Faculty of engineering

Ain Shams university

CESS Program

**CSE 223: Operating Systems**

**Page replacement algorithms assignment**

**Submitted by:**

Ahmed Hesham Mohamed AlSaey - 16P6000

Sherif Ashraf Ahmed Morsy - 16P9033

Amr Mohamed El Sayed Ali Badawi – 16P3034

**Submitted to:**

Dr. Gamal Ibrahim Abdel Shafy

**Group:** 2

**1. Assumptions**

**1.1 General assumptions**

1. Assume that the randomly generated reference string is always bigger than the page frame because if it is not, then the algorithms would be of no use (values will be added sequentially all the time)
2. Assume that the random reference string maximum size is 300, and it is randomly filled with values ranging from 0 to 99
3. Assume that the random page frame number can vary from 1 to 20

**1.2 Algorithm specific assumptions**

1. **LFU**Assume that if more than one value in the frames have the same use frequency, apply FIFO on these values
2. **Second chance**

Assume that each time we reference a value from the reference string, we apply FIFO but if the reference string of the first-in value is 1, we change it to 0 and move on to the next value in the FIFO queue.

Assume that reference bit of new value is initially 0

Assume that referencing an existing value makes it’s reference bit 1

1. **Enhance second chance**

Assume that there is a next victim pointer that loops through the page frame in a circular path (when It reaches the last frame, the next victim frame will be the first), this pointer loops until it finds a candidate victim frame, this happens each time a new reference is taken from the reference string.

Assume that when looking whether or not the reference value is in the page frame, we do not move the next victim pointer, we only move it if we are looking for a replacement.

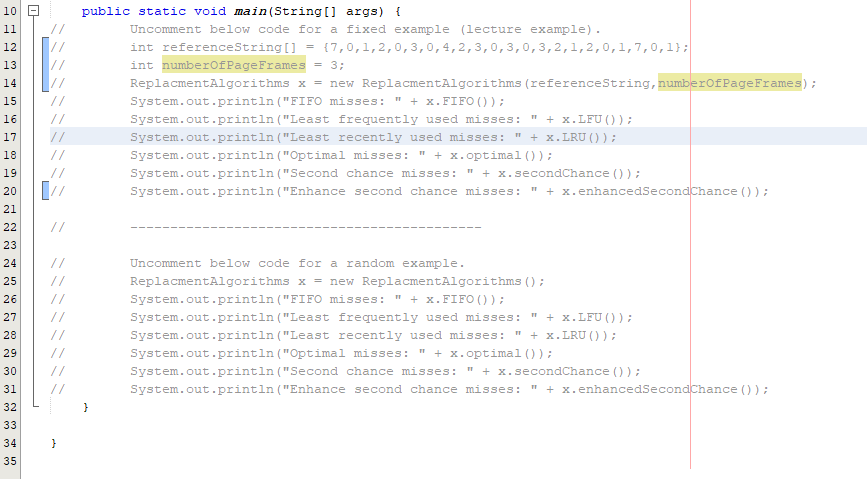
Assume that if looking for value in page frame and found it, we don’t change the reference bit or modify bit.

Assume that newly inserted values have reference bit of 1 and modify bit of 1 or 0 (randomly selected).

Actual algorithm explained below

**2. How to test the code**

Below you will see the main() function of the program

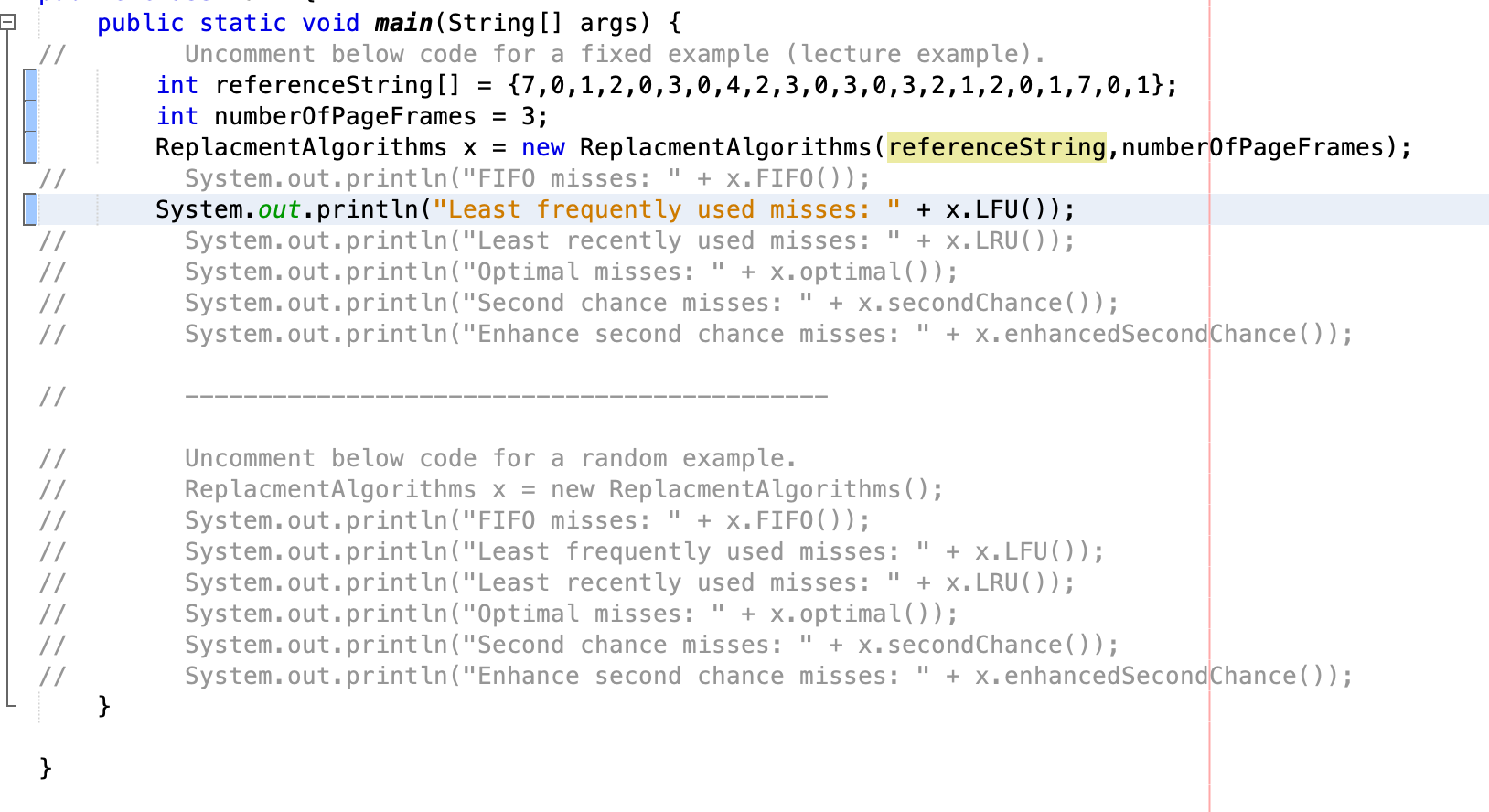
****

As mentioned in the comment, you can uncomment different parts of the code depending on whether or not you need to test a random or a fixed example.

