Wesly Lim

Design Document

Goals:

- Implement an n-way set associative cache
- Able to set any associativity and cache size
- Type safe (keys and values can be of any type)
- Clients can implement and use any replacement strategy

Cache Design:

- Used Python 3.6.2 to compile and run the program
- Used 3 classes to hold the 3 objects that make up the cache system
    - Cache, sets, cache items
- Cache holds m-sets, which is stored in the defaultdict() data structure
    - The defaultdict() is essentially the same as a regular dictionary that does not raise KeyError.
    - Replacement algorithm parameter passed upon initialization
    - Item placement in a set is determined by the (key's hash % size of the set)
- Sets hold n-items, depending on the associativity, which is stored in a defaultdict()
- Uses separate queue data structure to keep ordering of items in the set to use replacement algorithms in O(1) or O(log n) as opposed to O(N) in the previous implementation of OrderedDict()
- CacheItems contain relevant data that may be used for replacement algorithms
- This cache takes O(1) to get, set, and delete (it takes O(1) to get, set, and delete in a defaultdict(), so O(1) + O(1) → O(1)) *
- This cache takes O(1) to preserve order for LRU and MRU as opposed to the previous implementation of the OrderedDict() that takes O(n) to maintain sorted-ness.

For-client Design:

- Each file is well commented to depict functionality in detail without being excessive
- Variable names are obvious and follow a specific, consistent convention
- Variables are private, only accessible within the class or by public get functions
- Files separated between cache creation, replacement algorithm creation, and testing
    - Cache creation is organized into 3 classes, which is easy to read and modify
    - Replacement algorithms can be made and modified in cache_algorithm.py, which allows clients to easily implement their own work without modifying the internals of the cache (this was recently changed)
    - Test Driven Development was used to ensure clean and correct builds using many assert methods for each case
        - Each test and long trial is easily modifiable for custom tests
- Additional replacement algorithms now easier to implement and do not change the internals of the source code of the cache.

Replacement Algorithms:

LRU:

- Evicts item in a set in a cache that is the least recently used
    - Sorted by time_visited (old to new), and pops off the old

MRU:

- Evicts item in a set in a cache that is the most recently used
    - Sorted by time_visited (new to old), and pops off the new

* Source: https://stackoverflow.com/questions/8176513/ordereddict-performance-compared-to-deque

Usage:

```
from cache_setup import CacheSetup
from cache_algorithm import LRU_replacement, MRU_replacement
```

Use this code snippet when wanting to use the implemented n-way set associative cache via any version of Python 3.6.2.

```
cache = CacheSetup(num_sets, num_assoc, replacement())

cache.set_item(key, value)

cache.get_item(key)

cache.get_item(key).get_val()

cache.get_item(key).get_key()

cache.remove_item(key)
```

Basic methods to use when implementing the code.

Testing:

```
python3 cache_test.py
```