COT 6405: Analysis of Algorithms

Giri NARASIMHAN

www.cs.fiu.edu/~giri/teach/6405F19.html

Amortized Analysis

Problem 1: Binary Counter

- Data Structure: binary counter b.
- Operations: Inc(b).
  - Cost of Inc(b) = number of bits flipped in the operation.
- What's the total cost of N operations when this counter counts up to integer N?
- *Approach 1: simple analysis*
  - Size of counter is log(N). Worst case when every bit flipped. For N operations, total worst-case cost = O(Nlog(N))

Amortized Analysis: Potential Method

- For n operations, the data structure goes through states: D_0, D_1, D_2, …, D_n with costs c_1, c_2, …, c_n
- Define potential function $\Phi(D_i)$: represents the potential energy of data structure after i_{th} operation.
- The amortized cost of the i_{th} operation is defined by:

$$\boxed{\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})}$$

- The total amortized cost is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{N} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^{n} c_i$$

$$\sum_{i=1}^{n} c_i = -\left(\Phi(D_n) - \Phi(D_0)\right) + \sum_{i=1}^{n} \hat{c}_i$$

Potential Method for Binary Counter

- Potential function = ??
- $\Phi(D)$ = # of 1's in counter
- Assume that in i-th iteration Inc(b) changes
  - 1 → 0 (j bits)
  - 0 → 1 (1 bit)
  - $\Phi(D_{i-1}) = k$; $\Phi(D_i) = k - j + 1$
  - Change in potential = $(k - j + 1) - k = 1 - j$
  - Real cost = $j + 1$
  - Amortized cost = Real cost + change in potential
  - Amortized cost = $j + 1 - j + 1 = 2$

Problem 2: Stack Operations

- Data Structure: Stack
- Operations:
  - *Push(s,x)* : Push object *x* into stack *s*.
    - Cost: T(push) = O(1).
  - *Pop(s) :* Pop the top object in stack *s*.
    - Cost: T(pop) = O(1).
  - *MultiPop(s,k) ;* Pop the top *k* objects in stack *s*.
    - Cost: T(mp) = O(size(s)) worst case
- *Assumption:* Start with an empty stack
- *Simple analysis:* For N operations, maximum stack size = N. Worst-case cost of *MultiPop* = O(N). Total worst-case cost of N operations is at most N x T(mp) = $O(N^2)$.

Amortized analysis: Stack Operations

- Intuition: Worst case cannot happen all the time!
- Idea: pay a dollar for every operation, then count carefully.
- Pay $2 for each *Push* operation, one to pay for operation, another for "future use" (pin it to object on stack).
- For *Pop* or *MultiPop*, instead of paying from pocket, pay for operations with extra dollar pinned to popped objects.
- Total cost of N operations must be less than 2 x N
- Amortized cost = T(N)/N = 2.

Potential Method for Stack Problem

- Potential function $\Phi(D)$ = # of items in stack
- Push
  - Change in potential = 1; Real cost = 1
  - Amortized Cost = 2
- MultiPop [Assume j items popped in i^{th} iter]
  - $\Phi(D_{i-1}) = k$; $\Phi(D_i) = k - j$
  - Real cost = j
  - Change in potential = -j
  - Amortized cost = Real cost + change in potential
  - Amortized cost = j - j = 0

Pop: j = 1

Online Algorithms

Online Problems

- Should I buy a car/skis/camping gear or rent them when needed?
- Should I buy Google stocks today or sell them or hold on to them?
- Should I work on my homework in Algorithms or my homework in OS or on my research?
- Decisions have to be made based on past and current request/task

How to Analyze Online Algorithms?

- Competitive analysis
  - Compare with optimal offline algorithm (OPT)
- Algorithm A is **a-competitive** if there exists constants b such that for every sequence of inputs σ:
  - $cost_A(\sigma) \leq a cost_{OPT}(\sigma) + b$

Ski Rental Problem

- Should I buy skis or rent them?
  - Rental is \$A per trip
  - Purchase costs \$B
- Idea:
  - Rent for m trips, where
    - $m = B/A$
  - Then purchase skis
- Analysis:
  - Competitiveness ratio = 2. **Why**?

Paging Problem

- Given 2-level storage system
  - Limited Faster Memory (k pages) "CACHE"
  - Unlimited Slower Memory
- Input: Sequence of page requests
- Assumption: "Lazy" response (Demand Paging)
  - If page is in CACHE, no changes to contents
  - If page is not in CACHE, make place for it in CACHE by replacing an existing page
- Need: A "page replacement" algorithm

Infinite, Online

Well-known Page Replacement Algorithms

- **LRU**: evict page whose most recent access was earliest among all pages
- **FIFO**: evict page brought in earliest
- **LIFO**: evict page brought in most recently
- **LFU**: evict page least frequently used

Comparing online algorithms?

