Theory I
Algorithm Design and Analysis

(8 – Dynamic tables)

INSTITUT FÜR
INFORMATIK
FREIBURG

Dynamic Tables

Problem:

Maintenance of a table under the operations insert and delete such that

- the table size can be adjusted to the number of elements
- a fixed portion of the table is always filled with elements
- the costs for n insert or delete operations are in $O(n)$.

Organisation of the table: hash table, heap, stack, etc.

Load factor α_T : fraction of table spaces of T which are occupied.

Cost model:

Insertion or deletion of an element causes cost 1, if the table is not filled yet.
If the table size is changed, all elements must be copied.

Initialisation

```
class dynamicTable {
    private int [] table;
    private int size;
    private int num;
    dynamicTable () {
        table = new int [1];   // initialize empty table
        size = 1;
        num = 0;
    }
```

Expansion strategy: insert

Double the table size whenever an element is inserted in the fully occupied table!

```
public void insert (int x) {
    if (num == size) {
        int[] newTable = new int[2*size];
        for (int i=0; i < size; i++)
            insert table[i] in newTable;
        table = newTable;
        size = 2*size;
    }
    insert x in table;
    num = num + 1;
}
```

insert operations in an initially empty table

t_i = cost of the i-th insert operation

Worst case:

$t_i = 1$, if the table was not full before operation i

$t_i = (i - 1) + 1$, if the table was full before operation i

Hence, n insert operations require costs of at most

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

Tighter analysis

Let t_i be the cost of the ith insertion

$$t_i = \begin{cases} i & \text{if } i\text{-1 is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$$

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

Tighter analysis

Let t_i be the cost of the i th insertion

$$t_i = \begin{cases} i & \text{if } i - 1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$$

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

Tighter analysis

Cost of the n insertions

$$= \sum_{i=1}^{n} t_i$$

$$= n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j$$

$$<= 3n$$

Thus the average cost of each dynamic table operation is 3.

Amortized analysis

- An ***amortized analysis*** is any strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.
- Even though we're taking averages, however, probability is not involved!
- An amortized analysis guarantees the average performance of each operation in the *worst case*.

Types of amortized analysis

Three common amortization arguments:

– The ***aggregate*** method,

– The ***accounting*** method,

– The ***potential*** method.

We've just seen an aggregate analysis.

- The aggregate method, though simple, lacks the precision of the other two methods. In particular, the accounting and potential methods allow a specific ***amortized cost*** to be allocated to each operation.

Accounting method

- Charge *i*-th operation a fictitious ***amortized cost*** a_i, where \$1 pays for 1 unit of work (*i.e.*, time). This fee is consumed to perform the operation.
- Any amount not immediately consumed is stored in the ***bank*** for use by subsequent operations. The bank balance must not go negative!
- We must ensure that for all n,

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

Thus, the total amortized costs provide an upper bound on the total true costs.

Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.

0$	0$	0$	0$

Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.

0\$	0\$	2\$	0\$

Accounting analysis

Charge an amortized cost of a_i =$3 for the *i*-th insertion.

- $1 pays for the immediate insertion.
- $2 is stored for later table doubling.

When the table doubles, $1 pays to move a recent item, and $1 pays to move an old item.

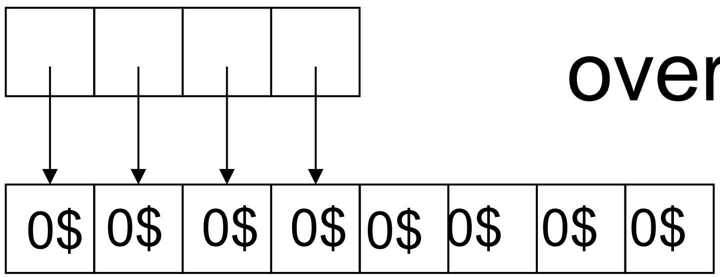
0$	0$	2$	2$

Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.



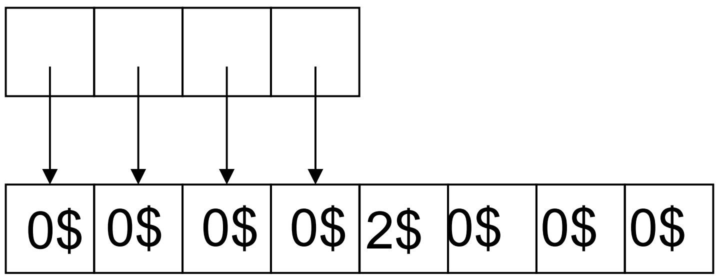
overflow

Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.

