Analysis of Algorithms

Amortized Analysis

Andres Mendez-Vazquez

February 26, 2018

Outline

Cinvestav

Outline

Cinvestav

History

Long Ago in a Faraway Land... too much The Hobbit

Aho, Ullman and Hopcroft in their book "Data Structures and Algorithms" (1983)

- They described a new complex analysis technique based in looking at the sequence of operations in a given data structure.
- They used it for describing the set operations under a binary tree data structure.

Cinvestav

History

Long Ago in a Faraway Land... too much The Hobbit

Aho, Ullman and Hopcroft in their book "Data Structures and Algorithms" (1983)

- They described a new complex analysis technique based in looking at the sequence of operations in a given data structure.
- They used it for describing the set operations under a binary tree data structure.

Robert Tarjan

Later on in the paper "Amortized Computational Complexity," Robert Trajan formalized the accounting and potential techniques of amortized analysis.

Cinvestav

History

Long Ago in a Faraway Land... too much The Hobbit

Aho, Ullman and Hopcroft in their book "Data Structures and Algorithms" (1983)

- They described a new complex analysis technique based in looking at the sequence of operations in a given data structure.
- They used it for describing the set operations under a binary tree data structure.

Robert Tarjan

Later on in the paper "Amortized Computational Complexity," Robert Trajan formalized the accounting and potential techniques of amortized analysis.

History

Long Ago in a Faraway Land... too much The Hobbit

Aho, Ullman and Hopcroft in their book "Data Structures and Algorithms" (1983)

- They described a new complex analysis technique based in looking at the sequence of operations in a given data structure.
- They used it for describing the set operations under a binary tree data structure.

Robert Tarjan

Later on in the paper "Amortized Computational Complexity," Robert Trajan formalized the accounting and potential techniques of amortized analysis.

Cinvestav

Outline

Cinvestav

Aggregate Analysis

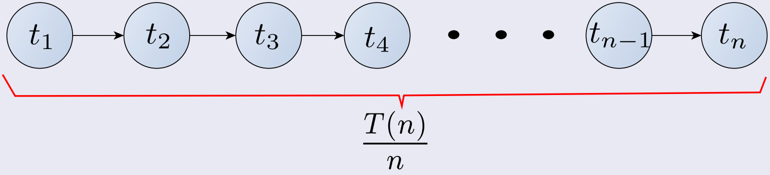
Aggregate Analysis

- The methods tries to determine an upper bound cost $T(n)$ for a sequence of n operations.
- Then, it calculates the amortized cost by using $\frac{T(n)}{n}$.

Aggregate Analysis

Aggregate Analysis

- The methods tries to determine an upper bound cost $T(n)$ for a sequence of n operations.
- Then, it calculates the amortized cost by using $\frac{T(n)}{n}$.



Accounting Method

Accounting Method

- The accounting method determines the individual cost of each operation, combining its immediate execution time and its influence on the running time of future operations by using a credit.

$$\textbf{Operation } real \textbf{ cost} + credit$$

Accounting Method

Accounting Method

- The accounting method determines the individual cost of each operation, combining its immediate execution time and its influence on the running time of future operations by using a credit.

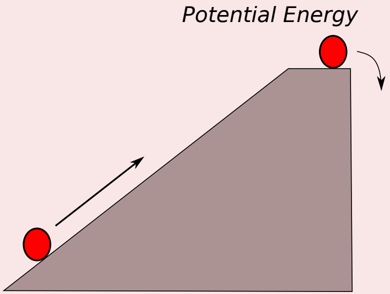
$$\textbf{Operation } real \textbf{ cost} + credit$$

Cinvestav

Potential Method

Potential Method

- The potential method is like the accounting method, but overcharges operations early to compensate for undercharges later.



Outline

Cinvestav

Aggregate Analysis

Stack with an extra Operation: Multipops

To begin exemplifying the aggregate analysis, let us add the following operation to the stack Data Structure.

```
Multipops(S, k)
    while not Stack-Empty(S) and k > 0
        POP(S)
        k = k - 1
```

Cinvestav

Aggregate Analysis

Case I Worst Case without Amortized Analysis

- Multipops is bounded by $\min(s, k)$, where $s =$number of elements in the stack.
- The worst case is $n-1$ pushes followed by a multipop with $k = n-1$.
- Then, we have that the worst complexity for an operation can be $O(n)$.
- Thus, for n operations we have $O(n^2)$ complexity.

Cinvestav

Aggregate Analysis

Case I Worst Case without Amortized Analysis

- Multipops is bounded by $\min(s, k)$, where $s =$number of elements in the stack.
- The worst case is $n-1$ pushes followed by a multipop with $k = n-1$.
- Then, we have that the worst complexity for an operation can be $O(n)$.
- Thus, for n operations we have $O(n^2)$ complexity.

Cinvestav

Aggregate Analysis

Case I Worst Case without Amortized Analysis

- Multipops is bounded by $\min(s, k)$, where $s =$number of elements in the stack.
- The worst case is $n - 1$ pushes followed by a multipop with $k = n - 1$.
- Then, we have that the worst complexity for an operation can be $O(n)$.
- Thus, for n operations we have $O(n^2)$ complexity.

Aggregate Analysis

Case I Worst Case without Amortized Analysis

- Multipops is bounded by $\min(s, k)$, where s =number of elements in the stack.
- The worst case is $n-1$ pushes followed by a multipop with $k = n-1$.
- Then, we have that the worst complexity for an operation can be $O(n)$.
- Thus, for n operations we have $O(n^2)$ complexity.

Cinvestav

Aggregate Analysis

Case II Now, we use the aggregate analysis

- Multipops depends on pops and pushes done before it.
- Then, any sequence of n pushes, pops and multipops on an initial empty stack cost at most $O(n)$
  - Because pop or multipops can be called in a non-empty stack is at most the number of pushes.
- Finally, the average cost for each operation is $\frac{O(n)}{n} = O(1)$.

Aggregate Analysis

Case II Now, we use the aggregate analysis

- Multipops depends on pops and pushes done before it.
- Then, any sequence of n pushes, pops and multipops on an initial empty stack cost at most $O(n)$
  - Because pop or multipops can be called in a non-empty stack is at most the number of pushes.
- Finally, the average cost for each operation is $\frac{O(n)}{n} = O(1)$.

Aggregate Analysis

Case II Now, we use the aggregate analysis

- Multipops depends on pops and pushes done before it.
- Then, any sequence of n pushes, pops and multipops on an initial empty stack cost at most $O(n)$
  - Because pop or multipops can be called in a non-empty stack is at most the number of pushes.
- Finally, the average cost for each operation is $\frac{O(n)}{n} = O(1)$.

Cinvestav

Aggregate Analysis

Case II Now, we use the aggregate analysis

- Multipops depends on pops and pushes done before it.
- Then, any sequence of n pushes, pops and multipops on an initial empty stack cost at most $O(n)$
  - Because pop or multipops can be called in a non-empty stack is at most the number of pushes.
- Finally, the average cost for each operation is $\frac{O(n)}{n} = O(1)$.

Cinvestav

Outline

Cinvestav

Example Binary Counter

We have something like this

0	0	0	.. ..	0	0
a_n	a_{n-1}	a_{n-2}		a_1	a_0

Basically

The Binary Counter is an array of bits to be used as a counter.

Example Binary Counter

We have something like this

0	0	0	.. ..	0	0
a_n	a_{n-1}	a_{n-2}		a_1	a_0

Basically

The Binary Counter is an array of bits to be used as a counter:

Example Binary Counter

Binary Counter

To begin exemplifying the aggregate method, let use the binary counter code.

A is an array of bits to be used as a counter. Each bit is a coefficient of the radix representation $x = \sum_{i=0}^{n} a_i 2^i$, where x is the counter.

```
Increment(A)
    i = 0
    while i < A.length and A[i] == 1
        A[i] = 0
        i = i + 1
    if i < A.length
        A[i] = 1
```

Example Binary Counter

Binary Counter

To begin exemplifying the aggregate method, let use the binary counter code.

A is an array of bits to be used as a counter. Each bit is a coefficient of the radix representation $x = \sum_{i=0}^{n} a_i 2^i$, where x is the counter.

Increment(A)

    $i = 0$

    while $i < A.length$ and $A[i] == 1$

        $A[i] = 0$

        $i = i + 1$

    if $i < A.length$

        $A[i] = 1$

Example Binary Counter

Binary Counter

To begin exemplifying the aggregate method, let use the binary counter code.

A is an array of bits to be used as a counter. Each bit is a coefficient of the radix representation $x = \sum_{i=0}^{n} a_i 2^i$, where x is the counter.

Increment(A)
    $i = 0$
    while $i < A.length$ and $A[i] == 1$
        $A[i] = 0$
        $i = i + 1$
    if $i < A.length$
        $A[i] = 1$

Example Binary Counter

Binary Counter

To begin exemplifying the aggregate method, let use the binary counter code.

A is an array of bits to be used as a counter. Each bit is a coefficient of the radix representation $x = \sum_{i=0}^{n} a_i 2^i$, where x is the counter.

```
Increment(A)
    i = 0
    while i < A.length and A[i] == 1
            A[i] = 0
            i = i + 1
    if i < A.length
            A[i] = 1
```

Example Binary Counter

Binary Counter

To begin exemplifying the aggregate method, let use the binary counter code.

A is an array of bits to be used as a counter. Each bit is a coefficient of the radix representation $x = \sum_{i=0}^{n} a_i 2^i$, where x is the counter.

Increment(A)
  $i = 0$
  while $i < A.length$ and $A[i] == 1$
    $A[i] = 0$
    $i = i + 1$
  if $i < A.length$
    $A[i] = 1$

Example Binary Counter

Binary Counter

To begin exemplifying the aggregate method, let use the binary counter code.

A is an array of bits to be used as a counter. Each bit is a coefficient of the radix representation $x = \sum_{i=0}^{n} a_i 2^i$, where x is the counter.

```
Increment(A)
    i = 0
    while i < A.length and A[i] == 1
            A[i] = 0
            i = i + 1
    if i < A.length
            A[i] = 1
```

Outline

Cinvestav

Example

Adding bits to a counter of size $k = 9$

$$k = 9$$

Bits in the Counter

0	0	0	0	0	0	0	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Example

Adding to the counter

1st Count	0	0	0	0	0	0	0	0	1
2nd Count	0	0	0	0	0	0	0	1	0
3rd Count	0	0	0	0	0	0	0	1	1
4th Count	0	0	0	0	0	0	1	0	0
5th	0	0	0	0	0	0	1	0	1
6th	0	0	0	0	0	0	1	1	0
7th	0	0	0	0	0	0	1	1	1
8th	0	0	0	0	0	1	0	0	0
9th	0	0	0	0	0	1	0	0	1
10th	0	0	0	0	0	1	0	1	0
11th	0	0	0	0	0	1	0	1	1
12th	0	0	0	0	0	1	1	0	0

Operations

We have

1. At the start of each iteration of the while loop in lines 2–4, we wish to add a 1 into position i.
2. If $A[i] == 1$, then adding 1 flips the bit to 0 in position i and a carry of 1 for $i+1$ on the next iteration of the loop.
3. If $A[i] == 0$ stop.
4. If i < A.length, we know that $A[i] == 0$, so flip to a 1.

