Advanced Data Structures Project Report

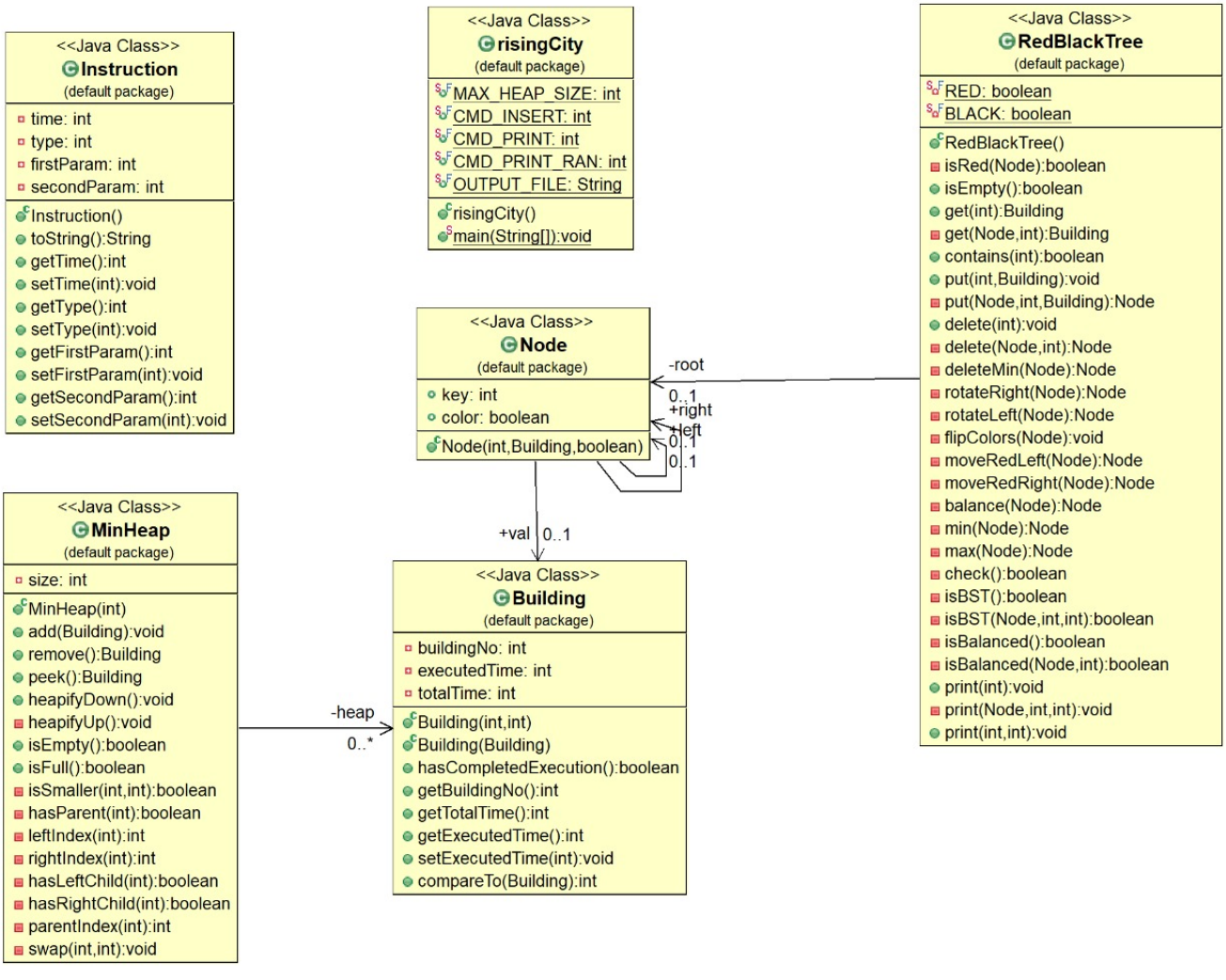
Sai Pradyumna Reddy Chegireddy (UFID: 34631711)
chegireddys@ufl.edu

Problem Statement:

The aim of the project is to help Wayne Enterprises keep track of buildings under construction with the help of a software program. The program uses a min heap to determine the next building that should be worked on while concurrently reading instructions from an input file and executing instructions when the global time matches the instruction time. The instructions can vary between new work orders i.e “Insert” and “PrintBuilding” to output the status of a particular building or to print the list of buildings under construction between two key identifiers. The program uses an implementation of a Red Black Binary Search Tree to perform the “PrintBuilding” operations.

