CS170–Fall 2014 — Solutions to Homework 1

Quoc Thai Nguyen Truong, SID 24547327, `cs170-ig`

September 7, 2014

Collaborators: Tho Giai Truong

1. Getting started

(1 page)

(a) I understand the course policies

(b) No, this is not allowed!

2. Proof of correctness

(2 pages)

To show the correctness of the algorithm, we want to show that:

$$P(x) = \sum_{k=0}^{n} a_k x^k$$

At line 3 where the execution the value of y, we have an invariant:

$$y_{n-(i+1)} = c * y_{n-i} + a_{n-(i+1)}$$

Base case: If n = 0, then we have:

$$Horner((a_0), c) = a_0$$

We know that at line 1, $y_n = a_n$, so if $n = 0$, then $y_0 = a_0$ and it will ignore line 2 and line 3, and go to line 4 to return $y = a_0$. Therefore the algorithm is true for n = 0.

Proving the Invariant:
We can see that if $n > 0$, then we will have a list (more than 1 element) of $(a_0, a_1, ..., a_n)$, so line 2 and 3 will be executed.

Using the invariant, we can see that

When $i = 0$ then $y_{n-1} = c * y_n + a_{n-1}$ where $y_n = a_n$ (from the line 1)
When $i = 1$ then $y_{n-2} = c * y_{n-1} + a_{n-2}$ where y_{n-1} is from the previous execution (above)
When $i = 2$ then $y_{n-3} = c * y_{n-2} + a_{n-3}$ where y_{n-2} is from the previous execution (above)
.
.
.
until when $i = n - 1$ then $y_0 = c * y_1 + a_0$

Clearly, after each iteration of the loop, a constant $'c'$ will be multiply with (previous) y_{n-i} and a coefficient $a_{n-(i+1)}$ will be added. Therefore, a new y which is $y_{n-(i+1)}$ is created.

Hence, after the complete loop is executed, we have :

$$y = a_0 + c(a_1 + c(a_2 + ...c(a_n))...)$$

Therefore, we have an induction hypothesis:

$$y_k = a_k + c(a_{k+1} + c(a_{k+2} + ...c(a_n))...)$$

where $0 <= k <= n - (i+1)$, and $0 <= i < n$

Prove by induction:

Base: if $i = 0$ then $y_{n-1} = c * y_n + a_{n-1}$ where $y_n = a_n$ (from the line 1)

Induction Hypothesis:

$$y_k = a_k + c(a_{k+1} + c(a_{k+2} + ...c(a_n))...)$$

where $0 <= k <= n - (i+1)$, and $0 <= i < n$

Induction step:

We want to show that $y_0 = a_0 + c(a_1 + c(a_2 + ...c(a_n))...) = P(c) = a_n c^n + ... + a_2 c^2 + a_1 c + a_0$

From line 3, when $i = n - 1$ we have:

$$y_0 = c y_1 + a_0$$

where $y_1 = a_1 + c(a_2 + c(a_3 + ...c(a_n))...)$

So

$$y_0 = c(a_1 + c(a_2 + c(a_3 + ...c(a_n))...)) + a_0$$

Which is equal to

$$y_0 = a_0 + c(a_1 + c(a_2 + ...c(a_n))...) = P(c) = a_n c^n + ... + a_2 c^2 + a_1 c + a_0$$

Conclusion:

Since $y_0 = P(c)$ by proving induction, the invariant guarantees this algorithm will produce the correct output. Therefore the algorithm is true for evaluating the polynomial P at the value x = c

3. Prove this algorithm correct

(3 pages)

(a) Prove that c never goes negative:
At the line 1, we know that $c = 0$ and $v = null$.
The algorithm is executed in the loop in lines 2,3, and 4.
1st iteration will be the case $c = 0$ at line 3, and set $v = A[0]$. Then c will be increase by 1 which give c = 1 since $v = A[0]$ at line 4.
Now $c > 0$, so the next iteration, the algorithm will be executed at line 4, which will be the case of $v = A[i]$ or $c \neq A[i]$. If $v = A[i]$ then c will be increment by one which is $c >= 0$. Otherwise, $v \neq A[i]$ c will be decrease by one which is $c >= 0$.
The next iteration, if $c = 0$ again, we go back to the very first iteration at the 3 which set $v = A[i]$ and then increasing c by 1 ($c = 1$) which give $c >= 0$ since $v = a[i]$ at line 4.

