Induction proofs and loop invariants

Algorithms & Data Structures (Module 7)

Rom Langerak

E-mail: r.langerak@utwente.nl

Februari 5, 2019

Outline of induction proofs

We want to prove some property $P(n)$ for all $n \geqslant 1$.

Base case: prove $P(1)$

Induction hypothesis: suppose $P(k)$ holds

Induction step: assume induction hypothesis $P(k)$, and prove that now $P(k+1)$ holds

According to the Principle of Mathematical induction:
now $P(n)$ holds for all $n \geqslant 1$.

Example

Prove $P(n): \quad 1 + 2 + \ldots + n = n(n+1)/2$ for all $n \geqslant 1$

Proof:

Base case: $1 = 1(1+1)$ so $P(1)$holds

Induction hypothesis: suppose $P(k)$, so $1 + 2 + \ldots + k = k(k+1)/2$

Induction step: $1 + 2 + \ldots + k + (k+1) =$ (because of induction hypothesis) $k(k+1)/2 + (k+1) = (k(k+1) + 2(k+1))/2 = (k+1)(k+2)/2$ so $P(k+1)$ holds

According to the Principle of Mathematical induction:
now $1 + 2 + \ldots + n = n(n+1)/2$ for all $n \geqslant 1$

Variations

base case: the base case can be any integer b

multiple base cases: prove $P(b),\ \ P(b+1), \ldots, P(b+a)$, as base cases, often necessary with recursive equations (see next example)

strong induction: assume $P(i)$ holds for all $1 \leqslant i \leqslant k$, and from this prove $P(k+1)$ (see next example)

Example (from lecture 1)

Fibonacci:

$$\begin{aligned} Fib(0) &= 0 \\ Fib(1) &= 1 \\ Fib(n{+}2) &= Fib(n{+}1) + Fib(n) \text{ for } n \geqslant 0 \end{aligned}$$

The # arithmetic steps $T_{fibRec}(n)$ for recursive Fibonacci computation:

$$\begin{aligned} T_{fibRec}(0) &= 0 \\ T_{fibRec}(1) &= 0 \\ T_{fibRec}(n{+}2) &= T_{fibRec}(n{+}1) + T_{fibRec}(n) + 3 \text{ for } n \geqslant 0 \end{aligned}$$

An induction proof

Prove that $T_{fibRec}(n) = 3{\cdot}Fib(n{+}1) - 3$

Base steps:
$(n = 0)$: $T_{fibRec}(0) = 3{\cdot}Fib(1) - 3 = 0$ so ok
$(n = 1)$: $T_{fibRec}(1) = 3{\cdot}Fib(2) - 3 = 0$ so ok

Induction step:

Suppose it holds for all $0 \leqslant i \leqslant k$. Then:

$T_{fibRec}(k+1) =$ (def)
$T_{fibRec}(k) + T_{fibRec}(k-1) + 3 =$ (ind. hyp.)
$3{\cdot}Fib(k+1) - 3 + 3{\cdot}Fib(k) - 3 + 3 =$
$3{\cdot}(Fib(k+1) + Fib(k)) - 3 = 3{\cdot}Fib(k+2) - 3$ so ok.

Fibonacci – asymptotic complexity

Using induction we have proven that $T_{fibRec}(n) = 3 \cdot Fib(n+1) - 3$

How large is $Fib(n)$? Can we give upper/lower bounds?

It follows (by induction) that $2^{(n-2)/2} \leqslant Fib(n)$ for $n \geqslant 2$

But, this means that the complexity of *fibRec* is exponential:

$$T_{fibRec}(n) \in \Omega((\sqrt{2})^n)$$

Linear search

Input: array E with n entries and item K to be looked up

Output: does K occur in the array E?

```
def linsearch(E, n, K):
    i = 0
    found = False
    while i<n and not found:
        found = (E[i]==K)
        i = i+1
    return found
```

Why is this correct? What should it do?

Loop invariants

General picture:

```
Initialisation
while Condition
    Action1
    Action2
    ...
    ActionN
```

We now look for an invariant `Inv` such that:

- `Inv` holds after initialisation
- if `Inv` holds befor the action block: `Inv` holds after the action block
- {`Inv and not Condition`} is what we want!

Loop invariants-2

So the general picture:

```
Initialisation  #Inv
while Condition
    Action1
    Action2
    ...
    ActionN   #Inv
#Inv and not Condition
```

Can you see the link with induction?

A factorial function

```
def fact1(n):
    k=1
    t=1
    while k<n:
        k=k+1
        t=t*k
    return t
```

Correctness

```
def fact1(n):
    k=1
    t=1           # t=k!
    while k<n:
        k=k+1     # t=(k-1)!
        t=t*k     # t=k!
    return t      # t=k! and k=n so t=n!
```

Another factorial function

```
def fact2(n):
    k=n
    t=1
    while k!=0:
        t=t*k
        k=k-1
    return t
```

Correctness

```
def fact2(n):
    k=n
    t=1             # t=n!/k!
    while k!=0:
        t=t*k       # t=n!/(k-1)!
        k=k-1       # t=n!/k!
    return t        # t=n!/k! and k==0 so t=n!
```

Linear search

```
def linsearch(E, n, K):
    i = 0
    found = False
    while i<n and not found:
        found = (E[i]==K)
        i = i+1
    return found
```

Linear search - correctness

```
i = 0
found = False                  #found==(K in E[0,i) )
while i<n and not found:
    found = (E[i]==K)          #found==(K in E[0,i+1) )
    i = i+1                    #found==(K in E[0,i) )

        # found==(K in E[0,i) ) and (i=n or found)
        # so: found== (K in E[0,n) )
```

Note that $[0, i)$ is intended in the Python sense, i.e. excluding i