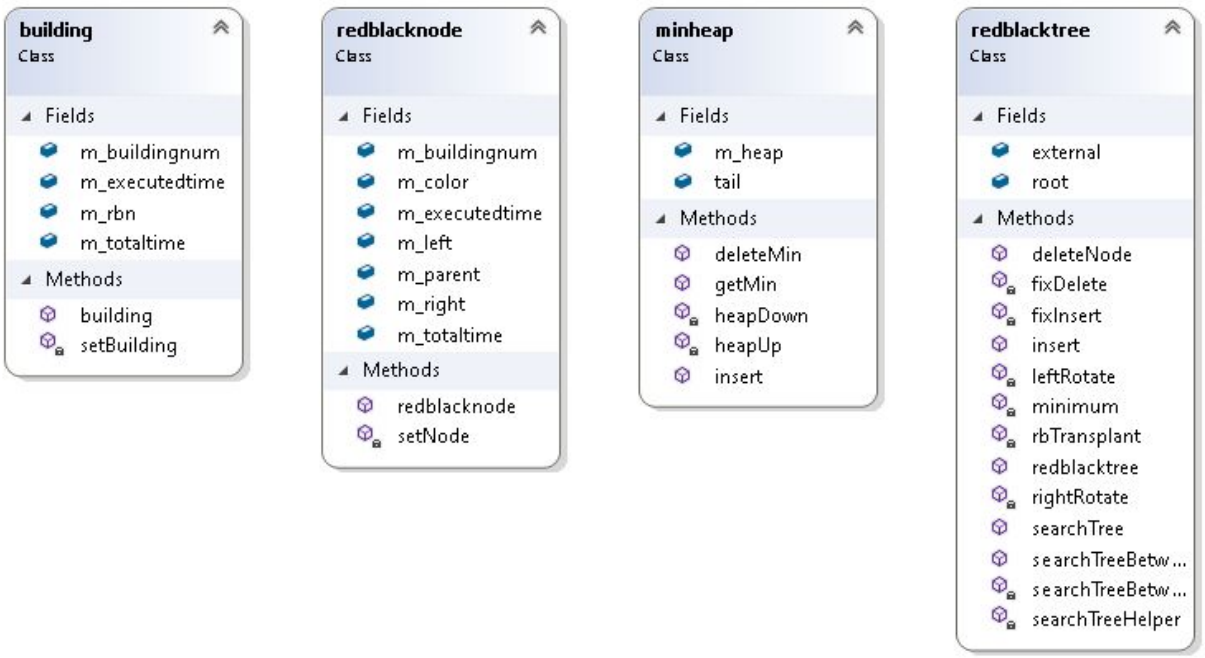
ADS Project - I Report

Author: Ajay Mittal
(UFID: 6017-1184)
ajay.mittal@ufl.edu

Source code consists of following files:

1. building.h & building.cpp - Defines building node (used as a minheap node).
2. minheap.h & minheap.cpp - Defines minheap.
3. redblacknode.h & redblacknode.cpp - Defines redblacktree's node.
4. redblacktree.h & redblacktree.cpp - Defines redblacktree.
5. main.cpp - Defines workflow of the whole program.



Building.cpp

Node structure:

long int buildingNum;
long int executedTime;
long int totalTime;
redblacknode rbn;* //pointer to the redblacktree's node.

Private member functions:

void setBuilding(long int buildingNum, long int executedTime, long int totalTime, redblacknode rbn);*

Sets building member variables provided with building number, execution time, total time of a building and the address of the corresponding redblacktree node.

Public member functions:

building(long int buildingNum, long int executedTime, long int totalTime, redblacknode rbn);*

Constructor of the building class. Takes in building number, execution time, total time of a building and the address of the corresponding redblacktree node. After that, calls the setBuilding to set member vars.

minheap.cpp
Property of minheap:
1. Execution time should be min.
2. In case, execution times are same, check for building num.

Private member functions:
void heapDown(int index);
Takes in an index (which is zero), and heapifies downwards towards leaf maintaining minheap property.

void heapUp(int index);
Takes in an index (which is the last index of active array), and heapifies upwards towards root maintaining minheap property.