Although possible, it is not recommended to uncomment multiple algorithms for testing to avoid confusion of different outputs, instead, please uncomment the code section with only one print statement at a time.

Output interpretation will be explained below.

Here is an example: (How to test the LFU algorithm on fixed example):



How to test LFU on random example:

**3. Code implementation**

All the test cases here are the example in the lecture:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Frames number: 3

**3.1 FIFO**

FIFO is implemented by having a dynamic array list that keeps track of new values’ index, added to the page frame, when a new value is added, it’s index is added to the last place in the array list, and if it replaces a value the index of the replaced value is removed from the array list, and again, the index of the new value is added to the end of the array list.

So the steps are:

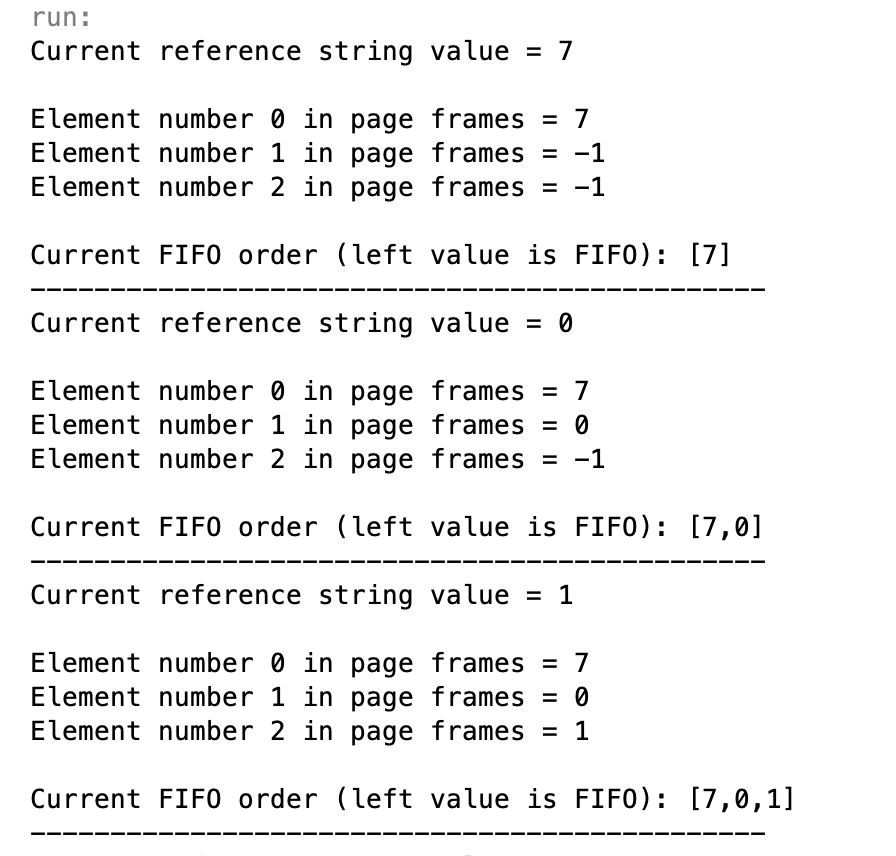
1. Look for the value in page frame

2. If not found, look for an empty frame

Here, the value to insert is 7, there is no 7’s in the frame so the next step is to find the next empty frame and add it, which is found.

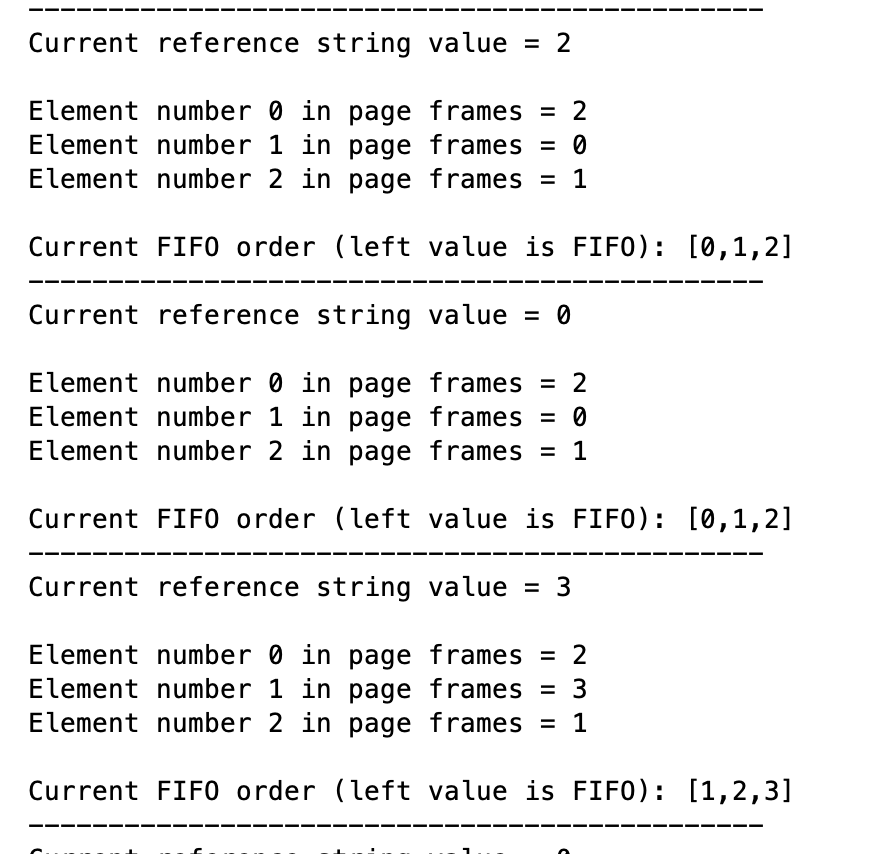
3. If not found, find and replace a value using FIFO

