



DATA STRUCTURE PROJECT

Binary search tree Phonebook

Prepared By:

Faisal Alhaqbani 443101660

Mohammed Alzubaidi 443101700

Mohammed Almuhaithheef 443101706

Introduction :

In this phase, we have evolved the application to utilize a Binary Search Tree (BST) for managing the phonebook. This shift aims to leverage the efficient search, insert, and delete operations of BST, which are crucial for large datasets typically found in phonebooks.

Our main Classes :


- **BST and BSTNode Classes:** The BST class, along with its nested BSTNode class, forms the backbone of our application's new data structure. The BST class manages the tree operations like insert, find, and delete, while BSTNode represents individual nodes in the tree, each holding a contact's data and keys for sorting.
- **Contact Class:** The Contact class now implements the Comparable interface, allowing contacts to be compared and sorted within the BST based on their names. This class also manages additional details like phone numbers, email addresses, and associated events.
- **Event Class:** The Event class is designed to handle event details, including title, date, location, and the list of contacts associated with an event. It also distinguishes between events and appointments.



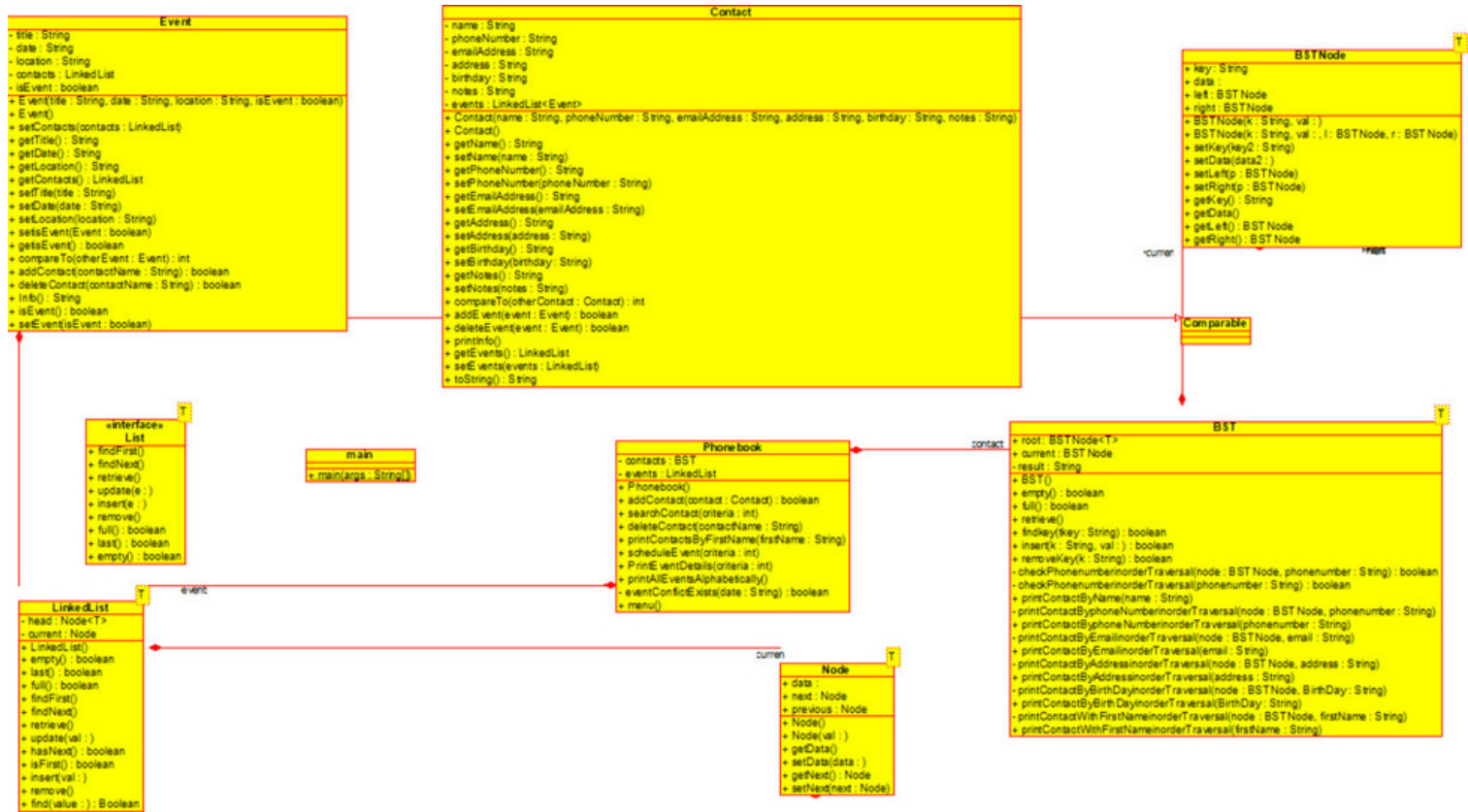
Modifications to Existing Classes

- **Phonebook Class:** This class serves as the primary interface for the application, handling user interactions and directing operations on the BST for contact management and event scheduling.
- **LinkedList Class:** While the main data structure has shifted to BST, the LinkedList class remains vital for managing lists of events within individual contacts and events themselves.

Interaction between Classes

- **The Phonebook** class uses the **BST** to perform operations like adding, searching, and deleting contacts. It also interacts with the Event class to handle event-related functionalities.
 - The **Contact** and **Event** classes are designed to work seamlessly with the **BST**, ensuring efficient data handling and retrieval.
- 

Class UML diagram



The main tasks for each person

Task	Assigned To
Implement Event Class Methods and contact class methods , calculate Big O , Write the report	Mohammed Almuhaitheef
Work on BST , BST NODE Classes and Work on some PhoneBook methods	Mohammed Alzubaidi
Lead the group and work on most Phonebook methods , and handle the exceptions	Faisal Alhaqbani

user-accessible methods

Command Option	Method Used	Description
Add a Contact	addContact	Allows users to add new contacts, prompting for details like name, phone number, email, address, birthday, and notes. Adds contacts to the BST.
Search for a Contact	searchContact	Provides search functionality based on criteria like name, phone number, email, address, or birthday. Displays matching contact information.
Delete a Contact	deleteContact	Enables deletion of a contact by name. Removes the contact from the BST and associated events.
Schedule an Event	scheduleEvent	Allows scheduling of events or appointments, collecting details such as title, date, time, location, and involved contacts. Checks for date conflicts.
