Advanced Data Structures (COP 5536)

Project Report - Fall 2019

Disha Nayar

UFID# 6199 1035

dnayar@ufl.edu

Project Description :

The project consists of three main implementations. First, is to implement a min heap using an array-based implementation, second, to implement a red black tree and finally to define a class that uses the two data structures to keep track of constructing buildings in a city.

The following logic is used to write the main code :

The Construction of the building starts at day 0. It is worked on for five days or till its execution time is same as the total time. The next building is then selected for execution based on the selection criteria. The program exits if there are no further inputs available and when the construction of all the active buildings is completed.

For example, building (1,7) is inserted at global time 0 and building (2,5) is inserted at global time 6. Then building 1 is worked on day 0,1,2,3 and 4. When global time = 5, since building 1 still has the minimum executed time, it is picked up for work again. When global time = 6, building 2 is inserted in the heap and the construction on building 1 continues. When global time = 7, Since the execution time of building 1 is equal to its total time, the building is deleted [output (1,7)]. Next, building 2 is picked for construction. It will be worked on global time 7,8,9,10 and 11. On global time = 12, the building is deleted. [output (2,12)].

Executing Instructions :

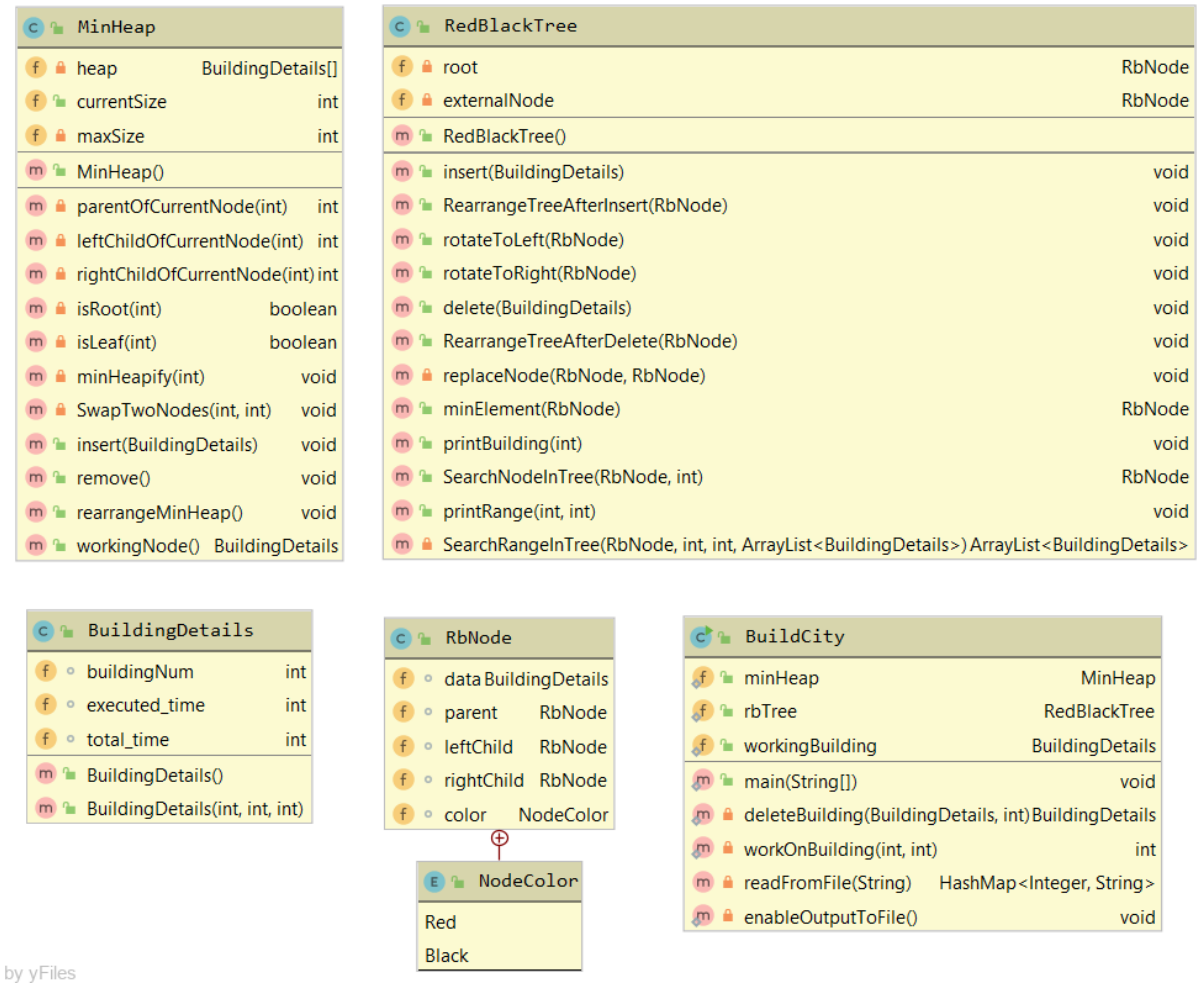
1. Unzip the folder
2. Paste the input file into the project folder
3. Enter command – make
4. Enter command – java risingCity < InputFile.txt >
5. View the output in output_file.txt

