ASSIGNMENT 04: HASHING

Brock Davis
Computer Programming III
April 2017

Goal

- Make a generic HashTable
- It will model the Java HashMap
  - *Use unique Keys that are hashed*
  - *Stores values in parallel table*
- Testing
  - *A book is read, storing each word's frequency*

Approach Overview

- Two parallel arrays, storing pairs of keys and values
- Index is determined by hash function modulus array size
- Automatically upsizes table after a certain load factor is reached

Hashing Keys

- Utilizes Java's built-in hashCode() function that every Object has
- Scrambles that hash integer to obtain an even distribution
- Returns absolute value to be able to use as an index

```
hashCode = key.hashCode();
hashCode ^= (hashCode >>> 20) ^ (hashCode >>> 12);
return abs(hashCode ^ (hashCode >>> 7) ^ (hashCode >>> 4));
```

Putting Key/Value pairs

```
i = hash(key)
coll = 0
while (keys[i + coll] != null)
    if (keys[i + coll].equals(key))
        return -1 //Doesn't allow duplicate keys
    coll ++
keys[i + coll] = key
values[i + coll] = value
size ++
updateTable() //Upsizes table if needed
return coll
```

Removing Key/Value pair from Key

- Finds the selected key in the chain, if it is present
- For each key/value pair that doesn’t match the hash code for the index, remove it and re-add it

Removing Key, Value pair from Key

```
val = null
i = hash(k)
toAddK = {}
toAddV = {}
while (keys[i] != null)
    if (keys[i].equals(k)) //Found key
        size --;
        val = values[i];
        keys[i] = null; //Remove from array
        values[i] = null;
        K next = keys[i + 1];
        while (next != null) //Search for pairs to remove and re-add
            if (hash(next) % (i + 1) == 0)
                toAddK.add(next);
                toAddV.add(values[i + 1]);
                keys[i + 1] = null;
                values[i + 1] = null;
                size--;
                break;
            i++;
            next = keys[i + 1];
        break;
    i++;
for (int ind = 0; ind < toAddK.size(); ind++)
    this.put(toAddK.get(ind), toAddV.get(ind));
return val;
```