Operations

We have

1. At the start of each iteration of the while loop in lines 2–4, we wish to add a 1 into position i.
2. If $A[i] == 1$, then adding 1 flips the bit to 0 in position i and a carry of 1 for $i + 1$ on the next iteration of the loop.
3. If $A[i] == 0$ stop.
4. If i < A.length, we know that $A[i] == 0$, so flip to a 1.

Complexity

1. The cost of each INCREMENT operation is linear in the number of bits flipped.
2. The worst case is $\Theta(k)$ in the worst case!!! Thus, for n operations we have $O(kn)$.

Operations

We have

1. At the start of each iteration of the while loop in lines 2–4, we wish to add a 1 into position i.
2. If $A[i] == 1$, then adding 1 flips the bit to 0 in position i and a carry of 1 for $i + 1$ on the next iteration of the loop.
3. If $A[i] == 0$ stop.
4. If i < A.length, we know that $A[i] == 0$, so flip to a 1.

Complexity

1. The cost of each INCREMENT operation is linear in the number of bits flipped.
2. The worst case is $\Theta(k)$ in the worst case!!! Thus, for n operations we have $O(kn)$.

Operations

We have

1. At the start of each iteration of the while loop in lines 2–4, we wish to add a 1 into position i.
2. If $A[i] == 1$, then adding 1 flips the bit to 0 in position i and a carry of 1 for $i+1$ on the next iteration of the loop.
3. If $A[i] == 0$ stop.
4. If i < A.length, we know that $A[i] == 0$, so flip to a 1.

Complexity

1. The cost of each INCREMENT operation is linear in the number of bits flipped.
2. The worst case is $\Theta(k)$ in the worst case!!! Thus, for n operations we have $O(kn)$.

Operations

We have

1. At the start of each iteration of the while loop in lines 2–4, we wish to add a 1 into position i.
2. If $A[i] == 1$, then adding 1 flips the bit to 0 in position i and a carry of 1 for $i + 1$ on the next iteration of the loop.
3. If $A[i] == 0$ stop.
4. If i < A.length, we know that $A[i] == 0$, so flip to a 1.

Complexity

1. The cost of each INCREMENT operation is linear in the number of bits flipped.
2. The worst case is $\Theta(k)$ in the worst case!!! Thus, for n operations we have $O(kn)$.

Operations

We have

1. At the start of each iteration of the while loop in lines 2–4, we wish to add a 1 into position i.
2. If $A[i] == 1$, then adding 1 flips the bit to 0 in position i and a carry of 1 for $i+1$ on the next iteration of the loop.
3. If $A[i] == 0$ stop.
4. If i < A.length, we know that $A[i] == 0$, so flip to a 1.

Complexity

1. The cost of each INCREMENT operation is linear in the number of bits flipped.
2. The worst case is $\Theta(k)$ in the worst case!!! Thus, for n operations we have $O(kn)$.

Better Analysis

Did you notice the following...

1. $A[0]$ flips $\lfloor n/2^0 \rfloor$ time
2. $A[1]$ flips $\lfloor n/2^1 \rfloor$ time
3. etc

Better Analysis

Did you notice the following...

1. $A[0]$ flips $\lfloor n/2^0 \rfloor$ time
2. $A[1]$ flips $\lfloor n/2^1 \rfloor$ time
3. etc

The total work is...

Look at the Board...

Better Analysis

Did you notice the following...

1. $A[0]$ flips $\lfloor n/2^0 \rfloor$ time
2. $A[1]$ flips $\lfloor n/2^1 \rfloor$ time
3. etc

The total work is...

Look at the Board...

Better Analysis

Did you notice the following...

1. $A[0]$ flips $\lfloor n/2^0 \rfloor$ time
2. $A[1]$ flips $\lfloor n/2^1 \rfloor$ time
3. etc

The total work is...

Look at the Board...

Outline

Cinvestav

Accounting Method

The use of credit

- When an operation, with an **amortized cost** $\widehat{c}_i$ (operation i), exceeds its actual cost, we give the difference to a $credit$.
  - This is to be stored in the data structure.

Cinvestav

Accounting Method

The use of credit

- When an operation, with an **amortized cost** $\widehat{c}_i$ (operation i), exceeds its actual cost, we give the difference to a $credit$.
  - This is to be stored in the data structure.

We have that

- As long as the charges are set so that it is impossible to go into debt.
  - one can show that there will never be an operation whose actual cost is greater than the sum of its charge plus the previously accumulated credit.

Accounting Method

The use of credit

- When an operation, with an **amortized cost** $\widehat{c}_i$ (operation i), exceeds its actual cost, we give the difference to a $credit$.
  - This is to be stored in the data structure.

We have that

- As long as the charges are set so that it is impossible to go into debt.
  - one can show that there will never be an operation whose actual cost is greater than the sum of its charge plus the previously accumulated credit.

Cinvestav

Accounting Method

The use of credit

- When an operation, with an **amortized cost** $\widehat{c}_i$ (operation i), exceeds its actual cost, we give the difference to a $credit$.
  - This is to be stored in the data structure.

We have that

- As long as the charges are set so that it is impossible to go into debt.
  - one can show that there will never be an operation whose actual cost is greater than the sum of its charge plus the previously accumulated credit.

More Formally

Something Notable

- Actual cost of the i^{th} operation is c_i.
- The amortized (charge) of the i^{th} operation is $\hat{c}_i$.

More Formally

Something Notable

- Actual cost of the i^{th} operation is c_i.
- The amortized (charge) of the i^{th} operation is $\widehat{c}_i$.

Properties

- If $\widehat{c}_i > c_i$ the i^{th} operation leaves some positive amount of credit, $credit = \widehat{c}_i - c_i$.

Cinvestav

More Formally

Something Notable

- Actual cost of the i^{th} operation is c_i.
- The amortized (charge) of the i^{th} operation is $\widehat{c}_i$.

Properties

- If $\widehat{c}_i > c_i$ the i^{th} operation leaves some positive amount of credit, $credit = \widehat{c}_i - c_i$.

Therefore

And as long

$$\sum_{i=1}^{n} \widehat{c}_i \geq \sum_{i=1}^{n} c_i \quad (1)$$

Therefore

And as long

$$\sum_{i=1}^{n} \widehat{c}_i \geq \sum_{i=1}^{n} c_i \qquad (1)$$

Then

The total available credit will always be nonnegative, and the sum of amortized costs will be an upper bound on the actual cost.

Outline

Cinvestav

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

- The Binary counter never charges a 0 flip if the bit was never changed to 1.

- Amortized cost of n operations is $O(n)$.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Thus

- The credit never becomes negative, or $\sum_{i=1}^{n} \widehat{c}_i \geq \sum_{i=1}^{n} c_i$.
- Amortized cost of n operations is $O(n)$.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
    1. One for the actual set of the bit to one.
    2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Thus

- The credit never becomes negative, or $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.
- Amortized cost of n operations is $O(n)$.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Thus

- The credit never becomes negative, or $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.
- Amortized cost of n operations is $O(n)$.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Thus

- The credit never becomes negative, or $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.
- Amortized cost of n operations is $O(n)$.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Thus

- The credit never becomes negative, or $\sum_{i=1}^{n} \widehat{c}_i \geq \sum_{i=1}^{n} c_i$.
- Amortized cost of n operations is $O(n)$.

Example Binary Counter II

Binary Counter Cost Operations

1. We charge 2 units of cost to flip a bit to 1.
   1. One for the actual set of the bit to one.
   2. Another for the flipping back to 0.
2. We do not charge anything to reset the bit to 0 because we use the credit stored at it.

Observation

- The numbers of 1 at the bit counter never becomes negative.
- The Binary counter never charges a 0 flip if the bit was never changed to 1.

Thus

- The credit never becomes negative, or $\sum_{i=1}^{n} \widehat{c}_i \geq \sum_{i=1}^{n} c_i$.
- Amortized cost of n operations is $O(n)$.

Outline

Cinvestav

The Potential Method

Basics

1. n operations are performed in initial data structure D_0.
2. c_i be the actual cost and D_i the data structure resulting of that operation.
3. Potential function $\Phi : \{D_0, D_1, ..., D_n\} \rightarrow \mathbb{R}$ that describe the potential energy on each data structure D_i.
4. Then, we have an **amortized cost**: $\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

The Potential Method

Basics

1. n operations are performed in initial data structure D_0.
2. c_i be the actual cost and D_i the data structure resulting of that operation.
3. Potential function $\Phi : \{D_0, D_1, ..., D_n\} \rightarrow \mathbb{R}$ that describe the potential energy on each data structure D_i.
4. Then, we have an **amortized cost**: $\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Telescopic Sum

$$\begin{aligned}\sum_{i=1}^{n} \widehat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)\\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).\end{aligned}$$

The Potential Method

Basics

1. n operations are performed in initial data structure D_0.
2. c_i be the actual cost and D_i the data structure resulting of that operation.
3. Potential function $\Phi : \{D_0, D_1, ..., D_n\} \to \mathbb{R}$ that describe the potential energy on each data structure D_i.
4. Then, we have an **amortized cost**: $\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Telescopic Sum

$$\begin{aligned}\sum_{i=1}^{n} \widehat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).\end{aligned}$$

The Potential Method

Basics

1. n operations are performed in initial data structure D_0.
2. c_i be the actual cost and D_i the data structure resulting of that operation.
3. Potential function $\Phi : \{D_0, D_1, ..., D_n\} \rightarrow \mathbb{R}$ that describe the potential energy on each data structure D_i.
4. Then, we have an **amortized cost**: $\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Telescopic Sum

$$\begin{aligned} \sum_{i=1}^{n} \widehat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0). \end{aligned}$$

The Potential Method

Basics

1. n operations are performed in initial data structure D_0.
2. c_i be the actual cost and D_i the data structure resulting of that operation.
3. Potential function $\Phi : \{D_0, D_1, ..., D_n\} \to \mathbb{R}$ that describe the potential energy on each data structure D_i.
4. Then, we have an **amortized cost**: $\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Telescopic Sum

$$\sum_{i=1}^{n} \widehat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$$

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

The Potential Method

Basics

1. n operations are performed in initial data structure D_0.
2. c_i be the actual cost and D_i the data structure resulting of that operation.
3. Potential function $\Phi : \{D_0, D_1, ..., D_n\} \to \mathbb{R}$ that describe the potential energy on each data structure D_i.
4. Then, we have an **amortized cost**: $\widehat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Telescopic Sum

$$\begin{aligned}\sum_{i=1}^{n} \widehat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).\end{aligned}$$

Observations

Observation

Because we do not know the number of operations:

- We ask for $\Phi(D_i) \geq \Phi(D_0)$ for all i or if $\Phi(D_0) = 0$ then $\Phi(D_i) \geq 0$.