Hence: there are 3 cases
If $c = 0$ then $c = 1$ which is $c >= 0$
If $v = A[i]$ then $c = c + 1$ which is $c >= 0$
Else we know that $c \neq 0$ (if $c = 0$, it would have go to the first case of $c = 0$) which is $c > 0$, and c is decease by one, so the minimum value of c after decrease the c value in this "Else" case is 0. Therefore $c >= 0$

Conclusion:
We see that these 4 cases give us $c >= 0$. Therefore, **c never goes negative**

(b) **Invariant**: At the start of any iteration of the loop, the elements of $A[0 :: i - 1]$ can be partitioned into two groups: a group U_i of at least c instances of the value v, and a group P_i of elements that can be paired off so that the two elements in each pair differ.

Base case: Group U and P are empty at the start of the loop. The start of the loop, $i = 0$, line 3 is executed, v will be set to $A[0]$, then $c = 1$ (at line 4, since $v = A[0]$). Therefore,$U = \{v\}$ (since $U \bigcup \{v\}$), and

P is still empty $\rightarrow$ the hypothesis is true for base case

Inductive Hypothesis: At the start of any iteration of the loop, the elements of $A[0 :: i-1]$ can be partitioned into two groups: a group U_i of at least c instances of the value v, and a group P_i of elements that can be paired off so that the two elements in each pair differ.

Induction Step:

There are 3 cases, and we want to show that U_{i+1} and P_{i+1} are also true for these 3 cases.

case 1: $c = 0$

Since $c = 0$,we know that U_i is empty, and P_i contains some pairs or no pairs. In the execution, at line 3, v will be set to $A[i]$, then $c = 1$ (at line 4). Therefore, $U_{i+1} \bigcup \{v\}$, and P_{i+1} is not changed $\rightarrow$ the hypothesis is true for this case

case 2: $c > 0$ and $v = A[i]$

Since $c > 0$, we know that U_i is not empty, and it has at least c element(s) of value v, and P_i contains some pairs that two elements in each pair differ or no pairs. In the execution, at line 4, c will be increased by 1 (since $v = A[i]$).Therefore, U_{i+1} will append v so that $U_{i+1} \bigcup \{v\}$, and P_{i+1} won't change $\rightarrow$ the hypothesis is true for this case

case 3: $c > 0$ and $v \neq A[i]$

Since $c > 0$, we know that U_i is not empty, and it has at least c element(s) of value v, and P_i contains some pairs that two elements in each pair differ or no pairs. In the execution, at line 4, c will be decreased by 1 (since it's "else" case). Therefore, U_{i+1} will lost one element of v, and P_{i+1} will add a pair of two element that are $(v, A[i]) \rightarrow$ the hypothesis is true for this case

(c) **Prove why the invariant from part (b) implies that the algorithm is correct**

Prove by contradiction:
Assume that there is a list A that has a majority elements v (more than 50% in A), and let n be the number of v in list A, let m be the half of total elements of list A. After the algorithm is terminated, group U is empty, and P contains some pairs that two elements in each pair differ

Prove:

Since n is the number of v in list A and appear more than 50% in list A, we know that $n > m$. Base on the Pigeonhole principle, with $n > m$, then at least one hole must contain more than one value of v. It tells that the value of c must be at least 1 so $c > 0$, since the $c > 0$ we can say that group U contain at least c instances of the value v. This is a contradiction because we assume that group U is empty. Therefore, group U must not be empty, and contains at least c instances of the value v.

Conclusion: We see that group U must not be empty, and contains at least c instances of the value v if list A has majority element of value v. Therefore, it's satisfy the invariant, so **the invariant implies that the algorithm is correct**.