Algorithms and Data Structures for Biology
1 April 2019 — Lab Session

Ugo Dal Lago Thomas Leventis

1 Solution: An Efficient Algorithm to Check Lists

Given a sorted list L of integers and an integer x, we want to check if x is in L in $O(\log(|L|)$. We use the following algorithm.

1.1 The Algorithm

```
Require: $L$ is a sorted list of integers, $x$ is an integer.
Ensure: Returns the Boolean stating whether $x$ is in $L$.
  $l \leftarrow 0$
  $r \leftarrow$ length of $L$
  while $l < r$ do
    $i \leftarrow (l + r)//2$
    if $L[i] = x$ then
      return True
    else if $L[i] < x$ then
      $l \leftarrow i + 1$
    else
      $r \leftarrow i$
    end if
  end while
  return False
```

1.2 The Implementation

The following Python function `InSorted` implements the above algorithm.

```
def InSorted(L,x):
        l = 0
        r = len(L)
        while l < r:
                i = (l+r)//2
                if L[i] == x:
                        return True
                elif L[i] < x:
                        l = i+1
                else:
                        r = i
        return False
```

Remark that in this case the algorithm is quite simple and its implementation is straightforward. This is not always the case, see for example the second assignment (18/03/2019).

1.3 Proof of Correctness

We will prove that once the variables l and r are defined, the following invariant holds.

$$x \text{ is in } L \text{ if and only if } x \text{ is in } L[l:r].$$

Additionally we check that $l \leq r$ and every index i with $l \leq i < r$ is a valid index of L.

Initialisation. At first we have $L = L[l:r]$ so the invariant holds, and the valid indices of L are exactly the is such that $l \leq i < r$.

Preservation. Let us assume that the invariant holds when entering the `while` loop. Remark that if $l < r$ then $i = (l+r)//2$ is such that $l \leq i < r$ and by hypothesis i is a valid index of L. We distinguish three cases:

- if $L[i] = x$ then x is both in L and in $L[l:r]$;
- if $L[i] < x$ then for all $j \leq i$, as L is sorted we have $L[j] \leq L[i]$, hence $L[j] < x$ and in particular $L[j] \neq x$; by hypothesis x is in L if and only if it is in $L[l:r]$, and we know x is not in $L[l:i+1]$ so x is in $L[l:r]$ if and only if it is in $L[i+1:r]$;
- otherwise $x < L[i]$ and by a similar reasoning we have that x is not in $L[i:r]$ so it is in $L[l:r]$ if and only if it is in $L[l:i]$.

Remark that in the last two cases the inequality $l \leq r$ is preserved, and in any case the set of indices between l and r is made smaller, so we never leave the scope of L.

Now our algorithm only returns `True` is $L[i] = x$ for some valid index i of L, in which case x is indeed in L. On the other hand it only returns `False` when $l \geq r$; we proved that x is in L if and only if it is in $L[l:r]$, but in this case $L[l:r]$ is empty so x is neither in $L[l:r]$ nor in L.

This concludes the proof that *if* our algorithm returns a result, then this result is correct. Next we will prove that it always returns a result.

1.4 Proof of Termination and Complexity

We want to prove that for any list L our algorithm terminates in at most $5\log_2(|L|)+6$ operations. To do so, we prove that whenever we reach the `while` instruction then the algorithm terminates in at most $5\log_2(r-l)+4$ operations. Remark that we already know we always have $l \leq r$.

Initialisation. If $l = r$ then $\log_2(r-l) = 0$, and the test $l < r$ fails (first operation) and the algorithm returns `False` (second operation).

Preservation. If $l < r$ then we enter the loop, i is defined and we check the equality $L[i] = x$ (three operations). Then there are three different cases.

- If $L[i] = x$ then the algorithm return `True`, for a total of four operations.
- If $L[i] < x$ then after five operations l has taken a new value $l' = (l+r)//2 + 1$ and we go back to the `while` instruction, and by hypothesis the program terminates after at most $5\log_2(r-l')+4$ more operations. Since $2(r-l') \leq r-l$ we have $\log_2(r-l')+1 \leq \log_2(r-l)$, hence $5\log_2(r-l')+9 \leq 5\log_2(r-l)+4$.
- Otherwise after five operations r has taken the value $r' = (l+r)//2$, hence $2(r'-l) \leq r-l$ and conclude in a similar way.

The algorithm always reaches the `while` loop for the first time after two operations, with $r - l = |L|$, so the algorithm always terminates in at most $5\log_2(|L|)+6$ operations.

1.5 Important Remark

Before proving the complexity of our algorithm we did not define what an "operation" is. In our reasoning we assume that an operation is the execution of a single line of code, but this is not very precise. For instance the computation of $(l+r)//2$ can be described not as one but as two operations, one addition and one division. But such a refinement would only change the constants (5 and 4), not the fact that the complexity is in $O(\log(|L|)$.

Nevertheless this approximation of the notion of operation is only valid when our lines of code always execute in constant time. The whole reason behind this work is that computing the simple expression `x in L` in Python is linear in $|L|$. Consider the following Python function `InList`.

```
def InList(L,x):
        return x in L
```

It is much shorter than our function `InSorted` and always execute a single line of code, but it is much less efficient on sorted lists. As another example note that slicing a list is *not* performed in constant time either.

This is where `cProfile` intervenes: you can check whether your theoretical complexity actually matches the real complexity of your program.

2 First Work in LaTeX

Following the example above, write a report in LaTeXon last week's second exercise (find the number of element o a list X which also belong to another list L).