COP 4531
Complexity & Analysis of Data Structures & Algorithms

Amortized Analysis

Thanks to the text authors who contributed to these slides
What is amortized analysis?

- Analyze a sequence of operations on a data structure.
- **Goal:** Show that although some individual operations may be expensive, on average the cost per operation is small.
- Average in this context does not mean that we’re averaging over a distribution of inputs.
- No probability is involved.
- We are talking about average cost in the worst case.
  - Thus worst case bounds still hold
  - We can often get a tighter bound on the running time using such analysis
We will consider three approaches

• aggregate analysis
  - determine an upper bound $T(n)$ on a sequence of $n$ operations
  - average cost per operation is then $T(n) / n$

• accounting method
  - determine amortized costs of each operation (may differ)
  - overcharge some operations (give some extra credit) and undercharge others (using credit)
  - ensure that credit is always $\geq 0$

• potential method
  - similar to the accounting method
  - uses a potential function that defines a “potential energy” value of the system (data structure) as each operation is done
Aggregate analysis
Example: stack operations

- **Operations**
  - **PUSH(S, x):** $O(n)$ for any sequence of $n$ operations (each push is $O(1)$)
  - **POP(S, x):** $O(n)$ for any sequence of $n$ operations (each pop is $O(1)$; POP if stack empty is an error)
  - **MULTIPOP(S, k):** POP $k$ objects in stack but only until stack is empty
Running time of MULTIPOP and a sequence of $n$ operations

- Linear in $\#$ of POP operations
- We are assuming each PUSH, POP has cost 1
- Based on this, MULTIPOP cost is $\min(s, k)$ where $s$ is the number of objects on the stack
- Now consider a sequence of $n$ operations (either POP, PUSH, or MULTIPOP)
  - worst case for MULTIPOP is $O(n)$
  - a naïve analysis would thus give a worst case cost for the sequence to be $O(n^2)$
Consider a more detailed analysis

- Each object can be popped only once per time that it’s pushed.
- If you have $\leq n$ PUSHes, then you must have $\leq n$ POPs (including those arising from MULTIPOP)
- Thus, total cost is $O(n)$
- Therefore, average over the $n$ operations is $O(1)$ on average
- Note that we are not discussing expected cost in the sense of probability
- For any sequence of operations of length $n$, we have shown that worst case cost is $O(n)$.
- The technique we used is “aggregate analysis”
Aggregate analysis
Example: binary counter

- **Binary counter**
  - k-bit binary counter $A[0..k-1]$, where $A[0]$ is least significant bit
  - Counts upward from 0
  - Value of binary counter is $\sum_{i=0}^{k-1} A[i]2^i$
  - Initially, counter value is 0.
  - To increment, add 1 (mod $2^k$)
  - We are interested in the number of bit change operations that we need to do in a sequence of increment operations. A single bit change operation is viewed as $O(1)$ or cost 1.
  - Each increment operation may require a different number of bit changes

bits that are flipped. The cost of the INCREMENT is the number of bits flipped
INCREMENT(A, k)

\[ i = 0 \]

\[ \textbf{while } i < k \textbf{ and } A[i] == 1 \]

\[ A[i] = 0 \text{ /reset 1 bit} \]

\[ i = i + 1 \]

\[ \textbf{if } i < k \]

\[ A[i] = 1 \text{ /set a 0 bit} \]
Naïve analysis and aggregate analysis

• Naïve
  - Each call could flip \( k \) bits, so \( n \) INCREMENTS take \( O(nk) \) time

• Aggregate
  - not every bit flips each time
    • \( 0 \) bit flips every time
    • \( 1 \) bit flips \( \frac{1}{2} \) time or \( \lfloor n/2 \rfloor \) times in \( n \) increments
    • \( i \)th bit flips \( 1/2^i \) of the time or \( \lfloor n/2^i \rfloor \) times in \( n \) increments
  - total number of flips in \( n \) INCREMENTS
    \[ \sum_{i=0}^{k-1} \lfloor n/2^i \rfloor < \sum_{i=0}^{\infty} n/2^i = 2n \]
  - \( n \) INCREMENTS cost \( O(n) \); average cost is \( O(1) \)
Accounting method

• Assign different charges to different operations
  - some are charged more than actual cost
  - some are charged less

• Amortized cost is the amount we charge
  - When amortized cost > actual cost, store the difference with specific objects in the data structure as credit
  - When an operation is executed such that actual cost > amortized cost, we use the credit at this time

• Differs from aggregate analysis as follows:
  - In accounting method, different operations can have different costs
  - In aggregate analysis, we look at n operations and simply average over these as if all operations have same cost
Amortized cost must be an upper bound of actual cost

• At any point in a sequence of operations we still want to ensure that we know the worst case cost.
• Thus at any point the amortized cost must be ≥ actual cost.
• This translates to the credit never becoming negative at any point in time.
• Let $c_i$ be the action cost of the $i$th operation.
Let $\hat{c}_i$ be the amortized cost of the $i$th operation.
• Then we require that:
  \[ \sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i \]
  for all sequences of $n$ operations.
• Total credit stored = $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$ must be ≥ 0.
Example with Stack

<table>
<thead>
<tr>
<th>operation</th>
<th>actual cost</th>
<th>amortized cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>POP</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MULTIPOP</td>
<td>min(k, s)</td>
<td>0</td>
</tr>
</tbody>
</table>

Intuition: When pushing an object, pay $2.
- $1 pays for the PUSH.
- $1 is prepayment for it being popped by either POP or MULTIPOP.
- Since each object has $1, which is credit, the credit can never go negative.
- Therefore, total amortized cost, =O(n), is an upper bound on total actual cost.
Binary counter

Charge $2 to set a bit to 1.
- $1 pays for setting a bit to 1.
- $1 is prepayment for flipping it back to 0.
- Have $1 of credit for every 1 in the counter.
- Therefore, credit $\geq 0$.

Amortized cost of INCREMENT:
- Cost of resetting bits to 0 is paid by credit.
- At most 1 bit is set to 1.
- Therefore, amortized cost $\leq $2.
- For n operations, amortized cost $= O(n)$
Summary

• We have shown how to get bounds on the worst case complexity of a sequence of $n$ operations by various ways of amortizing the cost.
• The accounting method and potential method are similar.
• The aggregate analysis method is quite commonly used to get a more precise cost on the running time.