Project Structure:

The figure below depicts the class diagram of the Project depicting the relationship between classes and their function prototypes.



Classes and Important Functions :

Node :

This is a POJO class that maintains pointers to the left subtree and right subtree of a node along with data fields such as key i.e building number and building metadata.

Attributes:

```
public Node left;
public Node right;
public int key;
public Building val;
public boolean color;
```

Building :

This class stores the building attributes and provides getters and setters for these attributes. It also implements the 'Comparable' interface providing ease of comparison in the min heap operations.

Attributes:

```
private int buildingNo;
private int executedTime;
private int totalTime;
```

Utility Methods:

public Building (Building building)
This constructor allows for safe copying when performing swap operations in the min heap implementation
public boolean hasCompletedExecution()
Checks whether building has completed its execution
public int compareTo(Building o)
Compares the two objects based on their executed time and then on building no fields

RedBlackTree :

This class follows an implementation of a Red Black Binary Search Tree using basic data structures and pointers. This class contains functions to perform basic operations such as insert, search and delete. It also has utility methods to print a particular node or print a range of nodes.

Standard Methods:

public void put(**int** key, Building val)
Insert building into red black tree with building no as key; updates if key already exists
public void delete(**int** key)
deletes node with specified key i.e building no

Helper Methods:

private Node rotateRight(Node h)
Make left leaning link lean to the right
private Node rotateLeft(Node h)
Make right leaning link lean to the left
private void flipColors(Node h)
Flip the colours of node and its children
private Node balance(Node h)
Maintains red black tree invariant property

Utility Methods:

public void print(**int** key)
Searches and prints the building metadata of the node with given key i.e building no
public void print(**int** low, **int** hi)
Prints building metadata of all nodes in the range of low and hi recursively traversing only those subtrees that have keys i.e building numbers in the given range

MinHeap :

This class is used to create a min heap which is used to keep track of the next building that the company needs to work on. The class implements helper methods to maintain the min heap invariant.

Standard Methods:

public void add(Building value)
Inserts building to heap
public Building remove()
Remove min element from heap
public Building peek()
Get min element from heap

Helper Methods:
public void heapifyDown()
This functions recursively compares a node with its children and keeps on progressing till the point the node is smaller than both its children.

public void heapifyUp()
The function recursively compares a node with its parent and swaps when the parent is bigger than it. The function stops if the parent is found to be smaller than the node or when the node is root node.

Instruction :

This class stores the parsed instruction metadata from the input and makes their attributes like time, command type i.e Insert or PrintBuilding available through standard getter and setter methods.

Attributes:

private int time;

private int type;
private int firstParam;
private int secondParam;

risingCity :

This class serves as the entry point to our program and performs the following steps

- Reads input file name from the command line arguments and parses the input data in this file and constructs a list of instructions with attributes of time, type, firstParam and secondParam
- Intializes an empty min heap and red black tree. Initializes and set global and build timers to zero.
- Creates an iterator to sift through the list of instructions. Runs a while loop as longs as the iterator has a next instruction or while heap is not empty.
- Perform instruction when its time attribute matches the global counter. If it is an 'Insert' instruction, add the new building only to the red black tree and maintain a list of pending insert instructions on the heap. If it is a 'PrintBuilding' instruction, appropriate method is called on the red black tree
- According to my interpretation of the problem statement, inserting a building into the min heap while another building is currently in execution can alter the min element while the latter is still executing its quota of 5 days. Therefore, the program stores a list of pending insert instructions to be inserted when the current building finishes its execution as stated above.
- When global time does not match the instruction time, the min heap execution comes into picture and here we check for three cases.
- Check if the current building which is executing has completed and if so, it is printed out with its building number and completion time i.e global time. It is then deleted from both the red black tree and min heap. Any pending insertions are performed on the heap and current building to be executed is updated.

- Else If build timer shows that current building had completed its quota of 5 days of execution and if pending insert instructions exist, they are performed, else heapify down is called to maintain the heap invariant and the current building to be executed is updated.
- If it is not one of the above two cases, current building execution time is updated and operations are performed on red black tree and min heap to update the same. Global and build timers are updated

Conclusion :

Using minHeap in combination with a redBlack tree has enabled us to perform all the potential operation in O(logn) where n corresponds to the no of buildings inserted in both the structures.