AMORTIZED ANALYSIS OF ALGORITHMS

Presented by:

Dr. Shivangi K. Surati

Assistant Professor,
Department of Computer Science and Engineering,
School of Technology,
Pandit Deendayal Energy University

Introduction

- Used for algorithms where an occasional operation is very slow
  - but most of the other operations are faster
- Analyze a sequence of operations
  - Such a way that the time required to perform a sequence of data structure operations is averaged over all the operations performed

Introduction…

- Amortized analysis can be used to show that the average cost of operation is small, if one averages over a sequence of operations, even though a single operation within a sequence might be expensive
  - guarantee a worst-case average time that is lower than the worst-case time of a particularly expensive operation
- Not an average case analysis of algorithm!
  - does not always take the average case scenario
  - cases that occur as a worst-case scenario of analysis

- Amortized cost = cost of *n* operations / *n*

Methods of Amortized Analysis

- Aggregate Method
- Accounting Method
- Potential Method

Aggregate Method

Algorithms

1. Dynamic Table
2. Binary Counter
3. Stack (Push, Pop, Multipop)

1. Dynamic Table

Item No (i)	Table Size	Cost of Operation	Cost of Doubling and Copying	Total Cost$_i$ (C_i)
1	1	1	0	1
2	2	1	1	2
3	4	1	2	3
4	4	1	0	1
5	8	1	4	5
6	8	1	0	1
7	8	1	0	1
8	8	1	0	1
9	16	1	8	9
10	16	1	0	1
11	16	1	0	1
12	16	1	0	1
13	16	1	0	1
14	16	1	0	1
15	16	1	0	1
16	16	1	0	1
17	32	1	16	17

Dynamic Table Algorithm

TABLE(T, x) ⟶ $\Theta(n^2)$

1) If size[T] == 0
2) then size[T] ← 1 — $\Theta(1)$
3) If num[T] == size[T] — $\Theta(n)$
4) then allocate a new-table of double size — $\Theta(n)$
5) insert x into new table. — $\Theta(1)$
6)
7) TABLE[T] ← new-table
8) size[T] ← 2*size[T]
9) num[T] ← num[T] + 1
10) return table[T]

