

## Problem Description

GatorLibrary is a fictional library that needs a software system to efficiently manage its books, patrons, and borrowing operations. The system should utilize a Red-Black tree data structure to ensure efficient management of the books. Implement a priority-queue mechanism using Binary Min-heaps as a data structure for managing book reservations in case a book is not currently available to be borrowed. Each book will have its own min-heap to keep track of book reservations made by the patrons.

## Index

<b>Sr No</b>	<b>Topic</b>	<b>Page No.</b>
<b>1</b>	Program Structure	<b>2</b>
<b>2</b>	Function Explanations	<b>2</b>
<b>2.2</b>	gatorLibrary.py	<b>2</b>
<b>2.3</b>	binaryminheap.py	<b>4</b>
<b>2.4</b>	redblacktree.py	<b>5</b>
<b>3</b>	Execution and Input/Output	<b>7</b>
<b>4</b>	ColorFlip Execution Logic	<b>9</b>

## 1) Program Structure

-Shah\_Stavan.zip

---gatorLibrary\_report.pdf

---gatorLibrary.py

---binaryheap.py

---redblacktree.py

---testcase.txt

---testcase\_output.txt

## 2) Function Explanations

### 2.1.gatorLibrary.py:

The GatorLibrary class acts as the central interface for the GatorLibrary system, encompassing the overall functionality. It incorporates instances of the Red-Black Tree (books\_rb\_tree) to handle book management and a MinHeap (reservationHeap) for reservation operations. Within this class, there are methods for reading input files, displaying book details, inserting, borrowing, returning, deleting books, finding the nearest book, and terminating the program. By coordinating these diverse functions, the GatorLibrary class ensures the effective management of the library, maintaining the accuracy of book data and reservations. Additionally, the class features a method to tally and exhibit the count of color flips during tree operations, offering insights into the Red-Black Tree's balancing process.

#### Structure:

##### GatorLibrary Class:

##### Attributes:

- **books\_rb\_tree:** Red-Black Tree to manage books.
- **reservation\_heap:** MinHeap for book reservations.

##### Detailed Explanation:

#### 1. readfile(self, file\_path)

- **Input: file\_path** - Path to the input file.
- **Output:** List of lines from the file.
- **Explanation:** Reads the content of the specified file, catching errors like file not found or general exceptions.

#### 2. PrintBook(self, book\_id)

- **Input: book\_id** - ID of the book to be printed.
- **Output:** Prints details of the specified book.

- **Explanation:** Searches for the book in the Red-Black Tree and prints its details if found.
3. **PrintBooks(self, book\_id1, book\_id2)**
- **Input:** **book\_id1** and **book\_id2** - Range of book IDs.
  - **Output:** Prints details of books within the specified ID range.
  - **Explanation:** Calls **\_print\_books\_range\_helper** to print book details within the given range.
4. **InsertBook(self, book\_id, book\_name, author\_name, availability\_status=True, borrowed\_by=None, reservation\_heap=None)**
- **Input:** Book details including ID, name, author, availability status, borrower list, and reservation heap.
  - **Output:** Inserts a new book into the Red-Black Tree.
  - **Explanation:** Checks if the book with the given ID already exists and inserts a new book if not.
5. **BorrowBook(self, patron\_id, book\_id, patron\_priority)**
- **Input:** Patron ID, book ID, and patron priority.
  - **Output:** Handles the process of borrowing a book.
  - **Explanation:** Searches for the book, updates its status or adds a reservation if unavailable.
6. **ReturnBook(self, patron\_id, book\_id)**
- **Input:** Patron ID and book ID.
  - **Output:** Handles the process of returning a book.
  - **Explanation:** Updates the book status, removes the borrower, and assigns the book to the next patron in the reservation heap if available.
7. **DeleteBook(self, book\_id)**
- **Input:** Book ID.
  - **Output:** Deletes a book and cancels reservations if any.
  - **Explanation:** Searches for the book, notifies patrons if reservations are canceled, and deletes the book.
8. **FindClosestBook(self, target\_id)**
- **Input:** Target ID.
  - **Output:** Prints details of the closest book(s) to the target ID.
  - **Explanation:** Calls the corresponding method in the Red-Black Tree and prints book details.

### 9. **Quit(self)**

- **Output:** Exits the program.
- **Explanation:** Prints a termination message and exits the program.

### 10. **ColorFlipCount(self)**

- **Output:** Prints and returns the color flip count of the Red-Black Tree.
- **Explanation:** Retrieves and prints the color flip count of the Red-Black Tree.

