

# Graduate Algorithms

CS673-2016F-12

## Amortized Analysis

David Galles

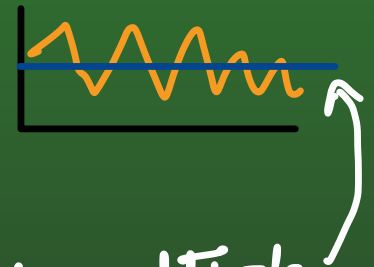
Department of Computer Science  
University of San Francisco

↳  $\frac{\text{agg all cost}}{\# \text{ operat}^n}$

↳ Done over multiple operations!

↳ look for each cost overall

↳ Done over:  
- Data struct  
- functionality  
- (e.g. search)



# 12-0: Amortized Analysis

---

- Standard Stack
  - Push(S,elem) –  $O(1)$ , independent
  - Pop(S) –  $O(1)$  of i/p
- How much time for each operation?

avg-case cost analysis  
= avg over all possible  
"inputs!"

# 12-1: Amortized Analysis

- Standard Stack
  - Push(S,elem)  $O(1)$
  - Pop(S)  $O(1)$
  - Multipop(S,k)
    - for  $i \leftarrow 1$  to  $k$  do
    - Pop(S) —  $O(1)$ , 'k' times
- How much time for multipop?
  - $O(k)$  :  $1 \leq k \leq n$   
 $O(n)$  ;  $k = n/2$  in avg.

# 12-2: Amortized Analysis

- Standard Stack
  - $\text{Push}(S, \text{elem}) \ O(1)$
  - $\text{Pop}(S) \ O(1)$
  - $\text{MultiPop}(S, k) \ O(k)$

e.g.  $\text{push} - O(1)$        $\text{push} - O(1)$   
 $\text{push} - O(1)$        $\text{push} - O(1)$   
 $\text{multiPop}(2)$        $\text{pop} - O(1)$   
     $\hookrightarrow O(2)$        $\text{push} - O(1)$   
                     $\text{multiPop}(1) - O(1)$

Thus 'multiPop' can never be more than  
# of push,  $\therefore \boxed{1 \leq \text{multiPop} \leq n}$

E.g. index : build : scan files  
                    'keyword  $\rightarrow$  list of files'  
search  $\nearrow$

- Thus overall to use index we have to build it & then use it, so initially it is costly <'built'> later efficient to use.



## 12-3: Amortized Analysis

---

- Do  $n$  operations, each of which could be either a Push, Pop, or Multipop
- How long will each operation take?
  - Push
  - Pop
  - Multipop

# 12-4: Amortized Analysis

- Do  $n$  operations, each of which could be either a Push, Pop, or Multipop
- How long will each operation take?

- Push  $O(1)$

- Pop  $O(1)$

- Multipop  $O(n)$  —  $'n-1'$  push + 1 multipop  $(S, n-1) = 2(n-1)$

*Handwritten notes: "Total work" with an arrow pointing to the equation, and " $\approx O(n)$ " below the equation.*

- What if we were to do  $n$  operations in a row, each of which is either a push/pop/multipop — how long would those  $n$  operations take?
- Handwritten notes: " $\rightarrow n \cdot O(1) / \text{operation}$ " with an arrow pointing to the question.*

## 12-5: Amortized Analysis

- $n$  operations in a row, each is either a push/pop/multipop. How long will it take?
  - Naive Method:  $n$  operations, each takes time  $O(n)$  – total time:  $O(n^2)$   
— Not necessary,  $O(1)$  at each step for prev. e.g.
- Looking closer:
  - How many times can Pop be called (even Pop in Multipop)?
    - Once for each push!
    - Total number of Pushes  $\in O(n)$
    - Total number of Pops (including pops in multipop)  $\in O(n)$
    - Total time for  $n$  operations:  $O(n)$

## 12-6: Amortized Analysis

---

- $n$  operations in a row, each is either a push/pop/multipop.
- Total time for  $n$  operations is  $O(n)$
- Amortized cost for a Push, Pop, Multipop is  $O(1)$ /operation<sup>n</sup>



