VISVESVARAYA TECHNOLOGICAL UNIVERSITY
"Jnana Sangama", Belagavi-590010

A Mini -Project Report

on

AVL TREE PHONEBOOK MANAGEMENT SYSTEM

A Dissertation work submitted in partial fulfillment of the requirement for the award of the degree

Bachelor of Engineering

In

Information Science & Engineering

For the Academic year 2022-2023

Submitted by

SHAKTHI RAJ	1MV21IS400
PAVAN KUMAR B	1MV21IS401
SANDEEP R C	1MV21IS404

Under the guidance of
Mr. Manohar R
Assistant Professor, Department of ISE

Sir M Visvesvaraya Institute of Technology, Bangalore

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING
HUNASAMARANAHALLI, BENGALURU – 562157

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY

Krishnadevaraya Nagar, International Airport Road,
Hunasmaranahalli, Bengaluru – 562157

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

It is certified that the **MINI PROJECT** entitled **"AVL TREE PHONEBOOK MANAGEMENT SYSTEM"** is carried out by **1MV21IS400 – M SHAKTHI RAJ , 1MV21IS401 – PAVAN KUMAR B, 1MV21IS404 – SANDEEP RC** bonafide **Students** of **Sir M Visvesvaraya Institute of Technology** in partial fulfillment for the 6th semester for the award of the Degree of Bachelor of Engineering in Information Science and Engineering of the **Visvesvaraya Technological University, Belagavi** during the academic year **2022-2023**.It is certified that all corrections and suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the course of Bachelor of Engineering.

Name & Signature of Guide	Name & Signature of HOD
Mr. Manohar R	**Dr. G. C. Bhanu Prakash**
Asst. professor & Internal Guide	HOD,
Dept. Of ISE,	Dept. Of ISE,
Sir MVIT	Sir MVIT
Bengaluru - 562157	Bengaluru - 562157

External Examination:

Name of Examiner	Signature with Date
1)	
.2)	

TABLE OF CONTENTS

ABSTRACT

The AVL Tree Phonebook Management System is a software application developed to efficiently manage a phonebook of contacts using the AVL tree data structure. The project aims to demonstrate the implementation of AVL trees in solving a real-life problem, showcasing their self-balancing property and efficient operations.

The system offers functionalities such as adding contacts, searching for contacts, removing contacts, and displaying the phonebook. The AVL tree ensures that the phonebook remains balanced, providing fast search, insert, and remove operations.

The project's implementation involves the development of classes for representing contacts, AVL nodes, and the AVL tree itself. The AVL tree implementation includes methods for adding contacts, searching contacts, removing contacts, and displaying the phonebook. The tree maintains balance through rotations and height updates.

The project also includes a user interface where users can interact with the phonebook management functionalities through a simple command-line interface. Users can add new contacts, search for contacts by name, remove contacts, and view the phonebook.

The AVL Tree Phonebook Management System showcases the practical application of AVL trees in solving a real-life problem. The project report provides an overview of the implementation details, usage instructions, and future enhancements.

By completing this project, we gain valuable experience in AVL trees, data structure implementation, and algorithmic problem-solving. The project serves as a practical example of applying data structures to solve specific problems and highlights the importance of choosing the right data structure for efficient operations.

CHAPTER 1 INTRODUCTION

1.1 Introduction to File Structures

1.2 File Structures is the Organization of Data in Secondary Storage Device in such a way that minimize the access time and the storage space. A File Structure is a combination of representations for data in files and of operations for accessing the data. A File Structure allows applications to read, write and modify data. It might also support finding the data that matches some search criteria or reading through the data in some particular order.

“File organization” refers to the logical relationship among the various records that constitute the file, particular with respect to means of identification access to any specific record. “File structure” refers to the format of label and data blocks and of any logical record control, information. The organization of the given file maybe sequential relative, or indexed.

Co-Sequential Processing

Co-sequential Processing is the coordinated processing of two or more sequential lists to produce single output list. Sometimes the processing results in a merging, or union, sometimes it results in a matching, or intersection and other times the operation is a combination of matching and merging of the items in the lists.

Aim of the File Structure

The need for file structures

- Ideally, we would like to get the information we need once access to the disk
- If it is impossible to get what we need in one access, we want structures that allow us to find the target information with as few accesses as possible.
- We want our file structures to group information so we are likely to get thing We need with only trip to the disk.

1.2 Fundamental Operations on File

Open: A function or system call that makes a file ready for use. It may also bind a logical filename to a physical file. Its arguments include the logical filename, the physical filename and may also include information on how the file is to be accessed.

Close: A function or system call that breaks the link between the logical filename and the corresponding physical filename.

Create: A function or system call that causes a file to be created on secondary storage and may also binds a logical filename to the file’s physical filename.

