CSC236 Tutorial #7

Sample Solutions

1. Let L be a list of integers.
Let $L[p:q]$ be an nonempty slice, i.e., $0 \leq p < q \leq len(L)$.
We say that $L[p:q]$ is *unimodal* iff there is a natural number m such that

   (a) $p \leq m < q$;

   (b) $L[p:m+1]$ strictly increasing;

   (c) $L[m:q]$ is strictly decreasing.

   Furthermore, such a number m, if it exists, is call the *mode* of $L[p:q]$.
   Since L is the same as $L[0:len(L)]$, we also say that L is unimodal if $L[0:len(L)]$ is unimodal.

   Give a proof of correctness for the program below with respect to the given specification.
   Precondition: A is a list of integers, $0 \leq p < q \leq len(A)$, $A[p:q]$ is unimodal.
   Postcondition: Return the maximum element of $A[p:q]$.

```
    def Max(A, p, q):
1.      if p + 1 == q:
2.          return A[p]
3.      else:
4.          m = ⌊(p+q)/2⌋
5.          if A[m − 1] < A[m]:
6.              return Max(A, m, q)
7.          else:
8.              return Max(A, p, m)
```

   Hint 1: Notice that the maximum element of a unimodal slice occurs at its mode.
   Hint 2: For any integers a, b such that $a + 1 < b$,

$$a < \lfloor \frac{a+b}{2} \rfloor < b.$$

   Solution:
   Let $P(n)$ denote the assertion that if A is a list of integers, $0 \leq p < q \leq len(A)$, $A[p:q]$ is unimodal, and $n = q - p$, then $Max(A, p, q)$ terminates and returns the maximum element of $A[p:q]$.
   Note that we are using $q - p$ (length of the slice being considered) as size of input.

   We prove that for all integers $n \in \mathbb{N}, n \geq 1$, $P(n)$ holds.

   Base Case: Let $n = 1$. Then $q - p = 1$, or equivalently, $p + 1 = q$.
   Thus $A[p:q]$ is a slice of one element, and $A[p]$ is its maximum element.
   By Line 2, $A[p]$ is returned as wanted.

Induction Step: Let $n \in \mathbb{N}, n > 1$. Suppose for all $j \in \mathbb{N}$, $1 \leq j < n, P(j)$ holds. **[IH]**
WTP: $P(n)$ holds.

Since $q - p = n > 1$, Lines 4-8 run.
There are two cases to consider.

Case 1: $A[m-1] < A[m]$.
Then by Line 6, $Max(A, m, q)$ is called and returned.
By Line 4 and Hint 2, $p < m < q$, and so $1 \leq q - m < q - p$.
By the IH, $Max(A, m, q)$ terminates and returns the maximum element of $A[m : q]$.
Since $A[m-1] < A[m]$, the mode of $A[p : q]$ must equal the mode of $A[m : q]$.
So the maximum element of $A[p : q]$ equals the maximum element of $A[m : q]$.
Therefore $Max(A, p, q)$ terminates and returns the maximum element of $A[p : q]$, as wanted.

Case 2: $A[m-1] > A[m]$.
Similar to Case 1; left as an exercise to the reader.

2. For each of the following programs:

   - give an appropriate loop invariant (LI) for the loop;
   - Use your LI and the loop exit condition to prove partial correctness;
   - Define an appropriate loop measure m which you can use for proving the termination of the loop.

   (a) **Precondition:** x is a non-empty string.
   Postcondition: returns x except with all the characters in reverse order.

```
    def backwards_string(x):
1.      i = 0; word = ''
2.      while i < len(x):
3.          word = x[i] + word
4.          i = i + 1
5.      return word
```

   Solution:

   - Let $LI(k)$ denote the assertion that if the loop is executed at least k times, then
     (a) $0 \leq i_k \leq len(x)$;
     (b) $word_k$ is the reverse of $x[0 : i_k]$.
   - Suppose the precondition holds and the program terminates. Since the program terminates, the loop is executed a finite number of times, say t.

     By part (a) in LI, $i_t \leq len(x)$.
     By the exit condition we have $i_t \geq len(x)$. So $i_t = len(x)$.
     By part (b) in LI, $word_t$ is the reverse of $x[0 : len(x)]$.
     Since $x[0 : len(x)] = x$, $word_t$ is the reverse of x.
     Therefore, in Line 5, the reverse of x is returned.
   - $m_k = len(x) - i_k$.

