08 – Amortized Analysis

Amortization

- Consider a sequence $a_1, a_2, \dots , a_n$ of n operations performed on a data structure D

- T_i = execution time of a_i

- $T = T_1 + T_2 + \dots + T_n$ total execution time

- The execution time of a single operation can vary within a large range, e.g. in $1,\dots,n$, but the worst case does not occur for all operations of the sequence.

- Average execution time of an operation, i.e. $1/n \cdot \Sigma_{1 \leq i \leq n} T_i$, is small even though a single operation can have a high execution time.

Analysis of algorithms

- Best case (Too optimistic)
- Worst case (Sometimes very pessimistic)
- Average case (Input drawn according to a probability distribution. However, distribution might not be known, or input is not generated by a distribution.)
- Amortized worst case

What is the average cost of an operation in a worst case sequence of operations?

© S. Albers

Amortization

Idea:

- Pay more for inexpensive operations
- Use the credit to cover the cost of expensive operations

Three methods:

1. Aggregate method
2. Accounting method
3. Potential method

1. Aggregate method: binary counter

Incrementing a binary counter: determine the bit flip cost

Operation	Counter value	Cost
	00000	
1	00001	1
2	00010	2
3	00011	1
4	00100	3
5	00101	1
6	00110	2
7	00111	1
8	01000	4
9	01001	1
10	01010	2
11	01011	1
12	01100	3
13	01101	1

Binary counter

In gneral:

For any *n*, estimate the total time of *n* increment operations.

Show:

Amortized cost of an operation is upper bounded by *c*.

→ Total cost is upper bounded by *cn*.

2. The accounting method

Observation:

In each operation exactly one 0 flips to 1.

Idea:

Pay two cost units for flipping a 0 to a 1

→ each 1 has one cost unit deposited in the banking account

The accounting method

Operation	**Counter value**
	0 0 0 0 0
1	0 0 0 0 1
2	0 0 0 1 0
3	0 0 0 1 1
4	0 0 1 0 0
5	0 0 1 0 1
6	0 0 1 1 0
7	0 0 1 1 1
8	0 1 0 0 0
9	0 1 0 0 1
10	0 1 0 1 0

The accounting method

Operation	**Counter value**	**Actual cost**	**Payment**	**Credit**
	0 0 0 0 0			
1	0 0 0 0 1	1	2	1
2	0 0 0 1 0	2	0+2	1
3	0 0 0 1 1	1	2	2
4	0 0 1 0 0	3	0+0+2	1
5	0 0 1 0 1	1	2	2
6	0 0 1 1 0	2	0+2	2
7	0 0 1 1 1	1	2	3
8	0 1 0 0 0	4	0+0+0+2	1
9	0 1 0 0 1	1	2	2
10	0 1 0 1 0	1	0+2	2

We only pay from the credit when flipping a 1 to a 0.

3. The potential method

Potential function Φ

Data structure $D \rightarrow \Phi(D)$

t_i = actual cost of the i-th operation

Φ_i = potential after execution of the i-th operation (= $\Phi(D_i)$)

a_i = amortized cost of the i-th operation

Definition:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

Example: binary counter

D_i = counter value after the i-th operation

$\Phi_i = \Phi(D_i)$ = # of 1‘s in D_i

i–th operation	# of 1‘s
D_{i-1}: .....0/1.....$0\underbrace{1.....1}_{b_i}$	B_{i-1}
$\Downarrow$	
D_i : .....0/1.....10.....0	$B_i = B_{i-1} - b_i + 1$

t_i = **actual bit flip cost of operation *i*** $= b_i+1$

$a_i = t_i + \Phi(D_i) - \Phi(D_{i-1})$

Binary counter

t_i = actual bit flip cost of operation *i*
a_i = amortized bit flip cost of operation *i*

$$\begin{aligned} a_i &= \left(b_i + 1\right) + \left(B_{i-1} - b_i + 1\right) - B_{i-1} \\ &= 2 \end{aligned}$$

$$\Rightarrow \sum_{i=1}^{n} a_i \leq 2n$$

$$\Rightarrow \sum_{i=1}^{n} a_i = \sum_{i=1}^{n} (t_i + \Phi(D_i) - \Phi(D_{i-1})) \leq 2n$$

$$\Rightarrow \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} a_i - \Phi(D_n) + \Phi(D_0) \leq 2n - \Phi(D_n) + \Phi(D_0) \leq 2n$$

Dynamic tables

Problem:

Maintain a table supporting the operations insert and delete such that

- the table size can be adjusted dynamically to the number of items
- the used space in the table is always at least a constant fraction of the total space
- the total cost of a sequence of n operations (insert or delete) is $O(n)$.

Applications: hash table, heap, stack, etc.

Load factor α_T: number of items stored in the table divided by the size of the table

Dynamic tables

Dynamic table *T*

```
size[T];                    // size of the table
num[T];                     // number of items
```

Initially there is an empty table with 1 slot, i.e.

size[*T*] = 1 and num[*T*] = 0.