Read: A function or a system call used to obtain input from a file or device. When viewed at the lowest level, it requires three arguments: a source file logical name corresponding to an open file; the destination address for the address and the size or amount of data to be read.

Write: A function or system call used to provide output capabilities. When viewed at the lowest level, it requires three arguments: a destination filename corresponding to an open file; the source address of the bytes that are to be written and the size or amount of data to be written.

Seek: A function or a system call that sets the read/write pointer to a specified position in a file. Languages that provide seeking functions allow programs to access specific.

1.3 Services provided to the user

The functions that the Source Code Storage System provides are as follows:

1. ADD_CONTACT: This operation is performed when new data needs to be added to the system.

2. REMOVE_CONTACT: This operation clears the existing records in the various files. It is used when for e.g. a member leaves Source Code Storage System or when source code file is to be deleted is disposed of from system.. But care must be taken while performing this operation and permission taken from the author of files because the system could lose any important data.

3. SEARCH_CONTACT: In Python,search() is a method of the module re. re.search(pattern,string):it is similar to re. match () but it doesn’t limit us to find matches at the beginning of the string only.
4. DISPLAY_CONTACT: This is used to display each and every record from the system.

CHAPTER 2 HARDWARE AND SOFTWARE REQUIREMENTS

2.1 Software requirements

- Documentation Tool: MS Word
- Development Tool: Visual Studio Code, Python 3.

2.2 Hardware requirements

- Windows 8/8.1/10
- 4GB RAM
- 64 bit operating system

16 GB Hard disk

CHAPTER 3 SYSTEM OVERVIEW

3.1 Problem Definition

Manual system involves a lot of paper work" so it becomes time consuming and costly .The chances of errors in calculation of delivery of Books are more in the current manual system. The calculation of total collection for day or month or year is very difficult. Currently no security is provided to the large amount of data of the every book details. It becomes very difficult to maintain details of every Book as records increases day by day.

3.1.1 Solution on problem

Creating new software we should analyze what is the basic need of the software. Analysis is nothing but a planning of creation of software to get proper output from it. Analysis is detail study of protects that you want to show in your software solving problems. The basic need of the software is to save the time of the user with the help of all useful information. And also to maintain the collection of data in your computer systematically" so that it's easy to understand. The proposed system provides lot of facility to the user to store information of the Books and it provides information in quick time in a systematic manner.

The processing time on the data is very fast. It provides required data quickly to the user and also in specified manner to the user. All the information of Books changes is given to the user and also the reports are also generated according to the requirement of the user. Today it is becoming very difficult to maintain record manually. This software system easily does the job of maintaining daily records as well as the transaction according to the user requirements.

3.2 Avial Trees:

In computer science, an AVL tree (named after inventors Adelson-Velsky and Landis) is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases, where

n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

AVL Tree Phonebook Management System

Introduction

The AVL Tree Phonebook Management System is a software application that allows users to manage a phonebook of contacts using an AVL tree data structure. The system provides functionalities such as adding contacts, searching for contacts, removing contacts, and displaying the phonebook.

This project aims to demonstrate the implementation of an AVL tree and its application in solving a real-life problem. The AVL tree ensures that the phonebook remains balanced, providing efficient search, insert, and remove operations.

Features

The AVL Tree Phonebook Management System offers the following features:

1. Add Contact: Users can add new contacts to the phonebook by entering the contact name and phone number.

2. Search Contact: Users can search for a contact by entering the contact name. The system displays the contact information if found; otherwise, it notifies the user that the contact was not found.

3. Remove Contact: Users can remove a contact from the phonebook by entering the contact name. The system removes the contact if it exists and updates the phonebook accordingly.