Print Event Details	PrintEventDetails	Offers options to print details of events by contact name or event title. Displays comprehensive event information.
Print Contacts by First Name	printContactsByFirstName	Lists contacts whose first names match the provided input. Searches the BST and displays relevant contact details.
Print All Events Alphabetically	printAllEventsAlphabetically	Displays all scheduled events and appointments in alphabetical order, providing an overview of all events.

Project Specification:

For BST class :

Method name	Requires	Input	Output	Results
empty	No specific conditions.	None	Boolean value indicating if the BST is empty.	Returns true if the BST has no nodes, false otherwise.
findkey	The BST is not empty.	A string tkey representing the contact's key to find.	Boolean value indicating if the key is found.	If found, current is set to the node containing tkey.
insert	The BST is not full, and the contact name and phone number must be unique.	A string k representing the contact's key, and val of type T representing the contact's data.	Boolean value indicating success or failure of insertion.	Inserts the contact into the BST if a contact with the same key does not exist.
removeKey	The key to be removed exists in the BST.	A string k representing the key of the contact to be removed.	Boolean value indicating success or failure of removal.	Removes the contact associated with key k and maintains BST properties.

Project Specification:

Method	Requires	Input	Output	Results
compareTo	Another Contact object to compare with.	Contact otherContact to compare against.	An integer value: negative if this contact is less than, zero if equal, and positive if greater than the other contact.	Compares two contacts based on their name.
addEvent	The event to be added is not already associated with the contact.	Event event to be added to this contact.	Boolean value indicating success or failure of adding the event.	Associates the event with the contact if it was not already associated.
deleteEvent	The event to be deleted is associated with the contact.	Event event to be removed from this contact.	Boolean value indicating success or failure of event removal.	Disassociates the event from the contact.

Project Specification:

Method	Requires	Input	Output	Results
compareTo	Another Event object to compare with.	Event otherEvent to compare against.	An integer value: negative if this event is less than, zero if equal, and positive if greater than the other event.	Compares two events based on their title.
addContact	The contact name to be added is not already associated with the event.	A string contactName representing the name of the contact to add to the event.	Boolean value indicating success or failure of adding the contact name.	Adds the contact name to the event's contact list if not already present.
deleteContact	The contact name to be removed is associated with the event.	A string contactName representing the name of the contact to remove from the event.	Boolean value indicating success or failure of contact name removal.	Removes the contact name from the event's contact list.

