Algorithms

Jordi Planes

Escola Politècnica Superior
Universitat de Lleida

2016

Vote in Sakai.

Syllabus

What's been done

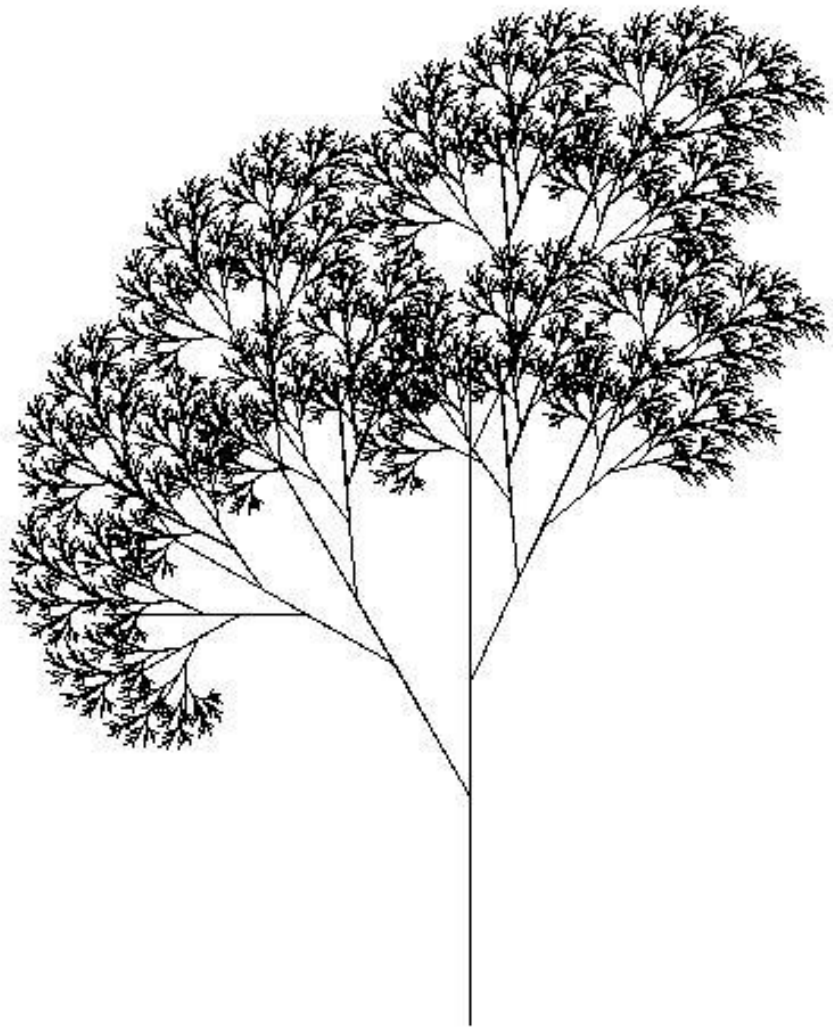
- Formal specification
- Cost
- Transformation recursion $\rightarrow$ iteration

