CS 255, Spring 2014, SJSU

Amortized Analysis

Fernando Lobo

Amortized Analysis

- Useful to analyze a sequence of operations on a data structure.
- Although a single operation might be expensive, the goal is to show that the average cost per operation is small.
  - Because there's a small number of expensive operations and lots of cheap operations.
- Amortized analysis allows us to obtain the worst case running time for any sequence of operations.

Various methods

There are several methods to do this kind of analysis. We shall see three:

1. Aggregate analysis
2. Accounting method
3. Potencial method

Example: Dynamic tables

- Problem: We would like to have a table (for example a hash table) but don't know beforehand how many elements the table might hold.
- The table should be as small as possible (to save memory resources), but also sufficiently large so there's no overflow.
- How shall we solve this problem?
  (For those familiar with Java, how to implement ArrayList?)

Example: Dynamic tables

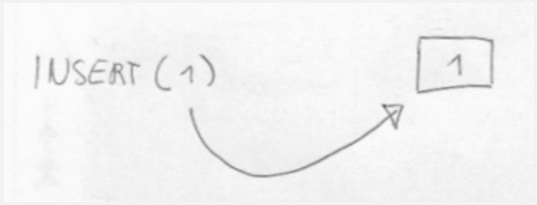
A possible solution:

- Start with a small table.
- If it becomes full, create a bigger table and copy the elements from the filled up table into the newly created one. Then delete the smaller table.

Example

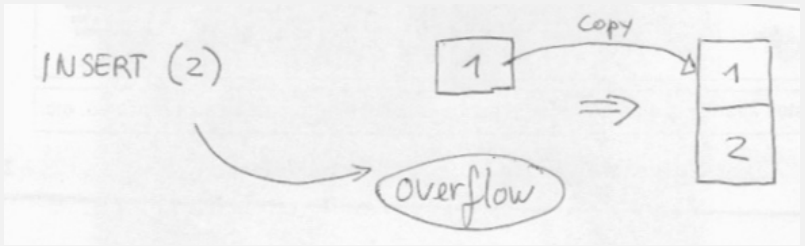
- Star with a table with a single position.
- Keep inserting elements.
- If table becomes full, duplicate its size.

INSERT(1)



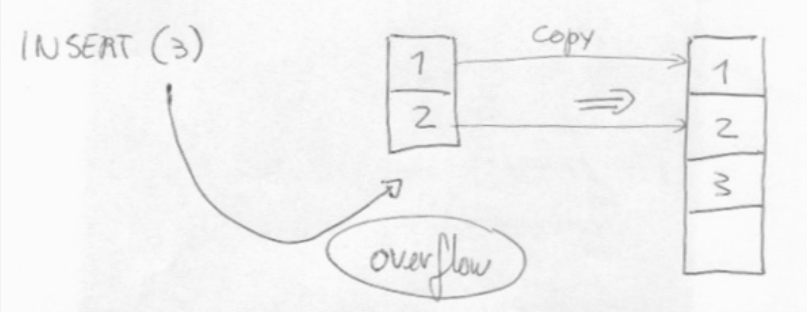
- Table becomes full. The next INSERT will give overflow.

INSERT(2)



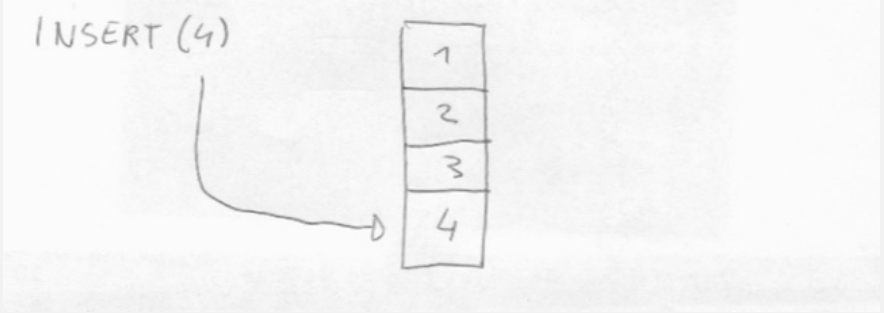
- Create a new table twice as big, copy the elements from the old table, insert the new element, and delete the old table.
- The new table now has size 2 and is holding 2 elements. It's full again.