Here is a sample code:



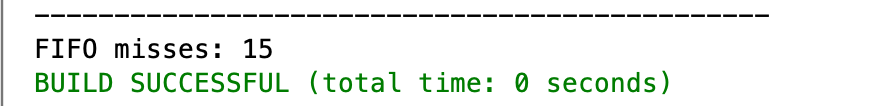
Notice the FIFO arraylist starting with first value in and ending with first value out (Page frame value is shown for clarity, but actual FIFO array list in the code contains indexes for these value (i.e. [0,1,2])

Same happened here, the next empty frame was at index 1



Notice that when 0 finding referenced value in page frame, no change in FIFO array list happened, this goes on until the end

2 replaced the first value in the FIFO array list, this value was removed and 2 was added to the end of the list



In the end, after the reference string is over, you will see the number of misses printed, keep in mind that misses happen whenever you don’t find the value in the page frame (Whether you replaced an old value or added the value in an empty frame)

**3.2 LFU**

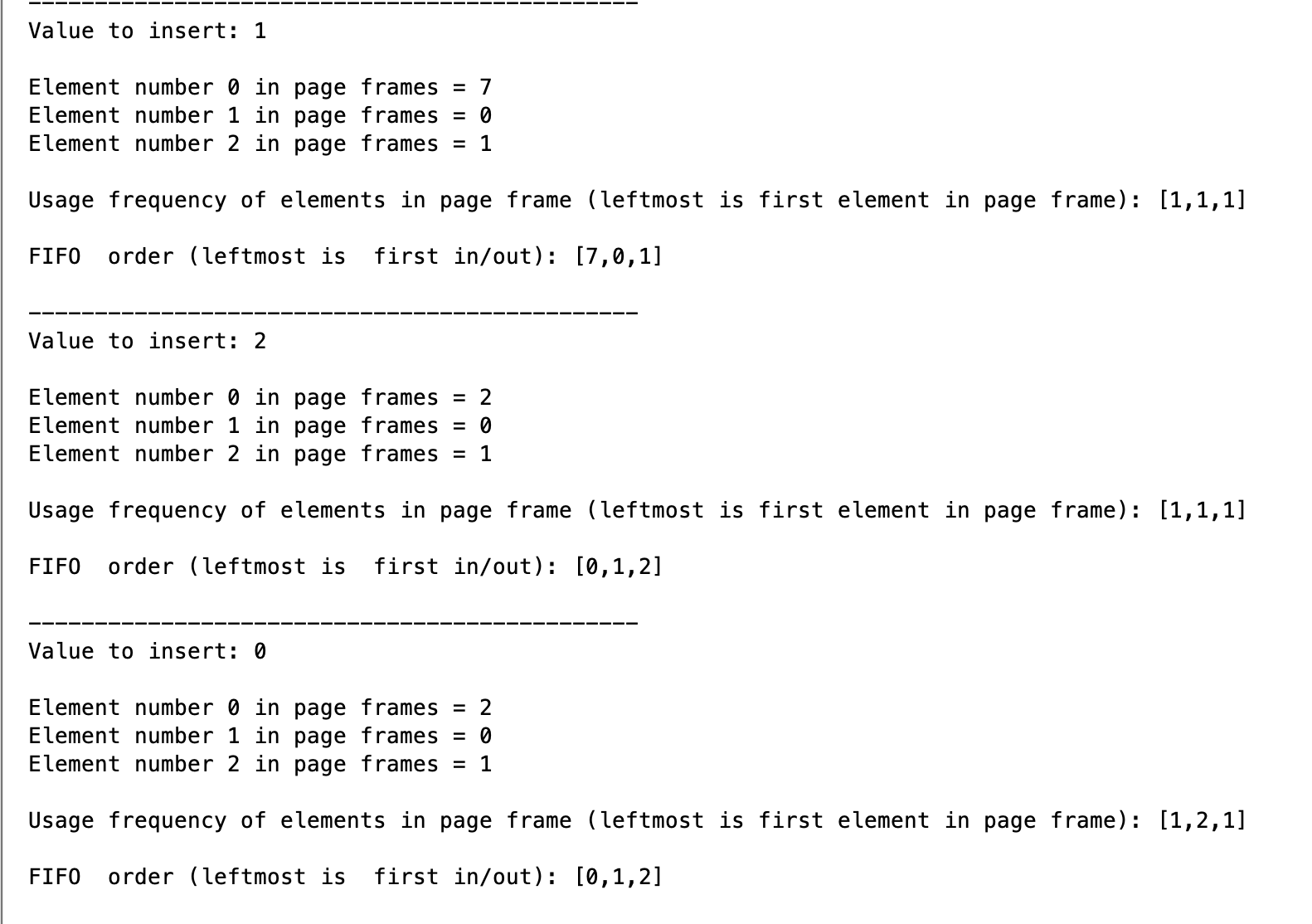
Here, we keep track of two arrays/array lists, the first one contains the count of each element in the page frame (this array’s size is, of course, the same as the page frame), and we keep an array list with the element FIFO order as explained in 3.1, each time a new value is referenced and found, it’s reference count increases by one. If a value is inserted newly, it’s initial reference count is 1. If a value replaced another value, the reference count of the value inside the same page frame pointer is reset (set to one). Selection algorithm works as follows:

1. Look if the value is in the page frame, if it is, increase it’s reference count by 1

2. Look if there is an empty page frame, if there is, add the value to it and set it’s reference count to 1

3. Look for the value with the least count to replace, if it is only one value, replace it and set the reference count of the frame of new value to 1

