MA214 Coursework Q2

Candidate No: 46724

April 2023

1 Question 2 - Sum Free Lists

We prove correctness of the function SumFreeLists for the inputs:

$$list1 = A, list2 = B, list3 = C.$$

Inner Loop Invariant: At the start of each iteration in the while loop for fixed i and indices j,k, the two conditions hold:

1. **There are no such elements a,b,c with a = A[i], b in the subarray B[0,1,..., j-1], c in C such that a + b + c = 0.**
2. **There are no such elements a,b,c with a = A[i], b in B, c in the subarray C[k+1, ..., len(C) - 1] such that a + b + c = 0**

Initialisation: At initialisation, $j = 0, k = len(C) - 1$.

Note that the subarray $B[: j] = B[0, 1, ..., j - 1]$ is empty for $j = 0$, and the subarray $C[k + 1 :] = C[k + 1, ...len(C) - 1]$ is empty for $k = len(C) - 1$.

Therefore ,there are no such $a = A[i], b$ in the subarray $B[0, 1, ..., j - 1]$, c in C with $a + b + c = 0$ (since $B[0, 1, ..., j - 1]$ is empty) and so the first condition of the loop invariant holds.

Also, there are no such $a = A[i]$, b in B, c in the subarray $C[k + 1, ..., len(C) - 1]$ with $a + b + c = 0$ since the subarray $C[k + 1, ..., len(C) - 1]$ is empty, and so the second condition of the loop invariant holds at initialisation.

Therefore, the loop invariant holds at initialisation.

Maintenance:
Suppose that the loop invariant holds before some iteration for given values of j and k. We want to show that the loop invariant will still hold before the next iteration.

We only proceed to the next iteration if either $A[i] + B[j] + C[k] < 0$ or $A[i] + B[j] + C[k] > 0$.

Case 1: If $A[i] + B[j] + C[k] < 0$, the algorithm updates j by increasing it by 1.

Since k remains the same, we have that at the start of the next iteration, the second condition of the loop invariant that there are no such elements a, b, c where $a = A[i]$, b in B and c in $C[k+1, ..., len(C)-1]$ such that $a + b + c = 0$ still holds true from the loop invariant at the start of the previous iteration.

To prove that the first condition of the loop invariant still holds at the start of the next iteration, we want to prove that there are now no solutions to $a + b + c = 0$, where $a = A[i]$, b in $B[0, 1, ..., j]$ and c in C, Since we had from the loop invariant at the start of the previous iteration that there were no solutions for $a = A[i]$, b in $B[0, 1, ..., j-1]$ and c in C, we can prove the required result by showing that there are no solutions to $a + b + c = 0$ for $a = A[i], b = B[j]$ and for any c in C.

We consider all elements c in C by looking at the cases when $c = C[k']$ for $k' < k, k' = k$ and $k' > k$ (for the given index k at the while loop iteration.)

We know that in the current loop iteration for indices $j, k : A[i] + B[j] + C[k] < 0$, and so there are no solutions to $a + b + c = 0$ for $a = A[i]$, $b = B[j]$, $c = C[k]$.

For any $k' < k : C[k'] < C[k]$ since C has been sorted prior to the while loop. Therefore:

$$A[i] + B[j] + C[k'] < A[i] + B[j] + C[k] < 0$$

and so there are no solutions to $a+b+c = 0$ for $a = A[i]$, $b = B[j]$, c in $C[0, 1, ...k-1]$

For the case $k' > k$, if $k = len(C) - 1$, the subarray $C[k+1, ..., len(C)-1]$ is empty so there are trivially no solutions.
If however $k < len(C) - 1$, then there must have been some $j' \leq j$ such that

$A[i] + B[j'] + C[k'] > 0$ at an earlier iteration for the index k' to have now decreased to k.

Since B was also sorted prior to the while loop, we have that $B[j] \geq B[j']$ for $j \geq j$ and thus:

$$A[i] + B[j] + C[k'] \geq A[i] + B[j'] + C[k'] > 0$$

and so there are no solutions to $a + b + c = 0$
for $a = A[i]$, $b = B[j]$, c in $C[k + 1 :] = C[k + 1, ..., len(C) - 1]$

Therefore, we have shown that there are no solutions to $a + b + c = 0$ for $a = A[i], b = B[j]$ and for any c in C. Combining this with the loop invariant at the start of the previous iteration, we have that at the start of the next iteration, there are no solutions to $a + b + c = 0$, where $a = A[i]$, b in $B[0, 1, ..., j]$ and c in C, and thus the first condition of the loop invariant also holds at the start of the next iteration.

We use a similar argument as above to show the loop invariant holds in the case where $A[i] + B[j] + C[k] > 0$.

Case 2 If $A[i] + B[j] + C[k] > 0$, we update k by decreasing it by 1.

Since j remains the same at the start of the next iteration, we have that the first condition of the loop invariant still holds: no elements $a = A[i], b$ in the subarray $B[0, 1, ..., j - 1]$ and c in C such that $a + b + c = 0$.

To show the second condition of the loop invariant still holds at the start of the next iteration, we need to show that no such elements a, b, c such that $a + b + c = 0$, where $a = A[i], b$ in B and c in the subarray $C[k :] = C[k, ..., len(C) - 1]$.
Using the loop invariant that we assume holds at the start of the prior iteration, this is equivalent to showing that there are no solutions to $a + b + c = 0$ for $a = A[i]$, for any b in B and $c = C[k]$.

