Online Algorithms

Unit 3.1

Introduction

- An online algorithm is one that can process its input piece-by-piece in a serial fashion, i.e., in the order that the input is fed to the algorithm, without having the entire input available from the start.
- An offline algorithm is given the whole problem data from the beginning and is required to output an answer which solves the problem at hand. (For example, selection sort requires that the entire list be given before it can sort it, while insertion sort doesn't.)

Competitive Analysis

- Because it does not know the whole input, an online algorithm is forced to make decisions that may later turn out not to be optimal, and the study of online algorithms has focused on the quality of decision-making that is possible in this setting.

- Competitive analysis formalizes this idea by comparing the relative performance of an online and offline algorithm for the same problem instance.

Competitive Analysis

- An on-line algorithm A is c-competitive if there is a constant b for all sequences s of operations
- $A(s) < c\ OPT(s) + b$
- where A(s) is the cost of A on the sequence s and OPT(s) is the optimal off-line cost for the same sequence.
- Competitive ratio is a worst case bound.

Ski Rental Problem

- Assume that you are taking ski lessons. After each lesson you decide (depending on how much you enjoy it, and what is your bones status) whether to continue to ski or to stop totally.
- You have the choice of either renting skis for 1$ a time or buying skis for y$.
- Will you buy or rent?

Ski rental Problem

- If you knew in advance how many times you would ski in your life then the choice of whether to rent or buy is simple. If you will ski more than y times then buy before you start, otherwise always rent.
- The cost of this algorithm is min(t, y).
- This type of strategy, with perfect knowledge of the future, is known as an offline strategy.

Ski Rental Problem(Cont..)

- In practice, you don't know how many times you will ski. What should you do?
- An online strategy will be a number k such that after renting k-1 times you will buy skis (just before your kth visit).
- Claim: Setting k = y guarantees that you never pay more than twice the cost of the offline strategy.
- Example: Assume y=7$ Thus, after 6 rents, you buy. Your total payment: 6+7=13$.

Ski Rental Problem (Online strategy)

- Theorem:
  - Setting k = y guarantees that you never pay more than twice the cost of the offline strategy.
- Proof: when you buy skis in your kth visit, even if you quit right after this time, t >= y.
  - Your total payment is k-1+y =2y-1.
  - The offline cost is min(t, y) = y
  - The ratio is (2y-1)/y = 2-(1/y).
- **We say that this strategy is [2-(1/y)]-competitive.**

Conclusion or General rule

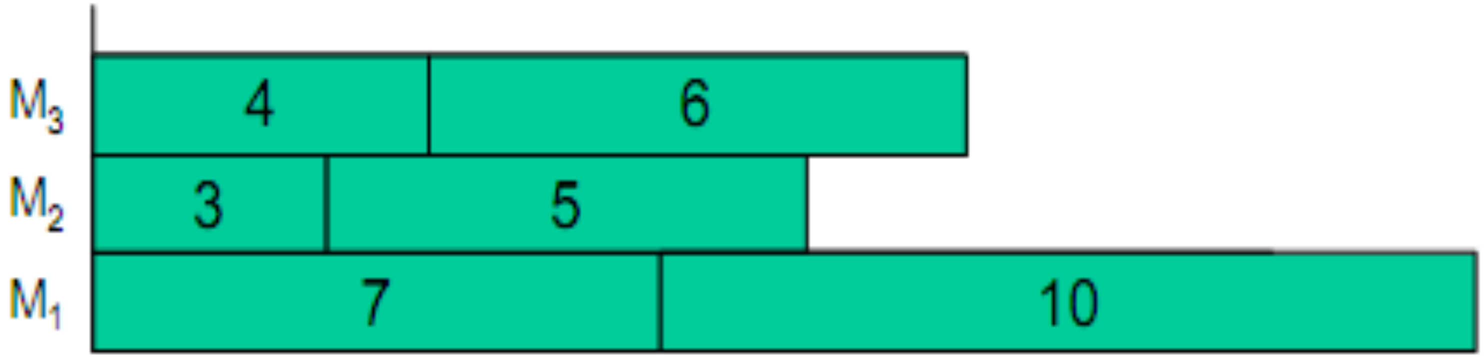
- When balancing small incremental costs against a big one-time cost, you want to delay spending the big cost until you have accumulated roughly the same amount in small costs.

