# Red-Black Tree and Binary Min-Heap

Project for Advanced Data Structures (COP5536)

Ujjwal Goel
ujjwalgoel@ufl.edu
UFID: 8247-9467

November 21, 2023

## Red-Black Tree and Binary Min-Heap

In this project, I have implemented a Red-Black Tree and a Binary Min-Heap. These two data structures are used to implement a library management system. Each node in the red-black tree represents a book. Further, each book has a heap that keeps track of reservations for that book.

### Installation

The code does not require any external dependencies, but if `graphviz` is installed, it can create a visualization of the Red-Black Tree.

### Usage

The code can be run using the following command:

```
python3 gatorLibrary.py file_name.txt
```

The `file_name.txt` should contain the commands to be executed. The program will create output file named as `file_name_output_file.txt` in the same directory.

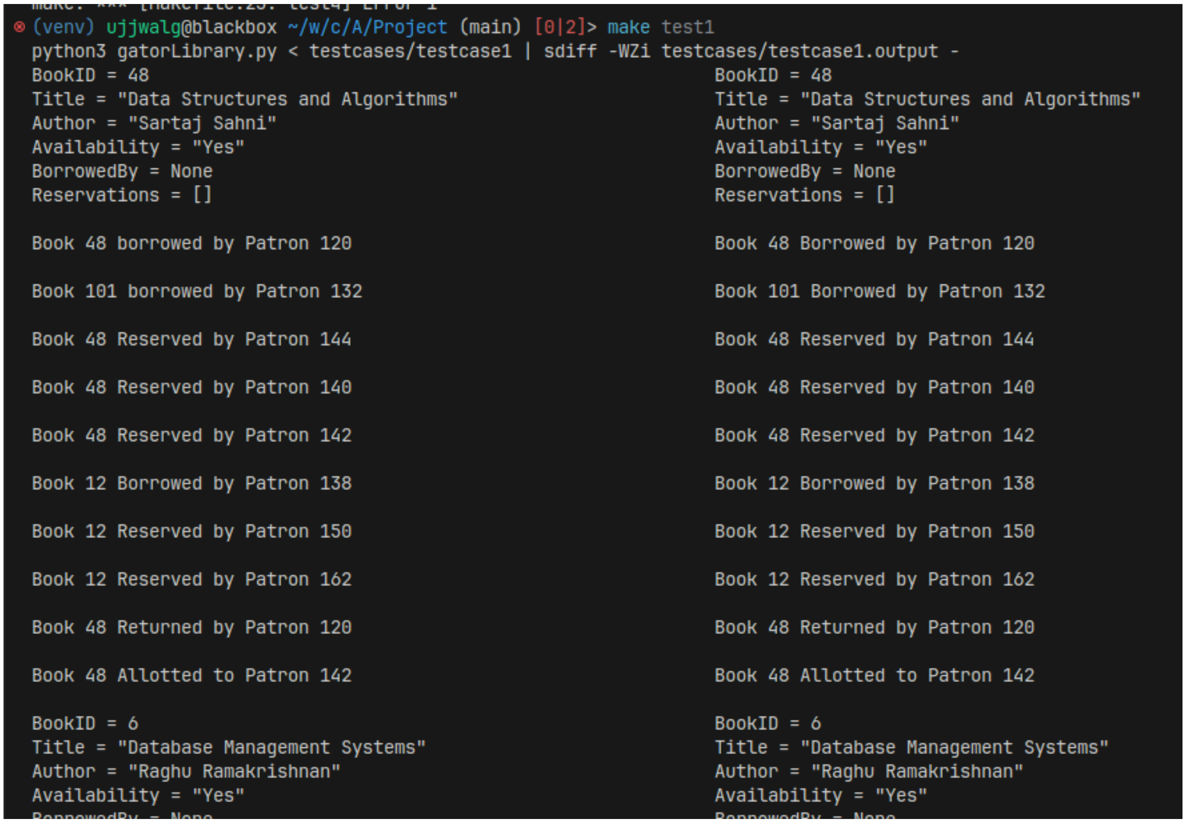
### Testing

The code was tested locally on Python 3.12. The code was also tested on the CISE Linux machines using Python 3.10. In both cases, the code ran without any issues.

The Makefile contains a number of testcases which execute the code for one of the sample input files and checks the output. The testcases can be run using the following command:

