Advanced Data Structures Project

COP 5536 Summer 2013

Name: Danjie Lu

UF-ID: 3201-1202

Email: daniellu789@gmail.com

Associated Files

1. heap.cpp: The file contains the main function and initiates the execution, it contains Min binomial Heap and Min Leftist Tree.
2. Makefile: To compile the heap.cpp using commend “make” on the terminal.

Compiler Description

The Project has been complied and tested under the following platform:

Platform/OS	Complier
Ubuntu 12.04LTS	g++

Steps to execute the project in Unix Environment using g++ with makefile:

1. Copy all the files including input file in a separate directory
2. Type make
3. The project is ready to be executed. Type in ./heap –r for Random mode and ./heap –ib <input_file_name> for binomial heap mode and ./heap –il <input_file_name> for leftist tree mode.

Class Description, Function & Structure Overview

Flow of structure for Main Program/Function: heap.cpp

If the input is –ib, then it use input file to execute binomial heap, including insert new node and delete minimum node. main() → read()→class binomial()

If the input is –il, then it use input file to execute leftist tree, including insert new node and delete minimum node. main() → read()→class leftist()

If the imput is –r, then it goes to execute random mode, where firstly insert 100, 200,…, 5000 nodes, then equally randomly insert node or delete minimum node. main() → randommode()

For –ib mode:

1. Open the file stream.
2. Read operation type from Input file by calling input_file.good().
3. If operation type is ‘I’, calling insert() in class of binomial to insert a node into the binomial heap whose key is the number after ‘I’.
4. If operation type is ‘D’, then delete the minimum node of the leftist tree by calling deletemin in the class of binomial, and after deleting call combinepair(); combinepair() will combine the binomial heaps whose degree are the same.
5. If operation type is ‘*’, do break.

6. Calling newdisplay() in the class of binomial to write the result in a file named 'binomial_heap.txt'.

For –il mode:

1. Open the file stream.
2. Read operation type from Input file by calling input_file.good().
3. If operation type is 'I', calling insert() in class of leftist to insert a node into the leftist tree whose key is the number after 'I'.
4. If operation type is 'D', then delete the minimum node of the leftist tree by calling deletemin in the class of leftist.
5. If operation type is '*', do break.
6. Calling display()in the class of leftist to write the result in a file named 'Leftist_tree.txt'.

For –r mode:

1. Generate a random permutation whose values are between 0 and n-1.
2. Calling insert() from class binomial to insert the n elements.
3. Calling clock() to record the time of begin of executing the follow operations.
4. Then do 5000 operations which include insert a new node whose key between 0 and n-1 and delete a minimum node of binomial heap.
5. Calling clock() again to calculate the time used for the 5000 operations.
6. Print the time cost for binomial heap.
7. Calling insert() from class leftist to insert the n elements.
8. Calling clock() to record the time of begin executing the follow operations.
9. Then do 5000 operations which include insert a new node whose key between 0 and n-1 and delete a minimum node of leftist tree.
10. Calling clock() again to calculate the time used for the 5000 operations.
11. Print the time cost for leftist tree.

<u>Flow of structure for binomial class</u>

There are two global node in the class of binomial, namely headnode and lastnode. headnode is the first node of the binomial heap, and lastnode is the last node of the binomial heap.

There is a global node array b[] which is used to hold a certain degree node.

Each node has the following element:

1. key. The value of the node.
2. degree. The degree of the node.
3. next. The next node of a node.
4. pre. The previous node of a node.
5. fchild. The leftmost child of a node.

6. echild. The rightmost child of a node.

binomial()

Initiate all global parameter.

inser(int)

1. If headnode is NULL, then set the newnode as headnode and lastnode.
2. If headnode is not NULL, set lastnode->next as the newnode, then set lastnode as the newnode.

deletemin()

1. Delete the minimum node of the binomial heap.
2. Connect the nodes between the minimum. If the minimum has children, connect their children between minimum->next and minimum->pre. If the minimum node is the headnode or the lastnode, reset them.
3. After delete the minimum node, if there is still mode then 1 node left, calling combinepair() to combine the nodes who has the same degree.

combinepair()

1. Begin from the headnode to the lastnode, to evaluate whether there is same degree nodes. If has, combine them by calling meld.
2. If after meld, the new melded node has another same degree node, then calling remeld to meld them recursively.

newdisplay()

1. Put all the key and level information into a string array named d[].
2. Display them in each level.

reload()

Used to load key and level information into the string array d[] only if a node has a child.

convertInt()

Convert the integer into string.

read()

Read the input file to execute the operations.

<u>Flow of structure for leftist tree</u>

There are one node in the class of leftist, namely root. root is the root of the leftist tree.