## 2.2 [binaryminheap.py](#)

The MinHeap class implements a min-heap, a binary heap data structure where the parent node's value is less than or equal to its children's values. This structure is employed to manage reservations in the GatorLibrary system. The class includes methods for inserting a new node, extracting the node with the minimum priority, and performing heapify operations to maintain the heap property. The heap is utilized to handle reservations, allowing for efficient retrieval of the highest-priority reservation when a book becomes available.

### Detailed Explanation:

#### 1. **MinHeapNode.\_\_init\_\_(var\_self, priority, value)**

- **Input:** **priority** - Priority of the node. **value** - Value of the node.
- **Output:** Initializes a MinHeapNode with the provided priority and value.
- **Explanation:** This is the constructor for the **MinHeapNode** class, creating a node with a specified priority and value.

#### 2. **MinHeap.\_\_init\_\_(var\_self)**

- **Output:** Initializes an empty MinHeap.
- **Explanation:** This is the constructor for the **MinHeap** class, creating an empty heap represented by a list.

#### 3. **MinHeap.insert(var\_self, priority, value)**

- **Input:** **priority** - Priority of the node to be inserted. **value** - Value of the node to be inserted.
- **Output:** Inserts a new node into the heap and maintains the heap property.
- **Explanation:** Creates a new node with the provided priority and value, appends it to the heap, and calls **\_heapify\_up** to maintain the heap property.

#### 4. **MinHeap.extract\_min(var\_self)**

- **Output:** Extracts the node with the minimum priority from the heap.
- **Explanation:** If the heap is not empty, it extracts the root (node with the minimum priority), replaces the root with the last node, and calls **\_heapify\_down** to maintain the heap property.

**5. MinHeap.is\_empty(var\_self)**

- **Output:** Returns **True** if the heap is empty, **False** otherwise.
- **Explanation:** Checks if the heap is empty by examining the length of the heap list.

**6. MinHeap.\_heapify\_up(var\_self, index)**

- **Input: index** - Index of the node to be moved up.
- **Output:** Moves the node at the given index up the heap to maintain the heap property.
- **Explanation:** Compares the node with its parent and swaps them if necessary until the heap property is satisfied.

**7. MinHeap.\_heapify\_down(var\_self, index)**

- **Input: index** - Index of the node to be moved down.
- **Output:** Moves the node at the given index down the heap to maintain the heap property.
- **Explanation:** Compares the node with its children and swaps it with the smallest child if necessary until the heap property is satisfied.

**8. MinHeap.get\_heap\_elements(var\_self)**

- **Output:** Returns a list of values of all nodes in the heap.
- **Explanation:** Retrieves and returns a list of values from all nodes in the heap

## 2.3 [redblacktree.py](#)

The RedBlackTree class implements a Red-Black Tree, a self-balancing binary search tree that maintains balance through color-coding of nodes. This class is crucial for organizing and managing books efficiently in the GatorLibrary system. It includes methods for searching, inserting, deleting nodes, and various balancing operations to ensure the tree remains balanced. The Red-Black Tree enhances search and retrieval performance, providing a reliable structure for managing the vast collection of books in the library.

### Detailed Explanation:

**1. MinHeapNode.\_\_init\_\_(var\_self, priority, value)**

- Initializes a MinHeapNode with the provided priority and value.

**2. MinHeap.\_\_init\_\_(var\_self)**

- Initializes an empty MinHeap.

**3. MinHeap.insert(var\_self, priority, value)**

- Inserts a new node into the heap and maintains the heap property.

**4. MinHeap.extract\_min(var\_self)**

- Extracts the node with the minimum priority from the heap.