Project Specification:

Method	Requires	Input	Output	Results
empty	No specific conditions.	None	Boolean value indicating if the list is empty.	Returns true if the list has no nodes, false otherwise.
insert	The list is not full.	Generic type T val representing the data to insert into the list.	None	Inserts the data into the list maintaining the order.
remove	The current node is not null.	None	None	Removes the current node from the list.

Time complexity of the methods

First LinkedList Methods

Method Name	Time Complexity Estimate
LinkedList()	O(1)
empty()	O(1)
last()	O(1)
full()	O(1)
findFirst()	O(1)
findNext()	O(1)
retrieve()	O(1)
update(T val)	O(1)
insert(T val)	O(n)
remove()	O(n)
find(T value)	O(n)

Summary code complexity

Method Name	Time Complexity Estimate
BST()	O(1)
empty()	O(1)
full()	O(1)
retrieve()	O(1)
findkey(String tkey)	O(log n) *
insert(String k, T val)	O(log n) *
removeKey(String k)	O(log n) *
checkPhonenumberinorderTraversal(BSTNode<T> node, String phonenummer)	O(n)
checkPhonenumberinorderTraversal(String phonenummer)	O(n)
printContactByName(String name)	O(log n) *
printContactByphoneNumberinorderTraversal(BSTNode<T> node, String phonenummer)	O(n)
printContactByphoneNumberinorderTraversal(String phonenummer)	O(n)
printContactByEmailinorderTraversal(BSTNode<T> node, String email)	O(n)
printContactByEmailinorderTraversal(String email)	O(n)
printContactByAddressinorderTraversal(BSTNode<T> node, String address)	O(n)
printContactByAddressinorderTraversal(String address)	O(n)
printContactByBirthDayinorderTraversal(BSTNode<T> node, String BirthDay)	O(n)
printContactByBirthDayinorderTraversal(String BirthDay)	O(n)
printContactWithFirstNameinorderTraversal(BSTNode<T> node, String firstName)	O(n)
printContactWithFirstNameinorderTraversal(String firstName)	O(n)

Summary code complexity

Method Name	Time Complexity Estimate
Contact(String name, ...)	O(1)
getName()	O(1)
setName(String name)	O(1)
getPhoneNumber()	O(1)
setPhoneNumber(String phoneNumber)	O(1)
getEmailAddress()	O(1)
setEmailAddress(String emailAddress)	O(1)
getAddress()	O(1)
setAddress(String address)	O(1)
getBirthday()	O(1)
setBirthday(String birthday)	O(1)
getNotes()	O(1)
setNotes(String notes)	O(1)
compareTo(Contact otherContact)	O(n)
addEvent(Event event)	O(n)
deleteEvent(Event event)	O(n)
printInfo()	O(1)

Summary code complexity

Method Name	Time Complexity Estimate
Event(String title, ...)	O(1)
getTitle()	O(1)
getDate()	O(1)
getLocation()	O(1)
getContacts()	O(1)
setTitle(String title)	O(1)
setDate(String date)	O(1)
setLocation(String location)	O(1)
addContact(String contactName)	O(n)
deleteContact(String contactName)	O(n)
Info()	O(n)
isEvent()	O(1)
setEvent(boolean isEvent)	O(1)
compareTo(Event otherEvent)	O(k)

O(k) represents the complexity of comparing strings, where k is the length of the strings being compared

Summary code complexity

Method Name	Time Complexity Estimate
Event(String title, ...)	O(1)
getTitle()	O(1)
getDate()	O(1)
getLocation()	O(1)
getContacts()	O(1)
setTitle(String title)	O(1)
setDate(String date)	O(1)
setLocation(String location)	O(1)
addContact(String contactName)	O(n)
deleteContact(String contactName)	O(n²)
Info()	O(n)
isEvent()	O(1)
setEvent(boolean isEvent)	O(1)
compareTo(Event otherEvent)	O(k)

O(k) represents the complexity of comparing strings, where k is the length of the strings being compared

Summary code complexity

Method Name	Time Complexity Estimate
Phonebook()	$O(1)$
addContact(Contact contact)	$O(\log n)$
searchContact(int criteria)	$O(n)$
deleteContact(String contactName)	$O(n)$
printContactsByFirstName(String firstName)	$O(n)$
scheduleEvent(int criteria)	$O(n^2)$
PrintEventDetails(int criteria)	$O(n^2)$
printAllEventsAlphabetically()	$O(n)$
eventConflictExists(String date)	$O(n)$

Time Complexity for DeleteContact()

