MEDICONNECT

BY HOTBYTES:

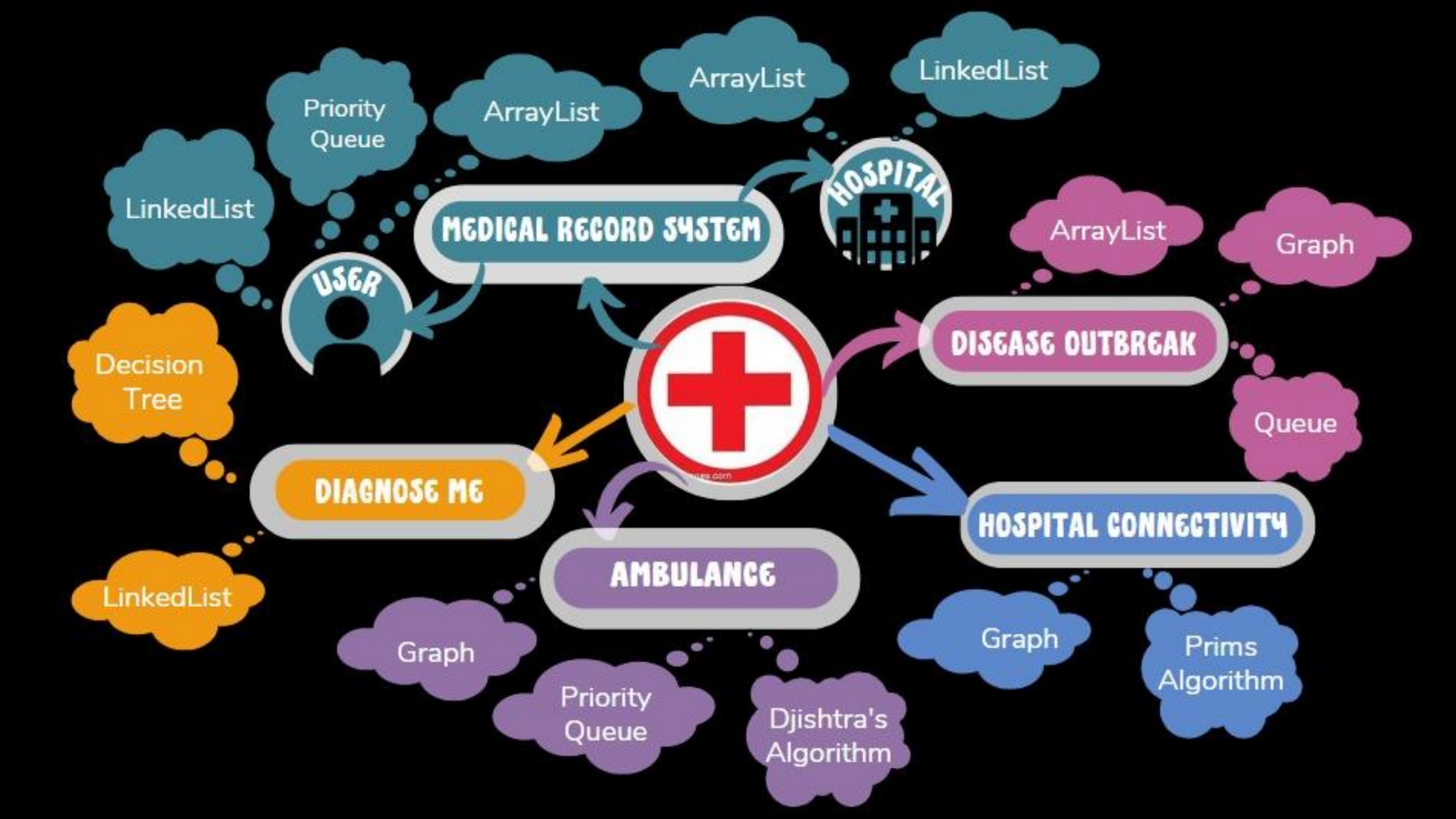
SY IT 2022-23

UIT2021815 - PRAGATI CHANDRA

UIT2021822 - KRITIKA DUBEY

UIT2021833 - GAURI JOSHI

UEC2021139 - AKANKSHA KALE


Priority Queue
ArrayList
ArrayList
LinkedList
LinkedList
MEDICAL RECORD SYSTEM
HOSPITAL
USER
ArrayList
Graph
Decision Tree
DISEASE OUTBREAK
Queue
DIAGNOSE ME
HOSPITAL CONNECTIVITY
AMBULANCE
LinkedList
Graph
Graph
Prims Algorithm
Priority Queue
Djishtra's Algorithm

ABOUT THE PROJECT:

- Features of the Patient's Portal:

✓ Secure Login: Patients can log in using their username and password to access their medical records and other functionalities.

✓ Medical Record Management: Patients can add or edit their name, diagnosis, diagnosis date, and reports. They can also view all their medical details and track their health trends.

✓ Appointment Booking: Patients can book appointments with doctors of their choice through the portal.

✓ Secure Logout: Patients can log out of their account to ensure the security of their medical information.

- Features of the Hospital's Portal:

✓ Secure Login: Hospital staff can log in using their username and password to access the patient's medical records and other functionalities.

✓ Medical Record Management: Staff can add new medical records, search for existing medical records, view all medical records, and add reports to a patient's medical record.

✓ Doctor Appointment Management: Staff can view the appointments of doctors for the day.

In addition to these features, we plan to implement several advanced functionalities, including:

- Diagnosis of Illness through Symptoms: Algorithm can diagnose illnesses based on patient symptoms.
- Shortest Route Finding for Ambulance: The system can find the shortest route for an ambulance to reach a patient in need.
- Disease Spreading Tracking: The system can track the spread of diseases and provide insights to control them.
- Tracking Movement of Spreader-Patients: The system can track the movement of spreader-patients to identify and isolate them.
- Hospital Connectivity in case of scarcity of resources.

PATIENT SIDE:

