Email Management System Project Specification

Project Overview

Develop a comprehensive Email Management System similar to Gmail using multiple data structures including Arrays, Stacks, Queues, Heaps, Trees, BSTs, Linked Lists, and Graphs. The system should provide complete email functionality with proper folder management and spam filtering.

Specifications:

1. System Setup

a. The system maintains user accounts with unique email addresses following the format: username@lhr.nu.edu.pk or any other format like abc@gmail.com

b. User data will be stored in text files and loaded into appropriate data structures:

- "users.txt" - User account information
- "emails.txt" - Email records
- "spam_words.txt" - Spam filtering keywords
- "social_graph.txt" - User connection data

c. Each user has the following folders:

- Inbox
- Sent
- Drafts
- Spam
- Trash
- Important (Priority-based)

2. Data Structures Implementation

a. Array

- Store spam filtering words: {"Winner", "Free", "Urgent", "Claim", "Bonus", "Limited", "Exclusive", "Gift", "Guaranteed", "Profit"}
- Fixed-size array for recent contacts (last 10 contacted users)

- Array for storing system configuration settings

b. Stack

- Undo/Redo functionality for email operations
- Navigation history between folders
- Recently deleted emails recovery stack

c. Queue

- Email sending queue for scheduled emails
- Priority queue for high-importance emails
- Processing queue for incoming emails

d. Heap

- Priority heap for important emails based on:
    - Sender priority
    - Keywords in subject/content
    - User interaction history
- Max-heap for organizing emails by timestamp

e. Trees & BSTs

- Binary Search Tree for quick user lookup by username/email
- Tree structure for email threading and conversations
- BST for organizing contacts by name/email

f. Linked Lists

- Singly linked list for individual email folders
- Doubly linked list for email navigation (previous/next email)
- Circular linked list for recent activity log

g. Graphs

- Social graph representing user connections
- Graph for email routing and delivery paths

- Weighted graph for spam probability calculation

3. Core Classes Structure

a. User Class

```
class User {
private:
   string userId;
   string username;
   string email;
   string password;
   BST<Contact>* contacts;
   GraphNode* socialNode;

public:
   // Constructor, getters, setters
   bool validatePassword(string pass);
   void addContact(Contact newContact);
   vector<Contact> searchContacts(string query);
};
```

b. Email Class

```
class Email {
private:
   string emailId;
   string sender;
   string receiver;
   string subject;
```

```
    string content;
    time_t timestamp;
    bool isRead;
    bool isSpam;
    int priority;

public:
    // Constructor, getters, setters
    bool containsSpamWords(string spamWords[]);
    void markAsRead();
    void setPriority(int p);
};
```

c. Folder Management Class

```
class EmailFolder {
private:
    string folderName;
    LinkedList<Email>* emails;
    MaxHeap<Email>* priorityHeap;
    Stack<Email>* recentEmails;

public:
    void addEmail(Email newEmail);
    Email removeEmail(string emailId);
    vector<Email> getEmailsByPriority();
    Email getRecentEmail();
};
```

d. Social Graph Class

```
class SocialGraph {
private:
   map<string, GraphNode*> users;

public:
   void addConnection(string user1, string user2);
   void removeConnection(string user1, string user2);
   vector<string> getMutualConnections(string user1, string user2);
   int calculateSpamProbability(string sender, string receiver);
};
```

4. System Operations

a. User Authentication

- Account creation with validation
- Login/logout functionality
- Password recovery system

b. Email Composition & Sending

- Compose new emails with rich text support
- Save drafts with auto-save functionality
- Send immediately or schedule for later
- Attachment handling (metadata only)

c. Email Management

- Move emails between folders
- Mark as read/unread
- Star important emails
- Delete and recover emails

- Empty trash functionality

d. Advanced Features

- Smart spam filtering using multiple algorithms
- Email threading and conversation view
- Quick search using BST indexing
- Priority inbox using heap sorting
- Social connection recommendations

e. Folder Operations

- Create custom folders
- Organize emails by priority
- Bulk operations (move, delete, mark)
- Folder statistics

5. System Interface

I. Welcome Screen

II. Main Menu

1. User Authentication

   a. Create Account

   b. Login

   c. Forgot Password

   d. Exit

2. Email Dashboard (After Login)

   a. Compose Email

   - To, Subject, Content
   - Save Draft / Send / Schedule
   - Add Attachments

b. Folder Management

- Inbox (Priority Sorted)
- Sent
- Drafts
- Spam
- Trash
- Custom Folders

c. Search & Filter

- Quick Search (BST-based)
- Advanced Search
- Filter by Date, Sender, Priority

d. Contacts Management

- Add Contact
- View Connections
- Mutual Connections
- Contact Suggestions

e. Settings

- Spam Filter Settings
- Priority Rules
- Notification Preferences

3. Administrative Functions

   a. System Statistics

   b. User Management

   c. Spam Analysis

   d. Social Graph Visualization

4. Data Persistence

   a. Save All Data

   b. Load Previous Session

   c. Export Data

6. File Structures

users.txt Format:

userId,username,email,password,createdDate,lastLogin

emails.txt Format:

emailId,sender,receiver,subject,content,timestamp,isRead,isSpam,priority,folder

spam_words.txt Format:

Winner,Free,Urgent,Claim,Bonus,Limited,Exclusive,Gift,Guaranteed,Profit

social_graph.txt Format:

user1,user2,connectionStrength,connectionDate

7. Implementation Requirements

a. All data structures must be implemented from scratch
b. Proper memory management and error handling
c. Efficient algorithms for search and sorting operations
d. Modular design with separate classes for each component
e. Console-based interface with clear navigation

8. Bonus Features

- Email scheduling with queue management
- Advanced spam detection using graph analysis

- Email analytics and statistics
- Backup and restore functionality
- Multi-language support

9. Testing Requirements

- Unit testing for each data structure
- Integration testing for system workflows
- Stress testing with large datasets
- Edge case handling for all operations

Important:-

- The system should demonstrate practical use of all specified data structures
- Code should be well-documented with clear comments
- Focus on efficiency and optimal algorithms
- Implement proper object-oriented programming principles
- Ensure data persistence across sessions

Final Notes:

- There are some undecided issues in this system but it doesn't mean that the system is not complete or not clear. The undecided issues are left, on purpose, to be answered by you people as long as they don't violate the main structure of the system.
- As usual, remember to write one class at a time and test it and then add another class and another class until you complete your system.

Good Luck!

Regards
Syed Aoun Haider Sherazi
Teaching Assistant
Data Structures Course