Syllabus

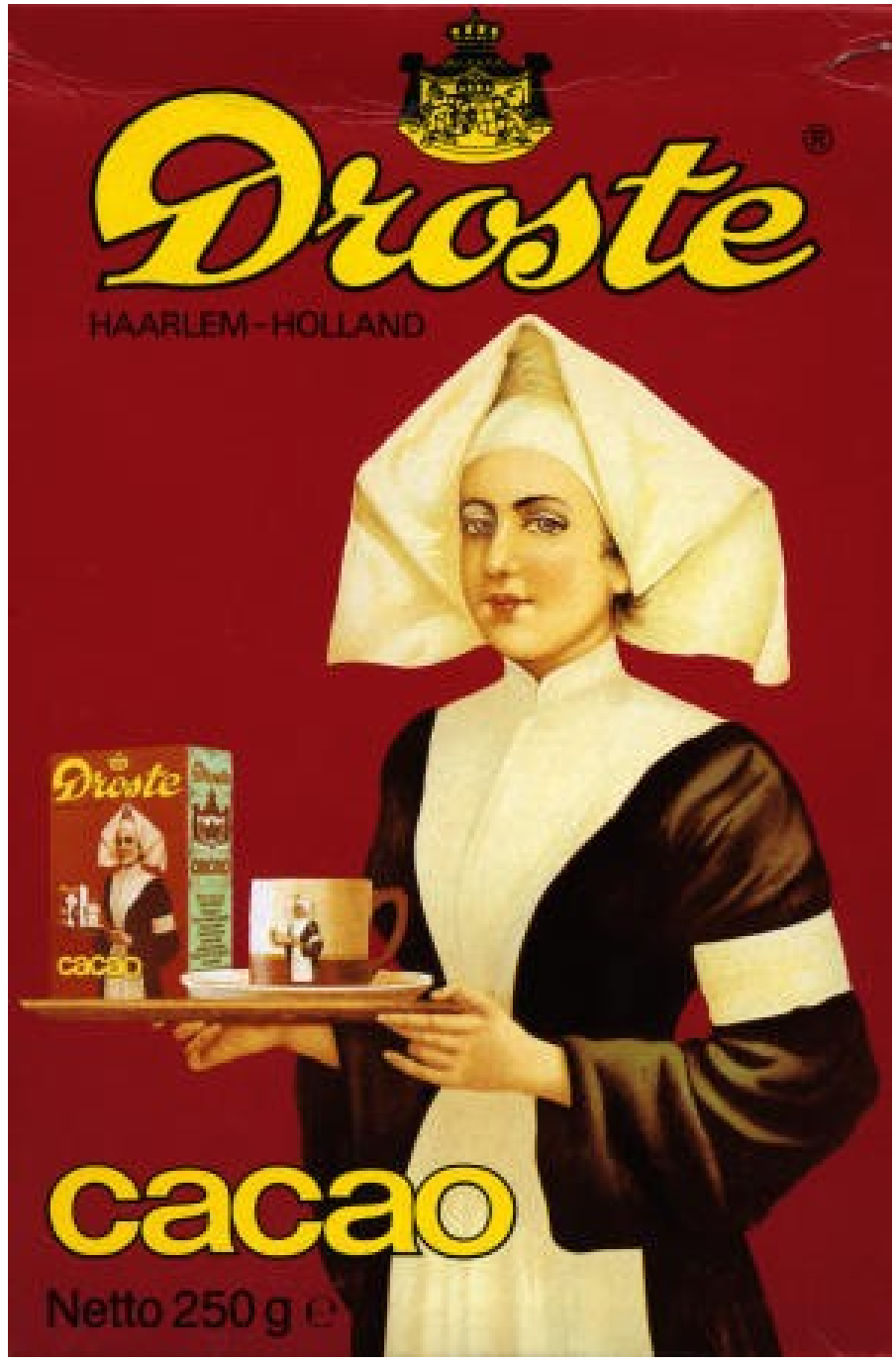
What we'll do today

- Formal specification
- Cost
- Transformation recursion $\rightarrow$ iteration

Recursion



“RecursiveTree” by Brentsmith101


Droste
HAARLEM - HOLLAND
Droste
cacao
cacao
Netto 250 g

Recursion

Recursion (Wikipedia)

Recursion is the process of repeating items in a self-similar way.

Recursion

Recursion (Wikipedia)

Recursion is the process of repeating items in a self-similar way.

