COP 5536 Fall 2019

Programming Project

Kalpak Seal
UFID: 8241 – 7219
kalpak.seal@ufl.edu

Problem description

Wayne Enterprises is developing a new city. They are constructing many buildings and plan to use software to keep track of all buildings under construction in this new city. A building record has the following fields:

- **buildingNum**: unique integer identifier for each building.
- **executed_time**: total number of days spent so far on this building.
- **total_time**: the total number of days needed to complete the construction of the building.

The needed operations are:

1. Print (buildingNum) prints the triplet buildingNum, executed_time, total_time.
2. Print (buildingNum1, buildingNum2) prints all triplets bn, executed_time, total_time for which buildingNum1 <= bn <= buildingNum2.
3. Insert (buildingNum, total_time) where buildingNum is different from existing building numbers and executed_time = 0.

Wayne Construction works on one building at a time. When it is time to select a building to work on, the building with the lowest executed_time (ties are broken by selecting the building with the lowest buildingNum) is selected.
The selected building is worked on until complete or for 5 days, whichever happens first. If the building completes during this period its number and day of completion is output and it is removed from the data structures. Otherwise, the building's executed_time is updated. In both cases, Wayne Construction selects the next building to work on using the selection rule. When no building remains, the completion date of the new city is output.

Implementation

• **MinHeap**: We store the 'buildings' ordered by Executed Time, so building with the minimum executed time will be at the top. If there are two buildings with the same executed time, the building with the lower building number gets the preference.

In Minheap, we get an O(log n) time for Insertion, Deletion and Remove Min operation.

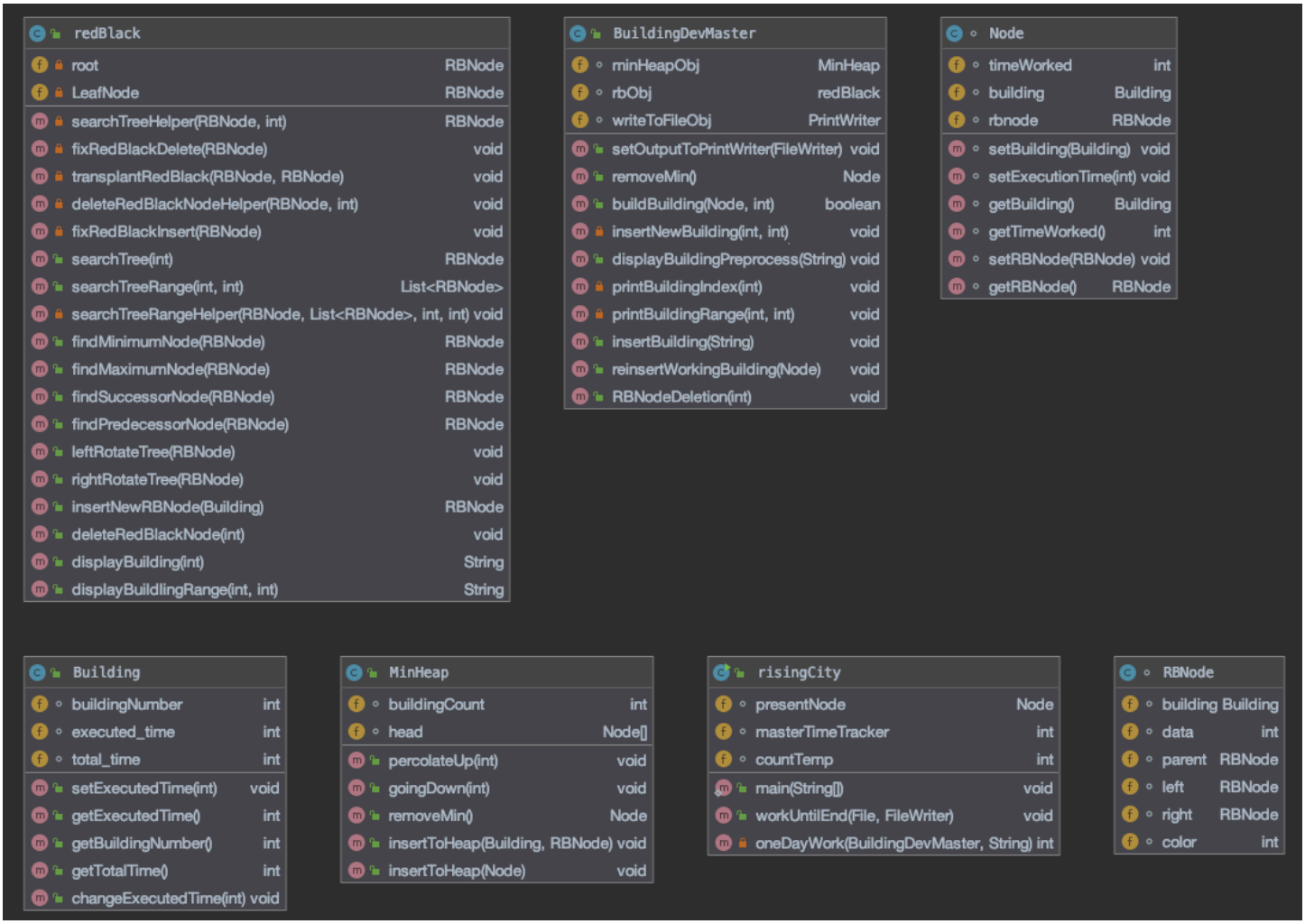
• **Red Black Tree**: A Red Black Tree is a Balanced Binary Search Tree with a color constraint. Due to faster search queries, a height balanced BST has been chosen thereby. Also, deletion is faster for a Red black tree as opposed to an AVL tree. Therefore, an RB tree justifies is usage.

