COMP160: Algorithms, **Homework 5**

- To obtain full credit, you must justify your answers. When describing an algorithm, do not forget to **state any assumptions that you make**, analyze its running time and explain why the algorithm is correct.
- Although not specifically stated, You can assume that we look for algorithms that are as fast as possible, and bounds as tight as possible.
- You may discuss these problems with others, but remember to write the answers on your own. In case of doubt, cite any source you used to do the assignment.
- Remember to submit each question in a separate page.

1. Let's compare the two methods for handling collisions (chaining and open addressing). For ease of calculations lets do the following assumptions:

   - Hash tables will not resize, regardless of their load factor
   - We will use the hash function $h(x) = x$ (for chaining) and $h(x, i) = x + i$ (for open addressing)

   Recall that after computing the hash, you should always compress that value (that is, the final result of the hashing should be computed modulo the size of the array)

   (a) Is it good that we never resize the hash table? What about the hash functions we defined? Justify your answer with a couple of sentences

   (b) For this example, table size will be 10 and we insert 8 elements. Insert numbers $10, 97, 23, 4, 26, 32, 56$ and 35 into each of the tables (in that order). Draw the two resulting tables and then answer the following questions separately for each table:

   - What is the worst value case query time? For which element? Give an example for both successful and unsuccessful search for each table
   - What is the average number of key comparisons needed in each of the tables? Compute the average over all possible unsuccessful queries. When counting comparisons just add up the number of comparisons between keys (you can ignore the checking if some pointer is null or similar computations)

   (c) The answers to the previous questions show that chaining is faster than open addressing. But this is an unfair comparison, can you see why?

   Hint: look at next question

   (d) Say that each pointer and each element occupies 1 unit of memory. How much memory does each hash table need? Answer this question for any general table (having m buckets and having stored n elements in).

   (e) So, for fairness purposes what we need to do is have tables that they occupy the same amount of memory. How big should the open addressing table be? Express the size as a function of m (the size of the chaining table) and n (the number of elements inserted in the table)

(f) Repeat the previous example for open addressing with the adjusted table size. What conclusions do we get?

2. Let's practice insertions into a BST and AVL trees

   (a) Let T be the BST obtained by iteratively inserting $[2, 0, 1, 5, 4, 6, 7, 3]$. Draw T (final stage is ok, but we encourage you to draw a couple of intermediate steps).

   (b) Now insert numbers $[2, 0, 1, 5, 4, 6]$ into an AVL tree instead. Draw the tree before/after each rotation and the final tree. In each rotation indicate the type of rotation

   Note: it is fine to use AVL tree simulators that you find online to verify your answer (remember to cite your sources!)...but make sure that you understand why and when are rotations happening.

3. Say that we want to store blocks of data in memory. Each block of data occupies either B or $2B$ consecutive units of memory (for some large constant B). We want to design a data structure to handle the following operations:

   - INITIALIZE: standard initialization procedure (creates an empty data structure).
   - INSERTBLOCK (`id`, `address`, `length`): stores in the data structure that we have created a new block whose id is `id`, that starts in `address` and spans `length` units of memory. You can assume the following properties: `id` is a positive integer (and that we do not have repeated ids). `address` is a positive integer from 0 to some humongous constant M. `length` can only be B or $2B$. Also, you can assume that blocks of data will never overlap.
   - DELETEBLOCK (`id`) removes the associated block from our data structure. Your algorithm will have to handle the case in which the given id does not exist (in which case you simply send an error message).
   - FINDBLOCK (`queryAddress`) returns the id of block that would be occupying that position in memory (if any). Note that `queryAddress` could be anywhere within a block (does not have to be the start of the block).

   Describe the data structure that you would use to handle all of these operations. What is the runtime for each operation?

   Note: Below we give some guidelines on how to answer the question:

   - Previously, long questions like this one were split into smaller ones. We have progressed enough in the course, so it is time you now to do so on your own. How can you structure your answer to be as clear and readable as possible? When answering this question keep in mind that a TA will have to read your solution and grade it.
   - Make a mental map of all of the tools that you have learned. Which one do you think is best? Why? We encourage you to talk to a TA (or your Comp 15 rubber duck) and discuss your strategy before writing down the answer.
   - A big part of your grade will depend on the quality of your solution (measured mainly by runtime), but a larger portion will be dedicated to the structure of your solution and correctness discussion.

- As usual in these questions, the main priority is the runtime of query (FindBlock). If you have two possible solutions, but one has significantly slower FindBlock, then pick the other one.
- M is a humongous number and B is also large. They are technically speaking constants...but they are huge. If possible, make algorithms whose runtime does not depend on neither of the two (your algorithm should just depend on n, the number of blocks inserted). If that is not possible, $O(\log M)$ or $O(B)$ are acceptable as well.
- In case of ties of runtime and space between two strategies, focus on the one that is easier to explain (we do not care about difficulty of coding)
- Add a section justifying why you chose the DS that you chose. Something like "I could have used instead technique Y. That technique would have increased insertion and initialization, but I decided on X because queries are faster that way". Why did you discard other potentially faster techniques? A short paragraph is enough, but it will help you a lot
- Above all...have fun!

Note 2: this is a simplified version of a problem that happened in real life (it was needed in a big tech software company). Improve the current solution and we may be able to make some profit!