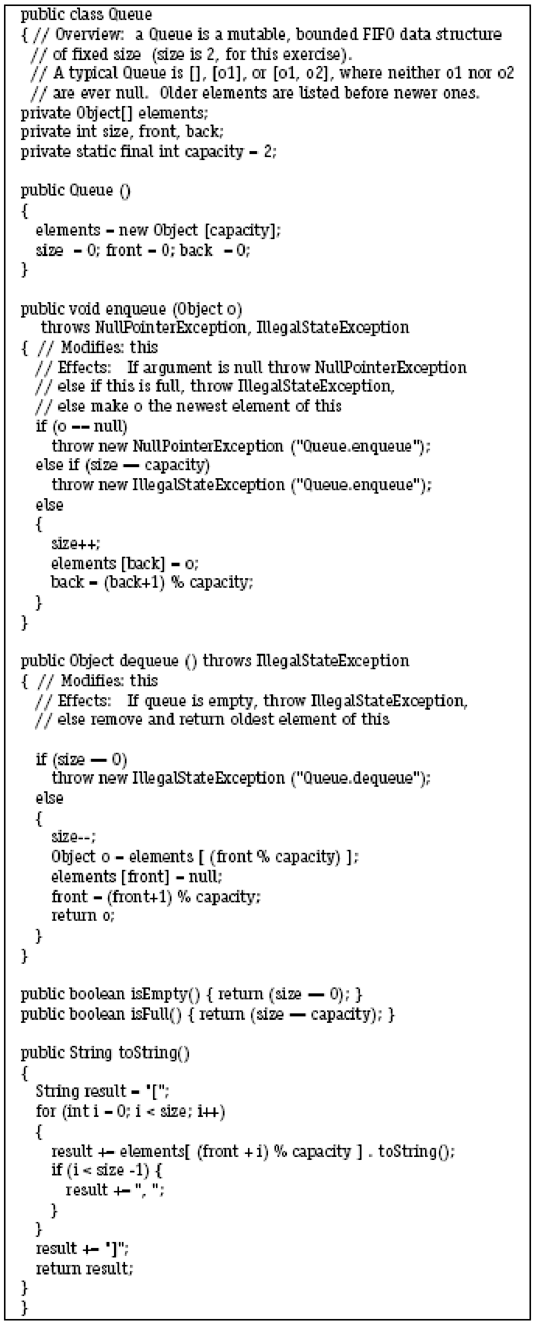
Question1: FSM

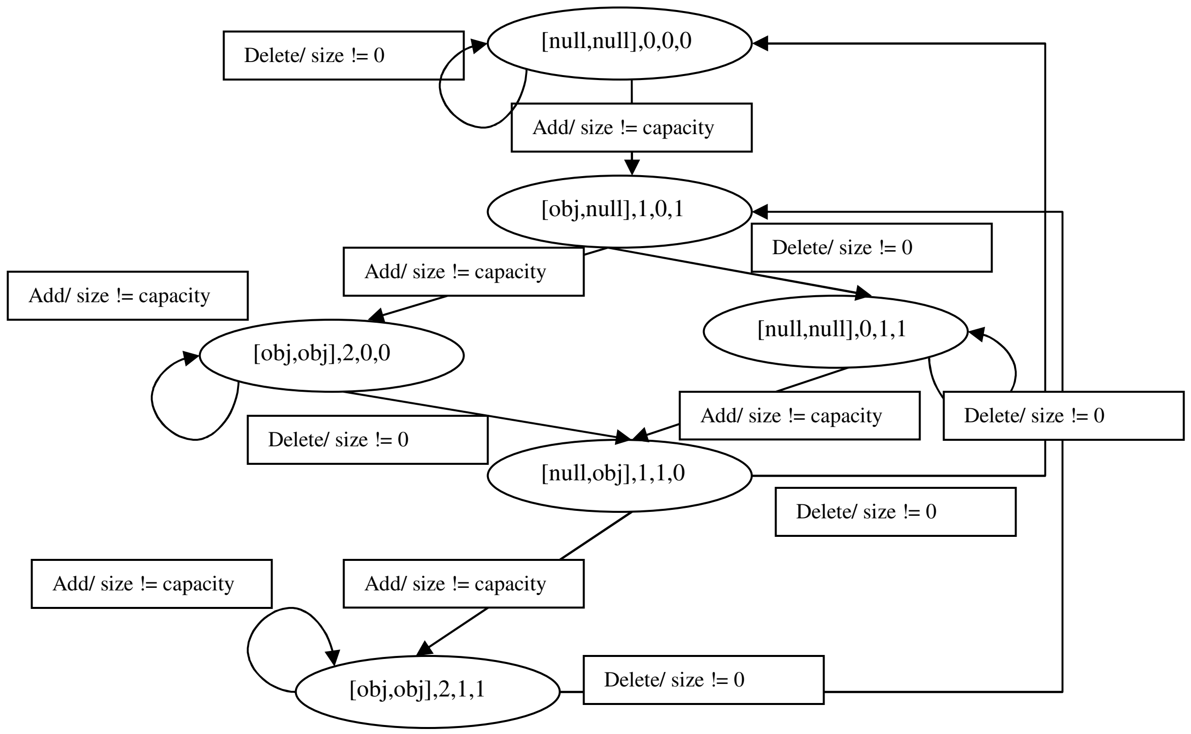


```
public class Queue
{ // Overview:  a Queue is a mutable, bounded FIFO data structure
  // of fixed size  (size is 2, for this exercise).
  // A typical Queue is [], [o1], or [o1, o2], where neither o1 nor o2
  // are ever null.  Older elements are listed before newer ones.
private Object[] elements;
private int size, front, back;
private static final int capacity = 2;

public Queue ()
{
   elements = new Object [capacity];
   size  = 0; front = 0; back  = 0;
}

public void enqueue (Object o)
    throws NullPointerException, IllegalStateException
{  // Modifies: this
   // Effects:   If argument is null throw NullPointerException
   // else if this is full, throw IllegalStateException,
   // else make o the newest element of this
   if (o == null)
      throw new NullPointerException ("Queue.enqueue");
   else if (size == capacity)
      throw new IllegalStateException ("Queue.enqueue");
   else
   {
      size++;
      elements [back] = o;
      back = (back+1) % capacity;
   }
}

public Object dequeue () throws IllegalStateException
{  // Modifies: this
   // Effects:   If queue is empty, throw IllegalStateException,
   // else remove and return oldest element of this

   if (size == 0)
      throw new IllegalStateException ("Queue.dequeue");
   else
   {
      size--;
      Object o = elements [ (front % capacity) ];
      elements [front] = null;
      front = (front+1) % capacity;
      return o;
   }
}

public boolean isEmpty() { return (size == 0); }
public boolean isFull() { return (size == capacity); }

public String toString()
{
   String result = "[";
   for (int i = 0; i < size; i++)
   {
      result += elements[ (front + i) % capacity ] . toString();
      if (i < size -1) {
         result += ", ";
      }
   }
   result += "]";
   return result;
}
}
```

Solution: A state [elements, size, front, back]

Total number of states: 4*3*2*2 = 48

Not all reachable. Reachable states are shown in FSM:



Question2: Interprocedural Testing: extracting d-u paths:

```java
// Jeff Offutt  - June 1989, Java version 2003

// stutter checks for repeat words in a text file.
// It prints a list of repeat words, by line number.
// stutter will accept standard input or a list
// of file names.

import java.io.*;

class stutter
{
   // Class variables used in multiple methods.
   private static boolean lastdelimit = true;
   private static String curWord = "", prevWord = "";
   private static char delimits [] =