Structure of the Program and Function Description

The projects has 5 classes. ‘BuildingDetails.java’ and ‘Minheap.java’ are used to create an array implementation of a heap, where every element of the array is of type BuildingDetails. ‘RbNode.java’ represents the structure of a node in a red black tree. It’s data is of type BuildingDetails, to hold the buildings. ‘RbNode.java’ and ‘RedBlackTree.java’ are used to implement a red black tree, where every node in the red black tree is of type RbNode. Finally, ‘BuildCity.java’ is the main driver class. It takes in the input, works on building construction and calls the heap and the red black tree functions to insert, delete and print buildings.

Detailed explanation of the classes and functions is given below.

Class Diagram



Class BuildingDetails : Represents a building and holds its number, execution time and total time.

Global Variables :

1. buildingNum (type - int) – holds the number of the building in the heap and the tree.
2. executed_time (type - int) – holds the number of days for which the building has been constructed.
3. total_time (type - int) – holds the number of days that a building requires to complete.

Constructor :

Parameters : bNum, Etime, Ttotal

The Constructor takes in three parameters and assigns these values to buildingNum, executed_time and total_time respectively.

Class Minheap : The class contains functions that implements a min heap.

Global Variables :

1. heap (type – BuildingDetails) - holds the buildings inserted for construction.
2. currentSize (type - int) - holds the current size of the heap. i.e the number of active buildings for construction.
3. maxSize (type – int) - gives the maximum number of building that's can be constructed.

Constructor :

Initializes the heap and sets the currentSize and maxSize.

Functions :

1. parentOfCurrentNode()
    a. return Type : int
    b. parameters : position

   This function calculates and returns the index of the parent for index position.

2. leftChildOfCurrentNode()
    a. return Type : int
    b. parameters : position

   This function calculates and returns the index of the left child for index position.

3. rightChildOfCurrentNode()
    a. return Type : int
    b. parameters : position

   This function calculates and returns the index of the right child for index position.

4. isRoot()
    a. return Type : Boolean
    b. parameters : position

    This function calculates if the position is the root and returns true, else it returns false.

5. isLeaf()
    a. return Type : Boolean
    b. parameters : position

    This function calculates if the position is the leaf node and returns true, else it returns false.

6. minHeapify()
    a. return Type : none
    b. parameters : currentPosition

    Functionality – It checks if the node at the currentPostion is not a leaf node. If it is not a leaf node, then it first compares the current node with its left child. If the left child is not null, and the execution time of the current node is greater than its left child, it calls the method to swap the current node and the left child. If the execution time for the current node and the left node is the same, it checks with the building numbers. If the building number of the current node is greater than its left child then it calls the method to swap the current node and the left child. It next compares the current node with the right child and repeats the above steps.