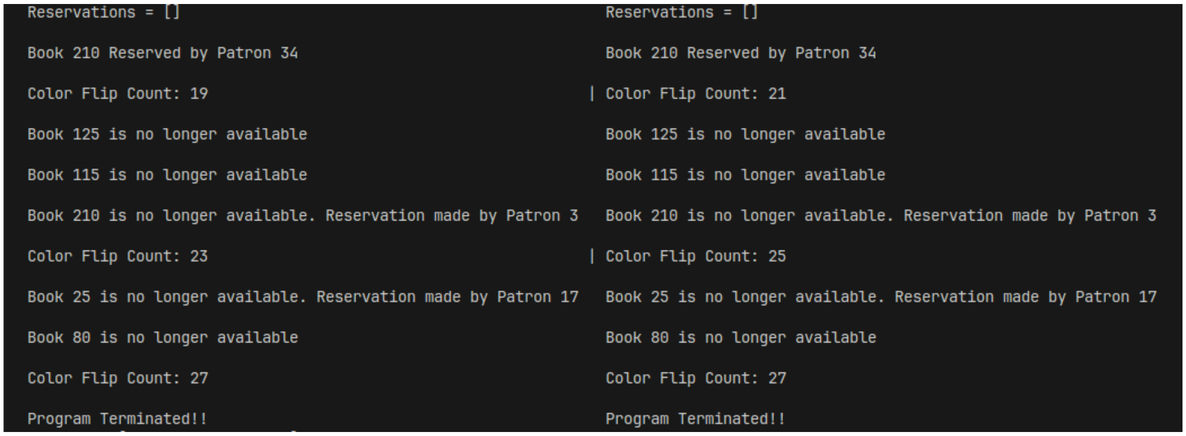
```
make test1
make test2
make test3
make test4
```

**Test case 1**



```
make: *** [Makefile:25: test4] Error 1
⊗ (venv) ujjwalg@blackbox ~/w/c/A/Project (main) [0|2]> make test1
python3 gatorLibrary.py < testcases/testcase1 | sdiff -WZi testcases/testcase1.output -
BookID = 48                                                 BookID = 48
Title = "Data Structures and Algorithms"                    Title = "Data Structures and Algorithms"
Author = "Sartaj Sahni"                                     Author = "Sartaj Sahni"
Availability = "Yes"                                        Availability = "Yes"
BorrowedBy = None                                           BorrowedBy = None
Reservations = []                                           Reservations = []

Book 48 borrowed by Patron 120                              Book 48 Borrowed by Patron 120

Book 101 borrowed by Patron 132                             Book 101 Borrowed by Patron 132

Book 48 Reserved by Patron 144                              Book 48 Reserved by Patron 144

Book 48 Reserved by Patron 140                              Book 48 Reserved by Patron 140

Book 48 Reserved by Patron 142                              Book 48 Reserved by Patron 142

Book 12 Borrowed by Patron 138                              Book 12 Borrowed by Patron 138

Book 12 Reserved by Patron 150                              Book 12 Reserved by Patron 150

Book 12 Reserved by Patron 162                              Book 12 Reserved by Patron 162

Book 48 Returned by Patron 120                              Book 48 Returned by Patron 120

Book 48 Allotted to Patron 142                              Book 48 Allotted to Patron 142

BookID = 6                                                  BookID = 6
Title = "Database Management Systems"                       Title = "Database Management Systems"
Author = "Raghu Ramakrishnan"                               Author = "Raghu Ramakrishnan"
Availability = "Yes"                                        Availability = "Yes"
BorrowedBy = None                                           BorrowedBy = None
```

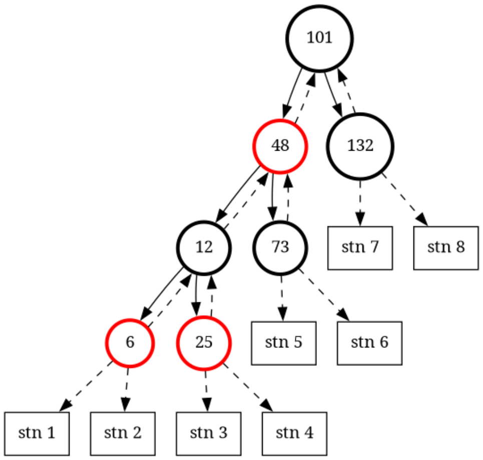
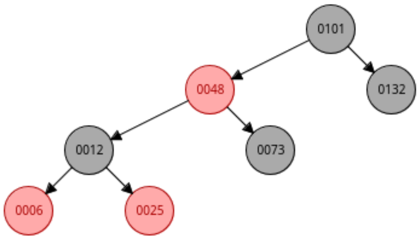
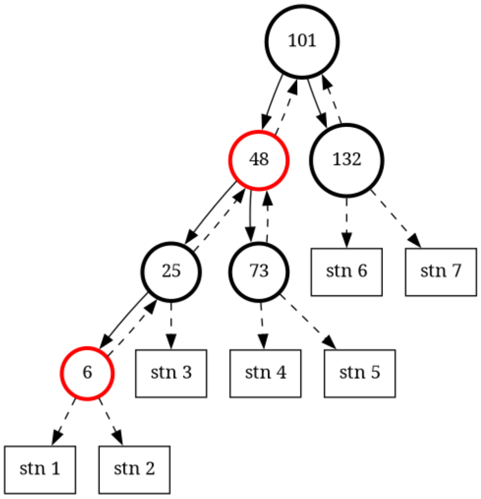
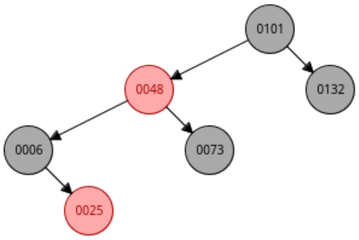
The image shows the expected output on the left and the actual output on the right side. The output is not exactly same as the expected output at two places.



```
Reservations = []                                       Reservations = []

Book 210 Reserved by Patron 34                          Book 210 Reserved by Patron 34

Color Flip Count: 19                                  | Color Flip Count: 21

Book 125 is no longer available                         Book 125 is no longer available

Book 115 is no longer available                         Book 115 is no longer available

Book 210 is no longer available. Reservation made by Patron 3  Book 210 is no longer available. Reservation made by Patron 3

Color Flip Count: 23                                  | Color Flip Count: 25

Book 25 is no longer available. Reservation made by Patron 17  Book 25 is no longer available. Reservation made by Patron 17

Book 80 is no longer available                          Book 80 is no longer available

Color Flip Count: 27                                    Color Flip Count: 27

Program Terminated!!                                    Program Terminated!!
```

