# Amortised Analysis

## Algorithms and Complexity Theory

Matei Popovici[1]

[1]POLITEHNICA University of Bucharest
Computer Science and Engineering Department, Bucharest, Romania

October 26, 2012

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position
  - **delete**: removes an element and frees a certain position

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position
  - **delete**: removes an element and frees a certain position
- When the table is **full**, it's capacity **doubles**

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position
  - **delete**: removes an element and frees a certain position
- When the table is **full**, it's capacity **doubles**

What is the *cost* of performing a **sequence** $S$ of **insert/delete** operations on a table $T$ ?

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position
  - **delete**: removes an element and frees a certain position
- When the table is **full**, it's capacity **doubles**

What is the *cost* of performing a **sequence** $S$ of **insert/delete** operations on a table $T$ ?
Notation:

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position
  - **delete**: removes an element and frees a certain position
- When the table is **full**, it's capacity **doubles**

What is the *cost* of performing a **sequence** $S$ of **insert/delete** operations on a table $T$ ?
Notation:

- $cost(S)$: *the cost of the sequence S of operations on T*

# Motivation

Let $T$ be a **table**, with **capacity** $n$.

- Possible operations on $V$:
  - **insert**: adds an element at the next free position
  - **delete**: removes an element and frees a certain position
- When the table is **full**, it's capacity **doubles**

What is the *cost* of performing a **sequence** $S$ of **insert/delete** operations on a table $T$ ?
Notation:

- $cost(S)$: *the cost of the sequence S of operations on T*
- $c_i^S$ (or just $c_i$ when $S$ is understood from the context): *the cost of the i-th operation in S*

# Insertion in $T$ (case 1)

Assume $T$ is in the following state:

# Insertion in $T$ (case 1)

Assume $T$ is in the following state:

# Insertion in $T$ (case 1)

Assume $T$ is in the following state:

Assume operation $i$ is **insertion**. Then, $T$ becomes:

# Insertion in $T$ (case 1)

Assume $T$ is in the following state:

Assume operation $i$ is **insertion**. Then, $T$ becomes:

# Insertion in $T$ (case 1)

Assume $T$ is in the following state:

Assume operation $i$ is **insertion**. Then, $T$ becomes:

The cost of operation $i$ is: $c_i = 1$

# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

What happens on **insertion**:

# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

What happens on **insertion**:

# Insertion in $T$ (case 2)

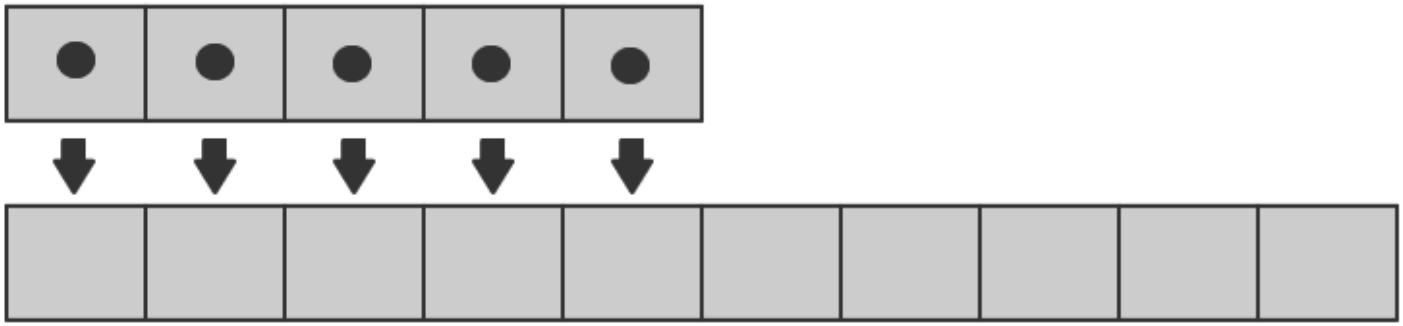
Assume $T$ is in the following state:

What happens on **insertion**:

# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

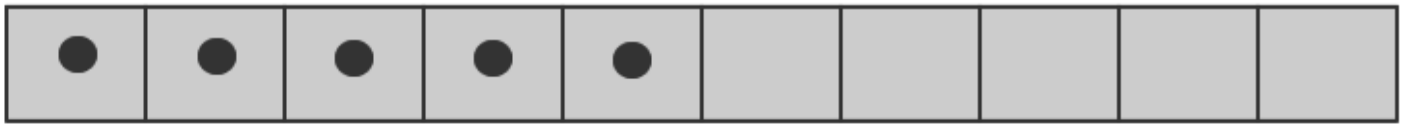
What happens on **insertion**:



# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

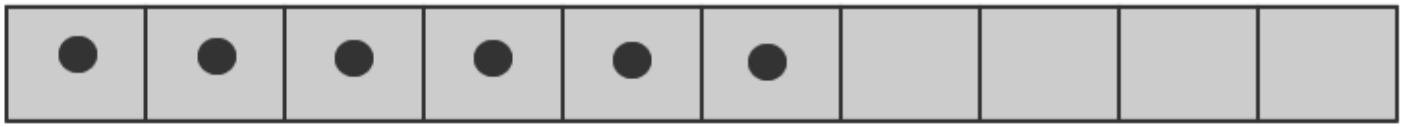
What happens on **insertion**:



# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

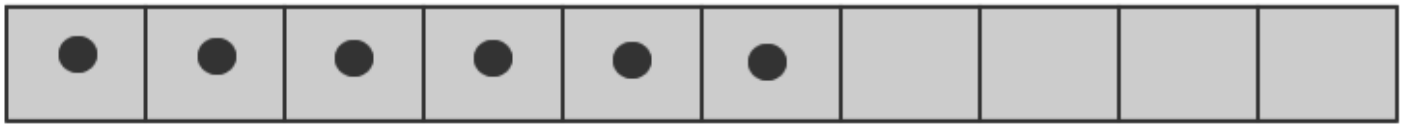
After **insertion** $T$ becomes:



# Insertion in $T$ (case 2)

Assume $T$ is in the following state:

After **insertion** $T$ becomes:



The cost of operation $i$ is: $c_i = k + 1$ where $k$ is the number of elements in $T$

# Deletion in $T$

Assume $T$ is in the following state:

# Deletion in $T$

Assume $T$ is in the following state:

# Deletion in $T$

Assume $T$ is in the following state:

Assume operation $i$ is **deletion**. Then, $T$ becomes:

# Deletion in $T$

Assume $T$ is in the following state:

Assume operation $i$ is **deletion**. Then, $T$ becomes:

# Deletion in $T$

Assume $T$ is in the following state:

Assume operation $i$ is **deletion**. Then, $T$ becomes:

The cost of operation $i$ is: $c_i = 1$

# More assumptions

Let $T$ be a table and $S$ be a sequence of operations:

- we consider $T$ to be of **capacity/size 0**, before running $S$.
- deleting an element from $T$ when it is empty has cost 1 and leaves $T$ unchanged.

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:
  - $T$ can contain at most $n$ insertions, therefore it cannot be of size bigger than $n$

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:
  - $T$ can contain at most $n$ insertions, therefore it cannot be of size bigger than $n$
  - The worst case occurs when $T$ is full, and the operation is **insertion**: all elements have to be copied

# Motivation

*What is the cost of performing a* ***sequence*** $S = op_1 \dots op_n$ *of* ***insert/delete*** *operations on a table* $T$ *?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:
  - $T$ can contain at most $n$ insertions, therefore it cannot be of size bigger than $n$
  - The worst case occurs when $T$ is full, and the operation is **insertion**: all elements have to be copied
- Hence, $cost(S) = n \cdot O(n) = O(n^2)$

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:
  - $T$ can contain at most $n$ insertions, therefore it cannot be of size bigger than $n$
  - The worst case occurs when $T$ is full, and the operation is **insertion**: all elements have to be copied
- Hence, $cost(S) = n \cdot O(n) = O(n^2)$

Code:

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:
  - $T$ can contain at most $n$ insertions, therefore it cannot be of size bigger than $n$
  - The worst case occurs when $T$ is full, and the operation is **insertion**: all elements have to be copied
- Hence, $cost(S) = n \cdot O(n) = O(n^2)$

Code:

- For $n = 50$, the cost hardly gets over 100, and never exceeds 113

# Motivation

*What is the cost of performing a **sequence** $S = op_1 \dots op_n$ of **insert/delete** operations on a table $T$ ?*

Reasoning I:

- In the worst case scenario, one operation costs $O(n)$:
  - $T$ can contain at most $n$ insertions, therefore it cannot be of size bigger than $n$
  - The worst case occurs when $T$ is full, and the operation is **insertion**: all elements have to be copied
- Hence, $cost(S) = n \cdot O(n) = O(n^2)$

Code:

- For $n = 50$, the cost hardly gets over 100, and never exceeds 113
- The approximation $O(n^2)$ is **poor** !

# Simplifying assumptions

Let $T$ be a table and $S$ be a sequence of operations on $T$. Then:

# Simplifying assumptions

Let $T$ be a table and $S$ be a sequence of operations on $T$. Then:

- $c_i \geq c_j$ for any **insertion** $i$ and **deletion** $j$ in $S$.