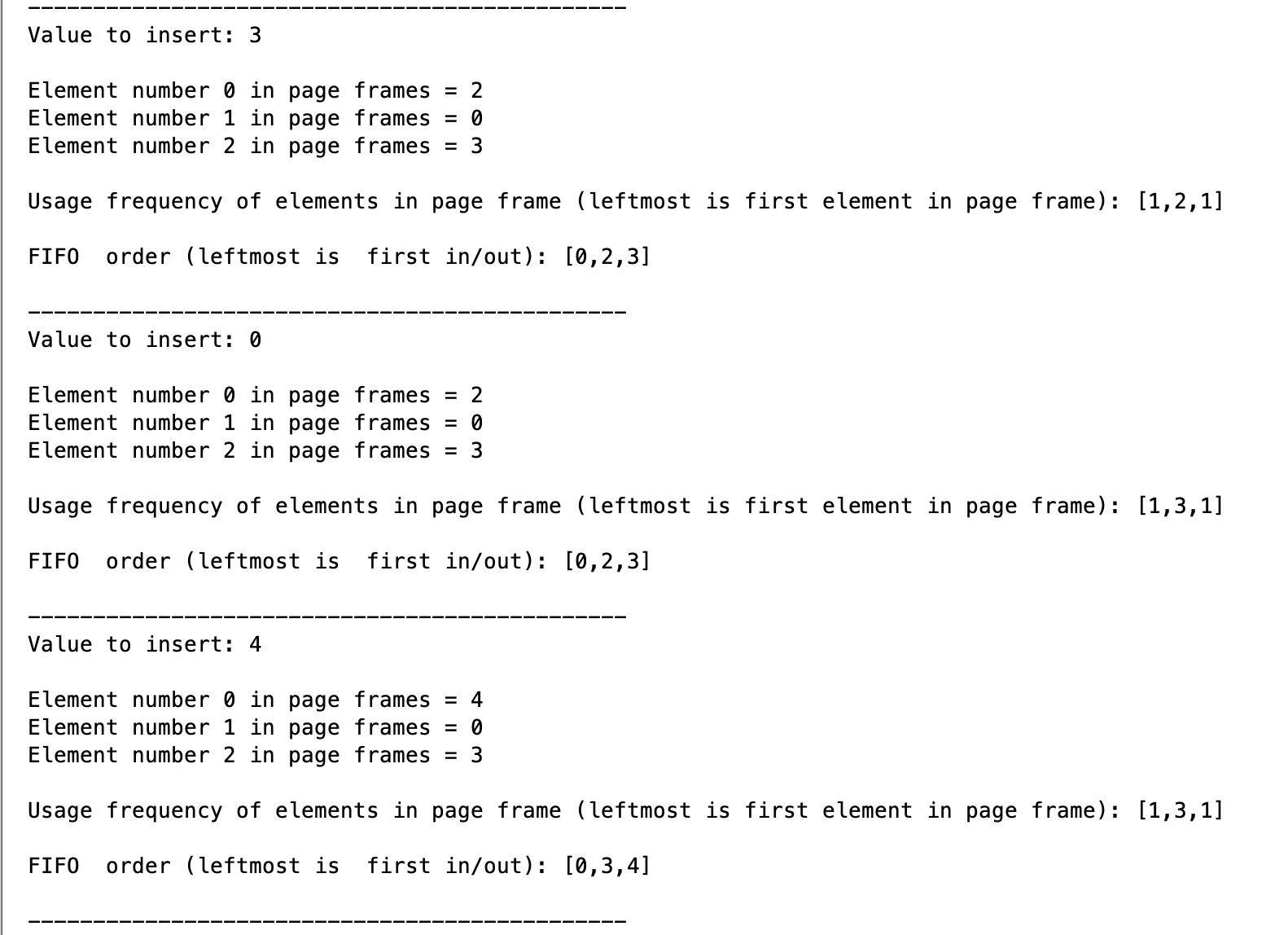
4. If there is more than one value with the least amount of count, replace first of these values in FIFO array list

Here is an output example:

Values 7,0,1 where added initially instead of the empty frames, you can see there reference count (Usage frequency) and there FIFO order in the last two lines

Here, 0 is found in page frame, so it’s reference count is increased by 1 (see last line)

Here, 2 needs to be added, but all the values have the same reference count, so we choose the first value in FIFO array list and replace it (which is 7)



4 needs to be added, but the first and the third value have the same reference count, the first value comes first in the FIFO array list, so it is replaced

0 is referenced again so we increase its reference count

Here, 3 needs to be added, but the first and third values in the page frame have the same reference count, and the third value come first in the FIFO list, so the first value is replaced

This goes on until the reference string is over and misses are printed, like the previous example, misses are counter when the value is not found in page frame, whether we replace an existing value or fill an empty frame.