               {'', ' ', ',', '.', '!', '-', '+', '=', ';', ':', '?', '&', '{', '}', '\\'};

//************************************************
// main parses the arguments, decides if stdin
// or a file name, and calls Stut().
//************************************************
public static void main (String[] args) throws IOException
{
   String fileName;
   FileReader myFile;
   BufferedReader inFile = null;

   if (args.length == 0)
   {  // no file, use stdin
      inFile = new BufferedReader (new InputStreamReader (System.in));
   }
   else
   {
      fileName = args [0];
      if (fileName == null)
      {  // no file name, use stdin
         inFile = new BufferedReader (new InputStreamReader (System.in));
      }
      else
      {  // file name, open the file.
         myFile = new FileReader (fileName);
         inFile = new BufferedReader (myFile);
      }
   }

   Stut (inFile);
}

//************************************************
//************************************************
private static void Stut (BufferedReader inFile) throws IOException
{
   String inLine;
   char c;
   int linecnt = 1;
```

```
 56
 57     while ((inLine = inFile.readLine()) != null)
 58     {  // For each line
 59
 60        for (int i=0; i<inLine.length(); i++)
 61        {  // for each character
 62           c = inLine.charAt(i);
 63
 64           if (IsDelimit (c))
 65           {  // Found an end of a word.
 66              checkDupes (linecnt);
 67           }
 68           else
 69           {
 70              lastdelimit = false;
 71              curWord = curWord + c;
 72           }
 73        }
 74        linecnt++;
 75        checkDupes (linecnt);
 76     }
 77  }  // end Stut
 78
 79  //************************************************
 80  //************************************************
 81  private static void checkDupes (int line)
 82  {
 83     if (lastdelimit)
 84        return; // already checked, keep skipping
 85     lastdelimit = true;
 86     if (curWord.equals(prevWord))
 87     {
 88        System.out.println ("Repeated word on line " + line + ": " + prevWord+ " " + curWord);
 89     }
 90     else
 91     {
 92        prevWord = curWord;
 93     }
 94     curWord = "";
 95  }  // end checkDupes
 96
 97  //************************************************
 98  //************************************************
 99  private static boolean IsDelimit (char C)
100  {
101     for (int i = 0; i < delimits.length; i++)
102        if (C == delimits [i])
103           return (true);
104     return (false);
105  }
106
107  } // end class stutter
```

