Advanced Data Structures
COP 5536: Fall 2019

Programming Project Report

Arpan Banerjee

UFID: 9359-9083

UF Email: arpanbanerjee@ufl.edu

Function signatures

File – building.h

*Class **BuildingDetails***

- This class contains all the parameters for a building's construction and a custom comparator for comparing buildings in the min heap

- **buildingNum**: Building number
- **executed_time**: The time the building has been worked on until now
- **total_time:** Total time required for the building to finish construction

- **BuildingDetails(int buildingNum, int executed_time, int total_time):** Constructor for the class for initializing with values

- **bool operator<(BuildingDetails other) const:** Overloads the less than < operator for comparing two objects of class BuildingDetails. Compares the executed_time first and breaks ties using buildingNum.

File – myheap.cpp

*Class **MyHeap***

- This class contains all the functionalities to implement a min-heap with BuildingDetails objects using a vector

- **std::vector<BuildingDetails *> v**: Vector of pointers to BuildingDetails objects

- **int parent(int i)**: Returns the parent index for node index i

- **int left(int i)**: Returns the left child index for node index i

- **int right(int i)**: Returns the right child index for node index i

- **void heapify(int i)**: Function to heapify down the subtree rooted at index i, recursively comparing the node to its children

- **bool is_empty()**: Function to check if the heap is empty or not, returns true if heap_size is greater than 0

- **BuildingDetails* extract_min()**: Function to extract the minimum building from the heap, returns pointer to the BuildingDetails object

- **void decrease_key(int i, int key)**: Function to decrease key (executed_time) of building at index i in the heap vector and set it equal to key, repeatedly checks if smaller than its parent

- **void insert(BuildingDetails *&bd)**: Function to insert a building to the heap, argument is a pointer to the building to be inserted

- **void print_heap()**: Debug function to print the heap vector

File – rbtree.cpp

*Class **RBNode***

- This class encapsulates all the node data for the red black tree including a comparison operator overload

- **BuildingDetails *bd**: Pointer to the BuildingDetails object that the node corresponds to
- **COLOR color**: An enumeration COLOR for the color of a node
- **RBNode *left, *right, *par**: Pointers to the left child, right child and parent of a node respectively

- **RBNode()**: Default constructor for the class
- **RBNode(BuildingDetails *bd)**: Constructor to initialize a node with a BuildingDetails pointer, sets color as Red

*Class **RBTree***

- This class implements all the functionalities of a red black tree using RBNode as nodes

- **RBNode *root**: Pointer to root node of RBT

- **void rotateLeft(RBNode *&)**: Function to perform a single left rotation at the argument node

- **void rotateRight(RBNode *&)**: Function to perform a single right rotation at the argument node

- **RBNode* insertBST(RBNode *&, RBNode *&)**: Function to insert node into tree like a BST, returns the node at the correct position

- **void fixInsert(RBNode *&)**: Function to handle all the cases of RBT insertion to make the tree retain RBT properties

- **RBNode *minValueNode(RBNode *&)**: Function to find the min value node (leftmost node) in a given tree

- **RBNode* deleteBST(RBNode *&, int)**: Function to delete a specific key from the tree like in a BST

- **void fixDelete(RBNode *&)**: Function to rebalance the tree after deletion and handle all the RBT cases

- **void inorderBST(RBNode *&)**: Helper function for inorder traversal of tree (for debugging)

- **RBTree()**: Default constructor, sets root to null

- **void insertBuilding(BuildingDetails *&bd)**: Function to insert a building into the red black tree, argument is pointer to BuildingDetails object

- **void deleteBuilding(int buildingNum)**: Function to delete a building with the given buildingNum from the red black tree

- **BuildingDetails get_building_helper(RBNode*, int)**: Helper function for getBuilding, it recursively finds the given buildingNum in the tree

- **BuildingDetails getBuilding(int)**: Function to find by buildingNum and return the BuildingDetails object

- **std::vector<BuildingDetails> get_between_helper(RBNode*, int, int, std::vector<BuildingDetails> &)**: Helper function for getBetween for print all values in range, recursively checks if node is between left and right boundaries

- **std::vector<BuildingDetails> getBetween(int, int, std::vector<BuildingDetails> &)**: Function to get a vector of BuildingDetails objects with buildingNums between the given range

- **void print()**: Debug function to print the red black tree inorder

File – main.cpp

- This file contains the main logic for reading input from the file and the execution loop along with accessing and calling heap and red black tree functionalities.

