COP 5536 Project Report

Maharshi Doshi
0533-9287
Maharshi.doshi@ufl.edu

Programming Language - C++

Files Created for the Project

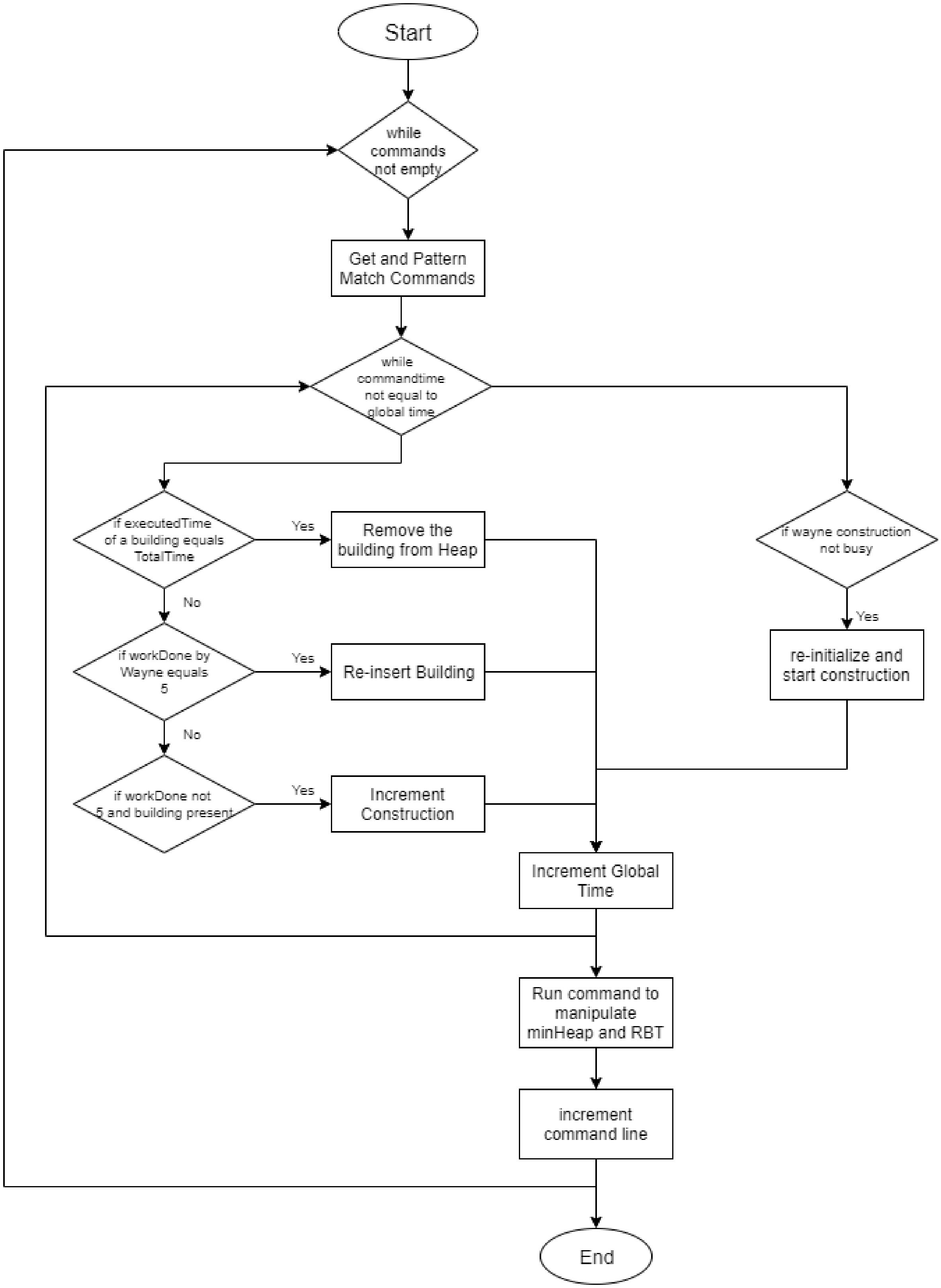
1. mainfile.cpp
2. minheap.cpp
3. minheap.h
4. redblacktree.cpp
5. redblacktree.h

mainfile.cpp – Contains the main flow of the program

- *int main(int argc, char *argv[])*
  - Both the Min Heap and Red-Black Tree are initialized at the beginning
  - Input File is read and the commands are stored in a vector
  - The commands are iterated over a while loop
  - The core logic of the program works here

- *void runInputLine(string cmd, string parseArgs, minheap* mh, redblacktree* rbt, ofstream& opF)*
  - Command (Insert/Display), parseArgs(values), Minheap object, RBT object and output file instance are sent as arguments into this function.
  - The command is recognized using regex pattern matching and the Insert and Display function runs accordingly.
  - If the command is "Insert", a new Red-black Node and min Heap node are created. The reference to red-black node is store in min heap array.

