Noah Abdelguerfi 11/17/14
CSCI 2125 - Data Structures
Instructor: Aaron Maus

Homework 3

Hash Table and AVL-Tree

Runtime comparison Hash-Table and AVL-Tree:

Value of N	10^2	$5x10^2$	10^3	$5x10^3$	10^4	$5x10^4$	10^5	$5x10^5$	10^6	$5x10^6$	10^7	$2x10^7$
Hash Table Runtime (ms)	4	5	8	24	24	73	79	256	1,175	3,902	5,730	24,897
AVL-Tree Runtime (ms)	5	11	16	47	36	67	92	188	387	4,332	8,622	14,713

Table 1. Runtimes Comparison of Insertion

From the above Table, it is seen that the general trend is that the Hash-Table runtime is faster than that of the AVL-Tree for smaller values of N such as 100, 1,000, 5,000, and 10,000. For larger values of N, the AVL-Tree seems to have the advantage in terms of runtime. For example, the AVL-Tree is faster for N equal to 500,000, 1,000,000, and 5,000,000.

The performance of the Hash-Table is strongly affected by its *load factor*. As the load factor increases, the number of collisions, when inserting, is likely to increase. Hence the size of the hash table in my program is doubled (to the next prime number) to guaranty that it is at least half empty. Another important fact is that the Hash-Table method works best when its size is a *prime* number.

Default size of Hash-Table:

The default size is an important factor in the performance of the Hash-Table algorithm. Firstly, the default size should be prime because the *Quadratic Probing* hash resolution method I used works best when the table size is prime. If the table size is not prime, the number of alternative locations when a collision occurs is severely limited (textbook page 182). Secondly, the Hash-Table has to be kept at least half empty at all times to guaranty that insertion of a new item is possible (otherwise insertion could fail). In my program, every time the array size is filled halfway, the *rehash* method is called to double the array size to the next prime number. If the default size is small and rehash has to be called often it can slow down the algorithm's runtime. To address this problem, the default size of the hash-table can be set to a large prime number so it reduces the amount of time *rehash* method is called (which degrades runtime performance), but this solution may lead to a waste of memory space. Table 2 shows the *Hash-Table* runtime as the *Default Table* Size is increased but still kept prime (n is kept constant at 100,000).

Default Size	11	101	1,001	10,001	1,000,001
Runtime (ms)	382	287	160	187	152

Table 2. Runtime of Hash-Table as a function of Default Size (n=100,000)

Hashing Function:

```
public static int hash( String key, int tableSize )
    {
      int hashVal = 0;

      for( int i = 0; i < key.length( ); i++ )
        hashVal = 37 * hashVal + key.charAt( i );

      hashVal %= tableSize;
      if( hashVal < 0 )
        hashVal += tableSize;

      return hashVal;
    }
```

The above hashing function method (page 173 in textbook) has been used. This hashing function computes a polynomial function of 37 using Horner's rule. The coefficients of the polynomial are the ASCII equivalent of each character in the key. A test at the end of the method checks for overflow and corrects it.

Collision Handling:

For collision resolution, I used the *quadratic probing* algorithm. This algorithm uses a quadratic hash function f(i). In case a collision occurs when inserting key "x", cells:

$$H_0(x), H_1(x), H_2(x), \ldots\ldots$$

are visited in sequence until an empty cell is located. $H_i(x)$ is defined as:

$$H_i(x) = (\text{Hash}(x) + f(i)) \bmod \textit{TableSize}, \quad i=0,1,2,\ldots\ldots$$

where Hash(x) is the Hashing function used and
TableSize is the size of the hash table

If the Table size is prime, and the hash table is at least half empty, then the quadratic probing algorithm guaranties that a new element can always be inserted (Theorem 5.1 page 182 in textbook) in the Hash-Table.

The quadratic probing algorithm has the advantage of not using an additional data structure such as the *Separated Chaining* algorithm, which uses a *linked list* for collision resolution. It also avoids the *Primary Clustering* problem that the *Linear Probing* algorithm has.