Game between Cruel Adversary and your algorithm

- Analyze: time? performance?
  - Input length?
  - Performance depends on request sequence
    - Probabilistic models? Markov Decision process
- Competitive analysis [Sleator and Tarjan]
  - Compare with optimal offline algorithm (OPT)
    - OPT is clairvoyant; no prob assumptions; "worst-case"
- Algorithm A is a-competitive if there exists constants b such that for every σ:
  - $cost_A(\sigma) \leq a cost_{OPT}(\sigma) + b$

Optimal Algorithm for Paging

- MIN (Longest Forward Distance): Evict the page whose next access is latest.
- Cost: # of page faults
- Competitive Analysis: Compare
  - # of page faults of algorithm A **with**
  - # of page faults of algorithm MIN
- We want to compute the competitiveness of LRU, LIFO, FIFO, LFU, etc.

Lower Bound for any online algorithm

- Cannot achieve better than k-competitive!
  - No deterministic algorithm is a-competitive, a < k
    - Fix online algorithm A,
    - Construct a request sequence σ, and
    - Show that: $cost_A(\sigma) \geq k\ cost_{OPT}(\sigma)$

Adversary Model

- Sequence σ will only have k+1 possible pages
  - make 1..k+1 the first k+1 requests
  - make next request as the page evicted by A
    - A will fault on every request
    - OPT? Will not fault more than once every k requests

Upper Bound: LRU is k-Competitive

- **Lemma 1**: If any subseq has k+1 distinct pages, MIN (any alg) faults at least once
- **Lemma 2**: Between 2 LRU faults on same page, there must be k other distinct faults
  - Let T be any subsequence of σ with exactly k faults for LRU & with p accessed just before T.
  - LRU cannot fault on same page twice within T
  - LRU cannot fault on p within T
  - Thus, p followed by T requests k+1 distinct pages and MIN must fault at least once on T

LRU is k-competitive

- Partition σ into subsequences as follows:
  - Let s_0 include the first request, p, and the first k faults for LRU
  - Let s_i include subsequence after s_{i-1} with the next k faults for LRU
  - Argument applies for $T = s_i$, for every $i > 0$
  - If both algorithms start with empty CACHE or identical CACHE, then it applies to $i = 0$ also
  - Otherwise, LRU incurs k extra faults
- Thus, $cost_A(\sigma) \leq k\ cost_{OPT}(\sigma) + k$

Other Page Replacement Algorithms

- FIFO is k-competitive (Homework!)
- MFU and LIFO?

How to Analyze Online Algorithms?

- Competitive analysis
  - Compare with optimal offline algorithm (OPT)
- Algorithm A is a-competitive if there exists constants b such that for every sequence of inputs σ:
  - $cost_A(\sigma) \leq a cost_{OPT}(\sigma) + b$

Alternative Analysis Technique

- Cannot consider requests separately since
  - If $cost_A = 1$ and $cost_{OPT} = 0$, ratio = infinity
- So **amortize** on a sequence of requests
- We achieve this using a **Potential Function**
  - Let's first do this for LRU

LRU Analysis using potential functions

- Define the potential function as follows:
  - $\Phi(t) = \Sigma_{x \varepsilon\ (LRU - OPT)}\ Rank(x)$
  - Here Rank(x) is its position in LRU counted from the least recently used item
- Consider an arbitrary request
- Assume that OPT serves request first
- Then LRU serves request
- We will show that for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$

LRU Analysis (Cont'd): OPT serves

- We will show that for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$
- If OPT has a hit, then
  - $cost_{LRU}(t) = cost_{OPT}(t) = \Delta\Phi = 0$
- If OPT has a miss, then
  - $cost_{LRU}(t) = 0$
  - $cost_{OPT}(t) = 1$
  - $\Delta\Phi \leq k$
    - Because OPT may evict something in LRU

LRU Analysis (Cont'd): LRU serves

- We will show that for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$
- If LRU has a hit, then
  - $cost_{LRU}(t) = cost_{OPT}(t) = 0$; $\Delta\Phi \leq 0$
- If LRU has a miss, then
  - $cost_{LRU}(t) = 1$; $cost_{OPT}(t) = 0$
  - There exists at least one item x in **LRU – OPT**
  - If x is evicted, then $\Delta\Phi \leq -w(x) \leq -1$
  - If not, its rank is reduced by ≥ 1. Thus $\Delta\Phi \leq -1$