Flowchart of Business Lo


Start
while commands not empty
Get and Pattern Match Commands
while commandtime not equal to global time
if executedTime of a building equals TotalTime
Yes
Remove the building from Heap
No
if workDone by Wayne equals 5
Yes
Re-insert Building
No
if workDone not 5 and building present
Yes
Increment Construction
if wayne construction not busy
Yes
re-initialize and start construction
Increment Global Time
Run command to manipulate minHeap and RBT
increment command line
End

Min Heap [minheap.cpp and minheap.h]

Class Node: Used for declaring variables and constructor of Min Heap

Variables

- *cBuildingnum* – Building Number stored
- *cExecutedtime* – Defines the time for which work is done on the building
- *cTotaltime* – Total time required for the completion
- *cRBTnode* – Reference of the corresponding Red-black node

Function Definitions

- *bNode(long int buildingNum, long int executedTime, long int totalTime, redblacknode* rbn);*
  - Constructor of the class which is called whenever a new node is created.
  - All the variable values of min heap node are initialized in the constructor

Class MinHeap: Contains all the functions of Min Heap data structure

- *Void insertNode(bNode* value)*
  - Insert a new Node in the MinHeap Array
  - Bubbleup function called which conserves the min heap property of the structure

- *Void deleteMin()*
  - Deletes the minimum node from the Min Heap
  - Bubbledown function called which conserves the min heap property of the structure

- *bNode* getMinFromHeap()*
  - Gets the minimum node from the Min Heap

- *Void bubbleDown(int index)*
  - When the min node is deleted, we replace it with the last element.
  - If replaced element is greater than any of its child node, swap the element with its smallest child(Min-Heap) or with its greatest child(Max-Heap).
  - Do above step recursively.

- *Void bubbleUp()*
  - When a new element is inserted, if inserted element is smaller than its parent, swap the element with its parent.
  - Do the above step recursively.

Red-Black Tree [redblacktree.cpp and redblacktree.h]

Class redblacknode: Used for declaring variables and constructor of Red-Black Tree

Variables

- *cBuildingnum* - Building Number stored
- *cExecutedtime* - Defines the time for which work is done on the building
- *cTotaltime* - Total time required for the completion
- *cLeft* – Store the reference to left child
- *cRight* – Store the reference to right child
- *cParent* – Store the reference to parent
- *cColor*– Store the color of the node. (0 – black node, 1 – red node)

Function Definitions

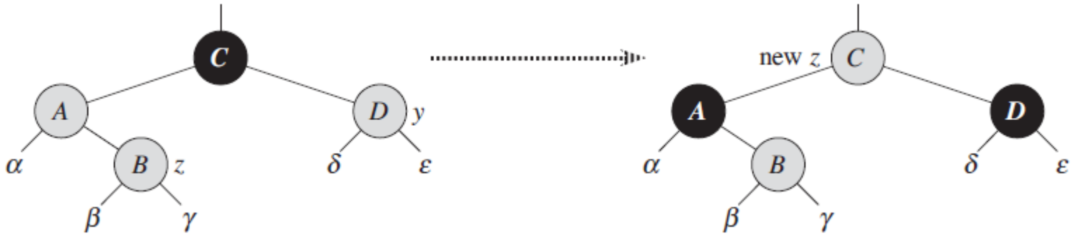
- *redblacknode(long int buildingNum, long int executedTime, long int totalTime, redblacknode* rbn);*
  - Constructor of the class which is called whenever a new node is created.
  - All the variable values of RBT node are initialized in the constructor

Class RedBlackTree: Contains all the functions of Min Heap data structure

- *void insertNode(redblacknode* node)*
  - Function for inserting a new node in RBT
  - The insertion is the same as that of Simple Binary Search Tree
  - After the node has been inserted, we fix the Insert by calling the reinsert function which fixes the RBT and conserves it's property.

