# Advanced Data Structures (COP5536)
# Project Report

**Akash Jajoo**

**UFID: 61326882** **akashjajoo@ufl.edu**

## 1. Description

The goal of the project is to find the completion time of each building being constructed by Wayne Enterprises and to print the buildings in the range or a specific building when Print command is inputted. The input to the project is a text file which has two kinds of operations i.e. Insert and Print. When Insert command comes, we have to put a new building to be constructed. When a Print command comes, we have to print the current execution time and building number of the buildings in the range of the Print function or the specific building which is to be printed.

The data structures used for this project are Min Heap and Red Black Tree. Min Heap is used to store all the buildings and execute the least executed time building or the lower building number (in case of a clash between least executed time). Red Black Tree is used to print the current status of the executed time of all the buildings when a Print command comes as an input. Red Black Tree is a self-balancing Binary Search Tree (BST) hence it makes printing the status of the buildings faster.

The project has been implemented in Java. The input to the program is a .txt file which has a series of Insert and Print commands. The Java BufferedReader and BufferedWriter are used to read the input file and write the output in a file respectively.

## 2. Running Instructions

To execute the program, unzip the file named “**Jajoo_Akash.zip**”.

From command line go the specific folder where the file was unzipped.

The following input syntax is required via command line-

```
java risingCity input.txt
```

The input file should be in the same directory as the program.

An output file named “**output_file.txt**” is created in the same folder as the program.

## 3. Program Structure

### <u>Class Main-</u>

The main class takes the input file into a BufferedReader and creates a BufferedWriter to write into the output file. The risingCity class constructor is called with a max size of 2001 as mentioned in the problem statement. An array list is created to store all the insert commands and put them in the Heap. The first insert is done outside the main loop of the main class. The first value is inserted in the Min Heap and RBT. After the first line, the code enters the main while loop in the main class, the terminating condition of this loop is while the input file has a new line or if there are any unfinished building. First, it is checked if there are any remaining building to be inserted in the ArrayList when all the buildings in the minheap are done executing, then it inserts them into the minheap and the minheap in min heapified. Now, when the buildings finish executing and they are at the top then the top building is removed from the data structures and printed in the file with the completion time. When the executed time of the root of the minheap is a multiple of 5 and it is not equal to 0 then it checks if there any pending inserts in the wait list (ArrayList), it inserts them and calls the minheap function to heapify the entire structure. The executed time of the root of minheap is incremented by 1. Now, it checks if there is a next line in the input file, if yes then the next line of the file is read and it checks if it is a print or an insert command. If it is a Print command and the data structures are not empty then it prints the required buildings as per the input and problem definition. If the data structure are empty then it prints (0,0,0) as there are no buildings currently in the data structures. If it is an insert command then the Building is put in the wait list and it is immediately put in the Red Black Tree. At the end, global counter is incremented by 1. After the code exits the while loop, the output file is closed.

## Class Building-

- **Description:** It creates a constructor to store the building number, executed time and total time of the buildings. The class has variables named bNo, etime and ttime which correspond to Building Number, Total executed time and Total time to be executed of each building.
- **Parameters:** int bNo assigns the building number

  int etime assigns the executed time as 0

  int ttime assigns the total time
- **Return value:** void

## Class risingCity-

- **Description**: This class has a constructor of rising city which calls the buildings constructor and assigns a specific position to every building structure and stores it a Building array called Heap. It also, defines the maximum size of the heap and the current size of the heap. The top of the heap is always the 1st index of the array, the 0th index of the array stores a garbage value as this makes the calling of other functions of minheap easier.
- **Parameters:** Building [] Heap makes Heap an array of Building structure
  int size gives the current size of the heap
  int maximum assigns the maximum value of the heap which is set as 2001.
  int front assigns the root of the min Heap to be the 1st element of the Heap array.
  int R = 0 assigns the red colour of the RBT to be 0.
  int B = 1 assigns the black colour of the RBT to be 1.
  RBT empty creates a constructor of class RBT with values (-1,-1,-1) i.e. when the RBT is empty.
- **Return value:** void

# Implementation of Red Black Tree

## RBT class-

- **Description:** RBT class is a constructor which defines the colour, left, right and parent of a node. It also makes an element of Building structure which is accessed by point by reference.
- **Parameters:** Building elem points to the Building structure

  int colour = B assigns the colour to a new node

  RBT l = empty as it has no left child. It is a child pointer.

