CPSC 420 Lecture 23: Today's announcements:

- Examlet 3 on Mar 17 in class. Closed book & no notes
- Reading: Randomized Algorithms [by Motwani and Raghavan]

Today's Plan

- Online Algorithms
  - Hiring problem ✓
  - Page replacement
  - List Update
  - Experts

Online Algorithms

For input sequence $p_1, p_2, \ldots, p_n$ an **online algorithm** must produce an output given $p_1, p_2, \ldots, p_i$ for each i **without seeing** $p_{i+1}, \ldots, p_n$.

Page replacement in a cache

$p_1, \ldots, p_n$ is a sequence of **page requests** made by a program.

k is cache size (in pages).

At the ith page request, p_i, the cache contains some k pages. If p_i is **not** in cache (**page fault**) some page must be evicted from cache to make room for p_i, then p_i is added to the cache.

The cost of a page replacement algorithm A on sequence $p_1, p_2, \ldots, p_n$ is

$$f_A(p_1, p_2, \ldots, p_n) = \# \text{ faults } A \text{ has on } p_1, p_2, \ldots, p_n$$

Online algorithm must decide what page to evict without knowing future requests.

Some paging algorithms

- **LRU** (Least Recently Used) Evict page whose most recent request occurred furthest in the past.
- **FIFO** (First In First Out) Evict page that has been in cache the longest.
- **LFU** (Least Frequently Used) Evict page that has been requested least often.
- **OPT** (Optimal) Evict best* page knowing the future requests.

Some paging algorithms

LRU (Least Recently Used) Evict page whose most recent request occurred furthest in the past.

FIFO (First In First Out) Evict page that has been in cache the longest.

LFU (Least Frequently Used) Evict page that has been requested least often.

OPT (Optimal) Evict best* page knowing the future requests.

* **next** request is furthest in the future

Paging algorithms: Example

page requests — time

	A	B	C	D	B	A	B	C	D	A	B	requests
LRU	A	B	C	D	D	D	D	C	C	C	B	
	Y	A	B	C	C	A	A	A	D	D	D	
	Z	Y	A	B	B	B	B	B	B	A	A	
	*	*	*	*		*		*	*	*	*	9 faults ✓
FIFO	A	B	C	D	D	A	B	C	D	A	B	
	Y	A	B	C	C	D	A	B	C	D	A	
	Z	Y	A	B	B	C	D	A	B	C	D	
	*	*	*	*		*	*	*	*	*	*	10 faults ✓
LFU	A	B	C	D	D	D	D	C	C	A	A	
	Y	A	B	C	C	A	A	A	D	D	D	
	Z	Y	A	B	B	B	B	B	B	B	B	
	*	*	*	*		*		*	*	*		8 faults ✓
OPT	A	B	C	D	D	D	D	D	D	D	B	
	Y	A	B	B	B	B	B	C	C	C	C	
	Z	Y	A	A	A	A	A	A	A	A	A	
	*	*	*	*				*			*	6 faults

Evaluating online algorithms

Which algorithm is best?

Worst case cost

$$n = \max_{p_1..p_n} f_{\mathrm{LRU}}(p_1..p_n) = \max_{p_1..p_n} f_{\mathrm{FIFO}}(p_1..p_n) = \max_{p_1..p_n} f_{\mathrm{LFU}}(p_1..p_n)$$

Every online algorithm can be made to fault on every request.

Average case cost

$$E[f_X(p_1..p_n)] = n(1 - k/m)$$

where $m = \#$ possible pages

Competitive analysis

How does online alg compare to best offline alg?

An online algorithm A is c-**competitive** if for all $p_1, p_2, \ldots, p_n$,

$$f_A(p_1..p_n) \leq c \cdot f_{\mathrm{OPT}}(p_1..p_n) + b$$

for some constant b.

Lower Bound on Competitive Factor

Claim: If A is a deterministic online alg for paging then $c \geq k$.
Proof: Idea: Find a sequence that is bad for A but good for OPT.
Suppose both A and OPT start with pages $1, 2, \ldots, k$ in cache.

k+1 one additional page

Request page $a_1 = k + 1$

A evicts some page, call it a_2

Request page a_2

A evicts some page, call it a_3

Request page a_3

etc...

$\Rightarrow$ Online alg A faults on every request.

How many different pages are there? k+1

Lower Bound on Competitive Factor

Claim: If A is a deterministic online alg for paging then $c \geq k$.
Proof: Idea: Find a sequence that is bad for A but good for OPT.
Suppose both A and OPT start with pages $1, 2, \ldots, k$ in cache.

Request page $a_1 = k + 1$
    A evicts some page, call it a_2
Request page a_2
    A evicts some page, call it a_3
Request page a_3
    etc...

