stack<int> stk;
	stk.push(node);
	visited[node] = 1;
	while(!stk.empty()){
		int i = stk.top();
		stk.pop();
		cout<<i<<"-->";
		for(int j=n-1;j>=0;j--){
			if(adj_mat[i][j]==1 && visited[j]!=1){
				stk.push(j);
				visited[j] = 1;
			}
		}
	}
	cout<<"NULL"<<endl;
}
void bfs(int adj_mat[10][10],int n,int node,int visited[10])
{
	cout<<"BFS"<<endl;
//	return;
	queue<int> q;
	q.push(node);
	visited[node] = 1;
	while(!q.empty()){
		int i = q.front();
		q.pop();

		cout<<i<<"-->";
```

```
		for(int j=0;j<n;j++){
			if(adj_mat[i][j]==1 && visited[j]!=1){
				q.push(j);
				visited[j] = 1;
			}
		}
	}
	cout<<"NULL"<<endl;
}
int main() {
	int n;
	int adj_mat[10][10] = {0},visited[10] = {0};
	cout<<"Enter the total number of nodes in the graph --> ";
	cin>>n;
	for(int i=0;i<n;i++){
		for(int j=i+1;j<n;j++){
			cout<<"Do you want to add the Edge between "<<i<<" and "<<j<<"?. (Y or N) ";
			char s;	cin>>s;
			if(s=='y' || s=='Y'){
				adj_mat[i][j] = adj_mat[j][i] = 1;
			}
		}
	}
	int ch;
	do{
		cout<<"\n\n";
		cout<<"1. DFS"<<endl;
		cout<<"2. BFS"<<endl;
		cout<<"0. Exit"<<endl;
		cout<<"Enter Choice:- ";
		cin>>ch;
		switch(ch){
			case 1:
				for(int i=0;i<n;i++)		visited[i] = 0;
				cout<<"DFS on the given graph is :- ";
				dfs(adj_mat,n,0,visited);
```

```
                break;
            case 2:
                for(int i=0;i<n;i++)        visited[i] = 0;
                cout<<"BFS on the given graph is :- ";
                bfs(adj_mat,n,0,visited);
                break;
            case 0:
                break;
            default:
                cout<<"Invalid Choice \n\n";
            break;
        }

    }while(ch!=0);

    cout<<"Program Finished ";
    return 0;
}
```

OUTPUT

```
C:\Users\Sanket\OneDrive\Dc

Enter the total number of nodes in the graph --> 7
Do you want to add the Edge between 0 and 1?. (Y or N) y
Do you want to add the Edge between 0 and 2?. (Y or N) y
Do you want to add the Edge between 0 and 3?. (Y or N) y
Do you want to add the Edge between 0 and 4?. (Y or N) n
Do you want to add the Edge between 0 and 5?. (Y or N) y
Do you want to add the Edge between 0 and 6?. (Y or N) n
Do you want to add the Edge between 1 and 2?. (Y or N) y
Do you want to add the Edge between 1 and 3?. (Y or N) n
Do you want to add the Edge between 1 and 4?. (Y or N) y
Do you want to add the Edge between 1 and 5?. (Y or N) n
Do you want to add the Edge between 1 and 6?. (Y or N) y
Do you want to add the Edge between 2 and 3?. (Y or N) n
Do you want to add the Edge between 2 and 4?. (Y or N) y
Do you want to add the Edge between 2 and 5?. (Y or N) n
Do you want to add the Edge between 2 and 6?. (Y or N) y
Do you want to add the Edge between 3 and 4?. (Y or N) n
Do you want to add the Edge between 3 and 5?. (Y or N) y
Do you want to add the Edge between 3 and 6?. (Y or N) n
Do you want to add the Edge between 4 and 5?. (Y or N) y
Do you want to add the Edge between 4 and 6?. (Y or N) y
Do you want to add the Edge between 5 and 6?. (Y or N) y


1. DFS
2. BFS
0. Exit
Enter Choice:- 1
DFS on the given graph is :- DFS
0-->1-->4-->6-->2-->3-->5-->NULL


1. DFS
2. BFS
0. Exit
Enter Choice:- 2
BFS on the given graph is :- BFS
0-->1-->2-->3-->5-->4-->6-->NULL
```

2- Selection sort(cpp)

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    int n;
    cout << "Enter the total number of elements => ";
    cin >> n;
    vector<int> arr(n);
    cout << "Enter " << n << " numbers:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    // Selection Sort
    for (int i = 0; i < n - 1; i++) {
        int minPos = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minPos]) {
                minPos = j;
            }
        }
        if (minPos != i) {
            swap(arr[i], arr[minPos]);
        }
    }
    cout << "------ Sorted Array is ------\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT

```
C:\Users\Sanket\OneDrive\Dc

Enter the total number of elements => 8
Enter 8 numbers:
89
21
1
10
99
67
56
34
------ Sorted Array is ------
1 10 21 34 56 67 89 99

--------------------------------
Process exited after 49.68 seconds with return value 0
Press any key to continue . . .
```

Selection sort(java)

```
public class SelectionSort {
    public static void selectionSort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (array[j] < array[minIndex]) {
                    minIndex = j;
                }
            }
            // Swap if a smaller element was found
            if (minIndex != i) {
                int temp = array[minIndex];
                array[minIndex] = array[i];
                array[i] = temp;
            }
        }
    }
    public static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int[] array = {99,56,34,4,10,100,77,85,54,32};
        System.out.println("Original array:");
        printArray(array);
        selectionSort(array);
        System.out.println("Sorted array:");
        printArray(array);
    }
}
```

OUTPUT

```
Output

Original array:
99 56 34 4 10 100 77 85 54 32
Sorted array:
4 10 32 34 54 56 77 85 99 100

=== Code Execution Successful ===
```

2.2Minimum Spanning Tree

```
import java.util.*;
class Edge implements Comparable<Edge> {
    int source, destination, weight;
    public Edge(int source, int destination, int weight) {
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }
    @Override
    public int compareTo(Edge other) {
        return Integer.compare(this.weight, other.weight);
    }
}
class DisjointSet {
    int[] parent, rank;
    public DisjointSet(int n) {
        parent = new int[n];
        rank = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }
    int find(int v) {
        if (parent[v] != v) {
            parent[v] = find(parent[v]);
        }
        return parent[v];
    }
    void union(int u, int v) {
        int rootU = find(u);
        int rootV = find(v);
        if (rootU != rootV) {
            if (rank[rootU] < rank[rootV]) {
                parent[rootU] = rootV;
            } else if (rank[rootU] > rank[rootV]) {
```

```
            parent[rootV] = rootU;
        } else {
            parent[rootV] = rootU;
            rank[rootU]++;
        }
    }
  }
}
public class Main {
  public static void kruskalAlgorithm(List<Edge> edges, int vertices) {
    DisjointSet ds = new DisjointSet(vertices);
    Collections.sort(edges);
    List<Edge> mst = new ArrayList<>();
    for (Edge edge : edges) {
      if (ds.find(edge.source) != ds.find(edge.destination)) {
        mst.add(edge);
        ds.union(edge.source, edge.destination);
      }
    }
    int totalWeight = 0;
    System.out.println("Edges in the Minimum Spanning Tree:");
    for (Edge edge : mst) {
      System.out.println(edge.source + " - " + edge.destination + " : " + edge.weight);
      totalWeight += edge.weight;
    }
    System.out.println("Total Weight of the MST: " + totalWeight);
  }
  public static void main(String[] args) {
    int vertices = 4;
    List<Edge> edges = new ArrayList<>();
    edges.add(new Edge(0, 1, 13));
    edges.add(new Edge(0, 2, 7));
    edges.add(new Edge(0, 3, 8));
    edges.add(new Edge(1, 3, 17));
    edges.add(new Edge(2, 3, 6));
 kruskalAlgorithm(edges, vertices);  }}
```

OUTPUT

```
Edges in the Minimum Spanning Tree:
2 - 3 : 6
0 - 2 : 7
0 - 1 : 13
Total Weight of the MST: 26


...Program finished with exit code 0
Press ENTER to exit console.
```

Djikstra Algorithm

```
import java.util.*;

class Node implements Comparable<Node> {
    int id, distance;

    public Node(int id, int distance) {
        this.id = id;
        this.distance = distance;
    }

    @Override
    public int compareTo(Node other) {
        return Integer.compare(this.distance, other.distance);
    }
}

public class Main {
    public static void dijkstra(int[][] graph, int source) {
        int n = graph.length;
        int[] distances = new int[n];
        boolean[] visited = new boolean[n];
        Arrays.fill(distances, Integer.MAX_VALUE);
        distances[source] = 0;