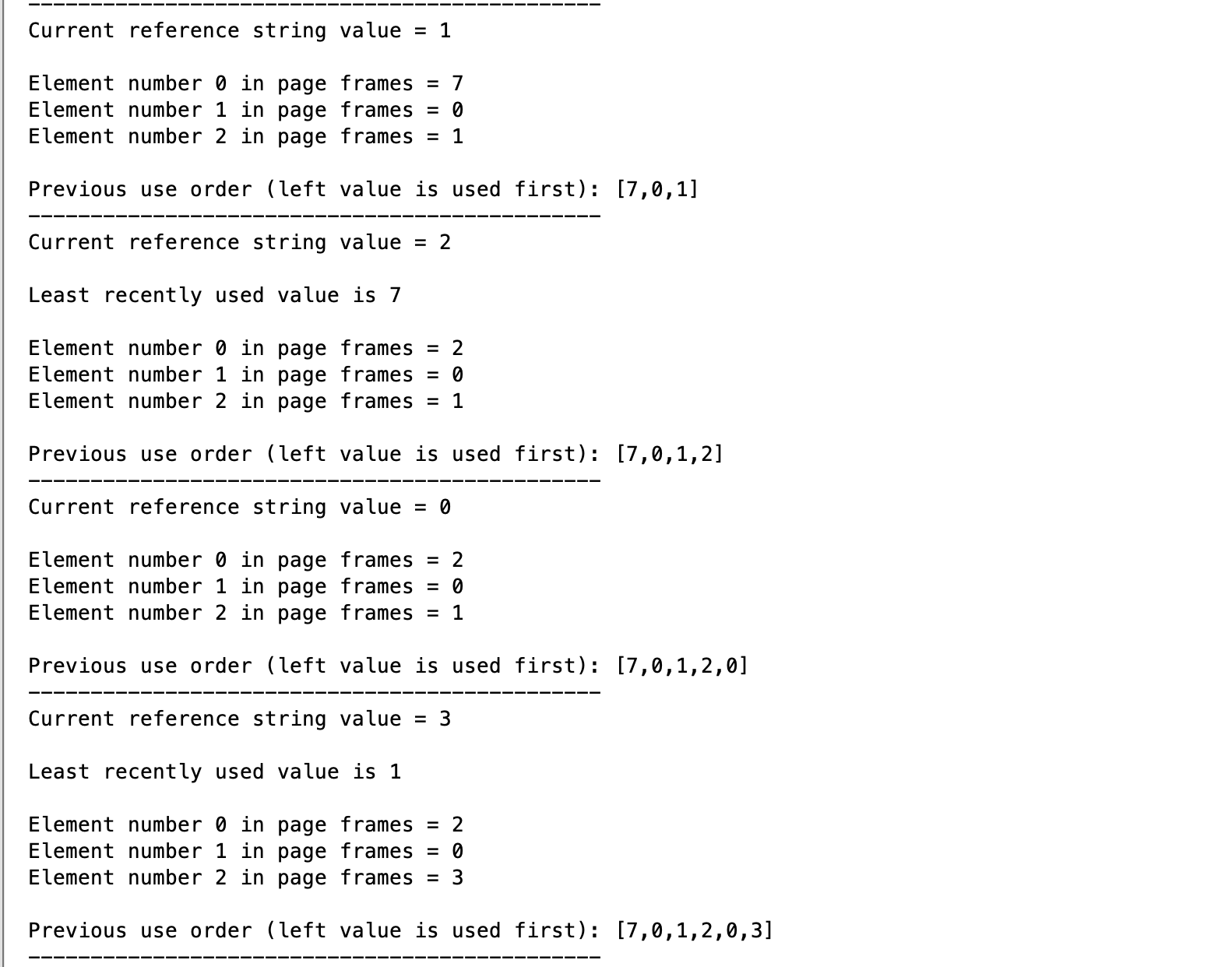
**3.3 LRU**

We don’t have to have special arrays or lists here that we manage or keep track of, the algorithm here is very simple:

1. Look if value is in page frame, if found, go to next reference value
2. If value not found, look for an empty frame to put value in it
3. If there are no empty frames, then we need to replace a value, we simply look at the previous reference string values, and compare them to the values inside the page frame to see which value is least recently used, (we move from the index of the current reference value – 1 towards index 0 in the reference frame)

Here is an example:

Values 7,0,1 entered sequentially just like the previous examples



3 is now referenced and it is not in the page frame, do we replace the LFU value in previous use order, which is 1

Here 0 is referenced, but is already in the page frames so nothing is done, keep in mind that 0 is now the most frequently used value and is the worst candidate for replacement (see previous use order)

When 2 was the current reference value, it needed to replace a value in the page frames, looking at the “Previous use order”, we see that 7 is the furthest value, so it is replaced.

Again, this keeps going until the reference string is over. Misses are printed in the end, and misses count when value is not found in page frames.

**3.4 Optimal algorithm**

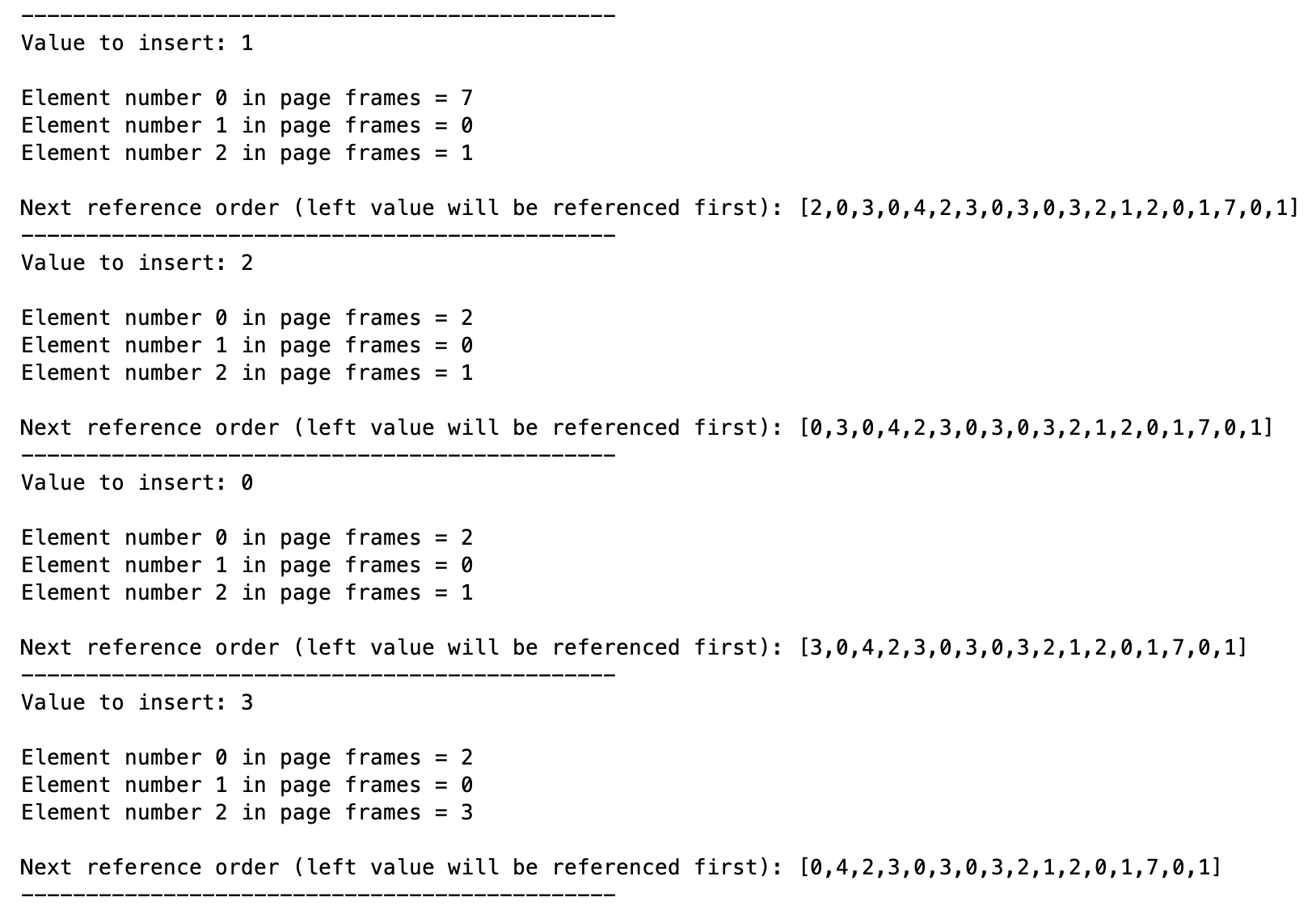
This is exactly the same as LRU, but instead of replacing the furthest value from the previously used elements, we replace the furthest value from the next (to be used) elements.