Execution

Steps to execute the project:

- Unzip the project
- Paste the input file into the project folder.
- make
- java risingCity <InputFile.txt>
- View the output in output.txt

Project Structure



- **Building –** Holds the details of a building, namely: buildingNumber, executed_time and total_time.
- **BuildingDevMaster** – This is the main operator for the project. Class ***risingCity*** calls this and BuildingDevMaster decides what action to perform, based on the input. It performs the job of

preprocessing the input command, take the necessary decisions of updating the MinHeap and RedBlackTree based on computation, and thereby write to the file.

- **risingCity** – This is where the execution starts. This class invokes the ***workUntilEnd*** method, which basically gets executed until all the buildings are constructed.
  The ***workUntilEnd*** method reads each line of input from the input file and performs the job by calling ***oneDayWork.*** Initially we start working on a building for its given duration, or for 5 days, whichever is less.
  After that, we push building, we were working on, into the minHeap and further process the input and perform another job based on the input.
  - If there's a print request: we print the building details from the Red Black Tree.
  - If there's an insert request: we insert the building into the Red Black Tree, take the same object and bundle into a minHeap node and insert into the MinHeap.

- **Minheap** - This class implements a Minheap. It has another class ***Node*** which makes minheap Node array. Class Node's ***timeWorked*** is the number of executed days of the Building.
  The Node also points to the corresponding Red Black Tree Node of the same building which helps facilitate deletion easier.

- **redBlack -** This class implements a class named RBNode which makes the nodes of a Red black tree. The redBlack tree holds the building data arranged on the basis of building number and performs basic operations like insertion, deletion and search.

Project Documentation

1. Class Name – Building

a. Member Variables

- int buildingNumber - Number of the building, i.e. building id
- int executed_time - Amount of time spent working on it
- int total_time - Total time required to construct the building

b. Methods

Return Type	Function Name	Purpose
void	setExecutedTime	Setter method to set the amount of work done on the building
int	getExecutedTime	Getter methods to receive the respective member variables.
	getBuildingNumber	
	getTotalTime	

2. Class Name – risingCity

a. Member Variables

- Node presentNode - present working building node
- int masterTimeTracker - tracks the global time
- int countTemp - tracks the no. of days worked

b. Methods

Return Type	Function Name	Purpose
void	main(String args[])	Starting point of the program. Reads the input file, Writes to the output file and invokes the workUntilEnd method to perform the job.
void	workUntilEnd(File fileInput, FileWriter outputFileWriterObj)	Takes the file input and works until its done constructing all the buildings
int	oneDayWork(BuildingDevMaster constructCitySimulate, String displayStr)	Invoked by workUntilEnd and performs the job based on the problem description for a particular day.