INSERT(3)



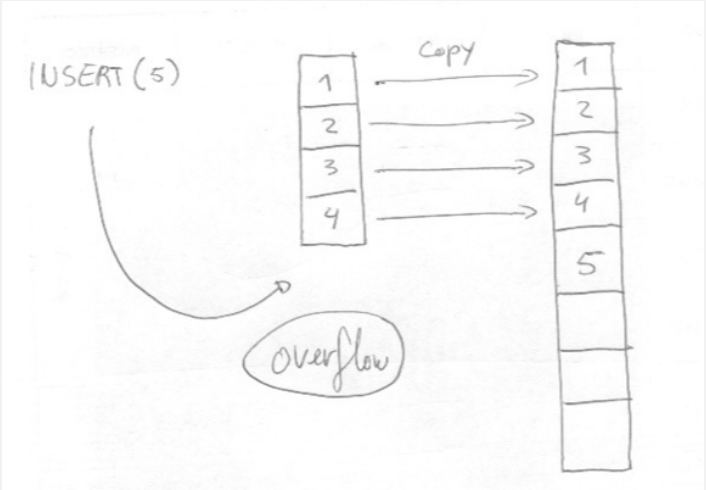
- Another overflow.
- Create a new table twice as big, copy the elements from the old table, insert the new element, and delete the old table.
- The new table now has size 4 and is holding 3 elements.

INSERT(4)



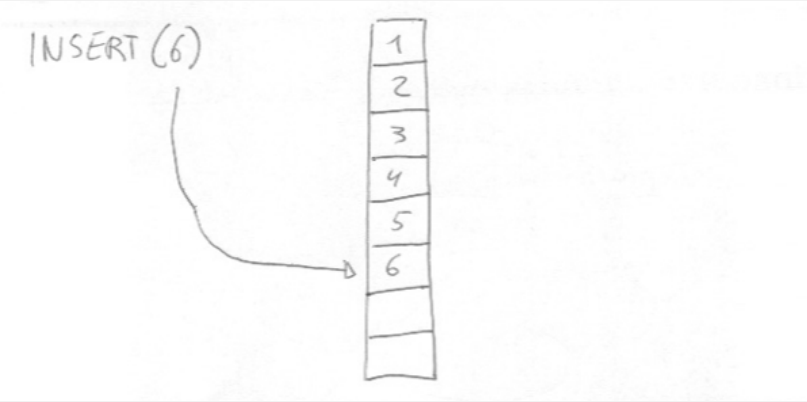
- No problem. Simply insert 4.
- The table is full again.

INSERT(5)



- Overflow. Create a new table twice as big, ...
- The new table now has size 8 and is holding 5 elements.

INSERT(6)



- No problem. Simply insert 6.
- Table has 6 elements.

INSERT(7), INSERT(8), ...

- You get the idea ...

Worst case analysis

- Consider a sequence of n INSERTs.
- In the worst case an INSERT operation takes $\Theta(n)$ time.
- Therefore, in the worst case n INSERTs take $n \cdot \Theta(n) = \Theta(n^2)$
- Correct? NO!
- As we shall see, in the worst case n INSERTs take only $\Theta(n)$ time.

A more careful analysis

Let c_i = cost of i-th INSERT.

$$c_i = \begin{cases} i & \text{, if } i-1 = 2^k \text{ , } k \in \mathbb{N} \\ 1 & \text{, otherwise} \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	...
$size_i$	1	2	4	4	8	8	8	8	16	16	...
c_i	1	2	3	1	5	1	1	1	9	1	...

$size_i$ is the table size immediately after the i-th INSERT.

Easier to see a pattern if we rewrite c_i

i	1	2	3	4	5	6	7	8	9	10	...
$size_i$	1	2	4	4	8	8	8	8	16	16	...
c_i	1	2	3	1	5	1	1	1	9	1	...
c_i	1	1+1	1+2	1	1+4	1	1	1	1+8	1	...

$$\text{Cost of } n \text{ INSERTs} = \sum_{i=1}^{n} c_i$$

$$= ?$$

Cost of n INSERTs

$$\begin{aligned}\text{Cost of } n \text{ INSERTs} &= \sum_{i=1}^{n} c_i \\ &= n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j \\ &\leq n + 2n \\ &= 3n \\ &= \Theta(n).\end{aligned}$$

- The average cost per operation is $\frac{\Theta(n)}{n} = \Theta(1)$.
- The average cost per operation is called amortized cost.

Amortized Analysis

- We call amortized analysis to any strategy that allow us to show that the average cost of an operation on a data structure is small, even though there can be individual operations that are quite expensive.