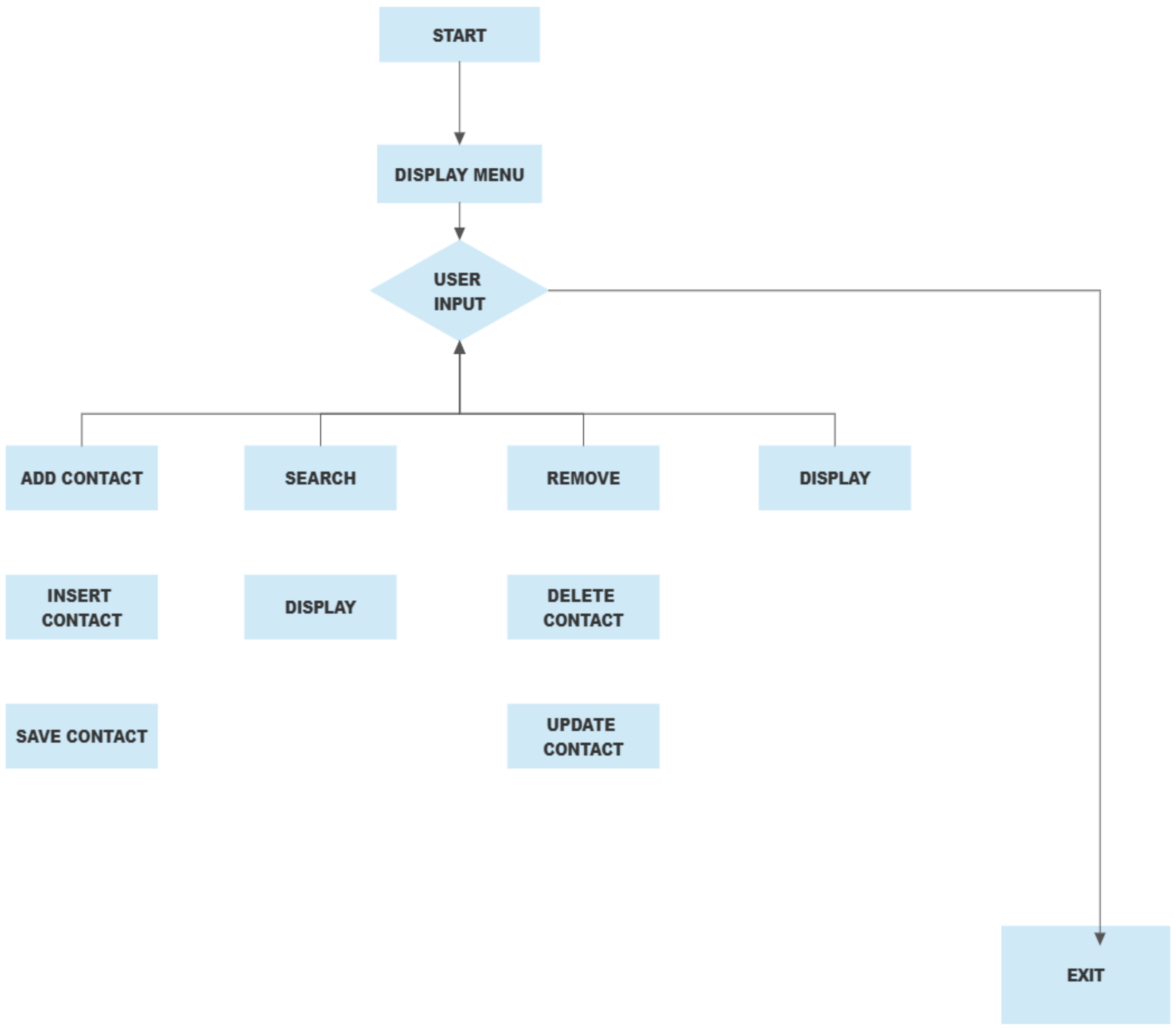
4. Display Phonebook: Users can view the entire phonebook, displaying all the contacts in alphabetical order.

