Algorithms Assignment 5 documentation report

Andrew Osborne

March 11, 2019

Algorithm structure

I have used an object–oriented approach to implement some limited functionality of a hash table. In the core of my program are two arrays. I construct a hash table by requiring a data size and a hash–method flag. That is, at initialization, the user specifies the total number of elements which may at any time be stored in the hash table. Also, the hash function is chosen by the value of an integer at construction. The three available hash functions are:

$$\begin{aligned} h_l(k,i) &= (k+i)\, mod\ m \\ h_q(k,i) &= (k + c_1 i + c_2 i^2)\, mod\ m \\ h_d(k,i) &= (k + i\,(1 + k\, mod\ (m-1)))\, mod\ m \end{aligned} \tag{1}$$

Where $c_1 = 1$ and $c_2 = 3$, and m is the table size. In this situation, we implement open addressing with linear probing (h_l), quadratic probing (h_q), and with double hash probing (h_d).

Then, inserting keys into my hash table is as follows:

```
insert(Table,key)
  for i = 1 to m
    location = h(key,i) // apropriately chosen
    if Table[location] != NIL
      Table[location] = key
      return i + 1 // number of probes for insertion


fill(T)
  key = random.int

  for i = 1 to 900
    while !inTable(T,key)
      key = random.int
    insert(linearProbe,key)
    insert(quadraticProbe,key)
    insert(doubleHash,key)
```

It is worth mentioning that, internally, java initializes allocated arrays to default values, and in order evaluate `Table[location]!=NIL` , I actually check the truth value of an element of an array of booleans. Note that we can recover the table index of an element with its key value, and the returned number of probes simply by evaluating

```
index = h(key,insert(T,key)-1)
```

On the other hand, If I had chosen to return the index value directly from `insert(Table,key)` , then recovering the number of probes would be less trivial and would result in more complicated code.

Finally, in the main function of my program, I simply generate a random integer, add ensure that it is not already in any of the tables, and then insert it into all of the tables. I do this by using the `search(..)` function in my code which implements a hash table's search function as in the pseudocode of the text. I repeat this 900 times without counting the probe numbers, and then 50 times while counting the probe numbers. The same random sequence of integers is inserted into all three hash tables (linear probing, quadratic probing, and double hashing).

Testing

Note that any function of the form

$$h(k,i) = (f(k) + i * g(k))\, mod\ m$$

is essentially running through the subgroup of $\mathbb{Z}_m$ which is generated by $g(k)$ and translated by $f(k)$. This is why it is important that m be prime or have few nonotrivial divisors. In the case of finite Abelian groups $\mathbb{Z}_n\ :\ n \in \mathbb{Z}$, for all d in $\mathbb{Z}$ such that $\frac{n}{d} \in \mathbb{Z}$ there is a subgroup $G \subset \mathbb{Z}_n$ which is isomorphic to $\mathbb{Z}_d$. With that in mind, in the case that $i = 0$, $h(k,i) = f(k)\, mod\ m$, which, in our case, is $k\, mod\ m$. Then, it is clear that, in our insertion routine, if all inserted elements fit in their $k\, mod\ m$ place, we should have no additional probing. As a test to this, we should insert $[0, 950] \cap \mathbb{Z}$ into our hash table, and we should see that there are fifty probes for the last fifty elements for all three probing method. This is indeed the case.

On the other hand, the number of probes is heavily dependent on the distribution of integers inserted into our hashtables. I created a large array of integers which were equally spaced over a large interval (so separation between values was large), and expected linear probing to outperform the other two probing methods. On the other hand, when each element of our array is sampled from a uniform distribution (of any width), we would expect that quadratic probing defeats linear probing, and that double hashing is better than quadratic probing. This seems to be the case, although the margin between quadratic probing and linear probing is much larger than the margin between quadratic probing and double hashing on average.