Amortized Analysis

Although we are averaging costs, this has nothing to do with an average case running time analysis.

- The analysis does not rely on input distributions and probabilities.
- Amortized analysis gives us the worst case running time on a sequence of operations, which is the same thing as the average performance of an operation in the worst case.

Aggregate analysis

- The method we've just seen is called aggregate analysis.
- Let's analyze the same problem using other methods:
  - Accounting method
  - Potencial method

Accounting method

- A banker's perspective.
- The idea is to charge a cost $\hat{c}_i$ (amortized cost) for each operation.
- 1 dollar corresponds to a unit of work (time).
- That amount can be consumed when the operation is executed. If something is left we can keep the extra dollars in the bank for future use.

Accounting method

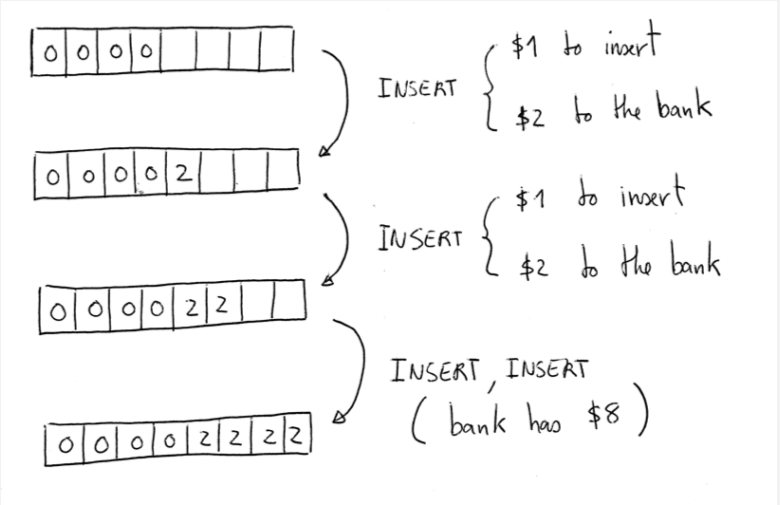
- The bank pays no interest.
- And gives no loans.
- The balance of the bank can never go negative.

$$\underbrace{\sum_{i=1}^{n} c_i}_{\text{actual cost}} \leq \underbrace{\sum_{i=1}^{n} \hat{c}_i}_{\text{amortized cost}}, \forall_n.$$

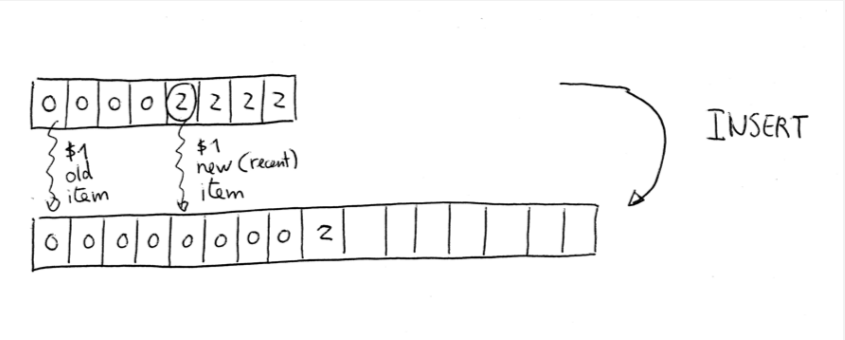
Dynamic table example

- Charge 3 dollars per INSERT.
- 1 dollar is for the INSERT operation itself.
- 2 dollars to keep in the bank so that we can use them later when we need to duplicate the table size.
  - 1 dollar to move a recent item.
  - 1 dollar to move an old item.
- Example:

Dynamic table example



Dynamic table example



- When we insert the 9th element we need to duplicate the table size.
- The 8 dollars in the bank allow us to copy the 8 elements in the old table into the new one.
- After that the bank is left with zero dollars.
- etc ...

Dynamic table example

- Invariant: Bank balance ≥ 0.
- Thus, the sum of the amortized costs is an upper limit for the sum of the real cost of the operations.

i	1	2	3	4	5	6	7	8	9	10	...
$size_i$	1	2	4	4	8	8	8	8	16	16	...
c_i	1	2	3	1	5	1	1	1	9	1	...
$\hat{c}_i$	2	3	3	3	3	3	3	3	3	3	...
$bank_i$	1	2	2	4	2	4	6	8	2	4	...

