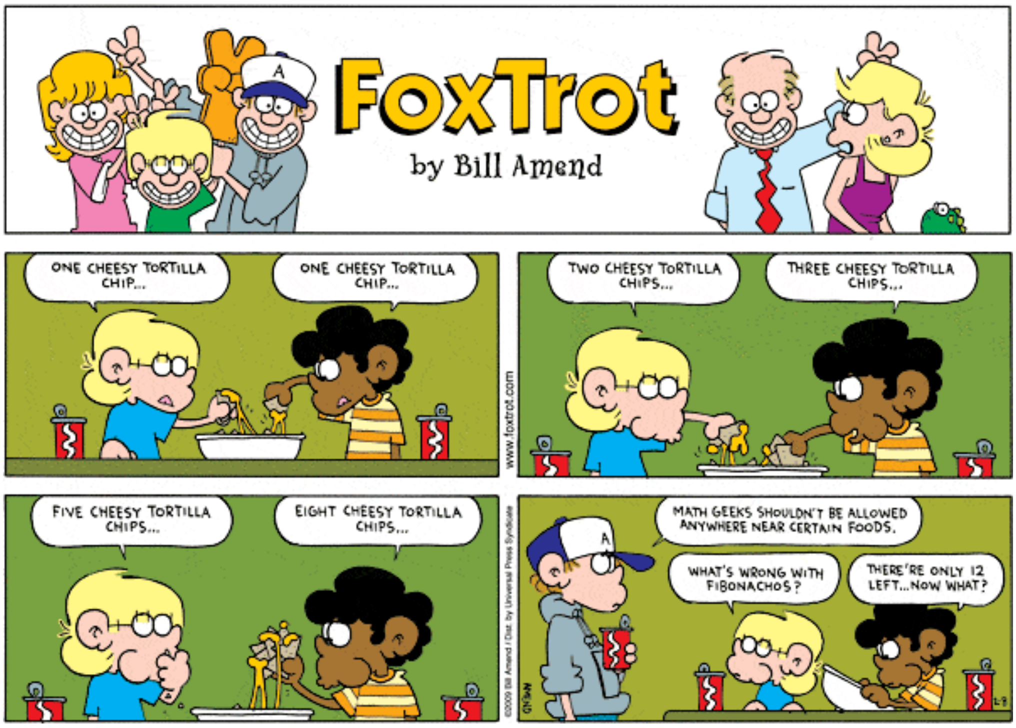
Algoritmi di Bioinformatica

Zsuzsanna Lipták

Laurea Magistrale Bioinformatica e Biotechnologie Mediche (LM9)
a.a. 2013/14, spring term

Computational efficiency I



Computational Efficiency

As we will see later in more detail, the efficiency of algorithms is measured w.r.t.

- running time
- storage space

We will make these concepts more concrete later on, but for now want to give some intuition, using an example.

Example: Computation of nth Fibonacci number

Fibonacci numbers: model for growth of populations (simplified model)

- Start with 1 pair of rabbits in a field
- each pair becomes mature at age of 1 month and mates
- after gestation period of 1 month, a female gives birth to 1 new pair
- rabbits never die[1]

Definition

$F(n)$ = number of pairs of rabbits in field after n months.

[1] This unrealistic assumption simplifies the mathematics; however, it turns out that adding a certain age at which rabbits die does not significantly change the behaviour of the sequence, so it makes sense to simplify.

Example: Computation of nth Fibonacci number

- month 1: there is 1 pair of rabbits in the field $F(1) = 1$
- month 2: there is still 1 pair of rabbits in the field $F(2) = 1$
- month 3: there is the old pair and 1 new pair $F(3) = 1 + 1 = 2$
- month 4: the 2 pairs from previous month, plus the old pair has had another new pair $F(4) = 2 + 1 = 3$
- month 5: the 3 from previous month, plus the 2 from month 3 have each had a new pair $F(5) = 3 + 2 = 5$

Recursion for Fibonacci numbers

$F(1) = F(2) = 1$
for $n > 2$: $F(n) = F(n-1) + F(n-2)$.

Example: Computation of nth Fibonacci number

Algorithm 1 (let's call it fib1) works exactly along the recursive definition:

Algorithm *fib1(n)*

if $n = 1$ or $n = 2$
  then return 1
  else
    return $fib1(n-1) + fib1(n-2)$

Analysis

(sketch) Looking at the computation tree, we see that every node has two children, and we go down n levels (in many branches); every node means one addition, so looks like about 2^n additions ...
The algorithm has exponential running time.

Example: Computation of nth Fibonacci number

Algorithm 2 (let's call it fib2) computes every $F(k)$, for $k = 1 \dots n$, iteratively (one after another), until we get to $F(n)$.

Algorithm *fib2(n)*

array of int $F[1 \dots n]$;
$F[1] \leftarrow 1$; $F[2] \leftarrow 1$;
for $k = 3 \dots n$
    do $F[k] \leftarrow F[k-1] + F[k-2]$;
return $F[n]$;

Analysis

(sketch) One addition for every $k = 1, \dots, n$. Uses an array of integers of length n.—The algorithm has linear running time and linear storage space.

Example: Computation of nth Fibonacci number

Algorithm 3 (let's call it fib3) computes $F(n)$ iteratively, like Algorithm 2, but using only 3 units of storage space.

Algorithm *fib3(n)*

int a, b, c;
$a \leftarrow 1$; $b \leftarrow 1$; $c \leftarrow 1$;
for $k = 3 \dots n$
    do $c \leftarrow a + b$;
        $a \leftarrow b$; $b \leftarrow c$;
return c;

Analysis

(sketch) Time: same as Algo 2. Uses 3 units of storage (called a, b, and c).—The algorithm has linear running time and constant storage space.

Example: Computation of nth Fibonacci number

Take-home message

- There may be more than one way of computing something.
- It is very important to use efficient algorithms.
- Efficiency is measured in terms of running time and storage space.
- In computational biology, inputs are often very large, therefore storage space is at least as important as running time.