To show that the above statement holds for any b in B, we consider the cases where $b = B[j']$ for $j' > j, j' = j, j' < j$.
(where j is the index value in the current iteration of the while loop)

We know that for the current index j, there are no solutions to $a + b + c = 0$ for $a = A[i], b = B[j], c = C[k]$ since we are in the case $A[i] + B[j] + C[k] > 0$.

For all indices $j' > j$, we have that $B[j'] > B[j]$ since B is sorted before the while loop begins and so:

$$A[i] + B[j'] + C[k] > A[i] + B[j] + C[k] > 0$$

and therefore there are no solutions to $a+b+c = 0$ for $a = A[i]$, b in $B[j+1:]$, $c = C[k]$.

For all $j' < j$: we must have had that there was some $k' \geq k$ such that: $A[i] + B[j'] + C[k'] < 0$ at an earlier iteration for the index j' to have increased to the current index j .
(unless $j = 0$ in which case $B[0 : j] = B[0, ..., j-1]$ is empty and thus there are trivially no solutions.)

Since C is sorted before the while loop begins, we have that $C[k] \leq C[k']$ and so:

$$A[i] + B[j'] + C[k] \leq A[i] + B[j'] + C[k'] < 0.$$

Therefore, we have that there are no solutions to $a + b + c = 0$ for $a = A[i]$, b in $B[0 : j]$ and $c = C[k]$.

W have proved that there are no solutions to $a + b + c = 0$ for $a = A[i]$, any b in $B, c = C[k]$, and so using the loop invariant at the start of the prior iteration, we have that at the start of the next iteration, no such elements a, b, c such that $a + b + c = 0$, where: $a = A[i]$, b in B and c in the subarray $C[k, ..., len(C) - 1]$ and so the loop invariant is maintained in this case.

Therefore, the loop invariant is maintained in both Cases 1 and 2.

Termination:
While loop ends if:

1. $A[i] + B[j] + C[k] = 0$
2. $j = len(B)$
3. $k = -1$

Case 1: $A[i] + B[j] + C[k] = 0$

We stop early if we find indices j, k for the given i such that $A[i] + B[j] + C[k] = 0$

and thus we have found $a = A[i], b = B[j], C = C[k]$ with $a+b+c = 0$ and the triple (a, b, c) is returned, which is the desired output of the algorithm if such a triple exists.

Case 2: $j = len(B)$
Note that for $j = len(B)$, $B[: j] = B[0, 1, ..., j-1] = B[0, 1, ..., len(B)-1] = B$. Therefore plugging $j = len(B)$ into the first condition in the loop invariant gives:

At the start of each iteration, there are no such elements a, b, c such that $a+b+c = 0$, where $a = A[i]$, for any b in $B[0, 1, ..., j-1] = B$ and for any c in C, which is a result useful for the outer loop invariant.

Case 3: $k = -1$
Note that for $k = -1$:

$$C[k+1 :] = C[k+1, ..., len(C)-1] = C[0, ...len(C)] = C$$

Plugging $k = -1$ into the second condition of loop invariant gives:

At the start of each iteration, there are no such elements a, b, c such that $a+b+c = 0$, where $a = A[i]$, for any b in B and for any c in $C[k+1, ..., len(C)-1] = C$, which is the same as the termination property in Case 2.

Therefore, at termination, we either return the correct triple (a, b, c) or we have that for the given index i, there are no entries in lists B and C such that $a+b+c = 0$ for $a = A[i], b$ in B, c in C, which is a property we will use in the outer loop invariant.

1.1 Outer Loop Invariant: At the start of each iteration i, there are no such entries a from the sub array A[0, 1, ... , i-1], b in B, c in C such that a + b + c = 0

Initialisation: i = 0:
At the start of the for loop ($i = 0$), we have that the list $A[0, ..., i-1]$ is empty. Thus, there are no values a in $A[0, ..., i-1]$ and as a result no such a in $A[0, ..., i-1]$, b in B, c in C such that $a+b+c = 0$, so loop invariant holds at initialisation.

Maintenance:
Suppose the loop invariant holds at the start of the i^{th} iteration. During the i^{th} iteration, the algorithm runs through the while loop in lines 7-13. For the loop to continue to the next iteration, we must have that that the inner while loop terminates when

$j = len(B)$ or $k = -1$.
Using the inner loop termination property for these cases, we have:

There are no such elements a, b, c such that $a + b + c = 0$, where $a = A[i]$, b in B and c in C.

Combining this with the loop invariant at the start of the i^{th} iteration, that there are no such a in $A[0, ..., i-1]$, b in B, c in C such that $a+b+c = 0$, we have that at the start of the $(i+1)^{th}$ iteration:

There are no such a in $A[0, ..., i]$, b in B, c in C such that $a + b + c = 0$, and thus the loop invariant holds true at the start of the $(i+1)^{th}$ iteration.

Termination:
The for loop terminates when:

1. $i = len(A)$
2. $a + b + c = 0$ for some a in A, b in B, c in C.

In the first case, when $i = len(A)$:

$$A[0, 1, ..., i-1] = A[0, 1, ..., len(A) - 1] = A$$

and so by the loop invariant, we have that there are no a in A, b in B, c in C such $a + b + c = 0$, in which case we correctly return the value None.

In the second case: we return the triple (a, b, c) such that $a + b + c = 0$, the desired result of the algorithm.

Therefore, we either return the triple (a, b, c) if it exists or return None when a triple doesn't exist, which gives us the desired output - thus proving correctness of the whole algorithm.