COP 5536 - Advanced Data Structures
Spring 2023

Programming Project

Name: Vaibhavi Hemant Deshpande
UFID: 9224-1963 | UF Email: deshpande.v@ufl.edu

COMMANDS TO RUN:

The commands for running the project are as follows:

```
make
java gatorTaxi <input_file>
```

PROGRAM STRUCTURE:

The program is implemented using an RBT and a Min Heap.

- Red - Black Tree: A Red - Black Tree is a self-balancing BST. Every node is either colored red or black. The following are the properties of RBT:
    - Every node is colored either - *black, or red*.
    - The root node is always colored *black*.
    - Every leaf node is colored *black*.
    - If a node in the RBT is *red*, then both its child nodes are colored *black*.
    - For every node, all simple paths going from the node to the leaf nodes always contain the same number of nodes that are colored black.

- Min Heap: A min - heap is a full binary tree in which every node's value is less than or equal to that of its child nodes.

The program is divided into the following classes for implementing the gator ride service using the above data structures:

RED - BLACK TREE IMPLEMENTATION

1. **RedBlackTreeNode.java:** The RedBlackTreeNode class defines the node structure for the red - black tree. The class defines the following attributes for the node:
   - **value** – The GatorTaxiRide value associated with the RBT node
   - **parent** – The parent of the RBT node
   - **left** – The left child node of the RBT node
   - **right** – The right child node of the RBT node
   - **color** – The color associated with the RBT node – Black or Red
   - **ptrToMinHeapNode** – The reference to the corresponding node in the min heap

   The class also defines the getter and setter methods associated with the above attributes. Upon Initializing a node, its color defaults to 'red'. It also defines the following methods associated with the node:
   - **public RedBlackTreeNode sibling ()** – Returns the sibling of the node. If the node does not have a parent, it returns null.
   - **public Boolean isLeftChild () –** Returns the value True if the node is left child
   - **public Boolean isRightChild () –** Return value True if the node is the right child

2. **RedBlackTree.java:** The RedBlackTree class implements the red - black tree data structure. The red - black tree orders the nodes based on the *'ride'*. The following attributes are associated with the RedBlackTree:
   - **root** – The root node for the red - black tree

   The class maintains and performs insert, delete, search, and range search operations on the red - black. It also defines the following methods associated with the tree:

   - **public RedBlackTreeNode insert (Ride data)** – This is the wrapper method for insert functionality. This method creates a new RedBlackTree node and inserts the node in the tree using the RBT insert utils.

   - **private RedBlackTreeNode redBlackTreeInsertUtils (RedBlackTreeNode node, RedBlackTreeNode newNode)** – Performs the actual insert operation and inserts the newNode at its correct position. After inserting it handles the Red-Red conflict resolution using the handleRRConflict method to restore the RBT properties. The RBT inserts a node in the worst-case complexity of *O (log n)*.

   - **private void handleRRConflict (RedBlackTreeNode node) –** This method resolves the Red-Red conflict and restores the properties of the red - black tree after insertion whenever there are two consecutive red nodes.

- **public void redBlackTreeDeleteNodeUtils (RedBlackTreeNode node) –** Performs the delete node operation on the red - black tree. It performs delete in accordance with the type of node to delete– if the node is a leaf node, if the node has one child, or if it has 2 children. It handles the double issue using the handleDoubleBlack method. The RBT deletes a node in the worst - case complexity of *O (log n)*.

- **private void handleDoubleBlackConflict (RedBlackTreeNode node)** – This method includes a series of operations to resolve the double - black conflict after executing the delete operation on the red - black tree to rebalance the tree after deleting a node.

- **private void rotateLeft (RedBlackTreeNode node)** – Helper method to perform left rotation to rebalance the red - black tree.

- **public void rotateRight (RedBlackTreeNode node)** – Helper method to perform right rotation to rebalance the red - black tree.

- **private RedBlackTreeNode getInorderSuccessor(RedBlackTreeNode node)** – Helper method to get the inorder successor of a node.

- **public RedBlackTreeNode redBlackTreeSearch (Integer rideNumber)** – Performs search operation on the red - black tree based on ride number. The time complexity of searching for a value in a red - black tree is *O (log n)*.

- **public List <RedBlackTreeNode> rangeSearch (Integer range1, Integer range2)** – Performs a range search based on specified upper and lower bound and returns a list of nodes within the range.

