Hash Table Implementation Lab:

Author: Mark McKenney
Organization: CS 534, SIUE

Overview:

You may be in a group of up to 3 people.

Create a hash table in your language of choice. The hash table should implement double hashing (i.e., if the first insertion fails, use a second hash function). If the insertion of a value fails on both hash functions, you will print to the screen that the insertion failed. The recipe for a hash function is covered in the course notes (http://www.cs.siue.edu/~marmcke/docs/cs340/hashing.html#good-hash-functions). You will run 2 experiments

Experiment 1:

Insert integers into a hash table of size 10 (the array should be of size 10). Insert any integers you wish (sequential or random). Upon failure, print how many integers were successfully entered. Run the experiment 10 times and indicate the average number of insertions before failure. Each time you re-run the experiment, CHANGE THE CONSTANTS IN THE HASH FUNCTION (but not the prime number).

Experiment 2:

Repeat Experiment 1, but use a hash table of size 100.

Expected output:

Each time an insertion fails, print the number of successful insertions.

Example output:

Here is an example output of a hash table of size 4. I have printed out more than is necessary in order to aid in debugging. The hash function constants are chosen by a random number library, so they are different each time the program is run. Notice that the 1st and 3rd runs of the program hashed perfectly! The programs below report when the first hash function has a collision, but does not exit unless both hash functions have a collision. The constant values for the hash functions are shown. For **Hash 1**, A and B are on the first line; for **hash 2**, A and B are on the second line.

```
$ python hash.py
182838385 1178405455
```

```
2279673840 2739249572
[1, None, None, None]
-------
182838385 1178405455
2279673840 2739249572
[1, 2, None, None]
-------
182838385 1178405455
2279673840 2739249572
[1, 2, 3, None]
-------
182838385 1178405455
2279673840 2739249572
[1, 2, 3, 4]
-------
cant insert  5  at  index1  0
cant insert  5  at  index2  3
182838385 1178405455
2279673840 2739249572
[1, 2, 3, 4]
-------

$ python hash.py
1352340964 1770176716
370766268 3154085076
[1, None, None, None]
-------
1352340964 1770176716
370766268 3154085076
[1, 2, None, None]
-------
cant insert  3  at  index1  1
cant insert  3  at  index2  0
1352340964 1770176716
370766268 3154085076
[1, 2, None, None]
-------

$ python hash.py
3597286616 2299654618
245993834 4043789589
[None, None, None, 1]
-------
3597286616 2299654618
245993834 4043789589
[2, None, None, 1]
-------
3597286616 2299654618
245993834 4043789589
[2, 3, None, 1]
```

```
-------
cant insert  4  at  index1  1
3597286616 2299654618
245993834 4043789589
[2, 3, 4, 1]
-------
cant insert  5  at  index1  2
cant insert  5  at  index2  0
3597286616 2299654618
245993834 4043789589
[2, 3, 4, 1]
-------

$ python hash.py
3300483063 2277869075
5023049 1636810222
[None, None, None, 1]
-------
cant insert  2  at  index1  3
3300483063 2277869075
5023049 1636810222
[2, None, None, 1]
-------
3300483063 2277869075
5023049 1636810222
[2, None, 3, 1]
-------
cant insert  4  at  index1  2
cant insert  4  at  index2  2
3300483063 2277869075
5023049 1636810222
[2, None, 3, 1]
-------
```

Your API:

Don't make this harder than it is. You need a very simple API. Here is the API you need:

```
class hasher:
  def __init__(self, arraySize):
  '''  constructor.  Generate random A and B
  values for both hash functions.  Allocate
  array. '''

  def getIndexHash1( self, value ):
  ''' compute the hash function on value,
  return the index where it should be
  inserted in the array.  USE HASH FUNCTION 1'''
```

```
def getIndexHash2( self, value ):
''' compute the hash function on value,
return the index where it should be inserted
in the array.  USE HASH FUNCTION 2'''

def insert( self, value ):
'''  Insert the value into the hash table.
Return a failing value if BOTH hash functions
have collisions.'''

def printTheTable( self ):
''' print it!  for debugging '''
```

What to Submit:

A SINGLE zip (or tar) file containing:

1. all source code
2. a Makefile if your program needs to be compiled
3. a README file listing group members and your results from your experiment. Be specific in you results. Indicate the percentage of runs that did not fail on any insertions. Indicate roughly the number of successful insertions when the hash table did fail for each experiment. This MUST be TXT file (plain text).

If you submit a RAR file, you will recieve a grade of 0.