1 Lab Details

- Maximum Possible Points: 70
- Lab Timings:
  1. Monday Lab: Oct 2, 9:00 AM–10:50 AM
  2. Wednesday Lab: Oct 4, 12:00 PM–1:50 PM
  3. Friday Lab: Oct 6, 12:00 PM–1:50 PM
- Lab Due:
  1. Monday Lab: Oct 8, 5:00 PM
  2. Wednesday Lab: Oct 10, 5:00 PM
  3. Friday Lab: Oct 12, 5:00 PM

2 Assignment Instructions

In this lab you will implement a min 3-heap and some basic operations on it. You will also do performance analysis on some of the operations you have implemented for the min 3-heap. The lab kit includes a main function that will be used to run and test your implementation of the data structure (in this case the min 3-heap). You will have to fill in the necessary code in function body marked as—\texttt{IMPLEMENT\_ME()}. You should comment out or delete the line containing \texttt{IMPLEMENT\_ME()} macros after you are done implementing the function.

Please use the Makefile given with the lab-kit to compile and test your code. The sample test data will be read from the file data.txt. You may not use the standard template library (STL). Please use the principles of object oriented programming to design your program. You are free to add new private data members and private methods in your classes. You may not change the signature of the public methods (declared in *.hpp files), the implementations of the public methods will be used to test your code.

2.1 Min3Heap operations

Implement the \texttt{Min3Heap} data structure and the associated operations on it as mentioned in \texttt{ IMPLEMENT\_ME()}. 

2.2 Performance analysis of building heap

(\texttt{void Min3Heap::buildHeap()})

Generate a list of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the method \texttt{void Min3Heap::buildHeap()} to heapify the word list of different sizes mentioned above and time this operation. For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the average time taken in milliseconds (or nanoseconds) (y-axis).
2.3 Performance analysis of deleting minimum element
(\texttt{void Min3Heap::deleteMinElem()})

Generate a min 3-heap of the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, and 10000 containing random words. Call the function \texttt{void Min3Heap::deleteMinElem()} to remove the minimum element from the heap and time it.

For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the \textit{average time} taken in milliseconds (or nanoseconds) (y-axis).

2.4 Performance analysis of searching an element
\texttt{bool Min3Heap::exists(std::string)}

Generate a min 3-heap of the the following sizes: 10, 50, 100, 250, 500, 750, 1000, 2500, 5000, 7500, 10000 containing random words. Generate a random word and call the function \texttt{bool Min3Heap::exists(std::string)} to search the randomly generated word from the min 3-heap and time this operation.

For each of the sizes mentioned above repeat the experiment at-least 10 times and note the timings for each run. Plot a graph of the list size (x-axis) vs. the \textit{average time} taken in milliseconds (or nanoseconds) (y-axis).

3 Operations on Min 3-Heap

You may use the array based design for min 3-heap implementation. The array size should be 15000. You may use lexicographic ordering to define \(<, =\) or \(>\) relations on the string elements.

1. Build heap: signature \texttt{void Min3Heap::buildHeap()}
   This method restructures input data into a proper heap format. This method may be called inside the constructor after instantiation of private member variables is completed. You may use the bottom-up technique to build the heap. fig. 1 is transformed in to heap as shown in fig. 2 after the operation is performed successfully.

   Figure 1: Tree structure after reading data.txt  
   Figure 2: Min3Heap structure after buildHeap

2. Add element: signature \texttt{void Min3Heap::addElem(std::string)};
   This function adds an element to min 3-heap. Eg. consider the min 3-heap generated from data.txt (Refer fig. 3). Adding an element, say “ability”, will change the heap structure to fig. 4

   Figure 3: Min3Heap structure after reading from data.txt  
   Figure 4: Min3Heap structure after adding element
3. Delete element: signature void Min3Heap::deleteElem(std::string elem);
Deletes the all the elements which are equal to the given argument elem from the heap. If the node to be deleted is not at lowest level, choose the right most node from the lowest level to replace the node to be deleted and then call buildHeap to re-instantiate the heap property. If the element does not exist, there should be no change in the heap structure.

4. Delete minimum element: signature std::string Min3Heap::deleteMinElem()
Deletes the minimum element from the heap and if necessary, restructures it to maintain the heap property. It should return the element deleted from the heap. Eg. executing the delete min function on fig. 5 will result in a heap as shown in fig. 6 and return

Figure 5: Sample Min3Heap structure
Figure 6: Min3Heap structure after deleting min element

5. Delete maximum element: signature std::string Max3Heap::deleteMaxElem()
Deletes the maximum element from the heap and if necessary, restructures it to maintain the heap property. It should return the element deleted from the heap.

