COMP108
Data Structures and Algorithms

Introduction to Programming Assignment

Professor Prudence Wong

pwong@liverpool.ac.uk

2022-23

Basic information

- Deadline: Friday 21st April 2023, 5:00pm
- Weighting: 15%
- Submission to codegrade via Canvas
- Learning outcomes assessed:
    - Be able to apply the data structure arrays & linked lists 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
    - Be able to carry out simple asymptotic analyses of algorithms
- Marking criteria:
    - Correctness: 80%
    - Time complexity analysis: 20%
- There are two parts of the assignment (Part 1: 32.5% & Part 2: 47.5%)

Part 1

Part 1. Paging/Caching

- two level virtual memory system
- slow memory contains more **pages** than fast memory called a **cache**
- requesting a page
  - if page is already in cache $\Longrightarrow$ ***hit***
  - if page not in cache $\Longrightarrow$ ***miss***
  - in case of a miss, need to *evict* a page from cache to make room
  - ***eviction algorithms***

Eviction algorithm - noEvict

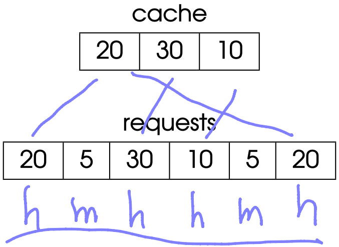
cache: 20 30 10 sequence of requests: 20 5 30 10 5 20

output: hmhhmh 4 h 2 m

nothing is evicted, i.e., cache contains exactly what it contains initially

⟹ counting how many requests are (are not resp.) in initial cache

hit-miss-pattern



cache

20	30	10

requests

20	5	30	10	5	20
h	m	h	h	m	h

Eviction algorithm - evictLRU (least recently used)

cache: 20 30 10 sequence of requests: 20 5 30 10 5 20

output: hmmmhm 2 h 4 m cache at the end 10 5 20

cache

20	30	10

requests

20	5	30	10	5	20

	cache	explanation
	20 30 10	initial cache 20 least recent; 30 2nd; 10 recent
20	20 30 10	20 is a hit 20 recent; 30 least recent; 10 2nd
5 miss	20 5 10	30 is evicted because its recent request is earlier than 10 20 2nd recent; 5 recent; 10 least recent
30	20 5 30	10 is evicted because its recent request is earliest 20 least
10	10 5 30	20 is evicted because its recent request is earliest 5 least
5	10 5 30	5 is a hit 10 2nd recent; 5 recent; 30 least recent
20	10 5 20	30 is evicted because its recent request is earliest

The program COMP108Paging.java - to be submitted

You are going to implement the four algorithms in four methods correspondingly.

```
noEvict(int[] cArray, int cSize, int[] rArray, int rSize)
evictLRU(int[] cArray, int cSize, int[] rArray, int rSize)
```

`cArrary`: cache content, `cSize`: its size; `rArrary`: sequence of requests, `rSize`: its size

The full header of these methods:

```
static COMP108PagingOutput noEvict(...)
static COMP108PagingOutput evictLRU(...)
```

You are **encouraged** to add your own methods in COMP108Paging.java

The program COMP108PagingOutput.java

NOT to be submitted! Don’t edit it, any changes won’t be graded!

A class is defined: `class COMP108PagingOutput`

Constructor: `public COMP108PagingOutput()`

The class has four attributes:

```
public int hitCount;
public int missCount;
public String hitPattern;
public String cache;
```

Already implemented printing method: `public void print()`

How to use it in COMP108Paging.java?

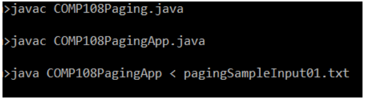
```
static COMP108PagingOutput noEvict(int[] cArray, int cSize, int[] rArray, int rSize) {
      COMP108PagingOutput output = new COMP108PagingOutput(); // this line is already there
      output.hitPattern += “m”;
      output.hitCount++;
      output.missCount++;
      output.cache = arrayToString(cArray, cSize); // arrayToString() already implemented
      return output; // this line is already there
  }
```

The program COMP108PagingApp.java

NOT to be submitted! Don't edit it, any changes won't be graded!

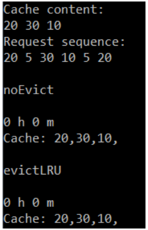
Help you to test your program only

how to run



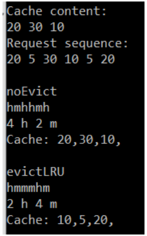
```
>javac COMP108Paging.java

>javac COMP108PagingApp.java

>java COMP108PagingApp < pagingSampleInput01.txt
```

output before implementation



```
Cache content:
20 30 10
Request sequence:
20 5 30 10 5 20

noEvict

0 h 0 m
Cache: 20,30,10,

evictLRU

0 h 0 m
Cache: 20,30,10,
```

output of correct implementation



```
Cache content:
20 30 10
Request sequence:
20 5 30 10 5 20

noEvict
hmhhmh
4 h 2 m
Cache: 20,30,10,

evictLRU
hmmmhm
2 h 4 m
Cache: 10,5,20,
```

Your tasks

Implement the four eviction algorithms

- Task 1.1: `noEvict`
- Task 1.2: `evictLRU`

Test cases - sample input & output

- Refer to the assignment
- You can assume that the input given are valid.
- The assignment has given three sample input for you to check your program.
- You should think of other (edge) cases to test your program.