- Allows the patient to log in using their username and password.
- Provides the patient with several options: add/edit their name, add/edit their diagnosis, add/edit their diagnosis date, add a report, view all their details, check their health trends, and book an appointment.
- Allows the patient to book an appointment with a doctor of their choice.
- Allows the patient to log out.

PATIENT SIDE:

✓ Functions:

- Patient: constructor to initialize the attributes of the patient object, i.e., name, password, patient ID, diagnosis, and date of diagnosis.
- getPatientName(), getPatientId(), getDiagnosis(), and getDateOfDiagnosis(): getter functions to return the corresponding patient attributes.
- setPatientName(), setPatientId(), setDiagnosis(), and setDateOfDiagnosis(): setter functions to set the corresponding patient attributes.
- bookappointment(): a function to book an appointment with a doctor. A slot is created for the patient, and this slot is added to the doctor's list of appointments.
- displayDetails(): a function to display the patient's details, including patient ID, name, diagnosis, and date of diagnosis. It also displays all the reports associated with the patient by calling viewAllReports() function.
- viewAllReports(): a function that returns a string of all the patient's reports.
- addReport(): a function to add a new report for the patient. It takes input from the user for details such as weight, height, blood pressure, haemoglobin, cholesterol, and sugar. Then, a new report object is created and added to the list of reports.
- addReport(Report r): a function that adds an existing report to the list of reports.
- checkHealthTrends(): a function that checks the increase or decrease in the patient's health factors by comparing the factors in the current report with the previous report. The function calculates the difference in blood pressure, haemoglobin, sugar, weight, and cholesterol values and stores the result in variables.

PATIENT SIDE:

✓ *Data Structures:*

ArrayList<Report>: A list to store all the reports associated with a patient.

LinkedList: To store all reports in a hospital.

