Advance Data Structure (COP-5536)

Assignment Project -1 Report

Priyam Selugar

UFID: 9423-1353

priyamselugar@ufl.edu

The project source code includes following files:

1) **Building.cpp and Building.h**:

   - This file defines node with building and its attributes that is BuildingNum,Execution_Time and Total_Time.

Building.cpp Structure:

- *long int building_Num*
- *long int executed_Time*
- *long int total_Time*
- *redblacknode*rb_pointer // pointer to red black node*

Member Functions of Building.cpp:

- *building(long int buildingNum, long int executed_Time, long int total_Time, redblacknode* rb_pointer);*

This method is a constructor of Building class with Building_Num,executed_Time,Total_Time as parameters. It invokes the *set_building* method to set values to these parameters.

- *set_building(buildingNum, executed_Time, total_Time, rb_pointer);*

This function sets data member variables of building(building_Num, executed_time and total_time) along with red_black node address.

2) **MinHeap.h and MinHeap.cpp**:

- Min_Heap is used to store Building_Num, exexcution_Time ,total_Time ordered by execution_Time.

- In case of same execution_Time , Building_Num is used as tie breaker

- MinHeap.cpp defines all the functionalities of a min heap that is it includes following methods and member variables:

Member Varibales of Min_Heap.cpp:

- *building* min_heap[MAX];//MinHeap Array of max_size 2000*
- *int leaf = 0;*

Member Functions of Min_Heap.cpp:

Heapify_up(int index):

- This method heapifies from bottom to top till the root maintaining the min heap property.

- Heapify_up(int index) is called corresponding to insert operation

Heapify_down(int index):

- Heapify_down(int index) heapifies from top to bottom , starting from index 0 till leaf maintaining the mon heap property.

- Heapify_down (int index)is called corresponding to delete operation.

- *insert_building(building* value)*:

  - This method adds address of building node to the min_heap after the last available index and updates the index pointer to point to the new inserted building.

  - Following which Hapify_up(int index) is called at current index to maintain min_heap property up to the root

- *get_minimum():*

  - Returns the minimum element which is the address of building node at first index of min_heap array.

- *delete_minimum():*

  - Delete_minimum() removes the topmost element of min heap and replaces it with the last leaf value , threby reduces array_size by 1.

  - Following which Heapify_down(int index) is called to perform heapify from top index till leaf until min_heap prpoperty is maintained .

3) **RedBalckNode.h and RedBlackNode.cpp:**

- Red black tree stores Building_Num, exexcution_Time ,total_Time ordered by Building_Num.

Member Varibales of RedBlackNode.cpp:

- *long int building_num;*
- *long int building_executed_time;*
- *long int building_total_time;*
- *redblacknode* rb_left;//left child pointer*
- *redblacknode* rb_right;//right child pointer*

- *redblacknode* rb_parent;//parent pointer*
- *int rb_color; // Assign colors: Red:1, Black:0*

Member Functions of RedBlackNode.cpp:

- *set_node(long int buildingNum, long int executed_Time, long int total_Time, int color, redblacknode* parent, redblacknode* left, redblacknode* right):*

  - This method sets member variables of red_black_node which are building_num,executed_Time,total_Time,rb_left,rb_right,rb_color(0 or 1)

- *redblacknode(long int buildingNum, long int executed_Time, long int total_Time, int color, redblacknode* parent, redblacknode* left, redblacknode* right):*

  - This is constructor to redblacknode.cpp which contains function call to *set_node(long int buildingNum, long int executed_Time, long int total_Time, int color, redblacknode* parent, redblacknode* left, redblacknode* right)*

4) RedbalckTree.h and RedBalcktree.cpp:

- RedBlackTree.cpp defines all the functionalities of a red black tree that is it includes following methods and member variables:

Member Varibales of RedBlackTree.cpp:

- *redblacknode* root;*
- *redblacknode* external;*

Member Functions of RedBlackTree.cpp:

- *void delete_rebalance(redblacknode* node):*

  - This function is called after deletion of any node to compensate for the delete imbalance and thus maintain red black tree properties

- Deletion of Node from red black tree has following cases:

  a) Deleted Node's Sibling is Red

  b) Deleted Node's Sibling is Black, left and right pointers of sibling's children are black

  c) Deleted Node's Sibling is Black, left pointer of sibling's child is black and right pointer of sibling's child is red

  d) Deleted Node's sibling is black, left pointer of sibling's child is red and right pointer of sibling's child is black

  Similarly, four other cases are handed when deleted node's sibling is black.