3. Class Name – BuildingDevMaster

a. Member Variables

- MinHeap minHeapObj - object of MinHeap class
- redBlack rbObj - object of RedBlack Tree class
- PrintWriter writeToFileObj - PrintWriter object to write the output to the file.

b. Methods

Return Type	Function Name	Purpose
void	setOutputToPrintWriter (FileWriter outputToPrintWriter)	Creates the PrintWriter Object to write to the output file.
Node	getMinOfMinHeap ()	Calls the removeMin function of MinHeap and returns the corresponding Node.
boolean	buildBuilding (Node currentBuildingNode)	Increments the execution time of the building and return true if the execution time matches the total time required to construct the building, and false otherwise,
void	displayBuildingPreprocess (String output)	Writes to the output file.

void	insertBuilding (String string)	Pre-processes the string from the input file and inserts the building into the minHeap and the Red Black Tree.
void	reinsertWorkingBuilding (Node insertNode)	If the current building on which work is being done is still incomplete, re-insert it into the minHeap.
void	RBNodeDeletion (int buildingNum)	Delete the building from the Red Black Tree.

4. Class Name – Node

a. Member Variables

- timeWorked – Amount of time spent working on the building
- building – Building object
- rbNode - Red Black Tree object to maintain a pointer from the MinHeap.

b. Methods

Return Type	Function Name	Purpose
void	setTimeWorked (int key)	Setter method for timeWorked.
Building	getBuilding ()	Gets the building object.
int	getTimeWorked ()	Gets the amount of time worked.

5. Class Name – MinHeap

a. Member Variables

- buildingCount – Used in the MinHeap operation
- head [] - Array which stores the Nodes based on minHeap computation.

b. Methods

Return Type	Function Name	Purpose
void	percolateUp (int idxChild)	Reinstates the heap property after insertion
void	goingDown (int idxChild)	Reinstates the heap property after removeMin operation
Node	removeMin ()	Removes the min node from the heap
void	insertToHeap (Node building)	Inserts the building into the heap that still has work to be done
void	insertToHeap (Building b, RBNode newRBNode)	Inserts the node after creating the node object to the minHeap.

6. Class Name – RBNode

a. Member Variables

- Building building - building object
- int data - holds the building id
- RBNode parent - this is pointing to the parent
- RBNode left - this is pointing to left child
- RBNode right - this is pointing to right child
- int color - This is 1 for Red and 0 for Black

7. Class Name – RedBlack

a. Member Variables

- root - root node of Red Black Tree
- LeafNode - dummy leaf node

b. Methods

Return Type	Function Name	Purpose
RBNode	searchTreeHelper (RBNode node, int token)	Search the Node that contains the token.
void	fixRedBlackDelete (RBNode rbNode)	Restores the balance of the RBTree after deletion.
void	deleteRedBlackNodeHelper (RBNode nodeToDelete, int token)	This is the helper class for red black tree delete operation.
void	fixRedBlackInsert (RBNode nodeX)	Restores the balance of the RBTree after insertion.
RBNode	searchTree(int k)	Search the RedBlack tree for the given value.
List<RBNode>	searchTreeRange (int low, int high)	Search the Tree for a particular range.
void	searchTreeRangeHelper (RBNode node, List<RBNode> result, int low, int high)	Perform an inorder traversal and returns the nodes in the given range.
RBNode	findMinimumNode (RBNode node)	Find the minimum RBNode.
void	leftRotateTree (RBNode nodeA)	Rotate Left the RB Tree.

void	rightRotateTree (RBNode nodeA)	Rotate Right the RB Tree.
RBNode	insertNewRBNode (Building b)	This function gets called from the BuildingDevMaster class to insert a new building into the RB Tree.
void	deleteRedBlackNode (int data)	Called from BuildingDevMaster class to delete the node.
String	displayBuilding (int buildingNum)	Returns the details of the building asked.
String	displayBuildlingRange (int low, int high)	Returns the details of the building by searching the RB Tree for the asked range.

Reference

Introduction to Algorithms – CLRS

Geekforgeeks.com

Algorithms – Robert Sedgewick