COP 5536 - Advanced Data Structures
Project Report

UFID	3735-1629
Name	Surya Tejaswini Kandavilli
Email	skandavilli@ufl.edu

The implementation is divided into 5 classes and each functionality is explained below in detail. The input and output requirements are also handled by the below classes.

1.1 HeapNode

Creates min Heap nodes every time an insert command is executed. It has a node structure defined and stores data during execution of insert command.

Class	Node Structure	Description
public class HeapNode	int bNum;	Unique integer identifier for each building
	int exTime;	Total number of days spent so far on this building
	int toTime;	Total number of days needed to complete the construction of the building
	private RBTNode ref;	Reference pointer to maintain correspondence between the min-Heap and RBT nodes

Function calls like getters and setters are used to get and update variables during construction and command execution.

Class	Function calls	Description
public class HeapNode	public HeapNode (int bNum, int exTime, int toTime)	Default constructor. Invoked every time a min heap node is created. Takes input parameters and creates a min-Heap node with the defined values. Additionally, assigns a default "nil" reference value to RBTNode ref
	public int getBN ()	Getter function. Used to access "Building number" of the node.
	public int getET ()	Getter function. Used to access current "Execution time" of the node.
	public int getTT ()	Getter function. Used to access "Total time" of the node.
	public RBTNode getRef ()	Getter function. Used to access the RBT reference node corresponding to the min-Heap node.
	public void setET (int newET)	Setter function. Used to update the "Execution time" of the node.
	public void setRef (RBTNode ref)	Setter function. Used to update the RBT reference for the current min-Heap node.

1.2 RBTNode

Creates RBT nodes every time an insert command is executed. It has a node structure defined and stores data during execution of insert command.

Class	Node Structure	Description
public class HeapNode	int bNum;	Unique integer identifier for each building.
	int exTime;	Total number of days spent so far on this building.
	int toTime;	Total number of days needed to complete the construction of the building.
	Boolean colour;	Denotes the RBT node's colour.
	RBTNode left;	Reference pointer to left child of current node.
	RBTNode right;	Reference pointer to right child of current node.
	RBTNode parent;	Reference pointer to parent of current node.

Function calls like getters and setters are used to get and update variables during construction and command execution.

Class	Function calls	Description
public class RBTNode	RBTNode (int bNum, int exTime, int toTime, boolean col)	Default constructor. Invoked every time an RBT node is created. Takes input parameters and creates an RBT node with the defined values.
	public int getBN ()	Getter function. Used to access "Building number" of the node.
	public int getET ()	Getter function. Used to access current "Execution time" of the node.
	public int getTT ()	Getter function. Used to access "Total time" of the node.
	public void setET (int newET)	Setter function. Used to update the "Execution time" of the node.

1.3 minHeap

Creates min Heap nodes every time an insert command is executed. Maintains an array of node references to maintain the min heap structure. The minimum element is stored at index 0.

Class	Node Structure	Description
public class minHeap	private HeapNode [] heapList;	List containing min heap node pointers.
	private int len;	Used to indicate size of min-Heap.
	private int capacity;	Used to indicate the maximum possible capacity of min heap.
	public RBT rbt = new RBT ();	Creating an object of RBT class in order to access functions implemented in RBT.

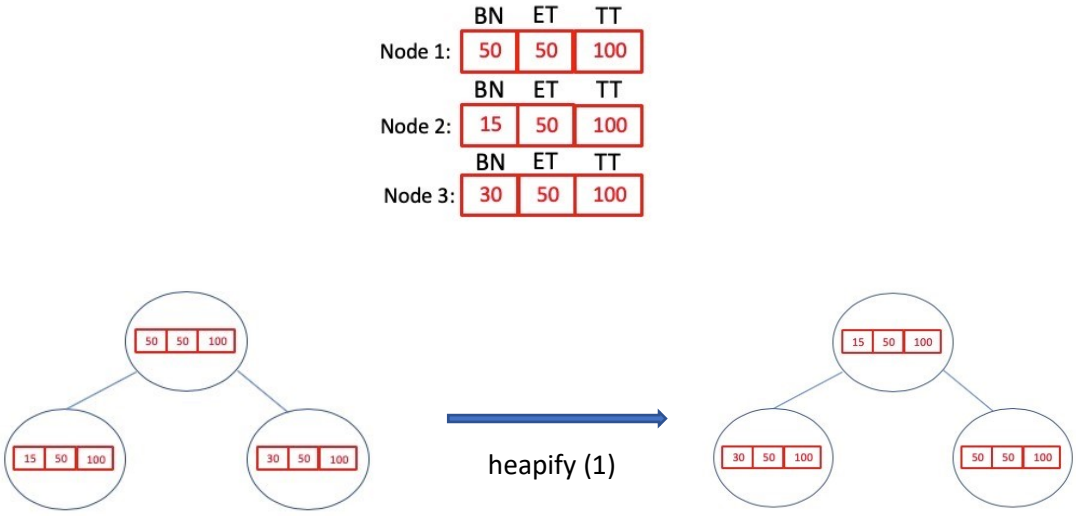

Figure: Demonstrates an example of how heapify sorts the node entries in min Heap, BN-Building number ET-Execution time TT- Total time

