GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computing & Mathematics

CP2SD – Data Structures & Algorithms
Lab 7: Hash Maps

Hash Maps are associative arrays, that use a hash function to map sets of keys to values, stored at indexed positions called buckets. Hash maps provide constant, **O**(1), running time for insertion, deletion and retrieval operations. Because of the rapid access time they provide, hash maps are commonly used data structures and are indispensible for many applications containing large amounts of data.

In this lab we will use a variation of the *Dictionary* class (from Lab 1) to illustrate the vast difference in speed between a hash map and an indexed list. We will also examine the *hashCode()* and *equals()* methods to study what happens if these methods are not implemented correctly.

- Download the Zip archive ***lab7-Maps.zip*** from Moodle and extract the files into a folder on your hard disk. As the classes involved are all packaged, ensure that you create the correct directory structure and study the source code before you compile.

There are just four classes used in this application:

- ***Dictionary.java:*** Provides access to a dictionary of 234,936 words using either an indexed list or a hash map.
- ***Word.java:*** A wrapper class for each word in the dictionary. The method *getWord()* puts the main thread of execution to sleep to slow down a search. This will help illustrate the performance difference associated with using a map over an indexed list.
- ***StrangeString.java:*** Another wrapper class for each word in the dictionary. This class will be used as the key in the hash map. Under normal circumstances, using a String as a key to search through a dictionary of words would be amply sufficient. However, in this lab, using our *StrangeString* class will allow us to observe and change how the hashing function is used.
- ***WordSearch.java:*** Contains the main method, instantiates the *Dictionary* object and searches both an indexed list and a hash map for a search term.

Now test the running time of the two approaches:

- Execute the *WordSearch* class. This will search for the word "Aaronical", which will be found at index position 11 in the array list. Note the running time of the list and map searches.
- Change the search term to "abacist" and execute the programme again. Note that the search time of the map is unaffected, while the list takes over twice as much time to search ("abacist" is located at index 26 in the indexed list).
- Change the search term to "abashedly", located at index 70 and execute the program again. It should be becoming clear that the search time of the indexed list is increasing linearly, while the map search time remains constant.

Every Java object has a *hashCode()* method that implements a hash function to generate an integer value, representing a hash table index. The default implementation of hashCode() in the String class calculates a hash code using the ASCII character code and position of each character in the string: $s[0]*31^{(n-1)} + s[1]*31^{(n-2)} + ... + s[n-1]$. This implementation is designed to avoid collisions in the hash map, as it is unlikely, yet still possible, that two strings with different characters will have the same hash code. If a collision occurs, the *equals()* method is executed to resolve any ambiguity.

- Examine the class ***StrangeString.java***. Note that our implementation of *hashCode()* merely delegates the call to the string contained by the class.
- Change the return statement in the *hashCode()* method to return 777. Execute the programme again and explain the running time…

<u>Questions & Exercises</u>

Examine the methods defined by the Map interface. The set of keys can be accessed using a set by calling the ***keySet()*** method:

Set<StrangeString> keys = map.keySet();

A *for-in* loop can then be used to iterate over the set of keys. Why is a set returned and not a list? The values stored in the map can also be accessed in the following way:

Collection<Word> words = map.values();

Again, a *for-in* loop can be used to iterate over the values in the map. Examine the other map methods. Make sure you understand how to insert and delete from a map and search for a key and a value.