CHAPTER 4 DESIGN

4.1 DATA FLOW DESIGN

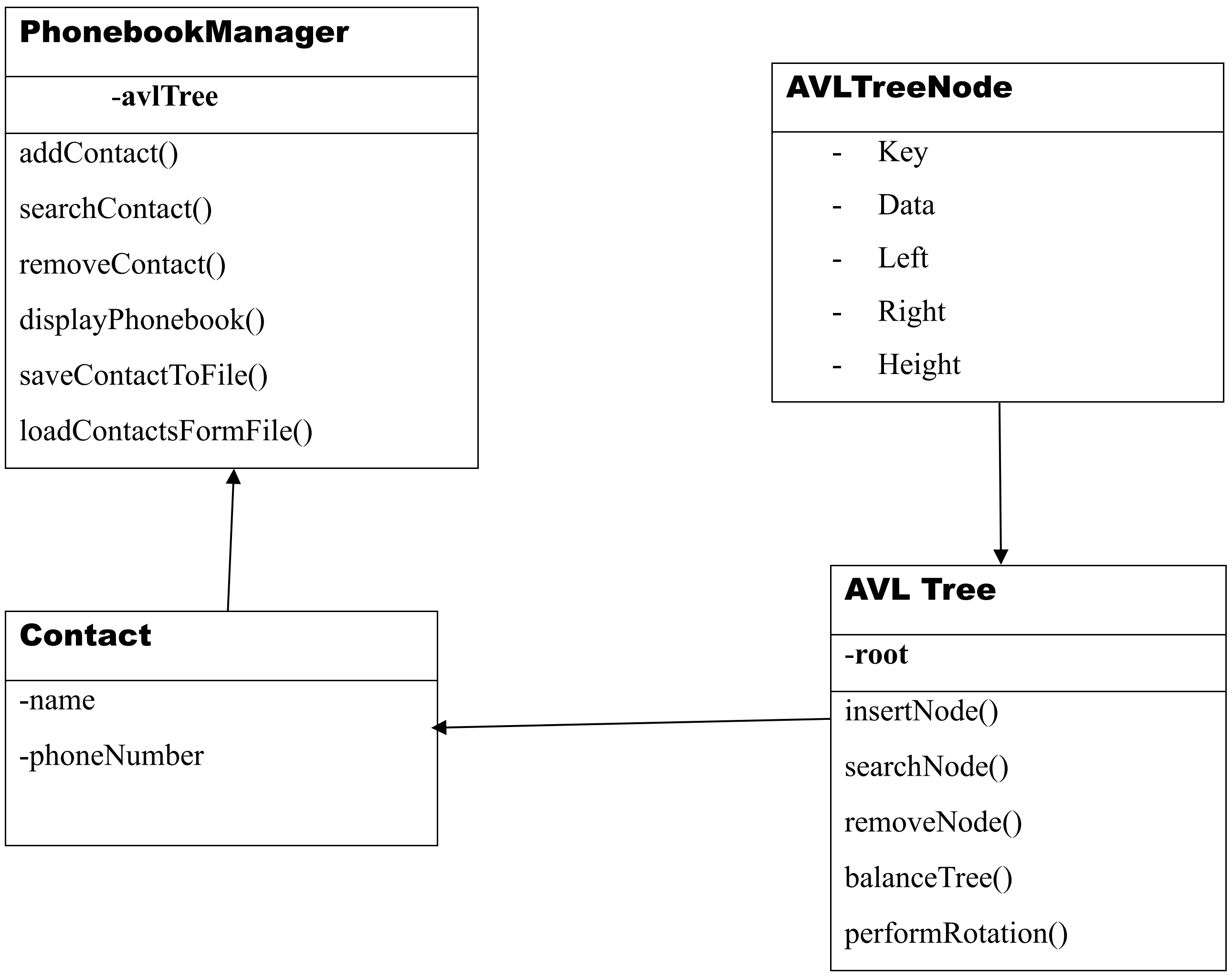
A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications

tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.



4.2 CLASS DIAGRAM

A class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML). In this context, a class defines the methods and variables in an object, which is a specific entity in a program or the unit of code representing that entity. Class diagrams are useful in all forms of object-oriented programming (OOP). The concept is several years old but has been refined as OOP modeling paradigms have evolved.



CHAPTER 5 IMPLEMENTATION

The AVL Tree Phonebook Management System is implemented using the following components:

1. Contact Class: Represents a single contact, storing the contact name and phone number.

2. AVL Node Class: Represents a node in the AVL tree. It contains the contact information, pointers to the left and right child nodes, and the height of the node.

3. AVL Tree Class: Implements the AVL tree data structure and provides methods for adding contacts, searching contacts, removing contacts, and displaying the phonebook. The AVL tree ensures that the tree remains balanced using rotations and height updates.

4. User Interface: The system provides a simple command-line interface where users can interact with the phonebook management functionalities.

Usage

To use the AVL Tree Phonebook Management System, follow these steps:

1. Run the program: Execute the program in a Python environment that supports the required dependencies.

2. Menu: The program displays a menu with different options:

- Add Contact: Choose this option to add a new contact to the phonebook. Enter the contact name and phone number as prompted.

- Search Contact: Choose this option to search for a contact in the phonebook. Enter the contact name to search.

- Remove Contact: Choose this option to remove a contact from the phonebook. Enter the contact name to remove.