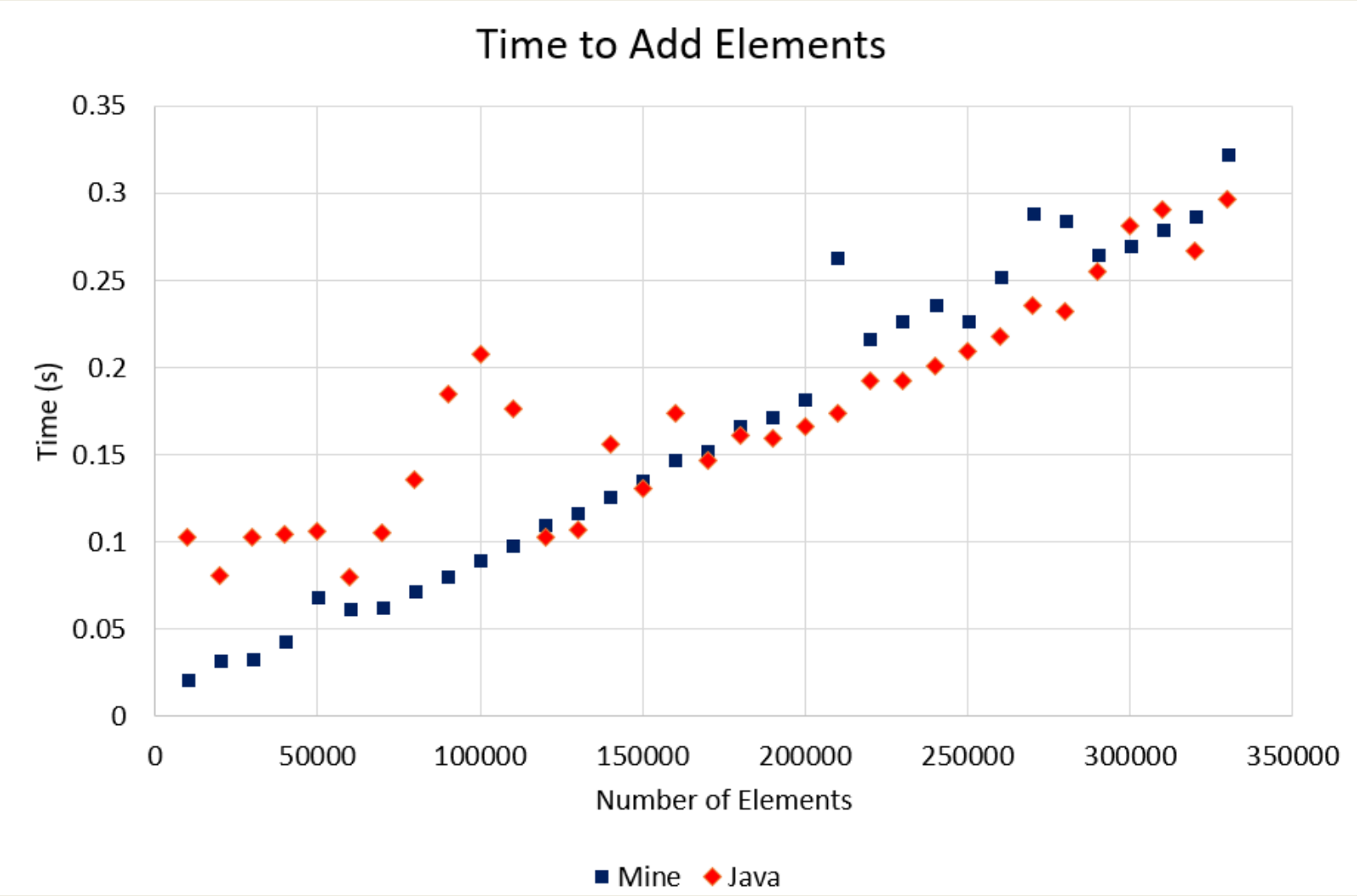
Other Notable Implemented Functions

- containsKey
  - *Checks key table in O(1) with hashing*
- containsValue
  - *Linear search through value table*
- Get
  - *Checks key table and returns associated value in O(1)*
- keyset
  - *Adds all elements in key table to a set*

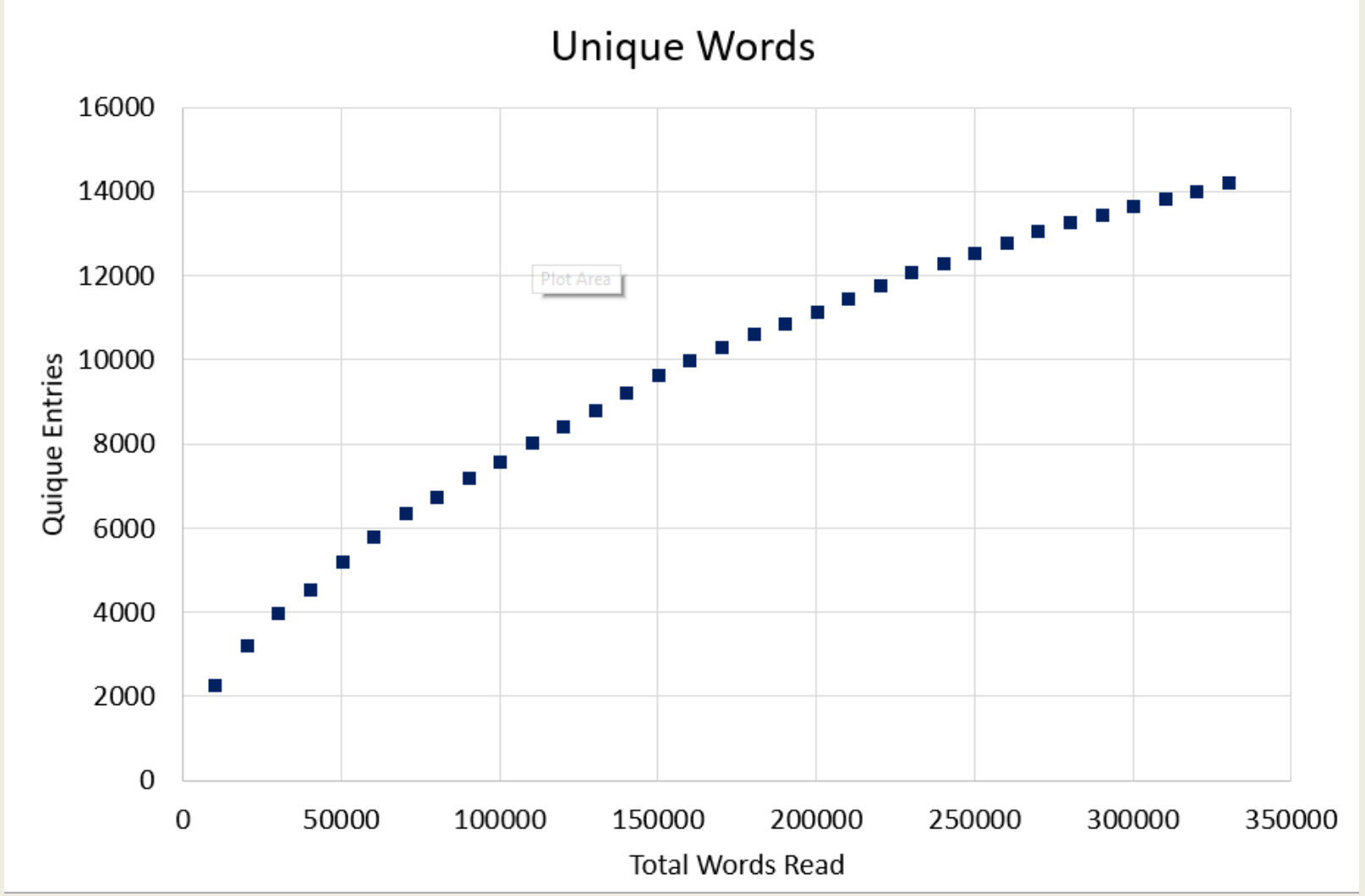
Results

- The HashTable was able to be implemented comparable to the Java HashMap implementation
- The functionality was tested by mapping a word as a key and its frequency in a book as the value into a HashTable and Java HashMap and recording various statistics
  - *The put(K key, V val), containsKey(K key), remove(K key), and keySet() functions were tested in this method*

