COMP108 Data Structures and Algorithms

Assignment 1

Deadline: Thursday 15th March 2018, 4:00pm

Important: Please read all instructions carefully before starting the assignment.

Basic information

- Assignment: 1 (of 2)
- Deadline: Thursday 15th March 2018 (Week 7), 4:00pm
- Weighting: 20%
- Electronic Submission:
  `https://sam.csc.liv.ac.uk/COMP/Submissions.pl?qryAssignment=COMP108-1`
- What to submit: a java file named A1Paging.java
- Learning outcomes assessed:
  - Be able to apply the data structure arrays and their associated algorithms
  - Be able to apply a given pseudo code algorithm in order to solve a given problem
  - Be able to apply the iterative algorithm design principle
- Marking criteria:
  - Correctness: 90%
  - Programming style: 10%

Background - The paging/caching problem

The paging/caching problem arises from memory management, between external storage and main memory and between main memory and the cache. We consider a two-level virtual memory system. Each level can store a number of fixed-size memory units called **pages**. The slow memory contains N pages. The fast memory has a fixed size $k < N$ which can store a subset of the N pages. We call the fast memory the **cache**. Given a *request* for a page is issued. If the requested page is already in the cache, we call it a ***hit***. If the requested page is not in the cache, we call it a ***miss*** and we have to ***evict*** (remove) a page from the cache to make room for the requested page.

Different eviction algorithms use different criteria to choose the page to be evicted. We consider the following eviction algorithms. To illustrate, we assume the cache contains 3 pages with initial ID content 20, 30, 10 and the sequence of requests is 20, 30, 5, 30, 5, 20.

- **No eviction.** This algorithm does not evict any pages, i.e., the cache stays as the initial content. Then the hit (h) and miss (m) sequence will be **hhmhmh** since 20 is a hit, 30 hit, 5 miss, 30 hit, 5 miss, 20 hit. There are **4 h and 2 m**.
- **Evict largest.** This algorithm evicts the largest ID when there is a miss. The hit and miss sequence will be **hhmmhm**, with **3 h and 3 m**. The following table shows how the cache content changes.

request	cache beforehand	hit/miss	cache afterward	remarks
20	20, 30, 10	h	no change	
30	20, 30, 10	h	no change	
5	20, 30, 10	m	20, 5, 10	30 is largest and evicted
30	20, 5, 10	m	30, 5, 10	20 is largest and evicted
5	30, 5, 10	h	no change	
20	30, 5, 10	m	20, 5, 10	30 is largest and evicted

- **Evict FIFO.** This algorithm evicts the pages in the FIFO (first-in-first-out) principle. In this case, we also assume that the initial cache content is added one by one with 20 first, then 30, then 10. The hit and miss sequence will now become **hhmhhm** with 4 hits and 2 miss.

request	cache beforehand	hit/miss	cache afterward	remarks
20	20, 30, 10	h	no change	
30	20, 30, 10	h	no change	
5	20, 30, 10	m	5, 30, 10	20 in the cache longest and evicted
30	5, 30, 10	h	no change	
5	5, 30, 10	h	no change	
20	5, 30, 10	m	5, 20, 10	30 in the cache longest and evicted

- **Evict LFU.** This algorithm evicts the pages in the LFU (least frequently used) principle. This means that the algorithm keeps a count of how many times a page in the cache has been requested and if there are two or more pages with the same count, the one with a smaller index (i.e., leftmost such page in the cache) is evicted. *We assume that the initial cache content each has* ***a count of 1*** *to start with.* The hit and miss sequence is **hhmhhh** with 5 hits and 1 miss. Note that the frequency count is reset if a page is evicted, in other words, a page entering the cache will have a count of 1.

request	cache beforehand	hit/miss	cache afterward	remarks
20	20, 30, 10	h	no change	
30	20, 30, 10	h	no change	
5	20, 30, 10	m	20, 30, 5	20 and 30 have count 2 while 10 has count 1 and is evicted
30	20, 30, 5	h	no change	
5	20, 30, 5	h	no change	
20	20, 30, 5	h	no change	

The program A1Paging.java

A skeleton program called **A1Paging.java** can be downloaded from
`http://www2.csc.liv.ac.uk/~pwong/teaching/comp108/201718/assess/A1Paging.java`
The program has a number of methods already written which you must **NOT** change, including

```
public static void main(String[] args)
static void init_array(int[] array, int n, int value)
static void print_array(int[] array, int n)
static void copy_array(int[] a1, int[] a2, int n)
```

The main method takes care of the input, reading (i) the size of cache, (ii) the cache content, (iii) the number of page requests, and (iv) the sequence of page requests. It then calls each of the methods for each eviction algorithm, each of which is supposed to report the hit-miss sequence and the number of hits and misses. The order of calling of the eviction algorithm has been determined

and you should not change the order, otherwise, you may lose all your marks. The header of each eviction method has been provided with four arguments:

- `int[] cache`: an integer array storing the initial cache content
- `int c_size`: the size of the cache, i.e., cache content is stored in `cache[0]..cache[c_size-1]`
- `int[] request`: an integer array storing the sequence of page requests
- `int r_size`: the number of page requests, i.e., requests stored in `request[0]..request[r_size-1]`

Your tasks

There are four tasks you need to do. Each task is associated with each of the evicting algorithm, each should print and only print the following.