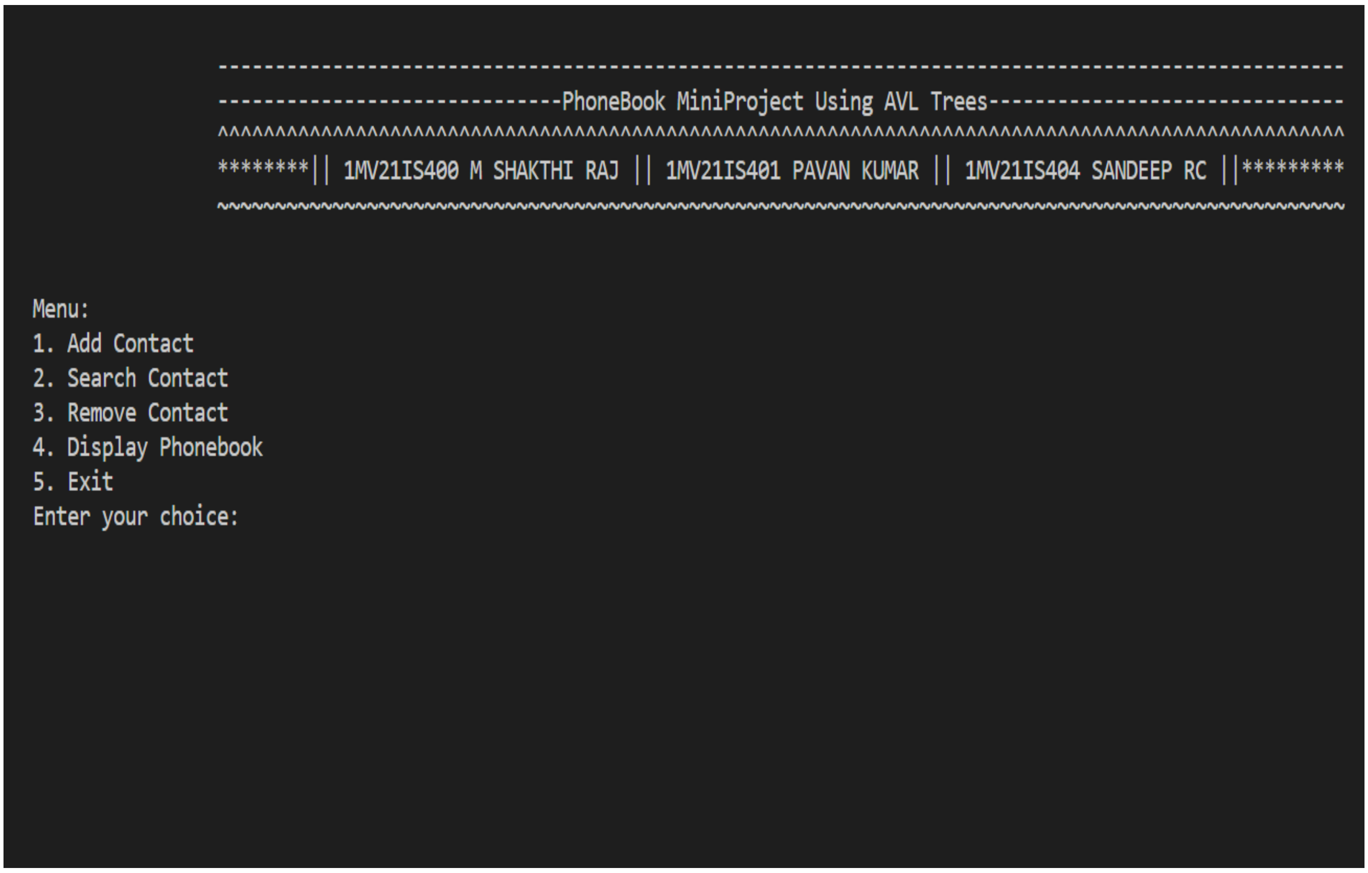
- Display Phonebook: Choose this option to view all contacts in the phonebook. The contacts are displayed in alphabetical order.

- Exit: Choose this option to exit the program.

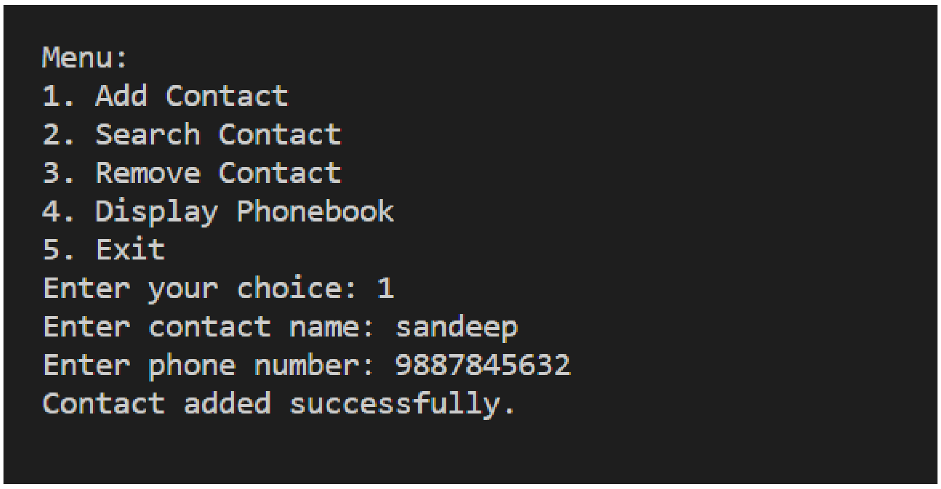
3. Follow the prompts and select the desired options to perform various operations on the phonebook.

