CSE 4321-002

Final Exam

Practice A

Fall 2020

ID:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Below is a faulty program. It includes test inputs that result in failure. Answer the following questions about the program.

/\*\*

\* Find last index of element

\*

\* @param x array to search

\* @param y value to look for

\* @return last index of y in x; -1 if absent

\* @throws NullPointerException if x is null

\*/

public int findLast (int[] x, int y)

{

for (int i=x.length-1; i > 0; i--)

{

if (x[i] == y)

{

return i;

}

}

return -1;

}

// test: x = [2, 3, 5]; y = 2; Expected = 0

// Book website: FindLast.java

// Book website: FindLastTest.java

* 1. Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
  2. If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.
  3. If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.
  4. If possible, give a test case that results in an error, but **not** a failure. If not, briefly explain why not. Hint: Don’t forget about the program counter.
  5. For the given test case, describe the first error state. Be sure to describe the complete state.
  6. Implement your repair and verify that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.

1. Below is a faulty program. It includes test inputs that result in failure. Answer the following questions about the program.

/\*\*

\* Count positive elements

\*

\* @param x array to search

\* @return count of positive elements in x

\* @throws NullPointerException if x is null

\*/

public int countPositive (int[] x)

{

int count = 0;

for (int i=0; i < x.length; i++)

{

if (x[i] >= 0)

{

count++;

}

}

return count;

}

// test: x = [-4, 2, 0, 2]; Expected = 2

// Book website: CountPositive.java

// Book website: CountPositiveTest.java

* 1. Explain what is wrong with the given code. Describe the fault precisely by proposing a modification to the code.
  2. If possible, give a test case that does **not** execute the fault. If not, briefly explain why not.
  3. If possible, give a test case that executes the fault, but does **not** result in an error state. If not, briefly explain why not.
  4. If possible, give a test case that results in an error, but **not** a failure. If not, briefly explain why not. Hint: Don’t forget about the program counter.
  5. For the given test case, describe the first error state. Be sure to describe the complete state.
  6. Implement your repair and verify that the given test now produces the expected output. Submit a screen printout or other evidence that your new program works.

1. Consider a system that consists of 4 parameters, P1, P2, P3, and P4. Each parameter has two values 0 and 1. Apply algorithm IPO to create a pairwise test set for this system. Use “-” to represent *don’t care* values, i.e., values that do not affect coverage. Clearly indicate your tie-breaking rules that may be needed in the test generation process. You must show intermediate steps to obtain full credits.
2. Answer questions a–g for the graph defined by the following sets:

Also consider the following (candidate) test paths:

* 1. Draw the graph.
  2. List the test requirements for Edge-Pair Coverage. (Hint: You should get 12 requirements of length 2.)
  3. Does the given set of test paths satisfy Edge-Pair Coverage? If not, state what is missing.
  4. Consider the simple path **[3, 2, 4, 5, 6]** and test path **[1, 2, 3, 2, 4, 6, 1, 2, 4, 5, 6, 1, 7]**. Does the test path tour the simple pathdirectly? With a sidetrip? If so, write down the sidetrip.
  5. List the test requirements for Node Coverage, Edge Coverage, and Prime Path Coverage on the graph.
  6. List test paths from the given set that achieve Node Coverage but not Edge Coverage on the graph.
  7. List test paths from the given set that achieve Edge Coverage but not Prime Path Coverage on the graph.

1. Answer questions a–d for the graph defined by the following sets:

Also consider the following (candidate) paths:

* 1. Which of the listed paths are test paths? For any path that is not a test path, explain why not.
  2. List the eight test requirements for Edge-Pair Coverage (only the length two subpaths).
  3. Does the set of **test** paths from part (a) above satisfy Edge-Pair Coverage? If not, state what is missing.
  4. Consider the prime path [3, 1, 3] and path *p*2. Does *p*2 tour the prime path directly? With a sidetrip?

1. (Graph II) Below is a graph which is defined by the sets of nodes, initial nodes, final nodes, edges, and defs and uses. The graph also contains some test paths. Answer the following questions about the graph.

**Graph II.**

// Assume the use of x in 3 precedes the def

**Test Paths:**

* 1. Draw the graph.
  2. List all of the du-paths with respect to *x*. (Note: Include all dupaths, even those that are subpaths of some other du-path).
  3. Determine which du-paths each test path tours. Write them in a table with test paths in the first column and the du-paths they cover in the second column. For this part of the exercise, you should consider both direct touring and sidetrips.
  4. List a minimal test set that satisfies *all defs* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.
  5. List a minimal test set that satisfies *all uses* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.
  6. List a minimal test set that satisfies *all du-paths* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.

1. (Graph III) Below is a graph which is defined by the sets of nodes, initial nodes, final nodes, edges, and defs and uses. The graph also contains some test paths. Answer the following questions about the graph.