   (b) **Precondition:** *word* is a non-empty string of small alphabetic characters.
   Postcondition: returns true iff *word* is palindrome.

```
    def Is_palindrome(word) :
1.      start_idx = 0; end_idx = len(word) - 1; result = True;
2.      while start_idx < end_idx:
3.          if word[start_idx] != word[end_idx] :
4.              result = False
5.          start_idx = start_idx + 1
6.          end_idx = end_idx - 1
7.      return result
```

   Solution:

   - Let $LI(k)$ denote the assertion that if the loop is executed at least k times, then
     (a) $start\_idx_k + end\_idx_k = len(word) - 1$;
     (b) For all $i_k \in \mathbb{N}$ with $i_k < start\_idx_k$, $result_k = True$ iff $word[i_k] = word[len(word)-1-i_k]$.
   - Suppose the precondition holds and the program terminates. Since the program terminates, the loop is executed a finite number of times, say t.

By exit condition we have $start\_idx_t \geq end\_idx_t$.
By part (a) in LI, $start\_idx_t + end\_idx_t = len(word) - 1$. So $start\_idx_t \geq (len(word) - 1)/2$.

Case 1: $result_t$ = False.
Then, by part (b) in LI, there exists $i \in \mathbb{N}$ such that $word[i] \neq word[len(word) - 1 - i]$. Then $word$ is not palindrome.
On the other hand, $Is\_palindrome(word)$ returns $False$, and so the postcondition holds.
Case 2: $result_t$ = True.
Then for all $i \in \mathbb{N}, i \leq (len(word) - 1)/2$, we have $word[i] = word[len(word) - 1 - i]$. Therefore, $word$ is palindrome.
On the other hand, $Is\_palindrome(word)$ returns $True$, and so the postcondition holds.

- $m_k = end\_idx_k$.

3. A *majority element* in a list is an element that appears in at least half (rounded up) of the list locations. Consider the following algorithm that finds a majority element in an array, if one exists (if one doesn't exist the algorithm returns an arbitrary wrong answer). Write a detailed proof that the algorithm is partially correct.

Precondition: A is a list of integers and $1 \leq len(A)$.
Postcondition: If A has an element that appears in more than half of the cells of A, then that element is equal to m at return.

```
    def Majority(A):
1.      c = 1
2.      m = A[0]
3.      i = 1
4.      while i < len(A):
5.          if c == 0:
6.              m = A[i]
7.              c = 1
8.          else if A[i] == m:
9.              c = c + 1
10.         else:
11.             c = c - 1
12.         i = i + 1
13.     return m
```

Solution: Before, we start with the formal solution, the key to solving this problem easily is to make the following three observations:

- The number of times that m occurs in A from 1 to i is at least c.
- The number of times that m occurs in A from 1 to i, minus c, is no greater than $(i-c)/2$.
- For all x (such that $x \neq m$), the number of times x occurs in A from 1 to i is no greater than $(i-c)/2$.

These three statements summarize the key reasons behind the correctness of this algorithm in discovering the majority element of an array. The proper loop invariant for this problem (the hardest part of the solution to discover) is basically a formalization of these three statements and some basic facts about the ranges of the variables i and c.

Let $Q(A, k, m)$ denote the number of times value m appears in $A[0:k]$. Then, we can rewrite the post condition as:
If exists x such that $Q(A, len(A), x) > \lfloor len(A)/2 \rfloor$, then $m = x$, where m here refers to the value of m at the return point.

Loop Invariant: Let $LI(k)$ denote the assertion that if the loop is executed at least k times, then

$$0 < i_k \leq len(A) \wedge 0 \leq c_k \leq len(A) \qquad LI-1$$
$$Q(A, i_k, m_k) \geq c_k \qquad LI-2$$
$$Q(A, i_k, m_k) - c_k \leq \lfloor (i_k - c_k)/2 \rfloor \qquad LI-3$$
$$\text{for all } x \text{ if } x \neq m_k \text{ then } Q(A, i_k, x_k) \leq \lfloor (i_k - c_k)/2 \rfloor \qquad LI-4$$

Basis (precondition implies LI):
Precondition: $1 \leq len(A)$
1. $c_0 = 1$
2. $m_0 = A[0]$
3. $i_0 = 1$
WTP: $LI(0)$

By Line (1), we have $c_0 = 1$. By Line (2), we have $m_0 = A[0]$. Thus, $Q(A, i_0, m_0) = Q(A_0, 1, A[0]) = 1$. By Line (3), we have $i_0 = 1$. Therefore, $LI(0)$ holds; LI-1 holds based on the values of i_0 and length, and LI-2 and LI-3 hold based on the values of c_0, i_0, and $Q(A, i_0, m_0)$, and LI-4 holds vacuously.

Induction Step: Let k be an arbitrary natural number, and assume that $LI(k)$ holds. **[IH]**
WTP: $LI(k+1)$ holds.

