C343 Project 6 - Hash Tables in Compression

Due 11:59pm, November 16, 2015

1 Assignment Description

This week we return to our string compression program, but replace the use of Python dictionaries with our own implementation of a hash table.

2 Your Task

We have given you a complete solution for Project 5, except that we have replaced the uses of Python dictionaries with uses of the Hashtable class, defined in `hashtable.py`. That class has stubs for all of the methods that you need to implement:

1. `__init__(self, dict)` - initializes the hash table, copying any items from the `dict` parameter into the new hash table.
2. `__getitem__(self, key)` - return the value associated with the `key`.
3. `__setitem__(self, key, value)` - associates the `value` with the `key` within the hashtable.
4. `__delitem__(self, key)` - remove the key-value item as specified by `key`.
5. `keys()` - return a list containing all the keys in the hashtable.

We recommend that you implement hashtables using the chaining method of collision avoidance and table doubling to control the load factor and space consumption of the hash table. You may not use Python dictionaries inside the implementation of your hash table!

3 Running Your Code

- To run, execute `python compress.py` This file runs a the exec huff method on a simple dictionary that maps each character to its frequrency in a string.
- To test, you can run the provides test cases with `python compress.py test`.

4 Deliverables

Your repo folder should contain all the files from the zip. These are the ones you need to modify:

- `hashtable.py` - containing your hash table implementation and unit tests
- `README.md` - where you explain your code.
- Hours - record the number of hours you spent on writing and debuging your code. Put your answers in the `README.md`.

5 Testing

There are tests in `tests.txt` file which you can run by executing *run compress.py test*. We also encourage you to write your own unit tests. A good way to do this is to use **assert** statements. So if your function must return `True` for given inputs, the test case looks like

```
assert f(i) == True
```

You can then put all your tests inside an `if` statement.

```
if __name__ == "__main__":
  # my unit tests
```

The tests will be executed when you execute this python script but not when you include this file into another script and run that.