COP5536 Advanced Data Structures(Fall 2019)

Venkata Sindhu Kandula (UF ID: 1914 5414)

v.kandula@ufl.edu

# Project

Expected to create a software to successfully compute three operations Insert(buildingNum,totalTime), PrintBuilding(buildingNum1,BuildingNum2) and printBuilding(buildingNum) to track all the buildings constructed by Wayne Enterprises in the new city. Each Building record has buildingNum which is unique for each building, executedTime which has the time in days the building has been worked upon and totalTime which is the no of days required to construct the building completely.

This project implements RedBlackTree of balanced binary trees for run-time enhancement and MinHeap where the buildings are ordered by the time they have been worked on and the building with minimum executionTime is stored in root. If two buildings have same executionTime, preference is given to the building with smaller buildingNum

# Class Structure:

4 classes are implemented in total.

* + - risingCity.java
  + Has main method. Inputfile that is passed through arguments is split and each functionality is called to effectivity produced the desired output in the output\_file.txt
  + Has a global variable ‘tm’ to maintain the time in days.
  + The number of active buildings will not exceed 2000. A check for this is implemented.
    - MinHeapImpl – To implement min heap functionality to store the building data (**stored as array of building data**) and order them according to execution time. When a new building is inserted with insert operation, it is inserted into the min heap with arranging the data according to execution time values. If two buildings have same execution time and different buildingNums preference is given to the building with lower building Num. If a building is already constructed and same buildingNum is in the input, the output should stop
    - RedBlackTree – A RedBlackTree is implemented with each tree node containing building data(buildingNum,executionTime,totalTime)
    - Bldg – This is a building Class with the given fields BuildingNum, executionTime, totalTime of the Building.

# Link between RedBlackTree and MinHeap:

For each insert operation, the building data object is inserted into both the min heap and RedBTree. Since the building inserted is same, the reference between them is taken by default. The heap is minified when a min element is retrieved from the root.

daysOfEffort is used in order to maintain the no. of operated days on the current building that is in construction.

*currentBldg – the building which we are working on now.*

# Order of Events and actions:

1. A global time -‘tm’ is maintained with each increment refer to no. of days.
2. When this global time ‘tm’ equals to the first split up letter in the input line, we perform that operationToPerform (can be insert or print)
3. Insert operation creates a building data object inserts it into minHeap and RedBlackTree
4. When the daysOfEffort becomes 5, the minHeapify operation is done and currentBldg is taken to work on it (here, in the min Heap, the root is taken and minHeapify is applied). Again, daysOfEffort is set to 0.
5. If a building construction is ongoing and daysOfEffort variable has not reached 5, then even when insert comes we don’t take that new building rather continue working on this building.
6. When the execution time of the building becomes equal to totalTime, we print its buildingNum and ‘tm’(global variable for time) and remove that building from RedBlackTree and select a new currentBldg from the minheap.
7. If PrintBuilding and a building is completely constructed at the same time, then print all the building data and then output (buildingNum,tm)(tm - globalTime)
8. We continue building until the heap is empty.

.

# Function Prototypes:

## Building.java

1. **public int getBldgID()**

the unique ID or the buildingNum is returned

1. **public void setBldgID(int bldgID)**

the buildingNum of the bldg object is set with the parameter bldgID

1. **public int getExecTime()**

execution time is returned by this method

1. **public void setExecTime(int execTime)**

the executionTime of the bldg object is set with the parameter execTime

1. **public int getTtlTime**

totaltime is returned

1. **public void setTtlTime(int ttlTime)**

the totalTime of the bldg object is set with the parameter totalTime.

## MinHeapImpl.java

1. **public int getSize()**

Returns the heap’s size

1. **private int LeftChildPosition (int pos)**

This method returns the position of the left child in the node present at pos

1. **private int RightChildPosition(int pos)**

This method returns the position of the right child in the node present at pos

1. **private boolean isLeaf(int pos)**

If the node is a leaf node, returns true.

1. **private void swap(int spos, int spos)**

to swap two nodes of the heap this method is used

1. **public void minHeapify()**

MinHeap property is maintained when the root node is being constructed, the rest of the nodes are arranged in the heap accordingly

1. **public void insert(Bldg element)**

a node is inserted into the heap using this method

1. **public void print()**

prints the heap contents

1. **public void minHeaps()**

Function to build minheap using minHeapify

1. **public Bldg remove()**

The min element from heap is removed and returned.

1. **public Bldg getMin()**

Method to return the root element of minheap

## RedBTree.java

1. **public TNode(Bldg data, char color)**

All the variables of TNode are initialized

1. **public TNode makeTNode(Bldg data)**

Method to create node of RBT

1. **public boolean add(Bldg data)**

Method to add node to RBT

1. **private static void balance(TNode treeNode, String color)**

After deleting the node this method balances the RBT

1. **public void remove(Bldg data)**

This method removes one node from RBT

1. **TNode findNext(TNode treeNode)**

This is used to find the next greater element in the RBT Node

1. **public static void balance(TNode treeNode)**

After inserting the node this method balances the RBT

1. **private static void leftRotate(TNode treeNode)**

This method does left rotation

1. **private static void rightRotate(TNode treeNode)**

This method does right rotation

1. **private static void preOrder(TNode treeNode)**

This method does pre-order transversal in a tree

1. **public static void display()**

Method To print the contents of the RBT

1. **public static Bldg search(TNode treeNode, int buildingNumber)**

Method to search for a buildingNumber entry

1. **public static ArrayList<Bldg> searchRange(ArrayList<Bldg> list, TNode treeNode, int startBldgID, int endBldgID)**

Method that searches and gets buildings from from startBldgID and endBldgID range

1. **private static boolean isBetween(int buildingNum, int startBldgID, int endBldgID)**

This method returns true if the provided buildingNum is in between startBldgID and endBldgID

## risingCity.java

1. **public static boolean insert(MinHeapImpl mHeap, RedBTree redbt, Bldg bldg) Building**

data is inserted into minHeap and RedBTree by this method

1. **private static void OperationToPerform(MinHeapImpl mHeap, RedBTree redbt, List<String> tmList, List<String> operationToPerform, BufferedWriter wr) throws IOException**

Performs operations based on the input file and where ever necessary, writes the output to writer file

1. **private static void print(Bldg bldg, long tm, BufferedWriter wr) throws IOException**

This method prints the BuildingNum and time and returns an exception if the output stream is closed.