(i) the hit-miss sequence in the form `<[h|m]*>` (see Test Cases below for examples), and

(ii) `x h y m` where x is the number of hits and y is the number of misses, hence, `x + y` should be equal to `r_size`.

- **Task 1 (30%)** Implement the `no_evict()` method that does not evict any page. It should iterate the request sequence to determine for each request whether it is a hit or a miss. Hint: you can make use of sequential search algorithm.
- **Task 2 (20%)** Implement the `evict_largest()` method which on misses evict the page with the largest ID value and replace it with the page being requested. No other pages in the cache should be changed. Hint: you can make use of finding minimum/maximum location algorithm.
- **Task 3 (20%)** Implement the `evict_FIFO()` method to evict pages in a first-in-first-out principle. You should assume that the initial cache content enters the cache in the order of `cache[0], cache[1], ...,` in other words, `cache[0]` should be evicted first if needed, then `cache[1]`, and so on. Hint: you may keep an index like `head` in queue and use the remainder operator % when you need to wrap around the index from c_size back to 0.
- **Task 4 (20%)** Implement the `evict_LFU()` method to evict the page which is least frequently used (requested). You should assume that each of the initial cache content is used once. When a cache page `cache[i]` is evicted and replaced by a new page, the counter of `cache[i]` should be reset to 1. When there are multiple pages with the same frequency of usage, the "leftmost" such page should be evicted; e.g., `cache[x]` and `cache[y]` have the same lowest frequency of usage and `x < y`, then `cache[x]` should be evicted and not `cache[y]`. Hint: you may use an additional array to store the frequency of usage of the cache element; you'll need to increment this counter array appropriately when a page is requested, evicted, or added to the cache.
- **Task 5 (10%)** (This is not a separate task.) In your implementation, you should keep a good programming style. Marks will be awarded based on consistent use of brackets, indentation, and meaningful variable names.

Test Cases

Below are some sample test cases and the expected output so that you can check and debug your program. Make sure that before submission you remove or comment all other print statements that you use for debugging. The order of output should be `no_evict`, `evict_largest`, `evict_fifo`, `evict_lfu` as determined in the main method. Your program will be tested in this order, therefore,

do NOT change the order (you should not change the main method anyway)! Your program will be marked by five other test cases that have not be revealed.

Test cases	Input	Output
#1	3 20 30 10 6 20 30 5 30 5 20	hhmhmh 4 h 2 m hhmmhm 3 h 3 m hhmhhm 4 h 2 m hhmhhh 5 h 1 m
#2	5 10 20 30 40 50 4 15 50 10 20	mhhh 3 h 1 m mmhh 2 h 2 m mhmm 1 h 3 m mhmh 2 h 2 m
#3	4 20 30 10 40 14 40 40 30 30 20 5 5 5 15 15 15 15 10 40	hhhhhmmmmmmmhh 7 h 7 m hhhhhmhhmhhhhm 11 h 3 m hhhhhmhhmhhhhh 12 h 2 m hhhhhmhhmhhhmh 11 h 3 m

These test cases can be downloaded as test01.txt, test02.txt, test03.txt on the assessment page: `http://www2.csc.liv.ac.uk/~pwong/teaching/comp108/201718/assess.html`
You can run the program easier by typing `java A1Paging < test01.txt` in which case you don't have to type the input over and over again. The test files have to be stored in the same folder as the java and class files.

Penalties

The following penalties apply but will not deduce the marks below the passing mark.

- UoL standard penalty applies: 5 marks shall be deducted for each 24 hour period after the deadline. Submissions submitted after 5 days past the deadline will no longer be accepted. Any submission past the deadline will be considered at least one day late. Penalty days include weekends and Easter break.
- If your code does not compile successfully, 5 marks will be deducted. If your code compile to a class of a different name from A1Paging, 5 marks will be deducted. If your output does not follow the same format as expected, 5 marks will be deducted. If your output does not follow the order of the algorithms as expected, 5 marks will be deducted.

Plagiarism

This assignment is an individual work and any work you submitted must be your own. You should not collude with another student, or copy other's work. Any plagiarism case will be handled following the University guidelines.