COP 5536 Fall 2019

Programming Project

Title: Wayne Enterprises Construction management using Min Heap and Red Black tree.

Name: Prajwala Nagaraj
UFID: 1099-2662
UF Email: prajwala.nagaraj@ufl.edu

Data structures used are array for heaps, classes for red black node and heap node within red black tree and Minheap tree implementation.

Problem Statement and Assumptions:

The Project implemented to manage construction of Wayne enterprises reads from a file passed as an argument to the risingCity executable created after compiling the makefile. Input is in the form as shown below:

```
0: Insert(50,20)
15: Insert(15,25)
16: PrintBuilding(0,100)
20: PrintBuilding(0,100)
25: Insert(30,30)
```

With numbers on left considered as arrival time for construction request and first argument in insert considered as building ID and second argument being the building's completion time. Execution time is started from 0 as the building is inserted into datastructures. Building execution time is considered key for min heap and building ID is considered key in

Red black tree. PrintBuilding being case insensitive, can be either with a range (0,100) or can be (50) implying print of range of buildings being constructed or a particular building being under construction. Printbuilding prints the building (building ID, Execution_time, total_time) in this format. Output is written to a file named "output_file.txt" in the order of construction and print commands.

All the buildings on completion of their construction, prints first before getting removed from min heap and red black tree. If a print statement also is requested at the same time as deletion, the deleted building along with other buildings in the print statement get printed as part of printBuilding and is removed later. On occurrence of tie between building's execution time in min heap i.e. when the execution times of two buildings are same and minimum, the building ID with least ID number is selected for execution by program. (0,0,0) is printed for all print statement for which no building exists at the given time. For duplicate insertion of buildings i.e. insertions with same building ID, program ignores that input and doesn't add it to the system.

Implementation:

The program has a total of 8 files namely : risingCity.java [contains the main function and handles the read, write and delegation of functions] , Building_Handler.java[contains logic to write to file, print buildings and insert into min heap and RB tree] , MinHeap.java[contains logic to build, maintain and delete heap nodes from the program], HeapNode.java [contains logic to store building ID, Execution time ,total time and a reference to RB Node processed through MinHeap], RB_tree.java [contains logic to insert, delete and search nodes in red black tree stored by building ID as key], RB_Node.java [stores building ID, Execution time ,total time and red black properties for each of that building], Color.java [stores an Enum to get values for red and black nodes] and a makefile that converts all the .java files into an combined executable "risingCity".

The function prototype of the above mentioned files is as follows:

1. risingCity.java :
```
public static void main(String args[])
//The program execution starts and stops here.
```

2. MinHeap.java:
```
public void insert(int total_time,int executionTime,RB_Node node)
```
//Inserts into Min heap with execution_time as key

```
public int getSize()
```
//returns size of Min heap

```
public void arrayIncreaseByTwo()
```
//increases the array size by doubling for heap expansion

```
public void buildHeap()
```
//Called post insert to ensure heap property

```
public HeapNode extractMin()
```
//To get the min node by execution time

```
public HeapNode remove(int pos)
```
//removes the node at position pos

```
private void Heapify(int i)
```
//checks for heap property and swaps accordingly for min heap maintenance

3. RB_Tree.java:

```
public ArrayList<Integer> PrintBuildings(int ID1, int ID2)
```
//returns the range of buildings from ID1 to ID2 present in Red black tree

```
public int PrintBuilding(int ID)
```
//returns the ID of found building if present else returns 0

```
public RB_Node search_rbNode(int ID, RB_Node root)
```
//searches for the ID th building in red black tree and returns Node with the passed ID

```
public RB_Node insert_rbNode(int ID, int total_time, int executionTime)
```
//inserts the passed building details by creating an object of type RB_Node and inserting into RB tree.

```
public void rotate_left(RB_Node cur_node)
```
//Left rotation of the passed node to balance RB tree

```
public void rotate_right(RB_Node cur_node)
```
//Right rotation of the passed node to balance the tree

```
public void replace_RB(RB_Node r1, RB_Node r2)
```
//Swapping of the passed two nodes

```
public void delete(RB_Node cur_node)
```

//deletion of node from RB_tree

```
public void rearrange(RB_Node cur_node)
```

//Rebalancing of red black nodes to maintain balance and color property

4. Building_Handler.java:

```
public void WritetoFile()
```

//Writes the outputs to the file output_file.txt[Print commands, construction completed buildings]

```
public void Insert(int ID, int total_time, int executionTime)
```

//Insert Building execution time, completion time with building ID into minheap and RB tree

```
public void Print_WayneB(int ID)
```