Implementation of ‘insert’

```
insert ($T$, $x$)
if num[$T$] = size[$T$] then
    allocate new table $T'$ with $2\cdot$size[$T$] slots;
    insert all items in $T$ into $T'$;
    $T := T'$;  free table $T'$;
    size[$T$] := $2\cdot$size[$T$];
endif;
insert $x$ into $T$;
num[$T$] := num[$T$]+1;
```

Cost of *n* insertions into an initially empty table

t_i = cost of the *i*-th insert operation

Worst case:

$t_i = 1$	if the table is not full prior to operation *i*
$t_i = (i - 1) + 1$	if the table is full prior to operation *i*.

Thus *n* insertions incur a total cost of at most

$$\sum_{i=1}^{n} i = \Theta\left(n^2\right).$$

Amortized worst case:

Aggregate method, accounting method, potential method

Potential method

T table with

- k = num[T] items
- s = size[T] size

Potential function

$$\Phi(T) = 2\,k - s$$

Potential method

Properties

- $\Phi_0 = \Phi(T_0) = \Phi$ (empty table) = -1
- Immediately before a table expansion we have $k = s$, thus $\Phi(T) = k = s$.
- Immediately after a table expansion we have $k = s/2$, thus $\Phi(T) = 2k - s = 0$.
- For all $i \geq 1$: $\Phi_i = \Phi(T_i) > 0$
  Since $\Phi_n - \Phi_0 \geq 0$

$$\sum_{i=1}^{n} t_i \leq \sum_{i=1}^{n} a_i .$$

Amortized cost a_i of the *i*-th insertion

k_i = # items stored in *T* after the *i*-th operation

s_i = table size of *T* after the *i*-th operation

Case 1: *i*-th operation does not trigger an expansion

$k_i = k_{i-1} + 1,\ s_i = s_{i-1}$

$$\begin{aligned} a_i &= 1 + (2k_i - s_i) - (2k_{i-1} - s_{i-1}) \\ &= 1 + 2(k_i - k_{i-1}) \\ &= 3 \end{aligned}$$

Case 2: *i*-th operation does trigger an expansion

$k_i = k_{i-1} + 1, s_i = 2s_{i-1}$

$$\begin{aligned} a_i &= k_{i-1} + 1 + (2k_i - s_i) - (2k_{i-1} – s_{i-1}) \\ &= 2(k_{i-1} + 1) - k_{i-1} + 1 - 2s_{i-1} + s_{i-1} \\ &= k_{i-1} + 3 - s_{i-1} \\ &= 3 \end{aligned}$$

Inserting and deleting items

Now: Contract the table whenever the load becomes too small.

Goal:

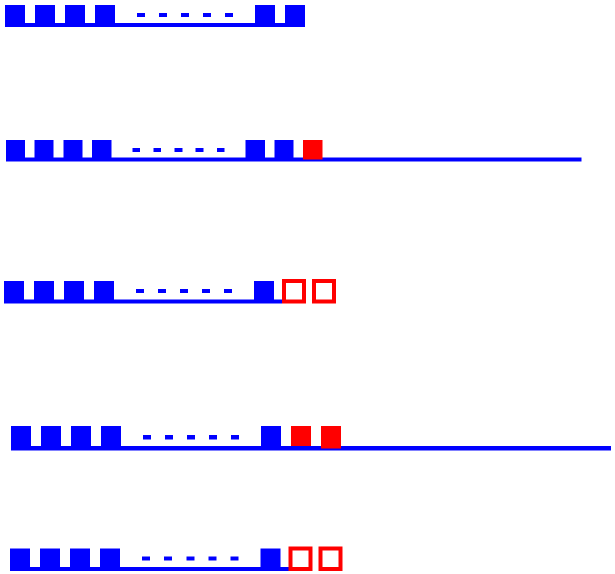
(1) The load factor is bounded from below by a constant.

(2) The amortized cost of a table operation is constant.

First approach

- Expansion: as before
- Contraction: Halve the table size when a deletion would cause the table to become less than half full.

„Bad“ sequence of table operations



	Cost
n/2 'insert' op. (table is full)	$n – 1$ $\Sigma a_i\text{-}\Phi_{n/2}+\Phi_0 =$ $3n/2\text{-}n/2\text{-}1$
I: expansion	$n/2 + 1$
D, D: contraction	$n/2 + 1$
I, I : expansion	$n/2 + 1$
D, D: contraction	

Total cost of the sequence of *n* operations, with $n \geq 2$: $I_{n/2}, I,D,D,I,I,D,D$

$$3n/2 + \lfloor 1/2 \cdot (n/2 - 1) \rfloor (n/2 + 1) > n^2/8$$

Second approach

Expansion: Double the table size when an item is inserted into a full table.

Contraction: Halve the table size when a deletion causes the table to become less than ¼ full.

Property: At any time the table is at least ¼ full, i.e.

$$\tfrac{1}{4} \leq \alpha(T) \leq 1$$

What is the cost of a sequence of table operations?

