Problem 1: Simple Recursive Algorithm

function routes1(n: integer, m: integer): integer

1. if (n == 0 or m == 0) then // Base Case

2. return 1 // Only one possible pathway

3. elseif (n > 0 and m > 0) then // When n and m are positive

4. return routes1(n-1,m) + routes1(n, m-1) // Recursive case

end if

end function

Problem 2: Proof of Correctness of Recursive Algorithm

*Claim*: The algorithm routes1 correctly solves the “Route counting” problem. That is, if this algorithm is executed with non-negative integers n and m as input, then the computation eventually terminates, and the value routes1(m,n) (as defined in the pseudo-code from Question 1) is returned as output.

Method of Proof: This algorithm will be proved by standard mathematical induction on the bound function (for the recursive algorithm) that is n + m. The case where n = 0 or m = 0 will be considered the base case.

*Basis Step*: If n = 0 or m = 0, then an execution of the algorithm on this input includes:

* An execution of the test at step 1, which is passed, and
* An execution of the return statement at step 2 – resulting in the termination of the algorithm, with the value 1 returned as output.

Since routes1(n,0) = routes1(0,m) = 1, the value routes1(n,m) = routes1(n,0) = routes(0,m) has been returned as output in this case, as required.

*Inductive Step:* Let k and l be arbitrary integers such that k, l ≥ 0. It is now necessary, and sufficient, to prove the following:

*Inductive Claim:* If the algorithm routes1 is executed with the input n = k+1 and m = l+1 then this execution of the algorithm eventually terminates, and routes1(k+1, l+1) is returned as output when this happens, assuming only the following:

*Inductive Hypothesis:* For any integer i such that 0 ≤ i ≤ k and for any integer j such that 0 ≤ j ≤l, if the algorithm routes1 is executed with the input n = i and m = j then this execution of the algorithm eventually terminates, and the value of routes1(i,j) is returned as output as a result.

With this hypothesis, consider an execution of the algorithm on the input n = k+1 and m = l+1.

Since k ≥ 0, n = k+1 ≥ 1, and this execution of the algorithm includes

* An execution of the test at step 1, which fails
* An execution of the test at step 3, which succeeds
* An execution of the return statement at step 4.

Similarly, since l ≥ 0, m = l + 1, and this execution of the algorithm includes

* An execution of the test at step 1, which fails
* An execution of the test at step 3, which succeeds
* An execution of the return statement at step 4.

Considering the execution of the return statement at step 4 in more depth:

First, the algorithm is called recursively with input n-1. Since n = k+1 and k ≥ 0, 0 ≤ n-1 ≤ k and it follows by the inductive hypothesis that this recursive execution of the algorithm eventually terminates, with routes1(n-1,m) returned as output.

Secondly, the algorithm is called recursively with input m-1. Since m = l +1 and l ≥ 0, 0 ≤ m -1 ≤ l and it follows by the inductive hypothesis that this recursive execution of the algorithm eventually terminates, with routes1(n, m-1) returned as output.

Finally, since n = k +1 ≥ 1, routes1(n,m) = routes1(n-1,m) + routes1(n, m-1) and, as shown by the return statement in step 4, the initial execution of the algorithm also terminates, with the value routes1(n,m) = routes(k+1, l+1) returned as output, as required.

*Conclusion*: Because the inductive step eventually reaches routes1(0, m) or routes1(n, 0), both of which we have defined as equal to 1, it now follows by induction on the input value n and m that the algorithm routes1 eventually terminates and thus solves the “Route Counting” problem, as initially claimed.

*This proof follows the structure of the sample recursive Fibonacci Algorithm as seen in the Lecture Slides by Dr. Wayne Eberly for the CPSC331 class of Fall 2013.*

Problem 4: Pseudo-code w/ Assertions and Loop Invariants for Non-Recursive Algorithm

/\* The following pages document our algorithm for solving the Routes(n, m)

\* counting problem as per Assignment 1 Problem 4, in pseudo-code. Assertions and

\* loop invariants have been included where deemed necessary. These have also been

\* included in cpsc331.assignment1.Routes2.java as inline documentation as well.

