COP 5536 S Summer 2012

Project Report

Name: Sivasrinivas Amara

UFID: 9994-2439

Email Id: samara@ufl.edu

Table of Contents:

1. Objective

Measuring and comparing the performance of Min Leftist Tree and Min Binomial heap by executing on a random sequence of insert and remove min operations.

2. Project Files

a. Heap.java: This class contains the main method, which is the starting point of the program execution. It contain methods for random mode operation and user input mode operation of Min Leftist Tree and Min Binomial Heap. Based on the arguments provided by the user, it will decide the program execution mode and call the corresponding method accordingly.

b. MinLeftistTree.java: This class implements the Min leftist Tree functionality. It uses inner classes Node, to create objects of Leftist Tree nodes and QueueStructure, to display nodes of Min leftist Tree nodes. Methods implemented in MinLeftistTree are insert, removeMin, meld, sValue, swapChilds, displayTree. Methods implemented in QueueStructure are insert, remove, isQueueEmpty.

c. BinomialHeap.java: This class implements the Min Binomial Heap functionality. It uses inner class HeapNode, to create objects of Binomial Heap nodes and copyData method to duplicate the Binomial Heap node. Methods implemented in BinomialHeap are insert, meld, removeMin, pairwiseCombine, display, saveLevelNodes.

3. Program Structure Overview

a. Heap.java

Heap{

void main(String[]): Program execution start point. Takes mode of execution as an argument and calls the corresponding method.

void randomMode(): Generates a sequence of n random numbers and initialize Min Leftist Tree and Min Binomial Heap. Then starts the random mode execution of both Min Leftist Tree and Min Binomial Heap.

void initializeLeftist(MinLeftistTree, int[][]): Initialize the Min Leftist Tree with the given sequence of values.

void intializeBinomial(BinomialHeap, int[][]): Initialize the Min Binomial Heap with the given sequence of values.

void leftistInputMode(String): Reads the input from the given file and executes the given operations on Min Leftist tree.

void binomialInputMode(String): Reads the input from the given file and executes the given operations on Min Binomial heap.

int[] random(int[]): Takes an array and assigns random numbers from 0 to n-1 to the array and return it.

}

b. MinLeftistTree.java

MinLeftistTree {

Node root; Root of the Min Leftist Tree

void insert(int): Takes an integer as input and insert a node in to the Min Leftist Tree by creating a new node.

void removeMin(): Removes the root of the Min Leftist Tree and melds its left child and right child.

void meld(Node, Node): Melds the given two nodes by recursive call to meld. Compares the s values of resulting tree and swaps left child and right child if necessary.

int sValue(Node): Finds the s value of the given node and returns it to the calling function.

void swapChilds(Node): Swaps left child and right child of a given node.

void displayTree(): Displays node values of Min Leftist Tree on console by using Queue data structure, after the given operations are performed on the tree.

}

Node{

int value; value of the node

int s; s value of the node

Node lChild; points to the left child of the node

Node rChild; points to the right child of the node

}

QueueStructure{

QueueNode queueHead; Front node of the queue

void insert(Node): Inserts a queue node at rear of the queue

Node remove(): Removes the front node of the queue and returns it to the calling function

boolean isQueueEmpty(): Returns boolean value true if queue is empty

QueueNode{

Node object; a Min Leftist Tree node

QueueNode link; points to next queue node

}

}

c. BinomialHeap.java

BinomialHeap{

HeapNode minElement; Head of the Min Binomial Heap

int numberOfNodes; Total number of nodes in the heap

int displayed; Total number of nodes displayed

int maxLevel; Stores the maximum level value of the binomial heap

void insert(int): Creates a new node with the given data value and inserts it into the circular linked list.

HeapNode meld(HeapNode, HeapNode): Melds the two circular linked list by using the given two head nodes

void removeMin(): Removes head element of the heap and pairwise combine of the remaining nodes.

