Amortized Analysis
Network Flows

Amit Sengupta

Introduction

- Analysis of an algorithm checks how the running time of the algorithm scales with the input size
- Four types of analysis:
    - Empirical analysis
    - Average case analysis
    - Worst case analysis
    - Amortized analysis

Introduction (Contd.)

- **Empirical analysis**
    - Write a program for the algorithm and test the performance of the algorithm on some problem instances
    - Drawbacks
        - Time consuming and expensive
        - Depends on the computing resources and the programmer skills
        - Often inconclusive

Introduction (Contd.)

- **Average case analysis**
  - Estimate the expected number of steps of the algorithm based on a probability distribution
  - Drawbacks:
    - Analysis depends on the choice of the probability distribution
    - Analysis is difficult
    - Performance prediction depends on situations where you solve many problem instances

Introduction (Contd.)

- **Worst case analysis**
    - Gives an upper bound on the number of steps the algorithm takes on any instance
    - Independent of the computing environment
    - Analysis is easier
    - Conclusive about comparing algorithms
    - Drawbacks:
        - A rare instance can determine the performance

Introduction (Contd.)

- **Amortized analysis**
  - Used when most operations are fast but an occasional operation is slow
  - Time needed to do a seq. of operations is averaged over all operations
  - No probability distribution
  - Not sensitive to a rare instance

Topics

- Aggregate method: the amortized cost per operation is $\frac{T(n)}{n}$ where $T(n)$ is the worst case time for a sequence of n operations
- Potential method: the amortized cost per each operation is the sum of its actual cost and the increase in potential due to the operation
- Examples
    - Stack operations
    - Incrementing a binary counter

Aggregate Method

- Consider a stack S (of size n) and these operations
  - Push (S, x): push object x into stack S
  - Pop (S): pop the top of stack S and return the object
  - Multipop (S, k): pop k top objectsof stack S
- Consider a sequence of n Push, Pop, and Multipop operations
- The worst case cost of a Multipop operation is $O(n)$
- So the worst case cost of any operation is $O(n)$
- The worst case cost of n operations is $O(n^2)$

Aggregate Method (Contd.)

- Aggregate method gives a better bound
- Each object can be popped at most once for each time it is pushed
- So the number of times Pop is called including calls in Multipop is at most the number of Push, which is $\leq n$
- So a sequence of n Push, Pop, and Multipop operations require $O(n)$ time
- The amortized cost of each operation is $\frac{O(n)}{n} = O(1)$, an improvement from $O(n)$

Aggregate Method (Contd.)

- Increment a binary counter
  - Consider the counter values 0,1,…,7 (rightmost bit is A[0] and leftmost bit is A[7]) and the increment operation

0	0	0	0	0	0	0	**0**
0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	**0**
0	0	0	0	0	**0**	**1**	**1**
0	0	0	0	0	1	0	**0**
0	0	0	0	0	1	**0**	**1**
0	0	0	0	0	1	1	**0**
0	0	0	0	**0**	**1**	**1**	**1**

Aggregate Method (Contd.)

```
Increment(A) {
    i = 0;
    while (i < length (A) and A[i] = 1) {
            A[i] = 0;
            i = i + 1
    }
    if (i < length(A))
            A[i] = 1
}
```

Aggregate Method(Contd.)

- An increment operation in the worst case takes $O(k)$ time if all k bits are 1
- So n increment operations take $O(nk)$ time in the worst case
- We can get a better bound using the aggregate method
- $A[0]$ flips each time increment is called
- $A[1]$ flips every second time increment is called
- $A[2]$ flips every fourth time increment is called

Aggregate Method (Contd.)

- Amortized cost of n increments $= n(1 + \frac{1}{2} + \frac{1}{4} + ..\text{up to } \log n \text{ terms}) \; < n \sum_{i=1}^{\infty} \frac{1}{2^i} = 2n$
- Amortized cost of one increment = $\frac{O(n)}{n} = O(1)$
- Improvement from $O(k)$ to $O(1)$

Potential Method

- c_i: the actual cost of the i^{th}operation and D_i is the data structure after the i^{th}operation
- $\emptyset$ is the potential function that maps D_i to a real number $\emptyset(D_i)$
- $\widehat{c_i}$: the amortized cost of the i^{th}operation
- $\widehat{c_i} = c_i + \emptyset(D_i) - \emptyset(D_{i-1})$

Potential Method(Contd.)

- Consider a stack S with Push, Pop, and Multipop operations
- Define ∅ as the number of objects in the stack
- Amortized cost of Push = $1 + ((k+1) - k) = 2$
- Amortized cost of Pop = $1 + \left((k-1) - k\right) = 0$
- Amortized cost of Multipop $= c_i + (\,\emptyset(D_i) - \emptyset(D_{i-1})\,)$

  $= k + (-k) = 0$
- Amortized cost of n operations $= O(n)$
- Amortized cost of one operation $= O(1)$

Potential Method (Contd.)

- Consider incrementing a binary counter
- Define potential as the number of 1s after the i^{th} operation, say y_i
- Let the i^{th} operation resets x_i bits
- So the actual cost of the i^{th} operation is x_i+1 (1 for setting one bit)
- The number of $1s$ after the i^{th} operation is $y_i \leq y_{i-1} - x_i + 1$

Potential Method (Contd.)

- $\emptyset(D_i) - \emptyset(D_{i-1}) \leq (y_{i-1} - x_i + 1) - y_{i-1} = 1 - x_i$
- $\widehat{c_i} = c_i + \emptyset(D_i) - \emptyset(D_{i-1})$

  $\leq (x_i + 1) + (1 - x_i) = 2$
- So the amortized cost of n increment operations is $O(n)$
- The amortized cost of one increment operation is $O(1)$, an improvement from $O(n)$

Summary

- Amortized analysis is used when most operations are fast but an occasional operation is slow
- Discussed the aggregate method and potential method for stack operations and incrementing a binary counter
- Demonstrated improvement of the amortized cost of n operations over the worst case cost of n operations
- Another application is operations in hash table where most insert/find take $O(1)$ time but an occasional insert/find can take $O(n)$ due to collision

Reference

- Thomas Cormen, Charles Leiserson, and Ronald Rivest, Introduction to Algorithms, the MIT Press (1990)
- Ravindra Ahuja, Thomas Magnenti, and James Orlin, Network Flows, Prentice Hall (1993)