Public member functions::
void insert(building value);*
Appends address of building node into the minheap array (added at index after last available index). heapUp is called after to maintain minheap property.

building getMin();*
Returns the first entry of minheap array which is basically the address of building node at root of minheap.

void deleteMin();
Replaces the first entry of minheap array with last entry and reduces array size by 1. Also, deletes the building node from the memory. Then, headDown is called to maintain minheap property.

redblacknode.cpp
Node structure:
long int buildingNum
long int executedTime
long int totalTime
int color // red - 1, black - 0
redblacknode parent*
redblacknode left*
redblacknode right*

Private member functions:
void setNode(long int buildingNum, long int executedTime, long int totalTime, int color, redblacknode parent, redblacknode* left, redblacknode* right);*
Sets redblacknode member variables provided with building number, execution time, total time of building, color of node, parent pointer, left pointer and right pointer).

Public member functions:
redblacknode(long int buildingNum, long int executedTime, long int totalTime, int color, redblacknode parent, redblacknode* left, redblacknode* right);*
Constructor of the redblacknode. Takes in all the above args and calls setNode to set member vars.

redblacktree.cpp

Private member functions:

void fixDelete(redblacknode node);*

Is called after deletion of any node to maintain the redblack property of the redblacktree.
Following cases are handled:

1. Case 1: node's sibling is red.
2. Case 2: node's sibling is black, and both of the sibling's children are black.
3. Case 3: node's sibling is black, sibling's left child is red, and sibling's right child is black.
4. Case 4: node's sibling is black, and sibling's right child is red.

Vice versa is node's sibling is black.

void rbTransplant(redblacknode node_a, redblacknode* node_b);*

When a node is deleted, it links the node's child (if any) to node's parent (if any).

void fixInsert(redblacknode node);*

Is called after insertion of any node to maintain the redblack property of the redblacktree.
Following cases are handled:

1. Case 1: node's uncle is red.
2. Case 2: node's uncle is black and node is a right child.
3. Case 3: node's uncle is black and node is a left child.

redblacknode searchTreeHelper(redblacknode* node, long int key);*

Returns the node of the redblacknode to be searched. Takes root pointer and the key as input.

void leftRotate(redblacknode node);*

Left rotates with node as pivot.

void rightRotate(redblacknode node);*

Right rotates with node as pivot.

redblacknode minimum(redblacknode* node);*

Returns the left-most leaf from the node.

void searchTreeBetweenHelper(redblacknode node, long int building1, long int building2,*
vector<redblacknode>* list);*

Returns the list of redblacknodes between building 1 and building 2. Input arg is root and building numbers and the list which is to be used.

Public member functions:

redblacknode root;*

Root of the tree.

redblacknode external;*

Node to represent external nodes.

redblacktree();

Constructor of redblacktree. Initializes external and then sets root as external.

redblacknode searchTree(long int buidingNum);*
Gets a building number and calls searchTreeHelper. Returns the searched node. Cases such as not found are handled in main.cpp.

void insert(redblacknode node);*
Gets a redblacknode and inserts in the redblacktree. fixInsert is called to ensure that the rb properties are satisfied.

void deleteNode(redblacknode node);*
Gets a redblacknode and deletes it from redblacktree. rbTransplant called to ensure link and fixDelete is called to ensure that rb properties are satisfied.

