Hash Tables in Java

Malte Berg - ID1021

HT 2024

Introduction

This report covers different implementations of hash tables in Java, as well as looking at different ways to solve a given problem, being ways to store and search zip codes.

Using Regular Arrays

Array Size $n = 10000$

The first task was to save a list of 9675 zip codes with other information, like city name and population count, in an array with an array of similar size. To save the information, a class `Area` had to be created.

```java
public class Area {
    private String areaName, zipCode;
    private Integer population;

    public Area(String zip, String name, Integer pop) {
        zipCode = zip;
        areaName = name;
        population = pop;
    }
}
```

Along with this code, some getters were implemented to be able to get the information of the `Area` object for different uses.

An array of `Area` objects with a size of 10000 was created to hold all of the zips. The array then had to be filled with the zips.

```java
try (BufferedReader br = new BufferedReader(new FileReader(file))) {
    String line;
    int i = 0;
    while ((line = br.readLine()) != null && i < this.max) {
```

```
 5              String[] row = line.split(",");
 6              postnr[i++] = new Area(row[0], row[1], Integer.valueOf(row[2]));
 7          }
 8          this.max = i;
 9      } catch (Exception e) {
10          System.out.println(" file " + file + " not found");
11      }
```

When creating a `Zip` object, one passes the path to the `.csv` file where the zip codes are stored, and the above code then creates new `Area` objects and places them in the array from index 0 to 9764.

From this, searches can be made with both linear and binary search since the list is ordered. Before showing the result of this, the code can be further improved by changing line 6 to the following to lines which will convert the strings to integers before adding them to the array (which means changing the data types in the `Area` as well), to find out if integers perform better than strings.

```
Integer code = Integer.valueOf(row[0].replaceAll("\\s",""));
postnr[i++] = new Area(code, row[1], Integer.valueOf(row[2]));
```

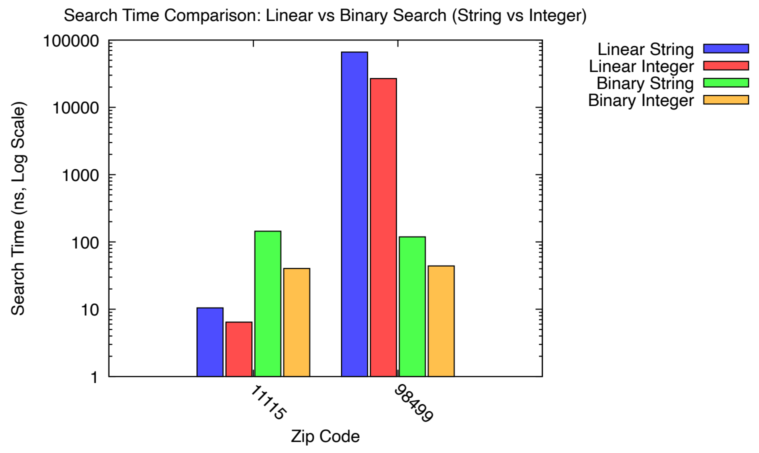

Figure 1: Runtime for linear and binary searches for String and Integer zip codes, in nanoseconds

In Figure 1, it is first clear that storing and searching for integers are faster than strings. Next, searching for the first zip code in the array is faster with linear searching, while searching for the last is faster with binary searching. The reason for this is due to the time complexity for the two search functions. The linear search algorithm has a time complexity of $O(n)$ and the binary search algorithm has a time complexity of $O(\log n)$. This means that as the index of where the key can be found gets higher, the linear search no longer outperforms the binary search.

Array Size $n = 100000$

As covered in previous assignments, the times for accessing elements in an array are fast, and are much faster than having to search the array. If the array has a size big enough to fit all the zip codes with the zip code as the index, the lookup time will be just the time for looking at a specific index in the array.

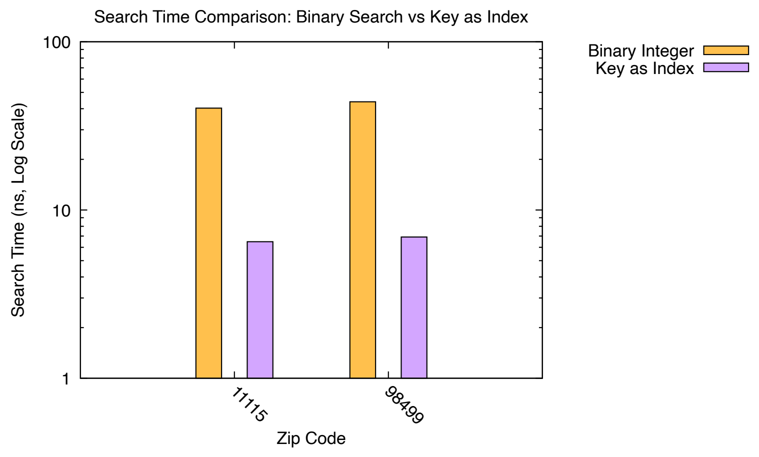

Figure 2: Runtime for key index and binary searches for Integer zip codes, in nanoseconds

As Figure 3 shows, there is a clear benefit of using the key as an index when thinking about time, but it has to be remembered that the array now has a 10 times bigger size, so one has to weigh the benefits and drawbacks.

Using Hash Tables

Finding a Suitable Modulo value

To create a hash tables, one must first find a modulo value to use to hash the key to be looked for. Using the code provided in the assignment and testing different values, it seemed that prime values gave the least amount of high-number collisions. After testing with a number of prime numbers, the modulo value that was used was 14683 which was low enough to not waste too much space and still had a low number of high-number collisions.

Handling Collisions

There are two ways in which the handling of collisions will be made in this assignment, with ”buckets” or with a slightly bigger array.

Buckets

A bucket in this case is simply a linked list that will contain `Area` objects. The collisions that can happen means that the hashed index for the array is the same for at least two different keys. For the bucket implementation, an array of size 14683 is created with buckets being initialized in the array whenever needed. If a zip code is added to an already ”occupied” spot in the array of buckets, the new area will just be added in the linked list, meaning that one has to search the linked lists when doing a lookup method.

```
public Area lookup(int key) {
    int hashed = hash(key);
    if (buckets[hashed] == null) return null;
    LinkidList.Node current = buckets[hashed].head;
    while (current != null) {
        if (current.getData().getZipCode() == key)
            return current.getData();
        current = current.getNext();
    }
    return null;
}
```

The above code is the lookup method made for searching in the array of buckets. If found the relevant area is checked if the zip code is the same as the key, and returned if they are the same, else `null` is returned.

Bigger Array

Another way to implement the hash tables is to not use buckets but instead use the array itself to store the areas. If an area is to be added but the hashed index is already used, the hashed value is then incremented until it finds an empty index to store the value in. This means that for the lookup method, it will also have to increment the hashed value if not found on the hashed index.

Array size	22024	29366	36707	44049	51390	58732
Number of lookups	51614	20127	20211	16110	14695	12759

Table 1: Mod values with Array size and Number of lookups

The number of lookups needed to find all zip codes in the array is provided above for different mod values. As can be seen, multiplying the original mod value from 1.5x with increments of 0.5x, shows less number of lookups. However, the trade off is of course the need to increase the size of the array.