Sample:

Test cases	Input	noEvict	evictLRU
#1	3 20 30 10 6 20 5 30 10 5 20	hmhhmh 4 h 2 m 20,30,10,	hmmmhm 2 h 4 m 10,5,20,

You will be graded on another FIVE test cases not revealed.

Part 2

Part 2. List Accessing Problem

- file cabinet containing files with (unsorted) IDs - organised as a list
- a sequence of request for files - stored in an array
- upon each request, we want to
  - determine whether the file is already in the cabinet
  - how many comparisons needed to find this out?
  - put the file back after usage - by different ``reorganisation'' algorithms

Append If Miss

initial: 20, 30, 10 sequence of requests: 20, 30, 5, 30, 5, 20

output:

```
appendIfMiss
1,2,3,2,4,1
5 h 1 m
From head to tail:  20,30,10,5,
From tail to head:  5,10,30,20,
```

request	list beforehand	hit?	# comparisons	list afterward
20	20 30 10	yes	1	no change
30	20 30 10	yes	2	no change
5	20 30 10	no	3	20 30 10 5
30	20 30 10 5	yes	2	no change
5	20 30 10 5	yes	4	no change
20	20 30 10 5	yes	1	no change

Frequency Count

initial: 20, 30, 10   sequence of requests: 20, 30, 5, 30, 5, 20

output:

```
init frequency: 1 1 1
```

```
    freqCount
    1,2,3,2,4,2,
    5 h 1 m
    From head to tail:  30,20,5,10,
    From tail to head:  10,5,20,30,
    Frequency from head to tail:  3,3,2,1,
```

request	list beforehand	hit?	# comp	list afterward	freq afterward
20	20 30 10	yes	1	no change	2 1 1
30	20 30 10	yes	2	no change	2 2 1
5	20 30 10	no	3	20 30 10 5	2 2 1 1
30	20 30 10 5	yes	2	30 20 10 5	3 2 1 1
5	30 20 10 5	yes	4	30 20 5 10	3 2 2 1
20	30 20 5 10	yes	2	no change	3 3 2 1

3 2 1 2 ×

The program COMP108Cab.java - to be submitted

Two attributes: `head` and `tail`

You are going to implement the ~~three~~ two algorithms in ~~three~~ two methods correspondingly.

```
public COMP108CabOutput appendIfMiss(int rArray[], int rSize)
public COMP108CabOutput freqCount(int rArray[], int rSize)
```

`rArrary`: sequence of requests, `rSize`: its size

In these methods, you can access the `head` and `tail` if needed, e.g.:

- traversing from head to tail:

```
curr = head;
while (curr != null)
    curr = curr.next;
```

- traversing from tail to head:

```
curr = tail;
while (curr != null)
    curr = curr.prev;
```

The program COMP108Cab.java (cont'd)

The class also contains a few pre-written methods to help with the tasks.

```
public void insertHead(COMP108Node newNode)
public void insertTail(COMP108Node newNode)
public COMP108Node deleteHead()
public void emptyCab()
public String headToTail()
public String tailToHead()
public String headToTailFreq()
```

You must NOT change the content of these methods

`insertHead()`,`insertTail()`,`deleteHead()`,`emptyCab()`: creates/demolishes the list

`headToTail()`,`tailToHead()`,`headToTailFreq()`: help with output only, DONT'T use them for other purposes

The program COMP108Node.java

NOT to be submitted! Don't edit it, any changes won't be graded!

A class is defined: `class COMP108Node`

Constructor: `public COMP108Node(int i)`

The class has the following attributes:

```
public int data;
public COMP108Node next;
public COMP108Node prev;
public int freq; // only to be used in freqCount algorithm
```

Output should be stored in an object of the class COMP108CabOutput

NOT to be submitted! Don't edit it, any changes won't be graded!