CHAPTER 6

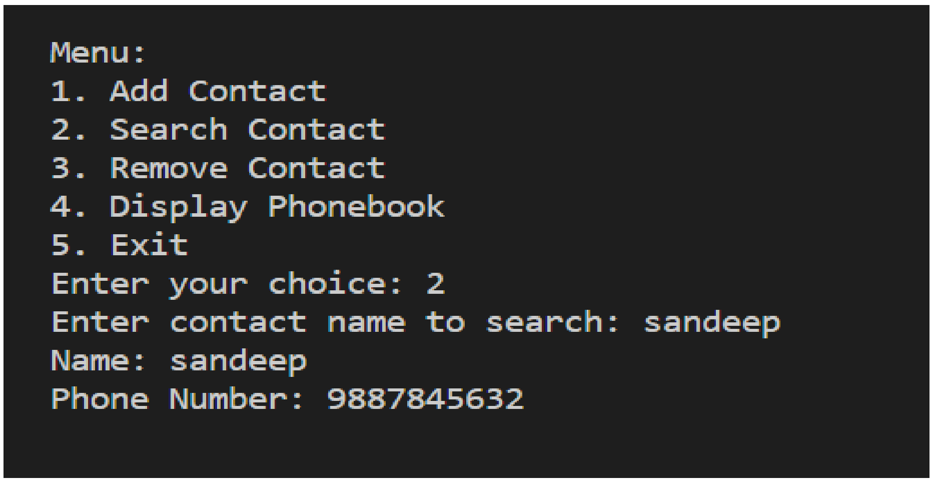
SNAPSHOTS



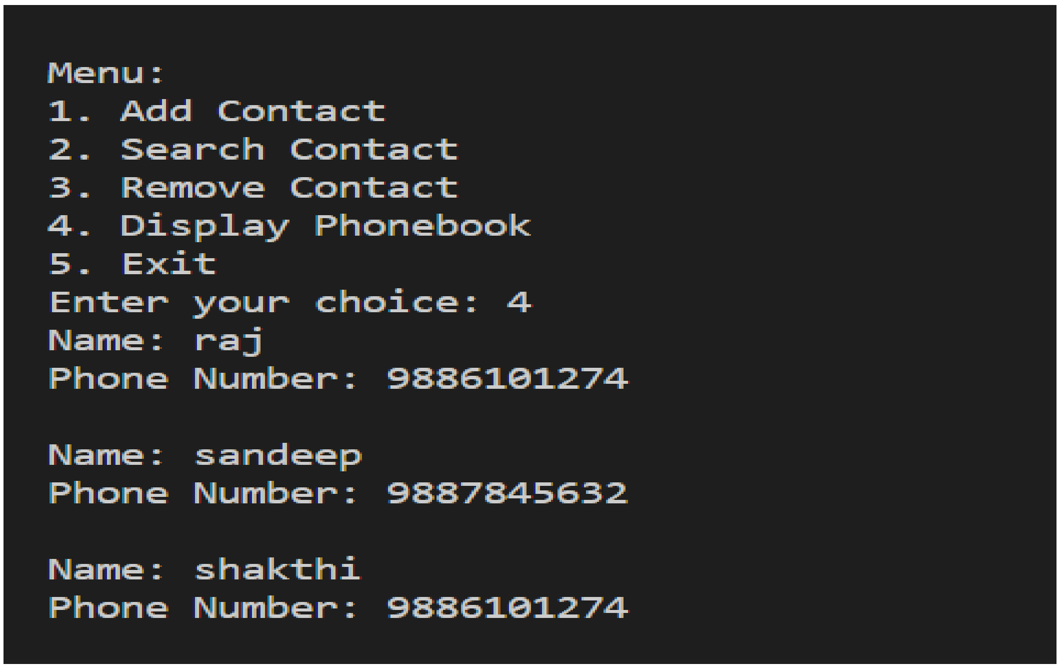
```
               -------------------------------------------------------------------------------------------------
               ------------------------------PhoneBook MiniProject Using AVL Trees------------------------------
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
               ********|| 1MV21IS400 M SHAKTHI RAJ || 1MV21IS401 PAVAN KUMAR || 1MV21IS404 SANDEEP RC ||*********
               ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice:
```