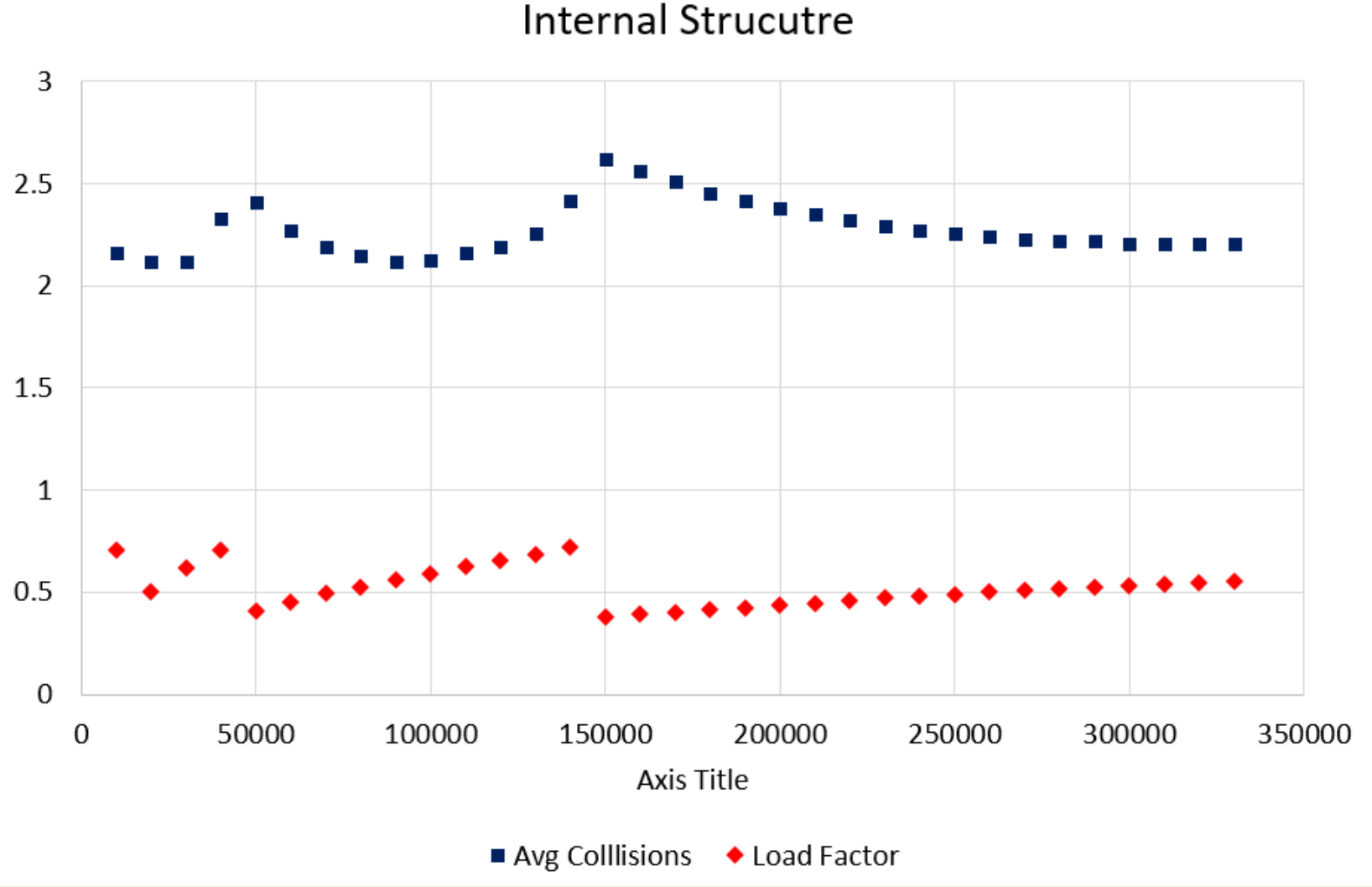
Results


Time to Add Elements
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Time (s)
0
50000
100000
150000
200000
250000
300000
350000
Number of Elements
Mine
Java

Results


Unique Words
16000
14000
12000
10000
8000
6000
4000
2000
0
Quique Entries
Plot Area
0
50000
100000
150000
200000
250000
300000
350000
Total Words Read

Results


Internal Strucutre
3
2.5
2
1.5
1
0.5
0
0
50000
100000
150000
200000
250000
300000
350000
Axis Title
Avg Colllisions
Load Factor

Conclusion / Further Research

- A generic Hash Table was successfully implemented with access and removal times comparable to the Java implementation
- Additional Functions could include:
  - *Dynamic resizing*
  - *Better hash function*
  - *Hop scotch hatching*
- Additional testing could include:
  - *Testing for limits of space*
  - *Testing for collisions in hash function*