5. **MinHeap.is\_empty(var\_self)**
  - Checks if the heap is empty.
6. **MinHeap.\_heapify\_up(var\_self, index)**
  - Moves the node at the given index up the heap to maintain the heap property.
7. **MinHeap.\_heapify\_down(var\_self, index)**
  - Moves the node at the given index down the heap to maintain the heap property.
8. **MinHeap.get\_heap\_elements(var\_self)**
  - Returns a list of values of all nodes in the heap.
9. **Node.\_\_init\_\_(var\_self, book)**
  - Initializes a Node with the provided book.
10. **Book.\_\_init\_\_(var\_self, book\_id, book\_name, author\_name, availability\_status=True, borrowed\_by=None, reservation\_heap=None)**
  - Initializes a Book object with the provided attributes.
11. **Book.\_\_str\_\_(var\_self)**
  - Returns a string representation of the Book object.
12. **RedBlackTree.\_\_init\_\_(var\_self)**
  - Initializes an empty Red-Black Tree.
13. **RedBlackTree.search\_tree\_helper(var\_self, node, key)**
  - Searches the tree for a given key and returns the corresponding node.
14. **RedBlackTree.color\_flip\_count\_reset(var\_self)**
  - Resets the color flip count.
15. **RedBlackTree.color\_flip\_count\_increment(var\_self)**
  - Increments the color flip count.
16. **RedBlackTree.flip\_count\_update(var\_self, previous, updated)**
  - Updates the color flip count based on changes in color.
17. **RedBlackTree.fix\_delete(var\_self, x)**
  - Balances the tree after a node deletion.
18. **RedBlackTree.fix\_insert(var\_self, k)**
  - Balances the tree after a node insertion.
19. **RedBlackTree.left\_rotate(var\_self, x)**
  - Performs a left rotation around the specified node.

**20. RedBlackTree.right\_rotate(var\_self, x)**

- Performs a right rotation around the specified node.

**21. RedBlackTree.insert(var\_self, key, book)**

- Inserts a new node into the Red-Black Tree.

**22. RedBlackTree.get\_min\_value\_node(var\_self, node)**

- Returns the node with the minimum value in the subtree rooted at the given node.

**23. RedBlackTree.get\_max\_value\_node(var\_self, node)**

- Returns the node with the maximum value in the subtree rooted at the given node.

**24. RedBlackTree.delete\_node(var\_self, key)**

- Deletes a node with the specified key from the tree.

**25. RedBlackTree.delete\_node\_helper(var\_self, root, key)**

- Helper function for deleting a node.

**26. RedBlackTree.transplant(var\_self, u, v)**

- Replaces the subtree rooted at node u with the subtree rooted at node v.

**27. RedBlackTree.find\_closest\_books(var\_self, target\_id)**

- Finds the closest books to a given target ID.

**28. RedBlackTree.find\_closest\_lower(var\_self, node, target\_id)**

- Finds the node with the closest lower value to the target ID.

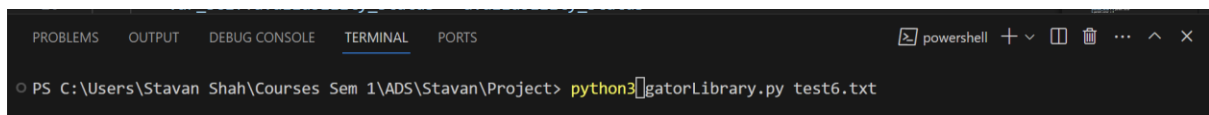
**29. RedBlackTree.find\_closest\_higher(var\_self, node, target\_id)**

- Finds the node with the closest higher value to the target ID.

### 3. Execution and Input/Output

How to run the program:

```
python3 gatorLibrary.py test6.txt
```

A screenshot of a terminal window with a dark background. The title bar at the top shows 'powershell' and several window control icons. The terminal text shows the command prompt 'PS C:\Users\Stavan Shah\Courses Sem 1\ADS\Stavan\Project>' followed by the command 'python3 gatorLibrary.py test6.txt'.

Sample Input:

```
InsertBook(1, "Book1", "Author1", "Yes")
PrintBook(1)
BorrowBook(101, 1, 1)
InsertBook(2, "Book2", "Author2", "Yes")
BorrowBook(102, 1, 2)
```

```
PrintBooks(1, 2)
```

```
ReturnBook(101, 1)
```

```
Quit()
```

Sample Output:

```
BookID = 1
```

```
Title = "Book1"
```

```
Author = "Author1"
```

```
Availability = "Yes"
```

```
BorrowedBy = None
```

```
Reservations = []
```

```
Book 1 Borrowed by Patron 101
```

```
Book 1 Reserved by Patron 102
```

```
BookID = 1
```

```
Title = "Book1"
```

```
Author = "Author1"
```

```
Availability = "No"
```

```
BorrowedBy = 101
```

```
Reservations = [102]
```

```
BookID = 2
```

```
Title = "Book2"
```

```
Author = "Author2"
```

```
Availability = "Yes"
```

```
BorrowedBy = None
```

```
Reservations = []
```

```
Book 1 Returned by Patron 101
```

```
Book 1 Allotted to Patron 102
```

```
Program Terminated!
```