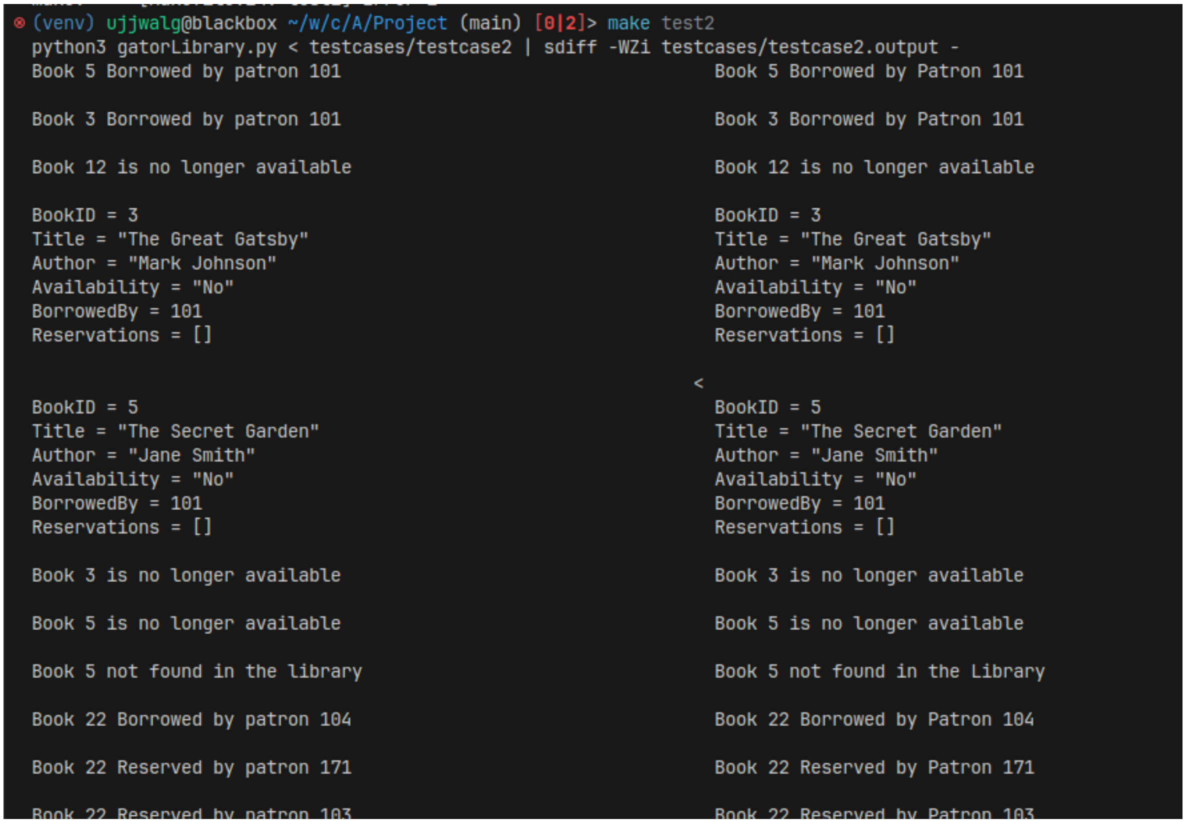
The difference is because of deletion operation. There can be multiple correct ways of implementing the deletion operation as long as the red-black tree properties are satisfied. Because of this, the color flip count can be different.

Consider the dry run of the insertion and deletion operations in the red-black tree:

| Operation | This code | Reference implementation |
|---|---|---|
| ... |  101<br>48<br>132<br>12<br>73<br>stn 7<br>stn 8<br>6<br>25<br>stn 5<br>stn 6<br>stn 1<br>stn 2<br>stn 3<br>stn 4<br>Color flip count: 7 |  |
| Delete 12 |  101<br>48<br>132<br>25<br>73<br>stn 6<br>stn 7<br>6<br>stn 3<br>stn 4<br>stn 5<br>stn 1<br>stn 2<br>Color flip count: 8 |  |

As we can see that `delete 12` produces a different tree in both the cases even though both the trees are valid red-black trees. This is the reason why color-flip count is different in both the cases.