//Print the Building with passed ID

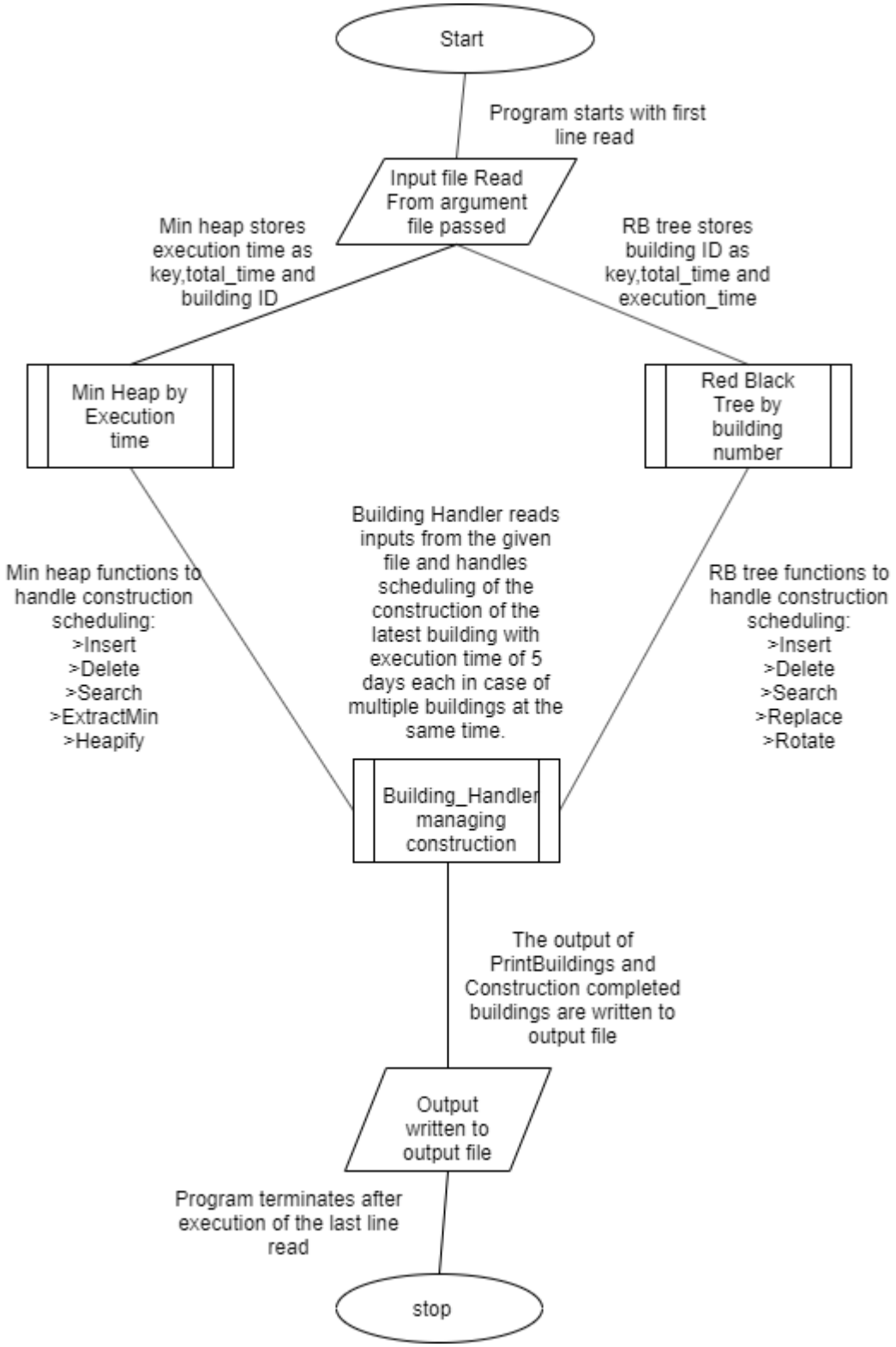
```
public void print_completed_buildings(int ID, long completed_time)
```

//Print construction completed buildings

```
public void Print_WayneBInRange(int ID1, int ID2)
```

//Print all the buildings in the passed range

High level overview of flow of control is as shown in the below flowchart:


Start
Program starts with first line read
Input file Read From argument file passed
Min heap stores execution time as key,total_time and building ID
RB tree stores building ID as key,total_time and execution_time
Min Heap by Execution time
Red Black Tree by building number
Min heap functions to handle construction scheduling:
>Insert
>Delete
>Search
>ExtractMin
>Heapify
Building Handler reads inputs from the given file and handles scheduling of the construction of the latest building with execution time of 5 days each in case of multiple buildings at the same time.
RB tree functions to handle construction scheduling:
>Insert
>Delete
>Search
>Replace
>Rotate
Building_Handler managing construction
The output of PrintBuildings and Construction completed buildings are written to output file
Output written to output file
Program terminates after execution of the last line read
stop

Sample Input:

0: Insert(50,20)
15: Insert(15,25)
16: PrintBuliding(0,100)
20: PrintBuliding(0,100)
25: Insert(30,30)

Sample Output corresponding to above input by our program:

```
(15,1,25),(50,15,20)
(15,5,25),(50,15,20)
(50,60)
(15,65)
(30,75)
```