  RBT r = empty as it has no right child. It is a child pointer.

  RBT uppernode = empty as it has no parent yet. It is a parent pointer.

- **Return value:** void

## Function RBT find-

- **Description:** It finds a node which is to be found while performing any operation of the Red Black Tree. It recursively calls RBT find to find the exact node. It iterates the left subtree if the node to be found is less than the root. It iterates the right subtree if the node to be found is more then the root. It returns the root if the current is equal to the node to be found.
- **Parameters:** RBT find that is the node to be found
  RBT current that is the root of Red Black Tree
- **Return value:** RBT

## Function insertrb-

- **Description**: It inserts the new node in the Red Black Tree. It inserts the new node as a red node and recursively calls the same function to find the perfect position in the RBT to put the node. It calls the fixinsertion function to fix the rank of the nodes.
- **Parameters:** RBT current is the node to be inserted

  RBT temp is a temporary empty node

- **Return value:** void

## Function fixinsertion-

- **Description**: It fixes the ranks of the nodes when a new node is inserted of any node is deleted from the RBT. It calls rotate right or rotate left according to the conditions.
- **Parameters:** RBT current is the node to be fixed
  RBT aunt is initially empty but it is used to find the colour of the sibling of the parent of the child.
- **Return value:** void

## Function rL-

- **Description**: It rotates the current node to the left according to the according to the conditions.
- **Parameters:** RBT current points to the node to be rotated.
- **Return value:** void

**Function rR-**

- **Description:** It rotates the current node to the right according to the according to the conditions.
- **Parameters:** RBT current points to the node to be rotated.
- **Return value:** void

**Function transplant-**

- **Description:** It connects the tree when a node is deleted and two separate trees are formed.
- **Parameters:** RBT target is the node deleted
  RBT w is the parent of the child where the subtree of target should be connected.
- **Return value:** void

**Function printRange-**

- **Description:** When a print command(range) comes, this function is invoked which in turn calls the recurprint function and returns the output string to the main class.
- **Parameters:** int b1 that is the start of the range

  int b2 that is the end of the range

  RBT cur that is pointing to the root

  String s is the output string to be returned

- **Return value**: String

**Function recurPrint-**

- **Description:** It recursively finds the nodes which are in the range of the print command and appends them in a string and returns it.
- **Parameters:** RBT curr which is the current node to be checked

  int b1 is the lower limit of the range

  int b2 is the upper limit of the range

- **Return value:** String

**Function inRange-**

It checks if the node is in the range.

**Function deleterb-**

It deletes the node from the tree and calls the transplant function to find the parent of the child of deleted node. It takes the node to be deleted as a parameter.

**Function deletefixup-**

It fixes the ranks of the nodes when a node is deleted. It calls the rotate left or rotate right according to the condition satisfied.

**<u>Function treeMinimum-</u>**

Finds the left most child i.e. the smallest element in the tree.

**<u>Function printspecific-</u>**

When a print command comes to print a particular building it finds it in the BST and returns its values.

**<u>Function insertrbt-</u>**

It creates a new node and calls the insertrb function.

**<u>Function deleterbt-</u>**

It deletes a node and calls the deleterb function.

# Implementation of Min Heap

**Function top-**

Returns the top of the minheap.

**Function lc-**

Returns the left index of the current node

**Function rc-**

Returns the right index of the current node

**Function isLeaf-**

Checks if the current node is a leaf node or not.

**Function nodeswap-**

Swaps the two nodes of the heap

**Function Heapify-**

If the node is not a leaf then it checks whether the left and right child both exists of the current node. If both of them exists then we check which one has lower execution time amongst the parent, left and right child and swaps it with the root and calls the heapify function with the node with lower execution time. In case of a clash between execution time, it sorts it according to building no. All the edge cases are handled in the function.

**Function insertminheap-**

It inserts the node at the bottom and then minHeapify is called from the main class itself.

**Function minheap-**

It recursively calls the heapify function from size/2 and goes to i=1 i.e. to the top of the heap.

**Function remove-**

It pops the top of the tree and reduces the current size of minheap and calls the heapify function from the root of the tree. It returns the popped value to the main class.