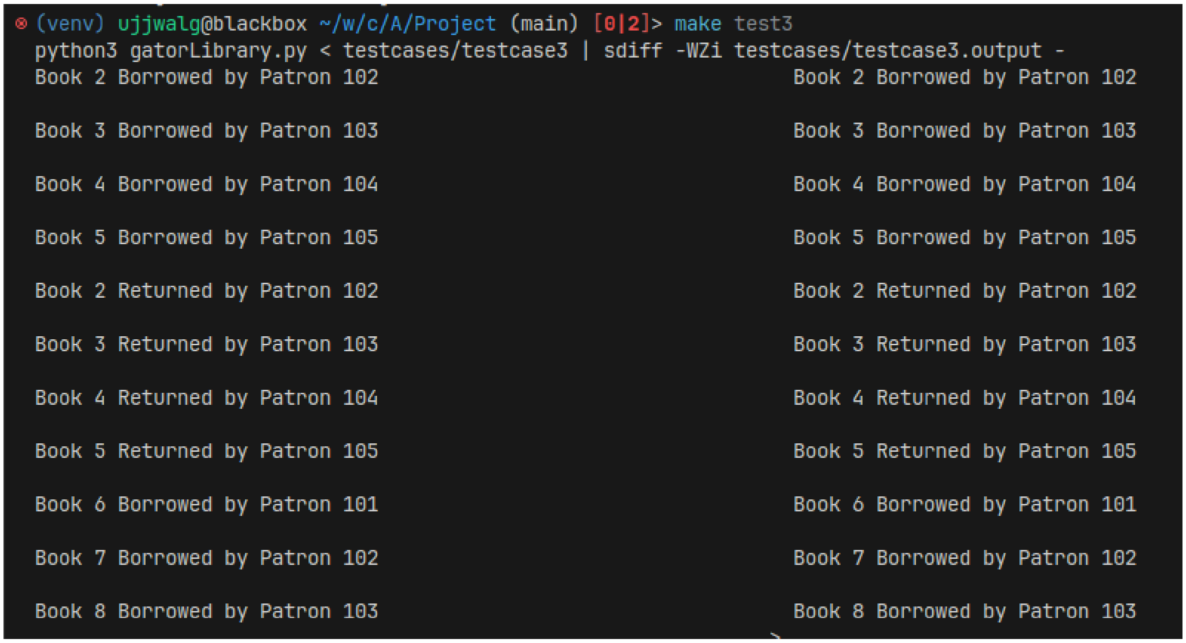
### Test case 2



```
⊗ (venv) ujjwalg@blackbox ~/w/c/A/Project (main) [0|2]> make test2
  python3 gatorLibrary.py < testcases/testcase2 | sdiff -WZi testcases/testcase2.output -
  Book 5 Borrowed by patron 101                                Book 5 Borrowed by Patron 101

  Book 3 Borrowed by patron 101                                Book 3 Borrowed by Patron 101

  Book 12 is no longer available                               Book 12 is no longer available

  BookID = 3                                                   BookID = 3
  Title = "The Great Gatsby"                                   Title = "The Great Gatsby"
  Author = "Mark Johnson"                                      Author = "Mark Johnson"
  Availability = "No"                                          Availability = "No"
  BorrowedBy = 101                                             BorrowedBy = 101
  Reservations = []                                            Reservations = []

                                                             <
  BookID = 5                                                   BookID = 5
  Title = "The Secret Garden"                                  Title = "The Secret Garden"
  Author = "Jane Smith"                                        Author = "Jane Smith"
  Availability = "No"                                          Availability = "No"
  BorrowedBy = 101                                             BorrowedBy = 101
  Reservations = []                                            Reservations = []

  Book 3 is no longer available                                Book 3 is no longer available

  Book 5 is no longer available                                Book 5 is no longer available

  Book 5 not found in the library                              Book 5 not found in the Library

  Book 22 Borrowed by patron 104                               Book 22 Borrowed by Patron 104

  Book 22 Reserved by patron 171                               Book 22 Reserved by Patron 171

  Book 22 Reserved by patron 103                               Book 22 Reserved by Patron 103
```

The output is same as the expected output for test case 2.