        PriorityQueue<Node> pq = new PriorityQueue<>();
        pq.add(new Node(source, 0));

        while (!pq.isEmpty()) {
            Node current = pq.poll();
            int u = current.id;

            if (visited[u]) continue;
            visited[u] = true;
```

```
            for (int v = 0; v < n; v++) {
                int weight = graph[u][v];
                if (weight > 0 && !visited[v]) {
                    int newDist = distances[u] + weight;
                    if (newDist < distances[v]) {
                        distances[v] = newDist;
                        pq.add(new Node(v, newDist));
                    }
                }
            }
        }

        System.out.println("Vertex\tDistance from Source");
        for (int i = 0; i < n; i++) {
            System.out.println(i + "\t" + distances[i]);
        }
    }

    public static void main(String[] args) {
        // 5-node graph example
        int[][] graph = {
            {0, 10, 0, 30, 100},
            {10, 0, 50, 0, 0},
            {0, 50, 0, 20, 10},
            {30, 0, 20, 0, 60},
            {100, 0, 10, 60, 0}
        };

        int source = 0;
        dijkstra(graph, source);
    }
}
```

OUTPUT

```
Output

Vertex  Distance from Source
0   0
1   10
2   50
3   30
4   60

=== Code Execution Successful ===
```

Job Scheduling

```
import java.util.*;
class Job {
    String id;
    int deadline;
    int profit;
    public Job(String id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
}
public class Main { // Changed class name to 'Main'
    public static void jobScheduling(Job[] jobs) {
        // Sort jobs in decreasing order of profit
        Arrays.sort(jobs, (job1, job2) -> job2.profit - job1.profit);

        int maxDeadline = 0;
        for (Job job : jobs) {
            maxDeadline = Math.max(maxDeadline, job.deadline);
        }
        // To track free slots
        int[] maxDeadlines = new int[maxDeadline + 1];
        Arrays.fill(maxDeadlines, -1); // -1 means slot is empty

        int totalProfit = 0;
        System.out.println("Selected Jobs:");

        // Iterate through jobs, scheduling them as per the available slot
        for (Job job : jobs) {
            for (int i = job.deadline; i >= 1; i--) {
                if (maxDeadlines[i] == -1) {
                    maxDeadlines[i] = job.profit;
                    totalProfit += job.profit;
                    System.out.println("Job: " + job.id + ", Deadline: " + i + ", Profit: " + job.profit);
                    break;
```

```
            }
        }
    }

    System.out.println("Total Profit: " + totalProfit);
  }

  public static void main(String[] args) {
    Job[] jobs = {
      new Job("J1", 4, 70),
      new Job("J2", 1, 14),
      new Job("J3", 1, 26),
      new Job("J4", 2, 25),
      new Job("J5", 3, 13)
    };

    jobScheduling(jobs);
  }
}
```

OUTPUT

```
Selected Jobs:
Job: J1, Deadline: 4, Profit: 70
Job: J3, Deadline: 1, Profit: 26
Job: J4, Deadline: 2, Profit: 25
Job: J5, Deadline: 3, Profit: 13
Total Profit: 134


...Program finished with exit code 0
Press ENTER to exit console.
```

N-Queen Problem

```
#include <iostream>
#include <vector>
using namespace std;
// Function to check if placing a queen at (row, col) is safe
bool isSafe(int board[][10], int row, int col, int n) {
    // Check the column for another queen
    for (int i = 0; i < row; i++) {
        if (board[i][col] == 1) {
            return false;
        }
    }
    // Check upper-left diagonal
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j] == 1) {
            return false;
        }
    }
    // Check upper-right diagonal
    for (int i = row, j = col; i >= 0 && j < n; i--, j++) {
        if (board[i][j] == 1) {
            return false;
        }
    }
    return true;
}
// Backtracking function to solve N-Queens
bool backtrack(int board[][10], int row, int n) {
    // If all queens are placed, return true
    if (row == n) {
        return true;
    }
    // Try placing a queen in every column of the current row
    for (int col = 0; col < n; col++) {
        if (isSafe(board, row, col, n)) {
            board[row][col] = 1; // Place queen
```

```
            if (backtrack(board, row + 1, n)) { // Move to next row
                return true;
            }
            board[row][col] = 0; // Backtrack
        }
    }
    return false;
}
int main() {
    int n;
    cout << "Enter the size of the chessboard: ";
    cin >> n;