- [illegible] represents an overcharge to the ith operation.

Observations

Observation

Because we do not know the number of operations:

- We ask for $\Phi(D_i) \geq \Phi\left(D_0\right)$ for all i or if $\Phi\left(D_0\right) = 0$ then $\Phi\left(D_i\right) \geq 0$.

Note

- If $\Phi\left(D_i\right) - \Phi\left(D_{i-1}\right)$ is positive, then
  - $\widehat{c}_i$ **represents an overcharge to the** i**th operation.**

Observations

Observation

Because we do not know the number of operations:

- We ask for $\Phi(D_i) \geq \Phi(D_0)$ for all i or if $\Phi(D_0) = 0$ then $\Phi(D_i) \geq 0$.

Note

- If $\Phi(D_i) - \Phi(D_{i-1})$ is positive, then
  - $\widehat{c}_i$ represents an overcharge to the ith operation.

Observations

Observation

Because we do not know the number of operations:

- We ask for $\Phi(D_i) \geq \Phi(D_0)$ for all i or if $\Phi(D_0) = 0$ then $\Phi(D_i) \geq 0$.

Note

- If $\Phi(D_i) - \Phi(D_{i-1})$ is positive, then
  - **$\widehat{c}_i$ represents an overcharge to the ith operation.**

Outline

Cinvestav

Example Stack Operations I

Potential Function

- Φ on stack as the number of elements in the stack. Then:
  - $\Phi(D_0) = 0$.
  - $\Phi(D_i) \geq 0 = \Phi(D_0)$.

Cinvestav

Example Stack Operations I

Potential Function

- Φ on stack as the number of elements in the stack. Then:
  - $\Phi(D_0) = 0$.
  - $\Phi(D_i) \geq 0 = \Phi(D_0)$.

Case "PUSH"

- If the ith operation on a stack containing s objects is a push:

$$\begin{aligned} \widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + s + 1 - s \\ &= 1 + 1 = 2 \end{aligned}$$

Cinvestav

Example Stack Operations I

Potential Function

- Φ on stack as the number of elements in the stack. Then:
  - $\Phi(D_0) = 0$.
  - $\Phi(D_i) \geq 0 = \Phi(D_0)$.

Case "PUSH"

- If the ith operation on a stack containing s objects is a push:

$$\begin{aligned} \widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + s + 1 - s \\ &= 1 + 1 = 2 \end{aligned}$$

Example Stack Operations I

Potential Function

- Φ on stack as the number of elements in the stack. Then:
  - $\Phi\left(D_0\right) = 0$.
  - $\Phi\left(D_i\right) \geq 0 = \Phi\left(D_0\right)$.

Case "PUSH"

- If the ith operation on a stack containing s objects is a push:

$$\begin{aligned} \widehat{c}_i &= c_i + \Phi\left(D_i\right) - \Phi\left(D_{i-1}\right) \\ &= c_i + s + 1 - s \\ &= 1 + 1 = 2 \end{aligned}$$

Cinvestav

Example Stack Operations I

Potential Function

- Φ on stack as the number of elements in the stack. Then:
  - $\Phi\left(D_0\right) = 0.$
  - $\Phi\left(D_i\right) \geq 0 = \Phi\left(D_0\right).$

Case "PUSH"

- If the ith operation on a stack containing s objects is a push:

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi\left(D_i\right) - \Phi\left(D_{i-1}\right) \\
&= c_i + s + 1 - s \\
&= 1 + 1 = 2
\end{aligned}$$

Example Stack Operations I

Potential Function

- Φ on stack as the number of elements in the stack. Then:
  - $\Phi(D_0) = 0$.
  - $\Phi(D_i) \geq 0 = \Phi(D_0)$.

Case "PUSH"

- If the ith operation on a stack containing s objects is a push:

$$\begin{aligned} \widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + s + 1 - s \\ &= 1 + 1 = 2 \end{aligned}$$

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= c_i + s - k' + s \\
&= k' - k' = 0
\end{aligned}
$$

- It is similar to multipop, 0

- The amortized cost for all the three operations is $O(1)$.
- The worst-case cost of n operations is $O(n)$.

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$\begin{aligned}\widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + s - k' + s \\ &= k' - k' = 0\end{aligned}$$

Case "POP"

- It is similar to multipop, 0.

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$\begin{aligned} \widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + s - k' + s \\ &= k' - k' = 0 \end{aligned}$$

Case "POP"

- It is similar to multipop, 0.

Finally

- The amortized cost for all the three operations is $O(1)$.
- The worst-case cost of n operations is $O(n)$.

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$\begin{aligned}\widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + s - k' + s \\ &= k' - k' = 0\end{aligned}$$

Case "POP"

- It is similar to multipop, 0.

Finally

- The amortized cost for all the three operations is $O(1)$.
- The worst-case cost of n operations is $O(n)$.

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= c_i + s - k' + s \\
&= k' - k' = 0
\end{aligned}
$$

Case "POP"

- It is similar to multipop, 0.

Finally

- The amortized cost for all the three operations is $O(1)$.
- The worst-case cost of n operations is $O(n)$.

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= c_i + s - k' + s \\
&= k' - k' = 0
\end{aligned}
$$

Case "POP"

- It is similar to multipop, 0.

Finally

- The amortized cost for all the three operations is $O(1)$.
- The worst-case cost of n operations is $O(n)$.

Example Stack Operations II

Case "MULTIPOP"

- The ith operation on the stack with s elements is a multipop, thus $k' = \min(k, s)$:

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= c_i + s - k' + s \\
&= k' - k' = 0
\end{aligned}
$$

Case "POP"

- It is similar to multipop, 0.

Finally

- The amortized cost for all the three operations is $O(1)$.
- The worst-case cost of n operations is $O(n)$.

Outline

Cinvestav

We have the following

Definition

- Consider a linear list of items (such as a singly-linked list).
- To access the item in the ith position requires time i.

We have the following

Definition

- Consider a linear list of items (such as a singly-linked list).
- To access the item in the ith position requires time i.

Constraints

Also, any two contiguous items can be swapped in constant time

- One item may be accessed many times within a sequence
- Starting from some set initial list configuration

We have the following

Definition

- Consider a linear list of items (such as a singly-linked list).
- To access the item in the ith position requires time i.

Constraints

Also, any two contiguous items can be swapped in constant time

Goal

- The goal is to allow access to a sequence of n items in a minimal amount of time
  - One item may be accessed many times within a sequence
  - Starting from some set initial list configuration.

We have the following

Definition

- Consider a linear list of items (such as a singly-linked list).
- To access the item in the ith position requires time i.

Constraints

Also, any two contiguous items can be swapped in constant time

Goal

- The goal is to allow access to a sequence of n items in a minimal amount of time
  - One item may be accessed many times within a sequence
  - Starting from some set initial list configuration.

We have the following

Definition

- Consider a linear list of items (such as a singly-linked list).
- To access the item in the ith position requires time i.

Constraints

Also, any two contiguous items can be swapped in constant time

Goal

- The goal is to allow access to a sequence of n items in a minimal amount of time
  - One item may be accessed many times within a sequence
  - Starting from some set initial list configuration.

We have the following

Definition

- Consider a linear list of items (such as a singly-linked list).
- To access the item in the ith position requires time i.

Constraints

Also, any two contiguous items can be swapped in constant time

Goal

- The goal is to allow access to a sequence of n items in a minimal amount of time
  - One item may be accessed many times within a sequence
  - Starting from some set initial list configuration.

Thus, we have two cases

First

If the sequence of accesses is known in advance, one can design an optimal algorithm for swapping items to rearrange the list according to how often items are accessed, and when.

Second

However, if the sequence is not known in advance, a heuristic method for swapping items may be desirable.

Thus, we have two cases

First

If the sequence of accesses is known in advance, one can design an optimal algorithm for swapping items to rearrange the list according to how often items are accessed, and when.

Second

However, if the sequence is not known in advance, a heuristic method for swapping items may be desirable.

MTF Heuristic

Reality!!!

If item i is accessed at time t, it is likely to be accessed again soon after time t (i.e., there is **locality of reference**).

MTF Heuristic Example

Example Heuristic Bring to the front

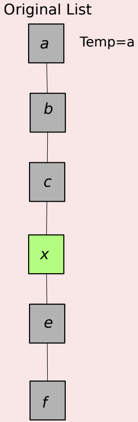


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

MTF Heuristic Example

Example Heuristic Bring to the front

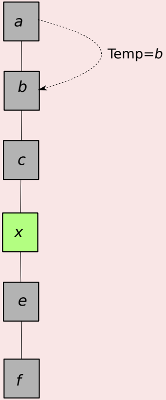


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

MTF Heuristic Example

Example Heuristic Bring to the front

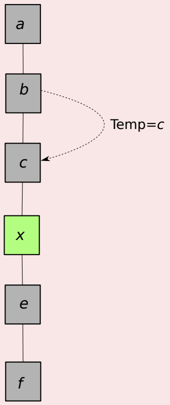


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

MTF Heuristic Example

Example Heuristic Bring to the front

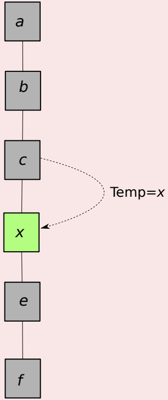


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

MTF Heuristic Example

Example Heuristic Bring to the front

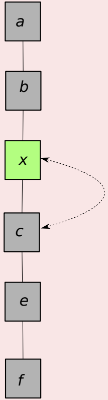


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

MTF Heuristic Example

Example Heuristic Bring to the front

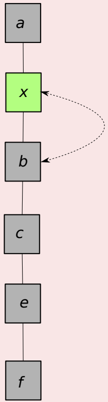


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

MTF Heuristic Example

Example Heuristic Bring to the front

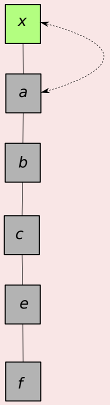


Figure: To access 'c' in the original list (left), walk down from 'a', then move 'c' to front by swapping with 'b' then 'a' (right)

Why does this work?!

Imagine the following

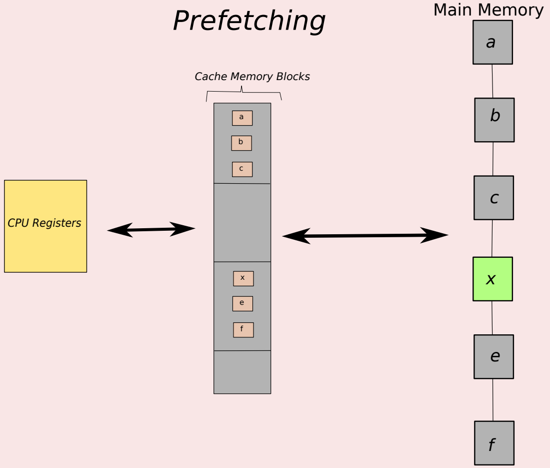


Figure: Here, we have the following situation: Swapping inside a block does not change the block itself!!!

