
Name : Harsh Prajapati
ID : 202001145
Lab : 07 - Testing in software engineering

Section A

Equivalent Partitioning:

- Valid Input
  - Give the input in a given range
- Invalid Input
  - Give the invalid input like NULL array etc.

Boundary Value Analysis:

- Valid boundary values
  - Give the constraints that lies in the boundary, like maximum or minimum
- Invalid boundary values
  - Give the input that is not into range and more value than the maximum or less value then minimum

Program 1:

P1. The function `linearSearch` searches for a value `v` in an array of integers `a`. If `v` appears in the array `a`, then the function returns the first index `i`, such that `a[i] == v`; otherwise, -1 is returned.

```
int linearSearch(int v, int a[])
{
      int i = 0;
      while (i < a.length)
      {
            if (a[i] == v)
                  return(i);
            i++;
      }
      return (-1);
}
```

Write the Junit test code:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class LinearSearchTest {

	@Test
	public void testLinearSearch() {
		int[] a = {1, 2, 3, 4, 5};
		int v = 3;
		int expected = 2;
		int actual = LinearSearch.linearSearch(v, a);
		assertEquals(expected, actual);


	assertEquals(0, obj.linearSearch(2, arr1));
	assertEquals(1, obj.linearSearch(4, arr1));
	assertEquals(-1, obj.linearSearch(6, arr1));
	assertEquals(-1, obj.linearSearch(0, arr1));

	}
}
```

Considering 0-base indexing

Testcase	Expected output
a=[45,56,78,89,45],v = 56	1
a=[],v=45	-1
a=NULL, v=12	Error Message
a=[45,56,78,89,45,56],v=45	0
a=[45,56,78,89,45,56],v=56	5
a=[45,56,78,89,45,56],v=100	-1
a=[45,56,78,89,45,56],v=0	-1
a=[100000000000,1000000001010,50000],v=50000	Error Message
a=[45,56,78,89,45,56],v=a	Error Message
a=[45,56,78,89,45,56],v=4.5	Error Message

Program-2

In this unit we are given an array and one value and unit is returning the total count of the value into the array.

Assuming the o based indexing,

```java
package tests;

public class UnitTesting {
    int countItem(int v, int a[])
    {
        int count = 0;
        for (int i = 0; i < a.length; i++)
        {
            if (a[i] == v)
            count++;
        }
        return (count);
    }
}

```

Junit Code:

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class CountItemTest {

        @Test
        public void testCountItem() {
                int[] a = {1, 2, 1, 4, 5};
                int v = 1;
                int expected = 2;
                int actual = CountItem.countItem(v, a);
                assertEquals(expected, actual);

        assertEquals(2, obj.countItem(1, arr1));
        assertEquals(0, obj.countItem(6, arr1));
        assertEquals(0, obj.countItem(0, arr1));

        }
}
```

TestCase	Expected outcome
a=[1,2,3,4,3],v = 3	2
a=[1,2,3,4,3],v = 0	0
a=[],v = don’t care	0
a=NULL, v = don’t care	Error Message
a=[1,2,3,4,5], v = 1	1
a=[1,1,1,1,1], v = 1	5

a=[1,2,3,4,1] v = “1”	Error Message
a=[1,1,2,3,4], v = 1.1	Error Message

Problem : 3

In this Unit we are applying the binary search in the array and given one value to find and if not present then return -1;

```
1  package tests;
2
3  public class UnitTesting {
4      int binarySearch(int v, int a[])
5      {
6      int lo,mid,hi;
7      lo = 0;
8      hi = a.length-1;
9      while (lo <= hi)
10     {
11         mid = (lo+hi)/2;
12         if (v == a[mid])
13             return (mid);
14         else if (v < a[mid])
15             hi = mid-1;
16         else
17             lo = mid+1;
18
19     }
20         return(-1);
21     }
22 }
23
```

Junit code:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class BinarySearchTest {

        @Test
        public void testBinarySearch() {
                int[] a = {1, 2, 3, 4, 5};
                int v = 3;
                int expected = 2;
                int actual = BinarySearch.binarySearch(v, a);
                assertEquals(expected, actual);

        assertEquals(0, obj.binarySearch(2, arr1));
        assertEquals(1, obj.binarySearch(4, arr1));
        assertEquals(-1, obj.binarySearch(6, arr1));
        assertEquals(-1, obj.binarySearch(0, arr1));