Each node has the following element:

1. key. The value of the node.
2. level. The level of a node.
3. The s value of a node.

4. left. The left child of a node.
5. right. The right child of a node.

leftist()

Initiate the root.

inser(int)

1. Initiate the newnode.
2. Meld the newnode with root by calling meld().

deletemin()

Meld the left child and right child by calling meld().

meld(node* firstnode, node* secondnode)

1. Exchange node 1 and secondnode if the key value of node 1 is larger then secondnode.
2. Meld firstnode and secondnode, if they meets the need the meld requirement, just meld them.
3. If they the firstnode has right child, then recursively calling meld() to meld the right child of n1 with n2.
4. If the n2 has a larger s value than n1, then swamp n1 and n2

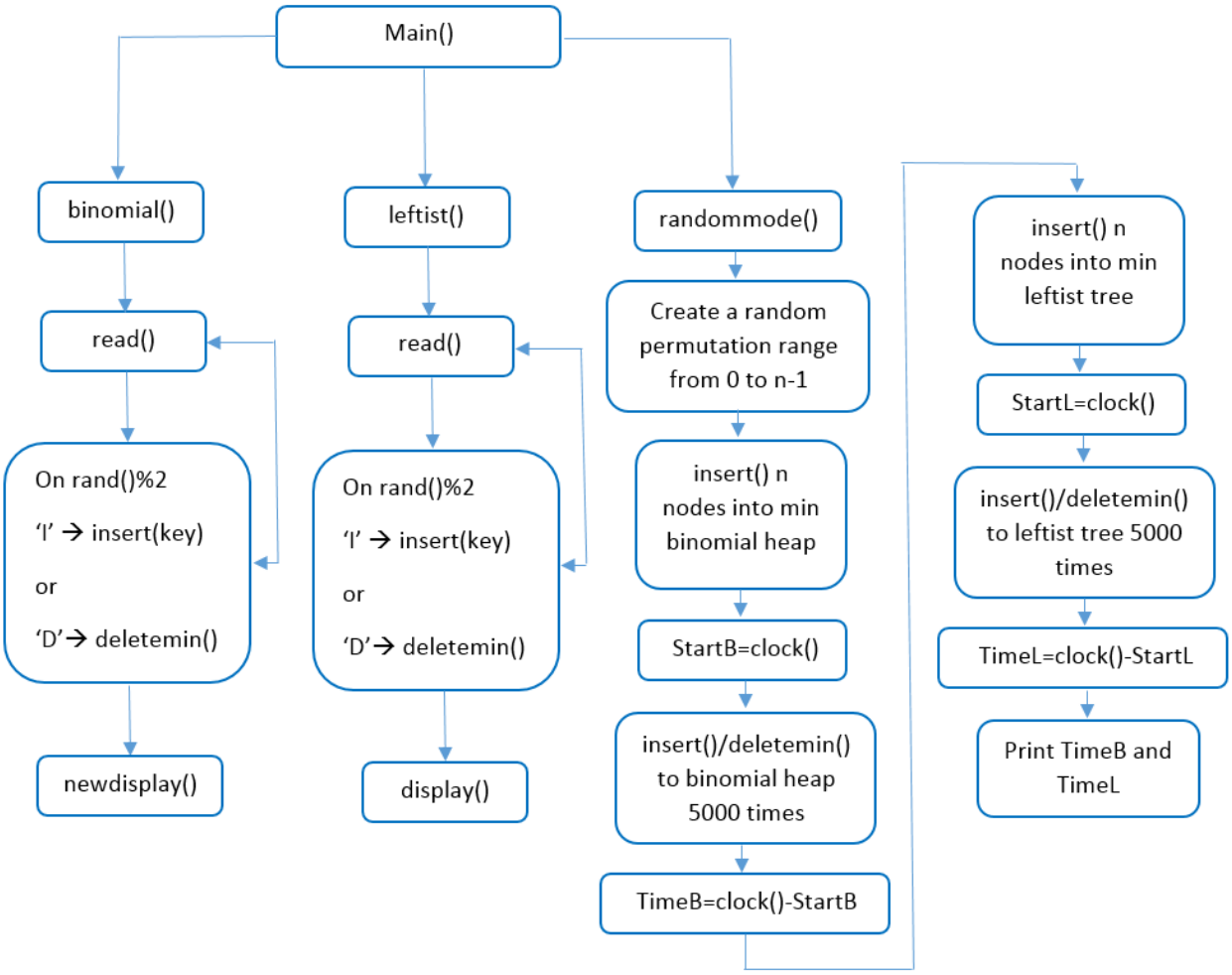
display()

1. Based on the relationship between each node, set the level for each node .
2. Display them based on the level.

read()

Read the input file to execute the operations.