APPOINTMENT SCHEDULING:

- Functions:
- Doctor: Constructor function to initialize a new Doctor object with the given name.
- viewappointment: Function to view appointments of a doctor for the day. It returns a string containing the patient's name for each appointment.
- dequeAppointment: Function to remove the first appointment from the queue of a doctor.
- Data structures used:
- Priority Queue: A linked list based data structure used to store the appointments of a doctor.
- StringBuilder: A class used to create mutable strings.
- Algorithm:
- The Doctor class is defined with a constructor that initializes a new Doctor object with the given name and an empty queue to store the appointments.
- The viewappointment function iterates through the appointments queue of the doctor and appends the patient's name for each appointment to a StringBuilder.
- The dequeAppointment function removes the first appointment from the queue of a doctor.

HOSPITAL SIDE:

- Allows the hospital staff to log in using their username and password.
- Provides the staff with several options: add a new medical record, search for a medical record, view all medical records, add a report to a patient, and see the appointments of a doctor for the day.
- Allows the staff to add a new medical record for a patient.
- Allows the staff to search for a medical record based on the patient's password.
- Allows the staff to view all the medical records stored in the system.
- Allows the staff to add a report to a patient's medical record.
- Allows the staff (doctors) to see their appointments for the day.

HOSPITAL SIDE:

✓ Functions:

- addMedicalRecord() - This function prompts the user to enter the patient's details such as name, ID, diagnosis, password, and date of diagnosis. It creates a new Patient object with these details and adds it to the ArrayList medicalRecords.
- searchMedicalRecord() - This function prompts the user to enter the patient ID and then searches for a Patient object in the medicalRecords ArrayList with the same ID. If found, it displays the patient's details.
- viewAllMedicalRecords() - This function returns a StringBuilder object with details of all the patients in medicalRecords, including their name and details of any reports associated with them.
- addReportToPatient() - This function prompts the user to enter the patient ID and adds a new report to the corresponding Patient object in medicalRecords.

HOSPITAL SIDE:

✓ Data Structures Used:

- ArrayList<Patient> medicalRecords - This ArrayList is used to store all the Patient objects and their details.
- Scanner - Scanner is used to read input from the user.

FIND SHORTEST ROUTE FOR AMBULANCE:

✓ Functions:

- AmbulanceGUI(int numNodes) : Constructor that initializes shortestdistance[], visited[] and adjecent[] for each node.
- void addEdge(int u, int v, int weight) : Adds an edge between node u and v with given weight in the adjacency list adjecent[].
- int shortestPath(int source, int destination) : Implements Dijkstra's algorithm to find the shortest path between source and destination. Returns the shortest distance between source and destination.
- void initialize() : Initializes the GUI components.
- AmbulanceGUI() : Constructor that calls initialize().
- main() : The entry point of the program that initializes the GUI.

✓ *Data Structures:*

- *int[] shortestdistance: Stores the shortest distance from the source node to all other nodes.*
- *boolean[] visited: Keeps track of the nodes that have been visited while running Dijkstra's algorithm.*
- *List<Edge>[] adjecent: A list of adjacency lists where each adjacency list represents the edges going out from a node.*
- *PriorityQueue<Node> pq: A priority queue that stores nodes to be visited while running Dijkstra's algorithm.*

✓ Algorithm:

- Dijkstra's Algorithm: An algorithm to find the shortest path between a source node and all other nodes in a weighted graph. It works by maintaining a priority queue of nodes to be visited next, and updating the shortest distance of a node if a shorter path is found. The algorithm terminates when the destination node is reached or all reachable nodes have been visited.

MEDICAL DIAGNOSIS

✓ Functions:

- Node(): A constructor that initializes a new Node with the given symptom and sets its yes and no nodes to null.
- diagnose(): A function that prompts the user to input the type of symptoms they are experiencing and calls the appropriate function based on the user's choice.
- eyes(): A function that prompts the user to input their symptoms related to the eyes and based on the input, navigates through a binary tree structure to provide a diagnosis.
- ent(): A function that prompts the user to input their symptoms related to the ear, nose, and throat and based on the input, navigates through a binary tree structure to provide a diagnosis.