So the steps are:

1. Look if value is in page frame, if found, go to next reference value
2. If value not found, look for an empty frame to put value in it

If there are no empty frames, then we need to replace a value, we simply look at the next reference string values, and compare them to the values inside the page frame to see which value is to be used the furthest, (we move from the index of the current Example:

1. reference value + 1 towards the last index of reference string)



3 needs to be added, so we look at the next reference order and we see that 1 is the furthest away, so we replace it

0 is found in page frames so nothing is done

2 needs to be added, so we look at the next reference order (0,3,0,4…), we see that 7 is the furthest away, so we replace it

Again, values 7,0,1 added sequentially like the previous examples

This keeps going until reference string is over. Misses are again printed like the previous examples, and misses are counted when value is not found in page frame

**3.5 Second chance algorithm**

In this algorithm, each page frame has a corresponding reference bit, initially set to 0 when a new value is inserted into frame from reference string.

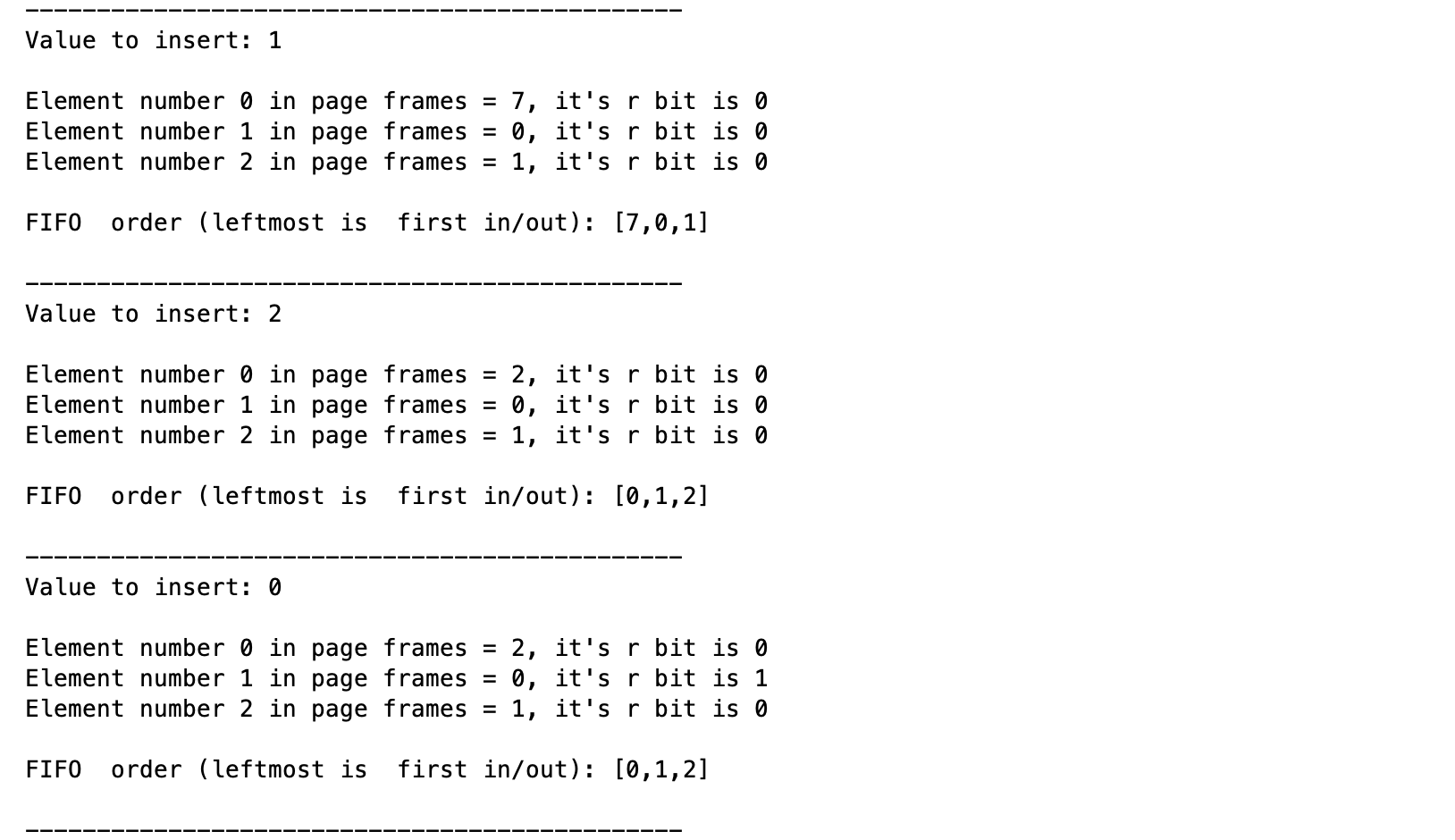
If a value is referenced and it is already in the page frame, we set the reference bit of it to zero.

This algorithm uses the FIFO variant, it is basically FIFO but if a new value is to be inserted, the next value in FIFO queue has a reference bit of 1, we go to the next value in the queue instead and do the same, until we either find a value with reference bit of zero or we reach the end of the FIFO queue, in this case, start looping the FIFO queue again.

So, the steps are:

1. Check to see if value exists in page frame, if so, set its reference bit to 1
2. Check to see if there is an empty frame, if so, put reference value in it and set its reference bit to 0
3. Check first element index in FIFO queue, if it has reference bit of 0, replace the value in this index with reference value.
4. If next index in FIFO queue has a corresponding reference bit of 1, set it to 0 and move on to the next FIFO element
5. Repeat step 4 until you see an index that has a corresponding reference string of 0, or you reach the end of the FIFO queue, in which case you start from the beginning of the queue, and it is guaranteed that you find an index with reference bit of 0 (due to step 4)