# Simplifying assumptions

Let $T$ be a table and $S$ be a sequence of operations on $T$. Then:

- $c_i \geq c_j$ for any **insertion** $i$ and **deletion** $j$ in $S$.
- $cost(S') \geq cost(S'')$ **for all sequences** $S'$ and $S''$ of the same size, such that $S'$ **contains only insertions**

# Simplifying assumptions

Let $T$ be a table and $S$ be a sequence of operations on $T$. Then:

- $c_i \geq c_j$ for any **insertion** $i$ and **deletion** $j$ in $S$.
- $cost(S') \geq cost(S'')$ **for all sequences** $S'$ and $S''$ of the same size, such that $S'$ **contains only insertions**

Let us assume that $S$ is a sequence with **only insertion operations**

# Simplifying assumptions

Let $T$ be a table and $S$ be a sequence of operations on $T$.
Then:

- $c_i \geq c_j$ for any **insertion** $i$ and **deletion** $j$ in $S$.
- $cost(S') \geq cost(S'')$ **for all sequences** $S'$ and $S''$ of the same size, such that $S'$ **contains only insertions**

Let us assume that $S$ is a sequence with **only insertion operations**
Then:

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

# The aggregate method

## The aggregate method

# The aggregate method

## The aggregate method

- we *aggregate* the cost of the sequence

# The aggregate method

**The aggregate method**

- we *aggregate* the cost of the sequence
- we **average** over the number of operations

# The aggregate method

**The aggregate method**

- we *aggregate* the cost of the sequence
- we **average** over the number of operations
- we obtain the **average cost** of an operation, in the worst case

# The aggregate method

**The aggregate method**

- we *aggregate* the cost of the sequence
- we **average** over the number of operations
- we obtain the **average cost** of an operation, in the worst case

Downsides:

# The aggregate method

**The aggregate method**

- we *aggregate* the cost of the sequence
- we **average** over the number of operations
- we obtain the **average cost** of an operation, in the worst case

Downsides:

- the average cost may be **imprecise** ($O(1)$)

# The aggregate method

**The aggregate method**

- we *aggregate* the cost of the sequence
- we **average** over the number of operations
- we obtain the **average cost** of an operation, in the worst case

Downsides:

- the average cost may be **imprecise** ($O(1)$)
- we do not distinguish between **types** of operations

# The accounting method

## The accounting method

# The accounting method

## The accounting method

- we assign *prices* for **each type** of operation. We call the price **amortized cost**

# The accounting method

## The accounting method

- we assign *prices* for **each type** of operation. We call the price **amortized cost**
- we require that $\sum_{i \in S} \hat{c}_i \geq \sum_{i \in S} c_i$ **for any sequence** $S$

# The accounting method

**The accounting method**

- we assign *prices* for **each type** of operation. We call the price **amortized cost**
- we require that $\sum_{i \in S} \hat{c}_i \geq \sum_{i \in S} c_i$ **for any sequence** $S$

Downsides:

# The accounting method

**The accounting method**

- we assign *prices* for **each type** of operation. We call the price **amortized cost**
- we require that $\sum_{i \in S} \hat{c}_i \geq \sum_{i \in S} c_i$ **for any sequence** $S$

Downsides:

- sometimes it is hard to find the proper amortised cost for each operation

# The potential method

## The potential method

# The potential method

## The potential method

- we assign a *potential* $\Phi(D_i)$ to any **state** $D_i$ of our data structure. $D_i$ is the state of our data structure after operation $i$ from sequence $S$ was performed.

# The potential method

## The potential method

- we assign a *potential* $\Phi(D_i)$ to any **state** $D_i$ of our data structure. $D_i$ is the state of our data structure after operation $i$ from sequence $S$ was performed.
- we require that $\Phi(D_n) - \Phi(D_0) \geq 0$ **for any sequence of** $n$ **operations**

# The potential method

## The potential method

- we assign a *potential* $\Phi(D_i)$ to any **state** $D_i$ of our data structure. $D_i$ is the state of our data structure after operation $i$ from sequence $S$ was performed.
- we require that $\Phi(D_n) - \Phi(D_0) \geq 0$ **for any sequence of** $n$ **operations**
- we **derive** amortized costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

# The potential method

**The potential method**

- we assign a *potential* $\Phi(D_i)$ to any **state** $D_i$ of our data structure. $D_i$ is the state of our data structure after operation $i$ from sequence $S$ was performed.
- we require that $\Phi(D_n) - \Phi(D_0) \geq 0$ **for any sequence of** $n$ **operations**
- we **derive** amortized costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Downsides:

# The potential method

**The potential method**

- we assign a *potential* $\Phi(D_i)$ to any **state** $D_i$ of our data structure. $D_i$ is the state of our data structure after operation $i$ from sequence $S$ was performed.
- we require that $\Phi(D_n) - \Phi(D_0) \geq 0$ **for any sequence of** $n$ **operations**
- we **derive** amortized costs: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Downsides:

- sometimes it is hard to find the proper potential function $\Phi$

# Bibliography I


Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson.
*Introduction to Algorithms*.
McGraw-Hill Higher Education, 2nd edition, 2001.