Complexity of the Heuristic

Cost

It the ith item was accessed the cost is

1. i to access the item
2. $i-1$ for the swaps

Complexity of the Heuristic

Cost

It the ith item was accessed the cost is

1. i to access the item
2. $i-1$ for the swaps

Now, assume that

You have an optimal algorithm A that knows the access sequence in advance.

Complexity of the Heuristic

Cost

It the ith item was accessed the cost is

1. i to access the item
2. $i - 1$ for the swaps

Now, assume that

You have an optimal algorithm A that knows the access sequence in advance.

Potential of MTF at time t

As the $2 \times \{$the number of pairs of items whose order in the MTF's list differs from their order in A's list at time $t\}$ or

$$\phi(D_t) = 2 \times \{\text{the number of pairs of items whose order differs}\} \quad (2)$$

Complexity of the Heuristic

Cost

It the ith item was accessed the cost is

1. i to access the item
2. $i-1$ for the swaps

Now, assume that

You have an optimal algorithm A that knows the access sequence in advance.

Potential of MTF at time t

As the $2 \times \{$the number of pairs of items whose order in the MTF's list differs from their order in A's list at time $t\}$ or

$$\phi(D_t) = 2 \times \{\text{the number of pairs of items whose order differs}\} \quad (2)$$

Complexity of the Heuristic

Cost

It the ith item was accessed the cost is

1. i to access the item
2. $i - 1$ for the swaps

Now, assume that

You have an optimal algorithm A that knows the access sequence in advance.

Potential of MTF at time t

As the $2 \times \{$the number of pairs of items whose order in the MTF's list differs from their order in A's list at time $t\}$ or

$$\phi(D_t) = 2 \times \{\text{the number of pairs of items whose order differs}\} \quad (2)$$

Complexity of the Heuristic

For example

For example, if MTF's list is ordered (a, b, c, e, d) and A's list is ordered (a, b, c, d, e), then the potential for MTF will be equal to 2, because one pair of items (d and e) differ in their ordering between A's list and MTF's list.

Cinvestav

Complexity of the Heuristic

For example

For example, if MTF's list is ordered (a, b, c, e, d) and A's list is ordered (a, b, c, d, e), then the potential for MTF will be equal to 2, because one pair of items (d and e) differ in their ordering between A's list and MTF's list.

In addition

- The potential at $t = 0$ is 0, as both algorithms begin with the same list by definition.
- Also, it is impossible for the potential to be negative.

Complexity of the Heuristic

For example

For example, if MTF's list is ordered (a, b, c, e, d) and A's list is ordered (a, b, c, d, e), then the potential for MTF will be equal to 2, because one pair of items (d and e) differ in their ordering between A's list and MTF's list.

In addition

- The potential at $t = 0$ is 0, as both algorithms begin with the same list by definition.
- Also, it is impossible for the potential to be negative.

Thus, we have that

First

- Let x be at position k in MTF's list
- Let x be at position i in A's list

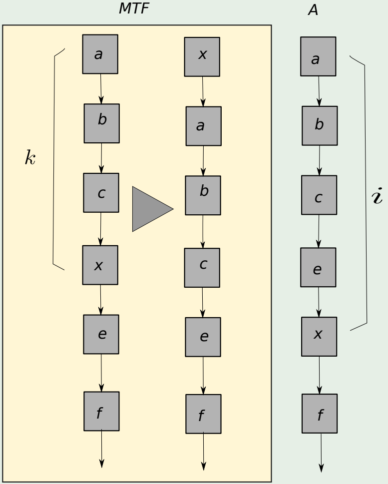
Thus, we have that

First

- Let x be at position k in MTF's list
- Let x be at position i in A's list

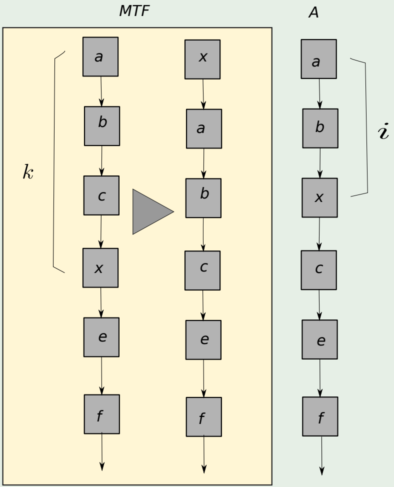
Case I

We can have this $i - 1 > k - 1$



Case II

We can have this $i - 1 < k - 1$



Cost

Then, the cost is for the MTF's list

$$c_i = 2\,(k-1) \tag{3}$$

Because the swapping can be done by putting you at position $k-1$ and doing $k-1$ swaps.

The cost for the A's list

$$c_i = i \tag{4}$$

Cost

Then, the cost is for the MTF's list

$$c_i = 2\,(k-1) \tag{3}$$

Because the swapping can be done by putting you at position $k-1$ and doing $k-1$ swaps.

The cost for the $A's$ list

$$c_i = i \tag{4}$$

Then

Why?

- Note that moving x to the front of the list reverses the ordering of all pairs including x and an item originally in location 1 to k–1
  - i.e., k–1 pairs in total.

Then

Why?

- Note that moving x to the front of the list reverses the ordering of all pairs including x and an item originally in location 1 to k–1
  - i.e., k–1 pairs in total.

In addition

- The relative positions of all other pairs are unchanged by the move.

Then

Why?

- Note that moving x to the front of the list reverses the ordering of all pairs including x and an item originally in location 1 to k–1
  - i.e., k–1 pairs in total.

In addition

- The relative positions of all other pairs are unchanged by the move.

What is $\phi(D_t) - \phi(D_{t-1})$?

We have that

- In A's list, there are i–1 items ahead of x.
- All of these will be behind x in MTF's list once x is moved to the front.

What is $\phi(D_t) - \phi(D_{t-1})$?

We have that

- In A's list, there are i–1 items ahead of x.
- All of these will be behind x in MTF's list once x is moved to the front.

Thus

- There are at most $\min\{k-1, i-1\}$ pair inversions that are **added** by the move to the front of x
  - i.e., disagreements in pair order between MTF's list and A's list.

What is $\phi(D_t) - \phi(D_{t-1})$?

We have that

- In A's list, there are i–1 items ahead of x.
- All of these will be behind x in MTF's list once x is moved to the front.

Thus

- There are at most $\min\{k–1, i–1\}$ pair inversions that are **added** by the move to the front of x
  - i.e., disagreements in pair order between MTF's list and A's list.

Thus, we have that the added inversions after x is moved to the front to be

$$\min\{k–1, i–1\} \quad (5)$$

What is $\phi(D_t) - \phi(D_{t-1})$?

We have that

- In A's list, there are i–1 items ahead of x.
- All of these will be behind x in MTF's list once x is moved to the front.

Thus

- There are at most $\min\{k{-}1, i{-}1\}$ pair inversions that are **added** by the move to the front of x
  - i.e., disagreements in pair order between MTF's list and A's list.

Thus, we have that the added inversions after x is moved to the front to be

$$\min\{k{-}1, i{-}1\} \tag{5}$$

What is $\phi(D_t) - \phi(D_{t-1})$?

We have that

- In A's list, there are i–1 items ahead of x.
- All of these will be behind x in MTF's list once x is moved to the front.

Thus

- There are at most $\min\{k–1, i–1\}$ pair inversions that are **added** by the move to the front of x
  - i.e., disagreements in pair order between MTF's list and A's list.

Thus, we have that the added inversions after x is moved to the front to be

$$\min\{k–1, i–1\} \tag{5}$$

Now, What about ?

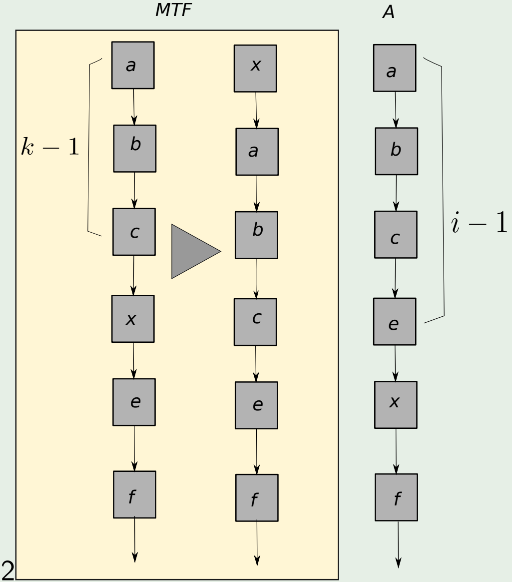
We have

All other ordering reversals must result in pair inversion **removals** or the places where MTF and A agree:

$$\text{At least } k{-}1{-}\min\{k{-}1, i{-}1\} \tag{6}$$

Example

We can have this



Then

We have that

We have that $\phi(D_t) - \phi(D_{t-1})$ can be seen as twice the difference of inversions between D_t and D_{t-1}

- i.e. the potential change

The maximum number of inversion that exist after moving x to the front is

$$\min\{k-1, i-1\} - (k-1-\min\{k-1, i-1\})$$

The potential change incurred in this single access and move to front is bounded above by

$$2(\min\{k-1, i-1\} - (k-1-\min\{k-1, i-1\})) = 4\min\{k-1, i-1\} - 2(k-1).$$

Then

We have that

We have that $\phi(D_t) - \phi(D_{t-1})$ can be seen as twice the difference of inversions between D_t and D_{t-1}

- i.e. the potential change

The maximum number of inversion that exist after moving x to the front is

$$\min\{k-1, i-1\} - (k-1-\min\{k-1, i-1\})$$

The potential change incurred in this single access and move to front is bounded above by

$$2(\min\{k-1, i-1\} - (k-1-\min\{k-1, i-1\})) = 4\min\{k-1, i-1\} - 2(k-1).$$

Then

We have that

We have that $\phi(D_t) - \phi(D_{t-1})$ can be seen as twice the difference of inversions between D_t and D_{t-1}

- i.e. the potential change

The maximum number of inversion that exist after moving x to the front is

$$\min\{k–1, i–1\}–(k–1–\min\{k–1, i–1\})$$

The potential change incurred in this single access and move to front is bounded above by

$$2(\min\{k–1, i–1\}–(k–1–\min\{k–1, i–1\})) = 4\min\{k–1, i–1\}–2(k–1).$$

Using the Upper Bound of the Potential Change

And Taking in account that the real cost of swapping is

$$c = 2\,(k-1)$$

Using the Upper Bound of the Potential Change

And Taking in account that the real cost of swapping is

$$c = 2(k-1)$$

We have the Upper bound for the Potential Cost

$$\begin{aligned}\hat{c} &= c + \phi(D_t) - \phi(D_{t-1}) \\ &\leq 2(k-1) + 4\min\{k-1, i-1\} - 2(k-1) \\ &\leq 4\min\{k-1, i-1\}\end{aligned}$$

Cinvestav

Using the Upper Bound of the Potential Change

And Taking in account that the real cost of swapping is

$$c = 2(k-1)$$

We have the Upper bound for the Potential Cost

$$\begin{aligned}\hat{c} &= c + \phi(D_t) - \phi(D_{t-1}) \\ &\leq 2(k-1) + 4\min\{k-1, i-1\} - 2(k-1) \\ &\leq 4\min\{k-1, i-1\}\end{aligned}$$

Using the Upper Bound of the Potential Change

And Taking in account that the real cost of swapping is

$$c = 2(k-1)$$

We have the Upper bound for the Potential Cost

$$\begin{aligned}\hat{c} &= c + \phi(D_t) - \phi(D_{t-1}) \\ &\leq 2(k-1) + 4\min\{k-1, i-1\} - 2(k-1) \\ &\leq 4\min\{k-1, i-1\}\end{aligned}$$

Cinvestav

Potential Change

If $\min\{k-1, i-1\} = k-1$

Then $\hat{c} = c + \Delta\Phi \leq 4(k-1) \leq 4(i-1) \leq 4i$

Similarly, if $\min\{k-1, i-1\} = i-1$

Then $\hat{c} = c + \Delta\Phi \leq 4(i-1) \leq 4i$

The Total Amortized Cost

Therefore, the total amortized cost is an upper bound on the total actual cost of any access sequence.

