COT 6405: Analysis of Algorithms

Giri NARASIMHAN

Amortized Analysis
Problem 1: Binary Counter

- **Data Structure**: binary counter \( b \).
- **Operations**: \( \text{Inc}(b) \).
  - Cost of \( \text{Inc}(b) \) = 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 = \( O(N\log(N)) \)
Amortized Analysis: Potential Method

- For \( n \) operations, the data structure goes through states: \( D_0, D_1, D_2, \ldots, D_n \) with costs \( c_1, c_2, \ldots, c_n \).
- 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:
  \[
  \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})
  \]
- The total amortized cost is
  \[
  \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^{n} c_i
  \]
  \[
  \sum_{i=1}^{n} c_i = -(\Phi(D_n) - \Phi(D_0)) + \sum_{i=1}^{n} \hat{c}_i
  \]
Potential Method for Binary Counter

- Potential function = ??
- $\Phi(D) = \#\ of\ 1's\ in\ counter$
- Assume that in $i$-th iteration $Inc(b)$ changes
  - $1 \rightarrow 0$ ($j$ bits)
  - $0 \rightarrow 1$ (1 bit)
- $\Phi(D_{i-1}) = k$; $\Phi(D_i) = k - j + 1$
- Change in potential = $(k - j + 1) - k = -j$
- 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 \( N \times T(mp) = O(N^2) \).
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 = \( \frac{T(N)}{N} = 2 \).**
Potential Method for Stack Problem

- Potential function $\Phi(D) = \# \text{ of items in stack}$
- 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 \textit{a-competitive} if there exists constants b such that for every sequence of inputs $\sigma$:
    - $\text{cost}_A(\sigma) \leq a\text{cost}_{OPT}(\sigma) + b$
Ski Rental Problem

- Should I buy skis or rent them?
  - Rental is $A per trip
  - Purchase costs $B

- Idea:
  - Rent for m trips, where
    - \( m = \frac{B}{A} \)
  - 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
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?

- 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 $\alpha$-competitive if there exists constants $b$ such that for every $\sigma$:
  - $\text{cost}_A(\sigma) \leq \alpha \text{cost}_{OPT}(\sigma) + b$

Game between Cruel Adversary and your algorithm
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 $\alpha$-competitive, $\alpha < k$
    - Fix online algorithm $A$,
    - Construct a request sequence $\sigma$, and
    - Show that: $\text{cost}_A(\sigma) \geq k \times \text{cost}_{\text{OPT}}(\sigma)$

- Sequence $\sigma$ 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
    - $\text{OPT}$? 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 $\sigma$ 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 $\sigma$ 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, $\text{cost}_A(\sigma) \leq k \text{cost}_{OPT}(\sigma) + k$
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 \( \alpha \)-competitive if there exists constants \( b \) such that for every sequence of inputs \( \sigma \):
    - \( \text{cost}_A(\sigma) \leq \alpha \text{cost}_{\text{OPT}}(\sigma) + b \)
Alternative Analysis Technique

- Cannot consider requests separately since
  - If $\text{cost}_A = 1$ and $\text{cost}_{\text{OPT}} = 0$, 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:
  \[ \Phi(t) = \sum_{x \in (LRU - OPT)} \text{Rank}(x) \]
  - Here \( \text{Rank}(x) \) 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
  \[ \text{cost}_{LRU}(t) + \Phi(t) - \Phi(t-1) \leq k \text{cost}_{OPT}(t) \]
LRU Analysis (Cont’d): OPT serves

- We will show that for each step $t$, we have
  \[ \text{cost}_{\text{LRU}}(t) + \Phi(t) - \Phi(t-1) \leq k \text{cost}_{\text{OPT}}(t) \]

- If OPT has a hit, then
  \[ \text{cost}_{\text{LRU}}(t) = \text{cost}_{\text{OPT}}(t) = \Delta \Phi = 0 \]

- If OPT has a miss, then
  \[ \text{cost}_{\text{LRU}}(t) = 0 \]
  \[ \text{cost}_{\text{OPT}}(t) = 1 \]
  \[ \Delta \Phi \leq k \]
  - Because OPT may evict something in LRU
LRU Analysis (Cont’d): LRU serves

- We will show that for each step $t$, we have
  \[ \text{cost}_{\text{LRU}}(t) + \Phi(t) - \Phi(t-1) \leq k \text{cost}_{\text{OPT}}(t) \]

- If LRU has a hit, then
  \[ \text{cost}_{\text{LRU}}(t) = \text{cost}_{\text{OPT}}(t) = 0; \Delta \Phi \leq 0 \]

- If LRU has a miss, then
  \[ \text{cost}_{\text{LRU}}(t) = 1; \text{cost}_{\text{OPT}}(t) = 0 \]
  There exists at least one item $x$ in $\text{LRU} - \text{OPT}$
  - If $x$ is evicted, then $\Delta \Phi \leq -w(x) \leq -1$
  - If not, its rank is reduced by $\geq 1$. Thus $\Delta \Phi \leq -1$
Thus for each step $t$, we have

$$\text{cost}_{\text{LRU}}(t) + \Phi(t) - \Phi(t-1) \leq k \text{cost}_{\text{OPT}}(t)$$

Adding over all steps $t$, we get

$$\Sigma \text{cost}_{\text{LRU}}(t) + \Sigma (\Phi(t) - \Phi(t-1)) \leq k \Sigma \text{cost}_{\text{OPT}}(t)$$

$$\Sigma \text{cost}_{\text{LRU}}(t) + \Phi(m) - \Phi(0) \leq k \Sigma \text{cost}_{\text{OPT}}(t)$$

But $\Phi(0) = 0$, and

$$\Phi(m) \geq 0$$

Thus, $\text{cost}_A(\sigma) \leq k \text{cost}_{\text{OPT}}(\sigma)$
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
Analyzing Rand Online Algorithms?

- Algorithm A is \( \alpha \)-competitive if there exists constants \( b \) such that for every sequence of inputs \( \sigma \):
  \[
  \text{cost}_A(\sigma) \leq \alpha \text{cost}_{\text{OPT}}(\sigma) + b
  \]

- Randomized Algorithm R is \( \alpha \)-competitive if there exists constants \( b \) such that for every sequence of inputs \( \sigma \):
  \[
  \mathbb{E}[\text{cost}_R(\sigma)] \leq \alpha \text{cost}_{\text{OPT}}(\sigma) + b
  \]
What to read next?

- Heaps and Priority Queues
- Heap Sort