HeapNode pairwiseCombine(HeapNode): Combines all the trees having same degree by using table which stores the heap node at by degree value.

void display(): Displays node data values of heap by using saveLvelnodes method

String[] saveLevelNodes(HeapNode,int,String[]): Saves all the node data values into given string array and returns it to calling function.

}

HeapNode{

int data; Data value of the heap node

int degree; degree value of the heap node

HeapNode child; Pointer to the child node

HeapNode sibling; Pointer to the sibling node

HeadNode(int): HeapNode constructor to create a new node.

void copyData(HeapNode): copies one heap node to another heap node.

}

4. Compiling Instructions

We can execute this project on any Java IDES/Java Environments. Please follow the below instructions to execute this project on Eclipse IDE / Unix Environment

a. Steps to execute the project on ECLIPSE – Java EE IDE:

1. Create a new empty project on Eclipse.
2. Add all the .java files mentioned above to the src/default project in the Project explorer.
3. Copy the user input file to the root directory in the project for user input mode.
4. Go to Project on menu bar. Select Build all.
5. Go to Run on menu bar. Select Run Configurations.
6. Select Java Application. Select heap.
7. Go to Arguments Tab. Following arguments can be used :
    I. –r : for Random mode.
    II. –il <file_name>.txt : for user input mode for Min Leftist Tree.
    III. –if <file_name>.txt : for user input mode for Min Binomial Heap.
8. The project is ready to be executed. Select Run by choosing a run configuration.

b. Steps to execute the project on UNIX environment:

1. Create a directory on UNIX environment
2. Copy all the java files including user input files to the directory using winSCP.
3. Connect to the server using putty
4. Go to the corresponding directory
5. Compile Heap.java class using command: javac Heap.java
6. Execute the program using following commands :
    I. java Heap –r : To run in Random mode.
    II. java Heap –il <file_name>.txt : To run user input mode of Min Leftist Tree.
    III. java Heap –if <file_name>.txt : To run user input mode of Fibonacci Heap.

5. Result Comparisons

a. Expected Results:

1. For insert operation, the complexity of Min leftist tree is O(log n) and Min Binomial heap is O(1). So Min Binomial heap should run faster than Min leftist tree for Insert only operations.
2. For remove min operation, the complexity of Min Leftist tree is O(log n) and Min Binomial heap have worse case complexity O(n). So Min leftist tree should perform better than Binomial heap in remove min operations.
3. Overall performance of random inserts and delete mins will depend on the size of the structure. As the structure will grow, the performance of Min Leftist tree will be better than that of Min Binomial heap because the function f(n) grows faster than f(log n) hence remove min operation will take more time on Fibonacci heap.