6. Exists: signature bool Min3Heap::exists(std::string elem);
Should return true if the element exists in the heap, return false otherwise

7. Levelorder print: signature void Min3Heap::levelorderPrint();
Prints the min 3-heap structure in a level order fashion. For example, the heap structure created with data.txt should be printed as shown below:

beggar
chisel earth west
work men spell end

4 Questions
Please answer the following questions in not more than 5 lines each and submit it with your implemented code in the PDF format.

1. What is the worst case algorithmic asymptotic complexity i.e. $O(\cdot)$ of each of the operations that you have implemented. (10 points)
   a. Add element in the min 3-heap (void Min3Heap::addElement(std::string elem)).
   b. Delete min element from the min 3-heap (void Min3Heap::deleteMinElem()).
   c. Delete max element from the min 3-heap (void Min3Heap::deleteMaxElem()).

2. Is the average case complexity equal to the worst case complexity in each of the 3 cases above? If not, mention the average case complexity for each of the above operations in a. b. and c.

5 Report
Your report should consist of the following sections:

1. Overall organization of the experiment in not more than 10 lines.
2. Tabulated data obtained from the 3 analysis tasks. You should include all the 10 timing values for each size and also the average value you calculate for plotting the graph.

3. 3 Graphs generated from the tabulated data (1 for each operation).

6 Grading Scheme

- Your submitted code should compile and run on the EECS unix machines.
  (Please use cycle2.eecs.ku.edu/cycle3.eecs.ku.edu/EECS lab machines. g++ v6.2.1) There should be no memory leaks and compilation warnings. (10 points)
- Your code will be tested against the example flow given at the end of the instruction document. (15 points)
- Hidden test case suite run by the grader on your implementation. (20 points)
- Write up answering questions. (10 points)
- Report on performance analysis with graphs and generated data. (15 points)
- All function implementations are equally weighted.

7 Deliverables

1. Code that compiles and runs with Makefile.
2. Write up answering questions from §4
3. Report on performance analysis as per §5 on operations as explained in §2.2, §2.3 and §2.4
4. Code archive and write-up should have correct naming conventions as described in §8

8 Submission and Miscellaneous Hints

1. Please add the grader’s (Dravid Joseph) email id in the To section of the mail (dravidjoseph@ku.edu) and my (Apoorv Ingle) email id in CC (apoorv.ingle@ku.edu)
2. Your subject line for the submission should be of the form [EECS 560] Lab-<Lab #> <Lab Day> <Your KU username> eg. [EECS 560] Lab-6 M j543h898
3. Your reports should be named as <your KU username>-heap-analysis-lab5-report.pdf and your writeups should be named as <your KU username>-heap-analysis-lab6-writeup.pdf
4. Your code tar archive will be automatically named in correct format by running make tar
5. You may use the pre-existing random number generation helper functions rand and srand to generate random values for your analysis tasks.
6. You may use random-word-generator-demo.cpp to generate random words. It uses word-list.txt as its data store.
7. You may use Timer class from timer.hpp to time the operation given in the lab-kit.
8. Expand the tar ball: $ tar xvf <filename>.tar.gz
9. Make cheat-sheet:
   compiling and linking your program: make clean install
   testing your program: make test
   bundle your code in a tar archive: make tar
   Note: Please change XXXXX in first line of the Makefile into your KU username of the format (j052h567) before running make tar
10. Counting always starts from 0
9 Sample Test Cases

Please note this is just for illustration using data.txt file as input

$ make clean install
$ ./main

No input file given, using default data.txt
Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
beggar
chisel earth west
work men spell end

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 1
Enter element to be added: ability

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 6
Printing heap:
ability
chisel beggar west
work men spell earth end

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit

>> 4
Deleted Min element: ability
Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 6

Printing heap:
beggar
chisel earth west
work men spell end

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 5

Deleted max element: work

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 6

Printing heap:
beggar
chisel earth west
end men spell

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 5

Deleted max element: west

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 6
Printing heap:

beggar
chisel earth spell
end men

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 3
Enter element to check existence: beggar
Could not find element

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 2
Enter element to be deleted: chisel

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 6
Printing heap:
beggar
end earth spell
men

Please choose one of the following commands:
1: add
2: delete
3: exists
4: delete min
5: delete max
6: print level order
7: exit
>> 7