MIN HEAP IMPLEMENTATION

3. **MinHeapNode.java:** The MinHeapNode class defines the node structure for the min heap. The class defines the following attributes for the node and their corresponding getters and setters:
   - **value** – The GatorTaxiRide value associated with the Min Heap node
   - **index** – Index of the Min Heap node
   - **ptrToRBTreeNode**– The reference to the corresponding node in the red - black heap

4. **MinHeap.java:** The MinHeap class implements the min - heap data structure. The priority of the node is decided based on the ride cost, and in case of any conflict, the trip duration property is used for resolution. The following attributes are associated with the MinHeap:
   - **minHeap** – The min - heap data structure
   - **currCapacity** – The current capacity of the min heap
   - **maxCapacity** – The maximum capacity of the min heap

The class maintains and performs insert, get minimum, and delete, operations on the min heap. It also defines the following methods associated with the tree:

- **public MinHeapNode insertNode (Ride ride)** – This method performs an insert node operation on the min heap. The node is inserted at the last and the helper method heapify up is executed to restore the min - heap property**.**

- **private Integer compareNodes (MinHeapNode node1, MinHeapNode node2)** – The comparator function which decides the priority between the two nodes based on the ride cost, and in case of any conflict, the trip duration property is used for resolution.

- **private void swapNodesBasedOnIndex (Integer index1, Integer index2) –** This method resolves the Red-Red conflict and restores the properties of the RBT whenever there are two consecutive red nodes.

- **public MinHeapNode getMinNode () –** Perform extract minimum operation on the min, where minimum or root node is removed, replaced with the last node, and then helper method heapify down is executed to restore the min heap properties. This operation takes the time equivalent to *O (log n),* here *n* represents the heap capacity.

- **public void deleteNode (MinHeapNode node)** – This method resolves the double black conflict after executing the delete operation on the red - black tree.

- **private void heapifyDown (Integer index)** – Heapify down restores the min - heap property by comparing the node to its children's nodes and swapping it with the smallest child if the node is larger than the child. This process is repeated until the node is smaller than its child nodes or there are no child nodes. The time - complexity of the heapify operation is *O (log n)* in the worst case..

- **private void heapifyUp (Integer index)** – Heapify up is used to restore the min - heap property of a node by comparing it to its parent node and swapping them if the parent is greater than the node. This method is repeated until either the parent or the root is reached. The time - complexity of the heapify operation is *O (log n)* in the worst - case.

GATOR TAXI RIDE – MAIN CLASS, INPUT, AND OUTPUT WRITER IMPLEMENTATION:

5. **Ride.java:** The Ride class encapsulated the three ride parameters along with its getter and setter methods.

   - **rideNumber** – The unique value associated with the ride
   - **rideCost** – The cost associated with the ride
   - **tripDuration** – The duration of the trip

The Ride class contains the following methods:

- **public String toString (Ride ride)** – Returns the formatted string for the ride object for writing in the output file overriding the toString method, for instance (1,2,3).

6. **gatorTaxi.java:** The gatorTaxi class maintains both the data structures – min heap and red - black trees and executes the operations associated with the gator taxi ride service. The following attributes are associated with the gatorTaxi class:

- **redBlackTree** – Maintains the red - black tree
- **minHeap** – Maintains the min - heap data structure
- **gatorTaxiOperations** – List of operations to be executed
- **gatorTaxiInputReader** – File input reader object
- **gatorTaxiOutputReader** – File output reader object

The class contains the following methods:

- **private void readAndExecuteOperations (String inputFileName)** – Wrapper method for reading the operations from the input file and executing them.

- **private void readOperations (String inputFileName)** – Gets the list of operations to be executed from the input file.

- **private void executeOperations ()** – Executes the list of operations read from the input file.

- **private void writeOutput (String output)** - Wrapper method for writing output to the file.

- **private void closeWriter ()** – Wrapper class to close the file writer.

- **private String print (Integer rideNumber)** – Searches the rideNumber in the red - black tree and prints the data associated with that rideNumber. The time taken to search a node in an RBT tree is proportional to the RBT's height, which is *O (log n).*

- **private String print (Integer rideNumber1, Integer rideNumber2)** – Returns the rides with ride numbers between the range of rideNumber1 and rideNumber2 from the RBT data structure. The total time taken is *O (log n + s)* where and *s* represents the rides to be printed.

- **private String getNextRide ()** – Returns the next ride based on the minimum ride cost from the min - heap. The found minimum node is then removed from the data structure each with a time complexity of *O (log n).*

- **private Ride insert (Integer rideNumber, Integer rideCost, Integer tripDuration)** – Inserts the new ride in the red - black tree and min heap data structures each with a time complexity of *O (log n)*. If the ride number is duplicate, throw the error "*Duplicate RideNumber*" and terminate the program.

- **private void cancelRide (Integer rideNumber)** – Cancel the ride with the specified ride number and delete the ride from both data structures each with a time complexity of *O (log n)*.

- **private void updateTrip (Integer rideNumber, Integer newTripDuration)** – Update the ride with the new trip duration based on the specified condition and update the associated structures accordingly. Here we perform a search operation on the red - black tree to get the node to update. If the newTripDuration is less than or equal to the old value, update the ride value. If the newTripDuration is between the range of the old value and twice the same, delete the ride and insert it with a penalty in ride cost, and if the newTripDuration is greater than twice the old value, the ride is declined. Here each operation – search, insert and delete from both the data structures take an *O (log n)* time.