Line	Code	Time Complexity
1	if (contacts.findkey(contactName)) {	O(n)
2	Contact temp = contacts.retrieve();	O(1)
3	if (contacts.removeKey(contactName)) {	O(n)
4	LinkedList<Event> contactEvents = temp.getEvents();	O(1)
5	if (!contactEvents.empty()) {	O(1)
6	contactEvents.findFirst();	O(1)
7	while (!contactEvents.last()) {	O(n)
8	events.find(contactEvents.retrieve());	O(n)
9	events.retrieve().deleteContact(contactName);	O(1)
10	if (events.retrieve().getContacts().empty()) {	O(n)
11	events.remove();	O(n ²)
12	contactEvents.findNext();	O(n)
13	}	O(1)
14	System.out.println(events.find(contactEvents.retrieve()));	O(n)
15	System.out.println(events.retrieve().deleteContact(contactName));	O(1)
16	if (events.retrieve().getContacts().empty()) {	O(1)
17	events.remove();	O(1)
18	}	O(1)
19	System.out.println("\nContact deleted successfully.");	O(1)
20	return;	O(1)
21	}	O(1)
-	Total Complexity	O(n ²)

Time Complixtiy for PrintEventDetails()

Line	Code	Time Complexity
1-4	Variable declarations and Scanner initialization	O(1)
5	if (events.empty()) {	O(1)
6-8	Handling empty events case	O(1)
9	events.findFirst();	O(1)
10	String result = "";	O(1)
11	int count = 0;	O(1)
12	switch (criteria) {	O(1)
-	case 1:	O(1)
13-16	Getting contact name and handling	O(n)
17	while (!temp.last()) {	O(n)
18	result += temp.retrieve().Info();	O(n)
19	temp.findNext();	O(n)
20	count++;	O(n)
-	case 2:	O(1)
21-26	Searching for event by title	O(m)
27-33	Printing event details	O(1)
34-35	Default case handling	O(1)
-	Total Complexity	O(n)

Time Complexity for printAllEventsAlphabetically()

Line	Code	Time Complexity
1	if (events.empty())	O(1)
2	return;	O(1)
3	String result = "";	O(1)
4	events.findFirst();	O(1)
5	while (!events.last()) {	O(n)
6	result += events.retrieve().Info();	O(n)
7	events.findNext();}	O(n)
8	result += events.retrieve().Info();	O(1)
9	System.out.println(result);	O(1)
-	Total Complexity	O(n)

Time Complixtiy for scheduleEvent()

Line	Code	Time Complexity
1-5	Variable declarations and Scanner initialization	O(1)
6	if (criteria == 1) {	O(1)
7-12	Setting event details	O(1)
13	String[] names = name.split(",");	O(n)
14	for (int i = 0; i < names.length; i++) {	O(n)
15-24	Checking and adding contacts to event	O(n)
25-35	Setting event details and inserting event	O(n)
36-39	else if (criteria == 2) {	O(1)
40-49	Setting appointment details	O(1)
50	if (!eventConflictExists(tmp.getDate() && contacts.findkey(name)) {	O(n^2)
51-56	Inserting appointment and updating contact	O(n)
57-60	Else condition handling	O(1)
-	Total Complexity	O(n^2)

Time Complexity for SearchContact()

Line	Code	Time Complexity
1	String searchValue = "";	O(1)
2	Scanner input = new Scanner(System.in);	O(1)
3	switch (criteria) {	O(1)
-	case 1:	O(1)
4	System.out.print("\nEnter the contact's name:");	O(1)
5	searchValue = input.nextLine();	O(1)
6	System.out.print("");	O(1)
7	contacts.printContactByName(searchValue);	O(n)
-	case 2:	O(1)
-	... (similar structure as case 1)	O(n)
-	case 3:	O(1)
-	... (similar structure as case 1)	O(n)
-	case 4:	O(1)
-	... (similar structure as case 1)	O(n)
-	case 5:	O(1)
-	... (similar structure as case 1)	O(n)
-	default:	O(1)
8	System.out.println("\nInvalid criteria");	O(1)
	TOTAL	O(n)

Conclusion :

In conclusion, Phase 2 of our Phonebook project, which integrates a Binary Search Tree (BST), has been successfully executed.

The upgrade from a LinkedList to a BST has notably enhanced search efficiency and data management

. We appreciate the chance to advance this project and demonstrate our commitment to developing optimized software solutions.

Thank you for recognizing our work.

-Team Null Pointer