ADS PROJECT REPORT

First Name: Aseesh

Last Name: Mullapudi

UFID: 9175-1971

UF email ID: aseesh.mullapudi@ufl.edu

Project Name: Rising City

Programming Language: C++

Source code consists of the following files:

1) Building.hpp – Defines the structure of Building node.
2) Min_Heap.hpp – Explains the structure of Min_Heap.
3) Min_Heap.cpp – Defines the methods possible on Min_Heap.
4) Red_Black_Tree_Node.hpp – Explains the structure of a Red-Black Tree node.
5) Red_Black_Tree_Node.cpp – Defines the methods possible on Red-Black Tree node.
6) Red_Black_Tree.hpp – Explains the structure of Red-Black Tree.
7) Red_Black_Tree.cpp – Defines the methods possible on Red-Black Tree.
8) risingCity.cpp – Defines the main work flow.

Building – Building():

1) Attributes:

   int buildingNumber

   int executionTime

   int totalTime

2) Member Functions:

   Building(int buildingNumber, int executionTime, int totalTime)

Building constructor takes in the buildingNumber, executionTime, totalTime as input and assigns them to the Building's corresponding attributes.

Min_Heap – Min_Heap():

1) Attributes:

   pair<Building, RBTNode*> *min_heap_array // Min Heap node values holds a pair which holds the address of the corresponding building's red-black-tree node address.

   int capacityOfMinHeap

   int min_heap_size

2) Member Functions:

   Min_Heap(int totalSize) // Constructor to set the min_heap_size and initialize the min_heap_array.

   int getParentNodeIndex(int i) // Returns the parent node index of a given index.

   int getLeftNodeIndex(int i) // Returns the left node index of a given index.

   int getRightNodeIndex(int i) // Returns the right node index of a given index.

   void push(pair<Building, RBTNode*> nodeValue) // Insert the given pair to Min Heap and fixes the violation of Min Heap property.

   void minHeapify(int) // Sets the Min Heap property when violated during deletion.

   pair<Building, RBTNode*> top() // Returns the first element in the min_heap_array.

   void pop() // Deletes the first min_heap_array element.

   unsigned int size() // Returns the size of the Min Heap.

Red_Black_Tree_Node – RBTNode():

1) Attributes:

   Building value // Building stored in the Red_Black_Node.

   COLOR color // Color of the Red_Black_Node – Either Red or **Black**.

   RBTNode *leftChild, *rightChild, *parent // 3 pointers to left, right and parent nodes.

2) Member Functions:

   RBTNode(Building val) // Constructor to make a Red Black Tree Node.

   RBTNode* getUncleNode() // Returns the uncleNode address of a given Node.

   bool isOnLeftSide() // return true if the given node is a left Child of a node.

   RBTNode* getSiblingNode() // Returns the siblingNode address of a given Node.

   void moveDown(RBTNode *newParent) // Rearranges the node with the newly given ParentNode address.

   bool hasRedChild() // return true if the given node has a Red Color child.

Red_Black_Tree – RBTree():

1) Attributes:

   RBTNode *root // root of the Red-Black Tree.

2) Member Functions:

   RBTree() // Constructor for the Red Black Tree (Initiates root to NULL initially)

   void leftRotate(RBTNode *node) // Rotates the given node towards its left side by changing the parent and child pointers respectively.

   void rightRotate(RBTNode *node) // Rotates the given node towards its right side by changing the parent and child pointers respectively.

   void fixRedRed(RBTNode *node) // Fixes the Red-Black Tree violation when there is a Red-Red conflict during new insertion into Red-Black Tree.

   void swapColors(RBTNode *node1, RBTNode *node2) // Swaps the colors fo 2 given Red-Black Tree nodes.

   void swapValues(RBTNode *node1, RBTNode *node2) // Swaps the values of 2 given Red-Black Tree nodes.

   RBTNode* successor(RBTNode *node) // returns the immediate successor (in inorder traversal) of a given node in Red-Black Tree.

   RBTNode* BSTreplace(RBTNode *node) // Replace the given node with its immediate successor.

   void deleteNode(RBTNode *node) // Deletes the given node from the Red-Black Tree.

void fixDoubleBlack(RBTNode *node) // fixes the double Black case formed during deletion of given node from the Red-Black Tree.

void update(RBTNode *root, int buildingNumber, int executionTime) // updates the execution time of a given Building in a Red Black Tree.

void searchAndStore(RBTNode *root, int firstBuilding, int lastBuilding, string &rangeValues) // Searches & stores (In a reference variable) the buildings currently under construction in the given range from firstBuilding <= building <= lastBuilding.

RBTNode* search(RBTNode *root, int buildingNumber) // Returns the address of a Red-Black Tree node that has the building number given as input.

RBTNode* insertBST(RBTNode *root, RBTNode *ptr) // Insert helper function which inserts given node into Red-Black Tree using normal Binary Search Tree insert procedure.

RBTNode* insert(Building value) // Insert the Building plan in to the Red-Black Tree.

void deleteFromRBTree(int buildingNumber) // Deletes the given node with building number same as building number which is given as input.

risingCity.cpp – Main working Function:

1) Attributes:

   globalTime // Timer that is global to the entire work flow.

   Min_Heap min_heap(2000) // Min Heap declaration.

   RBTree red_black_tree // Red-Black Tree declaration.

   line // Input line from the file.

   localTime // Time at which input comes to wayne construction,

   operation // Either "Insert or PrintBuilding".

   read // Flag is set to one if there is an incoming command from the file and set to zero if the command is executed.

2) Member Functions:

   void insertBuilding(string line, ofstream &output) // Insert Building into Red-Black tree and Min Heap - Takes care of Insert.

   void printBuildingInfo(string range, ofstream &output) // Prints the building info from Red-Black Tree - Takes care of PrintBuilding(a, b) or PrintBuilding(a).

   void constructBuilding(ofstream &output) // Construct building function that takes care of Wayne Construction's construction work.

   int main(int argc, char *argv[]) // Starts processing the input file.

Work Flow of Main Function:

i) Starts processing the file and execute the first command. (Assumes first command is definitely Insert at globalTime == 0)

ii) Enters into a while loop till there are commands to be read from the input file.

a. If localTime <= globalTime
    i. Execute Command – Either PrintBuilding or Insert

b. If localTime > globalTime and min_heap.size() > 0
    i. Start constructing the building by taking the top from min_heap
        1. If globalTime == localTime during construction
            a. Check for incoming command and execute the command – Either PrintBuilding or Insert
            b. Take the next command from the input file
    ii. If executionTime == totalTime of the constructed building
        1. Remove the element from both the data structures
    iii. Else
        1. Modify the execution time of the constructed building in both the data structures.

c. If min_heap.size() == 0 at any point in time and there are more input commands to be executed from the input file
    i. Make globalTime = localTime
    ii. Take incoming command and execute it.

iii) while there are no more inputs to be read and min_heap.size() > 0:

a. Construct the buildings available for construction in min_heap by taking the top element from the min_heap

iv) Finish when min_heap.size() == 0 and there are no more commands to be executed from the input file.