7. SwapTwoNodes()
    a. return Type : none
    b. parameters : currentPosition, NodeToSwapPosition

    The function takes in two integer positions and swaps the nodes in the two positions.

8. Insert()
    a. return Type : none
    b. parameters : node (Type - BuildingDetails)

    Functionality – If the current size of the heap is less than the max size, then it inserts the new node at current size plus 1 and increments the current size.

9. Remove()
    a. return Type : none
    b. parameters : none

    Functionality – The function removes the root which is at index position 1. If the current size of the heap is greater than 1, then it assigns the node at the last position to the root position. It reduces the size of the heap by 1 and then calls the rearrangeMinHeap() that fixes the heap. If the current size of the heap is 1 then it just reduces the current size by 1.

10. rearrangeMinHeap()
    a. return Type : none
    b. parameters : none

Functionality : Starting from the parent of the node at the last position till the root, it calls the minheapify function on every position. This fixes the heap even if there are multiple changes in different nodes. Example when a new building is inserted while working on one building, there will be two different nodes in the heap.

11. workingNode()
    a. return Type : BuildingDetails
    b. parameters : none

    Functionality – If the current size of the heap is not zero, then it returns the root of the heap. That is, the building to start construction on.

Class RbNode : It defines a node in a red black tree.

Global Variables :

1. data (type - BuildingDetails) - holds the values of the building.
2. parent (type – RbNode) - holds the parent node.
3. leftChild (type – RbNode) - holds the left child of the current node.
4. rightChild (type – RbNode) - holds the right child of the current node.
5. Color (type – NodeColor) - represents the color of the current node.

Enum nodeColour :

It takes the values of either Black or red representing the color of the node, in a red black tree.

Class RedBlackTree : The class contains functions that implements a Red Black Tree.

Global Variables :

1. root – holds the building with the minimum building number
2. externalNode – defines an external node in the tree

Constructor :

Initializes the values for an external node and sets the root of the tree to an external node.

Functions :

1. insert()
    a. return Type : none
    b. parameters : element (type - BuildingDetails)

    Functionality - It first traverses from top to bottom and finds the parent whose child will be the new node. If the parent is null then the node is assigned to root, else the node becomes the left or the right child of the root depending on its building number. If the new node

inserted is the root, its color is changed to black. It then calls the RearrangeTreeAfterInsert function to fix the red black tree.

2. RearrangeTreeAfterInsert()
    a. return Type : none
    b. parameters : node (type - RbNode)

    Functionality - While the parent of the node is red in color, the function performs left and right rotations depending on the color of the uncle to satisfy the red black tree properties. In the end it changes the color of the root to black.

3. rotateToLeft()
    a. return Type : none
    b. parameters : parent (type - RbNode)

    Functionality - The function rotates the subtree to the left about node parent. After the end of the operation, its right child (lets say x) becomes it parent. And node becomes the left child of x.

4. rotateToRight()
    a. return Type : none
    b. parameters : parent (type - RbNode)

    Functionality - The function rotates the subtree to the right about node parent. After the end of the operation, its left child (let's say x) becomes it parent. And node becomes the right child of x.

5. delete()
    a. return Type : none
    b. parameters : element (type - BuildingDetails)

    Functionality - It first finds the node containing element and assigns the node to variable nodeToDelete. Its copy is stored in variable DeleteNode and colour is assigned to colorOfDNode. It then handles the following cases :

    1. Node to delete has one child
    2. Node to delete has two children – this intern handles cases when the children have further children or not.

    It then calls the function replace node, which replaces the node to be deleted. If the colour of the deleted node is Black, the function calls RearrangeTreeAfterDelete to fix the red black tree

6. RearrangeTreeAfterDelete()
    a. return Type : none
    b. parameters : node (type - RbNode)

    Functionality - It iterates till the colour of the node is black and the node is not the root. The function performs color change, left and right rotations depending on the color and the position of the sibling to satisfy the red black tree properties

7. replaceNode()
    a. return Type : none
    b. parameters : nodeToReplace (type - RbNode) , replacingNode(type - RbNode)

   Functionality - If the nodetoReplace is the root then it simply assigns root to replacing value. If it is not the root, then, the node to replace is replaced by the replacing node. The replacingNode is made the left or the right child of the parent of the nodeToReplace, depending on the initial position of the nodeToReplace.

