Advanced Data Structures

COP5536

Fall 2019

Project Report for Rising City

Akib Maredia

UFID: 3885-6489

akibmaredia@ufl.edu

1. Problem Statement.

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.

The needed operations are:

1. Print (buildingNum) prints the triplet buildingNum, executed_time, total_time.

2. Print (buildingNum1, buildingNum2) prints all triplets buildingNum, executed_time, total_time for which buildingNum1 <= buildingNum <= buildingNum2.

3. Insert (buildingNum, total_time) where buildingNum is different from existing building numbers and executed_time = 0.

In order to complete the given task, Min-Heap and a Red-Black Tree (RBT) will be used. Also, you may assume that the number of active buildings will not exceed 2000.

A min heap should be used to store (buildingNum, executed_time, total_time) triplets ordered by executed_time. A suitable mechanism is needed to handle duplicate executed

times. An RBT should be used store (buildingNum, executed_time, total_time) triplets ordered by buildingNum. You are required to maintain pointers between corresponding nodes in the min-heap and RBT.

Wayne Construction works on one building at a time. When it is time to select a building to work on, the building with the lowest executed_time (ties are broken by selecting the building with the lowest buildingNum) is selected. The selected building is worked on until complete or for 5 days, whichever happens first. If the building completes during this period, its number and day of completion is output and it is removed from the data structures. Otherwise, the building's executed_time is updated. In both cases, Wayne Construction selects the next building to work on using the selection rule. When no building remains, the completion date of the new city is output.

Programming Environment

I have used Java for this project. My submission will include a makefile that creates an executable file named risingCity.

2. Input and Output Requirements

Input Format

For java:$ java risingCity input.txt. Where input.txt is the name of the file that has the input test data. Input test data will be given in the following format; Insert (buildingNum, total_time), PrintBuilding (buildingNum), PrintBuilding (buildingNum1, buildingNum2). Inserting a building for construction to the data structures unless global time equals to the arrival time of the construction is not allowed. Data is given in days, following is an example of input.

0: Insert(5, 25)
2: Insert(9, 30)
7: Insert(30, 3)
9: PrintBuilding(30)
10: Insert(1345, 12)
13: PrintBuilding(10, 100)
14: Insert(345, 14)
39: Insert(4455, 14)

The number at the beginning of each command is the global time that the command has appeared in the system. A global time counter is used named 'time', which starts at 0. Input command is read only when the global time matches the time in the input command. When

no input remains, construction continues on the remaining buildings until all are complete. PrintBuilding (buildingNum) query should be executed in O(log n) and PrintBuilding (buildingNum1, buildingNum2) should be executed in O(log n + S) where n is number of active buildings and S is the number of triplets printed. For this, your search of the RBT should enter only those subtrees that may possibly have a building in the specified range. All other operations will take O(log n) time.

Output format

- Insert(buildingNum, total_time) should produce no output unless buildingNum is a duplicate in which case you should output an error and stop.
- PrintBuilding (buildingNum) will output the (buildingNum, executed_time, total_time) triplet if the buildingNum exists. If not print (0, 0, 0).
- PrintBuilding (buildingNum1, buildingNum2) will output all (buildingNum, executed_time, total_time) triplets separated by commas in a single line including buildingNum1 and buildingNum2; if they exist.

If there is no building in the specified range, output (0, 0, 0). You should not print an additional comma at the end of the line. Other output includes completion date of each building and completion date of the city. All output goes to a file named "output_file.txt".

3. Implementation.

We are not allowed to use complex data structures which are provided by programming languages. We have to implement min-heap and RBT data structures on our own using fundamental data structures. We are not allowed to use map related libraries.

Program flow:

The flow of the program is given as follows:

```
//work until input file has commands or buildings are to be worked upon
while(minheap!=empty or currentBuilding != null or readLine != NULL)

      C = Command from input file with time.

      //check if current time and the time of the command is same only
then execute the command
      if(c.time == currentTime)
            Execute C.command;

      //check for building that is finished
      if(completedBuilding != Null)
            print completedBuilding;
            delete from RBT and MinHeap
```

```
	//check if currentBuilding construction session is completed
	if(currentBuilding == null)
		if(MinHeap != empty)
			currentBuilding is set to extractMin();

			timeLeft = totalTime - executedTime;
		else
			increment the global counter;

	increment executedTime of the currentBuilding;

	time++;

	//check if construction of a building is finished
	if(currentBuilding.executedTime == currentBuilding.totalTime)
		delete currentBuilding reference from RBT

	//check if the construction of a building is not finished after
slot ends
	if(currentBuilding.executedTime != currentBuilding.totalTime)
		reinsert into the minheap;

	set current building to null;

end while

if a building is finished, write it to the output file
remove from the RBT

end()
```

risingCity.java:

This is the main java file which has reads the input from the input.txt file and uses the begin function to start the city construction.

The main function also calls insertBuilding which inserts the building into the RBT and min-heap.

And finally writes the output to the output_file.txt.

BuildinNode.java:

This is the building structure. It has three parameters.
buildingNum: Every building has a unique building number

executedTime: Executed time of a building.
totalTime: Total time of a building.

RbNode.java:

This is java class is the node structure of the red black tree.

Rbt.java:

This is the actual implementation of the RBT. Here we can retrieve the building which we need during a PrintBuilding(buildingNum) command and also provides the buildingNums of all buildings for PrintBuilding(buildingNum1, buildingNum2) for which search and searchInRange functions are used. After the building is finished the construction, it is removed from the RBT using the RBT delete cases. During an insert, we need to apply RBT insert cases in order to maintain binary search tree property where the tree uses the buildingNum as the key according to which the data is sorted. For maintaining RBT properties, deleteFix() and insertFix() functions are used which apply rotations and color flips.

MinHeap.java:

Implements minheap using executionTime as the key to maintain order. Removes the building with minimum executed time when extract min is called and calls heapify also removes the building with that building_

HeapNode.java:

Uses executed_time as the main key.

4. Submission

The following contents are required;

1. Makefile: Make file must be directly under the zip folder.
2. Source program has comments.
3. Report has all the information about the functions and classes.

Submission is compressed in a single file with name Maredia_Akib.zip.

This file contains the “input.txt” file as a sample input.