Online Scheduling and Load Balancing

- Problem Statement:
  - A set of m identical machines,
  - A sequence of jobs with processing times p,….
  - Each job must be assigned to one of the machines.
  - When job j is scheduled, we don't know how many 1…,p additional jobs we are going to have and what are their processing times.
- Goal: schedule the jobs on machines in a way that
  - minimizes the makespan

List Scheduling

- A greedy algorithm: always schedule a job on the least loaded machine.
- Example: m=3 σ= 7 3 4 5 6 10



Makespan = 17

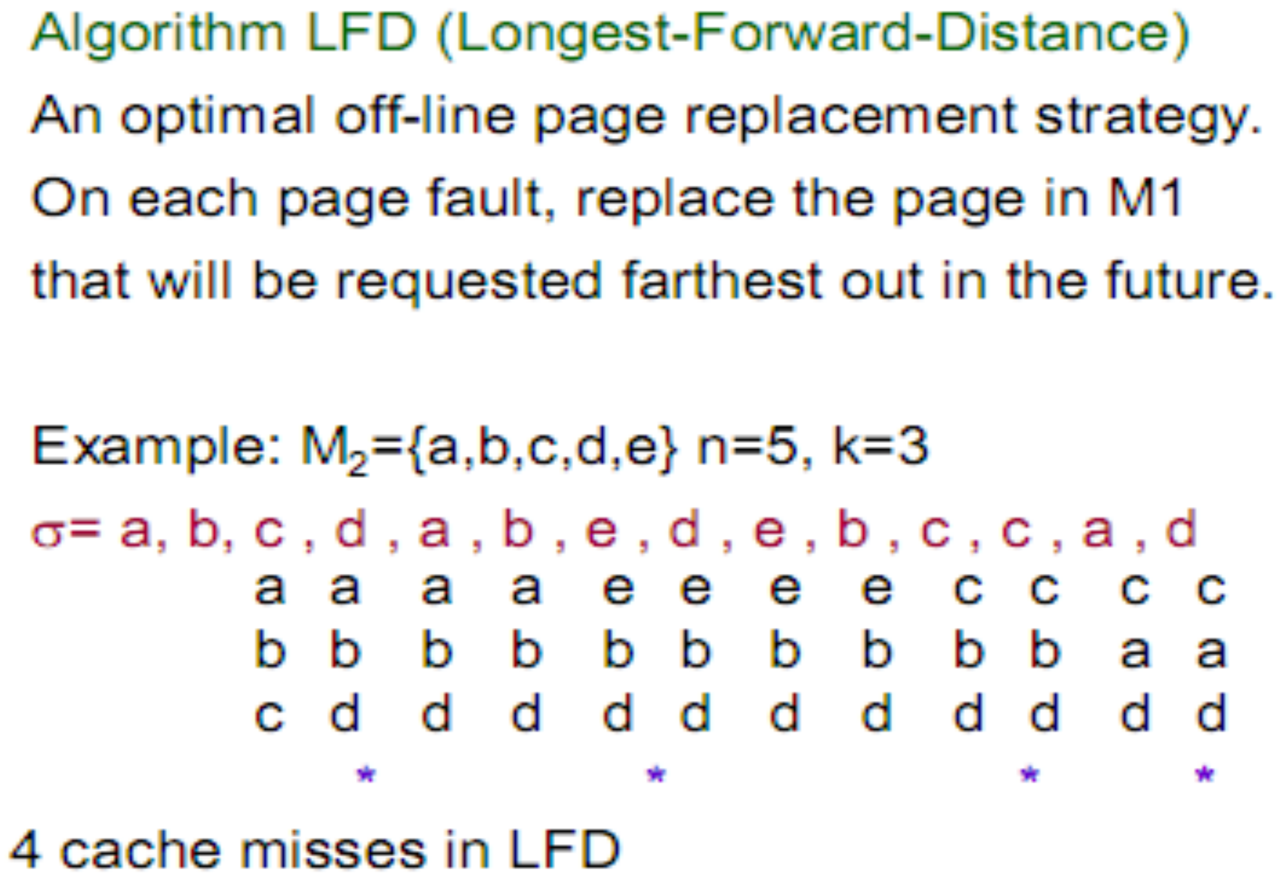
Paging- Cache Replacement Policies

- Problem Statement:
  - There are two levels of memory:
    - fast memory M1 consisting of k pages (cache)
    - slow memory M2 consisting of n pages ($k < n$).
  - Pages in M1 are a strict subset of the pages in M2
  - Pages are accessible only through M1
  - Accessing a page contained in M1 has cost 0.
  - When accessing a page not in M1 it must first be brought in from M2 at a cost of 1 before it can be accessed.
  - This event is called a page fault.

Paging- Cache Replacement Policies

- Problem Statement (cont.):
  - If M1 is full when a page fault occurs, some page in M1 must be evicted in order to make room in M1.
  - How to choose a page to evict each time a page fault occurs in a way that minimizes the total number of page faults over time?

Paging- An Optimal Offline Algorithm



Algorithm LFD (Longest-Forward-Distance)

An optimal off-line page replacement strategy.

On each page fault, replace the page in M1

that will be requested farthest out in the future.

Example: $M_2=\{a,b,c,d,e\}$ $n=5$, $k=3$

σ= a,	b,	c ,	d ,	a ,	b ,	e ,	d ,	e ,	b ,	c ,	c ,	a ,	d