2. **Binary Counter**

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

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

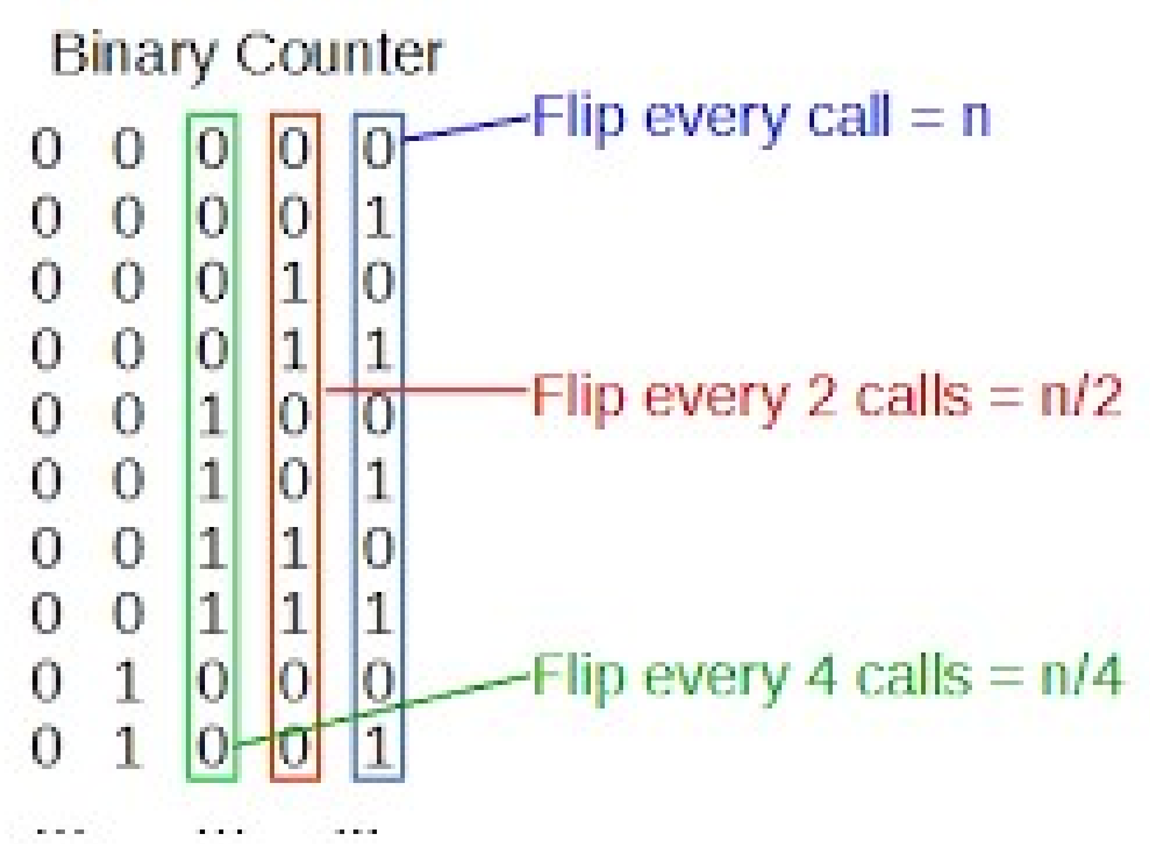
Binary Counter…

Binary	Flips	Total	
0 0 0 0 0			
0 0 0 0 1	1	1	
0 0 0 1 0	2	3	2^i-1
0 0 0 1 1	1	4	
0 0 1 0 0	3	7	2^i-1
0 0 1 0 1	1	8	
0 0 1 1 0	2	10	
0 0 1 1 1	1	11	
0 1 0 0 0	4	15	2^i-1
0 1 0 0 1	1	16	
… …	…	…	

Worst case Analysis

- Number of bits =k
- Number of elements = n
- A single execution of INCREMENT takes time θ(k) in the worst case.
- Hence, n elements (n INCREMENTS) take O(nk) in the worst case.

Amortized Analysis



Amortized Analysis…

Observation: The running time determined by #flips but not all bits flip each time INCREMENT is called.

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

A[0] flips every time, total n times.

A[1] flips every other time, $n/2$ times.

A[2] flips every forth time, $n/4$ times…..

for i=0,1,…,k-1, A[i] flips $n/2^i$ times.

Amortized Analysis…

- A single exec. of INCREMENT takes $O(k)$ worst case time.
- A sequence of n increments would take time $O(nk)$.
- We can tighten our analysis to yield a worst case of $O(n)$ op.
- $A[0]$ is flipped every time we call INCREMENT.
- $A[1]$ is only flipped every other time.
- $A[2]$ is flipped $\lfloor n/4 \rfloor$ times.
- In general, for $i = 0, 1, \ldots, \lfloor \lg n \rfloor$, bit $A[i]$ flips $\lfloor n/2^i \rfloor$ times in a sequence of n increments.
- For $i > \lfloor \lg n \rfloor$, bit $A[i]$ never flips at all.
- The total number of flips is $\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$.

Amortized Analysis…

- The worst case time for a sequence of n INCREMENT operations on an initially zero counter is therefore $O(n)$.
- The average cost of each operation and therefore the amortized cost per operation is $O(n)/n = O(1)$.

Accounting Method
(Banker's Method)

Idea

- Investing for future
- Assign differing charges to different operations.
- The amount of the charge is called amortized cost.
- Amortized cost is more or less than actual cost.
- When amortized cost **>** actual cost, the difference is saved in specific objects (termed as bank) as credits.
- The credits can be used by later operations whose amortized cost **<** actual cost.