        }
}
```

Array	Number to find	Expected outcome
Valid Input		
[1,2,3,4,5,6]	3	2
[1,2,3,4,5,6]	100	-1
Invalid input		
[“1”,”2”,”3”,”4”,”5”]	Don’t care	Invalid Input Error Message
[1,2,3,4,5,6]	“a”	Invalid input error message
[1,2,3,4,5,6]	“1”	Invalid input error message
[1,2,3,4,5,6]	1.2	Invalid input error message
Boundry value analysis		
[1,2,3,4,5,6]	6	5
[1,2,3,4,5,6]	1	0
[1,2,3,4,5,6]	0	-1
[1,2,3,4,5,6]	7	-1
[1,1,2,3,4,5]	1	1
[1,3,2,5,4]	Don't care	Invalid input error message

Problem: 4

In this Unit we are classifying the triangle by given three different sides length in the input.

```java
package tests;

public class UnitTesting {
    final int EQUILATERAL = 0;
    final int ISOSCELES = 1;
    final int SCALENE = 2;
    final int INVALID = 3;
    int triangle(int a, int b, int c)
    {
        if (a >= b+c || b >= a+c || c >= a+b)
            return(INVALID);
        if (a == b && b == c)
            return(EQUILATERAL);
        if (a == b || a == c || b == c)
            return(ISOSCELES);

        return(SCALENE);

    }
}

```

Junit Code

```java
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class TriangleTest {

	@Test
	public void testTriangle() {
		int a = 5;
		int b = 5;
		int c = 5;
		int expected = 0;
		int actual = Triangle.triangle(a, b, c);
		assertEquals(expected, actual);

		a = 5;
		b = 5;
		c = 6;
		expected = 1;
		actual = Triangle.triangle(a, b, c);
		assertEquals(expected, actual);

		a = 4;
		b = 5;
		c = 6;
		expected = 2;
		actual = Triangle.triangle(a, b, c);
		assertEquals(expected, actual);

		a = 4;
		b = 5;
		c = 11;
		expected = 3;
		actual = Triangle.triangle(a, b, c);
		assertEquals(expected, actual);

		a = 0;
		b = 5;
		c = 6;
		expected = 3;
		actual = Triangle.triangle(a, b, c);
		assertEquals(expected, actual);
	}
}
```

Side lengths (a,b,c)	Expected outcome
a=b=c (5,5,5)	EQUILATERAL
a=b<c (5,5,6)	ISOSCELES
a<b<c (4,5,6)	SCALENE
a+b<=c (4,5,11)	INVALID
a+c<=b (4,11,4)	INVALID
b+c<=a (14,5,1)	INVALID
a=0 or b=0 or c=0 (0,5,6)	INVALID
a=float or b=float or c=float (2.2,3,3), (2.2,2.5,3)	INVALID INPUT MESSAGE
a=”string” or b=”string” or c=”string” (“a”,2,5)	INVALID INPUT MESSAGE

Problem:5

The function prefix (String s1, String s2) returns whether or not the string s1 is a prefix of string s2 (you may assume that neither s1 nor s2 is null).

```java
package tests;

public class UnitTesting {
    public static boolean prefix(String s1, String s2)
    {
        if (s1.length() > s2.length())
        {
            return false;
        }
        for (int i = 0; i < s1.length(); i++)
        {
            if (s1.charAt(i) != s2.charAt(i))
            {
                return false;
            }
        }
        return true;
    }
}
```

Junit Code:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class PrefixTest {

	@Test
	public void testPrefix() {
		String s1 = "";
		String s2 = "";
		boolean expected = true;
		boolean actual = Prefix.prefix(s1, s2);
		assertEquals(expected, actual);

		s1 = "";
		s2 = "hello";
		expected = true;
		actual = Prefix.prefix(s1, s2);
		assertEquals(expected, actual);

		s1 = "hello";
		s2 = "helloall";
		expected = true;
		actual = Prefix.prefix(s1, s2);
		assertEquals(expected, actual);

		s1 = "hello";
		s2 = "hiall";
		expected = false;
		actual = Prefix.prefix(s1, s2);
		assertEquals(expected, actual);

		s1 = "hello";
		s2 = "hello";
		expected = true;
		actual = Prefix.prefix(s1, s2);
		assertEquals(expected, actual);
	}
}
```

String s1	String s2	Expected outcome
“” (empty string)	“” (empty string)	True
“”(empty string)	“abc”	True
“abc”	“”(empty string)	False
NULL	“abc”	NULLPOINTER
“abc”	NULL	NULLPOINTER
“harsh”	“harshtest”	True
“harsh”	“testharsh”	False
“harshtest”	“harsh”	False
Boundry Values Analysis		
“”	“”	True

