

Amortized Analysis and Dynamic Arrays

Amortized Analysis



- Cost per-operation over a sequence of operations
 - Can take the total work for N operations and divide by N , i.e. average cost per operation.
 - Different than average case
 - We don't average over the possible inputs
 - Still consider worst case input

Example: Incrementing a Binary Counter



A[m]	A[m-1]	...	A[3]	A[2]	A[1]	A[0]	cost
0	0		0	0	0	0	
0	0		0	0	0	1	1
0	0		0	0	1	0	2
0	0		0	0	1	1	1
0	0		0	1	0	0	3
0	0		0	1	0	1	1
0	0		0	1	1	0	2
0	0		0	1	1	1	1
0	0		1	0	0	0	4
0	0		1	0	0	1	1
0	0		1	0	1	0	2
0	0		1	0	1	1	1
0	0		1	1	0	0	3

- Cost is the number of bits that change to go from one value to the next

Binary Counter Analysis



A[m]	A[m-1]	...	A[3]	A[2]	A[1]	A[0]	cost
0	0		0	0	0	0	
0	0		0	0	0	1	1
0	0		0	0	1	0	2
0	0		0	0	1	1	1
0	0		0	1	0	0	3
0	0		0	1	0	1	1
0	0		0	1	1	0	2
0	0		0	1	1	1	1
0	0		1	0	0	0	4

- Over a sequence of N increments, how many times does:
 - $A[0]$ change?
 - ✓ N times
 - $A[1]$ change
 - ✓ $N/2$ times
 - $A[2]$ change
 - ✓ $N/4$ times

Binary Counter Analysis

A[m]	A[m-1]	...	A[3]	A[2]	A[1]	A[0]	cost
0	0		0	0	0	0	
0	0		0	0	0	1	1
0	0		0	0	1	0	2
0	0		0	0	1	1	1
0	0		0	1	0	0	3
0	0		0	1	0	1	1
0	0		0	1	1	0	2
0	0		0	1	1	1	1
0	0		1	0	0	0	4

- Therefore, the total number of changes is:

$$N + N/2 + N/4 + N/8 \dots + 2 + 1$$

$$= \sum_{i=0}^{\log N} \frac{N}{2^i}$$

$$\leq N \cdot \sum_{i=0}^{\infty} \frac{1}{2^i}$$

$$(\text{But } \sum_{i=0}^{\infty} \frac{1}{2^i} = 2)$$

$$\leq N \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2N$$

- So the total work is $2N$

Binary Counter Banker's Method



A[m]	A[m-1]	...	A[3]	A[2]	A[1]	A[0]	amortized cost
0	0		0	0	0	0	
0	0		0	0	0	1*	2
0	0		0	0	1*	0	2
0	0		0	0	1*	1*	2
0	0		0	1*	0	0	2
0	0		0	1*	0	1*	2
0	0		0	1*	1*	0	2
0	0		0	1*	1*	1*	2
0	0		1*	0	0	0	2

Different idea, assume the computer runs on tokens:

- Give each operation 2 tokens. Pay for the conversion of the 0 into a 1 with a token and store the remaining token there.
- All the other costs are turning 1s into 0s. Pay for those with the token stored there.

Dynamic Arrays



- Suppose we want to implement arrays without a fixed size limit.
 - insert operation adds a new element to the end of the array.
 - Why, when and how much?
 - When do we “re-size” the array?
 - When it’s full
 - How much space do we add?
 - Double
 - Why?

Dynamic Arrays



- What is the total cost of a sequence of $N=2^k$ insert operations?

N for the inserts +

$$2 + 4 + 8 + 16 + \dots 2^{k-1} + 2^k$$

$$= N + N/2 + N/4 + \dots 2 < 2N$$

So total cost is at most $3N$

Dynamic Arrays Bankers Method



- Suppose we give each insert 3 tokens.
 - 1st token pays for the insert itself.
 - Remaining 2 tokens stored with the item.
 - Use the tokens in the full array to pay for the copy.

