Assignment 1, Question 2

Recall the LRU page-replacement algorithm, presented in Section 10.4.4 in the textbook. Show that LRU does *not* incur Belady's Anomaly (on any cache size and any request sequence).

Let σ be any request sequence, $\sigma = p_1, p_2, \ldots, p_n$, and let

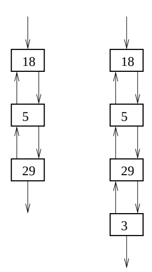
$$\text{LRU}_i(\sigma)$$

be the number of faults that LRU incurs on σ with a cache of size i. We are going to show that for all i and σ, the following property holds:

$$\text{LRU}_i(\sigma) \geq \text{LRU}_{i+1}(\sigma) \qquad (1)$$

Once you have (1), it is easy to show that for any $i < j$ and σ, $\text{LRU}_i(\sigma) \geq \text{LRU}_j(\sigma)$ (you do this!). Conclude that LRU does not incur Belady's Anomaly.

The work is showing (1). To show it, we define a property of doubly-linked lists which we call embedding. We say 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 may have one more item at the "end." For example:



Here, the doubly-linked list of size 3 on the left can be embedded in the doubly-linked list of size 4 on the right.

At the beginning of processing the request sequence, when the caches are getting filled up, the two lists are identical, but once the caches are full, the LRU_{i+1} cache will have one more item in general.

Now prove the following Claim:

Claim: After each step of processing a sequence of requests, the doubly-linked list of LRU_i can be embedded in the doubly-linked list of LRU_{i+1}.

We prove this claim by induction on the number of steps. Basis Case: if $n = 1$, then both LRU_i and LRU_{i+1} incur a fault and bring in p_1.

Induction Step: suppose that the claim holds after step n; we show that it also holds after step $n+1$. Consider the following cases:

1. LRU_i has a hit on p_{n+1}.
2. LRU_i has a fault on p_{n+1}.
   (a) LRU_{i+1} also has a fault.
   (b) LRU_{i+1} does not have a fault.

Show that in each case the embedding property is being preserved (using the Induction Hypothesis, and some basic reasoning about the LRU algorithm).

Finally, use the claim to prove (1).

Marking Algorithms

Consider a cache of size k. Fix any request sequence σ. We divide the request sequence into **phases** as follows: phase 0 is the empty sequence.

For every $i \geq 1$, phase i is the maximal sequence following phase $i-1$ that contains at most k distinct page requests; that is, if it exists, phase $i+1$ begins on the request that constitutes the $(k+1)$-st distinct page request since the start of the i-th phase.

Such a partition is called a **k-phase partition**. This partition is well defined and is independent of how any particular algorithm processes σ.

Example of a 3-phase partition:

$$\underbrace{1,2,1,2,1,2,3}_{1},\underbrace{4,5,6,6,6,6,6,6,6,4,5,4}_{2},\underbrace{7,7,7,7,1,2}_{3}$$

Let σ be any request sequence and consider its k-phase partition. Associate with each page of the slow memory a bit called the **mark**.

The marking is done for the sake of analysis (this is not implemented by the OS, but by “us” to keep track of the doings of an algorithm).

Each page, when its mark bit is set we say that the page is **marked**, and otherwise, **unmarked**.

Suppose that at the beginning of each k-phase we unmark all the pages, and we mark a page when it is first requested during the k-phase. A **marking algorithm** never evicts a marked page from its fast memory. We can consider FWF as the most elementary marking algorithm.

We are interested in the following two results:

- LRU is a marking algorithm
- Let ALG be any marking algorithm with a cache of size k, and let OPT be the optimal offline algorithm with a cache of size $h \leq k$. Then:

$$\text{ALG is } \left(\frac{k}{k-h+1}\right)\text{-competitive}$$

Example

Suppose that $k = 2$, and σ is a request sequence. We show the 2-phases of σ:

$$\sigma = \underbrace{1,1,3,1}_{\text{2-phase \#1}}, \underbrace{5,1,5,1,5,1}_{\text{2-phase \#2}}, \underbrace{3,4,4,4}_{\text{2-phase \#3}}, \underbrace{2,2,2,2}_{\text{2-phase \#4}}$$

In this table, we show who is marked after step i. Note that after each phase, everyone is unmarked, and we begin marking fresh:

step	1	2	3	4	5
1	x				
2	x				
3	x		x		
4	x		x		
5					x
6	x				x
7	x				x
8	x				x
9	x				x
10	x				x
11			x		
12			x	x	
13			x	x	
14			x	x	
15		x			
16		x			
17		x			
18		x			

Observation 1: Except for the last phase, all phases are always complete (they have exactly k distinct requests, 2 in this case).

Observation 2: With a marking algorithm, once a request for page p in phase i is made, p stays in the cache until the end of phase i (the first time p is requested, it is marked, and it stays marked for the entire phase, and a marking algorithm never evicts a marked page).

Observation 3: Intuitively, marking algorithms seem like good page-replacing algorithms because, in a any given phase, there are at most k distinct pages, so they all fit in a cache of size k, so it does not make sense to evict them, in that phase (we can only loose by evicting, as the evicted page might be requested again).

LRU is a marking algorithm

PROOF: We argue by contradiction: suppose that LRU is not a marking algorithm.

Let σ be a request sequence where there exists a k-phase partition, during which some marked page p_i is evicted. Consider the first request for p_i during this k-phase:

$$\sigma = p_1, \ldots, \ldots, \underbrace{\ldots, p_i, \ldots, \ldots, \ldots}_{k\text{-phase}}, \ldots, \ldots$$

Immediately after p_i is serviced, it is marked as the most recently used page in the cache (i.e., it is at the top of the doubly-linked list).

In order for p_i to leave the cache, LRU must incur a page fault while p_i is the least recently used page. It follows that during the k-phase in question, $k+1$ distinct pages were requested: there are the $k-1$ pages that pushed p_i to the end of the list, there is p_i, and the page that got p_i evicted.

Contradiction; a k-phase has at most k distinct pages.

Exercise: Show that FWF is a marking algorithm (the demand-paging version that marks all the pages when the cache is full, and evicts, when necessary, an arbitrary marked page). Show that FIFO is not a marking algorithm.

Note that FWF is an **explicit** marking algorithm (pages are actually marked by the OS). LRU is an **implicit** marking algorithm (the pages are not given marks, but LRU *is* a marking algorithm).

Note: Lets assume that we always consider the following version of the FWF algorithm: each slot in the cache has a single bit associated with it. At the beginning, all these bits are set to zero. When a page p is requested, FWF checks only the slots with a marked bit. If p is found, it is serviced. If p is not found, then it has to be brought in from the slow memory (even if it actually is in the cache, in an unmarked slot). FWF looks for a slot with a zero bit, and one of the following happens:

- A slot with a zero bit (an unmarked page) is found. Then, FWF replaces that page with p.

- A slot with a zero bit is not found (all pages are marked). Then, FWF unmarks all the slots, and replaces any page with p, and it marks p's bit.