CS 321 Data Structures (Spring 2022)

Programming Assignment #1 (60 points), Due on 1/20/2022, Thursday (11:00 PM)

Introduction:
This programming assignment asks you to design a **cache** implementation using linked list data structure. That is, write a `Cache` class having at least the following public methods – constructor, getObject, addObject, removeObject, clearCache and some others. The data to be stored in cache should be generic objects. Also, write a test program to test your cache implementation.

Description:
A cache is a storage in memory. If a data item has a copy in cache, application can read this data item from cache directly. The usage of cache is as follows. Whenever an application requires a data item, it searches the cache first. If it is a cache hit, then the cache returns the data item to the application and the data item will be move to the first position in the cache (we call it the Most Recently Used MRU scheme). On the other hand, if it is a cache miss, then the application needs to read the data item from disk and then the data item from disk will be added to the first position of the cache. Note that if the cache is full, the last entry (oldest one) in the cache will be removed before a new entry can be added.

Similarly, whenever an application writes a data item to disk, the system will perform the same write operation to the cache copy of the data item (if any) and then move it to the first position in cache. Note that the `write` operation is equivalent to a `remove` operation followed by an `add` operation.

One-level Cache:
A single-level cache and it works as described above.

Two-level Cache:
A 2nd-level cache sits behind the 1st-level cache. Usually, the 2nd-level cache is much bigger than the 1st-level cache. Assume the 2nd-level cache contains all data in the 1st level cache, which is called (`multilevel inclusion property`). Two-level cache works as follows:

```
Search cache1;
Increment the # of cache1 references;
if cache1 hits
   then Increment the # of cache1 hits;
        Move the hit data item to the top of both cache.
   else Search cache2;
        Increment the # of cache2 references;
        if cache2 hits
           then Increment the # of cache2 hits;
                Move the hit data item to the top of cache2;
                Add the data item to the top of cache1;
           else Add the data item to the top of both cache;
```

Hit Ratio:
Some terms used to define hit ratio are:
HR_1: 1st-level cache hit ratio
HR_2: 2nd-level cache hit ratio

HR: (global) cache hit ratio

NH_1: number of 1st-level cache hits
NH_2: number of 2nd-level cache hits
NH: total number of cache hits ($= NH_1 + NH_2$, in case of two-level cache simulation)

NR_1: number of references to 1st-level cache
NR_2: number of references to 2nd-level cache (= number of 1st-level cache misses)
NR: total references to cache ($= NR_1$, in case of two-level cache simulation)

- One-level cache: $HR = \frac{NH}{NR}$
- Two-level cache: $HR1 = \frac{NH_1}{NR_1}$ $\quad HR_2 = \frac{NH_2}{NR_2}$ $\quad HR = \frac{NH}{NR} = \frac{NH_1+NH_2}{NR_1}$

What you need to do:

1. Create a directory ~/`cs321/lab1` for this assignment.
2. Write a `Cache` class.
3. Execute the following command (assuming you are in the directory ~/`cs321/lab1`) to make a link to the text file. Don't copy this file as it is quite large!

   ```
   ln -s /home/JHyeh/cs321/labs/lab1/files/Encyclopedia.txt
   ```

4. Write a test program `Test.java`. It's usage should be

   ```
   java Test 1 <cache size> <input textfile name> or
   java Test 2 <1st-level cache size> <2nd-level cache size> <input textfile name>
   ```

   The cache size(s) and the text file should be input as command line arguments. Your program should create a cache (option 1) or two cache (option 2) with the specified size(s) and read in the input text file word by word. For each word, search the cache(s) to see whether there is a cache hit and update the cache accordingly. I will use the file `Encyclopedia.txt` and a small text file `small.txt` (in the same directory as the Encyclopedia.txt does) to test your program.

5. Your program should output the cache hit ratio(s) after reading all words from the input text file. You can find the sample outputs, `result1k2k`, `result1k5k` and `result.small`, in the directory

   ```
   /home/JHyeh/cs321/labs/lab1/files/
   ```

6. Measure the elapsed time (in Milli-seconds) to run your program

   ```
   java Test 2 1000 2000 Encyclopedia.txt
   ```

   and report the time required in a README file. You can call the method `System.currentTimeMillis()` to measure the time.

Submission:

Submit your program(s) from `onyx` by copying all of your files to an empty directory (with no subdirectories) and typing the following FROM WITHIN this directory:

```
submit jhyeh cs321 p1
```