A class is defined: `class COMP108CabOutput`

Constructor: `public COMP108CabOutput(int cabSize,` ~~`int graphSize`~~`)`

The class has the following attributes:

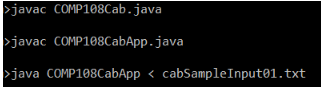
```
    public int hitCount;
    public int missCount;
    public int[] compare;
    public String cabFromHead;
    public String cabFromTail;

    public String cabFromHeadFreq;
```

How to use it in COMP108Cab.java?

```
public COMP108CabOutput appendIfMiss(int rArray[], int rSize) {
   COMP108CabOutput output = new COMP108CabOutput(rSize, ~~1~~); // this line is already there
   output.cabFromHead = headToTail(); // this line is already there
   output.cabFromTail = tailToHead(); // this line is already there
   output.cabFromHeadFreq = headToTailFreq(); // this line is already there for freqCount()
   return output; // this line is already there

}
```

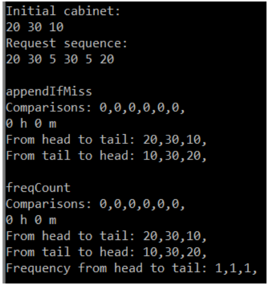
The program COMP108CabApp.java

NOT to be submitted! Don’t edit it, any changes won’t be graded! Help you to test your program only

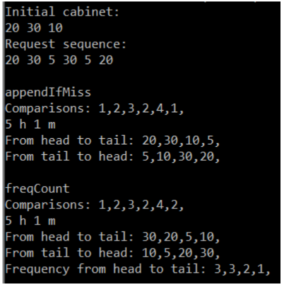


```
>javac COMP108Cab.java

>javac COMP108CabApp.java

>java COMP108CabApp < cabSampleInput01.txt
```

Output (left: before implementation; right: correct implementation)



```
Initial cabinet:
20 30 10
Request sequence:
20 30 5 30 5 20

appendIfMiss
Comparisons: 0,0,0,0,0,0,
0 h 0 m
From head to tail: 20,30,10,
From tail to head: 10,30,20,

freqCount
Comparisons: 0,0,0,0,0,0,
0 h 0 m
From head to tail: 20,30,10,
From tail to head: 10,30,20,
Frequency from head to tail: 1,1,1,
```



```
Initial cabinet:
20 30 10
Request sequence:
20 30 5 30 5 20

appendIfMiss
Comparisons: 1,2,3,2,4,1,
5 h 1 m
From head to tail: 20,30,10,5,
From tail to head: 5,10,30,20,

freqCount
Comparisons: 1,2,3,2,4,2,
5 h 1 m
From head to tail: 30,20,5,10,
From tail to head: 10,5,20,30,
Frequency from head to tail: 3,3,2,1,
```

Your tasks (65%)

Implement the three accessing/reorganising algorithms

- Task 2.1: `appendIfMiss`
- Task 2.2: `freqCount`

Test cases - sample input & output

`cabSampleInput01.txt`, `cabSampleInput02.txt`, `cabSampleInput03.txt`

- Refer to the assignment
- You can assume that the input given are valid.
- The assignment has given three sample input for you to check your program.
- You should think of other (edge) cases to test your program.

You will be graded on other FIVE test cases not revealed.

Worst Case Time Complexity

Worst Case Time complexity analysis

- Fill in comment section in the beginning of COMP108Paging.java and COMP108Cab.java
- No separate file to be submitted
- For Part 1, you can use in your formula:
  - n to represent the number of requests,
  - p the cache size
- For Part 2, you can use in your formula:
  - f: the initial cabinet size,
  - n: the number of requests,
  - d: the number of distinct requests
- Give also a short justification of your answer.

More information

How to get help on the assignment?

- Lab exercises in Weeks 04-08 are all relevant to the assignment. Do the exercises and check solutions when they are released on Mondays.
  - Week 04: simple loops
  - Week 05: sorting and searching on arrays
  - Week 06: nested loops
  - Week 07: list traversal, sequential search
  - Week 08: moving a node to head / tail
  - Week 09: dedicated to answering questions related to the assignment
- Post your questions on Canvas discussion board

Penalties

- UoL standard late penalty applies
- Make sure that you pass all the visible autotests on codegrade, including compiling and running with *App.java. Otherwise, you may get 0 marks.
- Marks will be deducted if
  - your code compile to a class of different name than COMP108Paging / COMP108Cab
- **No** in-built classes/methods (e.g., ArrayList, Array.sort(), Array.fill(), etc.) can be used.
  - For example, if you want to sort an array, you have to write your own code to do so and you are not allowed to use Array.sort(). Otherwise, you won't be able to tell the time complexity if you use these methods.
  - Using in-built methods would get half of the marks deducted (possibly below the passing mark).
- You must implement Part 2 using the concept of linked list. If you convert the file cabinet list to an array or other data structures before processing, you will lose all the marks.

Plagiarism / Collusion

DON'T share your code with anyone!

Last year, 11 COMP108 students had been awarded a mark of 0.

- *Plagiarism occurs when a student misrepresents, as his/her own work, work in the public domain, written or otherwise, of any other person (including another student) or of any institution.*
- *Collusion is the active cooperation of two or more students to produce work, including knowingly allow academic work to be presented by another student as if it's his/hers, and providing work to another student.*

If you encounter difficulties, better make use of the lab sessions and my office hours to ask questions.

Summary: Introduction to Programming Assignment

For note taking