\*

\* Submitted by Bryan Huff (UCID 10096604), Michael Hung (UCID 10099049), and Arnold

\* Padillo (UCID 10097013)

\*/

function count (columns: integer, rows: integer): BigInteger

// Assertion: inputs columns or rows are an integer less than 0

1. if (columns < 0 or rows < 0) then

2. throw IllegalArgumentException

// Assertion: inputs columns or rows are an integer greater than or equal to 0

3. elseif (columns == 0 or rows == 0) then

// Assertion: inputs columns or rows are an integer exactly equal to 0

4. return 1

/\* Assertion:

\* 1. inputs columns or rows are an integer exactly equal to 0

\* 2. A BigInteger with value 1 has been returned

\*/

// Assertion: columns and rows are both non-negative integers

else

5. Declare grid to be an array, with length (columns+1), of arrays with length

(rows+1) of BigIntegers

6. Declare i to be an integer variable with value 0

7. Declare j to be an integer variable with value 1

8. Declare p to be an integer variable with value 1

9. Declare q to be an integer variable with value 1

/\* Loop Invariant:

\* 1. columns is a non-negative integer input

\* 2. rows is a non-negative integer input

\* 3. The value of i is between 0 and (columns+1), inclusive

\* 4. grid is an array, with length (columns+1), of BigInteger arrays with

\* length (rows+1)

\*

\* Bound Function: columns - i

\*/

/\* Assertion - Before Loop Execution:

\* 1. The loop invariant is satisfied

\* 2. i is equal to 0

\* 3. grid[x][y] == null, for every x and y such that 0 <= x < (columns+1),

\* 0 <= y < (rows+1)

\*/

10. while (i < columns + 1) do

/\* Assertion - Before Each Iteration:

\* 1. The loop invariant is satisfied

\* 2. The value of i is between 0 and (columns+1), inclusive

\* 3. grid[x][0] == BigInteger("1") for every x such that 0 <= x < i

\* 4. grid[x][0] == null for every x such that i <= x < (columns+1)

\*/

11. grid[i][0] := 1

12. i := i + 1

/\* Assertion - After Each Iteration:

\* 1. The loop invariant is satisfied

\* 2. The value of i is between 1 and (columns+1), inclusive

\* 3. grid[x][0] == BigInteger("1") for every x such that 0 <= x < i

\* 4. grid[x][0] == null for every x such that i <= x < (columns+1)

\*/

end while

/\* Assertion - After Loop Execution:

\* 1. The loop invariant is satisfied

\* 2. The value of i is equal to (columns+1)

\* 3. grid[x][0] == BigInteger("1") for every x such that

\* 0 <= x < (columns+1)

\* 4. grid[x][y] == null for every x and y such that 1 <= x < (columns+1),

\* 0 <= y < (rows+1)

\*/

/\* Loop Invariant:

\* 1. columns is a non-negative integer input

\* 2. rows is a non-negative integer input

\* 3. The value of j is between 1 and (rows+1), inclusive

\* 4. grid is an array, with length (columns+1), of BigInteger arrays with

\* length (rows+1)

\* 5. grid[x][0] == BigInteger("1") for every x such that

\* 0 <= x < (columns+1)

\*

\* Bound Function: rows - j

\*/

/\* Assertion - Before Loop Execution:

\* 1. The loop invariant is satisfied

\* 2. The value of j is equal to 1

\* 3. grid[x][y] == null, for every x and y such that 0 <= x < (columns+1),

\* 1 <= y < (rows+1)

\*/

13. while (j < rows + 1) do

/\* Assertion - Before Each Iteration:

\* 1. The loop invariant is satisfied

\* 2. The value of j is between 1 and (rows+1), inclusive

\* 3. grid[0][y] == BigInteger("1") for every y such that 0 <= x < j

\* 4. grid[0][y] == null for every y such that j <= y < (rows+1)

\*/

14. grid[0][j] := 0

15. j := j + 1

/\* Assertion - After Each Iteration:

\* 1. The loop invariant is satisfied

\* 2. The value of j is between 2 and (rows+1), inclusive

\* 3. grid[0][y] == BigInteger("1") for every y such that 0 <= x < j

\* 4. grid[0][y] == null for every y such that j <= y < (rows+1)

\*/

end while

/\* Assertion - After Loop Execution:

\* 1. The loop invariant is satisfied

\* 2. The value of j is equal to (rows+1)

\* 3. grid[0][y] == BigInteger("1") for every y such that 0 <= y < (rows+1)

\* 4. grid[x][y] == null for every x and y such that 1 <= x < (columns+1),

\* 1 <= y < (rows+1)

\*/

/\* Outer Loop Invariant:

\* 1. columns is a non-negative integer input

\* 2. rows is a non-negative integer input

\* 3. The value of p is between 1 and (columns+1), inclusive

\* 4. The value of q is between 1 and (rows+1), inclusive

\* 5. grid is an array, with length (columns+1), of BigInteger arrays with

\* length (rows+1)

