COP 5536 Fall 2019

Programming Project Report

Name: Yihui Wang
UFID# 8316 4355
E-mail: yihui.wang@ufl.edu

Project Description

In this project, a Java program is implemented to help Wayne Enterprises to keep track of buildings under construction in a new city. Each building record inserted to system has three fields: buildingNum, executed_time and total_time. A Red-Black Tree (RBT) is implemented to print the record with certain building number, and a Min-heap is implemented to pick the building with the lowest executed time to work on. When a building's executed time equals total time, the building number and the completed date will be printed and the record of it will be removed from the data structures.

Executing Instruction

To execute the program, first, compile the code by using the command "make" in the terminal. Then, enter the command "$ java risingCity file_name". The file_name is the name of the file with input data. The output will be printed in the output_file.txt in the root directory of the program.

Structure of Program and Function Description

There are six classes in the program. The RedBlackTree class and the Node class are used to implement the Red-Black Tree. The MinHeap class and the Triplet class are used to implement the Min-heap. The risingCity class and the Input class and are used to implement to main function of the program.

Triplet.java

This class describes the triplet structure of the record of each building. There are three variables in this class.

int buildingNum: the unique ID of each building.
int executed_time: the days have been worked on of each building.
int total_time: the total days that need to work on of each building.

Node.java

This class describes the node structure used in the Red-Black Tree.
There are six variables in this class.

Node left: the left child of the node.
Node right: the right child of the node.
Node parent: the parent of the node.
boolean color: the color of the node, true represents black, and false represents red.
Triplet triplet: the building record of the node.
boolean isnull: whether the node becomes a null node after delete operation.

MinHeap.java

This class describes the structure of Min-heap implemented in the program. The Min-heap is implemented by array. There are two variables and seven functions in this class.

Triplet[] heap: the array used to implement the Min-heap.
int size: the size of the Min-heap.

Triplet removeMin():
Get the min element out of the Min-heap.
Return: the min element(root) of the Min-heap. Null if the heap is empty.

void swap(int x, int y):
Swap the position of two elements in the Min-heap.
Parameters: **int x** and **int y** are the index of the element in the array.

void minHeapify(int x):
Heapify the heap when the heap is not satisfied as a Min-heap according to the executed_time of each element. When two elements have the same executed_time, the element with smaller buildingNum become the smaller element in the Min-heap.
Parameters: **int x** is the index of the element in the array that need to be heapify.

void add(Triplet x):
Add new element to the Min-heap.
Parameters: **Triplet x** is the record of the building that need to be added to the Min-heap.

int parent(int x):
Get the parent element of a certain element.
Parameters: **int x** is index of the current element in the array.
Return: the index of the parent of the input element.

int left(int x):
Get the left child element of a certain element.
Parameters: **int x** is index of the current element in the array.
Return: the index of the left child of the input element.

int right(int x):
Get the right child element of a certain element.
Parameters: int x is index of the current element in the array.
Return: the index of the right child of the input element.

RedBlackTree.java

This class describes the structure of Red-Black Tree implemented in the program. There are one variable and seven functions in this class.

Node root: the root of the Red-Black Tree.

Node lookUpNode(Node node):
Look up a certain node in the Red-Black Tree.
Parameters: **Node node** is the node with the same buildingNum of the node that need to be looked up.
Return: the node found or null if the node is not found in the Red-Black Tree.

void addNode(Node node):
Add a node into the Red-Black Tree.
Parameters: **Node node** is node that need to be added to the Red-Black Tree.

void insertFix(Node node):
Rebalance the Red-Black Tree after inserting a new node. There are five different situations to handle when rebalance the tree: XYr, RRb, LLb, LRb and RLb.
Parameters: **Node node** is node that have been added to the Red-Black Tree.

void deleteNode(Node node, boolean isrecursion, boolean left):
Delete a node from the Red-Black Tree. A null node will be added to the tree when the deleted node is a black node, which will be removed after rebalancing.
Parameters: **Node node** is the node that need to be deleted. **boolean isrecursion** is to note whether this executing is a part of a recursion which may happen when a black non-leaf node is deleted. **boolean left** is to note whether the null node added to the tree is the left child of its parent in a recursion.

void deleteFix(Node node):
Rebalance the Red-Black Tree after deleting a node. There are eighteen different situations to handle when rebalance the tree: Rb0case1, Rb0case2, Rb1case1, Rb1case2, Rb2, Rr0, Rr1case1, Rr1case2, Rr2, Lb0case1, Lb0case2, Lb1case1, Lb1case2, Lb2, Lr0, Lr1case1, Lr1case2 and Lr2.
Parameters: **Node node** is the y node of the rebalancing strategy described in the lecture.

void LL(Node pp, Node gp):
Perform a LL rotation.

Parameters: **Node pp** and **Node gp** is the pp node and gp node of the rotation strategy described in the lecture.

void RR(Node pp, Node gp):
Perform a RR rotation.
Parameters: **Node pp** and **Node gp** is the pp node and gp node of the rotation strategy described in the lecture.

String midOrder(Node node, int low, int high):
Perform a middle order traversal in a certain range.
Parameters: **Node node** is the root of the tree. **int low** and **int high** is the range of the middle order traversal: high <= buldingNum<= low.
Return: a String contains the record of buildings.

Input.java
This class describes the structure of Input command in the program. All the input commands are split into four parts: day, command, num1, and num2. There are four variables in this class.

int day: the date when the input command is executed.
String command: the type of command, such as Insert and PrintBuilding.
int num1: the first number in the command
int num2: the second number in the command. 0 when there is no second number in the command.

risingCity.java
This class describes the main function of the program. There are two functions in the class.

void main(String[] args):
the main function of the program.
Commands are stored in a Input[] list. A global counter counting the days. The program works on one building at a time, which is the min from the Min-heap. The building's executed_time is updated when being picked. When the global counter equals the input command date, the command will be executed. The picked 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 to the output file and it is removed from Red-Black Tree. Otherwise, it will be added to the Min-heap again and a new building will be picked to work on.
Parameters: **String[] args** is the args from the command line, which is the file name of the input file.

void writeFile(String str):
print all the output into the output file.
Parameters: **String str** is the String that contains the outputs need to be printed into the output file.

Conclusion

The Min-heap and Red-Black Tree is well implemented according to what professor described in the lecture. All the functions in the program are well tested, and the output is the same as the output file provided on Canvas when using the provided test cases.