Min heap stores nodes in increasing order of their execution time, in case a tie occurs between two nodes, the building number is used to sort them. Every min heap node has a reference of the corresponding RBT node stored in its *"ref"* variable. Min heap "*insert*", internally invokes RBT node insert and only upon successful insert in RBT min heap insert is performed. This ensures that there are no duplicate entries of a node, i.e. no two nodes can share same building number. Heap array consisting of node references is sorted every time a new node is inserted; this endures that the "*getMin ()*" function always return the updated min heap node. "*heapify*" is invoked after every min element deletion, as it internally sorts the nodes based on their execution time. It also maintains min heap properties such as – (a) parent having minimum value compared to its children, (b) Left child having lesser execution time compared to right child. (c) In case of a tie, same action is done based on "*building number*". Above figure shows an example of how heapify works on the min heap.

Class	Function calls	Description
public class minHeap	`public minHeap (int max)`	Default constructor, initializes min-Heap array size and adds first node to the heap.
	`private void exchange (int child, int parent)`	Swaps index values of two heap nodes.
	`public int insert (int BN, int ET, int TT)`	Inserts a new node with user specified data.
	`public RBTNode RBTNodeRef (int BN, int ET, int TT)`	Inserts an RBT node and returns the newly inserted node's reference.
	`public void heapify (int positionNode)`	Parent should have minimum value compared to its children Minimum value of the entire heap is stored as the root at index 0. Left child has lesser execution time compared to right child. In case of a tie, building number is used to resolve the tie and all the above three rules are applied using the building number.

	`public void deleteMin ()`	Delete the root node which has the minimum execution time on the Heap. After deletion sort the heap to update arrangement.
	`public void deleteRBTNode (HeapNode min)`	Accesses RBT node by referring to the min-Heap minimum node data, then deletes the RBT node from RBT.
	`public HeapNode getMin ()`	Returns the node with minimum execution time.
	`public void updateET (HeapNode node, int newET)`	Updates execution time in the RBT node corresponding to the specified min-Heap node.
	`public String print (int BN)`	Accesses RBT and prints the triplet (Building number, Execution time, Total time) for the specified node.
	`public String print (int BN1, int BN2)`	Accesses RBT and prints range of triplets for which BN1 <= bn <= BN2.

1.4 RBT

Is invoked within the minHeap "*insert*" function every time a new node is inserted. The functions resolve all the imbalances occurring during insertion and deletion of a node. Insertion and deletion are done using building number of the node as the identifier.

Class	Node Structure	Description
`public class RBT`	`private final boolean Red = false;`	Declare global entity.
	`private final boolean Black =true;`	Declare global entity.
	`private final RBTNode nul = new RBTNode (-1, -1, -1, Black);`	Create a nul node containing dummy data.
	`private RBTNode rootNode = nul;`	Initialize root to nil node.

Function calls are defined to handle all the cases occurring during insertion and deletion of the node. The methods discussed in class have been implemented for handling imbalance scenarios.

Class	Function calls	Description
`public class RBT`	`public RBTNode insertRBT (int bNum, int exTime, int toTime)`	Returns a RBT node reference consisting input parameters if the insertion was successful. Returns nul node if there are duplicate entries.
	`private void balanceRBT (RBTNode buildingNode)`	Balances the RBT structure to include the newly inserted node without violating any properties. Implements RRr, RLr, LRr, LLr and invokes RLb, RRb, LRb, LLb when required.
	`private void RLb (RBTNode buildingNode)`	This function is invoked when, pp is right child of gp, buildingNode is left child of pp and z is black.
	`private void RRb (RBTNode buildingNode)`	This function is invoked when, pp is right child of gp, buildingNode is right child of pp and z is black.

