ADS Programming Project

Name: Simran Sunil Kukreja
UFID: 72070368
Email: s.kukreja@ufl.edu

Implementation Details:

- *The Min-Heap data structure* has been implemented to store the ride details i.e., (rideNumber, rideCost, tripDuration) *ordered by the rideCost and tripDuration*(in case two rides have the same rideCost - given that every rideCost-tripDuration combination will be unique).
- *The Red-Black tree data structure* has been implemented to store the ride details i.e., (rideNumber, rideCost, tripDuration) *ordered by the rideNumber*.
- The implementation can handle only *one ride at a time*, for a *maximum of 2000 active ride requests*.
- *Operations supported* by the implementation and their time complexities:
  (n = number of active rides, S = number of ride triplets printed)

Print(rideNumber)	Searches the Red-Black Tree for the given ride and prints it if the ride is found. [*O(log(n))*]
Print(rideNumber1, rideNumber2)	Searches the Red-Black Tree for rides in the given range and prints them if any ride is found. [*O(log(n)+S)* time]
Insert(rideNumber,rideCost,tripDuration)	Inserts a ride with the given ride triplet values in the min-heap and the Red-Black tree, each having pointers to the other. [*O(log(n))* time]
GetNextRide()	Returns the ride at the root of the min-heap if the heap is not empty and deletes it from the min-heap and the red-black tree. [*O(log(n))* time]
CancelRide(rideNumber)	Deletes the ride having given rideNumber from the min-heap and red-black tree. [*O(log(n))* time]
UpdateTrip(rideNumber, new_tripDuration)	1. new_tripDuration<=existing tripDuration – Deletes the existing ride and inserts a new ride in the min-heap and red-black tree with updated cost, and links them

	2. existing_tripDuration<new_tripDuration<=2*(existing tripDuration) - Deletes the existing ride and inserts a new ride in the min-heap and red-black tree with updated cost as (currentCost+10), and links them 3. new_tripDuration > 2*(existing tripDuration) - Deletes the existing ride from the min-heap and red-black tree with - Deletes and inserts are performed in *O(log(n))* time.

- Implementation is in Java

Project directory structure:

gatorTaxi.java	*Driver code for the implementation that calls the min-heap and red-black tree functions as per the input.*
MinHeap.java	*Builds a min-heap data structure and includes functions for search, insert, delete min, delete key, and minheapify on the min-heap.*
HeapNode.java	*Data Structure to store a min-heap node with ride details and a pointer to the TreeNode of the red-black tree.*
RedBlackTree.java	*Builds a red-black tree data structure and includes functions for search, insert, delete, and corresponding red-black tree rotations and transformations.*
TreeNode.java	*Data Structure to store a red-black tree node with ride details and a pointer to the HeapNode of the min-heap.*
output_file.txt	*Output file generated by the program.*
Makefile	

Time and Space Complexity:

Min-Heap Operation	**Time Complexity**	**Space Complexity**
Insert	O(log(n), where log(n) is the number of levels in the heap we will traverse at max in the worst case to insert a node	O(1), no additional space required

Delete Min	O(1), since we are deleting the root directly	O(1), no additional space required
Delete key	O(log(n), where log(n) is the number of levels in the heap we will traverse at max in the worst case to delete a node	O(1), no additional space required
MinHeapify	O(log(n), where log(n) is the number of levels in the heap we will traverse at max in the worst case to balance the heap	O(1), no additional space required
Swap	O(1) to swap the min-heap nodes	O(1), no additional space required

Red-Black Tree Operation	**Time Complexity**	**Space Complexity**
Insert [insert, fixInsert functions]	O(log(n)), where log(n) is the height of the balanced search tree	O(1), no additional space required
Delete [delete using deleteHelper, deleteFixup functions]	O(log(n)), where log(n) is the height of the balanced search tree	O(1), no additional space required
Search [search using searchHelper function]	O(log(n)), where log(n) is the height of the balanced search tree	O(1), no additional space required
leftRotate	O(1) to perform node rotations, there is no recursion	O(1), no additional space required
rightRotate	O(1) to perform node rotations, there is no recursion	O(1), no additional space required
rbTransform	O(1) to perform tree transofrm, there is no recursion	O(1), no additional space required
getRoot	O(1) to just return the root value	O(1), no additional space required

Function prototypes and program structure:

1. gatorTaxi : Driver class

Functions:
'public static void main (String args [])' method:

- It takes the input file name as a command line argument, processes the input as per the 6 different operations listed above, and writes the output to the 'output_file.txt' file.
- For performing the operations, the main method calls the respective 'MinHeap' and 'RedBlackTree' class functions.

2. HeapNode: Class defining the Node Structure of the Min-Heap

Constructor:
public HeapNode(int rideNumber, int rideCost, int tripDuration)

- Initializes the heap node with the ride triplet details when a new ride instance is created for the min-heap.
- Stores a pointer to a TreeNode of the red-black tree to maintain a link from the min-heap to the red-black tree.