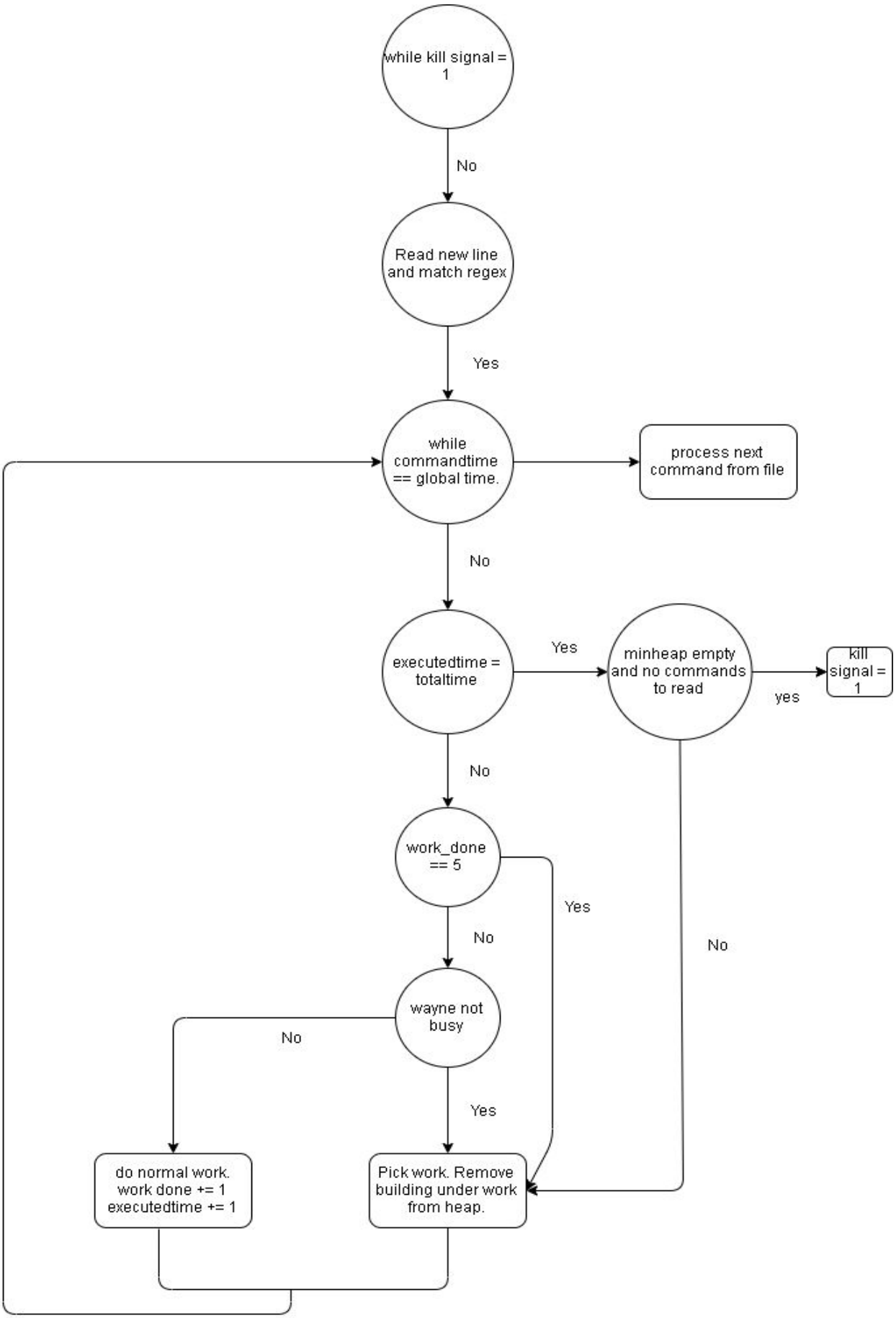
void searchTreeBetween(long int building1, long int building2, vector<redblacknode>* list);*
Gets two building numbers and a list. Calls searchTreeBetweenHelper. Returns list filled with pointers to redblacknodes which lie between b1 and b2 (included).

Main.cpp
Functions:
void processCommand(string command, string args, minheap mh, redblacktree* rbt, ofstream& outfile)*
Gets command, command arguments, pointer to minheap, pointer to redblacktree and outstream object. Inserts into tree and heap if its an insert command. If print is issued, calls searches the node in tree and returns it.

*int work(int argc, char** argv)*
Takes in input file from command line. The logic for wayne industry processing, command reads and global time are implemented in this function itself.
The execution flow is exactly as described in the project description provided.
It expects adherence to a specific command pattern to work, otherwise, it would throw error.
Following is the format:
T1: Insert(n1,n2)
T2: PrintBuilding(b1)
T3: PrintBuilding(b1,b2)

*int main(int argc, char** argv)*
Entry point of the program. Takes input file from command line.

FLOW CHART OF MAIN FUNCTION: (See next page)


while kill signal = 1
No
Read new line and match regex
Yes
while commandtime == global time.
process next command from file
No
executedtime = totaltime
Yes
minheap empty and no commands to read
yes
kill signal = 1
No
work_done == 5
Yes
No
wayne not busy
No
Yes
do normal work.
work done += 1
executedtime += 1
Pick work. Remove building under work from heap.
No