7. **GatorTaxiInputReader.java:** Helper class for reading the operations from the input file. It contains the following methods:
    - **public List < GatorTaxiOperation > getOperationListFromInputFile (String inputFileName)** – Returns the list of operations to be executed
    - **private GatorTaxiOperation getOperationFromInputStr (String s)** - Parses the string from the input file and returns the operation to be executed and its associated input parameters.

8. **GatorTaxiOutputWriter.java:** Helper class for writing the result from the output file. It contains the following methods:
    - **private void initOutputFile ()** – initializes the file writer
    - **public void writeOutputToFile (String outputStr) –** Writes outputStr to the output file
    - **public void closeOutputWriter () –** Close the output writer

9. **GatorTaxiOperationCode.java:** Enum representing the operation to be executed. The following operations are enumerated:
    - **Print**
    - **GetNextRide**
    - **Insert**
    - **CancelRide**
    - **UpdateTrip**

10. **GatorTaxiOperation.java:** Class for representing the operation to be executed read from the input file. The following attributes and their getters and setters are associated with this class:

- **gatorTaxiOperationCode** – Enum for the operation
- **input1 –** Input parameter 1 read from the input file
- **input2 –** Input parameter 2 read from the input file
- **input3 –** Input parameter 3 read from the input file

TIME AND SPACE COMPLEXITY ANALYSIS:

RED - BLACK TREE:

Space Complexity: The space complexity of an RBT is based on the *n* in the tree. The class RedBlackTreeNode stores the following properties for each node – value, left, right, parent, color, and its corresponding node reference in the min - heap. Therefore, its space complexity is *O (n)*.

Time Complexity:

Operation	**Function in Code**	**Time Complexity**
Insert	private RedBlackTreeNode redBlackTreeInsertUtils (RedBlackTreeNode node, RedBlackTreeNode newNode)	O (log n)
CancelRide	public void redBlackTreeDeleteNodeUtils (RedBlackTreeNode node)	O (log n)
Print	public RedBlackTreeNode redBlackTreeSearch (Integer rideNumber)	O (log n)
Print	public List <RedBlackTreeNode> rangeSearch (Integer range1, Integer range2) private String print (Integer rideNumber1, Integer rideNumber2)	O (log n + s)
UpdateRide	private void updateTrip (Integer rideNumber, Integer newTripDuration)	O (log n)

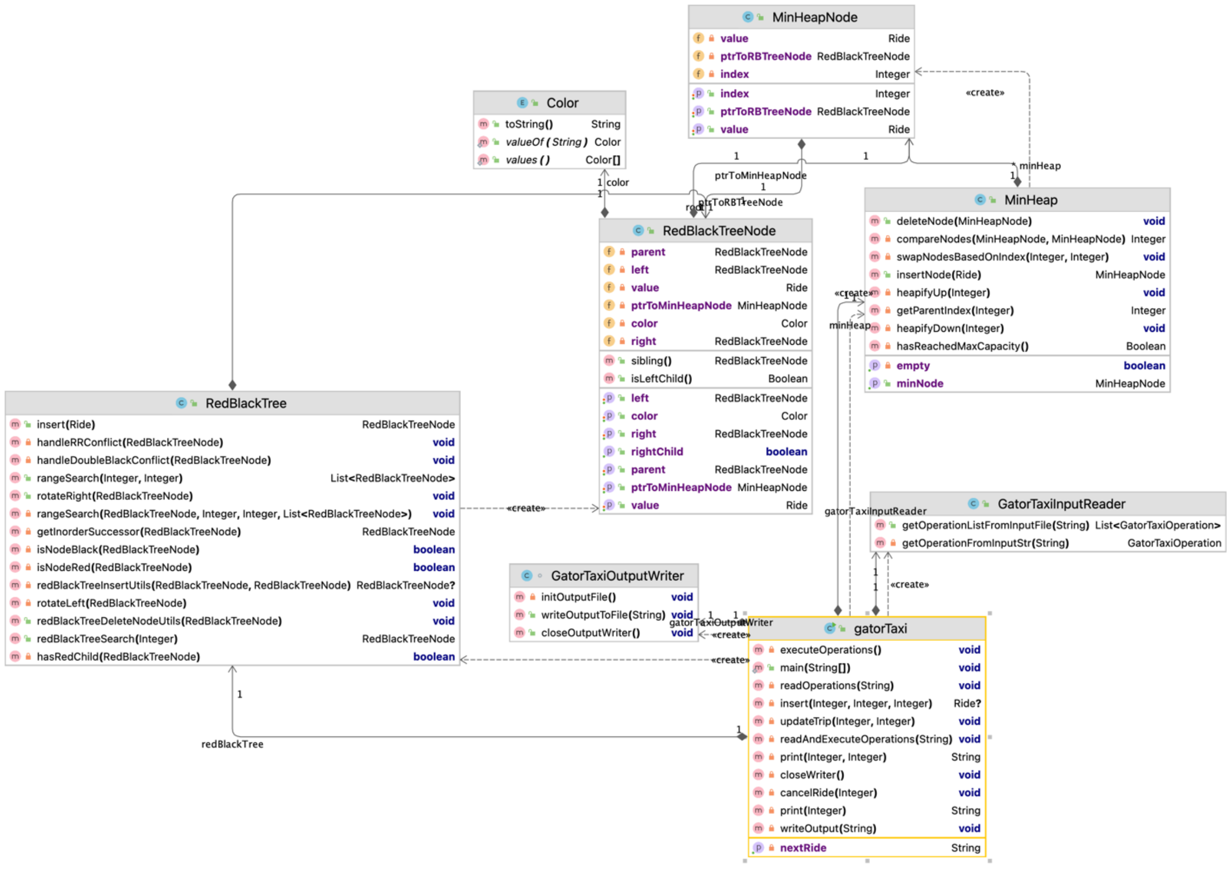
MIN - HEAP:

Space Complexity: The MinHeapNode class represents the node of the min heap and the MinHeap class stores the min - heap in the form of an array. Therefore, the space complexity of the min - heap depends on the number of nodes, making its space complexity equal to *O (n)*.

Time Complexity:

Operation	**Function in Code**	**Time Complexity**
Insert	public MinHeapNode insertNode (Ride ride)	O (log n)
CancelRide	public void deleteNode (MinHeapNode node)	O (log n)
UpdateRide	private void updateTrip (Integer rideNumber, Integer newTripDuration)	O (log n)

CLASS DIAGRAM:



MinHeapNode
value Ride
ptrToRBTreeNode RedBlackTreeNode
index Integer
index Integer
ptrToRBTreeNode RedBlackTreeNode
value Ride
Color
toString() String
valueOf(String) Color
values() Color[]
«create»
minHeap
ptrToMinHeapNode
ptrToRBTreeNode
color
root
MinHeap
deleteNode(MinHeapNode) void
compareNodes(MinHeapNode, MinHeapNode) Integer
swapNodesBasedOnIndex(Integer, Integer) void
insertNode(Ride) MinHeapNode
heapifyUp(Integer) void
getParentIndex(Integer) Integer
heapifyDown(Integer) void
hasReachedMaxCapacity() Boolean
empty boolean
minNode MinHeapNode
RedBlackTreeNode
parent RedBlackTreeNode
left RedBlackTreeNode
value Ride
ptrToMinHeapNode MinHeapNode
color Color
right RedBlackTreeNode
sibling() RedBlackTreeNode
isLeftChild() Boolean
left RedBlackTreeNode
color Color
right RedBlackTreeNode
rightChild boolean
parent RedBlackTreeNode
ptrToMinHeapNode MinHeapNode
value Ride
RedBlackTree
insert(Ride) RedBlackTreeNode
handleRRConflict(RedBlackTreeNode) void
handleDoubleBlackConflict(RedBlackTreeNode) void
rangeSearch(Integer, Integer) List<RedBlackTreeNode>
rotateRight(RedBlackTreeNode) void
rangeSearch(RedBlackTreeNode, Integer, Integer, List<RedBlackTreeNode>) void
getInorderSuccessor(RedBlackTreeNode) RedBlackTreeNode
isNodeBlack(RedBlackTreeNode) boolean
isNodeRed(RedBlackTreeNode) boolean
redBlackTreeInsertUtils(RedBlackTreeNode, RedBlackTreeNode) RedBlackTreeNode?
rotateLeft(RedBlackTreeNode) void
redBlackTreeDeleteNodeUtils(RedBlackTreeNode) void
redBlackTreeSearch(Integer) RedBlackTreeNode
hasRedChild(RedBlackTreeNode) boolean
redBlackTree
GatorTaxiInputReader
getOperationListFromInputFile(String) List<GatorTaxiOperation>
getOperationFromInputStr(String) GatorTaxiOperation
gatorTaxiInputReader
GatorTaxiOutputWriter
initOutputFile() void
writeOutputToFile(String) void
closeOutputWriter() void
gatorTaxiOutputWriter
gatorTaxi
executeOperations() void
main(String[]) void
readOperations(String) void
insert(Integer, Integer, Integer) Ride?
updateTrip(Integer, Integer) void
readAndExecuteOperations(String) void
print(Integer, Integer) String
closeWriter() void
cancelRide(Integer) void
print(Integer) String
writeOutput(String) void
nextRide String