Potential Change

If $\min\{k-1, i-1\} = k-1$

Then $\hat{c} = c + \Delta\Phi \leq 4(k-1) \leq 4(i-1) \leq 4i$

Similarly, if $\min\{k-1, i-1\} = i-1$

Then $\hat{c} = c + \Delta\Phi \leq 4(i-1) \leq 4i$

The Total Amortized Cost

Therefore, the total amortized cost is an upper bound on the total actual cost of any access sequence.

Cinvestav

Potential Change

If $\min\{k-1, i-1\} = k-1$

Then $\hat{c} = c + \Delta\Phi \leq 4(k-1) \leq 4(i-1) \leq 4i$

Similarly, if $\min\{k-1, i-1\} = i-1$

Then $\hat{c} = c + \Delta\Phi \leq 4(i-1) \leq 4i$

The Total Amortized Cost

Therefore, the total amortized cost is an upper bound on the total actual cost of any access sequence.

Finally

We have then

The amortized cost of a single access and movetofront by MTF is bounded above by four times the cost of the access by A.

BTW

A might independently perform swaps in response to a new access request.

Finally

We have then

The amortized cost of a single access and movetofront by MTF is bounded above by four times the cost of the access by A.

BTW

A might independently perform swaps in response to a new access request.

Cinvestav

For example

If A does swap in response to an access request.

- This incurs no additional actual cost on the part of MTF.
- But it will increase or decrease the new potential by 2 and the cost access of A will increase by 1.
- The bound on MTF's amortized cost still holds because
  - The amortized cost is increased by at most 2
  - but the bound is increased by 4 (Remember the multiplication by 2)

For example

If A does swap in response to an access request.

- This incurs no additional actual cost on the part of MTF.
- But it will increase or decrease the new potential by 2 and the cost access of A will increase by 1.
- The bound on MTF's amortized cost still holds because
  - The amortized cost is increased by at most 2
  - but the bound is increased by 4 (Remember the multiplication by 2)

Not only that

This is true no matter how many swap operations A performs.

Cinvestav

For example

If A does swap in response to an access request.

- This incurs no additional actual cost on the part of MTF.
- But it will increase or decrease the new potential by 2 and the cost access of A will increase by 1.
- The bound on MTF's amortized cost still holds because
  - The amortized cost is increased by at most 2
  - but the bound is increased by 4 (Remember the multiplication by 2)

Not only that

This is true no matter how many swap operations A performs.

Cinvestav

For example

If A does swap in response to an access request.

- This incurs no additional actual cost on the part of MTF.
- But it will increase or decrease the new potential by 2 and the cost access of A will increase by 1.
- The bound on MTF's amortized cost still holds because
  - The amortized cost is increased by at most 2
  - but the bound is increased by 4 (Remember the multiplication by 2)

Not only that

This is true no matter how many swap operations A performs.

Cinvestav

Final words

Using a MTF is more efficient

- Because in order to device A, it will require complex statistic estimators
- Against a simple MTF algorithm...

Final words

Using a MTF is more efficient

- Because in order to device A, it will require complex statistic estimators
- Against a simple MTF algorithm...

Outline

Cinvestav

Dynamic Tables

Definition

- A Dynamic Table T is basically a table where the following operations are supported:
  - TABLE-INSERT and TABLE-DELETE for individual elements.
  - Expansions: when more space is needed.
  - Contractions: when it is necessary to save memory.

Cinvestav

Dynamic Tables

Definition

- A Dynamic Table T is basically a table where the following operations are supported:
  - TABLE-INSERT and TABLE-DELETE for individual elements.
  - Expansions: when more space is needed.
  - Contractions: when it is necessary to save memory.

Possible Data Structures to Support Dynamic Tables

- Stack
- Heap
- Hash Tables
- Arrays

Dynamic Tables

Definition

- A Dynamic Table T is basically a table where the following operations are supported:
  - TABLE-INSERT and TABLE-DELETE for individual elements.
  - Expansions: when more space is needed.
  - Contractions: when it is necessary to save memory.

Possible Data Structures to Support Dynamic Tables

- Stack
- Heap
- Hash Tables
- Arrays

Dynamic Tables

Definition

- A Dynamic Table T is basically a table where the following operations are supported:
  - TABLE-INSERT and TABLE-DELETE for individual elements.
  - Expansions: when more space is needed.
  - Contractions: when it is necessary to save memory.

Possible Data Structures to Support Dynamic Tables

- Stack
- Heap
- Hash Tables
- Arrays

Dynamic Tables

Definition

- A Dynamic Table T is basically a table where the following operations are supported:
  - TABLE-INSERT and TABLE-DELETE for individual elements.
  - Expansions: when more space is needed.
  - Contractions: when it is necessary to save memory.

Possible Data Structures to Support Dynamic Tables

- Stack
- Heap
- Hash Tables
- Arrays

Dynamic Tables

Load Factor $\alpha(T)$

- Case I Empty Table
  - $\alpha(T) = 1$
- Case II Non-Empty Table
  - $\alpha(T)$ is the number of item stored at the table T divided by the size (number of slots) in the table T:

$$\alpha(T) = \frac{T.num}{T.size}$$

Dynamic Tables

Load Factor $\alpha(T)$

- Case I Empty Table
  - $\alpha(T) = 1$
- Case II Non-Empty Table
  - $\alpha(T)$ is the number of item stored at the table T divided by the size (number of slots) in the table T:

$$\alpha(T) = \frac{T.num}{T.size}$$

Observation

- If the load factor of a dynamic table is bounded by a constant, the unused space in the table is never more than a constant fraction of the total amount of space.

Dynamic Tables

Load Factor $\alpha(T)$

- Case I Empty Table
  - $\alpha(T) = 1$
- Case II Non-Empty Table
  - $\alpha(T)$ is the number of item stored at the table T divided by the size (number of slots) in the table T:

$$\alpha(T) = \frac{T.num}{T.size}$$

Observation

- If the load factor of a dynamic table is bounded by a constant, the unused space in the table is never more than a constant fraction of the total amount of space.

Dynamic Tables

Load Factor $\alpha(T)$

- Case I Empty Table
  - $\alpha(T) = 1$
- Case II Non-Empty Table
  - $\alpha(T)$ is the number of item stored at the table T divided by the size (number of slots) in the table T:

$$\alpha(T) = \frac{T.num}{T.size}$$

Observation

- If the load factor of a dynamic table is bounded by a constant, the unused space in the table is never more than a constant fraction of the total amount of space.

Outline

Cinvestav

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Code

Table-Insert(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.size == T.num
    allocate new − table with 2 · T.size slots
    insert items in T.table into new − table
    free T.table, T.table = new − table and T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Code

Table-Insert(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.size == T.num
    allocate new − table with 2 · T.size slots
    insert items in T.table into new − table
    free T.table, T.table = new − table and T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Code

Table-Insert(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.size == T.num
    allocate new − table with 2 · T.size slots
    insert items in T.table into new − table
    free T.table, T.table = new − table and T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Code

Table-Insert(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.size == T.num
    allocate new − table with 2 · T.size slots
    insert items in T.table into new − table
    free T.table, T.table = new − table and T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Code

Table-Insert(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.size == T.num
    allocate new − table with 2 · T.size slots
    insert items in T.table into new − table
    free T.table, T.table = new − table and T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```

Table Expansion

Heuristic

Allocate a new table with twice the size when $T.num = T.size$.

- We have only insertions:
  - The Load Factor is always $\geq \frac{1}{2}$.
  - Wasted space is never more than half the space.

Code

Table-Insert(T, x)

if $T.size == 0$
 allocate $T.table$ **with 1 slot**
 $T.size = 1$
if $T.size == T.num$
 allocate $new - table$ **with** $2 \cdot T.size$ **slots**
 insert items in $T.table$ **into** $new - table$
 free $T.table$, $T.table = new - table$ **and** $T.size = 2 \cdot T.size$
insert x **into** $T.table$
$T.num = T.num + 1$

Outline

1. Introduction
   - History
2. What is this all about amortized analysis?
   - The Methods
3. The Aggregate Method
   - Introduction
   - The Binary Counter
     - Example
4. The Accounting Method
   - Introduction
   - Binary Counter
5. The Potential Method
   - Introduction
   - Stack Operations
6. Real Life Examples
   - Move-To-Front (MTF)
   - Dynamic Tables
     - Table Expansion
     - Aggegated Analysis
     - Potential Method
     - Table Expansions and Contractions

Cinvestav

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - ★ $i - 1$ elements are copied, 1 for inserting the element i.
    - ★ Thus $c_i = i$

- Copy

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - ★ $i-1$ elements are copied, 1 for inserting the element i.
    - ★ Thus $c_i = i$

Observation

The worst case of an operation is $O(n)$ when you need to

- Expand
- Copy

Thus, for n operations the upper bound is $O\left(n^2\right)$ which is not a thigh bound!!!

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - ★ $i - 1$ elements are copied, 1 for inserting the element i.
    - ★ Thus $c_i = i$

Observation

The worst case of an operation is $O(n)$ when you need to

- Expand
- Copy

Thus, for n operations the upper bound is $O(n^2)$ which is not a thigh bound!!!

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - ★ $i-1$ elements are copied, 1 for inserting the element i.
    - ★ Thus $c_i = i$

Observation

The worst case of an operation is $O(n)$ when you need to

- Expand
- Copy

Thus, for n operations the upper bound is $O(n^2)$ which is not a thigh bound!!!

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - ★ $i-1$ elements are copied, 1 for inserting the element i.
    - ★ Thus $c_i = i$

Observation

The worst case of an operation is $O(n)$ when you need to

- Expand
- Copy

Thus, for n operations the upper bound is $O\left(n^2\right)$ which is not a thigh bound!!!

