Comp 2140 - Data Structures
Lab 5 - Dictionaries & Hashing

‘if you don’t have a seat [index] at the [hash] table you are probably on the menu... ’ Elizabeth Warren

Please print your name and your student id in this box. Return this paper (with your answers to the written questions) to the TA at the end of the lab. The TA will comment on your programming assignments, also on this paper.

your name: your id:

The objective of this lab is to explore the hash tables and observe their advantage for implementing dictionaries.

Please take the following steps in this lab. Download the file Lab5.java. This file contains partial implementations of a dictionary using a linked list as well as a hash table that uses chaining for collision handling.

1. Take a look at the class LinkedListDictionary which implements a dictionary using a linked list. Note that the insert method inserts a key-value pair at the beginning of the list. Also, search looks for a key and returns its value if they key is present in the dictionary. Method delete(int key) should look for a key in the dictionary, and if it is present, it deletes from the dictionary. This method is partially implemented. Complete the implementation of delete(key).

To test your implementation, uncomment testDictionary(10,20); from the main function. testDictionary(n,k); generates n integer keys in the range [0,k] and places them into a dictionary maintained by a linked-list (the values of a key x is √x). The method continues by reporting all key-value pairs, deleting those with odd keys, and reporting the remaining kvps. Run your code a few times and show the output to the TA.

2. Take a look at the class HashDictionary which implements a dictionary using a hash table. The variable hashTable is a table of pointers to a linked list. Here, hashTable[i] is a pointer to a linked-list which stores all kvp’s with hash-function equal to i. Initially, when there is nothing at index i, hashTable[i] points to null. Take the following steps to complete the implementation of HashDictionary:

(i) Implement private int hashFunction(int key). Note that the hash-function maps any key to an integer in the range [0,tableSize-1]. You can use any hash function that you desire. (our goal is not to optimize that in this lab.)

(ii) Implement the method insert(int newKey, double newValue). The method should find the the hash value i of newKey, and adds the input kvp to the linked list at index i of the hash table.

(iii) Implement the method delete(key). As before, the algorithm should find the hash value i of key and delete the kvp from the linked list at index i.

(iv) Test your code via the method testDictionary(10,20). Inside that function, replace LinkedListDictionary dict = new LinkedListDictionary() with HashDictionary dict = new HashDictionary() so that the dictionary is maintained by a hash table rather than the linked list. Other aspects of testDictionary remains as before (it adds and deletes random data to/from the dictionary). Show your code and output to the TA.

Answer: The output should be similar to part (1). TA's pay particular attention to ensure you are using a hast table for the dictionary rather than the linked list in your code.
3. Take a look at code `testTimeList(int n, int k)`. This method creates `n` kvp’s with random keys and inserts them into a dictionary `dict` maintained with a linked list and then deletes upto `n/3` randomly selected keys from the dictionary. The output is the time that these operations take. This time is printed.

Implement `public static long testTimeHash(int n, int k, int tableSize)` which is similar to `testTimeList` except that `dict` is maintained via a hash table of size `tableSize`.

Run `testTimeList(100000,20000);` and `testTimeHash(100000,200000,1000)`. Repeat a few times and show the numbers and your code to the TA. Write down your conclusion in the space below.

**Answer:** One possible output:
- Linked List Dictionary takes: 2716
- Hash Dictionary with table size 1000 takes: 47

Clearly, hash table has a big advantage over linked-lists (for deletion) and hence, it takes much less time to do the operations using a hash table.

4. Repeat the previous experiment. This time comment out `testTimeList` and run `testTimeHash` with different hash table sizes. Show your numbers to the TA. What can you conclude?

**Answer:** One possible output:
- Hash Dictionary with table size 1000 takes: 149
- Hash Dictionary with table size 100000 takes: 39

A larger hash table results in less collisions and hence shorter-length lists. Consequently, the operations take less time. A larger hash table is, however, a bit wasteful on memory.