Algorithms

1. Dynamic Table – in notebook
2. Binary Counter
3. Stack (Push, Pop, Multipop) - HW

2. **Binary Counter**

- Let \$1 represent each unit of cost (the flip of one bit).
- Charge an amortized cost of \$2 to set a bit to 1.
- Whenever a bit is set, use \$1 to pay the actual cost, and store another \$1 on the bit as credit.
- When a bit is reset, the stored \$1 pays the cost.
- At any point, a 1 in the counter stores \$1, the number of 1's is never negative, so is the total credits.
- At most one bit is set in each operation, so the amortized cost of an operation is at most \$2.
- Thus, total amortized cost of n operations is $O(n)$, and average is $O(1)$.

Potential Method

Idea

- Same as accounting method-
  - something prepaid is used later
- Different from accounting method-
  - The prepaid work not as credit, but as "potential energy", or "potential".
  - The potential is associated with the data structure as a whole rather than with specific objects within the data structure.

Idea…

- Designed on the concept of ‘Potential energy’
- Compute the quantity of potential energy that the state has to provide
- Define a potential function on entire data structure
  - that changes the data structure’s states into non-negative values

- Initial data structure D_0,
- n operations, resulting in D_0, $D_1, \ldots, D_n$ with costs $C_1, C_2, \ldots, C_n$.
- A potential function Φ: $\{D_i\} \rightarrow$ R (real numbers)
- $\Phi(D_i)$ is called the potential of D_i.
- Amortized cost c_i' of the ith operation is:

$C_i' = C_i + \Phi(D_i) - \Phi(D_{i-1})$ (actual cost + potential change)

- Amortized cost of all n operations is:

$$\sum_{i=1}^{n} c_i' = \sum_{i=1}^{n} (C_i + \Phi(D_i) - \Phi(D_{i-1}))$$
$$= \sum_{i=1}^{n} C_i + \Phi(D_n) - \Phi(D_0)$$

- Amortized cost c_i' of the *i*th operation is:

  $$C_i' = C_i + \Phi(D_i) - \Phi(D_{i-1})$$

  where $\Phi(D_i) - \Phi(D_{i-1}) = \Delta\Phi_\mathrm{i}$ = potential difference

- If $\Delta\Phi_\mathrm{i} > 0$,then $C_i' > C_i$. Operation *i* stores work in the data structure for later use.
- If $\Delta\Phi_\mathrm{i} < 0$,then $C_i' < C_i$. The data structure delivers up stored work to help pay for operation *i*.

Algorithms

1. Dynamic Table
2. Binary Counter
3. Stack (Push, Pop, Multipop) - HW

Dynamic Table

- Let the potential function be

Φ = 2 * No of items in the array – capacity of the array

- Amortized cost c_i' of the ith operation is:

$$C_i' = C_i + \Phi(D_i) - \Phi(D_{i-1})$$

Dynamic Table…

Item No (i)	Table Size	Cost of Operation	Cost of Doubling and Copying	Total Cost$_i$ (C_i)	Φ	Amortized Cost C_i + Pi - Pi-1
1	1	1	0	1	2 x 1 - 1 = 1	1 + (1 - 0) = 2
2	2	1	1	2	2 x 2 - 2 = 2	2 + (2 - 1) = 3
3	4	1	2	3	2 x 3 - 4 = 2	3 + (2 - 2) = 3
4	4	1	0	1	2 x 4 - 4 = 4	1 + (4 - 2) = 3
5	8	1	4	5	2 x 5 - 8 = 2	5 + (2 - 4) = 3
6	8	1	0	1	2 x 6 - 8 = 4	1 + (4 - 2) = 3
7	8	1	0	1	2 x 7 - 8 = 6	1 + (6 - 4) = 3
8	8	1	0	1	2 x 8 - 8 = 8	1 + (8 - 6) = 3
9	16	1	8	9	2 x 9 - 16 = 2	9 + (2 - 8) = 3
10	16	1	0	1	2 x 10 - 16 = 4	1 + (4 - 2) = 3
11	16	1	0	1	2 x 11 - 16 = 6	1 + (6 - 4) = 3
12	16	1	0	1	2 x 12 - 16 = 8	1 + (8 - 6) = 3
13	16	1	0	1	2 x 13 - 16 = 10	1 + (10 - 8) = 3
14	16	1	0	1	2 x 14 - 16 = 12	1 + (12 - 10) = 3
15	16	1	0	1	2 x 15 - 16 = 14	1 + (14 - 12) = 3
16	16	1	0	1	2 x 16 - 16 = 16	1 + (16 - 14) = 3
17	32	1	16	17	2 x 17 - 32 = 2	17 + (2 - 16) = 3