## 4. ColorFlip Execution Logic

The color flip logic in the provided code is part of the Red-Black Tree implementation and is used during tree balancing operations, particularly in the **fix\_insert** and **fix\_delete** methods. Here's a summary of the color flip logic:

1. **Incremental Counting:** The code keeps track of the number of color flips during tree operations, specifically during insertions and deletions.
2. **Color Representation:** Red-Black Trees use colors to maintain balance, where nodes are either red or black. The code uses 1 for red and 0 for black.
3. **Insertion Color Flips:** During insertion (**fix\_insert**), when a new node is added to the tree, the code tracks color changes in a way that ensures the tree remains balanced. Color flips are counted when adjusting the colors of nodes and their parents during the balancing process.
4. **Deletion Color Flips:** Similarly, during deletion (**fix\_delete**), color flips are counted as the tree is adjusted to maintain its red-black properties. The code ensures that after a node is deleted, the tree remains balanced, and color flips are incremented accordingly.
5. **Total Count:** The total count of color flips is stored and can be retrieved using the **ColorFlipCount** method.

Example from Testcase1 as provided:

```
InsertBook(101, "Introduction to Algorithms", "Thomas H. Cormen", "Yes")
InsertBook(48, "Data Structures and Algorithms", "Sartaj Sahni", "Yes")
PrintBook(48)
InsertBook(132, "Operating System Concepts", "Abraham Silberschatz",
"Yes")
InsertBook(25, "Computer Networks", "Andrew S. Tanenbaum", "Yes")
BorrowBook(120, 48, 2)
BorrowBook(132, 101, 1)
InsertBook(73, "Introduction to the Theory of Computation", "Michael
Sipser", "Yes")
InsertBook(12, "Artificial Intelligence: A Modern Approach", "Stuart
Russell", "Yes")
InsertBook(6, "Database Management Systems", "Raghu Ramakrishnan", "Yes")
BorrowBook(144, 48, 3)
BorrowBook(140, 48, 3)
BorrowBook(142, 48, 2)
BorrowBook(138, 12, 4)
BorrowBook(150, 12, 3)
BorrowBook(162, 12, 1)
ReturnBook(120, 48)
FindClosestBook(9)
DeleteBook(12)
ColorFlipCount()
```

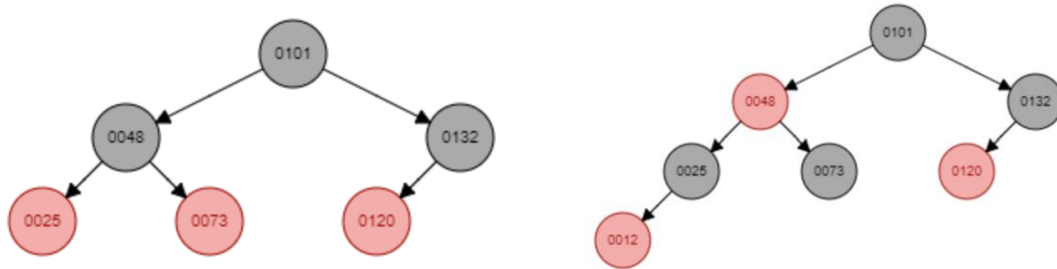
My code output:

**Color Flip Count: 10**

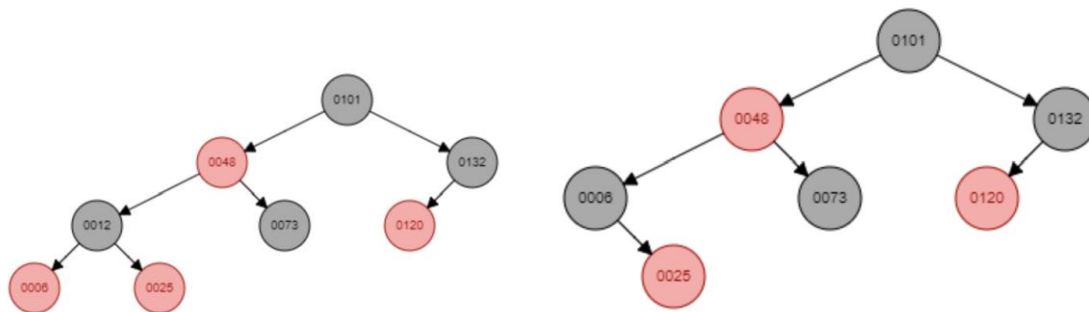
Explanation:



Color Flip: 3 (48, 132, 25)



Color Flip: 7 (48, 25, 73, 12)



Color Flip: 10 (12, 6, 25)