public class RBT	private void LRb (RBTNode buildingNode)	This function is invoked when, pp is left child of gp, buildingNode is right child of pp and z is black.
	private void LLb (RBTNode buildingNode)	This function is invoked when, pp is left child of gp, buildingNode is left child of pp and z is black.
	private void RRight (RBTNode buildingNode)	Right rotation is performed when this function is invoked.
	private void RLeft (RBTNode buildingNode)	Left rotation is performed when this function is invoked.
	public void deleteNode (RBTNode buildingNode)	Deletes a node from the tree and resolves imbalance. Node being deleted falls into three categories: degree 0, degree 1 and degree 2 After deletion in order to balance the RBT properties, rebalanceRBT function is invoked.
	private void resolveDelete (RBTNode replaceNode, RBTNode deleteNode, RBTNode rightNode)	This is invoked when a two-degree node is deleted and the smallest node from its right subtree is used to. Replace the deleted node, all the dependencies are handled during replacement.
	private void rebalanceRBT (RBTNode y, RBTNode py)	This function is invoked when the RBT needs to be balanced to satisfy all the properties it exhibits. Based on the imbalance functions like Lr0, Lb0, Lb1_case1, Lb1_case2, Lb2, Rr0, Rb0, Rb1_case1, Rb1_case2, Rb2 are invoked.
	private void Lb2(RBTNode v, RBTNode py,RBTNode y)	This function is invoked when y is Left child of py, colour of v node is Black and v node has 2 red children.
	private void Rb2(RBTNode v, RBTNode py, RBTNode y)	This function is invoked when y is Right child of py, colour of v node is Black and v node has 2 red children.
	private void Lb1_case1(RBTNode v, RBTNode py, RBTNode y)	This function is invoked when y is Left child of py, colour of v is Black and v node's left child colour is Red and v's right child colour is Black.
	private void Rb1_case1(RBTNode v,RBTNode py,RBTNode y)	This function is invoked when y is Right child of py, colour of v is Black and v node's left child colour is Black and v's right child colour is Red
	private void Lb0(RBTNode v)	This function is invoked when y is Left child of py, colour of v is Black and v doesn't have any red colored children.
	private void Lr0 (RBTNode v, RBTNode py)	This function is invoked when y is Left child of py, colour of v is Red.
	private void Rr0 (RBTNode v, RBTNode py)	This function is invoked when y is Right child of py, colour of v is Red.
	private void Rb0 (RBTNode v)	This function is invoked when y is Right child of py, colour of v is Black and v doesn't have any red children.
	private RBTNode getMinNode (RBTNode parentRoot)	Returns minimum RBT node, min node is the node that has smallest building number.
	public String print (int BN)	This function takes building number as input and returns a triplet consisting of (building number, execution time, total time) corresponding to the building number specified.
	public void printBN(RBTNode sRoot, int BN, Queue<RBTNode> q)	This function recursively looks for a match within a subtree and stores the matching node value in the queue.
	public String print (int BN1, int BN2)	This function takes two building numbers as input and prints triplets of all the buildings within the range BN1 to BN2.

`public class RBT`	`public void printRange (RBTNode node, int BN1, int BN2, Queue<RBTNode> q)`	This function recursively looks for all the building values falling within the BN1 to BN2 range and stores all the node references in the queue.

1.5 risingCity

Main function takes input file as an argument and then parses the file line by line. The commands are executed when the number mentioned before each command matches with the "*globalClock*". The functionalities such as- construction of existing buildings until "*globalClock*" reaches *"totalTime*" and execution of commands specified by user are done parallelly. Whenever, there is a command execution specified, it is prioritized over building construction. Construction of existing building happens in the background until *totalTime* >= *globalClock*, and "*totalTime"* is updated after every Insert. Even after all the input commands are exhausted, the construction still goes on until it meets the above condition.

Class	Node Structure	Description
`public class risingCity`	`private long globalClock = 0;`	Global clock
	`private long constructionTIme = 1;`	Construction day indicator
	`private long expectedTime = 0;`	Variable defined to perform construction in batches
	`private long totalTime = 0;`	Stores the sum of the total time values of all the inserted buildings
	`private minHeap minHeap = new minHeap(2000);`	Min-Heap declaration
	`private HeapNode minNode;`	Min node declaration
	`BufferedWriter w;`	Used to write to a output file

Command execution consists of three scenarios-
Insert (BN,TT), PrintBuilding (BN), PrintBuilding (BN1,BN2).
Construction handles the following scenarios- takes the minimum node from the minHeap and works on it until complete or for 5 days, whichever happens first. If the node's "*total time- execution time <=5",*then it is removed from the min Heap and construction continues until it reaches the point where its "*execution time = total time"*, then its building number and day of completion (globalClock) is saved in *output_file.txt* and it is removed from the RBT. Otherwise, the building's *executed time* is updated, and construction continues. Output is saved to "*output_file.txt*" int the format specified.

Class	Function calls	Description
`public class risingCity`	`public void put(String str) throws IOException`	Invoked when a write to the "output_file.txt" must be performed.
	`private void construction(String command)`	Handles two functionalities: 1. Construction of existing buildings until globalClock reaches totalTime

public class risingCity		2. Execute commands specified by user which consists of- Insert and Print
	private void completeConstruction()	Construction will be completed only when all the buildings have their Execution time == Total time. This function is invoked when the input commands are executed and only construction of building is remaining.
	private void executeCommand(String instruction)	This is invoked whenever the day entered by user matched the globalClock time. Instruction provides in the input file is executed. There are three instructions possible: Insert(BN,TT), PrintBuilding(BN), PrintBuilding(BN1,BN2). All the above cases are handled in the function.
	private void executeConstruction()	This is invoked every day, starting from the time globalClock = 1. This keeps track of the construction process. When it is time to select a building to work on, the minimum node from the min Heap is selected. The selected node is worked on until complete or for 5 days, whichever happens first. If the node reaches the state, where its execution time == total time, then its building number and day of completion (globalClock) is output and it is removed from the RBT. Otherwise, the building's executed_time is updated.
	public int checkCompletion()	This is invoked when parsing of input file is done, after all the commands are executed the construction should go on. Therefore, comparing globalClock with totalTime will ensure that all the nodes have reached convergence, where their execution time == total time. Returns 1 when construction is pending.
	void closeWrite()	Closes write to "output_file.txt"
	public static void main(String[] pars)	Main function takes input from text file and invokes functions from main class.