    // Create an n x n board initialized to 0
    int board[10][10] = {0};

    // Call backtracking function to solve the problem
    if (backtrack(board, 0, n)) {
        // Print the solution
        cout << "Solution found:" << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (board[i][j] == 0) {
                    cout << "_  "; // Empty space
                } else {
                    cout << "Q  "; // Queen
                }
            }
            cout << endl;
        }
    } else {
        cout << "No solution found." << endl;
    }

    return 0;
}
```

OUTPUT

```
C:\Users\Sanket\OneDrive\Dc

Enter the size of the chessboard: 7
Solution found:
Q  _  _  _  _  _  _
_  _  Q  _  _  _  _
_  _  _  _  Q  _  _
_  _  _  _  _  _  Q
_  Q  _  _  _  _  _
_  _  _  Q  _  _  _
_  _  _  _  _  Q  _

--------------------------------
Process exited after 9.419 seconds with return value 0
Press any key to continue . . .
```

Chatbot

```
def restaurant_chatbot():
    menu = {
        "pizza": 14.99,
        "burger": 11.99,
        "salad": 8.50,
        "pasta": 13.50,
        "soda": 5.00,
    }

    print("Welcome to our restaurant chatbot! How can I help you today?")

    while True:
        user_input = input("> ").lower()

        if "hello" in user_input or "hi" in user_input or "hey" in user_input:
            print("Hello! How can I assist you with your order?")
        elif "menu" in user_input:
            print("Our menu:")
            for item, price in menu.items():
                print(f"{item}: ${price:.2f}")
        elif "order" in user_input:
            print("What would you like to order?")
            order_item = input("> ").lower()

            if order_item in menu:
                print(f"You've ordered a {order_item}. That will be ${menu[order_item]:.2f}.")
                continue_order = input("Would you like to order anything else? (yes/no): ").lower()
                if "no" in continue_order:
                    print("Thank you for your order!")
                elif "yes" in continue_order:
                    continue
                else:
                    print("I'm sorry, I don't understand.")
            else:
                print("Sorry, we don't have that item on our menu.")
```

```
        elif "reservation" in user_input or "book" in user_input:
            print("Please provide your name, date, and time for the reservation.")
            name = input("Name: ")
            date = input("Date (YYYY-MM-DD): ")
            time = input("Time (HH:MM): ")
            print(f"Reservation for {name} on {date} at {time} confirmed.")
        elif "contact" in user_input or "phone" in user_input:
            print("You can contact us at 555-123-4567.")
        elif "address" in user_input:
            print("Our address is 123 Main Street.")
        elif "thank" in user_input or "appreciate" in user_input:
            print("You're welcome! Enjoy your meal.")
        elif "bye" in user_input or "goodbye" in user_input or "exit" in user_input:
            print("Goodbye! Have a great day.")
            break
        else:
            print("I'm sorry, I don't understand. Could you please rephrase your question?")

if __name__ == "__main__":
    restaurant_chatbot()
```

OUTPUT

```
PS C:\Users\Sanket\OneDrive\Desktop\internship project>  & 'c:\Users\Sanket\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Sanket\.vscode\extensio
ns\ms-python.debugpy-2025.4.1-win32-x64\bundled\libs\debugpy\launcher' '57030' '--' 'C:\Users\Sanket\OneDrive\Desktop\internship project\mens_eccomerce\templates
\store\chatbot.py'
Welcome to our restaurant chatbot! How can I help you today?
> HI
Hello! How can I assist you with your order?
> menu
Our menu:
pizza: $14.99
burger: $11.99
salad: $8.50
pasta: $13.50
soda: $5.00
> order
What would you like to order?
> pizza
You've ordered a pizza. That will be $14.99.
Would you like to order anything else? (yes/no): yes
> book
Please provide your name, date, and time for the reservation.
Name: Sanket
Date (YYYY-MM-DD): 2025-04-24
Time (HH:MM): 10:30
Reservation for Sanket on 2025-04-24 at 10:30 confirmed.
> contact
You can contact us at 555-123-4567.
> address
Our address is 123 Main Street.
> thank
You're welcome! Enjoy your meal.
> bye
Goodbye! Have a great day.
PS C:\Users\Sanket\OneDrive\Desktop\internship project>
```

Help Desk