(Note: For brevity, and more clarity, I have assigned new line numbers to each path)

Path 1 (LI for the first if statement):
Assuming $LI(k)$
1. $(i \leq len(A) - 1)$
2. $(c == 0)$
3. $m = A[i]$
4. $c = 1$
5. $i = i + 1$
WTP: $LI(k+1)$

By Line 2, we know $c_k = 0$. This fact combined with the loop invariant $LI(k)$ (LI-2 and LI-3) immediately implies that $0 \leq Q(A, i_k, m_k) \leq \lfloor i_k/2 \rfloor$.
By line 5, we know $i_{k+1} = i_k + 1$. Since $i_k > 0$, we have $i_{k+1} > 0$. Also, since $i_k \leq len(A) - 1$, we have $i_{k+1} \leq len(A)$. Since $c_{k+1} = 1$ (by Line 4), we can conclude that LI-1 in $LI(k+1)$ holds.

By Line 3, we know $m_{k+1} = A[i_k]$. This means that $Q(A, i_{k+1}, m_{k+1}) = Q(A, i_k + 1, m_{k+1}) \geq 1 = c_{k+1}$ (at the very least, the element at index i_k is equal to m_{k+1}).
Therefore, LI-2 in $LI(k+1)$ holds.

There are two possibilities for the value of m_{k+1}:
Case 1: $m_{k+1} = m_k$.
We have

$$\begin{aligned} Q(A, i_{k+1}, m_{k+1}) - c_{k+1} &= Q(A, i_k + 1, m_k) - 1 \\ &= Q(A, i_k, m_k) \leq \lfloor i_k/2 \rfloor \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $i_{k+1} = i_k + 1$ and $c_{k+1} = 1$.
Therefore, LI-3 in $LI(k+1)$ holds.

If $x \neq m_{k+1}$, then $x \neq m_k$ (since $m_k = m_{k+1}$). Therefore,

$$\begin{aligned} Q(A, i_{k+1}, x) &= Q(A, i_k + 1, x) \\ &= Q(A, i_k, x) \\ &\leq \lfloor (i_k - c_k)/2 \rfloor \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $i_{k+1} = i_k + 1$ and $c_{k+1} = 1$.
Therefore, LI-4 in $LI(k+1)$ holds.

Case 2: $m_{k+1} \neq m_k$.
We have

$$\begin{aligned} Q(A, i_{k+1}, m_{k+1}) - c_{k+1} &= Q(A, i_{k+1}, m_{k+1}) - 1 \\ &= Q(A, i_k, m_{k+1}) \\ &\leq \lfloor (i_k - c_k)/2 \rfloor \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The inequality holds by LI-4 in $LI(k)$ and the fact that $m_{k+1} \neq m_k$.
Therefore, LI-3 in $LI(k+1)$ holds.
If $x \neq m_{k+1}$, then x may or may not be equal to m_k.
If it is not, then the argument from Case 1 still applies and we can show that LI-4 in $LI(k+1)$ holds.
If $x = m_k$, then

$$\begin{aligned} Q(A, i_{k+1}, x) &= Q(A, i_{k+1}, m_k) \\ &= Q(A, i_k, m_k) \\ &\leq \lfloor i_k/2 \rfloor \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $i_{k+1} = i_k + 1$ and $c_{k+1} = 1$.
Therefore, LI-4 in $LI(k+1)$ holds.

Path 2 (LI for the second if statement):
Assuming $LI(k)$
1. $(i \leq len(A) - 1)$
2. $(c \mathrel{!}= 0)$
3. $(A[i] == m)$
4. $c = c + 1$
5. $i = i + 1$

WTP: $LI(k+1)$

By Line 2, we know $c_k \neq 0$. And, by Line 4, we know that $c_{k+1} = c_k + 1$. Also, m doesn't change on this path, therefore $m_{k+1} = m_k$.
By Line 5, we know $i_{k+1} = i_k + 1$. Since $i_k > 0$, we have $i_{k+1} > 0$. Also, since $i_k \leq len(A) - 1$, we have $i_{k+1} \leq len(A)$. Since $c_{k+1} = c_k + 1 \geq 1$, we can conclude that LI-1 in $LI(k+1)$ holds.
By Line 3, we know $m_{k+1} = m_k = A[i_k]$. This means that

$$\begin{aligned} Q(A, i_{k+1}, m_{k+1}) &= Q(A, i_k + 1, m_k) \\ &= Q(A, i_k, m_k) + 1 \\ &\geq c_k + 1 \\ &= c_{k+1}. \end{aligned}$$

The inequality comes from LI-2 of $LI(k)$.
Therefore, LI-2 of $LI(k+1)$ holds.
Since $A[i_k] = m_k = m_{k+1}$, we have

$$\begin{aligned} Q(A, i_{k+1}, m_{k+1}) - c_{k+1} &= Q(A, i_{k+1}, m_k) - c_k - 1 \\ &= Q(A, i_k, m_k) - c_k \\ &\leq \lfloor (i_k - c_k)/2 \rfloor \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $i_{k+1} = i_k + 1$ and $c_{k+1} = c_k + 1$.
Therefore, LI-3 in $LI(k+1)$ holds.
If $x \neq m_{k+1}$, then $x \neq m_k$ (since $m_k = m_{k+1}$). Therefore

$$\begin{aligned} Q(A, i_{k+1}, x) &= Q(A, i_k + 1, x) \\ &= Q(A, i_k, x) \\ &\leq \lfloor (i_k - c_k)/2 \rfloor \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $i_{k+1} = i_k + 1$ and $c_{k+1} = c_k + 1$.
Therefore, LI-4 in $LI(k+1)$ holds.