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - $i - 1$ elements are copied, 1 for inserting the element i.
    - Thus $c_i = i$

Observation

The worst case of an operation is $O(n)$ when you need to

- Expand
- Copy

Thus, for n operations the upper bound is $O(n^2)$ which is not a thigh bound!!!

Aggregated Analysis

Only Insertions in the table T

- Case Table is not full:
  - $c_i = 1$
- Case Table is full:
  - Table is expanded then
    - $i - 1$ elements are copied, 1 for inserting the element i.
    - Thus $c_i = i$

Observation

The worst case of an operation is $O(n)$ when you need to

- Expand
- Copy

Thus, for n operations the upper bound is $O\left(n^2\right)$ which is not a thigh bound!!!

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

- $i=2$ expand table and $i-1=2^0$. Then, $T.size = 2$
- $i=3$ expand table and $i-1=2$. Then, $T.size = 4$.
- $i=4$, table do not expand and $T.size = 4$.
- $i=5$, expand table and $i-1=2^2$ and $T.size = 8$

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

Example

- $i = 1$ start the table. Then, $T.size$ =1.
- $i = 2$ expand table and $i - 1 = 2^0$. Then, $T.size$ =2.
- $i = 3$ expand table and $i - 1 = 2$. Then, $T.size$ =4.
- $i = 4$, table do not expand and $T.size$ =4.
- $i = 5$, expand table and $i - 1 = 2^2$ and $T.size = 8$.

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

Example

- $i = 1$ start the table. Then, $T.size$ =1.
- $i = 2$ expand table and $i - 1 = 2^0$. Then, $T.size$ =2.
- $i = 3$ expand table and $i - 1 = 2$. Then, $T.size$ =4.
- $i = 4$, table do not expand and $T.size$ =4.
- $i = 5$, expand table and $i - 1 = 2^2$ and $T.size = 8$.

Final Cost

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

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

Example

- $i=1$ start the table. Then, $T.size$ =1.
- $i=2$ expand table and $i-1=2^0$. Then, $T.size$ =2.
- $i=3$ expand table and $i-1=2$. Then, $T.size$ =4.
- $i=4$, table do not expand and $T.size$ =4.
- $i=5$, expand table and $i-1=2^2$ and $T.size=8$.

Final Cost

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

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

Example

- $i=1$ start the table. Then, $T.size$ =1.
- $i=2$ expand table and $i-1=2^0$. Then, $T.size$ =2.
- $i=3$ expand table and $i-1=2$. Then, $T.size$ =4.
- $i=4$, table do not expand and $T.size$ =4.
- $i=5$, expand table and $i-1=2^2$ and $T.size=8$.

Final Cost

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

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

Example

- $i = 1$ start the table. Then, $T.size$ =1.
- $i = 2$ expand table and $i - 1 = 2^0$. Then, $T.size$ =2.
- $i = 3$ expand table and $i - 1 = 2$. Then, $T.size$ =4.
- $i = 4$, table do not expand and $T.size$ =4.
- $i = 5$, expand table and $i - 1 = 2^2$ and $T.size = 8$.

Final Cost

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

Aggregated Analysis

When expansions are done?

- The ith insertion can only cause an expansion of $i-1$ is a power of 2.

Example

- $i=1$ start the table. Then, $T.size$ =1.
- $i=2$ expand table and $i-1=2^0$. Then, $T.size$ =2.
- $i=3$ expand table and $i-1=2$. Then, $T.size$ =4.
- $i=4$, table do not expand and $T.size$ =4.
- $i=5$, expand table and $i-1=2^2$ and $T.size=8$.

Final Cost

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

Aggregated Analysis

Total Cost of n Table-Insert operations is

$$\sum_{i=1}^{n} c_i \leq \text{number of insertions} + \text{number of copies}$$

$$= n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j$$

$$= n + \frac{\left(1 - 2 \times 2^{\lfloor \lg n \rfloor}\right)}{1 - 2}$$

$$= n + 2 \times 2^{\lfloor \lg n \rfloor} - 1$$

$$< n + 2 \times 2^{\lfloor \lg n \rfloor}$$

$$= n + 2n^{\lg 2} = 3n$$

Cinvestav

Aggregated Analysis

Total Cost of n Table-Insert operations is

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\leq \text{number of insertions} + \text{number of copies} \\
&= n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j \\
&= n + \frac{\left(1 - 2 \times 2^{\lfloor \lg n \rfloor}\right)}{1 - 2} \\
&= n + 2 \times 2^{\lfloor \lg n \rfloor} - 1 \\
&< n + 2 \times 2^{\lfloor \lg n \rfloor} \\
&= n + 2n^{\lg 2} = 3n
\end{aligned}
$$

Aggregated Analysis

Total Cost of n Table-Insert operations is

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\leq \text{number of insertions} + \text{number of copies} \\
&= n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j \\
&= n + \frac{\left(1 - 2 \times 2^{\lfloor \lg n \rfloor}\right)}{1 - 2} \\
&= n + 2 \times 2^{\lfloor \lg n \rfloor} - 1 \\
&< n + 2 \times 2^{\lfloor \lg n \rfloor} \\
&\leq n + 2n^{\lg 2} = 3n
\end{aligned}
$$

Aggregated Analysis

Total Cost of n Table-Insert operations is

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\leq \text{number of insertions} + \text{number of copies} \\
&= n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j \\
&= n + \frac{\left(1 - 2 \times 2^{\lfloor \lg n \rfloor}\right)}{1 - 2} \\
&= n + 2 \times 2^{\lfloor \lg n \rfloor} - 1 \\
&< n + 2 \times 2^{\lfloor \lg n \rfloor} \\
&= n + 2n^{\lg 2} = 3n
\end{aligned}
$$

Cinvestav

Aggregated Analysis

Total Cost of n Table-Insert operations is

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\leq \text{number of insertions} + \text{number of copies} \\
&= n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j \\
&= n + \frac{\left(1 - 2 \times 2^{\lfloor \lg n \rfloor}\right)}{1-2} \\
&= n + 2 \times 2^{\lfloor \lg n \rfloor} - 1 \\
&< n + 2 \times 2^{\lfloor \lg n \rfloor} \\
&= n + 2n^{\lg 2} = 3n
\end{aligned}
$$

Aggregated Analysis

Total Cost of n Table-Insert operations is

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\leq \text{number of insertions} + \text{number of copies} \\
&= n + \sum_{j=1}^{\lfloor \lg n \rfloor} 2^j \\
&= n + \frac{\left(1 - 2 \times 2^{\lfloor \lg n \rfloor}\right)}{1-2} \\
&= n + 2 \times 2^{\lfloor \lg n \rfloor} - 1 \\
&< n + 2 \times 2^{\lfloor \lg n \rfloor} \\
&= n + 2n^{\lg 2} = 3n
\end{aligned}
$$

Cinvestav

Outline

Cinvestav

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- Then, $\Phi(T) = 2 \times T.num - T.size$.
  - After expansion $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0$.
  - Before expansion $T.num = T.size \Rightarrow \Phi(T) = T.num$

- $T.num \geq \frac{T.size}{2}$ always!!!.
- Therefore, $\Phi(T) \geq 0$

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- **Then,** $\Phi(T) = 2 \times T.num - T.size$**.**
  - After expansion $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0.$
  - Before expansion $T.num = T.size \Rightarrow \Phi(T) = T.num$

Observations

- The initial Potential Value is 0 because $T.num = 0$ and $T.size = 0$.
- $T.num \geq \frac{T.size}{2}$ always!!!.
- Therefore, $\Phi(T) \geq 0$

Cinvestav

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- **Then,** $\Phi(T) = 2 \times T.num - T.size$**.**
  - **After expansion** $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0$**.**
  - **Before expansion** $T.num = T.size \Rightarrow \Phi(T) = T.num$

Observations

- **The initial Potential Value is 0 because** $T.num = 0$ **and** $T.size = 0$**.**
- $T.num \geq \frac{T.size}{2}$ **always!!!.**
- **Therefore,** $\Phi(T) \geq 0$

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- **Then,** $\Phi(T) = 2 \times T.num - T.size$**.**
  - **After expansion** $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0$**.**
  - **Before expansion** $T.num = T.size \Rightarrow \Phi(T) = T.num$

Observations

- **The initial Potential Value is 0 because** $T.num = 0$ **and** $T.size = 0$**.**
- $T.num \geq \frac{T.size}{2}$ **always!!!.**
- **Therefore,** $\Phi(T) \geq 0$

Cinvestav

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- **Then,** $\Phi(T) = 2 \times T.num - T.size$**.**
  - **After expansion** $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0$**.**
  - **Before expansion** $T.num = T.size \Rightarrow \Phi(T) = T.num$

Observations

- **The initial Potential Value is 0 because** $T.num = 0$ **and** $T.size = 0$**.**
- $T.num \geq \frac{T.size}{2}$ **always!!!**
- **Therefore,** $\Phi(T) \geq 0$

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- **Then,** $\Phi(T) = 2 \times T.num - T.size$**.**
  - **After expansion** $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0$**.**
  - **Before expansion** $T.num = T.size \Rightarrow \Phi(T) = T.num$

Observations

- **The initial Potential Value is 0 because** $T.num = 0$ **and** $T.size = 0$**.**
- $T.num \geq \frac{T.size}{2}$ **always!!!.**
- **Therefore,** $\Phi(T) \geq 0$

Potential Method

Potential Function

- **We require potential Φ equal to 0 after expansion and builds after T is full.**
- **Then,** $\Phi(T) = 2 \times T.num - T.size$**.**
  - **After expansion** $T.num = \frac{T.size}{2} \Rightarrow \Phi(T) = 0$**.**
  - **Before expansion** $T.num = T.size \Rightarrow \Phi(T) = T.num$

Observations

- **The initial Potential Value is 0 because** $T.num = 0$ **and** $T.size = 0$**.**
- $T.num \geq \frac{T.size}{2}$ **always!!!.**
- **Therefore,** $\Phi(T) \geq 0$

Cinvestav

Potential Method

Notation for Analysis

- num_i =**Number of items stored at T after the ith operation.**
- $size_i$ =**The size of the table T after the ith operation.**
- Φ_i =**The potential after the ith operation.**

Potential Method

Notation for Analysis

- num_i =**Number of items stored at T after the ith operation.**
- $size_i$ =**The size of the table T after the ith operation.**
- Φ_i =**The potential after the ith operation.**

Potential Method

Notation for Analysis

- num_i =**Number of items stored at T after the ith operation.**
- $size_i$ =**The size of the table T after the ith operation.**
- Φ_i =**The potential after the ith operation.**

Potential Method

The ith Table-Insert operation does not trigger expansion

- Then, $size_i = size_{i-1}$.

Potential Method

The ith Table-Insert operation does not trigger expansion

- Then, $size_i = size_{i-1}$.

Thus

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot (num_i - 1) - size_i) \\
&= 3
\end{aligned}
$$

Cinvestav

Potential Method

The ith Table-Insert operation does not trigger expansion

- Then, $size_i = size_{i-1}$.