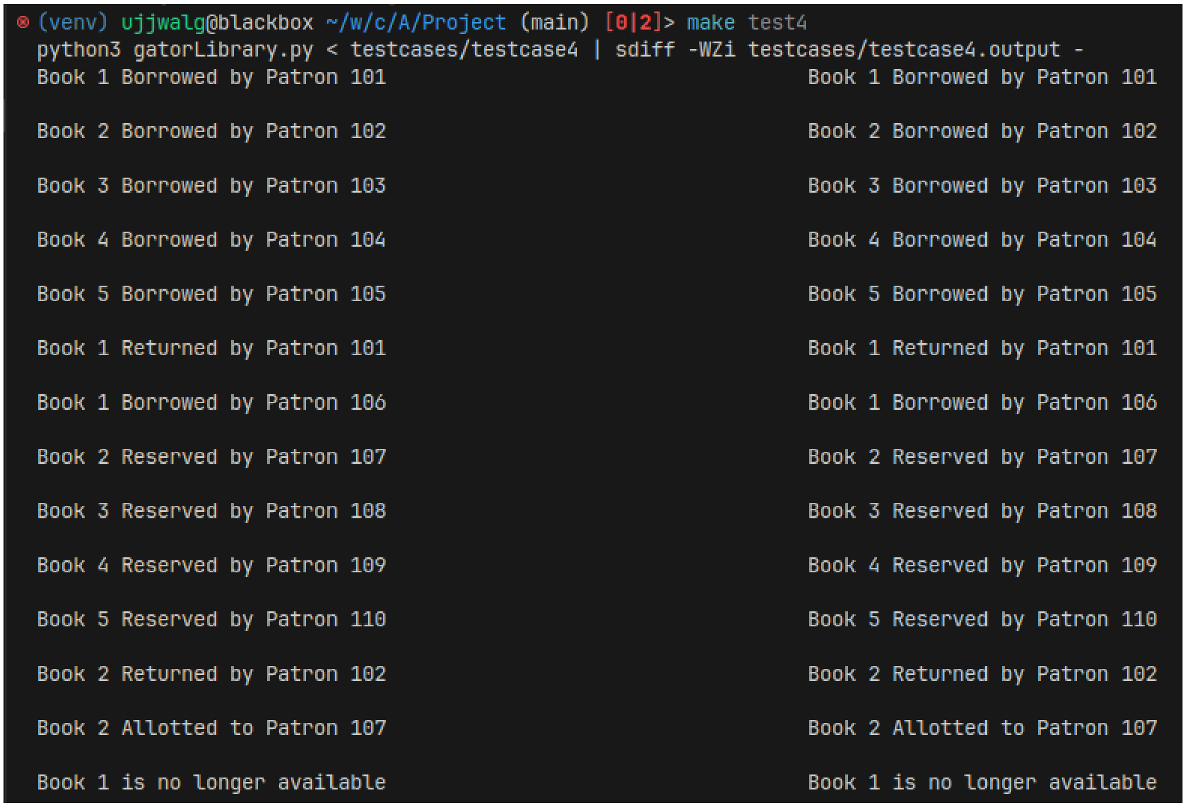
### Test case 3



```
⊗ (venv) ujjwalg@blackbox ~/w/c/A/Project (main) [0|2]> make test3
  python3 gatorLibrary.py < testcases/testcase3 | sdiff -WZi testcases/testcase3.output -
  Book 2 Borrowed by Patron 102                                Book 2 Borrowed by Patron 102

  Book 3 Borrowed by Patron 103                                Book 3 Borrowed by Patron 103

  Book 4 Borrowed by Patron 104                                Book 4 Borrowed by Patron 104

  Book 5 Borrowed by Patron 105                                Book 5 Borrowed by Patron 105

  Book 2 Returned by Patron 102                                Book 2 Returned by Patron 102

  Book 3 Returned by Patron 103                                Book 3 Returned by Patron 103

  Book 4 Returned by Patron 104                                Book 4 Returned by Patron 104

  Book 5 Returned by Patron 105                                Book 5 Returned by Patron 105

  Book 6 Borrowed by Patron 101                                Book 6 Borrowed by Patron 101

  Book 7 Borrowed by Patron 102                                Book 7 Borrowed by Patron 102

  Book 8 Borrowed by Patron 103                                Book 8 Borrowed by Patron 103
```

The output is same as the expected output for test case 3

### Test case 4



```
⊗ (venv) ujjwalg@blackbox ~/w/c/A/Project (main) [0|2]> make test4
  python3 gatorLibrary.py < testcases/testcase4 | sdiff -WZi testcases/testcase4.output -
  Book 1 Borrowed by Patron 101                                   Book 1 Borrowed by Patron 101

  Book 2 Borrowed by Patron 102                                   Book 2 Borrowed by Patron 102

  Book 3 Borrowed by Patron 103                                   Book 3 Borrowed by Patron 103

  Book 4 Borrowed by Patron 104                                   Book 4 Borrowed by Patron 104

  Book 5 Borrowed by Patron 105                                   Book 5 Borrowed by Patron 105

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

  Book 1 Borrowed by Patron 106                                   Book 1 Borrowed by Patron 106

  Book 2 Reserved by Patron 107                                   Book 2 Reserved by Patron 107

  Book 3 Reserved by Patron 108                                   Book 3 Reserved by Patron 108

  Book 4 Reserved by Patron 109                                   Book 4 Reserved by Patron 109

  Book 5 Reserved by Patron 110                                   Book 5 Reserved by Patron 110

  Book 2 Returned by Patron 102                                   Book 2 Returned by Patron 102

  Book 2 Allotted to Patron 107                                   Book 2 Allotted to Patron 107

  Book 1 is no longer available                                   Book 1 is no longer available
```

The output is same as the expected output for test case 4.