Example (Fibonacci)

$$fibonacci(x) = x \cdot fibonacci(x - 1)$$

Recursion (John D. Cook)

Be sure of three things:

1. The problem gets smaller each time.
2. You include a solution for the base case.
3. Each case is handled correctly.

General scheme

```
function recursive( x ) is
    base case → return trivial( x )
    recursive case →
        y = recursive( reduce( x ) )
        return compute( x, y )
```

Example: factorial

```
function factorial( x ) is
    x = 0 → return 1
    x > 0 →
        y ← factorial( x−1 )
      return  x * y
```

Example: product

Design the product by additions

Example: product

```
function product( x, y ) is
    x = 0 → return 0
    x > 0 →
        p ← product( x−1, y )
      return  y + p
```

Exercises

- Digit counter for an integer
- Addition of all the elements in a list
- Check if a list is sorted
- Power by products
- Division by substrations
- Integer Division
- Maximum in a list
- Lineal search

Transformations

Transformations

- Any recursive computation has each equivalent iterative computation, and viceversa

Transformations

- Any recursive computation has each equivalent iterative computation, and viceversa
- The easiest transformation: tail recursion

Transformations

- Any recursive computation has each equivalent iterative computation, and viceversa
- The easiest transformation: tail recursion
- General Scheme:

$$Recursive \rightarrow Tail\ recursive \rightarrow Iterative$$

Transformations

Tail call

A tail call is a subroutine call performed as the final action of a procedure.

```
function caller( x ) is
    y = some operations
    return callee( y )
```

Transformations

Tail call

A tail call is a subroutine call performed as the final action of a procedure.

```
function caller( x ) is
    y = some operations
    return callee( y )
```

Tail recursion

A tail recursion is a tail call which calls the caller.

```
function caller( x ) is
    y = some operations
    return caller( y )
```

General scheme

```
function recursive( x ) is
    base case → return trivial( x )
    recursive case →
        y = recursive( reduce( x ) )
        return compute( x, y )

function recursive( x, a ) is
    base case → a
    recursive case →
        a = compute( x, a )
        return recursive( reduce( x ), a )
```

General scheme

```
function recursive( x ) is
    base case → return trivial( x )
    recursive case →
        y = recursive( reduce( x ) )
        return compute( x, y )

function recursive( x, a ) is
    base case → a
    recursive case →
        a = compute( x, a )
        return recursive( reduce( x ), a )

recursive( x, trivial( x ) )
```

Example: factorial

```
function factorial( x ) is
        x = 0 → return 1
        x > 0 →
          y ← factorial( x−1 )
         return  x * y
```

Example: factorial

```
function factorial( x ) is
        x = 0 → return 1
        x > 0 →
          y ← factorial( x−1 )
         return  x * y

function factorial( x, a ) is
        x = 0 → return a
        x > 0 →
          a ← x * a
         return factorial( x−1, a )

factorial( x, 1 )
```

Exercises

- Digit counter for an integer
- Addition of all the elements in a list
- Check if a list is sorted
- Power by products
- Division by substrations
- Integer Division
- Maximum in a list
- Lineal search

General scheme

```
function recursive( x, a ) is
    base case → a
    recursive case →
        return recursive( reduce( x ), compute( x, a

recursive( x, trivial(x) ) is
```

General scheme

```
function recursive( x, a ) is
    base case → a
    recursive case →
        return recursive( reduce( x ), compute( x, a

recursive( x, trivial(x) ) is

function recursive( x ) is
    a ← trivial( x )
    while not base case ( x ) do
        a ← compute( x, a )
        x ← reduce( x )
    return a
```

Example: factorial

```
function factorial( x, a ) is
        x = 0 → return a
        x > 0 →
          a ← x * a
         return factorial( x−1, a )

factorial( x, 1 )
```

Example: factorial

```
function factorial( x, a ) is
        x = 0 → return a
        x > 0 →
          a ← x * a
         return factorial( x−1, a )

factorial( x, 1 )

function factorial( x ) is
        a ← 1
        while not x = 0 do
            a ← a * x
            x ← x − 1
        return a
```

Exercises

- Digit counter for an integer
- Addition of all the elements in a list
- Check if a list is sorted
- Power by products
- Division by substrations
- Integer Division
- Maximum in a list
- Lineal search

Compute the costs

Compute the costs for the previous exercises.