✓ *Data structures:*

- *Tree*
- *Node: A class that represents a node in a binary tree structure. Each node contains a symptom string and two child nodes, yes and no.*
- *LinkedList: A data structure used in the eyes() and ent() functions to store the user's symptoms as they are inputted.*

✓ Algorithms:

✓ Binary tree traversal: The eyes() and ent() functions both traverse a binary tree structure to diagnose the user's symptoms. The traversal is done by starting at the root node and moving left or right depending on the user's symptoms until a leaf node is reached, which contains the diagnosis for the user's symptoms.

PATIENT MOVEMENT TRACKING FOR DISEASE SPREAD DETECTION: The system can track the movement of spreader-patients to identify and isolate them.

✓ Functions:

- DiseaseSpread(): constructor function that initializes the GUI and its components.
- main(): main method that creates an instance of DiseaseSpread.
- actionPerformed(ActionEvent e): handles the user's interactions with the GUI components.
- getNode(String s): returns the Node object corresponding to a string that represents a locality.
- BFS(): performs a Breadth First Search algorithm on the graph of localities and outputs the order in which localities are visited.
- diseaseoutbreak(Node source, Node destination): returns the number of patients traveling from one locality to another locality using a Breadth First Search algorithm.
- hotspot(): returns the locality that has the most neighbors.
- resetNodes(): resets the "visited" attribute of all nodes.

✓ *Data Structures:*

- *Graph*
- *ArrayList: a dynamic array that can grow or shrink in size.*

✓ Algorithm:

- Breadth First Search algorithm: a graph traversal algorithm that visits all the vertices of a graph in breadth-first order. It uses a queue to keep track of the vertices that need to be visited. The algorithm starts at a source vertex, visits all its neighbors, then visits the neighbors of its neighbors, and so on until all vertices have been visited.

- DISEASE OUTBREAK: : The system can track the spread of diseases and provide insights to control them.

✓ Functions:

- main(): This function is the entry point of the application. It initializes the graph and creates the GUI layout using JavaFX.
- addEdge(): This function is used to add edges to the graph.
- BFS(): This function performs Breadth First Search on the graph to find the shortest path from the source node to all other nodes.
- diseaseoutbreak(): This function is used to simulate a disease outbreak in a locality and return the number of patients travelling from that locality.

✓ *Data structures:*

- *ArrayList<ArrayList<Pair1<Node, Integer>>> adj: This is an adjacency list representation of the graph. It is used to store the edges between the nodes.*
- *Graph*
- *Queue*

✓ Algorithm:

- BFS(): This function uses the Breadth First Search algorithm to traverse the graph and find the shortest path from the source node to all other nodes.
- diseaseoutbreak(): This function simulates the spread of a disease in a locality by using a BFS traversal of the graph. It returns the number of patients travelling from the locality.

LOGIN

✓ Functions:

- main(): The main function sets up a GUI frame, panel, labels, text fields, and button. It also adds an action listener to the login button, which checks the entered login and password and shows a success message or an error message accordingly.

✓ *Data structures:*

- *Hash-Maps: To map Patient object with Password.*

HOSPITAL CONNECTIVITY – for sharing of medical resources

✓ Functions and What They Do:

- (int key[], Boolean mstSet[]): returns the vertex with the minimum key value that has not yet been included in the MST set.
- `primMST(int graph[][]): constructs the minimum spanning tree (MST) for a graph represented by an adjacency matrix using Prim's algorithm.
- main(String[] args): creates a graph with distances between hospitals, finds the minimum spanning tree using Prim's algorithm, creates a JTable with the results, and displays it in a GUI.

✓ *Data Structures Used:*

- *int[] parent: an array to store the constructed MST.*
- *int[] key: key values used to pick minimum weight edge in a cut.*
- *Boolean[] mstSet: set of vertices not yet included in the MST.*
- *int[][] graph: adjacency matrix representation of a graph.*

✓ Algorithm Used:

- Prim's algorithm: a greedy algorithm that finds the minimum spanning tree for a weighted graph. The algorithm starts with a single vertex and iteratively adds the minimum weight edges until all vertices are included in the MST set.