## File Structure

- `tree.py`: The red-black tree is implemented in the file `tree.py`
- `heap.py`: the binary heap is implemented in the file `heap.py`
- `gatorLibrary.py` is the main file that uses both the data structures to perform the operations specified in the input file. The file is responsible for reading the input file, creating the output.

`gatorLibrary.py` file depends on `tree.py` and `heap.py`.

**Coupling and design**: Both, the red-black tree and the binary heap are implemented as classes and they are independent of each other. The red-black tree is designed to store any kind of data, while the binary heap is designed to store only a three-tuple of integers. There is no dependency between the two data structures which keeps the coupling to a minimum. Implementation of heap and red-black tree is completely abstracted from the `gatorLibrary.py` file. Also, the `tree.py` has no knowledge of `heap.py` or `gatorLibrary.py`.

## Implementation Details

Implementation for both the data structures is based on the pseudo-code provided in Introduction to Algorithms by Cormen et al.

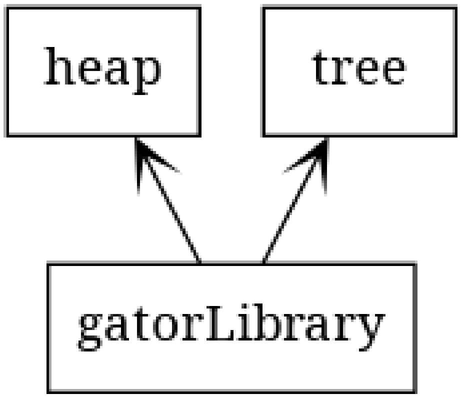


Figure 1: File Structure

### Red-Black Tree

Code for the red-black tree is implemented in the file `tree.py`. Following is the class diagram for the red-black tree generated using `pyreverse`:

The `Tree` class is the main class that implements the red-black tree. This has various methods like `insert`, `delete`, `search`, etc. This class holds pointer to the root node. These are the methods in the `Tree` class:

- `left_rotate`: This method performs a left rotation on the given node. $O(1)$ time and space complexity.
- `right_rotate`: This method performs a right rotation on the given node. $O(1)$ time and space complexity.
- `insert`: This method inserts a new node in the tree and calls `_insert_fixup` if needed to fix the tree. $O(\log n)$ time complexity.
- `_insert_fixup`: This method fixes the tree after an insertion.
- `delete`: This method deletes a node from the tree and calls `_delete_fixup` if needed to fix the tree. $O(\log n)$ time complexity.
- `_delete_fixup`: This method fixes the tree after a deletion.
- `transplant`: This method replaces one subtree as a child of its parent with another subtree. $O(1)$ time and space complexity.
- `search`: This method searches for a node in the tree with the given key. $O(\log n)$ time complexity.
- `find_closest`: This method finds the nodes with the closest key to the given key. $O(\log n)$ time complexity.
- `range_search`: This method finds all the nodes in the tree with the key in the given range. $O(n)$ time complexity.

The above functions are needed to implement the red-black tree as required by the specification. But there are some other functions that are used internally, for example, `visualize_binary_tree` is used to create a visualization of the tree and is useful for debugging. This functions works only if `graphviz` is installed.

The `TreeNode` class is actual class that represents a node in the red-black tree. This class holds the data, pointers to the parent, left and right child and the color of the node. The color is set to `BLACK` by default.

There is also a `SentinelNode` class which is used to represent the `NIL` nodes. This class inherits from the `TreeNode` class and overrides the `__init__` method.