8. minElement()
    a. return Type : RbNode
    b. parameters : element (type - RbNode)

   Functionality - the function returns the node with the minimum value from the path of the element to the external node.

9. printBuilding()
    a. return Type : none
    b. parameters : buildingNumber (type - int)

   Functionality - It calls the function SearchNodeInTree to search for the building number in the red black tree. If the building is present, it prints (buildingNum,executed_time,total_time). If the building is not present, it prints (0,0,0)

10. SearchNodeInTree()
    a. return Type : RbNode
    b. parameters : element (type - RbNode), buildingNumber (type - int)

   Functionality - It searches the tree and returns the node if its buildingNum matched with buildingNumber.

11. printRange()
    a. return Type : none
    b. parameters : number1 (type - int), number2 (type - int)

   Functionality - It calls the function SearchRangeInTree to search for all the buildings that's are currently active between number1 and number 2. The list is stored in variable buildingNumbers and prints all values comma separated. If the range contains no building it prints (0,0,0)

12. SearchRangeInTree()
    a. return Type : ArrayList<BuildingDetails>
    b. parameters : element (type - RbNode), number1 (type - int), number2 (type - int), bNums (type - ArrayList<BuildingDetails>)

   Functionality - The function recursively calls itself and looks for buildings in the range number1 and number2. The left child of the element is first checked to get the list in the ordered way (ascending order).

Class BuildCity : This class contains the functions to input the data and the main method to perform operations to construct the building.

Global Variables :

1. minheap – this is the object of the class minheap
2. rbTree – This is the object of a RedBlackTree
3. workingBuiding (type - BuildingDetails) – The variable holds the building for which the construction is in process.

Functions :

1. Main()
    a. Return Type : none
    b. Parameters : String[] arg (arg[0] is the input file name)

    Variables

    1. inputData – holds the data from the input file
    2. globalTime – global counter to instruct when to read perform inserts and print.
    3. currentExecutedTime – keeps track of the number of days the current building has been worked on.

    Functionality – globalTime and currentExecutionTime are initially set to zero. It first checks the inputdata has the key with global time and performs accordingly.

        1. Insert : a new object for building is created and inserted in both heap and the red black tree.
        2. Print : depending on the print statement, the building or the range of buildings are printed.

    It then calls the deleteBuilding function to check if the building needs to be deleted. Next the workOnBuilding function is called which picks the building to work on and updates the currentExecutedTime. The global time is then incremented by 1 and the process is repeated.

2. deleteBuilding()
    a. return Type : workingBuilding
    b. parameters : workingBuilding (Type – BuildingDetails), globalTime (Type - int)

    Functionality – If the execution time is equal to the total time for the working building then it calls the remove and delete functions from minheap and red black tree to remove the workingBuilding.

3. workOnBuilding()
    a. return Type : int
    b. parameters : globalTime (Type – int), currentExecutedTime (Type - int)

    Functionality – This function picks up the building to work on. currentExecutionTime keeps track of the number of days the current building is being worked on. If the currentExecutedTime reaches 5, then the heap is rearranged and the building at the root is

picked up for working. The function increments the execution time of the workingBuilding and currentExecutedTime and returns the currentExecutedTime.

4. readFromFile()
    a. return Type : HashMap<Integer,String>
    b. parameters : fileName (type - String)

    Functionality - This function reads the input file and stores it in a Hash map. The key is the global time at which the instructions are to be read and value is the insert or print statements. It returns the hash map with the data.

5. enableOutputToFile()
    a. return Type : none
    b. parameters : none

    Functionality – The function overrides the PrintStream method to print the output in output_file.txt file