b. Execution Results

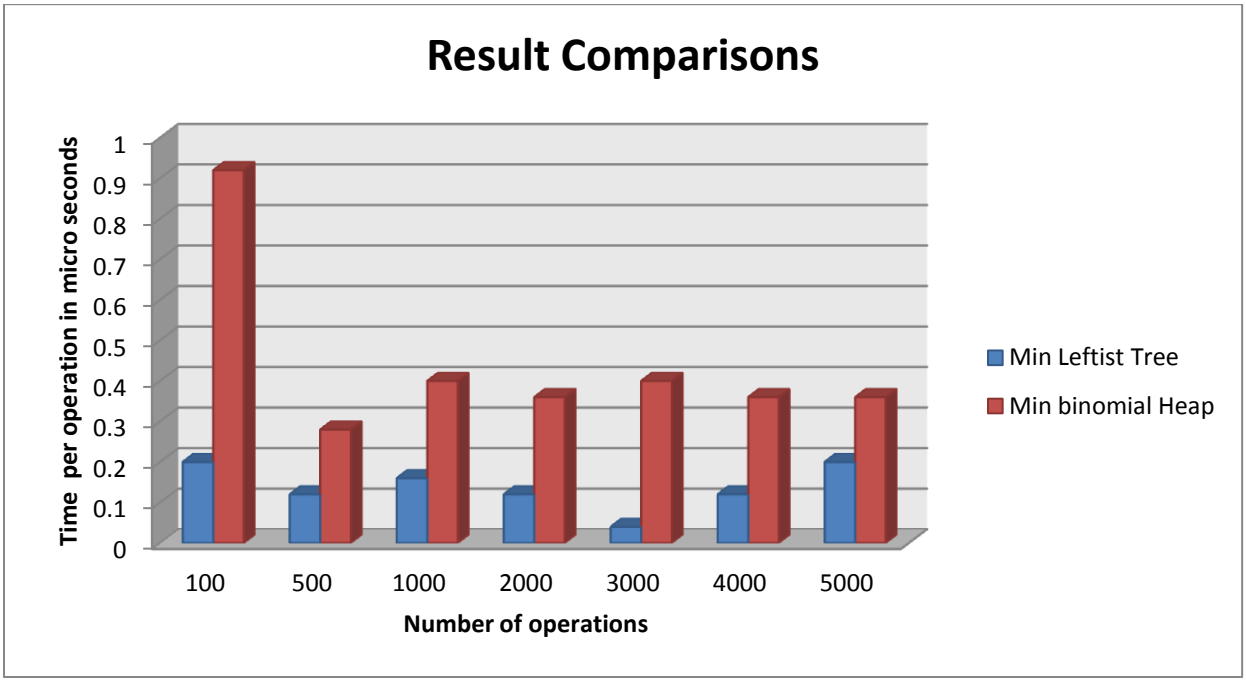
1. **Performance Table:**

The following table contains the average executions time per operation taken over 5 repetitions of application of insert/removeMin sequence (Total 5000 operations) on various initializations of Min Leftist Tree and Min Binomial Heap.

Number of Operations	Min Leftist Tree	Min Binomial Heap
100	0.2	0.92
500	0.12	0.28
1000	0.16	0.4
2000	0.12	0.36
3000	0.04	0.4
4000	0.12	0.36
5000	0.2	0.36

*running time in micro seconds and it was measured over thunder.cise.ufl.edu server.

2. Graph



c. Comment

From the above comparison, we can observe that results are matching with the expectations. For a random number of insert / removerMin operations, Min Leftist Tree runs faster than Min Binomial Heap. As the number of operations increases, time is mostly taken by removeMin operation because n increase hence O(n) grows more faster than that of O(log n).

And the other observation is, time per operation of Min Leftist Tree for insert/removerMin remains same as the number of operations increases. From this we can say that amortized time complexity of insert / removerMin operation remains O(log n).

6. Summary

From the above expected and execution results, we see that Min Leftist Tree performance is better than Min Binomial Heap. For a random sequence of insert and removeMin operations, time per operation for Min Leftist tree is less than that of Min Binomial Heap. This is because of the fact that removeMin operation in Min Binomial Heap consumes more time. And the other observation we see is that as the number of operations increase, time taken for Min Leftist Tree operations increases slightly.

Example for algorithm of Min Binomial Heap

I 3

I 7

I 4

D

D

I 1

I 9

I 13

I 6

D

*

Results after each operation:

After I 3

Level 1: [3]

After I 7

Level 1: [3, 7]

After I 4

Level 1: [3, 4, 7]

After D

Level 1: [4]

Level 2: [7]

After D

Level 1: [7]

After I 1

Level 1: [1, 7]

After I 9

Level 1: [1, 9, 7]

After I 13

Level 1: [1, 13, 9, 7]

After I 6

Level 1: [1, 6, 13, 9, 7]

After D

Level 1: [6]

Level 2: [13, 7]

Level 3: [9]

Example for algorithm of Min Leftist Tree

I 3

I 7

I 4

D

D

I 1

I 9

I 13

I 6

D

Results after each operation:

After I 3

[3]

After I 7

[3, 7]

After I 4

[3, 7, 4]

After D

[4, 7]

After D

[7]

After I 1

[1, 7]

After I 9

[1, 7, 9]

After I 13

[1, 7, 9, 13]

After I 6

[1, 7, 6, 9, 13]

After D

[6, 9, 7, 13]