- *void redblacktree::reInsert(redblacknode* node)*
  - This function is used to fix the imbalance in the RBT caused by the insertion of new node
  - There are 3 cases that are handled by this function

- Case1:

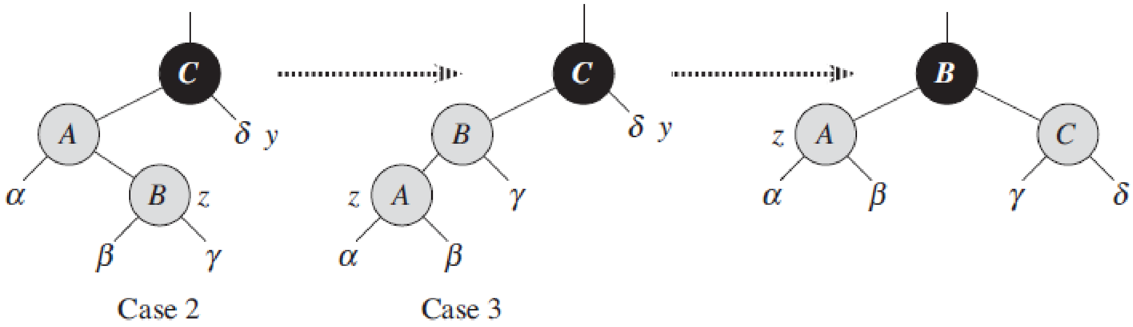


Picture Reference: CLRS

Explanation:

- Here z and its parent z->p are both red.
- Each of the subtrees has a black root, and each has the same black edges.
- The code for case 1 changes the colors of some nodes, preserving property 5. All downward paths from a node to a leaf have the same number of blacks.
- The while loop continues with node z's grandparent z->p->p as the new z.

- Case2 and Case3:



Explanation:

- In cases 2 and 3, the color of z's uncle y is black.
- In case 2, node z is a right child of its parent. We immediately use a left rotation to fix this issue.
- In case 3, we execute some color changes and a right rotation, which preserve property 5, and then, since we no longer have two red nodes in a row, we are done.

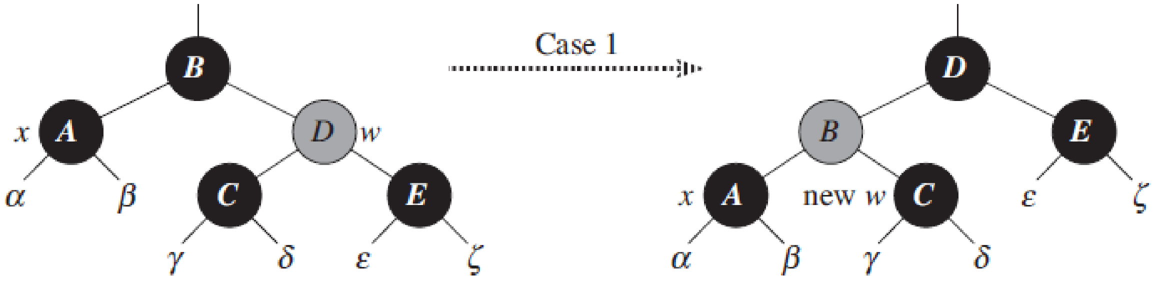
- *redblacknode* findMin(redblacknode* node)*
  - This function returns the minimum value node from the Red-Black Tree

- *void redblacktree::lRotate(redblacknode* node)*
  - Function performs left rotation in the RBT

- *void redblacktree::rRotate(redblacknode* node)*
  - Function performs right rotation in the RBT

- *Void deleteNode(redblacknode* node)*
  - A standard BST delete is performed at first
  - Following cases are handled:
    1. When we want to delete a node z which has fewer than two children, then is removed from the tree
    2. When a node z has 2 children
  - After deletion, redelete() function is called to fix the imbalances caused by the deletion.