		a	a	a	a	e	e	e	e	c	c	c	c
		b	b	b	b	b	b	b	b	b	b	a	a
		c	d	d	d	d	d	d	d	d	d	d	d
			*			*					*		*

4 cache misses in LFD

Paging- An Optimal Offline Algorithm

A classic result from 1966:

LFD is an optimal page replacement policy.

Proof idea: For any other algorithm A, the cost of A is not increased if in the 1st time that A differs from LFD we evict in A the page that is requested farthest in the future.

However, LFD is not practical.

It is not an *online* algorithm!

Online Paging Algorithms

FIFO: first in first out: evict the page that was entered first to the cache.

Example: $M_2=\{a,b,c,d,e\}$ $n=5$, $k=3$

$\sigma=$ a, b, c , d , a , b , e , d , e , b , c , c , a , d

a	d	d	d	e	e	e	e	e	e	a	a
b	b	a	a	a	d	d	d	d	d	d	d
c	c	c	b	b	b	b	b	c	c	c	c
	*	*	*	*	*			*		*	

7 cache misses in FIFO

Theorem: FIFO is k-competitive: for any sequence, #misses(FIFO) $\leq$ k #misses (LFD)

Online Paging Algorithms

LIFO: last in first out: evict the page that was entered last to the cache.

Example: M_2={a,b,c,d,e} n=5, k=3

σ= a, b, c , d , a , b , e , d , e , b , c , c , a , d

a	a	a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	b	b	b	b	b	b	b
c	d	d	d	e	d	e	e	c	c	c	d
	*			*	*	*		*			*

6 cache misses in LIFO

Theorem: For all n>k, LIFO is not competitive: For any c, there exists a sequence of requests such that #misses(FIFO) ≥ c #misses (LFD)

Online Paging Algorithms

LRU: least recently used: evict the page with the earliest last reference.

Example: $M_2=\{a,b,c,d,e\}$ $n=5$, $k=3$

σ= a, b, c , d , a , b , d , e , d , e , b , c

a	d	d	d	d	d	d	d	d	c
b	b	a	a	a	e	e	e	e	e
c	c	c	b	b	b	b	b	b	b
	*	*	*		*				*

Theorem: LRU is k-competitive