## 12-7: Aggregate Method

---

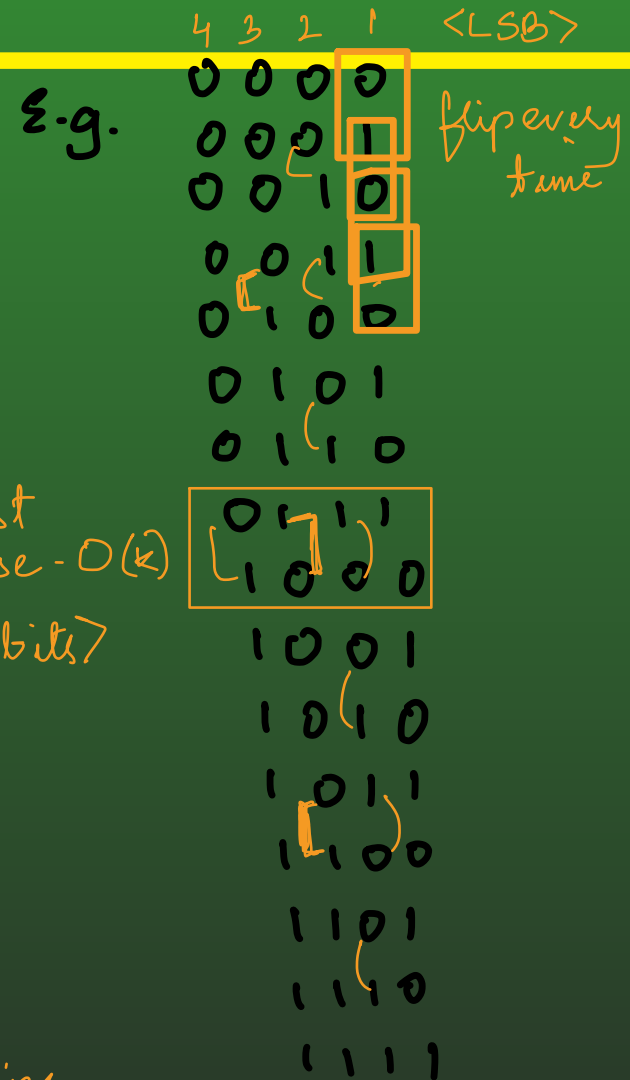
- Aggregate method
  - Total cost for  $n$  operations is  $g(n)$
  - Amortized cost for 1 operation is  $\frac{g(n)}{n}$
- Previous analysis of push/pop/Multipop used aggregate method

# 12-8: Aggregate Method

<Binary adder>

- Ripple counter, width  $k$ 
  - Examples on board
- How long does an increment take?

- (LSB)
- least sig. bit flip every time  
 $\therefore$  'n' flip for 'n' increment
  - 2<sup>nd</sup> LSB - every other time  
 flip from 1-2, 3-4, 5-6, 7-8,  
 $\therefore$   $n/2$  flip for 'n' increments
  - 3<sup>rd</sup> SB - flip half of time 2<sup>nd</sup> LSB flips =  $n/4$
  - 4<sup>th</sup> LSB: flips 1 / every other time 3<sup>rd</sup> LSB flips  
 $= n/8$



# 12-9: Aggregate Method

- Ripple counter, width  $k$
- How long does an increment take?
  - $O(k)$
  - But ...
    - Least sig. bit flips every time
    - 2nd least sig. bit flips every other time
    - 3rd least sig. bit flips every 4th time
    - $k$ th least sig. bit flips every  $2^k$ th time
- For  $n$  increments (if no overflow):

*∴ per inc. Amortized*

$$\text{cost} = \frac{2n}{n} \rightarrow \boxed{O(1)}$$

$$\sum_{i=0}^{\lg n} \left\lfloor \frac{n}{2^i} \right\rfloor < n * \sum_{i=0}^{\infty} \frac{1}{2^i} = \underline{2n}$$

$$1 + \frac{1}{2} + \frac{1}{4} + \dots$$

$$r = \frac{1}{2} \quad S_n = \frac{a}{1-r}$$
$$= \frac{1}{1-\frac{1}{2}} = 2.$$

## 12-10: Aggregate Method

---

- Ripple counter, width  $k$
- Worst case time for a sequence of  $n$  increment operations, if counter starts at 0:
  - $O(n)$
- Amortized cost for a single increment
  - $O(1)$

# 12-11: Accounting Method

---

- Accounting Method
  - Assign a cost for each operation
    - Called “amortized cost”
  - When amortized cost  $>$  actual cost, create a “credit” which can be used when actual cost  $>$  amortized cost
  - Must design costs so that all sequences of operations always leave a “positive account”

# 12-12: Accounting Method

---

	actual cost	amortized cost
Push	1	
Pop	1	
Multipop	$\min(k, s)$	

- What amortized costs should I give, so that any valid sequence of push/pop/multipop will never have a debt?