- *void rb_rejoin(redblacknode* node_a, redblacknode* node_b):*
  - After node deletion this function rejoins/relinks the children of deleted node to deleted node's parent.

- *void insert_rebalance(redblacknode* node):*
  - Following method manages all the new insert operations into red black trees.

  i)Node's uncle is red color

  ii) Node's uncle is black color and node is right child

  iii)Node's uncle is black and node is left child

- *redblacknode* search_tree_helper(redblacknode* node, long int key):*
  - This method returns address of red black node that is to be found.

- *void left_rotate(redblacknode* node);*

- Performs left rotation with node as center to rebalance the red black tree during insert and delete cases and maintains red black properties.

- *void right_rotate(redblacknode* node):*
  - Performs left rotation with node as center to rebalance the red black tree during insert and delete cases and maintains red black properties

- *redblacknode* min_left(redblacknode* node):*
  - This method is used to find and return left most leaf from the red black tree to handle delete cases from non-leaf node.

- void search_tree_middle_helper(redblacknode* node, long int building1, long int building2, vector<redblacknode*>* list):
  - Performs range search from Bulding1 to Building 2 both included. building1, building 2 are parameters to function along with the list that is returned containing the range.

- *redblacktree():*
  - Constructor of red black tree class to set root and external node values.

- *redblacknode* search(long int buidingNum):*
  - Invokes serach_tree_helper() corresponding to building_num as parameter during function call.

- *void insert(redblacknode* node):*
  - New building node are added to red black tree and insert_reblance() is invoked to maintain the red black property following every insert of building_num to tree.

- *void delete_node(redblacknode* node):*
  - Building_Num node is deleted and delete_rebalance() is called to ensure red black tree properties are followed.

- Also rb_rejoin() method is called to relink the pointers of deleted node's children and parent.

- *void search_tree_miidle(long int building1, long int building2, vector<redblacknode*>* list):*
  - Makes a function call to search_tree_middle_helper() and returns a list of nodes from red black tree that lie in range from building1 to building 2 which are parameters to this function.

5) Functions of Main.cpp:

- *void process_input_command(string input_command, string args, minheap* min_heap_pointer, redblacktree* rbt_pointer, ofstream& outfile):*

  - process_input_command takes command line arguments, command from input and inserts into minheap and red black tree if command was insert.

- *int main(int argc, char** argv):*

  - This is the main function which read input file from command line at every point.
  - It performs the following operations as described in the problem statement

i. Print (buildingNum) prints the triplet buildingNume,executed_time,total_time.
ii. Print (buildingNum1, buildingNum2) prints all triplets bn, executed_tims, total_time for which buildingNum1 <= bn <= buildingNum2.
iii. Insert (buildingNum,total_time) where buildingNum is different from existing building numbers and executed_time = 0.

- ❖ Main method includes the logic for the wayne industries :
- ❖ Wayne Construction works on one building at a time.
- ❖ Following is Algorithm Flow

Begin:

While command is not empty:

Perform pattern match of commands from input

While command time is not equal to global time counter:

If executed_time of building == total_Time:

Remove the Building from heap

Increment Global Counter

Else if work done by Wayne ==5:

Re-Insert Building

Increment Global Counter

If Wayne Construction Not busy:

Reinitialize and begin construction

Increment Global Counter

Run Command to change min heap and red black tree.

Increment Command Line

End