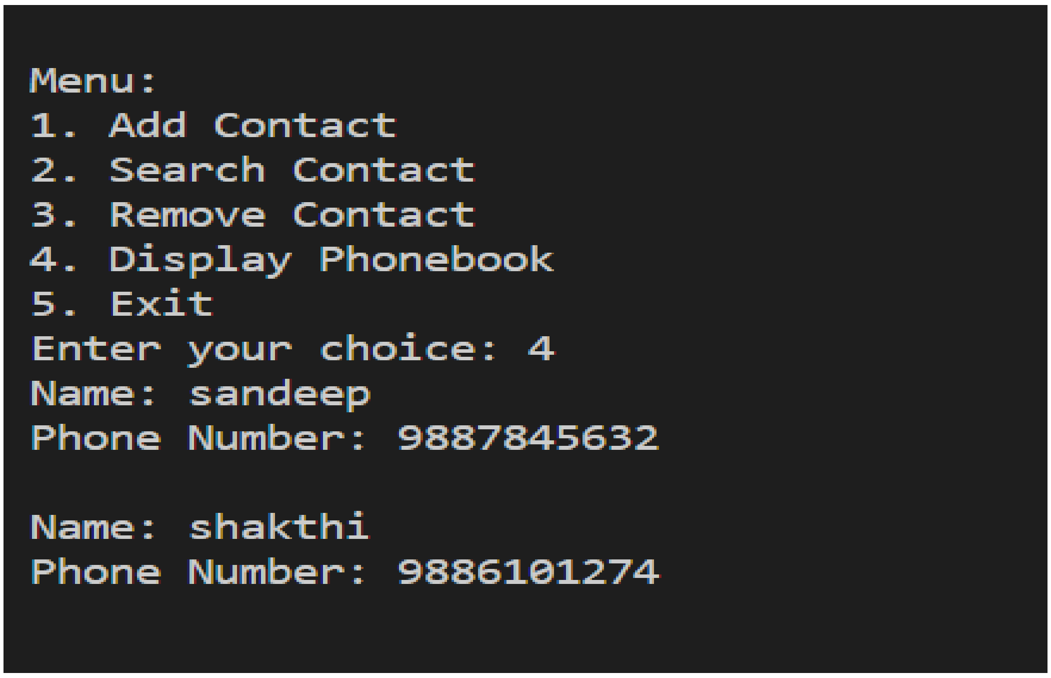
```
Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice: 1
Enter contact name: sandeep
Enter phone number: 9887845632
Contact added successfully.
```



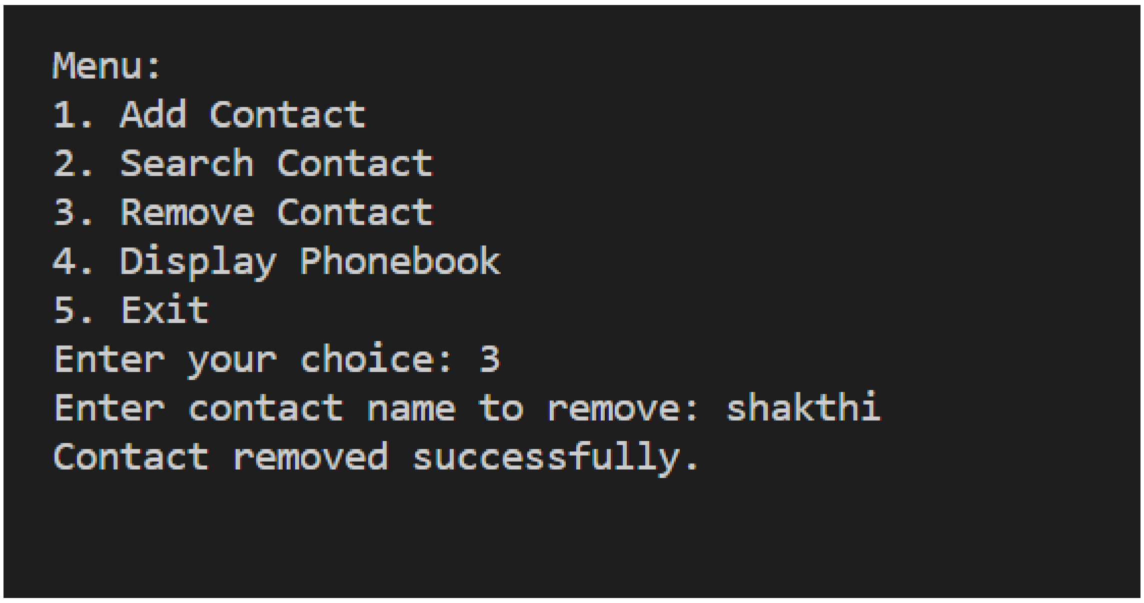
```
Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice: 2
Enter contact name to search: sandeep
Name: sandeep
Phone Number: 9887845632
```