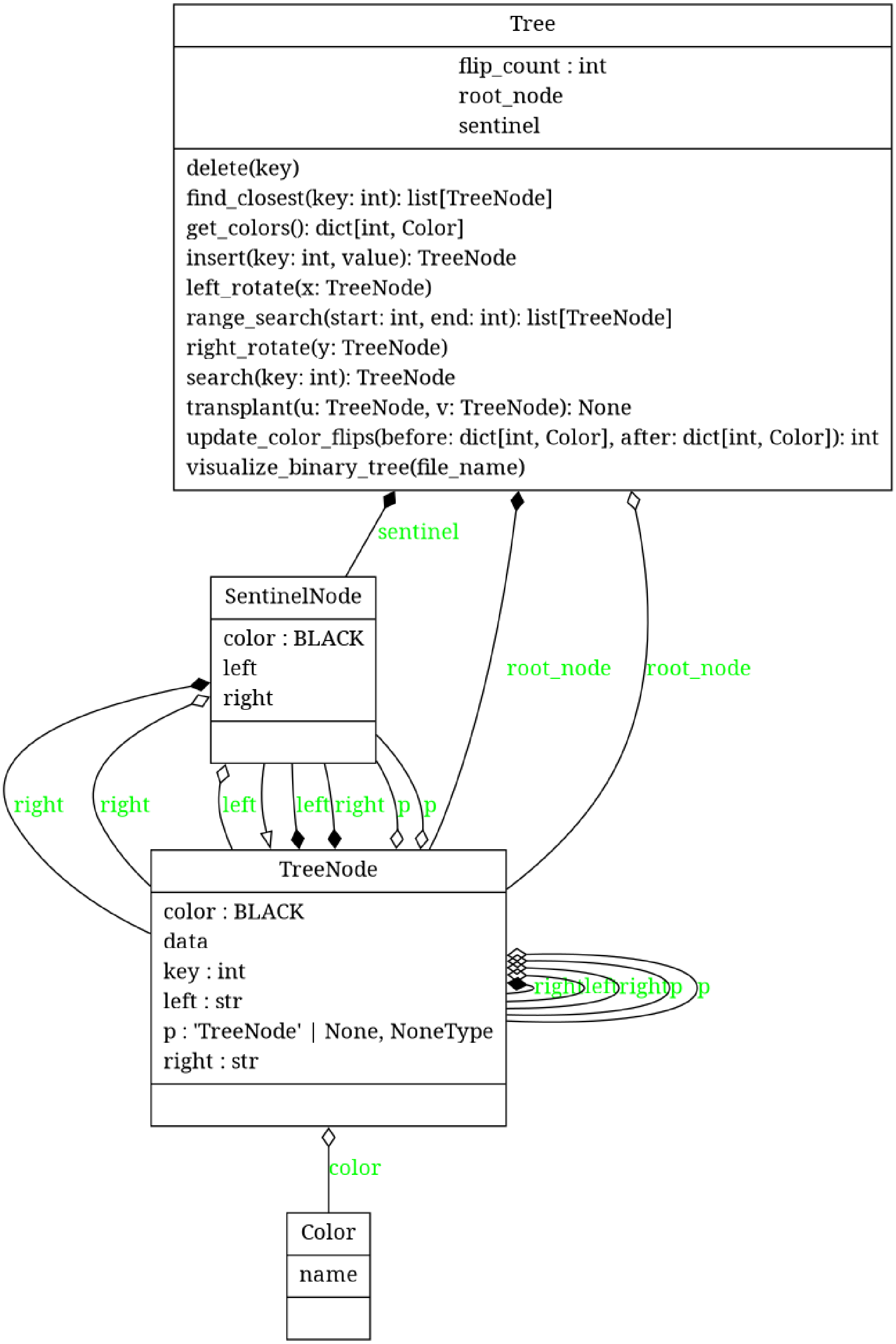
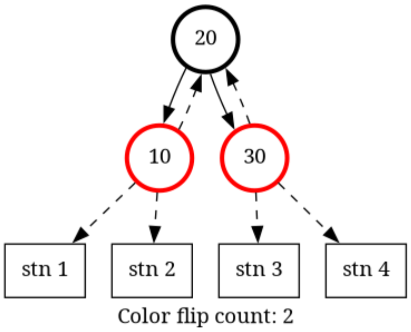
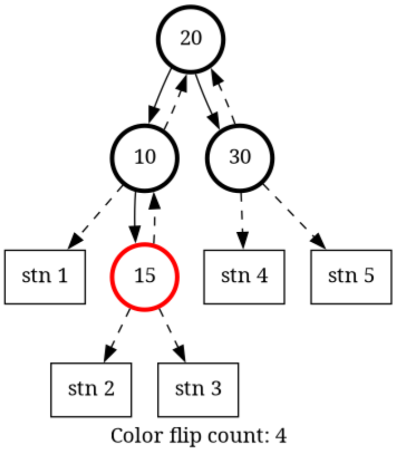
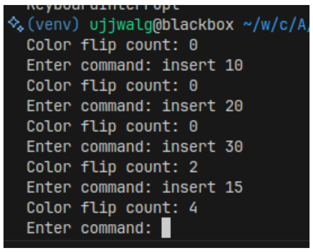

Tree
flip_count : int
root_node
sentinel
delete(key)
find_closest(key: int): list[TreeNode]
get_colors(): dict[int, Color]
insert(key: int, value): TreeNode
left_rotate(x: TreeNode)
range_search(start: int, end: int): list[TreeNode]
right_rotate(y: TreeNode)
search(key: int): TreeNode
transplant(u: TreeNode, v: TreeNode): None
update_color_flips(before: dict[int, Color], after: dict[int, Color]): int
visualize_binary_tree(file_name)
sentinel
SentinelNode
color : BLACK
left
right
root_node
root_node
right
right
left
left
right
p
p
TreeNode
color : BLACK
data
key : int
left : str
p : 'TreeNode' | None, NoneType
right : str
right
left
right
p
p
color
Color
name

Figure 2: Red-Black Tree

**Testing and visualizing the red-black tree** The Red-Black tree can be tested separately using the following command:

```
python3 tree.py
>>> insert 10
>>> insert 20
>>> insert 30
>>> insert 15
```

This will start a REPL that can be used to test the red-black tree and also create two visualizations of the tree – `before.png` and `after.png` . The former shows the tree before an operation and the latter shows the tree after the operation. The REPL also shows the color flip count.

