Homework 2

Adam Karl

September 22, 2020

1 Problem 6

1.1 Algorithm Least Recently Used (LRU)

LRU

- If k=2 (pages in fast memory) and n=4 (pages in slow memory), let the pages be A, B, C, and D. Note that the fast memory starts empty.
- Access sequence: A B C A
- A: page fault; A read into fast memory
- B: page fault; A read into fast memory
- C: page fault; A evicted, C read into fast memory
- A: page fault; B evicted, A read into fast memory
- total: 4 faults

OPT

- A: page fault; A read into fast memory
- B: page fault; A read into fast memory
- C: page fault; B evicted, C read into fast memory
- A: page hit
- total: 3 faults

Since OPT is a valid solution (only 2 pages in memory, pages must be read into fast memory upon access if they aren't already there) and has fewer page faults than LRU on input ABCA, LRU must not be the optimal solution for the page replacement problem for all inputs. (Proof by counterexample)

1.2 Algorithm Furthest in the Future (FF)

Approach: proof by contradiction

- Assume that FF is not an optimal algorithm
- therefore an input I exists such that FF(I) is not optimal
- let OPT(I) be an optimal solution for input I that has the same pages in fast memory as FF(I) for the most number of steps
- let u be the first point that FF(I) and OPT(I) have different pages in fast memory

Case 1: the memory access at u does not cause a page fault.

- In this case, since there is no page fault, FF(I) will not swap out any pages from memory. Therefore, since by definition FF(I) and OPT(I) disagree at point u, OPT(I) must swap out one or more pages from memory. Construct OPT'(I) such that OPT'(I) is the same as OPT(I) except with any page replacements at u instead happening at u+1. OPT'(I) is still feasible since it has the same number of page replacements as OPT(I) and because all pages will be in fast memory when accessed (since there was no page fault at u, it is inconsequential to push back these page replacements by 1 access). Therefore, OPT'(I) is an optimal solution that agrees with FF(I) for 1 more step than OPT'(I).

Case 2: the memory access at u does cause a page fault

- In this case, neither OPT(I) nor FF(I) have the page needed in fast memory (since they agree up to point u), so they must evict different pages in order to load that page into fast memory. Let the page FF(I) evicts be y and the page OPT(I) evicts be x. Note that by the description of FF(I), we know that the next time x is accessed comes before the next time y is accessed. Let the next time y is accessed be time v. There are 2 scenarios:
  - 2a) OPT(I) keeps y in fast memory until time v. Since x is accessed at time v (and OPT(I) is feasible), we know that at time v OPT(I) has both x and y in memory. We also know that in order to have x in fast memory for the access at time v, OPT(I) must evict some page before or at time v to load x. Construct OPT'(I) so that it is the same as OPT(I), except it keeps x instead of y at time u (agreeing with FF(I)), and at the point that OPT(I) swaps some page out for x after time u and before/at time v, instead swap y back into memory. This way, we have not added any swaps, just exchanged when x and y were swapped in/out between u and v. Constructing OPT'(I) this way, we know that it is feasible since neither x nor y are accessed between times u and v, and OPT'(I) is identical to OPT(I) before time u and after time v. In addition, no page swaps have been added, so it is no worse of a solution than OPT(I). OPT'(I) is no worse than OPT(I) and agrees with FF(I) for one more step than OPT(I) does.
  - 2b) OPT(I) evicts y sometime between time u and time v. In this case we construct OPT'(I) by making it the same as OPT(I) but instead pushing y's eviction up to time u. Since we are now evicting y at time u, we are free to keep x in memory (so time u is now identical to FF(I)'s time u), and since we know that y is not accessed again until time v, we can be sure that we are never missing an instance that y is accessed but not in memory since OPT already must re-load y at or before time v. This version of OPT'(I) has 1 fewer page replacement than OPT, and has every page in fast memory when accessed, so therefore it is correct and no worse than optimal. In addition, OPT'(I) agrees with FF(I) for at least 1 more step than OPT(I).

2 Conclusion

We started by defining OPT(I) as the optimal solution that agreed with FF(I) for the most number of steps, and our only assumption was that FF was not an optimal algorithm. However, for all logically possible cases (1, 2a, and 2b) we were able to construct solutions OPT'(I) such that they were all feasible and agreed with FF(I) for at least 1 more step than OPT(I), leading to the contradiction that OPT(I) both is and is not the optimal solution that agrees with FF(I) for the most number of steps. Therefore, due to this contradiction our assumption must be wrong. FF is the optimal algorithm for solving page replacement with the fewest number of page replacements.