“a”	“a”	True
“H….” (same)	“H….” (same as a)	True

Problem :6
Consider the triangle classification program (P4) again with a slightly different specification: The program reads floating values from the standard input. The three values A, B, and C are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output stating whether the triangle can be formed, is scalene, isosceles, equilateral, or right-angled. Determine the following for the above program:

a) Identify the equivalence classes for the system
Test case 1: All sides are positive, real numbers.
Test case 2: The summation rule is satisfied.
Test case 3: Any side is negative or zero.
Test case 4: The summation rule is not satisfied.

b) Identify test cases to cover the identified equivalence classes. Also, explicitly mention which test case would cover which equivalence class.
(Hint: you must need to ensure that the identified set of test cases cover all identified equivalence classes)
Test case 1: A=5, B=5, C=5 (equilateral triangle)
Test case 2: A=5, B=5, C=7 (isosceles triangle)
Test case 3: A=5, B=6, C=7 (scalene triangle)
Test case 4: A=5, B=6, C=0 (invalid input)
Test case 5: A=5, B=6, C=11 (invalid input)
Test case 6: A=5, B=6, C=-1 (invalid input)

c) For the boundary condition A + B > C case (scalene triangle), identify test cases to verify the boundary.
Test case 1 : A=5, B=6, C=10 (c=a+b-1) (valid scalene triangle)
Test case 2 : A=5, B=6, C=11 (c=a+b) (invalid input)
Test case 3 : A=5, B=6, C=12 (c=a+b+1) (invalid input)

d) For the boundary condition A = C case (isosceles triangle), identify test cases to verify the boundary.
Test case 1 : A=5, B=6, C=5 (c=a) (valid isosceles triangle)

Test case 2 : A=5, B=11, C=5 (c=a) (invalid input as b>a+c)
Test case 3 : A=5, B=5, C=5 (c=a) (equilateral triangle)

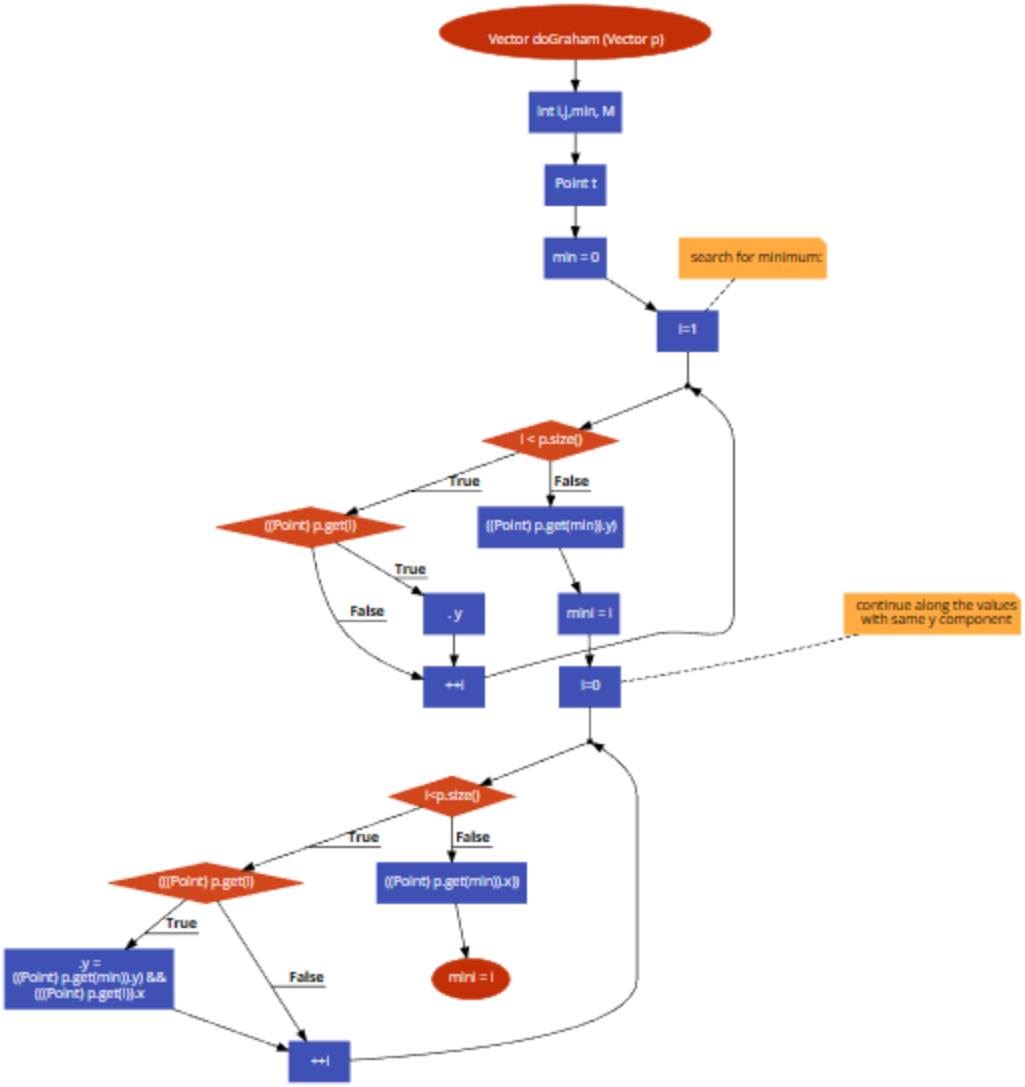
e) For the boundary condition A = B = C case (equilateral triangle), identify test cases to verify the boundary.
Test case 1 : A=5, B=5, C=5 (valid equilateral triangle)