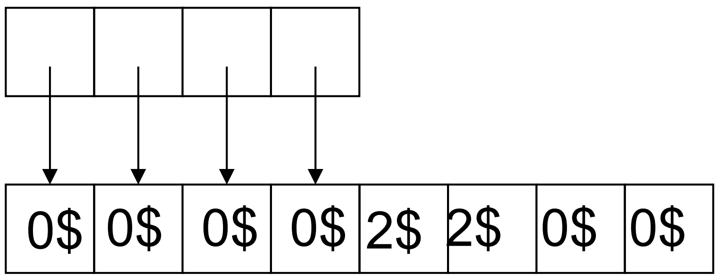


Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.

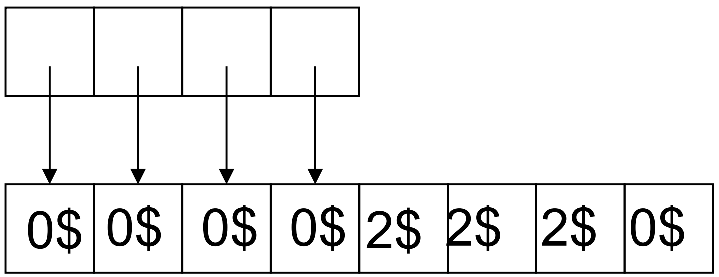


Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.

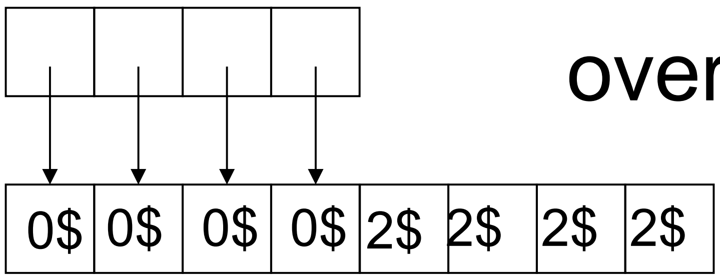


Accounting analysis

Charge an amortized cost of a_i =\$3 for the *i*-th insertion.

- \$1 pays for the immediate insertion.
- \$2 is stored for later table doubling.

When the table doubles, \$1 pays to move a recent item, and \$1 pays to move an old item.



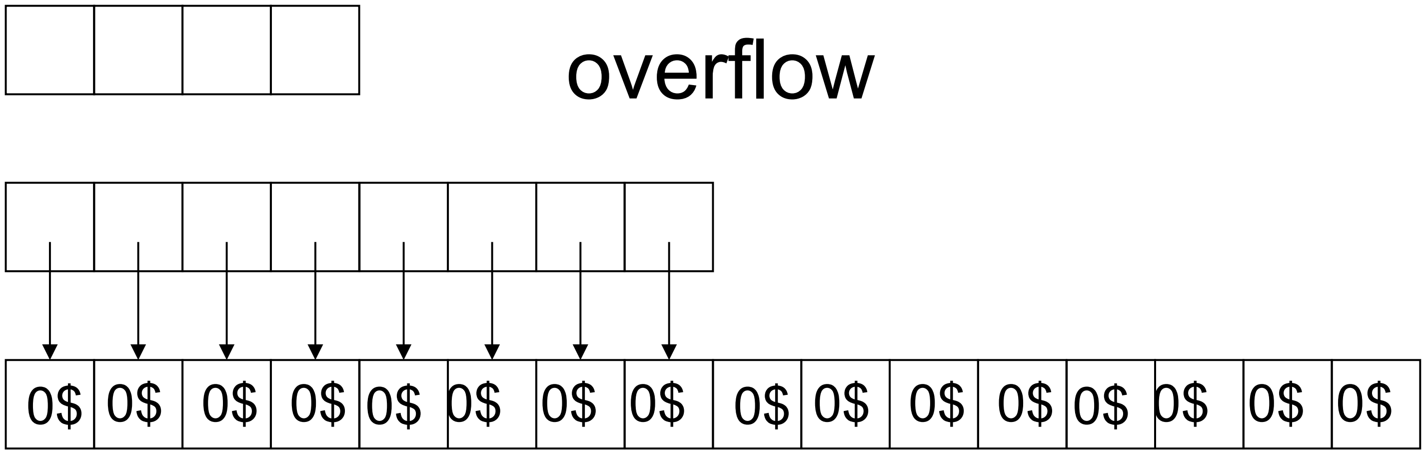
overflow

Accounting analysis

Charge an amortized cost of a_i =$3 for the *i*-th insertion.

