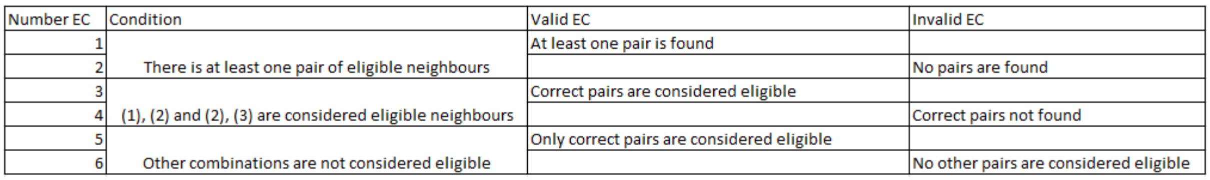
Test case Design

Black box Testing

Equivalence classes:



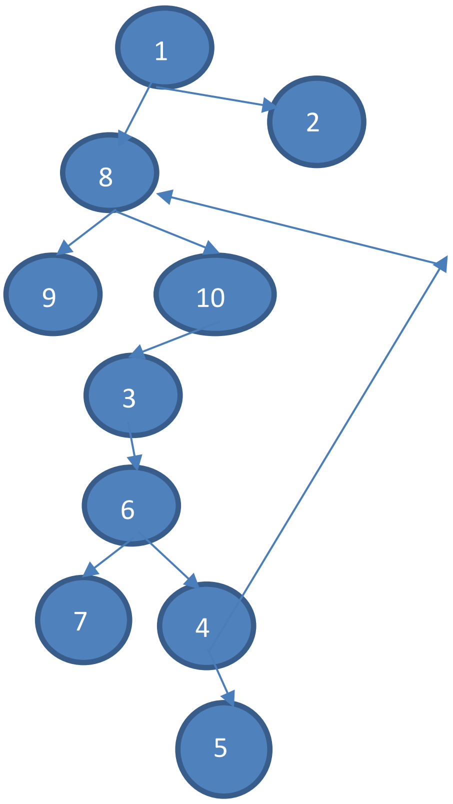
Number EC	Condition	Valid EC	Invalid EC
1	There is at least one pair of eligible neighbours	At least one pair is found	
2			No pairs are found
3	(1), (2) and (2), (3) are considered eligible neighbours	Correct pairs are considered eligible	
4			Correct pairs not found
5	Other combinations are not considered eligible	Only correct pairs are considered eligible	
6			No other pairs are considered eligible

Boundary value analysis was not done as there are no boundary values to test.

White Box testing

Code covered:

```
[1] if (animals.size() <= 1) {
   [2]return animals;
}

do {
   notFoundConflicts = true;
   Optional<Pair<Integer, Integer>> foundAnimalsPair = FindPair(newNeighbours);
  [3] if (foundAnimalsPair.isPresent()) {
      notFoundConflicts = false;
      int first = foundAnimalsPair.get().getFirst();
      int second = foundAnimalsPair.get().getSecond();
      InsertCow(newNeighbours, Math.min(first, second) + 1);
   }
 [4] } while (notFoundConflicts == false);


 [5] return newNeighbours;

public Optional<Pair<Integer, Integer>>

 FindPair(List<Integer> animals) {
   for (int index = 0; index < animals.size() - 1; index++) {
      int currentAnimal = animals.get(index);
      int nextAnimal = animals.get(index + 1);
      Pair<Integer, Integer> neighbours = new Pair<>(currentAnimal, nextAnimal);
      Pair<Integer, Integer> neighboursReverse = new Pair<>(nextAnimal, currentAnimal);

     [8]  if (neighboursPairs.containsKey(neighbours) ||
neighboursPairs.containsKey(neighboursReverse)) {
        [9]  return Optional.of(new Pair<>(index, index + 1));
       }
    }
  [10] return Optional.empty();
}

public void InsertCow(List<Integer> animals, int position) {
  [6] if(position < 0 || position >= animals.size()){
      [7] throw new IndexOutOfBoundsException();
   }
   animals.add(position, AnimalType.Cow);
}
```



Cyclomatic complexity		
CC_1	regions	3
CC_2	E-N+2	3
CC_3	Predicate+1	3

Individual Paths	
1	1,2
2	1,8,9,3,6,7
3	1,8,10,3,6,7
4	1,8,9,3,6,4,5
5	1,8,10,3,6,4,5

Integration Testing

Chosen way of testing is Big Bang testing, so we test components individually based on given input and expected output.

Tests for the BeFriends method:

TestBeFriends_ConflictInput_ShouldReturnAnimalsWithCowsBetween

Input: pair of lists of animals, first without cows added, second with cows correctly added

Expected output: True

This test compares lists generated by the BeFriends method with manually written lists for correctnness

TestBeFriends_NoConflictInInput_ShouldReturnTheSameList

Input: lists of integers representing animals where no cows can be added

Expected output: True

This test verifies if the method adds cows in places where it shouldn't.

TestBeFriends_EmptyOrOneElementInInput_ShouldReturnTheSameInputAsOutput

Input: List of integers of at most one element

Expected output: True

This test verifies if the method adds any values to strings that are too small to be modified