LRU Analysis

- Thus for each step t, we have
  - $cost_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k\ cost_{OPT}(t)$
- Adding over all steps t, we get
  - $\Sigma cost_{LRU}(t) + \Sigma(\Phi(t) - \Phi(t-1)) \leq k\ \Sigma cost_{OPT}(t)$
  - $\Sigma cost_{LRU}(t) + \Phi(m) - \Phi(0) \leq k\ \Sigma cost_{OPT}(t)$
  - But $\Phi(0) = 0$, and
  - $\Phi(m) \geq 0$
  - Thus, $cost_A(\sigma) \leq k\ cost_{OPT}(\sigma)$

DBL(2c)

- DBL(2c) has 2 lists
  - L_1 is list of pages accessed once
  - L_2 is list of pages accessed once
  - Any hit moves item to $MRU(L_2)$
  - Any miss has 2 cases
    - If L_1 has c items, then move new item to $MRU(L_1)$ and delete $LRU(L_1)$
    - If L_1 has at most c items, then move new item to $MRU(L_1)$ and delete $LRU(L_2)$

Adaptive Replacement Cache (ARC)

Megiddo & Modha, FAST 2003

ARC(c)

INPUT: The request stream $x_1, x_2, \ldots, x_t, \ldots$
INITIALIZATION: Set $p = 0$ and set the LRU lists T_1, B_1, T_2, and B_2 to empty.

For every $t \geq 1$ and any x_t, one and only one of the following four cases must occur.

Case I: x_t is in T_1 or T_2. A cache hit has occurred in ARC(c) and DBL($2c$). [illegible]
Move x_t to MRU position in T_2.

Case II: x_t is in B_1. A cache miss (resp. hit) has occurred in ARC(c) (resp. DBL($2c$)).

$\boxed{\text{ADAPTATION:}}$ Update $p = \min\{p + \delta_1, c\}$ where $\delta_1 = \begin{cases} 1 & \text{if } |B_1| \geq |B_2| \\ |B_2|/|B_1| & \text{otherwise.} \end{cases}$

REPLACE(x_t, p). Move x_t from B_1 to the MRU position in T_2 (also fetch x_t to the cache).

Case III: x_t is in B_2. A cache miss (resp. hit) has occurred in ARC(c) (resp. DBL($2c$)).

$\boxed{\text{ADAPTATION:}}$ Update $p = \max\{p - \delta_2, 0\}$ where $\delta_2 = \begin{cases} 1 & \text{if } |B_2| \geq |B_1| \\ |B_1|/|B_2| & \text{otherwise.} \end{cases}$

REPLACE(x_t, p). Move x_t from B_2 to the MRU position in T_2 (also fetch x_t to the cache).

Case IV: x_t is not in $T_1 \cup B_1 \cup T_2 \cup B_2$. A cache miss has occurred in ARC(c) and DBL($2c$).

Case A: $L_1 = T_1 \cup B_1$ has exactly c pages.
If ($|T_1| < c$)
Delete LRU page in B_1. REPLACE(x_t, p).
else
Here B_1 is empty. Delete LRU page in T_1 (also remove it from the cache).
endif
Case B: $L_1 = T_1 \cup B_1$ has less than c pages.
If ($|T_1| + |T_2| + |B_1| + |B_2| \geq c$)
Delete LRU page in B_2, if ($|T_1| + |T_2| + |B_1| + |B_2| = 2c$).
REPLACE(x_t, p).
endif
Finally, fetch x_t to the cache and move it to MRU position in T_1.

Subroutine REPLACE(x_t, p)
If (($|T_1|$ is not empty) and (($|T_1|$ exceeds the target p) or (x_t is in B_2 and $|T_1| = p$)))
Delete the LRU page in T_1 (also remove it from the cache), and move it to MRU position in B_1.
else
Delete the LRU page in T_2 (also remove it from the cache), and move it to MRU position in B_2.
endif

Analyzing Rand Online Algorithms?

- Algorithm A is **a-competitive** if there exists constants b such that for every sequence of inputs σ:
  - $cost_A(\sigma) \leq a cost_{OPT}(\sigma) + b$
- Randomized Algorithm R is **a-competitive** if there exists constants b such that for every sequence of inputs σ:
  - $E[cost_R(\sigma)] \leq a cost_{OPT}(\sigma) + b$

Adversary provides request sequence at start

What to read next?

- Heaps and Priority Queues
- Heap Sort