Path 3 (LI for the else statement):
Assuming $LI(k)$
1. $(i \leq len(A) - 1)$
2. $(c \mathrel{!}= 0)$
3. $(A[i] \mathrel{!}= m)$
4. $c = c - 1$
5. $i = i + 1$

WTP: $LI(k+1)$

By Line 2, we know $c_k \neq 0$. And, by Line 4, we know that $c_{k+1} = c_k - 1$. And, m doesn't change on this path, therefore $m_{k+1} = m_k$.
By Line 5, we know $i_{k+1} = i_k + 1$. Since $i_k > 0$, we have $i_{k+1} > 0$. Also, since $i_k \leq len(A) - 1$, we have $i_{k+1} \leq len(A)$. Since $c_{k+1} = c_k - 1 \geq 0$ ($c_k \geq 0$ and $c_k \neq 0$ imply $c_k > 0$), we can conclude that LI-1 in $LI(k+1)$ holds.
By Line 3, we know $m_k \neq A[i_k]$. Also, $m_{k+1} = m_k$. This means that

$$\begin{aligned} Q(A, i_{k+1}, m_{k+1}) &= Q(A, i_k + 1, m_k) \\ &= Q(A, i_k, m_k) \\ &\geq c_k = c_{k+1} + 1. \end{aligned}$$

The inequality comes from LI-2 in $LI(k)$. Therefore, LI-2 in $LI(k+1)$ holds.
Since $A[i_k] \neq m_k$ and $m_k = m_{k+1}$, we have

$$\begin{aligned} Q(A, i_{k+1}, m_{k+1}) - c_{k+1} &= Q(A, i_k + 1, m_k) - (c_k - 1) \\ &= Q(A, i_k, m_k) - c_k + 1 \\ &\leq \lfloor (i_k - c_k)/2 \rfloor + 1 \\ &= \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $\lfloor (i_{k+1} - c_{k+1})/2 \rfloor = \lfloor (i_k + 1 - c_k + 1)/2 \rfloor = \lfloor (i_k - c_k)/2 \rfloor + 1$.
Therefore, LI-3 in $LI(k+1)$ holds.

If $x \neq m_{k+1}$ then (since $m_{k+1} = m_k$),

$$\begin{aligned} Q(A, i_{k+1}, x) &= Q(A, i_k + 1, x) \\ &= Q(A, i_k, x) \\ &\leq \lfloor (i_k - c_k)/2 \rfloor \\ &< \lfloor (i_{k+1} - c_{k+1})/2 \rfloor. \end{aligned}$$

The last equality holds since $\lfloor (i_{k+1} - c_{k+1})/2 \rfloor = \lfloor (i_k + 1 - c_k + 1)/2 \rfloor = \lfloor (i_k - c_k)/2 \rfloor + 1$.
Therefore, LI-4 in $LI(k+1)$ holds.

Path 4 (loop exist and and LI imply postcondition):
Suppose the precondition holds and the program terminates. Since the program terminates, the loop is executed a finite number of times, say t.

Assuming $LI(t)$
1. $(i_t > len(A) - 1)$
2. return m
WTP: {if exists x such that $Q(A, len(A), x) > \lfloor len(A)/2 \rfloor$ then $m_t = x$}

When we exit the loop, the combination of the loop invariant LI-1 and the negation of the loop condition implies that $i_t = len(A)$. By LI-2 and LI-3 in $LI(t)$, we have $c_t \leq Q(A, len(A), m_t) \leq \lfloor (len(A) + c_t)/2 \rfloor$.
But, also, LI-4 of $LI(t)$ implies that for all x, if $x \neq m_t$ then,

$$Q(A, len(A), x) \leq \lfloor (len(A) - c_t)/2 \rfloor \qquad (\star)$$

Now assume that the array has a majority value (we are assuming the premise of the postcondition to be true; if it is false, then the postcondition vacuously holds). Let us call this value v. Then we know that (by the definition of majority), $Q(A, len(A), v) > \lfloor len(A)/2 \rfloor \geq \lfloor len(A) - c_t)/2 \rfloor$.
This mean that we cannot have $v \neq m_t$, since we would have a contradiction with $(\star)$.
Therefore, $v = m_t$, and the algorithm correctly returns the majority element if it exists.