- $1 pays for the immediate insertion.
- $2 is stored for later table doubling.

When the table doubles, $1 pays to move a recent item, and $1 pays to move an old item.



Potential method

IDEA: View the bank account as the potential energy of the dynamic set.

Framework:

- Start with an initial data structure D_0.
- Operation i transforms D_{i-1} to D_i.
- The cost of operation i is t_i.
- Define a ***potential function*** Φ: $\{D_i\} \to$R, such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all i.

The ***amortized cost*** a_i with respect to Φ is defined to be $a_i = t_i + \Phi(D_i) - \Phi(D_{i-1})$.

Potential method

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

$\Phi(D_i) - \Phi(D_{i-1})$ is called potential difference.

- If $\Phi(D_i) - \Phi(D_{i-1}) > 0$, operation i stores work in the data structure for later use.
- If $\Phi(D_i) - \Phi(D_{i-1}) < 0$, the data structure delivers up stored work to help pay for operation i.

Total amortized cost of n operation

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

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

$$\geq \sum_{i=1}^{n} t_i$$

since $\Phi(D_n) \geq 0$ and $\Phi(D_0) = 0$

Potential method

T table with

- $k = T.num$ elements and
- $s = T.size$ spaces

Potential function

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

•	•	•	•	•	•	•	

$\phi\,(T) = 2 * 7 - 8 = 6$

0$	0$	0$	0$	2$	2$	2$	

Accounting method

Properties of the potential function

Properties

- $\phi_0 = \phi(T_0) = \phi$ (empty table) = 0

- For all $i \geq 1$: $\phi_i = \phi(T_i) \geq 0$

  Since $\phi_n - \phi_0 \geq 0$, $\Sigma\, a_i$ is an upper bound for $\Sigma\, t_i$

- Directly before an expansion, $k = s$,

  hence $\phi(T) = k = s.$

- Directly after an expansion, $k = s/2$,

  hence $\phi(T) = 2k - s = 0$.

Amortized cost of insert (1)

k_i = # elements 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]

Amortized cost of insert (1)

k_i = # elements 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]

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

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

$$= 1 + 2(k_i - k_{i-1})$$

$= 1 + 2 = 3$

Amortized cost of insert (2)

Case 2: [*i*-th operation triggers an expansion]

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

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

$= s_{i-1} + 1 + 2 - 2s_{i-1} + s_{i-1}$

$=3$

Insertion and deletion of elements

Now: contract table, if the load is too small!

Goals:

(1) Load factor is always bounded below by a constant

(2) Amortized cost of a single insert or delete operation is constant.

First attempt:

- Expansion: same as before
- Contraction: halve the table size as soon as table is less than ½ occupied (after the deletion)!

„Bad“ sequence of insert and delete operations

		Cost
$n/2$ times insert (table fully occupied)		$3\ n/2$
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
$I_{n/2}, I, D, D, I, I, D, D, \ldots$ of length n:

Second attempt

Expansion: (as before) double the table size, if an element is inserted in the full table.

Contraction: As soon as the load factor is below ¼, halve the table size.

Hence:

At least ¼ of the table is always occupied, i.e.

$$1/4 \leq \alpha(T) \leq 1$$

Cost of a sequence of insert and delete operations?

Analysis: insert and delete

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

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: insert and delete

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

Directly after an expansion or contraction of the table:

$s = 2k$, hence $\phi(T) = 0$

insert

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: $\alpha_i \geq ½$

insert

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

Potential function ϕ

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

insert

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

Potential function ϕ

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

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

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

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

$$a_i = 0$$

insert

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

Potential function ϕ

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

insert

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

Potential function ϕ

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

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

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

$$a_i = 1 - (k_i - k_{i-1})$$

$$a_i = 0$$

delete

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

Case 1: $\alpha_{i-1} < ½$

Case 1.1: deletion causes no contraction

$s_i = s_{i-1}$

Potential function ϕ

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

delete

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

Case 1: $\alpha_{i-1} < ½$

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

$2s_i = s_{i-1}$

$k_{i-1} = s_{i-1}/4$

Potential function ϕ

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

delete

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

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

Case 2.1: $\alpha_{i-1} \geq ½$

Potential function ϕ

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

delete

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

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