Universitetet i Oslo
Institutt for Informatikk
I. Yu, D. Karabeg

INF2220: algorithms and data structures

Series 3

Topic Map and Hashing

Issued: 7. 09. 2016

Classroom

Exercise 1 (Hash table complexity) What is the complexity of finding order information, such as max, min, or range of stored values from a hash table?

Exercise 2 (Extendible hashing) Assume that you have an empty hashtable, and that $M = 4$. Show the hash table you get by inserting the following numbers: {000100, 001000, 100000, 011011, 111000, 010100, 101010, 010101, 111001, 001010, 000111, 001001, 101110, 100100}

Exercise 3 (Deletion from a hash table) Explain how deletion is performed both with probing and separate chaining hash-tables.

Exercise 4 (Hash table behavior) Given the input {`42,39,57,3,18,5,67,13,70,26`}, and a fixed table size of 13, and a hash-function `H(X) = X mod 13`, show the resulting

1. Linear probing hash-table
2. Separate chaining hash-table

Exercise 5 (Hash table behavior) Given the input {`4371,1323,6173,4199,4344,9679,1989`}, and a fixed table size of 10, and a hash-function `H(X) = X mod 10`, show the resulting

1. Linear probing hash-table
2. Quadratic probing hash-table
3. Separate chaining hash-table

Exercise 6 (Hash table behavior) Given the input {`15,78,56,25,19,38,57,76,34,53,72,91`}, and a fixed table size of 19, and a hash-function `H(X) = X mod 19`,

1. show the resulting Quadratic probing hash-table

2. show the resulting Double probing hash-table. Note that you have to first find the *largest prime number* which is *smaller* than the size of the hash-table.

Exercise 7 (Complexity questions: Binary hash map) A class `BinaryHashMap` serves as a basis for the first 3 questions in this assignments. The internal array (of lists) which is used by `BinaryHashMap` is of length 2. Consequently, the (very basic) hash-function hashes all keys with an even length to 0 and all keys with an odd length to 1. The following complexity questions should be anwered with big-O notation, both for average case and worst case.

1. What is the complexity of locating an element in an unordered list?
2. What is the complexity of locating an element inside the BinaryHashMap?

   `binHash` is an object of BinaryHashMap.

   `Object o = binHash.get(‘‘a_key’’)`

   Hint: the elements are distributed among two unordered lists.
3. What is the complexity of inserting an element into the BinaryHashMap?

   `binHash.put(‘‘some_key’’, some_obj_pointer);`

   Hint: keys are unique, we cannot simply add it to one of our internal lists.[1]
4. Let M be the number of list-pointers internally inside a hash-table, assume that we have a hash-function which is perfect. I.e., if we fill it with M key-value pairs, we have zero collisions and all our internal lists contain one elements each.

   What is the complexity of locating an element in a hash-table with a perfect hash-function, if it contains N elements, and it has M internal list pointers. I.e., *big-O* expressed with N and M.

Lab

Exercise 8 We are going to implement a few functions inside the class `BinaryHashMap`

1. Implement the function: `boolean remove(String key)`, which returns false if the elements is not present inside BinaryHashMap true otherwise.
2. Implement the function: `String[] keys()`, which returns all keys inside the `BinaryHashMap`.

   (HINT: we know how large this array has to be from `this.size`)
3. Implement the function: `Object[] toArray()`, which should return all values from the hash-table.

Exercise 9 Write an implementation for a hash table which uses *separate chaining* to handle hash collision. The implementation should include *inserting*, *deleting*, and *searching* an element in the hash table.

[1] It's not a "law of nature" that hash tables as concept work only with unique keys. However, the binary hash map of this exercise should be a degenerate version of Java's hash map (`java.util.HashMap<K,V>`. It's a hash table implementation for a "map", which a function assocuating values to keys (so a *mapping* from keys to values. Hence, keys must be unique. Note in passing: the `put` method from Java's hash map overrides the old value (if there's one) with the newly inserted and gives back the old value (if there has been one). This detail does not affect the complexity.