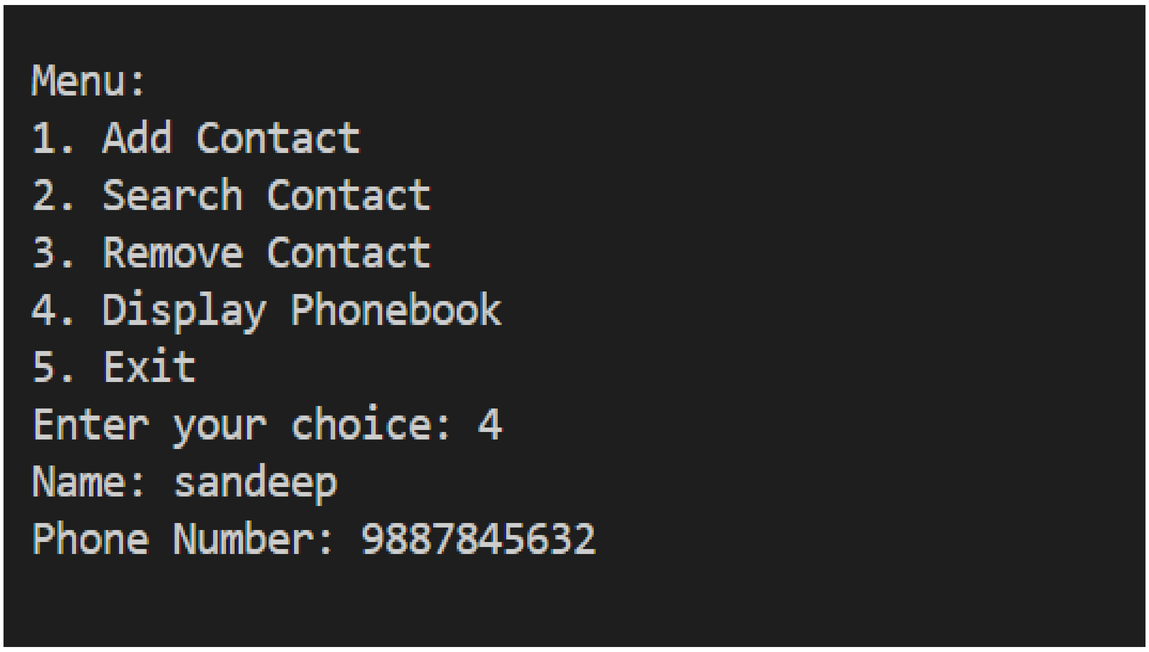
```
Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice: 4
Name: raj
Phone Number: 9886101274

Name: sandeep
Phone Number: 9887845632

Name: shakthi
Phone Number: 9886101274
```



```
Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice: 4
Name: sandeep
Phone Number: 9887845632

Name: shakthi
Phone Number: 9886101274
```



```
Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice: 3
Enter contact name to remove: shakthi
Contact removed successfully.
```



```
Menu:
1. Add Contact
2. Search Contact
3. Remove Contact
4. Display Phonebook
5. Exit
Enter your choice: 4
Name: sandeep
Phone Number: 9887845632
```

CONCLUSIONS

The AVL Tree Phonebook Management System demonstrates the application of AVL trees in managing a phonebook efficiently. The use of AVL trees ensures a balanced tree structure, which provides fast search, insert, and remove operations.

This project highlights the importance of choosing appropriate data structures for solving real-life problems. The AVL tree's self-balancing property makes it suitable for scenarios where efficient operations and maintaining balance are essential, such as phonebook management.

By implementing this project, we have gained practical experience in AVL tree data structure and its application in solving a real-life problem. We have also enhanced our programming skills and understanding of data structures and algorithms.

Overall, the AVL Tree Phonebook Management System project serves as an excellent example of applying data structures to solve a specific problem and demonstrates the effectiveness of AVL trees in maintaining balance and providing efficient operations.

FUTURE ENHANCEMENT

Possible enhancements to the AVL Tree Phonebook Management System include:

1. User Interface Improvements: Enhance the user interface by developing a graphical user interface (GUI) for a more user-friendly experience.

2. Additional Functionality: Add more features to the system, such as updating contact information, sorting contacts by different criteria, or exporting/importing contacts to/from different file formats.

3. Performance Optimization: Analyze and optimize the performance of the AVL tree operations for large-scale phonebooks, considering factors like memory usage, search speed, and scalability.

4. Error Handling: Implement robust error handling mechanisms to handle exceptions, input validation, and file I/O errors gracefully.

5. Data Persistence: Enhance the system to store contacts in a persistent database rather than a text file, allowing seamless data management and long-term storage.

With these potential enhancements, the AVL Tree Phonebook Management System can evolve into a more comprehensive and feature-rich application for efficient phonebook management.

ACKNOWLEDGMENTS

We would like to express our gratitude to the developers and contributors of the AVL tree data structure, whose work and research have made this project possible. We also thank the open-source community for providing valuable resources and support.

REFERENCES

[1] GeeksforGeeks. "AVL Tree | Set 1 (Insertion)." Available online: https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

[2] Wikipedia. "AVL Tree." Available online: https://en.wikipedia.org/wiki/AVL_tree

[3] Open Data Structures. "Chapter 10: AVL Trees." Available online: https://opendatastructures.org/ods-python/10_1_AVL_Tree.html

Appendix: Source Code

The complete source code for the AVL Tree Phonebook Management System in Python is available on [GitHub](https://github.com/raj9426/file-structures-mini-project-using-avl-trees-python).