3. TreeNode: Class defining the Node Structure of the Red-Black Tree

Constructor:
public TreeNode(int rideNumber, int rideCost, int tripDuration, HeapNode node)

- Initializes the red-black tree node with the ride triplet details when a new ride instance is created for the red-black tree.
- Stores a pointer to a HeapNode of the min-heap to maintain a link from the red-black tree to the min-heap.

4. MinHeap: Class implementing the Min-Heap Data Structure

Constructor:
public MinHeap(int size)

- Initializes a 'HeapNode' array with a maximum size of 2000(as given in the project description)

Functions:
private int parent(int i)

- Returns the index of the parent node for the node at index i in the HeapNode array

private int leftChild(int i)

- Returns the index of the left child node for the node at index i in the HeapNode array

private int rightChild(int i)

- Returns the index of the right child node for the node at index i in the HeapNode array

private boolean isLeaf(int i)

- Returns true if the node at index i in the HeapNode array is a leaf node, i.e., its right and left child are null.

private HeapNode insert(HeapNode element)

- Inserts ride in the min-heap based on rideCost comparisons, with the minimum rideCost ride being at the root of the min-heap.
- In case of the same ride costs of two rides, the ride having a smaller trip duration is put at the top of the min-heap.
- Returns the inserted HeapNode to set the corresponding red-black tree pointer in it in the driver class 'gatorTaxi'

public HeapNode remove()

- Remove the min-heap's root and replaces it with the last element in the min-heap.
- Performs minheapify operation(described below) to ensure it is a min-heap after the removal and replacement.
- Returns the root of the min-heap to the driver class to print it out.

public HeapNode deleteKey(int i)

- Deletes a specific ride having a certain rideNumber from the min-heap.
- Uses the 'idx' property in the HeapNode to delete the specific ride.
- Calls the decreaseKey function(described below) and remove() to delete the bubbled-up ride from the min-heap.
- Returns the deleted node copy to the driver class to delete the corresponding red-black tree node.

private void minHeapify(int i)

- Compares and swaps non-leaf node at index i with the minimum ride cost left or right child node, if the ride cost of any of the children is lesser.
- In case the node has the same cost as its left/right child, then the trip duration of the two nodes is compared.
- If the trip duration of the child node is lesser than the node at index i then the two nodes are swapped

private void decreaseKey(int i, HeapNode new_val)

- Bubbles up the ride having the idx value i to the top of the min-heap by replacing the node with the 'new_val' that has 'Integer.MIN_VALUE' rideCost.

private void swap(int x, int y)

- Swaps the two heap nodes present at indices x and y of the HeapNode array

5. Red-Black Tree: Class implementing the Red-Black Tree Data Structure

Constructor:
public RedBlackTree()

- Initializes the RedBlackTree instance with a 'TreeNode' root with TNULL i.e., a TreeNode(0,0,0,null), color=black and null left and right children.

Functions:
private TreeNode searchHelper(TreeNode node, int rideNumber)

- Checks if the rideNumber matches the current root node, if not, it recursively traverses the left/right subtree depending on if the rideNumber is lesser or greater than the current node value.
- Returns the TreeNode found for the given rideNumber or returns null if no TreeNode is found to the calling search method

private void deleteFixup(TreeNode x)

- Balances the red-black tree after the deletion of a ride from the tree considering the various red-black tree constraints.

private void rbTransform(TreeNode node1, TreeNode node2)

- Transform the red-black tree to satisfy the balanced red-black tree constraints.

private void deleteHelper(TreeNode node, int rideNumber)

- Helper method to delete the ride for the given rideNumber from the red-black tree.

private void insertFixup(TreeNode k)

- Balances the red-black tree after the insertion of a ride into the tree taking into account the various red-black tree constraints.

public search(int rideNumber)

- Calls a searchHelper function to recursively search for the given rideNumber in the tree
- Returns the TreeNode found for the given rideNumber or returns null if no TreeNode is found to the driver class**s**

public TreeNode minimum(TreeNode node)

- Returns the minimum node linked to the passed node i.e., the left node

public void findNodesInRange(TreeNode curr, int rideNumber1, int rideNumber2)

- Recursively searches for nodes in the given ride number range.
- Writes all the nodes that fall in thxe range to the output file.

public void leftRotate(TreeNode x)

- Performs a left rotation to balance the red-black tree after insertions and deletions.

public void rightRotate(TreeNode x)

- Performs a right rotation to balance the red-black tree after insertions and deletions.

public TreeNode insert(int rideNumber, int rideCost, int tripDuration, HeapNode heapNode)

- Inserts a new TreeNode into the red-black tree with the given ride triplet details and the pointer to the corresponding HeapNode in the min-heap.
- Returns the inserted TreeNode to the driver class to set it in the pointer of the corresponding HeapNode in the min-heap.

public TreeNode getRoot()

- Returns the root node in the red-black tree to the driver class.

public void deleteNode(int rideNumber)

- Calls the deleteHelper method to delete the node having the given rideNumber from the red-black tree.