Dynamic Table…

- Amortized cost = 3*n / n = O(1)

- Define potential function
  - Such that it has tight upper bound
  - Doesn’t charge too much in each operation

2. **Binary Counter**

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

```
INCREMENT(A)
i←0
while i<length[A] and A[i]=1
    A[i]=0
    i←i+1
endwhile
if i<length[A]
then A[i]←1
```

Amortized Analysis

- Define the potential of the counter after the *i*th INCREMENT is
  - $\Phi(D_i) = b_i$, the number of 1's.
    - $\Phi(0) = 0$, as all bits are zero
    - $\Phi(i) > 0$, all $i > 0$
- Define the cost C_i of i^{th} INCREMENT as
  - C_i = #bits flipped

    = #bits changed from 0 to 1 +

    #bits changed from 1 to 0

Cost function (C_i)

Row No	3	2	1	0	Ci
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	1
5	0	1	0	0	3
6	0	1	0	1	1
7	0	1	1	0	2
8	0	1	1	1	1
9	1	0	0	0	4
10	1	0	0	1	1
11	1	0	1	0	2
12	1	0	1	1	1
13	1	1	0	0	3
14	1	1	0	1	1
15	1	1	1	0	2
16	1	1	1	1	1

Amortized Cost

- Potential function is defined as:
  - $\Phi(D_i)$ = b_i, the number of 1's in the current state after i^{th} INCREMENT
- Amortized cost is:
  - $C_i' = C_i + \Phi(D_i) - \Phi(D_{i-1})$

Amortized Cost…

Row No	3	2	1	0	Φ	Ci	Amortized Cost Ci + Pi - Pi-1
1	0	0	0	0	0	0	0 + (0 - 0) = 0
2	0	0	0	1	1	1	1 + (1 - 0) = 2
3	0	0	1	0	1	2	2 + (1 - 1) = 2
4	0	0	1	1	2	1	1 + (2 - 1) = 2
5	0	1	0	0	1	3	3 + (1 - 2) = 2
6	0	1	0	1	2	1	1 + (2 - 1) = 2
7	0	1	1	0	2	2	2 + (2 - 2) = 2
8	0	1	1	1	3	1	1 + (3 - 2) = 2
9	1	0	0	0	1	4	4 + (1 - 3) = 2
10	1	0	0	1	2	1	1 + (2 - 1) = 2
11	1	0	1	0	2	2	2 + (2 - 2) = 2
12	1	0	1	1	3	1	1 + (3 - 2) = 2
13	1	1	0	0	2	3	3 + (2 - 3) = 2
14	1	1	0	1	3	1	1 + (3 - 2) = 2
15	1	1	1	0	3	2	2 + (3 - 3) = 2
16	1	1	1	1	4	1	1 + (4 - 3) = 2

Amortized Cost

- $\Phi(D_i) = b_i$, the number of 1's in the current state after i^{th} INCREMENT
- Suppose i^{th} operation resets (flip 1 to 0) t_i bits.
- Actual cost: $c_i = t_i + 1$
- If $b_i=0$, then the i^{th} operation resets all k bits, so $b_{i-1}=t_i=k$.
- If $b_i>0$, then $b_i = b_{i-1} - t_i + 1$
- In either case, $b_i \leq b_{i-1} - t_i + 1$
- So potential change is $\Phi(D_i)-\Phi(D_{i-1}) = b_{i-1} - t_i + 1- b_{i-1}$
  $= 1 - t_i$
- So amortized cost is: $C_i' = C_i + \Phi(D_i) - \Phi(D_{i-1})$
  $\leq t_i + 1 + 1 - t_i = 2$