COEN279 - Design and Analysis of Algorithms

Amortized Analysis

Dr. Tunghwa Wang

Fall 2016

Announcement

- Grader: Xiaoyu Li
  - Email: natalienew2016@gmail.com
  - Future submissions to her: CC to me
    - Filename should contain your name and type of homework: for example, TunghwaWang-Bonus2
    - You may resubmit, same file names: new file replaces the old one.
- Programming assignment #2:
  - Submission:
    - When you run “perl Submit P2”, you should see the message of successful submission to natalienew2016@gmail.com and twang1@cse.scu.edu.

Amortized Analysis

- Complexity analysis:
  - We average the resource, such as time, required to perform a sequence of data-structure operations over all operations performed:
    - Rather than analyzing a single operation.
    - Goal to:
      - Show average cost of an operation without involving probability.
      - Guarantee the average performance of each operation in the worst case.
  - Methods:
    - Aggregate analysis
    - Accounting method
    - Potential method

Stack Operations

- We know that for stack S:
  - S.Push: O(1)
  - S.Pop: O(1)
- Adding MultiPop operation:
  - Popping out k element from stack S in one call.
  - S.MltiPop(k): O(min(k, s)) where s is the number of elements in stack
- Analyze amortized time complexity.

Binary Counter

- For k-bit counter A with initial value 0.
  - A[i] is the i-th bit where i = 0, 1, …, k-1.
  - A[0] is the LSB and A[k-1] is MSB.
- Only operation A.Increment: O(k)

```
INCREMENT(A)
1     i = 0
2     while i < A.length and A[i] = 1
3           A[i] = 0
4           i = i + 1
5     if i < A.length
6           A[i] = 1
```

- Analyze amortized cost complexity of bits flipped.

Aggregate Analysis

- Key idea:
  - For a all n, a sequence of n operations takes worst-case T(n) in total.
  - Amortized complexity shall be T(n)/n.
- Stack:
  - Starting from empty stack, for pop operation to succeed, there must be at least once Push called.
    - S.Pop(): stack must have at least an element.
    - S.MultiPop(k): stack must have at least an element.
  - Thus, any n operations of Push, Pop, and MultiPop operations can cost at most O(n):
    - There can be no more successful pop operations than push operations.
  - So the amortized complexity is O(n)/n = O(1).

Aggregate Analysis

- Key idea:
  - For a all n, a sequence of n operations takes worst-case C(n) in total.
  - Amortized complexity shall be C(n)/n.
- Counter:
  - Starting from 0, number of times a bit get flipped depends on i:
    - i = 0: it is flipped each operation (once out of 2^0 operations)
    - i = 1: it is flipped every other operation (once out 2^1 operations)
    - i = 2: it is flipped once out 2^2 operations
    - …
    - i = k-1: it is flipped once out of 2^{k-1}

$$1 + 1/2 + 1/2^2 + \ldots + 1/2^n$$
$$= (1 - 1/2^{n+1}) / (1 - 1/2)$$
$$= 2 \text{ when n approaches infinity}$$

  - Thus, C(n) ≤ 2n.
  - So the amortized complexity is O(n)/n = O(1).

Accounting Method

- Key idea:
  - For each operation, we assign amortized cost which can be more or less than the actual cost.
    - Simplifying the complexity of actual cost with upper bound.
  - Let T(n) = sum of amortized costs from 1, 2, …, n
  - Amortized complexity shall be T(n)/n.
- Stack:
  - Actual cost:
    - S.Push(): 1
    - S.Pop(): 1
    - S.MultiPop(k): min(k, s)
  - Amortized cost:
    - S.Push(): 2
    - S.Pop(): 0
    - S.MultiPop(k): 0
  - So the amortized complexity is O(n)/n = O(1).

Accounting Method

- Key idea:
  - For each operation, we assign amortized cost which can be more or less than the actual cost.
    - Simplifying the complexity of actual cost with upper bound.
  - Let C(n) = sum of amortized costs from 1, 2, …, n
  - Amortized complexity shall be C(n)/n.
- Counter:
  - Starting from 0:
    - Assign amortized cost 2 for flipping 0 to 1
    - Assign amortized cost 0 for flipping 1 to 0
  - Pseudocode 2-4 (while loop) contributes no amortized cost, thus pseudocode 5-6 gives C(n) = 2n (without overflow).
    - With overflow, C(n) ≤ 2n.
  - So the amortized complexity is O(n)/n = O(1).

Potential Method

- Key idea:
  - Similar to accounting methods, but defines potential function Φ_i as potential energy for data structure D_i.
    - Initial data structure D_0 with potential energy Φ_0.
    - The amortized cost for D_i from D_{i-1} then is actual cost + Φ_i - Φ_{i-1}.
  - Let T(n) = sum of amortized costs from 1, 2, …, n
    - It is (sum of actual costs) + $\Phi_n - \Phi_0$.
    - For upper bound, we must have $\Phi_k \geq \Phi_0$ for all $1 \leq k \leq n$.
  - Amortized complexity shall be T(n)/n.

Potential Method

- Stack:
  - Define Φ_k = number of elements in stack S denoted as S_k.
  - Starting from empty S_0:
    - $\Phi_0 = 0$.
    - $\Phi_k = k \geq 0$ for all $1 \leq k \leq n$.
    - $\Phi_k - \Phi_{k-1} = k - (k-1) = 1$ for all $1 \leq k \leq n$.
  - Operations:
    - S.Push: $1 + (\Phi_k - \Phi_{k-1}) = 2$ for all $1 \leq k \leq n$.
    - S.Pop: $1 + (\Phi_{k-1} - \Phi_k) = 0$ for all $1 \leq k \leq n$.
    - S.MultiPop: $s' + (\Phi_{k-s'} - \Phi_k) = 0$ for $s' = \min(k', s)$ and all $1 \leq k \leq n$.
  - So the amortized complexity is $O(n)/n = O(1)$.

Potential Method

- Counter:
  - Define Φ_i = number of 1's in bit array A after i-th operation.
  - Starting from 0 as D_0:
    - $\Phi_0 = 0$.
    - $\Phi_i \geq 0$ for all $1 \leq i \leq n$.
    - $\Phi_i - \Phi_{i-1} = 1 - t_i$ where t_i is the number of bit reset to 0 for all $1 \leq i \leq n$. (without overflow)
      - With overflow, $\Phi_i - \Phi_{i-1} = - t_i \leq 1 - t_i$
  - Operations:
    - A.Increment: $(t_i + 1) + (\Phi_i - \Phi_{i-1}) \leq 2$ for all $1 \leq i \leq n$.
  - This gives $C(n) \leq 2n$.
  - So the amortized complexity is $O(n)/n = O(1)$.

Amortized Analysis

- For an operation:
  - We may apply the same technique to analyze a sequence of n identical operations.
  - Thus, we can still claim amortized complexity of an operation even though it is simply one of many operations of a data structure.

Backup Slides