f) For the boundary condition A2 + B2 = C2 case (right-angle triangle), identify test cases to verify the boundary.
test case 1: A=3, B=4, C=5 (valid right-angle triangle)
Test case 2: A=IntMax, B=IntMax, C=IntMax (overflow error)

g) For the non-triangle case, identify test cases to explore the boundary.
Test case 1 : A=5, B=6, C=11 (invalid input as a+b=c)
Test case 2 : A=5, B=6, C=12 (invalid input as a+b+1=c)

h) For non-positive input, identify test points.
Test case 1 : A=0, B=6, C=11 (invalid input as a=0)
Test case 2 : A=-5, B=6, C=11 (invalid input as a<0)

Section : 2
CFG



Construct test sets for your flow graph that are adequate for the following criteria:

a. Statement Coverage: To achieve statement coverage, we need to make sure that every statement in the code is executed at least once

To satisfy statement coverage, we need to ensure that each statement in the CFG is executed at least once. We can achieve this by providing a test case with a single point in the vector. In this case, both loops will not execute, and the return statement will be executed. A test set that satisfies statement coverage would be:
p = [Point (0,0)]

Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component
Test 4: p = vector with two points with different y components
Test 5: p = vector with three or more points with different y components
Test 6: p = vector with three or more points with the same y component

b. Branch Coverage
To satisfy branch coverage, we need to ensure that each branch in the CFG is executed at least once. We can achieve this by providing a test case with two points such that one of the points has the minimum y-coordinate, and the other has a greater x-coordinate than the minimum. In this case, both loops will execute, and the second branch in the second loop will be taken. A test set that satisfies branch coverage would be:
p = [Point (0,0), Point (1,1)]

Test 1: p = empty vector
Test 2: p = vector with one point
Test 3: p = vector with two points with the same y component
Test 4: p = vector with two points with different y components
Test 5: p = vector with three or more points with different y components, and none of them have the same x component
Test 6: p = vector with three or more points with the same y component, and some of them have the same x component
Test 7: p = vector with three or more points with the same y component, and all of them have the same x component

c.Basic Condition Coverage:
In order to meet the requirement for basic condition coverage, it is necessary to ensure that each condition in the control flow graph is evaluated to both true and false at least once. One approach to achieve this is to design a test case that includes three points, with two of the points having the same y-coordinate and the third point having an x-coordinate greater than the minimum. By doing so, both loops will be executed, and the second condition in the second loop will be evaluated to both true and false. To meet the criteria for basic condition coverage, a suitable test set would consist of the following three points: Point(0,0), Point(1,1), and Point(2,0).

Test 1: p = empty vector

Test 2: p = vector with one point

Test 3: p = vector with two points with the same y component

Test 4: p = vector with two points with different y components

Test 5: p = vector with three or more points with different y components, and none of them have the same x component

Test 6: p = vector with three or more points with the same y component, and some of them have the same x component

Test 7: p = vector with three or more points with the same y component, and all of them have the same x component

Test 8: p = vector with three or more points with the same y component, and some of them have the same x component, and the first point has a smaller x component

----------EOL----------