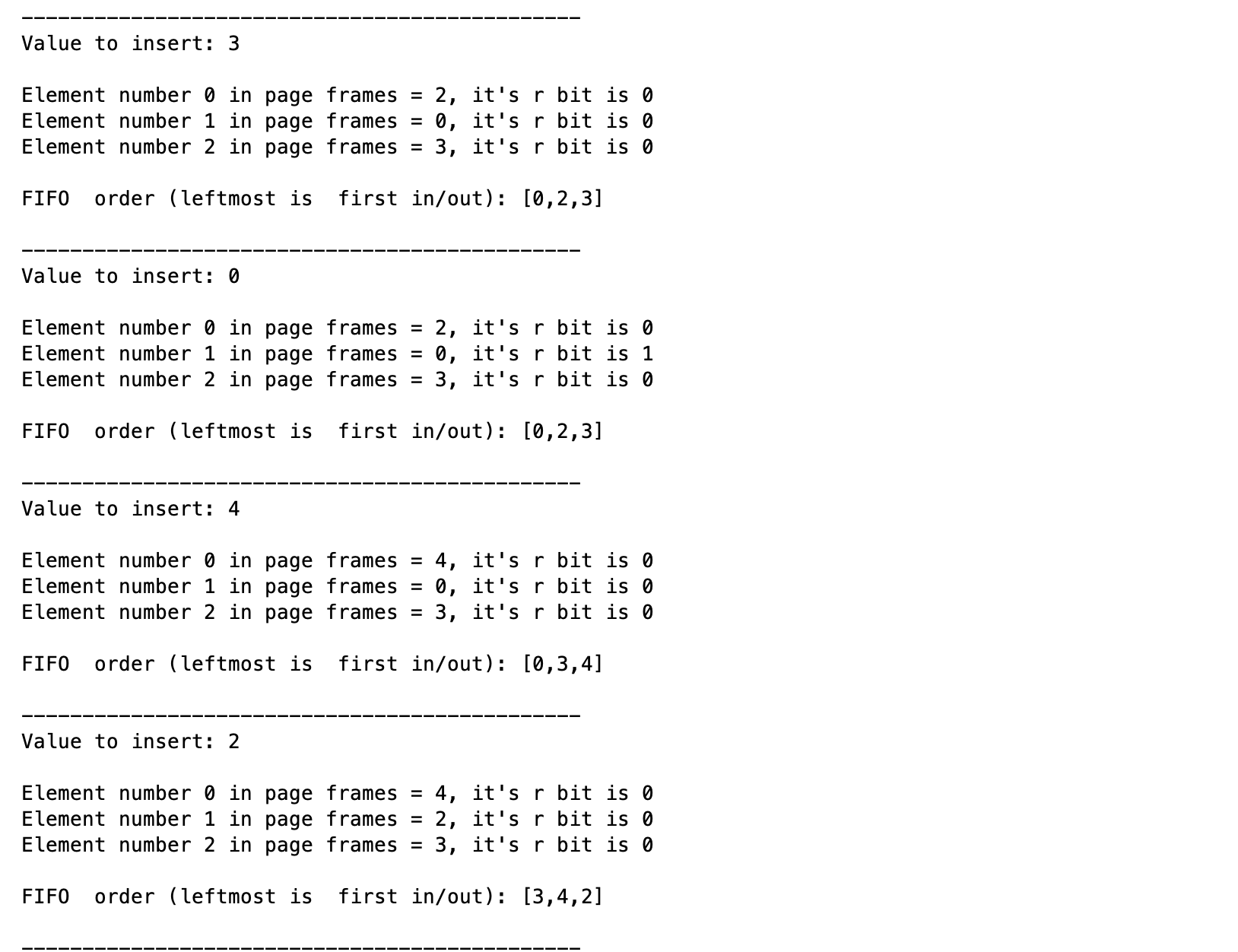
Here is an example:

****

0 is already in the frames, so we just set it’s reference bit to 1, notice that the FIFO order is unchanged.

Value 2 needs to be added, since all of the page frames have corresponding reference bit of 0, we normally replace the next FIFO element in the queue, which is 7

Values 7.0,1 added sequentially like the previous examples



Here, 2 is to be inserted, and all the page frames have corresponding reference bit of 0, so we apply normal FIFO here, and 0 is the first-in element so it is finally replaced.

4 is the next value to be inserted, according to FIFO, the first-in value is 0, but its reference bit is 1, so we set it to 0 and move on to the next value, which is 2, and has a reference bit of 0 which makes is a suitable replacement candidate

0 is the next reference value, and it is already in the page frames, so we set it’s reference bit to 1 and move on

3 needs to be added, but it is not in the page frame, so we need to replace an existing value, the FIFO order indicates that 0 is the first-in element, but we can’t replace it because it has a reference bit of 1, so we set it’s reference bit to 0 and we move on to the next value, 1, which has a reference bit of 0, which makes it the best candidate for replacement.

Again, this keeps going until the reference string is over, and like the previous examples, misses are printed in the end, and the misses count when a new value is being inserted into the page frames.

**3.6 Enhanced second chance algorithm**

This algorithm is a little bit sophisticated, instead of keeping track of a reference bit for each page frame, we also keep track of a modify bit, here’s how it works:

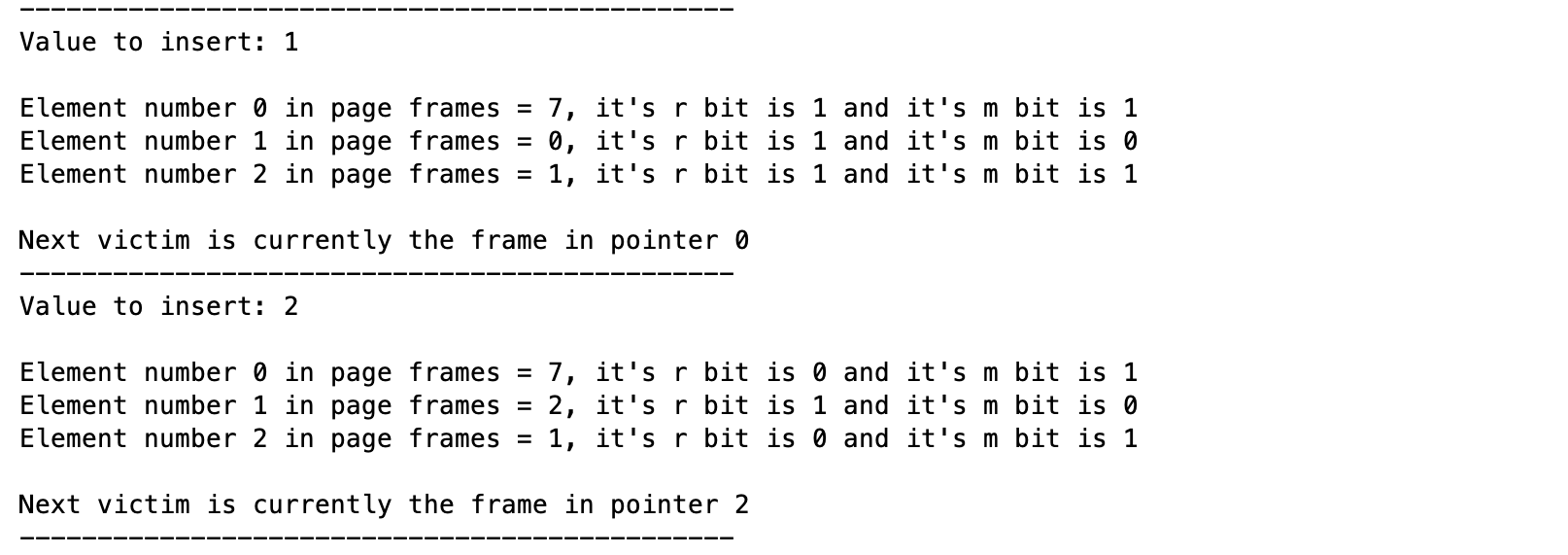
1. Look to see if we find a value in reference frame, if we found it, move on to the next value
2. If not found, stating looping with the next victim pointer to look for and empty frame, if found, insert value into it
3. If no empty frames found, use next victim pointer to loop through the frames, looking for a frame with reference bit of 0 and modify bit of 0, while looping, we do not change anything. We keep going until we find a frame with 0,0 values, in this case we replace the value in this frame, and point the next victim pointer to the next frame (if replaced value in the last frame, next victim pointer will point to first frame)
4. If no 0,0 values where found, we start looping again, looking for a frame with reference bit of 0 and modify bit of 1, just like looping to find 0,0 but instead, we change the reference bit of the frames across the loop to 0, if we found a frame with 0,1 r and m bits, we replace it and point the next victim pointer to the next frame.
5. Repeat step 3 and 4 again, this will guarantee finding a candidate frame.