Thus

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot (num_i - 1) - size_i) \\
&= 3
\end{aligned}
$$

Cinvestav

Potential Method

The ith Table-Insert operation does not trigger expansion

- Then, $size_i = size_{i-1}$.

Thus

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot (num_i - 1) - size_i) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation does not trigger expansion

- Then, $size_i = size_{i-1}$.

Thus

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot (num_i - 1) - size_i) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Implying, $size_i = 2 \cdot (num_i - 1)$. In addition, $c_i = num_i$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - \ldots \\
&\quad (2 \cdot (num_i - 1) - (num_i - 1)) \\
&= num_i + (2 \cdot num_i - 2 \cdot num_i - 2) - (num_i - 1) \\
&= num_i + 2 - (num_i - 1) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Implying, $size_i = 2 \cdot (num_i - 1)$. In addition, $c_i = num_i$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - \ldots \\
&\quad (2 \cdot (num_i - 1) - (num_i - 1)) \\
&= num_i + (2 \cdot num_i - 2 \cdot num_i - 2) - (num_i - 1) \\
&= num_i + 2 - (num_i - 1) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Implying, $size_i = 2 \cdot (num_i - 1)$. In addition, $c_i = num_i$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - ... \\
&\quad (2 \cdot (num_i - 1) - (num_i - 1)) \\
&= num_i + (2 \cdot num_i - 2 \cdot num_i - 2) - (num_i - 1) \\
&= num_i + 2 - (num_i - 1) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Implying, $size_i = 2 \cdot (num_i - 1)$. In addition, $c_i = num_i$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - ... \\
&\quad (2 \cdot (num_i - 1) - (num_i - 1)) \\
&= num_i + (2 \cdot num_i - 2 \cdot num_i - 2) - (num_i - 1) \\
&= num_i + 2 - (num_i - 1) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Implying, $size_i = 2 \cdot (num_i - 1)$. In addition, $c_i = num_i$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - ... \\
&\quad (2 \cdot (num_i - 1) - (num_i - 1)) \\
&= num_i + (2 \cdot num_i - 2 \cdot num_i - 2) - (num_i - 1) \\
&= num_i + 2 - (num_i - 1) \\
&= 3
\end{aligned}
$$

Potential Method

The ith Table-Insert operation triggers expansion

- Then, $size_i = 2 \cdot size_{i-1}$, $size_{i-1} = num_{i-1} = num_i - 1$

Implying, $size_i = 2 \cdot (num_i - 1)$. In addition, $c_i = num_i$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - ... \\
&\quad (2 \cdot (num_i - 1) - (num_i - 1)) \\
&= num_i + (2 \cdot num_i - 2 \cdot num_i - 2) - (num_i - 1) \\
&= num_i + 2 - (num_i - 1) \\
&= 3
\end{aligned}
$$

Potential Under Table Expansions

The expansions generate the following graph for Φ

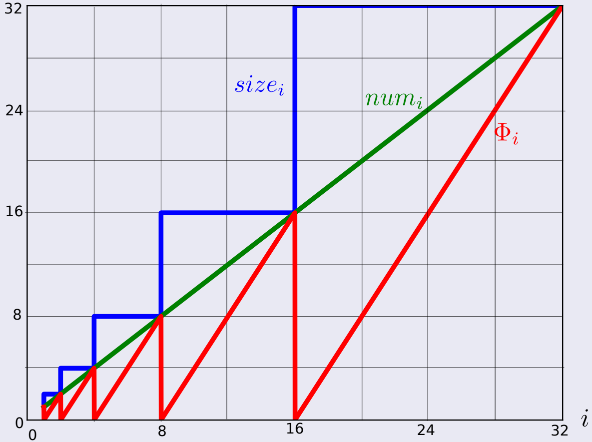


Figure: The Comparison between different quantities in the Dynamic Table.

Outline

Cinvestav

Table Expansions and Contractions

Properties to be maintained

- The load factor of the dynamic table is bounded below by a positive constant.
- The amortized cost of a table operation is bounded above by a constant.

- You halve the table size when deleting an item causes the table to become less than half full.

Table Expansions and Contractions

Properties to be maintained

- The load factor of the dynamic table is bounded below by a positive constant.
- The amortized cost of a table operation is bounded above by a constant.

Possible Heuristic, but not the correct one

- You **double** the table when inserting an item into a full table.
- You **halve** the table size, when deleting an item causes the table to become less than half full.

Table Expansions and Contractions

Properties to be maintained

- The load factor of the dynamic table is bounded below by a positive constant.
- The amortized cost of a table operation is bounded above by a constant.

Possible Heuristic, but not the correct one

- You **double** the table when inserting an item into a full table.
- You **halve** the table size, when deleting an item causes the table to become less than half full.

Cinvestav

Table Expansions and Contractions

Properties to be maintained

- The load factor of the dynamic table is bounded below by a positive constant.
- The amortized cost of a table operation is bounded above by a constant.

Possible Heuristic, but not the correct one

- You **double** the table when inserting an item into a full table.
- You **halve** the table size, when deleting an item causes the table to become less than half full.

Table Expansions and Contractions

Problem!!!

You could have $n = 2^t$ insertions and deletions in a sequence in the following sequence:

- First $\frac{n}{2}$ operations are insertions, thus $T.num = T.size = \frac{n}{2}$.

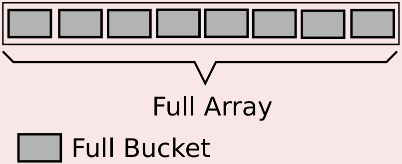
Table Expansions and Contractions

Problem!!!

You could have $n = 2^t$ insertions and deletions in a sequence in the following sequence:

- First $\frac{n}{2}$ operations are insertions, thus $T.num = T.size = \frac{n}{2}$.

Example $\frac{n}{2} = \frac{16}{2} = 8$



Then

For the second $\frac{n}{2}$ operations, the following sequence is performed

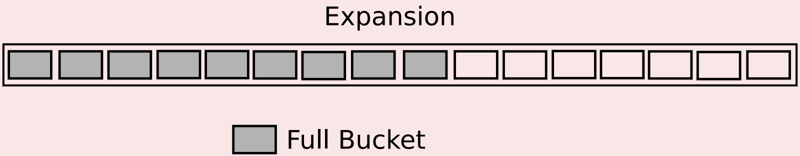
I,D,D,I,I,D,D,I,I,D,D,I,I,D,D,...

Then

For the second $\frac{n}{2}$ operations, the following sequence is performed

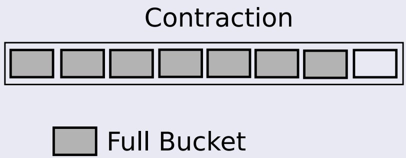
I,D,D,I,I,D,D,I,I,D,D,I,I,D,D,...

Thus, the first insertion cause a expansion to $T.size = n$



Next

The two following deletions trigger a contraction back to $T.size = \frac{n}{2}$



Next

The two following insertion trigger a expansion back to $T.size = n$

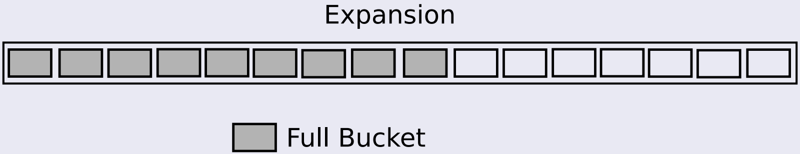


Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- The total cost of n operations is $\Theta(n^2)$.

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Potential Analysis

- Potential Function:
  - We require to have a function Φ that is 0 immediately after an expansion or contradiction.
  - Builds potential as the load factors increases to 1 or decreases to $\frac{1}{4}$.

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Potential Analysis

- Potential Function:
  - We require to have a function Φ that is 0 immediately after an expansion or contradiction.
  - Builds potential as the load factors increases to 1 or decreases to $\frac{1}{4}$.

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Potential Analysis

- Potential Function:
  - We require to have a function Φ that is 0 immediately after an expansion or contradiction.
  - Builds potential as the load factors increases to 1 or decreases to $\frac{1}{4}$.

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Potential Analysis

- Potential Function:
  - We require to have a function Φ that is 0 immediately after an expansion or contradiction.
  - Builds potential as the load factors increases to 1 or decreases to $\frac{1}{4}$.

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Potential Analysis

- Potential Function:
  - We require to have a function Φ that is 0 immediately after an expansion or contradiction.
  - Builds potential as the load factors increases to 1 or decreases to $\frac{1}{4}$.

Table Expansions and Contractions

Thus, we have that

We have two meetings one on Thursday at 5:00 PM at my office and another on Oracle at 11:00 AM Thanks... Doc Andrés

- The cost of each expansion and contraction is $\Theta(n)$.
- Then, there are $\Theta(n)$ operations.
- **The total cost of n operations is $\Theta(n^2)$.**

Improvement

- You double the table when inserting an item into a full table.
- You halve the table when deleting an item makes $\alpha(T) < \frac{1}{4}$.

Potential Analysis

- Potential Function:
  - We require to have a function Φ that is 0 immediately after an expansion or contradiction.
  - Builds potential as the load factors increases to 1 or decreases to $\frac{1}{4}$.

Table Expansions and Contractions

Final Potential Function

$$\Phi(T) = \begin{cases} 2 \cdot T.num - T.size & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{T.size}{2} - T.num & \text{if } \alpha(T) < \frac{1}{2} \end{cases}.$$

- Than, for an empty or not empty table
  - we always have $T.num = \alpha(T) \cdot T.size$

Table Expansions and Contractions

Final Potential Function

$$\Phi\left(T\right)=\begin{cases}2\cdot T.num-T.size & \text{if } \alpha\left(T\right)\geq\frac{1}{2}\\ \frac{T.size}{2}-T.num & \text{if } \alpha\left(T\right)<\frac{1}{2}\end{cases}.$$

Properties of this Function

- Empty table $T.num=T.size=0$, we have that $\alpha(T)=1$.
- Then, for an empty or not empty table
  - we always have $T.num=\alpha\left(T\right)\cdot T.size$.

Table Expansions and Contractions

Final Potential Function

$$\Phi\left(T\right)=\begin{cases} 2\cdot T.num - T.size & \text{if } \alpha\left(T\right)\geq\frac{1}{2}\\ \frac{T.size}{2}-T.num & \text{if } \alpha\left(T\right)<\frac{1}{2}\end{cases}.$$

Properties of this Function

- Empty table $T.num = T.size = 0$, we have that $\alpha(T) = 1$.
- Then, for an empty or not empty table
    - we always have $T.num = \alpha\left(T\right)\cdot T.size$.

Table Expansions and Contractions

Final Potential Function

$$\Phi(T) = \begin{cases} 2 \cdot T.num - T.size & \text{if } \alpha(T) \geq \frac{1}{2} \\ \frac{T.size}{2} - T.num & \text{if } \alpha(T) < \frac{1}{2} \end{cases}.$$

Properties of this Function

- Empty table $T.num = T.size = 0$, we have that $\alpha(T) = 1$.
- Then, for an empty or not empty table
  - we always have $T.num = \alpha(T) \cdot T.size$.