| Before Insertion | After Insertion | REPL |
|---|---|---|
|   20<br>10<br>30<br>stn 1<br>stn 2<br>stn 3<br>stn 4<br>Color flip count: 2 |   20<br>10<br>30<br>stn 1<br>15<br>stn 4<br>stn 5<br>stn 2<br>stn 3<br>Color flip count: 4 |  KeyboardInterrupt<br>(venv) ujjwalg@blackbox ~/w/c/A<br>Color flip count: 0<br>Enter command: insert 10<br>Color flip count: 0<br>Enter command: insert 20<br>Color flip count: 0<br>Enter command: insert 30<br>Color flip count: 2<br>Enter command: insert 15<br>Color flip count: 4<br>Enter command: |

### Binary min-heap

Code for the binary min-heap is implemented in the file `heap.py`. Following is the class diagram for the binary min-heap generated using `pyreverse`:

The heap is designed to store a three-tuple of integers.

These are the methods in the `Heap` class:

- `left`: This method returns the index of the left child of the given index. $O(1)$ time and space complexity.
- `right`: This method returns the index of the right child of the given index. $O(1)$ time and space complexity.
- `parent`: This method returns the index of the parent of the given index. $O(1)$ time and space complexity.
- `min_heapify`: This method maintains the min-heap property. $O(\log n)$ time complexity.
- `push`: This method inserts a new element in the heap. $O(\log n)$ time complexity.
- `pop`: This method removes the minimum element from the heap. $O(\log n)$ time complexity.
- `peek`: This method returns the minimum element from the heap. $O(1)$ time complexity.

**Testing the min-heap** The binary min-heap can be tested separately using the following command:

```
python3 heap.py
>>> push 1
```

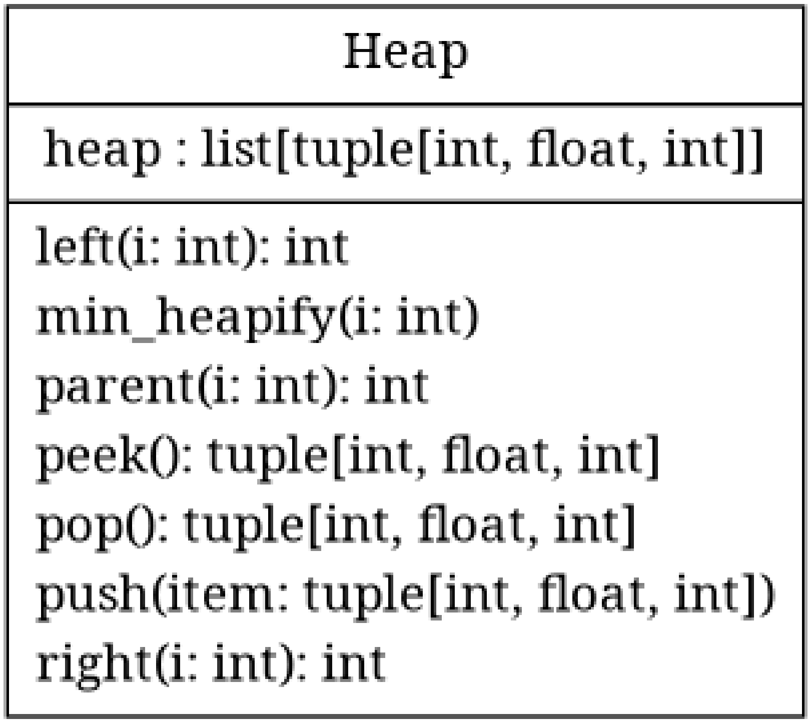

Heap
heap : list[tuple[int, float, int]]
left(i: int): int
min_heapify(i: int)
parent(i: int): int
peek(): tuple[int, float, int]
pop(): tuple[int, float, int]
push(item: tuple[int, float, int])
right(i: int): int

Figure 3: Binary Min-Heap

```
>>> push 2
>>> pop 2
```

**Library management system (`gatorLibrary.py`)**

The library management system is implemented in the file `gatorLibrary.py`. This file uses the red-black tree and the binary min-heap to implement the library management system. Following is a list of functions in the file:

- `PrintBook`: This function prints the details of a book.
- `BorrowBook`: This function assigns a book to a patron
- `InsertBook`: This function inserts a new book in the library and initializes the heap for the book.
- `DeleteBook`: This function deletes a book from the library.
- `ReturnBook`: This function returns a book to the library.
- `FindClosestBook`: This function finds the book with the closest key to the given key using the method `find_closest` of the red-black tree.
- `Quit`: This function quits the program.
- `ColorFlipCount`: This function prints the color flip count of the red-black tree by assessing the `flip_count` varible of the red-black tree.
- `PrintBooks`: This function prints the details of all the books in the library using the `range_search` method of the red-black tree.

This file is also responsible for reading the input file, parsing it and creating the output. Since the input format is compatible with Python's syntax, `eval` function is used to parse the input file.