Advanced Data Structures (COP 5536)

Fall 2019

Name:	Prajan Tikayyolla
UFID:	6690-9443
Email Address:	prajantikayyolla@ufl.edu

Flow of the project:

Input to the program is given from file input.txt, each line in the input file contains a global time and the program reads input only when the local timer is equal to global time. Once the line is scanned then we will check for the operation mentioned in the line and the building variables. The operation can be to insert a building or to print building/buildings.

If the command is to insert then we first must check if there exists a building with same building number if so then we write an error with building number and local time and terminate the program. If this is a new building then insert it into min-heap keyed on execution time and into RBT keyed on building number.

Inserting into min-heap gets done with ease but inserting into RBT might violate the properties of red-black tree and these are fixed by a sequence of rotations and color swapping which are performed by a function.

If the command is to print then we check the parameters, if only one parameter is passed then we search for that building in heap and write into the file the building number and its execution time, there exists a case where the building to be printed is the current building which we extracted from min heap so that case is handled by taking time from current building (i.e. local time). If two parameters are passed then we need to find the buildings in the given range from the tree and write it to the file. If there exists no building then we write (0,0,0) to file.

Once we have a building in heap then our construction starts by picking the building with minimum execution time from the heap and start working on it for 5 units or until complete whichever is minimum. For every unit execution we increase the local counter timer and check for any inputs at that time.

If the current building completes during the 5 units slot, it will be written into a file with building number and day of completion which is the local counter time. We must delete the building from the tree which might lead to a violation, if the node to be deleted is black, then we fix those violations by performing a sequence of rotations and color swapping under careful implementation for every possible case.

If the building is not yet completed after the 5 units slot then we reinsert the building into min heap and perform percolate up so that it goes to its correct position with updated execution time.

Once we process all the inputs from the input file then we just need to complete the construction of buildings that are present in the heap. Once all the buildings are completed then we write the completion time of the city which the completion time of the last executed building.

Functions Used:

- public void wayneConstructionProject (String [] args)
  This function matches local timer with the global time and if it matches it will execute the corresponding function associated with the command (insert or print).
  If the local timer is not matching with the global time then works on already inserted buildings.
  Parameters: String [] args (file name)
  Return: None
- private void constructBuilding (String [] inputValuePair)
  This function creates node to both red black tree and min heap and inserts into them. It maintains pointers from red black tree to min heap and vice versa.
  Parameters: String [] inputValuePair (buildingNumber, totaltime)
  Return: None
- private void Construction()
  This function gets building with least execution time from heap and starts working for a duration of 5 or until complete, whichever is lower.
  If completed during that period it is written into file and deleted from tree.
  If the duration is completed and still the building is not completed then insert back into heap.
  Parameters: None
  Return: None
- private void incrementLocalTime()
  This function increments the local counter and current building execution time.
  Parameters: None
  Return: None
- private void finalPhaseConstruction()
  This function is triggered when all input commands are executed but still we have some buildings in heap/tree, so it completes construction of remaining buildings.
  Parameters: None
  Return: None
- private void printBuilding(String[] inputValuePair)
  This function checks the parameter, if it contains only 1 value then that building details are printed with the help of printBuildingHelper and if it contains 2 values then buildings in that range are printed and if there exists no matching building the empty record (0,0,0) is printed.
  Parameters: String[] inputValuePair (building number)
  Return: None
- private void printBuildingHelper(RbNode node)
  This function prints the building details for given node which will be passed from printBuilding function when asked to print specific building details.

Parameters: RbNode node (building node)
Return: None

- public void insert(HeapNode p)
  This function insert building into min heap based on execution time in case of tie it checks with the building number. It inserts at the end of the heap and percolates it up until it finds its correct position.
  Parameters: HeapNode p (building node)
  Return: None
- private void heapify(int i)
  This function is called when we delete min node, last node is placed at root and we do percolate down of the root until it finds its correct position.
  Parameter: int I (index)
  Return: None
- public HeapNode getBuilding()
  This function returns building with least execution time which is the root of the min heap.
  Parameter: None
  Return: Building of type HeapNode is returned.
- private void swap(int i, int j)
  This function swaps buildings in min heap.
  Parameter: int i, int j (index of buildings in heap)
  Return: None
- public boolean isEmpty()
  This function checks whether heap is empty or not.
  Parameter: None
  Return : True/False
- private RbNode getBuilding(RbNode root, int buildingNumber)
  This function searches for building in tree with passed building number.
  Parameters: RbNode root (root of rbt), int buildingNumber (building number)
  Return: Returns RbNode of that corresponding buildingNumber
- public List<RbNode> getBuildingsInRange(int buildingNumber1, int buildingNumber2)
  This function creates a list and passes the root of rbt tree and list to helper function.
  Parameters: int buildingNumber1 (building number), int buildingNum (building number)
  Return: Return list of type RbNode after helper function adds buildings to list
- private void getBuildingsInRange(RbNode root, List<RbNode> list, int buildingNumber1, int buildingNumber2)
  This function searches for buildings in the given range in tree and adds to list.
  Parameters: RbNode root (rbt root), List<RbNode> list (buildings) , int buildingNumber1 (building number), int buildingNumber2 (building number)
  Return: None

- private RbNode rotateRight(RbNode pp)
  This function is called when left violation occurs and it fixes that violation by rotating child with respect to parent.
  Parameters: RbNode pp (rotation around violated nodes parent)
  Return: Child Node of pp after rotation and updating new parent and child.
- private RbNode rotateLeft(RbNode pp)
  This function is called when right violation occurs and it fixes that violation by rotating child with respect to parent.
  Parameters: RbNode pp (rotation around violated nodes parent)
  Return: Child Node of pp after rotation and updating new parent and child.
- public void insertNode(RbNode p)
  This function calls two more functions one to insert into rbt and other to fix the insertion if any violation.
  Parameters: RbNode p (building to be inserted)
  Return: None
- private void insertHelper(RbNode root, RbNode p)
  This function inserts given node into rbt based on building number, it works as a binary search tree and inserts at correct position.
  Parameters: RbNode root (root of rbt), RbNode p (building to be inserted)
  Return: None
- private void insertFix(RbNode p)
  This function fixes any violation caused during insertion i.e. 2 red nodes in sequence.
  Parameters: RbNode p (building to be inserted)
  Return: None
- private void Transplant(RbNode a, RbNode b)
  This function is useful in case of deletion to swap nodes i.e. when parent is deleted child take that place or its predecessor.
  Parameters: RbNode a (building), RbNode b (building)
  Return: None
- public boolean delete(int buildingNumber)
  This function deletes a node in rbt with given building number and replaces it with child or predecessor.
  Parameters: int buildingNumber (building number)
  Return: True/False
- private void deleteFix(RbNode py)
  This function is called when deleted node is black. It finds which case it belongs to performs fixation until there is no more violation.
  Parameters: RbNode py (building to be fixed)
  Return: None

- private void swapColors(RbNode pp, RbNode gp)
  This function is used to swap colors of two buildings while fixing violations.
  Parameters: RbNode pp (p's parent), RbNode gp (p's grand parent)
  Return: None