Online Caching Workshop

CS 181 Advanced Algorithms — Spring 2026

Plenty of Practice with Paging Problems!

This workshop is a self-guided (or rather, group-guided) exploration into the myriad algorithms that are used in paging and caching policies in real-world computer systems. The **main goal** of this workshop (aside for more practice with the concepts we've been learning) is to spark a sense of why one might choose one algorithm over another in a given application.

Get together in groups of $\approx$ three-four and go through the questions together. Turn in your sheet to Kartika at the end with your name on it (this will not graded for correctness, it's just a form of attendance/effort). If you finish early, you can use the remaining time work on assignment 3 or your final projects. Good luck!!

---

1. Review the definition of the competitive ratio with your group. Then, remind yourself how we know that the LRU algorithm is k-competitive, where k is the cache-size. What is the key idea in the proof?

2. (a) The LIFO (Last-In-First-Out) paging policy evicts whatever page was last added to the cache. What do you think of this caching policy? Try to come up with a sequence of page requests where this policy performs particularly poorly.

   (b) Explain why the LIFO paging policy is not α-competitive for any constant $\alpha > 1$.

3. (a) In contrast to LIFO, prove that the FIFO (First-In-First-Out) algorithm is k-competitive for the paging problem, where k is the cache size.

   Hint: Break the input sequence into phases as in the LRU proof. How many faults does FIFO incur in each phase?

   (b) The same argument proves that Flush-when-Full is k-competitive. Since FIFO, LRU and Flush-when-full all have the same competitive ratio, what might be some other pros and cons of these caching policies?

4. **Belady's Anomaly**. It turns out that for some algorithms, increasing the cache size can actually lead to an increased number of faults! Run the FIFO policy on the following sequence starting with an empty cache of size of $k = 4$, and then when $k = 3$. How many faults are incurred?

$$\sigma = 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5$$

   What do you think is the intuitive reason that this can happen in the FIFO policy?

5. Show that LRU (Least-Recently-Used) does not suffer from Belady's anomaly. That is, for LRU, the number of faults is nonincreasing in the cache size.

   Hint: A key property of LRU is that at any time, the cache of size k contains exactly the most recently used k distinct pages. Use this to show that the cache contents $C_k(t)$ at time t with capacity k satisfies the relation $C_k(t) \subseteq C_{k+1}(t)$.

6. **Resource augmentation**. [Challenge] Consider an online paging setting in which an LRU algorithm has a cache of size k, while the optimal offline algorithm OPT is restricted to a cache of size $h \leq k$. Show that the competitive ratio of LRU in this setting is at most $\frac{k}{k-h+1}$.

   Hint: Break the input sequence into phases. How many faults does LRU incur per phase? How much does OPT with a cache of size h incur on each offset phase?

7. What are you and other members of your group looking forward to this summer?