CSC2040 Data Structures, Algorithms and Programming Languages

Practical 14

Hash Tables

Wednesday 7 March 2018

This practical aims to implement the rehashing algorithm, thereby expanding the `HashTable` class used in our lecture notes. Please read the lecture notes section – *Reducing Collision by Rehashing* – for details of the algorithm. The job consists of two related parts: (1) given the current hash table size, which should be a prime number, to calculate a new table size which is about twice as big as the original table and is also a prime number, and (2) using the new table size to implement a larger hash table by performing the rehashing operations. The `HashTable` programs you will work on are on QOL under Lecture notes/DSA Lecture 8/code. Download these and add them into your project for this Practical.

Program 1

In the provided `HashTable` class, add a new private member function, say `size_t newTableSize()`. This function returns a new prime table size which is about twice as big as the current table size. The simplest primality test is trial division (GCSE math): Given an input number *n*, check whether any prime integer *m* from 2 to $\sqrt{n}$ evenly divides *n* (the division leaves no remainder). If *n* is divisible by any *m* then *n* is composite, otherwise it is prime.

For example, you can start the search for the new table size by assuming an initial value: newSize = 2 * oldSize + 1, which is an odd number since oldSize is a prime. You check whether any **odd** integer from 3 to $\sqrt{\text{newSize}}$ evenly divides newSize. If yes, adjust the newSize by adding 2 (so it is always an odd number) and repeat the trial, until you find a new prime. In the search, we eliminate all the even numbers since a prime can't be an even number (except 2).

Program 2

Next, in the `HashTable` class, add another private member function, say `void reHash()`. Inside this function, you call the above `newTableSize()` function to determine a new prime table size based on the current table size, and then perform the rehashing operations with the new table. These operations could include, for example:

- Backup the old `vector<X*> Table` to a temporary `vector<X*> oldTable`
- Delete the elements in the old `Table`
- Obtain the new table size
- Expand `Table` to the new size
- Reinsert elements from `oldTable` into the expanded new `Table`
- Delete the elements in the `oldTable`

As indicated in the supplied `HashTable` class, this rehashing function should be called inside the `insert` function, when the current load factor exceeds a pre-defined threshold. The current threshold has a default value of 0.75 but this can be adjusted to a lower or higher value if needed.

Thoroughly test each of your new functions and then your enhanced `HashTable` class to make sure that they all work as expected. You can take the supplied `testHashTable.cpp` program as an example to help develop your test program.

Practical Test 5

Practical Test 5 is due on Wednesday 21 March. This is an open book test as the previous tests. You will have access to all previous lecture & practical material.