# 12-13: Accounting Method

---

	actual cost	amortized cost
Push	1	2
Pop	1	0
Multipop	$\min(k, s)$	0

# 12-14: Accounting Method

---

- Binary Counter
  - Actual Cost of setting a bit to 1 is 1
  - Actual Cost of setting a bit to 0 is 1
  - Actual Cost of an increment = # of bits flipped from 1 to 0 + 1
- What should our amortized costs be, and why?



# 12-15: Accounting Method

---

- Binary Counter
  - Amortized cost of setting a bit to 1 is: 2
  - Amortized cost of setting a bit to 0 is: ?
  - Amortized cost of an increment is: ?

## 12-16: Accounting Method

---

- Binary Counter
  - Amortized cost of setting a bit to 1 is: 2
  - Amortized cost of setting a bit to 0 is: 0 (!)
  - Amortized cost of an increment is: 2

For  $n$  increments, the total amortized cost is  $O(n)$ , which is also a bound on the actual cost

## 12-17: Potential Method

---

- Define a “potential” for data structures that your algorithm uses
  - Kind of like potential energy
- When the amortized cost is greater than the actual cost, increase the potential of the data structure
- When the amortize cost is less than the actual cost, decrease the potential of the data structure
  - Potential can never be negative

# 12-18: Potential Method

- $\Phi(D)$  = potential of the data structure
- Amortized cost of operation  $c_i$  is  $\text{am}(c_i)$ 
  - $\text{am}(c_i) = c_i + (\Phi(D_i) - \Phi(D_{i-1}))$
- Total amortized cost for a sequence of  $n$  operations:

$$\begin{aligned}\sum_{i=1}^n \text{am}(c_i) &= \sum_{i=1}^n (c_i \Phi(D_i) - \Phi(D_{i-1})) \\ &= \left( \sum_{i=1}^n c_i \right) + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

- As long as the potential starts at 0 and never goes

## 12-19: Potential Method

---

- The potential function is on the *Data Structure*, not the operations
- Don't talk about the potential of a push or a pop
- Instead, talk about the potential of the stack
  - Define a potential function on the data structure
  - Use the potential function and actual cost to determine amortized cost

## 12-20: Potential Method

---

- Potential Method Examples:
  - Stack, with push/pop/multipop
    - What should the potential be?
    - What are the resulting amortized costs?

## 12-21: Potential Method

---

- Potential Method Examples:
  - Stack, with push/pop/multipop
    - Potential = # of elements in the stack
    - amortized cost = actual cost + change in potential
    - amortized cost of push =  $1 + 1 = 2$
    - amortized cost for pop =  $1 + (-1) = 0$
    - amortized cost for multipop =  $k + (-k) = 0$

## 12-22: Potential Method

---

- Potential Method Examples:
  - Binary Ripple-Carry Counter
    - What is the potential?
    - What are the resulting amortized costs



## 12-23: Potential Method

---

- Potential Method Examples:
  - Binary Ripple-Carry Counter
    - Potential = # of 1's in the counter
    - amortized cost = actual cost + change in potential
    - actual cost = # of bits flipped
    - Change in potential = # of bits flipped from 1 to 0 - # of bits flipped from 1 to 0
      - = - # of bits flipped, if counter reset to 0
      - = 2 - # of bits flipped, otherwise
    - Amortized cost  $\leq 2$

# 12-24: Dynamic Hash Tables

---

- Standard Hash Table
  - Insert/find in time  $O(1)$  (no delete for now)
  - Need to know an upper bound on the table size beforehand
- If we don't know the table size beforehand?
  - Pick a size to start with
  - If table fills, double the table size, and add everything from old table to new table
- What is the time for an insert if the table can grow?