**Graph III.**

**Test Paths:**

* 1. Draw the graph.
  2. List all of the du-paths with respect to *x*. (Note: Include all dupaths, even those that are subpaths of some other du-path).
  3. Determine which du-paths each test path tours. Write them in a table with test paths in the first column and the du-paths they cover in the second column. For this part of the exercise, you should consider both direct touring and sidetrips.
  4. List a minimal test set that satisfies *all defs* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.
  5. List a minimal test set that satisfies *all uses* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.
  6. List a minimal test set that satisfies *all du-paths* coverage with respect to *x*. (Direct tours only.) If possible, use the given test paths. If not, provide additional test paths to satisfy the criterion.

1. Use predicates (vii) and (ix) to answer the following questions.

**vii.**

**ix.**

* 1. List the clauses that go with predicate .
  2. Compute (and simplify) the conditions under which each clause determines predicate .
  3. Write the complete truth table for the predicate. Label your rows starting from 1. Use the format in the example underneath the definition of Combinatorial Coverage in Section 8.1.1. That is, row 1 should be all clauses true. You should include columns for the conditions under which each clause determines the predicate, and also a column for the value of the predicate itself.
  4. List **all** pairs of rows from your table that satisfy Gencral Active Clause Coverage (GACC) with respect to each clause.
  5. List **all** pairs of rows from your table that satisfy Correlated Active Clause Coverage (CACC) with respect to each clause.
  6. List **all** pairs of rows from your table that satisfy Restricted Active Clause Coverage (RACC) with respect to each clause.

1. Provide reachability conditions, infection conditions, propagation conditions, and test case values to kill mutants 2, 4, 5, and 6 in Figure 9.1.

|  |  |
| --- | --- |
| Original Method | With Embedded Mutants |
| int Min (int A, int B)  {  int minVal;  minVal = A;  if (B < A)  {  minVal = B;  }  return (minVal);  } // end Min | int Min (int A, int B)  {  int minVal;  minVal = A;Δ1 minVal = B;  if (B < A)  Δ2 if (B > A)  Δ3 if (B < minVal)  {  minVal = B;  Δ4 Bomb();  Δ5 minVal = A;  Δ6 minVal = failOnZero (B);  }  return (minVal)};  } // end Min |

**Figure 9.1.** Method Min and six mutants.

1. Answer questions (a) through (d) for the mutant on line 6 in the method sum().

|  |
| --- |
| /\*\*  \* Sum values in an array  \*  \* @param x array to sum  \*  \* @return sum of values in x  \* @throws NullPointerException if x is null  \*/  1. public static int sum(int[] x)  2. {  3. int s = 0;  4. for (int i=0; i < x.length; i++) }  5. {  6. s = s + x[i];  6’. // s = s – x[i]; //AOR  7. }  8. return s;  9. } |

* 1. If possible, find test inputs that do **not** reach the mutant.
  2. If possible, find test inputs that satisfy reachability but **not infection** for the mutant.
  3. If possible, find test inputs that satisfy reachability and infection, but **not propagation** for the mutant.
  4. If possible, find test inputs that strongly **kill** the mutants.

Note that for the following problems, you must show intermediate steps in order to get full credits.

1. Suppose program P has been executed against a test suite T consisting of six tests, t1, t2, t3, t4, t5 and t6. A total of six entities are covered by the tests as shown in the following table: 0 (or 1) in a column indicates that the corresponding entity is not covered (or covered). The entities could be basic blocks in the program, functions, def-uses, or any other testable element of interest. Follow procedure CMIMX to find the minimal cover set for the six entities.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| t1 | 0 | 1 | 0 | 1 | 0 | 1 |
| t2 | 1 | 0 | 0 | 0 | 1 | 0 |
| t3 | 1 | 1 | 0 | 1 | 0 | 0 |
| t4 | 1 | 0 | 1 | 0 | 0 | 0 |
| t5 | 0 | 1 | 0 | 1 | 1 | 0 |
| t6 | 1 | 0 | 0 | 0 | 0 | 0 |

1. Suppose that there is an application P consisting of 8 methods, m1 .. m8. Also suppose that there is a regression-test set T = {t1, t2, t3, t4, t5}. The methods covered by each test in T are listed in the following table. Follow procedure PrTest to obtain a prioritized list of tests based on residual coverage.

|  |  |  |
| --- | --- | --- |
| Test (t) | Methods covered (cov(t)) | |cov(t)| |
| t1 | m1, m3, m5, m6, m8 | 5 |
| t2 | m1, m7, m8 | 3 |
| t3 | m1, m2, m3, m5 | 4 |
| t4 | m1, m2, m3, m4 | 4 |
| t5 | m1, m5, m8 | 3 |