Advance Data Structures
Fall 2019 | Programming Assignment

Ghodkari Chowdary, Raghunatha Rao raghunat.ghodkar@ufl.edu| UFID: 6218-1051

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.

In order to complete the given task, you must use a min-heap and a Red-Black Tree (RBT).

Algorithm and Implementation

```
globalDayCount = 0

while ( true ){
        read input from file
        if globalDayCount != 0 {
                Do building construction on current building
        }
        // parse input to get the currentDayCount
        If globalDayCount == currentDayCount{
                If action is insert
                        Insertion into minheap and Redblacktree
                If action is printBuilding
                        Corresponding print is performed by searching in redblacktree
        }
        globalDayCount++;
        }
complete construction for remaining buildings
```

Explanation of the Understanding

- globalDayCount is initially zero and operations are read from input file line by line
- if globalDayCount is not equal to zero, current building construction is started from day 1
- In construction we extract root node by removeMin operation and store in risingCity object then we increase the executed time along with the global time
- If after 5 consecutive days construction is building is not complete it is reinserted to minheap else print to file and current building is made null
- Now currentDayCount is parsed from the line read from file

- If currentDayCount equals to globalDayCount and operation is insert the building is inserted into minheap, heapify is performed and also inserted into redblacktree
- If operation is printbuilding the buildings are print by searching from redblack tree

Algorithm Implementation:

Min Heap:

Min Heap is used to store buildings which always returns the building with minimum executed time and when executed times are same building with lower building number is selected. This helps in the construction of the city.

Operation the minheap supports:

insert(): A new building is added to array and a heapify is performed after each insert to maintain the min heap property.

removeMin() : the first element in min heap the root which is minimum is returned after replacing it with the last element and heapify is performed again to maintain the heap property.

RedBlackTree:

RedBlack Tree is used to store buildings which are inserted in the min heap. They support actions printBuilding where tree helps in fast retrieval of the output.

Operations on RedBacktree

insert(): A new building is added to redblack tree for every insert action

delete(): A node is deleted with corresponding building from the redBlack tree.

Understanding and Implementation:

1. A counter globalDayCount is maintained to track the global time
2. Each line is read from file
3. Construction is done on current building for every globalDayCount for 5 days or its completion whichever is earlier
4. Another currentDayCount is parse from each operation from line read from file
5. If globalDayCount is equal to currentDayCount the action in the line is performed
6. Now increase the globalDayCounter and repeat from step 2

Program Structure

The entire project has 5 files risingCity.java, minheap.java, RedBlackNode.java and RedBlackTree.java and Building.java

Building.java
This class describes the structure of the Building defined in the program

private int buildingNum;
// stores the Buidling number
private int executed_time;
// stores the executed time
private int total_time;
// stores the total time
public int workingDays = 0;
// stores the consecutive workingdays, used in the risingCity.java
public RedBlackNode refNodeinTree;
// reference to corresponding node in redback tree which helps to delete node rather than searching for node in the red black tree

The class contains constructors to initialize objects and also getters and setters methods for fields of Building.

RedBlackNode.java
This class defined the structure of node in the redblacktree

private Building bld;
// stores the building
private RedBlackNode left;
// link to left child of node
private RedBlackNode right;
// link to right child of node
private RedBlackNode parent;
// link to parent of the node
private char color;
// color of the node

The class contains constructors and required getters and setters for fields of the red black node

RedBlackTree.java
This class defined the structure of redblacktree and operations on it

```
// External Null Node
private RedBlackNode ext_null_node = new RedBlackNode();
// Root of the RedBlack Tree
private RedBlackNode root = ext_null_node;

public void insert(Building bld)
// insert operation on the redblack tree

public void checkRRConflicts(RedBlackNode node)
//to resolve red red adjacent conflict due to insertion

public void left_Rotate(RedBlackNode x)
//performs left rotate operation around the node

Public void right_Rotate(RedBlackNode x)
// performs right rotate operation around the node

public void removeBuilding(Building bld)
// removes building from redblack tree

public void delete( RedBlacknode node )
//remove node from the redblack tree

public RedBlackNode findminRightSubTree( RedBlackNode root)
//get the minimum node in the tree

public void checkDoubleBlackConditions(RedBlackNode node)
// function purpose is to balance the black nodes after deleting a black node in redblacktree

public void rb_transplant(RedBlackNode u, RedBlackNode v)
//replace subtree at node u with subtree at node v

public String printBuilding(int bld)
// print the building on file using building number by search on redblack tree

private String search( RedBlackNode node, int bld )
// search the node with building number bld in the redblack tree

public String printBuilding( int range1, int range2 )
// print the buildings in the redblack tree whose building numbers are in range 1 and range2
```

minHeap.java
This class defines the minHeap its fields and various operations on it
private Building[] Heap;
// an array of buildings in the heap
private int size;
// current size of the size
private int maxsize;
// maxsize that a heap can store

There are setters and getters of the fields in the minHeap java file

public minHeap(int maxsize)
// constructors to initialize the heap

private int parent(int pos)
// return the parent index of pos

private int leftChild(int pos)
// return the leftchild index of pos

private int rightChild(int pos)
// return the rightchild index of pos

public void insert(Building new_building)
//insert the new building into min heap

private void swap(int fpos, int spos)
//swap the buildings located at positions fpos and spos

public Building removeMin()
// remove operation on minHeap

private void minHeapify(int pos)
//heapify operation to maintain the heap property

risingCity.java

this class is the main java file where the program executes

```
private static final String OUTPUT = "output_file.txt";
// the final output file
private RedBlackTree rootRBT = null;
// the redblacktree root reference
private minHeap minHeapBld = null;
// the minHeap reference
Set<Integer> insertedBldNums = new HashSet<Integer>();
// set containing the processed building numbers to return error for duplicate insertion
Building currBld = null;
// the current building whose construction is done

public RisingCity()
// constructor that create empty redblack tree and empty minheap of size 2000

public void simulate(String input)
// starts the program at this method

private void simulateConstruction(File input_file, File output_file)
// starts the construction of the buildings at this method

private void checkInputOutput( File input_file, File output_file )
//function to validate the input and output files

private void executeBuilding( BufferedWriter bw, int globalDayCount )
// execute the current building at globalDayCount

private String executeRemainingBuildings( BufferedWriter bw, int globalDayCount )
// execute the remaining buildings in heap after reading the entire file

private String insertBuilding( String str )
// method used to insert building into minheap and redblack tree

private String printBuilding( String str, BufferedWriter bw )
//method to perform printBuilding action
```

Assumptions

- Construction is done from Day 1
- After Construction of current Building on specific day actions are performed if the global day is equal to local day
- After each insert over minheap heapify performed to maintain heap consistent
- After each removeMin operation over minheap last node is replaced with the first one and size is decreased by one, then heapify is performed

Conclusion

The project has been implemented successfully and tested on the sample input test cases, due to difference in understanding of the problem statement the results are different.