Alexandria University
Faculty of Engineering
Computer and Systems Engineering
Department

File Structures Course
Assigned: Sunday, April 7, 2024
Due: Tuesday, April 23, 2024

Assignment 2
Perfect Hashing

1 Introduction

In this assignment, you're required to implement a perfect hashing data structure. We say a hash function is perfect for S if all lookups involve O(1) work. In section 2, background about universal hashing is provided. Sections 3 and 4 describe two methods for constructing perfect hash functions for a given set S. You're required to design, analyze and implement a perfect hash table as described in sections 3 and 4.

2 Universal Hashing

A probability distribution H over hash functions from U to {1, ..., M} is universal if for all x=y in U, we have

$$Pr[h(x) = h(y)] \leq 1/M \tag{1}$$

2.1 Thereom 1

If H is universal, then for any set $S \subset U$, for any $x \in U$ (that we might want to insert or lookup), for a random h taken from H, the expected number of collisions between x and other elements in S is at most N/M.

2.2 Constructing a Universal Hash Family: The Matrix Method

Let's say keys are u-bits long. Say the table size M is power of 2, so an index is b-bits long with $M = 2^b$. What we'll do is pick h to be a random b-by-u 0/1 matrix, and define h(x)=hx, where we do addition mod 2. For instance:

$$\begin{array}{ccc} \text{h} & \text{x} & \text{h(x)} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} & = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \end{array}$$

We can show that for x = y, $Pr[h(x) = h(y)] = 1/M = 1/2^b$

3 O(N^2) - Space Solution

Say we are willing to have a table whose size is quadratic in the size N of our dictionary S. Then, here is an easy method. Let H be universal and $M = N^2$. Pick a random h from H and try it out, hashing everything in S. So, we just try it, and if we got any collisions, we just try a new h. On average, we will only need to do this twice.

4 O(N)-Space Solution

The main idea for this method is to use universal hash functions in a 2-level scheme. The method is as follows. We will first hash into a table of size N using universal hashing. This will produce some collisions. However, we will then rehash each bin using Method 1, squaring the size of the bin to get zero collisions. So, the way to think of this scheme is that we have a first-level hash function h and first-level table A, and then N second-level hash functions h1, ..., hN and N second-level tables A1, ..., AN. To look up an element x, we first compute i = h(x) and then find the element in Ai[hi(x)].

4.1 Perfect Hashing:

You're required to:

1. Implement an $O(N^2)$ as well as an $O(N)$-Space perfect hash table implemented as described in sections 3 and 4.
2. Verify that the hash table you constructed consumes $O(N^2)$-space in the the quadratic space method and $O(N)$-space in the linear space method.
3. Report the number of times required to re-build the hash table in the case of collision.

4.2 Application: English Dictionary

As an application based on the perfect hashing implementation, you are required to implement a simple English dictionary supporting the following functionalities:

1. Initialize (constructor): Takes the name of the type of the backend perfect hashing as an input and creates a new empty dictionary based on it.
2. Insert: Takes a single string key and tries to insert it.
3. Delete: Takes a single string key and tries to delete it.
4. Search: Takes a single string key, searches for it, and returns true if it exists and false otherwise.

5. Batch insert: Takes a path to a text file containing multiple words each in a separate line. And tries to insert all that words into the dictionary.

6. Batch delete: Takes a path to a text file containing multiple words each in a separate line. And tries to delete all that words from the dictionary.

4.3 Command Line Interface

You should implement a command line interface that will enable us to deal with the dictionary and apply all its implemented operations. This interface must take the type of the backend tree as an initial input then create a dictionary based on it and allow the user to apply subsequent operations on it from the following list:

1. Insert a string and prints a confirmation message or an error one if the the string already exists in the dictionary.

2. Delete a string and prints a confirmation message or an error one if the the string doesn't exist in the dictionary.

3. Search for a string and print whether it exists in the dictionary or not.

4. Batch insert a list of strings taking the path of the file containing these strings and prints the number of newly added strings and the number of already existing ones.

5. Batch delete a list of strings taking the path of the file containing these strings and prints the number of deleted strings and the number of non existing ones.

4.4 Java Unit Testing

You should provide a set of 15-20 JUnit tests that tests the correctness and effeciency of the different implemented parts. Also, these tests must show a comprehensive comparison between the 2 types of perfect hashing w.r.t space, number of times required to rebuild the hash table in case of collision, and time.

5 Notes

- You need to work in teams of 4 or 5.
- You need to use Java in your implementation.
- Each team should submit via teams the code and a **report** explaining the time analysis of the implemented functions in addition to a comparison between the 2 perfect hashing techniques implemented in this assignment w.r.t the mean search time and the mean insertion time at different table sizes.