- **enum Command {insert, print1, print2}**: Enumeration for command among the three - *insert building*, *print single building* and *print range of buildings* respectively

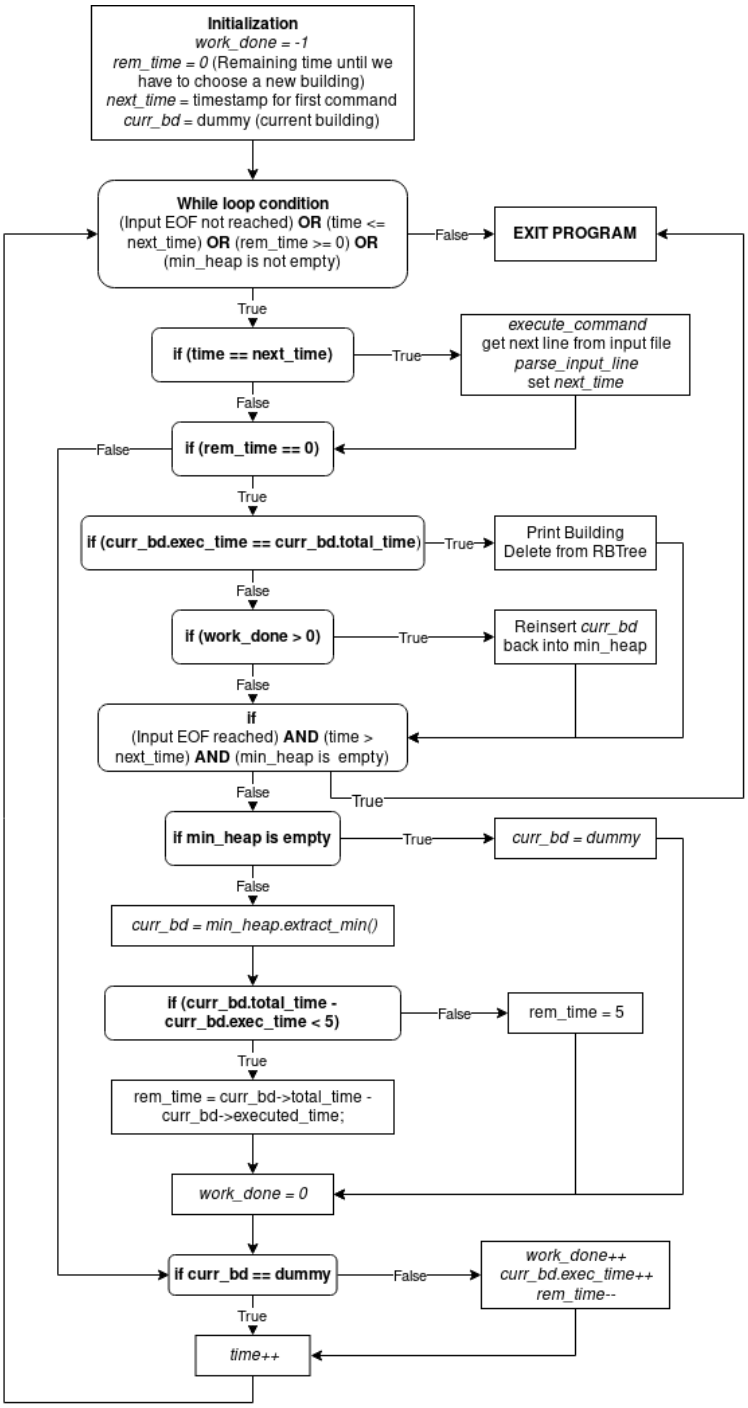
- **std::tuple<int, Command, std::vector<int>> parse_input_line(std::string)**: Parses the input line and returns a tuple containing the following -
    - int timestamp: the time at which the command is to be read
    - Command: one of the items of enum Command
    - std::vector<int> args: the args of the command. For insert it is buildingNum and total_time, for print it is the 1 buildingNum or two for a range

- **void execute_command(std::tuple<int, Command, std::vector<int>>, MyHeap&, RBTree&)**: Takes the above parsed tuple as input and executes the required command
    - Insert: Inserts a new building into the min heap and red black tree
    - Print (for single): Uses rbtree.getBuilding to get details from the red black tree
    - Print (range): Uses rbtree.getBetween to get all buildings in the range

- **int main(int argc, char const *argv[])**: The main function of the program, takes as input the input filename from the command line. Has a while loop for reading lines from the input file, executing commands and the main min heap logic for choosing buildings.

Project Structure

The main execution loop logic for reading input and calling heap and red black tree functions is as follows.



Complexity

- Complexity for heap insertion and extracting min is O(logN) as we recursively compare up to the root or go down a branch respectively.
- Complexity for red black tree operations like insert and delete is also O(logN).
- Complexity for print single building is O(logN) as we only need to go into one of the branches at each level, like search in BST.
- For printing range of buildings, we go into the left child if root value is greater than left boundary and similarly for right child. Hence complexity of operation is O(k) where k is the number of nodes (buildings) in the given range. In the worst case, it is O(n).