```
import uuid
from datetime import datetime
from enum import Enum

class Status(Enum):
    OPEN = "Open"
    ASSIGNED = "Assigned"
    IN_PROGRESS = "In Progress"
    RESOLVED = "Resolved"
    CLOSED = "Closed"

class Priority(Enum):
    LOW = "Low"
    MEDIUM = "Medium"
    HIGH = "High"
    URGENT = "Urgent"

class Role(Enum):
    USER = "User"
    AGENT = "Agent"
    ADMIN = "Admin"

class Ticket:
    def __init__(self, subject, description, submitted_by, category, priority):
        self.ticket_id = str(uuid.uuid4())
        self.subject = subject
        self.description = description
        self.creation_date = datetime.now()
        self.status = Status.OPEN
        self.priority = priority
        self.submitted_by = submitted_by
        self.assigned_to = None
        self.resolution_details = None
        self.resolution_date = None
        self.category = category
```

```
    def __str__(self):
        assigned_to_name = self.assigned_to.username if self.assigned_to else "Not Assigned"
        return (f"ID: {self.ticket_id}, Subject: {self.subject}, Status: {self.status.value}, "
                f"Priority: {self.priority.value}, Submitted by: {self.submitted_by.username}, "
                f"Assigned to: {assigned_to_name}")

class User:
    def __init__(self, username, password, email, role, department, contact_number):
        self.user_id = str(uuid.uuid4())
        self.username = username
        self.password = self._hash_password(password) # In a real application, use a proper hashing library
        self.email = email
        self.role = role
        self.department = department
        self.contact_number = contact_number

    def _hash_password(self, password):
        # In a real application, use a secure hashing library like bcrypt or hashlib with salt
        return password

    def check_password(self, password):
        # In a real application, compare against the hashed password
        return self._hash_password(password) == password

    def __str__(self):
        return f"ID: {self.user_id}, Username: {self.username}, Role: {self.role.value}"

class Department:
    def __init__(self, name):
        self.department_id = str(uuid.uuid4())
        self.name = name

    def __str__(self):
        return f"ID: {self.department_id}, Name: {self.name}"
```

```
class Category:
    def __init__(self, name):
        self.category_id = str(uuid.uuid4())
        self.name = name

    def __str__(self):
        return f"ID: {self.category_id}, Name: {self.name}"

class Comment:
    def __init__(self, ticket, user, comment_text):
        self.comment_id = str(uuid.uuid4())
        self.ticket = ticket
        self.user = user
        self.comment_text = comment_text
        self.creation_date = datetime.now()

    def __str__(self):
        return f"ID: {self.comment_id}, Ticket ID: {self.ticket.ticket_id}, User: {self.user.username}, Comment:
{self.comment_text}"

class TicketRepository:
    def __init__(self):
        self.tickets = {}

    def create_ticket(self, ticket):
        self.tickets[ticket.ticket_id] = ticket

    def get_ticket_by_id(self, ticket_id):
        return self.tickets.get(ticket_id)

    def get_all_tickets(self):
        return list(self.tickets.values())

    def update_ticket(self, ticket):
        if ticket.ticket_id in self.tickets:
            self.tickets[ticket.ticket_id] = ticket
```

```
    def delete_ticket(self, ticket_id):
        if ticket_id in self.tickets:
            del self.tickets[ticket_id]

    def find_tickets_by_status(self, status):
        return [ticket for ticket in self.tickets.values() if ticket.status == status]

    def find_tickets_by_user(self, user):
        return [ticket for ticket in self.tickets.values() if ticket.submitted_by == user]

    def assign_ticket(self, ticket_id, agent):
        ticket = self.get_ticket_by_id(ticket_id)
        if ticket:
            ticket.assigned_to = agent

class UserRepository:
    def __init__(self):
        self.users = {}

    def create_user(self, user):
        self.users[user.user_id] = user

    def get_user_by_id(self, user_id):
        return self.users.get(user_id)

    def get_user_by_username(self, username):
        for user in self.users.values():
            if user.username == username:
                return user
        return None

    def get_all_users(self):
        return list(self.users.values())

    def update_user(self, user):
```

```
        if user.user_id in self.users:
            self.users[user.user_id] = user

    def delete_user(self, user_id):
        if user_id in self.users:
            del self.users[user_id]

class DepartmentRepository:
    def __init__(self):
        self.departments = {}

    def create_department(self, department):
        self.departments[department.department_id] = department