$\Rightarrow$ Online alg A faults on every request.

How many different pages are there? $k + 1$

Lower Bound on Competitive Factor

How often does OPT fault?

	a_1 a_2 a_3 ... a_j a_{j+1} ...	requests
OPT	* $\qquad\qquad\qquad\qquad$ *	faults

Sub-claim: OPT faults at most once in k successive page requests from this sequence

Proof.
OPT evicts the page p in cache that is requested furthest in the future. Since there are only $k+1$ different pages, the next k pages requested can be kept in cache. □

$\Rightarrow f_A(a_1, a_2, \ldots, a_n) \geq k \cdot f_{\text{OPT}}(a_1, a_2, \ldots, a_n)$ □

LRU is k-competitive

Theorem

LRU is k-competitive

Proof.

Let $p_1, p_2, \ldots, p_n$ be **any** sequence of page requests. Partition this sequence into contiguous subsequences (**phases**) such that LRU faults on the first page of the phase and the phase contains exactly k different pages (or $\leq k$ in the last phase).

OPT faults ≥ 1

$k=3$	A	C	D	C	C	B	C	A	D	A	A	requests
LRU	*	~~?~~ *	~~?~~ *			*		*	*			faults
OPT	?	?	?			*			*			faults
	phase 1					ph 2			ph 3			

$$\frac{k \ \#phases}{\#phases - 1}$$

LRU faults $\leq k$ times per phase.

OPT must have the first page of a phase in cache at the beginning of a phase. Since the remainder of the phase plus the first page of the next phase consists of k different pages, OPT must fault at least once during these requests.$\Rightarrow$ OPT faults $\geq$ #phases $- 1$

Any Marking Algorithm is k-competitive

Marking Algorithm MARK

```
0. Start with all k pages in cache unmarked
1. On page request p
2.      if p not in cache then
3.           evict any unmarked page
             (if no unmarked page, first unmark all k pages)
5.           bring p into cache
6.      mark p
```

	A	B	C	D	B	A	B	C	D	A	B
MARK	•A	•A	•A	•D	•D	•D	•D	•C	•C	•C	•B
	Y	•B	•B	B	•B	•B	•B	~~B~~	•D	•D	D
	Z	Z	•C	C	C	•A	•A	A	A	•A	A
	*	*	*	*		*		*	*		*
LRU	A	B	C	D	D	D	D	C	C	C	B
	Y	A	B	C	C	A	A	~~A~~	D	D	D
	Z	Y	A	B	B	B	B	B	B	A	A
	*	*	*	*		*		*	*	*	*

Any Marking Algorithm is k-competitive

Marking Algorithm MARK

0. Start with all k pages in cache unmarked
1. On page request p
2.   if p not in cache then
3.   evict any unmarked page
           (if no unmarked page, first unmark all k pages)
5.   bring p into cache
6.   mark p

Proof.

Partition $p_1, p_2, \ldots, p_n$ into **phases**, a maximal subsequence with k distinct pages. (The first starts with p_1.) Assume p_1 is not in cache. MARK faults $\leq k$ times per phase.

OPT must have the first page p_i of a phase in cache at the beginning of a phase. Since the remainder of the phase plus the first page of the next phase consists of k different pages (different from p_i), OPT must fault at least once during these requests.

$\Rightarrow$ OPT faults $\geq$ #phases $- 1$ times. $\square$

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move

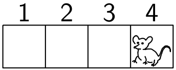
1	2	3	4

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move



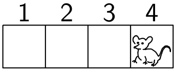
OPT(1 2 3 4 1 2 3 4) = 2

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move



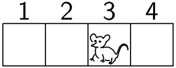
OPT(1 2 3 4 1 2 3 4) = 2

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move



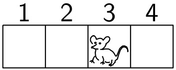
OPT(1 2 3 4 1 2 3 4) = 2

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move



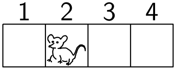
$\text{OPT}(1\ 2\ 3\ 4\ 1\ 2\ 3\ 4) = 2$

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move



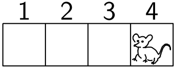
OPT(1 2 3 4 1 2 3 4) = 2

Randomized online algorithm

Online Hide and Seek

Mouse hides in one of m hiding places.
Cat looks in one hiding place each time step.
If Cat finds Mouse, Mouse runs to another place.

Cost = #times Mouse moves
OPT = min #times future-knowing Mouse must move



$$\mathrm{OPT}(1\ 2\ 3\ 4\ 1\ 2\ 3\ 4) = 2$$

$$\mathrm{OPT}(1\ 2\ 1\ 1\ 3\ 4\ 1\ 2) = 1$$