# Algorithms Theory

# 06 – Amortized Analysis

Prof. Dr. S. Albers

# Amortization

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

- $T_i$ = execution time of $a_i$

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

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

- Average execution time of an operation is small, even though a single operation can have a high execution time.

# Analysis of algorithms

- Best case
- Worst case
- Average case
- Amortized worst case

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

# 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 | |
| 8 | 01000 | |
| 9 | 01001 | |
| 10 | 01010 | |
| 11 | 01011 | |
| 12 | 01100 | |
| 13 | 01101 | |

# 2. The accounting method

**Observation:**

In each step 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 |

# 3. The potential method

**Potential function $\phi$**

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

$t_i$ = actual cost of the $i$-th operation

$\phi_i$ = 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.....01.....1 | $B_{i-1}$ |
| ↓ | |
| $D_i$ : .....0/1.....10.....0 | $B_i = B_{i-1} - b_i + 1$ |

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

# Binary counter

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

$$a_i = (b_i + 1) + (B_{i-1} - b_i + 1) - B_{i-1}$$
$$= 2$$
$$\Rightarrow \sum t_i \le 2n$$

# 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 $\alpha_T$: number of items stored in the table divided by the size of the table

# Implementation of ‘insert’

```
class dynamic table {

    int [] table;

    int size;                  // size of the table
    int num;                   // number of items

dynamicTable() {               // initialization of an empty table
    table = new int [1];
    size = 1;
    num = 0;
    }
```

# Implementation of ‘insert’

```
insert ( int x) {
    if (num == size ) {
        newTable = new int [2*size];
        for (i = 0; i < size; i++)
            insert table[i] into newTable;
        table = newTable;
        size = 2*size;
    }
    insert x into table;
    num = num + 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

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

**Amortized worst case:**

Aggregate method, accounting method, potential method

# Potential method

$T$ table with

- $k = T.num$ items
- $s = T.size$ size

**Potential function**

$$\phi(T) = 2k - s$$

# 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 $\phi_n - \phi_0 \geq 0$,

$$\sum t_i \leq \sum a_i$$

# 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

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

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

$= 3$

# 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.

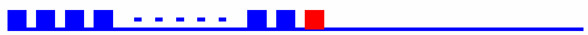
# „Bad“ sequence of table operations

Cost

*n/2* ‘insert’ op.
(table is full)

3 *n/2*

*I:* expansion

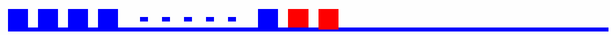


*n/2* + 1

*D, D*: contraction

*n/2* + 1

*I, I* : expansion



*n/2* + 1

*D, D*: contraction

Total cost of the sequence of operations:
$I_{n/2}$, *I,D,D,I,I,D,D*,... of length *n* is

# 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.

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

What is the cost of a sequence of table operations?

# Analysis of ‘insert’ and ‘delete’ operations

*k = T.num,  s = T.size,* $\alpha$ *= k/s*

**Potential function $\phi$**

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

# Analysis of ‘insert’ and ‘delete’ operations

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

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: $\alpha_{i-1} \geq ½$

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 $\phi$**

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

# Analysis of an 'insert' operation

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

**Potential function $\phi$**

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

# 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 $\phi$**

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

# 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 $\phi$**

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

# 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 $\phi$**

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

# 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 $\phi$**

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