# 12-25: Dynamic Hash Tables

---

- Standard Hash Table
  - Insert/find in time  $O(1)$  (no delete for now)
  - Need to know an upper bound on the table size beforehand
- If we don't know the table size beforehand?
  - Pick a size to start with
  - If table fills, double the table size, and add everything from old table to new table
- What is the time for an insert if the table can grow?
  - $O(n)$

## 12-26: Dynamic Hash Tables

- Any single insert into a Dynamic Hash Table can take time  $O(n)$
- What is the amortized cost for an insert?
  - Aggregate method

$$c_i = \begin{cases} i & \text{if } i - 1 \text{ is a power of 2} \\ 1 & \text{otherwise} \end{cases}$$

*when it doubles*

# 12-27: Dynamic Hash Tables

- Aggregate Method
  - Total cost for  $n$  inserts:

1 2 4 1 1 8 1 1 1 1 1 1 16

$$\begin{aligned}\sum_{i=1}^n c_i &\leq n + \sum_{i=1}^{\lg n} 2^i \\ &\leq n + 2n \\ &\leq 3n\end{aligned}$$

$\lg n$  : as it doubles almost half of times

rehashing prev stored elements

$\frac{2^{\lg n} - 1}{2 - 1} \approx \frac{2(n-1)}{1}$

- Amortized cost per insert is thus  $O(1)$

# 12-28: Dynamic Hash Tables

---

- Accounting Method
    - Amortized cost for insert = 3
      - Cost to insert the element
      - rehashing part* { • Cost to move element when the table is expanded next time
      - Cost to move one other element when the table is expanded next time
- (Examples)

# 12-29: Dynamic Hash Tables

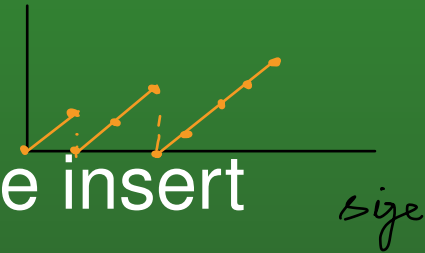
---

- Potential Method
  - Potential starts at 0, grows as we insert elements
  - When the table size increases, potential drops back to 0
    - Extra potential is used to grow the table

# 12-30: Dynamic Hash Tables

- Potential Method

- Potential starts at 0, grows as we insert elements
- When the table size increases, potential drops back to 0



size	#
1	1
2	2
4	3
4	4
8	5

- $\Phi(T) = 2 * num[T] - size[T]$

- $num[T]$  = number of elements in the table
- $size[T]$  = size of table

- Always positive (assuming we start with a table size of 0, when first element is added we go to a table size of 2 containing 1 element)



fill the new space



# 12-31: Dynamic Hash Tables

- Potential Method

- Amortized cost for an insert = actual cost + change in potential

- No doubling →
- If  $i$ th insert did not cause the table to grow:

$$\begin{aligned}
 am(c_i) &= 1 + (2 * num_i - size_i) - (2 * num_{i-1} - size_{i-1}) \\
 &= 1 + 2 * i - size_i - 2 * (i - 1) + size_{i-1} \\
 &= 3
 \end{aligned}$$

- Doubling →
- If  $i$ th insert did cause the table to grow:

$$\begin{aligned}
 am(c_i) &= 1 + \overset{\text{rehash}}{num_{i-1}} + \overset{\text{Potential def}^n}{(2 * num_i - size_i)} - (2 * num_{i-1} - size_{i-1}) \\
 &= 1 + (i - 1) + (2 * i - 2 * (i - 1)) - (2 * (i - 1) - (i - 1)) \\
 &= \left( 3 \right) \xrightarrow{\text{rehash all values}} 1 + 2 * i - 2 * (i - 1) + (i - 1) - 2 * (i - 1) + (i - 1) \\
 &= \underline{\underline{3}}
 \end{aligned}$$

*Note: In the final step, the terms are rearranged with a  $\times 2$  multiplier on the last term, and the final result is 3.*

## 12-32: Dynamic Hash Tables

---

- Add in deletes
- Want to keep the table from being too big
- Shrink the table when it gets too large (freeing space)
- First try:
  - When table gets full, double the size of the table, copying elements
  - When table gets less than half full, cut the size of the table in half, copying elements
- Will this still give us  $O(1)$  amortized cost for an insert/delete?

## 12-33: **Dynamic Hash Tables**

---

- Consider a table that is full
- What happens when we do the following operations:
  - Insert, Delete, Delete, Insert, Insert Delete, Delete, . . .

## 12-34: Dynamic Hash Tables

---

- Consider a table that is full
- What happens when we do the following operations:
  - Insert, Delete, Delete, Insert, Insert Delete, Delete, . . .
  - Every other operation takes time  $O(n)$ !
  - Amortized cost per operation is  $O(n)$ , not  $O(1)$ !
- What can we do?