    def get_department_by_id(self, department_id):
        return self.departments.get(department_id)

    def get_all_departments(self):
        return list(self.departments.values())

    def update_department(self, department):
        if department.department_id in self.departments:
            self.departments[department.department_id] = department

    def delete_department(self, department_id):
        if department_id in self.departments:
            del self.departments[department_id]

class CategoryRepository:
    def __init__(self):
        self.categories = {}

    def create_category(self, category):
        self.categories[category.category_id] = category

    def get_category_by_id(self, category_id):
```

```
        return self.categories.get(category_id)

    def get_all_categories(self):
        return list(self.categories.values())

    def update_category(self, category):
        if category.category_id in self.categories:
            self.categories[category.category_id] = category

    def delete_category(self, category_id):
        if category_id in self.categories:
            del self.categories[category_id]

class CommentRepository:
    def __init__(self):
        self.comments = {}

    def add_comment(self, comment):
        self.comments[comment.comment_id] = comment

    def get_comments_by_ticket_id(self, ticket_id):
        return [comment for comment in self.comments.values() if comment.ticket.ticket_id == ticket_id]

class TicketService:
    def __init__(self, ticket_repository, user_repository):
        self.ticket_repository = ticket_repository
        self.user_repository = user_repository

    def submit_new_ticket(self, subject, description, user_id, category, priority):
        submitted_by = self.user_repository.get_user_by_id(user_id)
        if submitted_by:
            new_ticket = Ticket(subject, description, submitted_by, category, priority)
            self.ticket_repository.create_ticket(new_ticket)
            print(f"Ticket submitted successfully with ID: {new_ticket.ticket_id}")
            return new_ticket
        else:
```

```
        print("Error: User not found.")
        return None

    def view_ticket_details(self, ticket_id):
        return self.ticket_repository.get_ticket_by_id(ticket_id)

    def assign_ticket_to_agent(self, ticket_id, agent_id):
        ticket = self.ticket_repository.get_ticket_by_id(ticket_id)
        agent = self.user_repository.get_user_by_id(agent_id)
        if ticket and agent and agent.role == Role.AGENT:
            ticket.assigned_to = agent
            self.ticket_repository.update_ticket(ticket)
            print(f"Ticket {ticket_id} assigned to agent {agent.username}")
        elif not ticket:
            print(f"Error: Ticket with ID {ticket_id} not found.")
        elif not agent:
            print(f"Error: Agent with ID {agent_id} not found.")
        else:
            print("Error: User is not an agent.")

    def update_ticket_status(self, ticket_id, new_status, resolution_details=None):
        ticket = self.ticket_repository.get_ticket_by_id(ticket_id)
        if ticket and isinstance(new_status, Status):
            ticket.status = new_status
            if resolution_details:
                ticket.resolution_details = resolution_details
                ticket.resolution_date = datetime.now()
            self.ticket_repository.update_ticket(ticket)
            print(f"Ticket {ticket_id} status updated to {new_status.value}")
        elif not ticket:
            print(f"Error: Ticket with ID {ticket_id} not found.")
        else:
            print("Error: Invalid status.")

    def get_tickets_by_status(self, status):
        return self.ticket_repository.find_tickets_by_status(status)
```

```
    def get_tickets_for_user(self, user_id):
        user = self.user_repository.get_user_by_id(user_id)
        if user:
            return self.ticket_repository.find_tickets_by_user(user)
        else:
            print("Error: User not found.")
            return []

class UserService:
    def __init__(self, user_repository):
        self.user_repository = user_repository

    def register_new_user(self, username, password, email, role, department, contact_number):
        if self.user_repository.get_user_by_username(username):
            print("Error: Username already exists.")
            return None
        new_user = User(username, password, email, role, department, contact_number)
        self.user_repository.create_user(new_user)
        print(f"User {username} registered successfully with ID: {new_user.user_id}")
        return new_user

    def login_user(self, username, password):
        user = self.user_repository.get_user_by_username(username)
        if user and user.check_password(password):
            print(f"User {username} logged in successfully.")
            return user
        else:
            print("Error: Invalid username or password.")
            return None

    def get_user_details(self, user_id):
        return self.user_repository.get_user_by_id(user_id)

    def update_user_profile(self, user):
        self.user_repository.update_user(user)
```

```
        print(f"User {user.username} profile updated.")

