COP5536 : Advanced Data Structures

Rising City Report

Sri Greeshma Avadhootha

UFID : 16136609

avadhoothas @ufl.edu

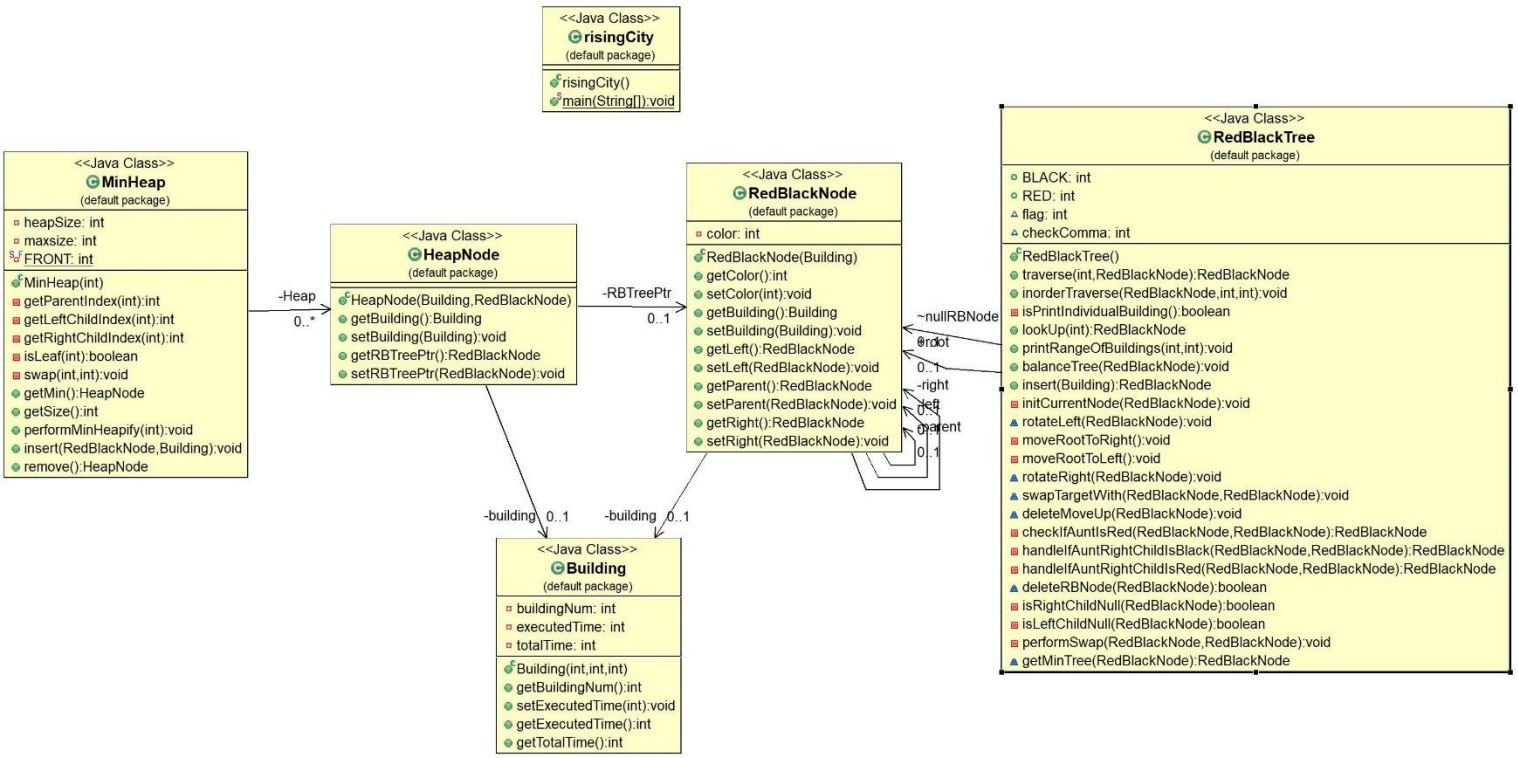
Problem Statement

The aim of the project is to solve keep track of all the building being constructed by the Wayne Enterprise. The program uses a min heap to keep track of the next building to work on while simultaneously read in the various commands from the input file and executes it when it matches the global timer. The program works on a building for maximum of 5 days in a row and then selects the next building to work on. If the building is finished while executing the completion date is printed.

How to Compile and Run:

Type below commands in the terminal after changing your directory to project directory :
make
java risingCity inputfile.txt

Class Diagram



Classes And Important Functions

a) Building

Encapsulation of building attributes and provide setter and getter methods to access the attributes.

- **Attributes**

```
private int buildingNum;
private int executedTime;
private int totalTime;
```

- **Getter and Setter Methods**

```
public int getBuildingNum()
public void setExecutedTime(int execTime)
public int getExecutedTime()
public int getTotalTime()
```

b) RedBlackNode

Represents a node in the red black tree. It encapsulates the Building object and has its own separate attributes. Encapsulation of Building object is needed because when the 'PrintBuilding' is encountered, we can lookup the Building to be printed from the RedBlackNode itself.