Table Expansions and Contractions

Therfore, we have that

- When $\alpha\left(T\right) = \frac{1}{2}$, the potential is 0.
- When $\alpha\left(T\right) = 1$, we have $T.size = T.num \Rightarrow \Phi\left(T\right) = T.num$. It can pay for an expansion, if an item is inserted.
- When $\alpha\left(T\right) = \frac{1}{4}$, we have $T.size = 4 \cdot T.num \Rightarrow \Phi\left(T\right) = T.num$. It can pay for a contraction, if an item is deleted.

Table Expansions and Contractions

Therfore, we have that

- When $\alpha(T) = \frac{1}{2}$, the potential is 0.
- When $\alpha(T) = 1$, we have $T.size = T.num \Rightarrow \Phi(T) = T.num$. It can pay for an expansion, if an item is inserted.
- When $\alpha(T) = \frac{1}{4}$, we have $T.size = 4 \cdot T.num \Rightarrow \Phi(T) = T.num$. It can pay for a contraction, if an item is deleted.

Table Expansions and Contractions

Therfore, we have that

- When $\alpha(T) = \frac{1}{2}$, the potential is 0.
- When $\alpha(T) = 1$, we have $T.size = T.num \Rightarrow \Phi(T) = T.num$. It can pay for an expansion, if an item is inserted.
- When $\alpha(T) = \frac{1}{4}$, we have $T.size = 4 \cdot T.num \Rightarrow \Phi(T) = T.num$. It can pay for a contraction, if an item is deleted.

Table Expansions and Contractions

Initialization

- $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$ and $\Phi_0 = 0$.

Table Expansions and Contractions

Initialization

- $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$ and $\Phi_0 = 0$.

Case ith operation is a Table-Insert

- If $\alpha_{i-1} \geq \frac{1}{2}$, if the table expand or not $\widehat{c}_i = 3$.
- If $\alpha_i < \frac{1}{2}$, then

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_i}{2} - (num_i - 1)\right) = 0
\end{aligned}$$

Table Expansions and Contractions

Initialization

- $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$ and $\Phi_0 = 0$.

Case ith operation is a Table-Insert

- If $\alpha_{i-1} \geq \frac{1}{2}$, if the table expand or not $\widehat{c}_i = 3$.
- If $\alpha_i < \frac{1}{2}$, then

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_i}{2} - (num_i - 1)\right) = 0
\end{aligned}
$$

Table Expansions and Contractions

Initialization

- $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$ and $\Phi_0 = 0$.

Case ith operation is a Table-Insert

- If $\alpha_{i-1} \geq \frac{1}{2}$, if the table expand or not $\widehat{c}_i = 3$.
- If $\alpha_i < \frac{1}{2}$, then

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_i}{2} - (num_i - 1)\right) = 0
\end{aligned}
$$

Cinvestav

Table Expansions and Contractions

Initialization

- $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$ and $\Phi_0 = 0$.

Case ith operation is a Table-Insert

- If $\alpha_{i-1} \geq \frac{1}{2}$, if the table expand or not $\widehat{c}_i = 3$.
- If $\alpha_i < \frac{1}{2}$, then

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_i}{2} - (num_i - 1)\right) = 0
\end{aligned}$$

Cinvestav

Table Expansions and Contractions

Initialization

- $num_0 = 0$, $size_0 = 0$, $\alpha_0 = 1$ and $\Phi_0 = 0$.

Case ith operation is a Table-Insert

- If $\alpha_{i-1} \geq \frac{1}{2}$, if the table expand or not $\widehat{c}_i = 3$.
- If $\alpha_i < \frac{1}{2}$, then

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_i}{2} - (num_i - 1)\right) = 0
\end{aligned}
$$

Cinvestav

Table Expansions and Contractions

Case ith operation is a Table-Insert

- If $\alpha_{i-1} < \frac{1}{2}$ and $\alpha_i \geq \frac{1}{2}$ then

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2num_i - size_i) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + (2\,(num_{i-1} + 1) - size_{i-1}) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 3 \cdot \alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 = 3
\end{aligned}$$

Table Expansions and Contractions

Case ith operation is a Table-Insert

- If $\alpha_{i-1} < \frac{1}{2}$ and $\alpha_i \geq \frac{1}{2}$ then

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2num_i - size_i) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + (2\left(num_{i-1} + 1\right) - size_{i-1}) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 3 \cdot \alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 = 3
\end{aligned}
$$

Table Expansions and Contractions

Case ith operation is a Table-Insert

- If $\alpha_{i-1} < \frac{1}{2}$ and $\alpha_i \geq \frac{1}{2}$ then

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2num_i - size_i) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + (2\left(num_{i-1} + 1\right) - size_{i-1}) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 3 \cdot \alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 = 3
\end{aligned}
$$

Table Expansions and Contractions

Case ith operation is a Table-Insert

- If $\alpha_{i-1} < \frac{1}{2}$ and $\alpha_i \geq \frac{1}{2}$ then

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2num_i - size_i) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + (2\left(num_{i-1} + 1\right) - size_{i-1}) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 3 \cdot \alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 = 3
\end{aligned}
$$

Table Expansions and Contractions

Case ith operation is a Table-Insert

- If $\alpha_{i-1} < \frac{1}{2}$ and $\alpha_i \geq \frac{1}{2}$ then

$$
\begin{aligned}
\widehat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2num_i - size_i) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + (2\left(num_{i-1} + 1\right) - size_{i-1}) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 3 \cdot \alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 = 3
\end{aligned}
$$

Cinvestav

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does not trigger a contraction

In this case, $num_i = num_{i-1} - 1$. Now, if $\alpha_{i-1} < \frac{1}{2}$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i \right) - \left(\frac{size_{i-1}}{2} - num_{i-1} \right) \\
&= 1 + \left(\frac{size_i}{2} - num_i \right) - \left(\frac{size_{i-1}}{2} - (num_i + 1) \right) = 2
\end{aligned}
$$

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does not trigger a contraction

In this case, $num_i = num_{i-1} - 1$. Now, if $\alpha_{i-1} < \frac{1}{2}$

$$
\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - (num_i + 1)\right) = 2
\end{aligned}
$$

Cinvestav

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does not trigger a contraction

In this case, $num_i = num_{i-1} - 1$. Now, if $\alpha_{i-1} < \frac{1}{2}$

$$\begin{aligned}
\widehat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= 1 + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - (num_i + 1)\right) = 2
\end{aligned}$$

Cinvestav

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$\alpha_{i-1} < \frac{1}{2}$$

$$c_i = num_i + 1$$

$$\frac{size_i}{2} = \frac{size_{i-1}}{4} = num_{i-1} = num_i + 1$$

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$\alpha_{i-1} < \frac{1}{2}$$

$$c_i = num_i + 1$$

$$\frac{size_i}{2} = \frac{size_{i-1}}{4} = num_{i-1} = num_i + 1$$

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$\alpha_{i-1} < \frac{1}{2}$$

$$c_i = num_i + 1$$

$$\frac{size_i}{2} = \frac{size_{i-1}}{4} = num_{i-1} = num_i + 1$$

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$\widehat{c}_i = (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right)$$

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$
\begin{aligned}
\widehat{c}_i &= (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= (num_i + 1) + (num_i + 1 - num_i) - (2 \cdot num_i + 2 - (num_i + 1)) \\
&= 1
\end{aligned}
$$

Case ith operation is a Table-Delete

- For $\alpha_{i-1} \geq \frac{1}{2}$.
- You can do an analysis and the amortized cost is bounded by a constant.

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$
\begin{aligned}
\widehat{c}_i &= (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right)\\
&= (num_i + 1) + (num_i + 1 - num_i) - (2 \cdot num_i + 2 - (num_i + 1))\\
&= 1
\end{aligned}
$$

Case ith operation is a Table-Delete

- For $\alpha_{i-1} \geq \frac{1}{2}$.
  - You can do an analysis and the amortized cost is bounded by a constant.

Therfore, we have that

The Time for any sequence of n operations on a Dynamic Table is $O(n)$.

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$
\begin{aligned}
\widehat{c}_i &= (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= (num_i + 1) + (num_i + 1 - num_i) - (2 \cdot num_i + 2 - (num_i + 1)) \\
&= 1
\end{aligned}
$$

Case ith operation is a Table-Delete

- For $\alpha_{i-1} \geq \frac{1}{2}$.
- You can do an analysis and the amortized cost is bounded by a constant.

Therfore, we have that

The Time for any sequence of n operations on a Dynamic Table is $O(n)$.

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$\begin{aligned}\widehat{c}_i &= (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right)\\ &= (num_i + 1) + (num_i + 1 - num_i) - (2 \cdot num_i + 2 - (num_i + 1))\\ &= 1\end{aligned}$$

Case ith operation is a Table-Delete

- For $\alpha_{i-1} \geq \frac{1}{2}$.
- You can do an analysis and the amortized cost is bounded by a constant.

Therfore, we have that

The Time for any sequence of n operations on a Dynamic Table is $O(n)$.

Table Expansions and Contractions

Case ith operation is a Table-Delete and it does trigger a contraction

$$\begin{aligned}
\widehat{c}_i &= (num_i + 1) + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right) \\
&= (num_i + 1) + (num_i + 1 - num_i) - (2 \cdot num_i + 2 - (num_i + 1)) \\
&= 1
\end{aligned}$$

Case ith operation is a Table-Delete

- For $\alpha_{i-1} \geq \frac{1}{2}$.
- You can do an analysis and the amortized cost is bounded by a constant.

Therfore, we have that

The Time for any sequence of n operations on a Dynamic Table is $O(n)$.

Change of Potential Under Expansions and Contractions

The Changes in Potential Φ

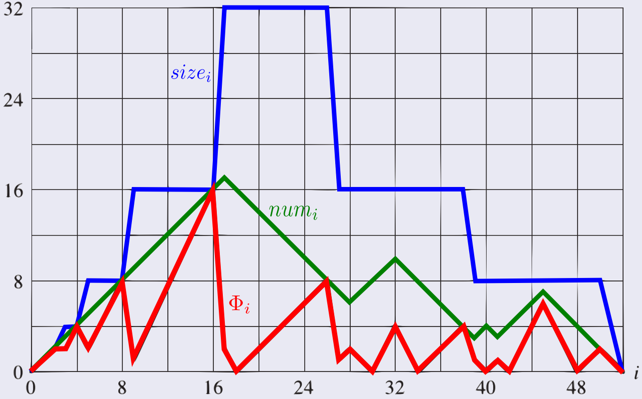


Figure: The Comparison between different quantities in the Dynamic Table.

Exercises

- 17.1-1
- 17.1-2
- 17.1-3
- 17.2-1
- 17.2-2
- 17.2-3
- 17.3-1
- 17.3-2
- 17.3-3
- 17.3-4
- 17.3-5
- 17.3-6
- 17.3-7