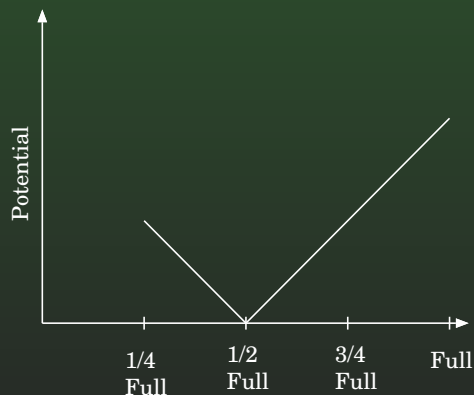
## 12-35: **Dynamic Hash Tables**

---

- When table gets full, double the size of the table, copying elements
- When table gets less than  $1/4$  full, halve the size of the table, copying elements

# 12-36: Dynamic Hash Tables

- Potential Function  $\Phi$ :
  - 0 when list is exactly half full
  - Increase as # of elements in the list increases, so that the potential = # of elements in the list when the list is full
  - Increase as # of elements decreases (below 1/2 full) so that the potential = # of elements in the list when the list is 1/4 full



## 12-37: Dynamic Hash Tables

---

- Potential Function  $\Phi$ :
  - $\alpha$  = load of the table: Size of table / # of elements

$$\Phi(T) = \begin{cases} 2 * num[T] - size[T] & \text{if } \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \text{if } \alpha(T) < 1/2 \end{cases}$$

# 12-38: Dynamic Hash Tables

---

- Amortized cost for insert:
  - Amortized cost = actual cost + growth in potential
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$
  - Several cases:
    - $1/4 < \alpha < 1/2$
    - $1/2 \leq \alpha < 1$
    - $\alpha = 1$



# 12-39: Dynamic Hash Tables

---

- Amortized cost for insert,  $1/2 \leq \alpha < 1$ 
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

$$\begin{aligned} am(c_i) &= 1 + (2 * num_i - size_i) - (2 * num_{i-1} - size_{i-1}) \\ &= 1 + 2 * i - size_i - 2 * (i - 1) + size_i \\ &= 3 \end{aligned}$$

# 12-40: Dynamic Hash Tables

---

- Amortized cost for insert,  $\alpha = 1$ 
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

$$\begin{aligned} am(c_i) &= 1 + num_{i-1} + (2 * num_i - size_i) - (2 * num_{i-1} - size_{i-1}) \\ &= 1 + (i - 1) + (2 * i - 2 * (i - 1)) - (2 * (i - 1) - (i - 1)) \\ &= 3 \end{aligned}$$

# 12-41: Dynamic Hash Tables

---

- Amortized cost for insert,  $1/4 < \alpha < 1/2$ 
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

$$\begin{aligned} am(c_i) &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (size_i/2 - size_i/2) - num_i + (num_i) - 1 \\ &= 0 \end{aligned}$$

# 12-42: Dynamic Hash Tables

---

- Amortized cost for delete:
  - Amortized cost = actual cost + growth in potential
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$
  - Several cases:
    - $1/4 < \alpha < 1/2$
    - $1/2 \leq \alpha \leq 1$
    - $\alpha = 1/4$

# 12-43: Dynamic Hash Tables

---

- Amortized cost for delete,  $1/2 \leq \alpha \leq 1$ 
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

$$\begin{aligned} am(c_i) &= 1 + (2 * num_i - size_i) - (2 * num_{i-1} - size_{i-1}) \\ &= 1 + 2 * num_i - 2 * (num_i + 1) + size_i - size_i \\ &= -1 \end{aligned}$$

# 12-44: Dynamic Hash Tables

---

- Amortized cost for delete,  $1/4 < \alpha < 1/2$ 
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

$$\begin{aligned} am(c_i) &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (size_i/2 - size_i/2) + num_{i-1} - num_i \\ &= 2 \end{aligned}$$

# 12-45: Dynamic Hash Tables

---

- Amortized cost for delete,  $\alpha = 1/4$ 
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

$$size_i/2 = size_{i-1}/4 = num_{i-1} = num_i + 1$$

$$\begin{aligned} am(c_i) &= 1 + num_i + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + num_i + (num_i + 1 - num_i) - ((2 * num_i + 2) \\ &\quad - (num_i + 1)) \\ &= 1 \end{aligned}$$