class DepartmentService:
    def __init__(self, department_repository):
        self.department_repository = department_repository

    def create_department(self, name):
        new_department = Department(name)
        self.department_repository.create_department(new_department)
        print(f"Department '{name}' created with ID: {new_department.department_id}")
        return new_department

    def get_department_by_id(self, department_id):
        return self.department_repository.get_department_by_id(department_id)

    def get_all_departments(self):
        return self.department_repository.get_all_departments()

class CategoryService:
    def __init__(self, category_repository):
        self.category_repository = category_repository

    def create_category(self, name):
        new_category = Category(name)
        self.category_repository.create_category(new_category)
        print(f"Category '{name}' created with ID: {new_category.category_id}")
        return new_category

    def get_category_by_id(self, category_id):
        return self.category_repository.get_category_by_id(category_id)

    def get_all_categories(self):
        return self.category_repository.get_all_categories()

class CommentService:
    def __init__(self, comment_repository, ticket_repository, user_repository):
```

```
        self.comment_repository = comment_repository
        self.ticket_repository = ticket_repository
        self.user_repository = user_repository

    def add_comment_to_ticket(self, ticket_id, user_id, comment_text):
        ticket = self.ticket_repository.get_ticket_by_id(ticket_id)
        user = self.user_repository.get_user_by_id(user_id)
        if ticket and user:
            new_comment = Comment(ticket, user, comment_text)
            self.comment_repository.add_comment(new_comment)
            print(f"Comment added to ticket {ticket_id} by user {user.username}")
            return new_comment
        elif not ticket:
            print(f"Error: Ticket with ID {ticket_id} not found.")
        else:
            print(f"Error: User with ID {user_id} not found.")
            return None

    def get_comments_by_ticket_id(self, ticket_id):
        return self.comment_repository.get_comments_by_ticket_id(ticket_id)

class HelpDeskCLI:
    def __init__(self, ticket_service, user_service):
        self.ticket_service = ticket_service
        self.user_service = user_service
        self.logged_in_user = None

    def run(self):
        print("Welcome to the Help Desk System!")
        while True:
            if not self.logged_in_user:
                print("\n1. Register\n2. Login\n3. Exit")
                choice = input("Enter your choice: ")
                if choice == '1':
                    self.register_user()
                elif choice == '2':
```

```
            self.login()
        elif choice == '3':
            print("Exiting...")
            break
        else:
            print("Invalid choice. Please try again.")
    else:
        print(f"\nWelcome, {self.logged_in_user.username} ({self.logged_in_user.role.value})")
        if self.logged_in_user.role == Role.USER:
            print("1. Submit New Ticket\n2. View My Tickets\n3. Logout")
            choice = input("Enter your choice: ")
            if choice == '1':
                self.submit_ticket()
            elif choice == '2':
                self.view_my_tickets()
            elif choice == '3':
                self.logout()
            else:
                print("Invalid choice. Please try again.")
        elif self.logged_in_user.role in [Role.AGENT, Role.ADMIN]:
            print("1. View All Tickets\n2. Assign Ticket\n3. Update Ticket Status\n4. Logout")
            choice = input("Enter your choice: ")
            if choice == '1':
                self.view_all_tickets()
            elif choice == '2':
                self.assign_ticket()
            elif choice == '3':
                self.update_ticket_status()
            elif choice == '4':
                self.logout()
            else:
                print("Invalid choice. Please try again.")

def register_user(self):
    username = input("Enter username: ")
    password = input("Enter password: ")
```

```
    email = input("Enter email: ")
    role_str = input("Enter role (USER, AGENT, ADMIN): ").upper()
    try:
        role = Role[role_str]
    except KeyError:
        print("Invalid role.")
        return
    department = input("Enter department: ")
    contact_number = input("Enter contact number: ")
    self.user_service.register_new_user(username, password, email, role, department, contact_number)

def login(self):
    username = input("Enter username: ")
    password = input("Enter password: ")
    user = self.user_service.login_user(username, password)
    if user:
        self.logged_in_user = user

def logout(self):
    self.logged_in_user = None
    print("Logged out successfully.")

def submit_ticket(self):
    subject = input("Enter subject: ")
    description = input("Enter description: ")
    category_name = input("Enter category: ")
    priority_str = input("Enter priority (LOW, MEDIUM, HIGH, URGENT): ").upper()
    try:
        priority = Priority[priority_str]
    except KeyError:
        print("Invalid priority.")
        return