Keep in mind, that values newly inserted have a reference bit of 1 and a modify bit of 1 or 0 (selected randomly)

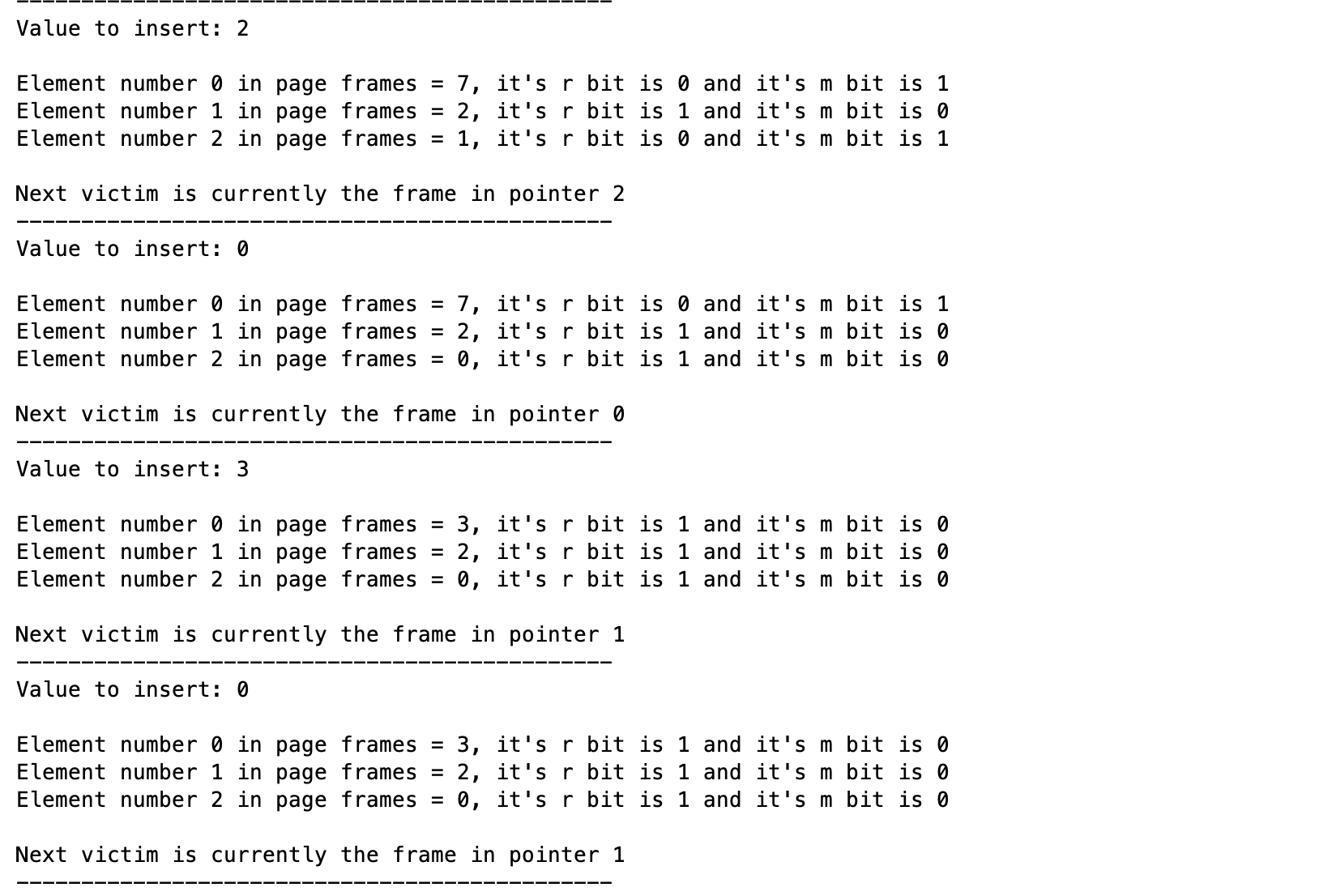
Also, each loop cycle (4 or 5) starts with the current next victim frame and ends with it too (circular loop)

Here is an example:

Values 7,0,1 inserted sequentially like the previous example, notice that all the values newly inserted have r bit of 1 and m bit of 0 or 1



2 is to be inserted and it is not in the frames, so we loop to look for 0,0, we don’t find 0,0, we loop for 0,1 and on the way change r bit of all frames to 0, no 0,1, found, look for 0,0 and we find it in second frame, we replace it and set next victim pointer to the next frame (third)



0 needs to be added but it is found in page frame, so nothing is done

3 needs to be inserted, but it is not found in page frames, so we look for 0,0, we do not find it, we look for 0,1 and the next victim pointer finds it in the next value, we replace it and increment next victim pointer without changing anything else

0 needs to be inserted and it is not found in page frames, so we look for 0,0 and nothing is found, we look for 0,1 and charge r bit of frames on the loop to 0, but we find 0,1 on next victim pointer frame, so we replace frame value and nothing else is changed, next victim pointer will point to next frame (0)

2 needs to be inserted and it is found in page frames, so nothing is done

Again, this keeps going until the reference string is over, and misses are printed like the previous examples, misses, like the previous examples, are counted when inserting new values only.