COP 5536 Fall 2019 Programming Assignment

Name: Udit Jaitly

UFID: 8013-7110

UF-mail: uditjaitly@ufl.edu

The project has been implemented in C++ language. The IDE used is XCode.

Problem Statement:

This project aims to keep track of the construction process of buildings in a city. To implement this, we are using a Red Black Tree and an array-based Min Heap data structure. Wayne Construction, the company which oversees construction can only construct one building at a time. The selection process of the building to be worked upon is simple. The building with the minimum Executed Time is selection. In case of a tie, we select the building with the lowest Building Number. We repeat this process until all the buildings have finished the construction process.

This project consists of the MinHeap in which the nodes (building number, executed time, total time) are stored in the order of their Executed Times. The RBT stores the buildings in order of their Building Numbers. When the insert command is encountered, the node is inserted in the RBT and the MinHeap. Basically, the root of the min heap would be the building which will be selected to undergo the construction process. After each day, the root's executed time would be updated by 1 and it is updated it in the Red Black Tree and the MinHeap. The work is done on the selected building for 5 consecutive days. After 5 days, the selection process is repeated. After the completion of the building, we remove the building from the Red Black Tree and the MinHeap.

The structure of the project consists of 4 parts:

1. MinHeap
2. Red Black Tree
3. GetInput & construct (Driver Functions)
4. Main

The flow of the program is as follows:

The Main function is called when the program is executed. It calls the main driver function **GetInput**. This driver function keeps on reading the input file and performs respective operations (Insert/Print) on the MinHeap and the RBT. For each operation, driver function **getInput** calls **construct**. **construct** function after performing construction checks the conditions and returns a true/false value to the **getInput** function. Based on that value, **getInput** takes appropriate action for print/insert. Both driver functions call Data Structure methods for MinHeap and Red Black Tree. All these methods are explained below.

- ### Implementation of MinHeap

Methods:

- **insertKey**: It inserts a node in the Heap, after insertion as the heap is altered, it makes the required changes so that the resultant heap retains the properties of a MinHeap
- **printHeap**: It prints the current heap
- **parent**: returns the parent index of a node
- **heapify**: Maintains the heap properties of the heap by making appropriate changes. It is called after we delete the root element or after work has been executed on an element for five days. It also handle tie-breakers (when Executed Time of 2 buildings is the same)
- **printDelete**: Prints the building number and current global counter at the time when a building gets removed from the node
- **deleteRoot**: Deletes the root of the minHeap and calls **printDelete** and **heapify**.

- ### Implementation of Red Black Tree

Methods:

- **searchRBT:** Search RBT for a specific Building Number
- **print_RBT:** Print RBT in in-order sequence
- **find_minRBT:** Finds the minimum element in the RBT
- **find_maxRBT:** Finds the maximum element in the RBT
- **buildingsInRange:** Finds the buildings within a specified range and store them in temporary arrays
- **updateExecutedTimeRBT:** Updates the executed time for the buildings in the RBT
- **print_B_Info:** Looks for a specific element in the RBT and prints Building Number, Executed Time, Total Time in the output file.
- **print_B_In_Range:** Calls **BuildingsInRange** function and prints the buildings information from the temporary arrays to the output file
- **rotateLeft:** Performs left rotation with respect to the input node.
- **rotateRight:** Performs right rotation with respect to the input node.
- **fixInsertRBT:** Fixes the RBT after a node is inserted in the RBT
- **insertRBT:** Inserts a node in the RBT
- **transpantRBT:** Rebalances the RBT
- **fixDeleteRBT:** Fixes the RBT after a node is deleted in the RBT

- **Driver Methods-**

♦ **getInput:**

- Reads the input file line by lime and looks for the Insert/Print operations are performed. It checks if the global counter value is less than or equal to the time mentioned in the input file. It executes the construct operation until the global counter is equal to the time which is read in.
- If the global counter is equal to the time read in, check for the command. If it is a print command, the appropriate print function is called. In case of a insert operation, it checks for the value returned by the boolean **construct** function. If it is true, we can go ahead and perform the new insert operation. If it is false, we store the building in a vector which consists of buildings whose insert operation is pending. These are inserted when the **construct** function returns true.
- After all this reading is done, the program keeps on performing construction on buildings until the minheap size is zero.

♦ **construct**: It is the main logic behind this project. It increments each building's executed time one by one. It also checks if a building has been executed for 5 consecutive days. If a building has finished construction ,ie. the executed time=total time, it inserts the pending buildings in the MinHeap. It is a function which returns boolean to the **get_Input** method. It returns true when a building has finished construction or if it has been constructed for 5 consecutive days. It returns false if the selected building is yet to complete five days of construction. When it returns false and there are buildings that are read from the input file, we insert them in the heap once the selected building finishes its construction for 5 consecutive days

Also, the precedence is given to input command over deletion command in case of edge cases.

- **Main Method-** It takes in the command line argument as the name of the input file and runs the main driver method **getInput**.

Time Complexity:

Red Black Tree- Red Black Tree operations are performed in O(log(n)+s) where s is the visited nodes.

MinHeap- The heap has ceil($\log_2$(N+1)) levels .The insert and delete operations are performed at most once each level. Hence, its complexity is O(logN).

Prototypes for all the methods:

- NODE search_RBT(NODE root, int bNum)
- void print_RBT(NODE n)
- NODE find_minRBT(NODE root)
- NODE find_maxRBT(NODE root)
- void BuildingsInRange(NODE n,int min,int max,int b1,int b2)
- void updateExecutedTimeRBT(NODE root, node *BuildingRecord)
- void Print_B_Info(NODE root, int bNum, int indicator)
- int Print_B_In_Range(NODE root,int flag,int b1,int b2)
- void rotateLeft(NODE *root, NODE n)
- void rotateRight(NODE *root, NODE n)
- void fixInsertRBT(NODE *root, NODE n)
- NODE insertRBT(NODE *root, int bNum, int ttime)
- void transplantRBT(NODE *root, NODE u, NODE v)
- void fixDeleteRBT(NODE *root, NODE i)
- NODE deleteRBT(NODE *root, int z)
- int parent(int i)
- void insertKey(node *buildingRecord,int buildingNum, int total_time)
- void print_heap(node *buildingRecord)
- void heapify(node *buildingRecord,int n,int i)
- void print_delete(node* buildingRecord)
- int delete_Root(node *buildingRecord)
- bool construct(node *buildingRecord, node *tree)
- void get_Input(node *buildingRecord)