- **Attributes**

```
private Building building;
private int color;
private RedBlackNode parent;
private RedBlackNode left;
private RedBlackNode right;
```

- **Getter and Setter Methods**

```
public int getColor()
public void setColor(int color)
public Building getBuilding()
public void setBuilding(Building building)
public RedBlackNode getLeft()
public void setLeft(RedBlackNode node)
public RedBlackNode getParent()
public void setParent(RedBlackNode node)
public RedBlackNode getRight()
public void setRight(RedBlackNode node)
```

c) HeapNode

Represents a node in the MinHeap. It encapsulates the Building object along with an RedBlackNode. These two objects are needed in the HeapNode since the minheap is the main backend part of the RisingCity program.

- **Attributes**

```
private Building building;
private RedBlackNode RBTreePtr;
```

- **Getter and Setter Methods**

```
public Building getBuilding()
public void setBuilding(Building building)
public RedBlackNode getRBTreePtr()
public void setRBTreePtr(RedBlackNode RBPtr)
```

d) RedBlackTree

Represents the RedBlackTree in which the Building data is stored. This class has the standard functions to insert and delete a node and traverse a tree. Along with this, there are some utility functions to rotate the nodes in the tree so that the Red-Black Property is maintained. The balancing of the tree uses the rotate utility functions

- **Standard Functions**

`public RedBlackNode traverse(int buildingNum, RedBlackNode node):` *Traverse the tree from the root to leaf.*

`public void printRangeOfBuildings(int lower, int upper):` *Prints the range of buildings.*

`public RedBlackNode lookUp(int buildingNum) :` *Looks up a particular building and prints the building and its execution time at that particular instant.*

`public RedBlackNode insert(Building building) :` *Inserts building into the Red Black Tree*

`boolean deleteRBNode(RedBlackNode currentNode) :` *Deletes a particular Red Black node*

- **Utility Functions**

```
void rotateLeft(RedBlackNode node)
void rotateRight(RedBlackNode node)
public void balanceTree(RedBlackNode rbNode)
private void performSwap(RedBlackNode currentNode, RedBlackNode y)
private boolean isRightChildNull(RedBlackNode currentNode)
private boolean isLeftChildNull(RedBlackNode currentNode)
private void moveRootToRight()
private void moveRootToLeft()
private RedBlackNode checkIfAuntIsRed(RedBlackNode rbNode, RedBlackNode aunt)
private RedBlackNode handleIfAuntRightChildIsBlack(RedBlackNode currentNode,
RedBlackNode aunt)
private RedBlackNode handleIfAuntRightChildIsRed(RedBlackNode currentNode,
RedBlackNode aunt)
```

e) MinHeap

Represents the min heap which is used to keep track of the next building that the company needs to work on. The class supports standard minheap functions. The class declares a contiguous array of size 2000 (MAX_HEAP_SIZE) according to the constraints mentioned.

- **Standard Functions**

`public void insert(RedBlackNode rbNode, Building building) :` *Insert node into min heap*

`public void performMinHeapify(int index) :` *Min heapify the tree and get the current min based on execution time or building time to the top.*

```
public HeapNode remove() : Removes the current min from the min heap
```

- **Utility Functions**

```
private int getParentIndex(int index)
private int getLeftChildIndex(int index)
private int getRightChildIndex(int index)
private boolean isLeaf(int index)
private void swap(int val1, int val2)
public int getSize()
```

f) risingCity

This class is where the program execution starts. There is a "globalTime" variable which represents the global timer. There is a "timer" variable to keep track of how long the current min building has been executed. It resets once the current min has reached the 5 days limit. The risingCity class handles the following functionality :

1. Reads the input file and parse each instruction into a queue of lists called "operations". This queue of list is used for further processing.
2. Writes the complete result to file "output_file"
3. Each instruction in the queue "operations" is read and the corresponding operation is performed.
4. Handles multiple cases of when an when an insert is performed while the current building hasn't completed its five days of execution, printing the range or individual building when the print command comes in and also running the buildings to completion when the operations given in the file are completed

The main function loops until all the commands have been read and executed and till all buildings have completed their executions. The logic does not execute work on any building during the interval [0-1).

Code flow:

1. Reading Inputs : The command line argument gives us the file from which we will read the instructions. Each of line from the input file is stored in a List where 4 indexes are assigned as follows: Eg: [18 : Insert(12,90)]
   0 - stores the global time
   1 - stores the instruction type,
   2 - has the Building number
   3 - has the totalTime in case of "Insert" and -1 in case of "PrintBuilding"
2. Run a while loop with the conditionality that "operations" queue is not empty, minheap is not empty or the current min node is not null
3. We poll each command from the queue and store it in "inputCommand" variable
4. Once we get the current "inputCommand", we check if the command time is global time match. If they do, we perform the operation in the input command and poll for the next input command. Else, we continue to check the timer state.

5. There is a condition to check if the current building's execution time has reached its total time. If it has reached, we print the building and remove the red black node from the red black tree and from the min heap. If the condition hasn't reached, continue to check the timer state.
6. Check if the timer has reached 5. If it has, it means that the current building has completed its 5 day execution and the min heap needs to be minified.
7. If timer hasn't reached 5, increment the timer state and executed time by one.
8. Increment the global time by one for every iteration of the loop.

Conclusions

The combination of Red Black Tree with Min Heap enables us to perform all the operations in O(logn) time where 'n' is the number of buildings that both the structures hold.