    # In a real application, you might fetch or create the category object
    # For this example, we'll just pass the name
    self.ticket_service.submit_new_ticket(subject, description, self.logged_in_user.user_id, category_name, priority)
```

```
def view_my_tickets(self):
    if self.logged_in_user:
        tickets = self.ticket_service.get_tickets_for_user(self.logged_in_user.user_id)
        if tickets:
            print("\nYour Tickets:")
            for ticket in tickets:
                print(ticket)
        else:
            print("You have no open tickets.")
    else:
        print("You are not logged in.")

def view_all_tickets(self):
    if self.logged_in_user and self.logged_in_user.role in [Role.AGENT, Role.ADMIN]:
        tickets = self.ticket_service.ticket_repository.get_all_tickets()
        if tickets:
            print("\nAll Tickets:")
            for ticket in tickets:
                print(ticket)
        else:
            print("No tickets available.")
    else:
        print("You do not have permission to view all tickets.")

def assign_ticket(self):
    if self.logged_in_user and self.logged_in_user.role in [Role.AGENT, Role.ADMIN]:
        ticket_id = input("Enter the ID of the ticket to assign: ")
        agent_username = input("Enter the username of the agent to assign to: ")
        agent = self.user_service.user_repository.get_user_by_username(agent_username)
        if agent and agent.role == Role.AGENT:
            self.ticket_service.assign_ticket_to_agent(ticket_id, agent.user_id)
        elif not agent:
            print(f"Agent with username '{agent_username}' not found.")
        else:
            print(f"{agent_username} is not an agent.")
```

```
        else:
            print("You do not have permission to assign tickets.")

    def update_ticket_status(self):
        if self.logged_in_user and self.logged_in_user.role in [Role.AGENT, Role.ADMIN]:
            ticket_id = input("Enter the ID of the ticket to update: ")
            status_str = input("Enter the new status (OPEN, ASSIGNED, IN_PROGRESS, RESOLVED, CLOSED): ").upper()
            try:
                new_status = Status[status_str]
                resolution_details = None
                if new_status == Status.RESOLVED or new_status == Status.CLOSED:
                    resolution_details = input("Enter resolution details (if any): ")
                self.ticket_service.update_ticket_status(ticket_id, new_status, resolution_details)
            except KeyError:
                print("Invalid status.")
        else:
            print("You do not have permission to update ticket statuses.")

if __name__ == "__main__":
    ticket_repo = TicketRepository()
    user_repo = UserRepository()
    dept_repo = DepartmentRepository()
    cat_repo = CategoryRepository()
    comment_repo = CommentRepository()

    user_service = UserService(user_repo)
    ticket_service = TicketService(ticket_repo, user_repo)
    dept_service = DepartmentService(dept_repo)
    cat_service = CategoryService(cat_repo)
    comment_service = CommentService(comment_repo, ticket_repo, user_repo)

    cli = HelpDeskCLI(ticket_service, user_service)
    cli.run()
```

OUTPUT

```
PS C:\Users\Sanket\OneDrive\Desktop\internship project>
 & 'c:\Users\Sanket\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\Sanket\.vscode\extensions\ms-python.debugpy-2025.4.1-win32-x64\bundled\libs\debugpy\launcher' '5
7337' '--' 'C:\Users\Sanket\OneDrive\Desktop\helpdesk.py'
Welcome to the Help Desk System!

1. Register
2. Login
3. Exit
Enter your choice: 1
Enter username: Sanket
Enter password: asdf
Enter email: asas@gmail.com
Enter role (USER, AGENT, ADMIN): user
Enter department: computer
Enter contact number: 87688778
User Sanket registered successfully with ID: 22ce558b-a416-4eaf-ba5a-e33c8cca437d

1. Register
2. Login
3. Exit
Enter your choice: 2
Enter username: Sanket
Enter password: asdf
User Sanket logged in successfully.

Welcome, Sanket (User)
1. Submit New Ticket
2. View My Tickets
3. Logout
Enter your choice: 1
Enter subject: app not working
Enter description: the current version app is not working
Enter category: software
Enter priority (LOW, MEDIUM, HIGH, URGENT): medium
Ticket submitted successfully with ID: d2665ba8-d057-4215-bb15-8d6c4e612ed8

Welcome, Sanket (User)
1. Submit New Ticket
2. View My Tickets
3. Logout
Enter your choice: 2
```