COP 5536 Fall 2007

Assignment #2 (Programming Project)

Due Date: Nov 5th 2007, 11:59 PM Eastern Time

---

1. General

1. Problem

You are required to implement ***min binomial heaps*** and ***min leftist trees***. Plus, measure and compare the relative performance of these two implementations under the assumption that the only permissible operations are insert and delete-min.

Refer to Chapter 9 of the textbook and lecture 11-13 for a detailed description of the leftist tree and the binomial heap.

2. Programming Environment

You may implement this assignment in Java, C++, or C. Your program must be compliable by the compilers and run on the following environments:

- Java and g++ on UNIX system under the Sun UltraSparc Workstation

- Microsoft Visual C++ on PC

Please email to TA before starting this project if you want to use any other programming language or system.

2. Input Requirements

1. Running modes:

The name of your compiled program should be **heap.c** for C or C++ and **heap.java** for Java. Your program **MUST** support all of the following modes.

(1) random mode: run with a sequence of insert and delete-min operations generated by a random sequence generator.

$ heap -r

(2) user input mode: run with the operation sequence from user

(i) user input mode using ***min leftist tree scheme***

$ heap -il file-name // read the input from a file file-name

(ii) user input mode using ***min binomial heap scheme***

$ heap -ib file-name // read the input from a file file-name

Basically, there is **NO** input from user in the random mode, whereas in the user input mode, your program has to read a given sequence of operations from the file file-name.

Note: Your program should provide exactly same format specified above.

2. Input format in the user input mode:

In the user input mode, your program must conform the following input format:

```
I   D1        // insert a new element D1
I   D2        // insert a new element D2
D             // delete the min element
I   D3        // insert a new element D3
...

*             // the end of input
```

An example input is shown below:

```
I   4
I   8
D
I   1
D
*
```

The scenario of running the above example is that a delete-min operation is performed immediately after the insertion of 4 and 8, then 1 is inserted, followed by another delete-min operation.

- ***Please do not use any other input format for the operation sequence representation.***

3. Output Requirements

In the user input mode, your program should display the min leftist tree and the min binomial heap both in **level** order after executing the given sequence of insert and delete-min operations. The purpose is to test the correctness of your algorithm implementations.

Your program should not display any other information such as time taken.

In the random mode, instead, your program should not display any tree or heap information but the overall running time of two different algorithms. The purpose of this is to measure and compare the performance of the min leftist tree and the min binomial heap.

4. Operation Sequence Generation

The following gives an outline about how to generate the operation sequence in the random mode.

Assume we have a function random(k) that returns a random integer uniformly distributed in the interval [0..k-1]. Also assume that the key value of any element is in the range between **0** and **n-1**.

In order to make the number of delete-mins and inserts approximately equivalent, we denote ***d*** as a selector and let ***d*** = random(2). We determine the exact operation type by the value of ***d*** before we create a new operation. If ***d*** equals 1, then a delete-min operation "D" is created, otherwise we shall generate an insert, set x = random(**n**), and an insert operation "I x" is then put into the operation sequence. Repeat the above procedure, until the length of the operation sequence becomes ***m***, where ***m*** = 5000.

5. Performance Measurements

The performances will be measured **ONLY** in the ***random mode***.

Before you start your experiments, a random list of **n** elements should be created first. You can think of this list as a random permutation of integer **0** through **n-1**. The pseudocode about how to obtain such random permutation is shown in section 8. Next, initialize a min leftist tree and a min binomial heap respectively by inserting those **n** elements in the same order as the generated permutation. Once the initialization is done, start to measure the running time to perform the ***m*** operations using the min leftist tree as well as the min binomial heap. Divide this time by ***m*** to get the average time per operation. Do this for **n** = 100, 500, 1000, 2000, 3000, 4000, and 5000. Let ***m*** be 5000.

To get reliable results, run your program at least 5 times for each case and average them by dividing the number of runs.

6. Submission

The following contents are required for submission:

1. Makefile: If you do not use this file, provide detail instructions on how compile and run in your REPORT file.

2. Source Program: Provide comments.

3. REPORT:

- State what compiler you use, how to compile, and etc.
- Function prototypes showing the ***structure*** of your programs.
- A summary of result comparison: You should put first your expectation of the comparison before running your program: i.e. what you think about the relative performance of each scheme, and why.
- ***Please include the structure of your program. List of function prototypes is not enough.***

4. Draw a plot or a table with execution time followed by comments whether it confirms your expectation. If different, why it is the case and state the factors that influence the results.

To submit,

please compress all your files together using a **zip** or the **tar** utility and send to **TA.**

7. Grading Policy

Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.

- Correct implementation and execution: 60%
- Comments and readability: 15%
- REPORT: 25%

8. Miscellanies

- ***Do not use complex data structures provided by programming languages.*** You have to implement data sturctures using primitive data structures such as arrays and pointers.

  For example, linked list sturcture provided by libraries cannot be used. The reason is that most complex data structures provided by libraries, e.g., C++ STL, give you flexibility with the cost of degrading performance in terms of running time.

- You may refer to leftist trees and binomial heaps implemented by others. This can be done by searching the Internet. However, your implementation should be your own. ***You have to work by yourself for this assignment (discussion is allowed).*** Program code will be checked and you may have negative result if it has reasonable doubt.

- Assume that all values are integers, i.e., key values are non-negative integers.

- Pseudocode about random permutation generation
  The following function generates a random permutation of integer 0 through n-1

```
RANDOM_PERMUTATION(int A[]){

    step 1: initialize A = [0, 1, 2, 3, ..., n-1].

    step 2: for i = 0 to n-1

                do swap A[i] and A[random(i,n-1)]

}
```

Note: in iteration i, the element A[i] is chosen randomly from among elements A[i] through A[n-1]. And A[i] is never altered in the remaining iterations.

- Measuring execution time (C++)
  You can measure execution time like below:

```
#include <time.h>

clock_t Start, Time;

Start = clock();


..... (your algorithm)

Time = clock() - Start;          // time in micro seconds
```

- Measuring execution time (Java)

```
long start = 0;


start = System.currentTimeMillis();


..... run your algorthim;

stop  = System.currentTimeMillis();
// execution time for your algorithm.
time = stop - start;
```

9. Updates

A sample file input and result are as follows:

Sample input

Sample result

If you have any question, please contact TA.

Last updated: 10/16/07