Structure of Program



Radom Mode Performance Analysis

Expectation:

Based on my implementation of my project, I consider the cost per operation of min binomial heap should exceed the cost per operation of min leftist tree. And each operation will be increasing when the number of nodes increases in each data strcutre.

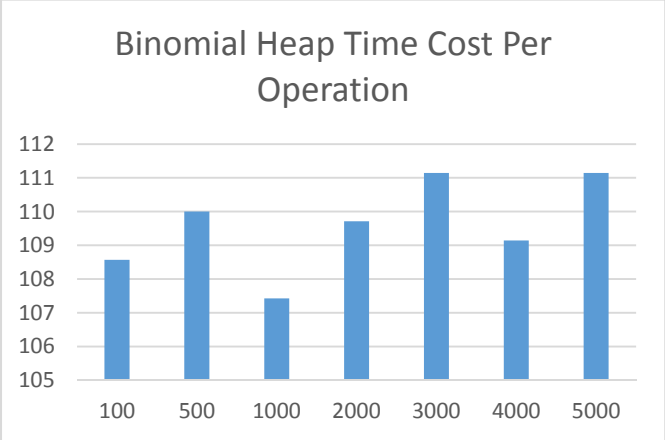
Test:

All test included using m = 5000 insert/deletemin operations after inserting n nodes both for min binomial heap and min leftist tree.

The following is raw data for Binomial Heap test, each n test 7 times, and then get the average values.

Test \ n	100	500	1000	2000	3000	4000	5000
1	108	112	108	110	112	110	110
2	108	108	108	108	112	108	112
3	108	112	108	112	110	110	112
4	108	112	106	110	112	110	112
5	110	110	106	108	110	108	110
6	106	108	108	110	112	112	110
7	112	108	108	110	110	106	112
Average	108.5714	110	107.4286	109.7143	111.1429	109.1429	111.1429

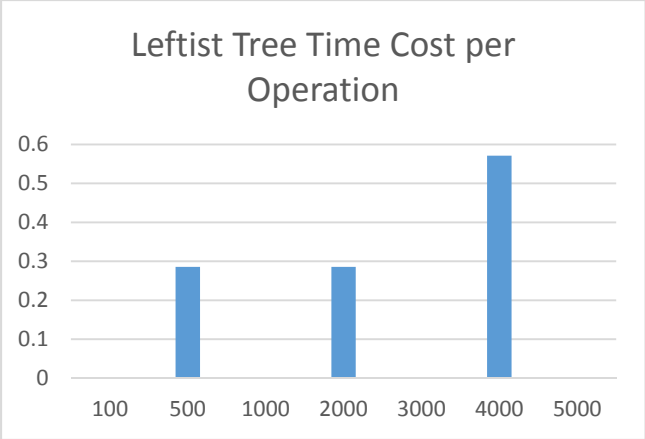
The following the Average cost per operation for Binomial Heap, n varies from 100 to 5000.



The following is raw data for Leftist Tree, each n test 7 times, and then get the average values.

Test \ n	100	500	1000	2000	3000	4000	5000
1	0	0	0	0	0	0	0
2	0	0	0	2	0	0	0
3	0	0	0	0	0	2	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	2	0	0	0	2	0
7	0	0	0	0	0	0	0
Average	0	0.2857	0	0.2857	0	0.5714	0

The following the Average cost per operation for Leftist Tree, n varies from 100 to 5000.



Analysis:

I realize my first part expectation, the cost of each operation for min binomial heap exceed the min leftist tree. As the test results show, the average cost for min binomial heap is about 110 clock clicks, while for leftist tree, the cost is almost 0 clock clicks.

My second part expectation, the cost of each operation will increase with the increasing nodes in each data structure does not completely match the test result. Because for deleting a minimum node in binomial heap, it has to search each nodes in the first floor of binomial heap or after delete the minimum node, it has to find a new minimum node. If there is more nodes, it need more time to find the minimum node. Secondly, for leftist tree, if there is much more nodes in the tree, during the meld progress, it will likely have more iteration step of meld, which also increase the cost of per operation.

The following is going to analyze the result of test.

1. With the n increasing, the cost for each operation of binomial heap is not increasing rapid. The reason I think is that the major different cost between few nodes and large numbers of nodes is to find the minimum node. But to find the minimum node is only happens in the first floor of binomial heap. Although the nodes is increasing in the binomial heap, but the nodes in the first floor is increasing slowly compare to increasing speed of the totally number of nodes.
2. For min leftist tree, the clock() function may not be accurate for these ‘few number’ test. As the program shows that n + 5000 operations cost very little time for leftist tree. If we do more tests, the result may be more obvious to my expectation.