Sabancı University

Faculty of Engineering and Natural Sciences

CS 300 Data Structures

Homework 3

Assigned: Nov 18, 2016 **Due: November 29, 2016 at 11:55pm**

PLEASE NOTE:

- **SOLUTIONS HAVE TO BE YOUR OWN. NO COLLABORATION OR COOPERATION AMONG STUDENTS IS PERMITTED.**
- **10% PENALTY WILL BE INCURRED FOR EACH DAY OF OVERTIME. SUBMISSIONS THAT ARE LATE MORE THAN 3 DAYS WILL NOT GET ANY CREDITS.**
- **SUBMISSIONS WILL BE MADE TO THE SUCOURSE SERVER. NO OTHER METHOD OF SUBMISSION WILL BE ACCEPTED.**

The intention of this homework is to make you understand program instrumentation and performance measurement and compare your real results with analytical complexity. You will also understand the details of hash tables.

In this homework, you will measure and analyze the performance of hash-tables with linear probing. You will perform the following:

1. You will create a hash table of integers employing linear probing of the appropriate size (call this M)

2. You will randomly generate hash-table transactions (that is, inserts, deletes and finds).[1] You can assume that inserts, deletes and finds occur in proportion $i:d:f$ where $i+d+f=8$ and none of i, d, or f equal to 0, and $i > d$ (so the load factor monotonically grows) and $d,f \geq 1$. So, you can decide (randomly) on the type of a transaction by generating a random integer t, and then computing $m = t \bmod 8$. If m is between 0 and $i-1$, then decide on an insert transaction, if m is between i and $i+d-1$, then decide on a delete transaction, otherwise decide on a find transaction. Assuming the random number generator is giving out truly random numbers, this procedure should approximate the desired transaction probabilities satisfactorily when you have a large number of transactions.

[1] You should use the **rand( )** function available in C++ to generate random integers. If you initialize the random number seed (using **srand(seed), e.g., srand(4)**) to the same number, you will always get the same random sequence of random numbers. This is actually what you want, since you want the same sequence every time you run the program as you are developing it so that it would be easier to debug.

There are 8 possible combinations of the parameters satisfying the requirements, but we suggest that you experiment with only 3 of them

i	*d*	*f*
6	1	1
4	2	2
2	1	5

3. Before each transaction you will note
   - the type of the transaction,
   - the current load factor of the hash table (λ= N / M where N is the number of entries currently in the hash table)

   and after the transaction is complete you will note
   - whether the transaction succeeded or failed
   - how many probes were conducted.

   Since for a given table size M you will be counting the transactions, you need not store the information for each transaction, but you can keep 6 tables of M+1 entries (one for each combination of the type of transaction and outcome). Each table entry will contain the **total number of transactions** and the **total number of probes** of that type and outcome conducted. For example, when operating with M=10000 (you really should use a prime M!), the first table may hold the data for successful inserts (where you were able actually insert the data – as opposed to a failed insert where the data to be inserted was actually in the table or the table was full.) An entry in this table, say entry 6000, can hold the total number of such transactions and the total number of probes for those transactions where there were exactly 6000 entries in the hash table just before the successful insert. After all these transactions are completed, you can compute for each the load factor, (e.g., 6000/10000), the average number of probes for each specific transaction and outcome type.

4. You can generate the integers to be used for each transaction again using the random number generator. It would better to limit the range of the integers to 10M so that most integers have a decent chance of being deleted or searched. You can generate such integers by generating a random number and taking mod 10M.

5. You should generate transactions until the either the table becomes full or you have a total of 1,000,000 transactions.

6. You should output your statistical data in a format (say comma or tab separated format) that can be imported into Microsoft Excel or Open Office Spreadsheet. You should figure out how to do this.

7. Once you have all the data, you should then use Microsoft Excel or Open Office Spreadsheet so that you can generate nice plots of λ versus average number of probes for each of the transaction and outcome combinations. You should prepare a total of 6 plots each showing three curves for the three parameter combinations above. For example, the first graph may be titled "The average number of probes for successful insert operations versus load factor".

8. You should **either** use a hash function like *x* mod *M* that we used in class **or** another hash function of your choice that is suitable for linear probing (like extracting some bits from the middle of the integer (using bit operations) and then treating these bits as an unsigned integer

and then computing that integer modulo *M*. You can do research on the web or in other books in the library for interesting integer hash function to use.

9. You should use the hash-table implementation given in the text book. However, you will need to modify this implementation, so that the resizing operation is removed (just for the sake of this homework, though!). Another important change will be to change the `findpos(x)` method so that it handles deleted items properly; the code in the course slides assumes that deleted items are skipped over when inserting new items.[2] This implies an insert is implemented by actually a find followed by an insert if find fails. You should also add new private data items and possibly new private methods so that you can keep the statistics properly, and modify the implementation of the methods (such as insert, delete and find) so that the number of probes are counted properly and recorded.

At the end of the homework, you should submit the following:

1. A short report (as a word or pdf document)
   - Describing the changes you made to the hash table code to implement linear probing and keeping statistics and the hash function you chose to use.
   - Showing the graphics you obtained from the statistics you collected in your runs. Please make sure that your plots are accurate and readable with the curves and axes appropriately labeled.
   - Ending with any conclusions you can draw from your results.

2. Your source code files.

Your homework should be submitted to SUCourse at the deadline given on the first page. We suggest that you take our warning above that indicates that you should submit only your work and not collaborate with others. **We assume that by submitting your homework in the manner below, you are certifying that you are submitting your own work.**

You should follow the following steps:

- Name the folder containing your source files as *XXXXX-NameLastname* where *XXXXX* is your student number. Make sure you do NOT use any Turkish characters in the folder name. You should remove any folders containing executables (Debug or Release), since they take up too much space. **You should also put your report file labeled as *XXXXX-NameLastname.doc* or *XXXXX-NameLastname.pdf* in this directory.**
- Use the Winzip program to compress your folder to a compressed file named *XXXX-NameLastname.zip*. After you compress, please make sure it uncompresses properly and the uncompress produces the original folder exactly.
- Submit this compressed file in accordance with the deadlines above.

If the file you submitted is not labeled properly as above, if it does not uncompress, if the source code(s) do(es) not compile and if we can not open and print your report file, we regretfully will have to give you 0 credits for the homework. So please make sure all these steps work properly.

[2] You may actually want to get rid of findpos() altogether and embed it into the relevant other methods.