Exercise 3.3 Show that LRU does not incur Belady’s anomaly but that FIFO does incur the anomaly

**Belady’s Anomaly**: Some reference strings generate more page faults when more page frames are allotted.

1) **FIFO** (First-In/First-Out): Replace the page that has been in the fast memory longest.

Intuition: FIFO algorithm replaces a frequently used variable which causes the extra work of reading it in the page frames again (page fault) since this variable is probably the one which was just replaced.

**Proof**: Considering the reference string

\[1, 2, 3, \ldots, p, p+1, 1, 2, 3, \ldots, p-1, p+2, 1, 2, 3, \ldots, p, p+1, p+2\]

Segment 1  Segment 2  Segment 3  Segment 4

a. Calculating page faults for cache size \(p+1\):

Segment 1: \(p+1\) page faults (initially empty)
Segment 2: 0 page fault (all hits)
Segment 3: 1 page fault (replace 1 with \(p+2\) when applying FIFO)
Result: 2, 3, \ldots, \(p+1\), \(p+2\)
Segment 4:
Cache size \(p+1\)

\[\begin{align*}
\text{End} & \quad \text{Beginning} \\
3, 4, \ldots, \ldots, p, p+1, p+2, 1 \text{ (after reading 1) 1 page fault} \\
4, 5, \ldots, \ldots, p+1, p+2, 1, 2 \text{ (after reading 2) 1 page fault} \\
\ldots & \quad \ldots \\
p, p+1, p+2, 1, 2, \ldots, p-3, p-2 \text{ (after reading } p-2) \text{ 1 page fault} \\
p+1, p+2, 1, 2, 3, \ldots, p-2, p-1 \text{ (after reading } p-1) \text{ 1 page fault} \\
p+2, 1, 2, 3, 4, \ldots, p-1, p \text{ (after reading } p) \text{ 1 page fault} \\
1, 2, 3, 4, \ldots, p-1, p, p+1 \text{ (after reading } p+1) \text{ 1 page fault} \\
2, 3, 4, \ldots, p-1, p, p+1, p+2 \text{ (after reading } p+2) \text{ 1 page fault}
\end{align*}\]
Total # of page faults for cache size p+1 using FIFO
= p+1+p+2=2p+4

b. Calculating page faults for cache size p:

Segment 1: p+1 page faults
Result: 2, 3, …, p, p+1

Segment 2:
Cache size p

End Beginning
3, 4, ………..p-1, p, p+1, 1 (after reading 1) 1 page fault
4, 5, ………..p, p+1, 1, 2 (after reading 2) 1 page fault
………………………………
………………………………
………………………………
p, p+1, 1, 2, …, p-3, p-2 (after reading p-2) 1 page fault
p+1, 1, 2, 3, …, p-2, p-1 (after reading p-1) 1 page fault

Segment 3: 1 page fault (replace p+1 with p+2 when applying FIFO)
Result: 1, 2, 3, …, p-1, p+2

Segment 4:
2 page faults
p-1 hits for first p-1 inputs
after reading p: 2, 3, …, p-1, p+2, p (1 page fault)
after reading p+1: 3, …, p-1, p+2, p, p+1 (1 page fault)
after reading p+2: 2, 3, …, p-1, p, p+1, p+2 (hit)

Total # of page faults for cache size p using FIFO
= p+1+p-1+1+2=2p+3

Therefore, Total # of page faults for cache size p using FIFO
< Total # of page faults for cache size p+1 using FIFO
since 2p+3 < 2p+4. This incurs Belady’s Anomaly
when p is at least 3.

2) LRU (Least-Recently-Used): When eviction is necessary, replace the
page whose most recent request was earliest.
Intuition: Due to the method of this algorithm, which is that it will replace the page (variable) whose most recent request was earliest, this algorithm significantly avoids the case of replacing a frequently used variable if this coming variable appears to be closer to the current reading variable.

Proof:
Given any reference string S=a1, a2, …, an. Let LRUi(S) be the number of faults that LRU incurs on S with a cache of size i, we need to show for all i and S, and i < j,
\[ \text{LRUi}(S) \geq \text{LRUi+1}(S) \geq \text{LRUi+2}(S) \geq \ldots \geq \text{LRUj}(S) \]

Defining that a doubly-linked list of size i can be embedded in another doubly-linked list of size i+1, if the two doubly-linked lists are identical, except that the longer one has one more item, which is the last one.

Claim: After each step of processing a sequence of requests, the doubly-linked list of LRUi can be embedded in the doubly-linked list of LRUi+1.

We prove this claim by induction on the number of steps.
1) Basic case: if n=1, both LRUi and LRUi+1 incur a fault and bring in a1.
2) Induction Hypothesis: the claim is true after step n.
3) To show it is also true after step n+1.
   a. Suppose before reading a_{n+1}, a_{n+1} is in the cache of LRUi (hit).
      According to IH, a_{n+1} is also in the cache of LRUi+1 (hit).
      Both moving a_{n+1} to the beginning of their lists after reading a_{n+1}.
      So the claim is also true after step a_{n+1}.
   b. Suppose before reading a_{n+1}, a_{n+1} is NOT in the cache of LRUi (fault).
      i) a_{n+1} is also not in the cache of LRUi+1 (fault): both moving a_{n+1} to the beginning of their lists after reading a_{n+1}. Claim holds.
      ii) a_{n+1} is in the cache of LRUi+1 (a_{n+1} must be the last page based on IH): LRUi+1 moves its a_{n+1} to the beginning of their lists after reading a_{n+1}. LRUi brings a_{n+1} and replaces one of the old elements. By IH, all remaining items are same or evict page which is always at the end of list.

Example:

<table>
<thead>
<tr>
<th></th>
<th>Before reading a_{n+1} (7)</th>
<th>After reading a_{n+1} (7)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU4</td>
<td>End 3→4→5→6</td>
<td>Beginning 4→5→6→7</td>
</tr>
<tr>
<td>LRU_{i+1}</td>
<td>End 7→3→4→5→6</td>
<td>Beginning 3→4→5→6→7</td>
</tr>
</tbody>
</table>
The claim is proved. So at each step, if $\text{LRU}_{i+1}$ has a fault, then $\text{LRU}_i$ has a fault since $\text{LRU}_{i+1}$ list elements contain (embed) $\text{LRU}_i$ list elements. So $\text{LRU}_i(S) \geq \text{LRU}_{i+1}(S)$. Finish proof for LRU.

Reference: http://www.cas.mcmaster.ca/~soltys/cs4sh3-w02/index.html