5	3		
---	---	--	--

Insert 7 then insert 9

5	3	7 **	9 **
---	---	------	------

Re-size and copy

5	3	7	9				
---	---	---	---	--	--	--	--

Dynamic Arrays



5	3	7	9	2**	10**	4**	
---	---	---	---	-----	------	-----	--

Insert 1

5	3	7	9	2**	10**	4**	1**
---	---	---	---	-----	------	-----	-----

Insert 8, re-size first

5	3	7	9	2	10	4	1	8**							
---	---	---	---	---	----	---	---	-----	--	--	--	--	--	--	--

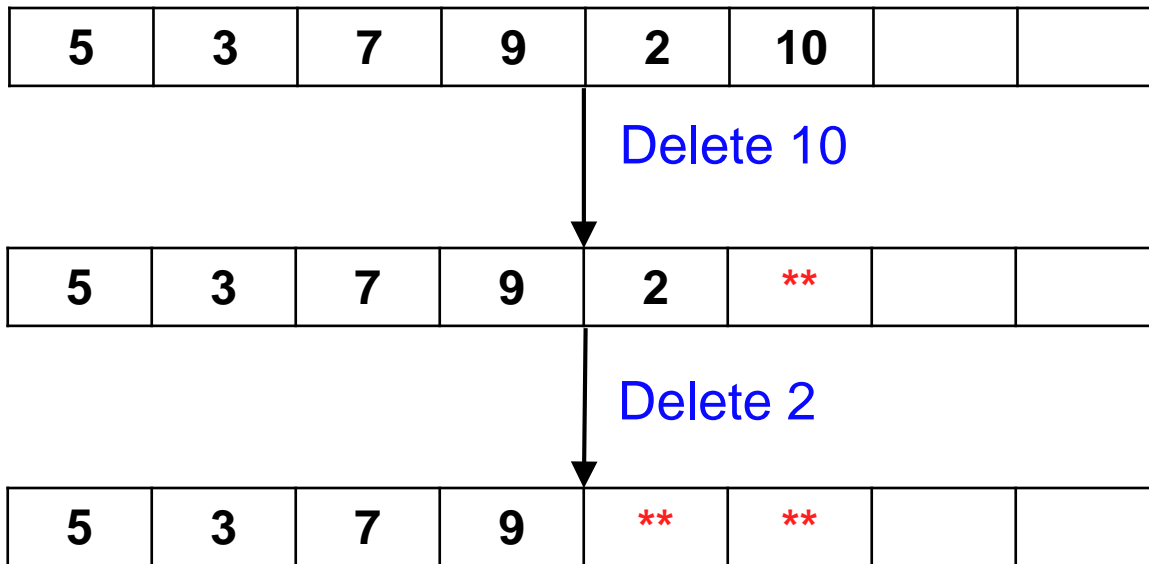
The 8 tokens stored at 2,10,4,and 1 “pay” for the copy.
The newly inserted 8 has its two tokens.

Dynamic Arrays Deletion



Suppose we want to add the operation deletion:
The delete operation gets 3 tokens.

- 1 pays for the delete itself
- 2 tokens go to the empty cell



Dynamic Arrays



- But there is no guarantee of tokens in the green
- 25% of the items must have 2 tokens each.
- Pays for a copy to an array of $\frac{1}{2}$ the current size.

Result:

5	3	7	9				
---	---	---	---	--	--	--	--

Dynamic Arrays



Continuing from this array:

5	3	7	9				
---	---	---	---	--	--	--	--

Remove 9:

5	3	7	**				
---	---	---	----	--	--	--	--

Remove 7:

5	3	**	**				
---	---	----	----	--	--	--	--

Shrink array to half its current size:

5	3		
---	---	--	--

Dynamic Arrays Summary



- Array always has at least 25% of the positions in use
- 3 Tokens for insert
- 3 Tokens for delete
- $O(1)$ Amortized time per operation.
- $O(N)$ worst case time for any single operation.