Dynamic table example

i	1	2	3	4	5	6	7	8	9	10	...
$size_i$	1	2	4	4	8	8	8	8	16	16	...
c_i	1	2	3	1	5	1	1	1	9	1	...
$\hat{c}_i$	2	3	3	3	3	3	3	3	3	3	...
$bank_i$	1	2	2	4	2	4	6	8	2	4	...

- The first operation only needs 2 dollars. No big deal if we give 3 dollars to the bank.

Potencial method

- Gives us the physics perspective.
- The idea is to look at the amount that is in the bank account as if is was energy.
- The energy is associated to the data structure as a whole.

Potencial method

- Start with a data structure D_0.
- The i-th operation transforms D_{i-1} into D_i.
- Actual cost of the i-th operation is c_i.
- Define a potential function, $\Phi : \{D_i\} \to \mathbb{R}$, such that:
  - $\Phi(D_0) = 0$
  - $\Phi(D_i) \geq 0 \quad , \forall_i$

Potencial method

- The amortized cost $\hat{c}_i$ is defined by,

$$\hat{c}_i \quad = \quad c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{potencial difference } \Delta\Phi_i}$$

- If $\Delta\Phi_i > 0$, then $\hat{c}_i > c_i$.
  $\rightarrow$ i-th operation stores energy in the data structure (that can be used later).
- If $\Delta\Phi_i < 0$, then $\hat{c}_i < c_i$.
  $\rightarrow$ data structure spends stored energy to help pay the real cost of the i-th operation.

Potencial method

The total amortized cost of the n operations is:

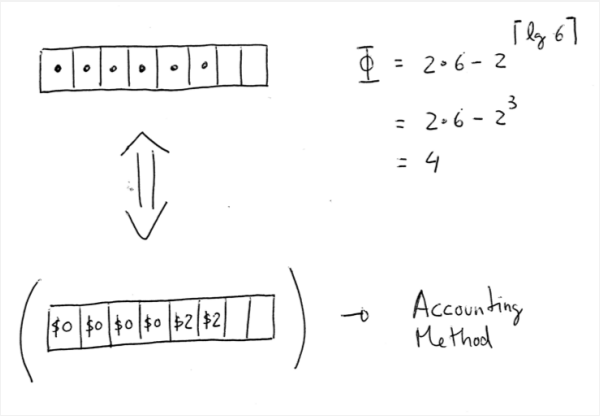
$$\begin{aligned}
\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\
&= \left(\sum_{i=1}^{n} c_i \right) + \Phi(D_n) - \Phi(D_0) \\
&\geq \sum_{i=1}^{n} c_i \quad \text{, because } \Phi(D_n) \geq 0 \text{ and } \Phi(D_0) = 0
\end{aligned}$$

Dynamic table example

- We define the table potencial immediately after the i-th INSERT by:

$$\Phi(D_i) \quad = \quad 2i - 2^{\lceil \lg i \rceil} \quad \text{(assume } 2^{\lceil \lg 0 \rceil} = 0\text{)}$$

- Note $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0 \ , \forall_i$



Amortized cost of the i-th operation is:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= \left\{ \begin{array}{ll} i & \text{, if } i-1 \text{ is an exact power of 2} \\ 1 & \text{, otherwise} \end{array} \right\} \\
&\quad + \left(2i - 2^{\lceil \lg i \rceil}\right) - \left(2(i-1) - 2^{\lceil \lg (i-1) \rceil}\right) \\
&= \{\ldots\} + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil}.
\end{aligned}$$

Case based analysis

- Case 1: $i-1$ is an exact power of 2.

$$\begin{aligned}
\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
&= i + 2 - 2(i-1) + (i-1) \\
&= i + 2 - 2i + 2 + i + 1 \\
&= 3.
\end{aligned}$$

- Case 2: $i-1$ is not an exact power of 2.

$$\begin{aligned}
\hat{c}_i &= 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
&= 3. \quad \text{(because in this case } 2^{\lceil \lg i \rceil} = 2^{\lceil \lg (i-1) \rceil}\text{)}
\end{aligned}$$

- Therefore, n INSERTs cost $3n = \Theta(n)$, in the worst case.

Which method shall we use in practice?

- We can use any method we want (aggregate, accounting, potential).
- Some methods might be easier to apply in certain situations. Choose whatever feels more confortable to you.