\* 6. grid[x][0] == BigInteger("1") for every x such that 0 <= x <

\* (columns+1)

\* 7. grid[0][y] == BigInteger("1") for every y such that 0 <= y < (rows+1)

\*

\* Bound Function: columns - p

\*/

/\* Assertion - Before Outer Loop Execution:

\* 1. The outer loop invariant is satisfied

\* 2. The value of p is equal to 1

\* 3. The value of q is equal to 1

\* 4. grid[x][y] == null, for every x and y such that 1 <= x < (columns+1),

\* 1 <= y < (rows+1)

\*/

16. while (p < columns + 1) do

/\* Assertion - Before Each Outer Loop Iteration:

\* 1. The outer loop invariant is satisfied

\* 2. The value of p is between 1 and (columns+1), inclusive

\* 3. The value of q is between 1 and (rows+1), inclusive

\* 4. grid[x][y] != null for every x and y such that 0 <= x < p,

\* 0 <= y < q

\* 5. grid[x][y] == null for every x and y such that p <= x < (columns+1),

\* q <= y < (rows+1)

\*/

17. q := 1

/\* Inner Loop Invariant:

\* 1. columns is a non-negative integer input

\* 2. rows is a non-negative integer input

\* 3. The value of p is between 1 and (columns+1), inclusive

\* 4. The value of q is between 1 and (rows+1), inclusive

\* 5. grid is an array, with length (columns+1), of BigInteger arrays with

\* length (rows+1)

\* 6. The value of p in the Inner Loop is kept constant

\*

\* Bound Function: rows - q

\*/

/\* Assertion - Before Inner Loop Execution:

\* 1. The outer and inner loop invariants are satisfied

\* 2. The value of q is equal to 1

\* 3. grid[x][y] != null for every x and y such that 0 <= x < p, 0 <= y < q

\* 4. grid[x][y] == null, for every x and y such that p <= x < (columns+1),

\* 1 <= y < (rows+1)

\*/

18. while (q < rows + 1) do

/\* Assertion - Before Each Inner Loop Iteration:

\* 1. The outer and inner loop invariants are satisfied

\* 2. The value of q is between 1 and (rows+1), inclusive

\* 3. grid[x][y] != null for every x and y such that 0 <= x < p,

\* 0 <= y < q

\* 4. grid[x][y] == null for every x and y such that

\* p <= x < (columns+1),

\* q <= y < (rows+1)

\*/

19. grid[p][q] := grid[p - 1][q] + grid[p][q - 1]

20. q := q + 1

/\* Assertion - After Each Inner Loop Iteration:

\* 1. The outer and inner loop invariants are satisfied

\* 2. The value of q is equal to (rows+1)

\* 3. grid[x][y] != null for every x and y such that 0 <= x < p,

\* 0 <= y < q

\* 4. grid[x][y] == null for every x and y such that

\* p <= x < (columns+1),

\* q <= y < (rows+1)

\*/

end while

/\* Assertion - After Inner Loop Execution:

\* 1. The outer and inner loop invariants are satisfied

\* 2. The value of q is equal to (columns+1)

\* 3. grid[x][y] != null for every x and y such that 0 <= x < (columns+1),

\* 0 <= y < (rows+1)

\* 4. grid[x][y] == null for every x and y such that p <= x < (columns+1),

\* q <= y < (rows+1)

\*/

21. p := p + 1

/\* Assertion - After Each Outer Loop Iteration:

\* 1. The outer loop invariant is satisfied

\* 2. The value of p is between 2 and (columns+1), inclusive

\* 3. The value of q is equal to (rows+1)

\* 4. grid[x][y] != null for every x and y such that 0 <= x < p, 0 <= y < q

\* 5. grid[x][y] == null for every x and y such that p <= x < (columns+1),

\* q <= y < (rows+1)

\*/

endwhile

/\* Assertion - After Outer Loop Execution:

\* 1. The outer loop invariant is satisfied

\* 2. The value of p is equal to (rows+1)

\* 3. The value of q is equal to (columns+1)

\* 4. grid[x][y] != null for every x and y such that 0 <= x < (columns+1),

\* 0 <= y < (rows+1)

\*/

/\* Assertion:

\* 1. columns and rows are non-negative integer inputs

\* 2. The value of grid[columns][rows] is not null

\* 3. grid[columns][rows] contains an object of class BigInteger that has some

\* returnable value

\*/

22. return grid[columns][rows]

/\* Assertion:

\* 1. columns and rows are non-negative integer inputs

\* 2. A BigInteger containing the correct value solving Routes(n, m) has been

\* returned

\*/

endif

end function