Analysis of ‘insert’ and ‘delete’ operations

k = num[T], s = size[T], $\alpha = k/s$

Potential function Φ

$$\boxed{\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}}$$

Analysis of ‘insert’ and ‘delete’ operations

$$\boxed{\Phi(T) = \begin{cases} 2k - s, \ \text{if } \alpha \geq 1/2 \\ s/2 - k, \ \text{if } \alpha < 1/2 \end{cases}}$$

Immediately after a table expansion or contraction:

$s = 2k$, thus $\Phi(T) = 0$

Analysis of an ‘insert’ operation

i-th operation: $k_i = k_{i-1} + 1$

Case 1: $\alpha_{i-1} \geq ½$

Potential function before and after the operation is $\Phi(T) = 2k-s$. We have already proved that the amortized cost is equal to 3.

Case 2: $\alpha_{i-1} < ½$

Case 2.1: $\alpha_i < ½$

Case 2.2: $\alpha_i \geq ½$

Analysis of an ‘insert’ operation

Case 2.1: $\alpha_{i-1} < ½$, $\alpha_i < ½$ no expansion

Potential function Φ

$$\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}$$

$$a_i = 1 + (s_i/2 - k_i) - (s_{i-1}/2 - k_{i-1})$$
$$= 1 - (k_{i-1} + 1) + k_{i-1}$$
$$= 0$$

Analysis of an 'insert' operation

Case 2.2: $\alpha_{i-1} < ½$, $\alpha_i \geq ½$ no expansion

Potential function Φ

$$\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}$$

$$\begin{aligned} a_i &= 1 + (2k_i - s_i) - (s_{i-1}/2 - k_{i-1}) \\ &= 1 + 2(k_{i-1} + 1) - 3s_{i-1}/2 + k_{i-1} \\ &= 3 + 3(k_{i-1} - s_{i-1}/2) \\ &< 3 \end{aligned}$$

The last inequality holds because $k_{i-1} / s_{i-1} < ½$.

Analysis of a ‘delete’ operation

$k_i = k_{i-1} - 1$

Case 1: $\alpha_{i-1} < ½$

Case 1.1: deletion does not trigger a contraction

$s_i = s_{i-1}$

Potential function Φ

$$\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}$$

$$\begin{aligned} a_i &= 1 + (s_i/2 - k_i) - (s_{i-1}/2 - k_{i-1}) \\ &= 1 - (k_{i-1} - 1) + k_{i-1} \\ &= 2 \end{aligned}$$

Analysis of a ‘delete’ operation

$k_i = k_{i-1} - 1$

Case 1: $\alpha_{i-1} < ½$

Case 1.2: $\alpha_{i-1} < ½$ deletion does trigger a contraction

$s_i = s_{i-1}/2 \quad k_{i-1} = s_{i-1}/4$

Potential function Φ

$$\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}$$

$$\begin{aligned} a_i &= 1 + k_{i-1} + (s_i/2 - k_i) - (s_{i-1}/2 - k_{i-1}) \\ &= 1 + k_{i-1} + s_{i-1}/4 - (k_{i-1} - 1) - s_{i-1}/2 + k_{i-1} \\ &= 2 - s_{i-1}/4 + k_{i-1} \\ &= 2 \end{aligned}$$

Analysis of a ‘delete’ operation

Case 2: $\alpha_{i-1} \geq ½$

A contraction only occurs if $s_{i-1} = 2$ and $k_{i-1} = 1$.

In this case $a_i = 1 + s_i/2 – k_i – (2\ k_{i-1} - s_{i-1})$
$= 1 + 1/2 - 2 + 2 < 2$.

Therefore, in the following, we may assume that no contraction occurs.

Analysis of a ‘delete’ operation

Case 2: $\alpha_{i-1} \geq ½$ no contraction

$s_i = s_{i-1} \quad k_i = k_{i-1} - 1$

Case 2.1: $\alpha_i \geq ½$

Potential function Φ

$$\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}$$

$$\begin{aligned} a_i &= 1 + (2k_i - s_i) - (2k_{i-1} - s_{i-1}) \\ &= 1 + 2(k_{i-1} - 1) - 2k_{i-1} \\ &< 0 \end{aligned}$$

Analysis of a ‘delete’ operation

Case 2: $\alpha_{i-1} \geq ½$ no contraction

$$s_i = s_{i-1} \quad k_i = k_{i-1} - 1$$

Case 2.2: $\alpha_i < ½$

Potential function Φ

$$\Phi(T) = \begin{cases} 2k - s, & \text{if } \alpha \geq 1/2 \\ s/2 - k, & \text{if } \alpha < 1/2 \end{cases}$$

$$\begin{aligned} a_i &= 1 + (s_i/2 - k_i) - (2k_{i-1} - s_{i-1}) \\ &= 1 + s_{i-1}/2 - k_{i-1} + 1 - 2k_{i-1} + s_{i-1} \\ &= 2 + 3(s_{i-1}/2 - k_{i-1}) \\ &\leq 2 \end{aligned}$$

The last inequality holds because $k_{i-1} \geq s_{i-1}/2$.