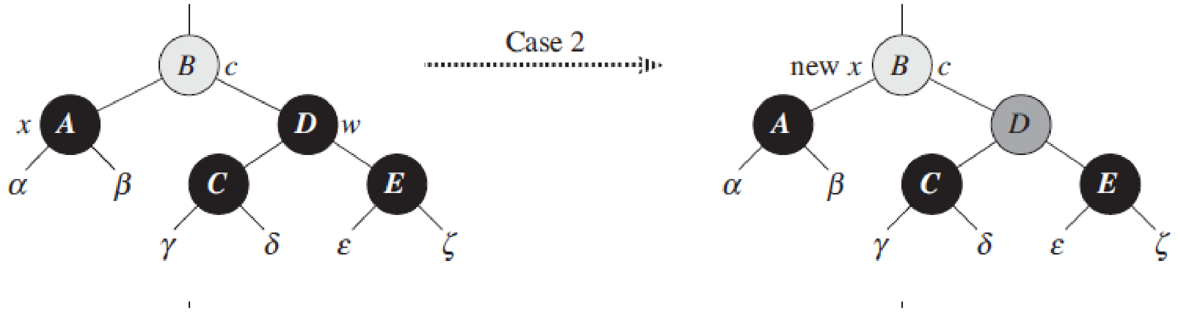
- *Void reDelete(redblacknode* node)*
  - Following Cases are handled in this function
    1. x's sibling w is red



Reference: CLRS

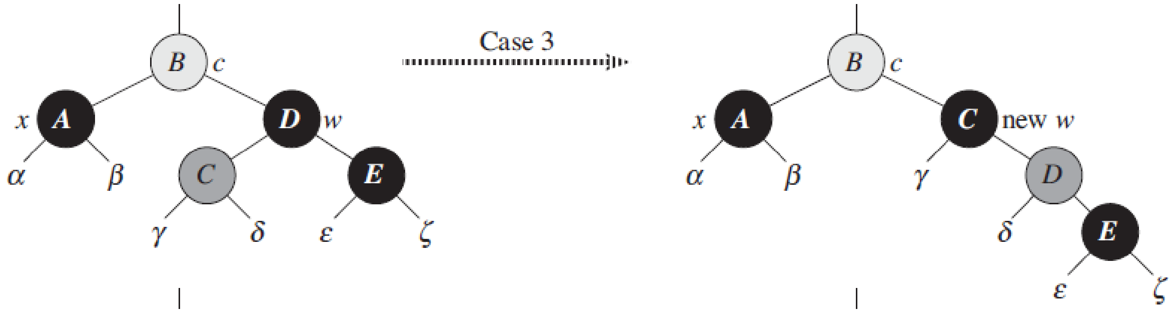
Here a Left rotation is performed in the redelete() function

2. x’s sibling w is black, and both of w’s children are black



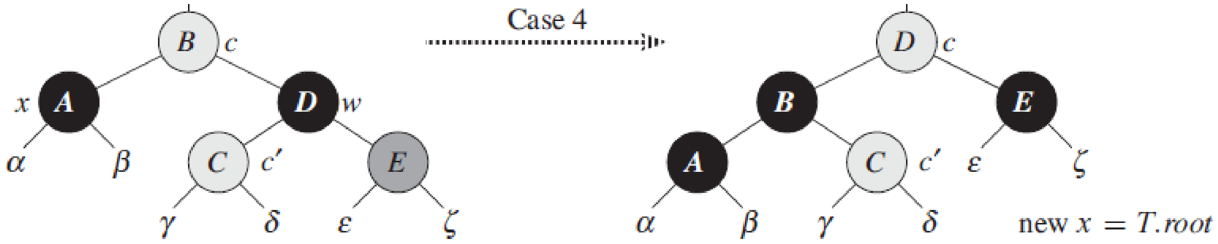
- Take one black off from x (making x singly black) and off w (making w red).
- Move that black to x.parent.
- Do the next iteration with x.parent as the new x.

3. x’s sibling w is black, w’s left child is red, and w’s right child is black



- Make w red and w's left child black.
- Then right rotate on w.
- The new sibling w of x is black with a red right child, and we go immediately into case 4.

4. x’s sibling w is black, and w’s right child is red



- Make w be x.parent's color (c).

  - Make x.parent black and w's right child black.
  - Then left rotate on x.parent.

All the cases are recursively fixed in a while loop till the RBT properties are not conserved.

- *Void rbtRelink(redblacknode* n1, redblacknode* n2)*
  - This function replaces one subtree as a child of its parent with another subtree.

- *Redblacknode* searchTree(long int bnum)*
  - This function takes a building number as input and searches it in the red-black tree
  - It calls searchTreeHelper() which is a recursive function

- *Void searchTreeInRange(long int bnum1, long int bnum2, vector<redblacknode*>* vect)*
  - This function takes two building numbers as input
  - It called searchTreeInRangeHelper() which is a recursive function that helps in finding the building numbers between the given inputs.

References:

- Introduction to Algorithms [CLRS]