***Task1***

Aya Elsayed Shabana 20196012

Habiba ahmed El-amir 20196093

a) Fault description:

in the for loop, i< arr.length-1, the loop will stop before it reaches the last element in the array , thus the last element will not be read and there will be a missing instruction.

Modification:

For(int i=0;i<= arr.length-1;i++)

b) in this case, there will be no cases that implement the fault without an error state; since the program counter will always be missing an instruction when calculating the sum even if the sum is of a correct value but the program counter will be in an error state.

c) test input: arr= [6.5, 4.5, 8.7, 0.0]

expected: 4.925

program counter will iterate 3 times only not 4:

pc= 1 (i=0) element: 6.5

pc= 2 (i=1) element: 4.5

pc= 3 (i=2) element: 8.7

//exits for loop without the 4th iteration to read the element : 0.0

d) test input: arr= [3.0, 5.2, 6.0, 9.0]

expected: 5.8

e) [90.5, -65.0, 72.25]

tracing to describe the error state:

expected:

pc= 1 sum= 90.5

pc= 2 sum= 25.5

pc=3 sum=97.75

pc= 4 average= 32.5

actual:

pc= 1 sum= 90.5

pc= 2 sum= 25.5

pc=3 average= 25.5/3= 8.5 //first error state

since it doesn’t reach the last element in the array, when pc= 3, it will go to calculate the average before adding the element 72.25 to the sum.

***Task2 boundary analysis:***

The Chosen Coverage Criteria is **the base choice criterion** it recognizes that - Some blocks are more important than others - Using diverse combinations can strengthen testing and let testers bring in domain knowledge of the program.

Base Choice Coverage (BCC): A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each other characteristic.

Number of tests is one base test + one test for each “non- base” other block:

**1) encodeHashToLong (latitude,longitude,length)**

Equivalence class partitions: n

E1 value< 1, E2 value>12, E3 1<value<12

Boundaries:

0 1 2 11 12 13

E1 E2 E3

**2) encodeHash (double latitude, double longitude, int length)**

Equivalence partitions:

Latitude

E1 value<-90, E2 value> 90, E3 -90 <=value <= 90

Length

E4 value< 1, E5 value>12, E6 1<value<12

Boundaries:

Latitude

-91 -90 -89 98 90 91

E1 E2 E3

length

0 1 2 11 12 13

E4 E5 E6

**3) decodeHash (geohash)**

Equivalence partitions:

geohash E1 != null

E2 length of geohash <1

E3 length of geohash>12

E4 1<length of geohash<12

E5 non-empty length of geohash = 0

E6 Contains any character of: [o-l-i-a]

Boundaries:

Length of geohash

0 1 2 11 12 13

E1 E2 E3

E4 Geohash=klmnc

E6= “ “

**4) heightDegrees (int n)**

Equivalence class partitions: n

E1: n<=0

E2: n>0

Boundaries:

-1 0 1

E1 E2

**5) WidthDegrees(int n)**

Equivalence class partitions: n

E1: n<=0

E2: n>0

Boundaries:

-1 0 1

E1 E2

**6) right (string hash)**

Equivalence class partitions:

Length of hash

E1 value< 1, E2 value>12, E3 1<value<12

E4 Contains any character of: [o-l-i-a]

Boundaries:

Length of hash

0 1 2 11 12 13

E1 E2 E3

E4: hash = ghiks

**7) left (string hash)**

Equivalence class partitions:

Length of hash

E1 value< 1, E2 value>1

E3 Contains any character of: [o-l-i-a]

Boundaries:

Length of hash

0 1 2

E1 E2

E3: hash = okjhc

**8) top (string hash)**

Equivalence class partitions:

Length of hash

E1 value< 1, E2 value>1

E3 Contains any character of: [o-l-i-a]

Boundaries:

Length of hash

0 1 2

E1 E2

E3: hash = basdv

**9) bottom (string hash)**

Equivalence class partitions:

Length of hash

E1 value< 1, E2 value>1

E3 Contains any character of: [o-l-i-a]

Boundaries:

Length of hash

0 1 2

E1 E2

E3: hash = niugh

**10) coverBoundingBoxLongs (topLeftLat, topLeftLon, bottomRightLat,BottomRightLon, Length)**

Equivalence partitions

E1 Length>0

E2 topLeftLat>= bottomRightLat

Boundaries:

E1: 0 1

E2: topLeftLat = bottomRightLat+1

E3: topLeftLat = bottomRightLat-1

**11) neighbours(String Hash)**

Equivalence partitions

E1 emty list

E2 non-emptyList

Boundaries:

E1: null

E2: 1 element in the array

**12)FromLongToString(Long hash)**

Equivalence partitions

E1: null

E2: non-null

Boundaries:

E1: null

E2: non-null and doesn’t contain any character from (a,i,l,o)

E3: E2: non-null and contain any character from (a,i,l,o)

**13) refineInterval(double []interval, cd,mask)**

Equivalence partitions

E1 emty list

E2 non-emptyList

Boundaries:

E1: null

E2: 1 element in the array

**14) checkHash()**

Equivalence partitions

E1: null

E2: non-null

Boundaries:

E1: null

E2: non-null and doesn’t contain any character from (a,i,l,o)

E3: non-null and contain any character from (a,i,l,o)

**15) adjacentHash(hash,direction,steps)**

Equivalence partitions

E1: Length>0

E2: Length<=0

Boundaries:

E1: Length>0

E2: Length<0

E3: Length=0

**16)hashLengthToCoverBoundingBox**

Equivalence partitions

E1: Length>0

E2: Length<=0

Boundaries:

E1: Length>0

E2: Length<0

E3: Length=0

**17)hashContains(hash,lan,Lon)**

Equivalence partitions

E1: True

E2: False

Boundaries:

E1: True

E2: False

E3: null

**18) coverBoundingBox**

Equivalence class partitions:

E1: Length < 1

E2: Length >12

E3: 1< Length <12

Boundaries:

0 1 2 11 12 13

E1 E2 E3

**19) coverBoundingBoxMaxHashes**

Equivalence class partitions:

E1: Maxhash<=12

E2: Maxhash>12

Boundaries:

11 12 13

E1 E2

**20) calculateHeightDegrees**

Equivalence class partitions: n

E1: n<=0

E2: n>0

Boundaries:

-1 0 1

E1 E2

**21) calculateWidthDegrees**

Equivalence class partitions: n

E1: n<=0

E2: n>0

Boundaries:

-1 0 1

E1 E2

**22) GridAsString**

Equivalence class partitions:

E1: empty text

E2: non-empty text

Boundaries:

E1: empty text

E2: non-empty text

E3: Null