Fall 2017 CIS 3362 Homework #7 Sample Solution

First Attempt

I wrote the program **hash_sol.java** which simply generates random printable strings. First we randomly choose the number of characters 1 - 10, then we choose a random printable character for each slot. Printable characters are the ones with Ascii values in the range [32, 126]. After generating a bit more than 3 million strings, my program found the following match:

```
1)t/o`(P1h}
2)&%J`(PcbX
```

The structure of these strings proves that there's some sort of structural problem with my hash function as the lengths are the same and multiple characters in corresponding positions are the same.

To double check that this is indeed a match, I wrote a quick main to print out the hash function for both. Indeed, both give the hash value: 53905630794605.

In my next approach, I will limit myself to lowercase strings of letters of length 10. I may also go through them "in order" instead of randomly.

Second Attempt

This time I limited myself to generating random only lower case strings of length 8. I got an answer much faster (in less than 100,000 strings generated):

```
1)amiattgz 16382627544431
2)vuiattpb 16382627544431
```

This code is in **hash_sol2.java.**

Third Attempt

Based on a tip from a student, I tried a third attempt: I created one random string of length 8, and tried every permutation of that string. Then, I looked to see if any of the permutations caused a collision. If not, then I just generate a new random string of length 8 and redo the process. This way, my hash table never gets bigger than 40,000 terms. It turns out that my hash function doesn't do well with reordering of the same letters. This ran nearly instantaneously as it worked on the first string it tried. Here is the output I got when running my program once:

```
1)ukbuphul 1401898269555
2)ulbuphuk 1401898269555
```

Running it a couple more times shows me that exchanging the 2^{nd} and 8^{th} letters seems to not affect the hash value. The code for this one is in **hash_sol3.java.** I tested this hypothesis, and in practice it worked 100 out of 100 times (shown in **hash_sol_test.java.)**