Solution:
The callsites are:
i. Line 46, main() → Stut()
ii. Line 64, Stut() → IsDelimit()
iii. Line 66, Stut() → checkDupes()
iv. Line 75, Stut() → checkDupes()

List all du-pairs for each call site.

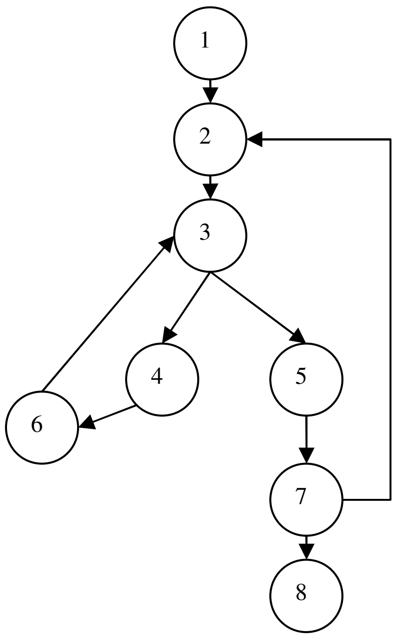
(main(), curWord, 14) → (Stut(), curWord, 71) – line 46
ii. (main(), inFile, 30) → (Stut(), inFile, 57) – line 46
iii. (main(), inFile, 37) → (Stut(), inFile, 57) – line 46
iv. (main(), inFile, 42) → (Stut(), inFile, 57) – line 46
v. (Stut(), c, 62) → (IsDelimit(), C, 102) – line 64
vi. (Stut(), linecnt, 55) → (checkDupes(), line, 88) – line 66
vii. (Stut(), linecnt, 74) → (checkDupes(), line, 88) – line 66
viii. (Stut(), curWord, 71) → (checkDupes(), curWord, 86) – line 66
ix. (Stut(), lastdelimit, 70) → (checkDupes(), lastdelimit, 83) – line 66
x. (checkDupes(), curWord, 94) → (Stut(), curWord, 71) – line 66
xi. (Stut(), linecnt, 74) → (checkDupes(), line, 88) – line 75

(Note that the def at 55 is not last-def)
xii. (Stut(), curWord, 71) → (checkDupes(), curWord, 86) – line 75
xiii. (Stut(), lastdelimit, 70) → (checkDupes(), lastdelimit, 83) – line 75
xiv. (checkDupes(), curWord, 94) → (Stut(), curWord, 71) – line 75

Create test data to satisfy All-Coupling Use Coverage for Stutter.
• t1:
hello
• t2:
Hello hello
• t3:
first line
hello hello
i. (main(), curWord, 14) → (Stut(), curWord, 71) – line 46
Test needs to start with a non-delimiter: t1.
ii. (main(), inFile, 30) → (Stut(), inFile, 57) – line 46
Test needs to come from standard input.
iii. (main(), inFile, 37) → (Stut(), inFile, 57) – line 46
Test not possible in normal execution.
iv. (main(), inFile, 42) → (Stut(), inFile, 57) – line 46
Test needs to come from file.
v. (Stut(), c, 62) → (IsDelimit(), C, 102) – line 64
Test needs to be nonempty: t1.
(Stut(), linecnt, 55) → (checkDupes(), line, 88) – line 66
Test needs to stutter on first line: t2.
vii. (Stut(), linecnt, 74) → (checkDupes(), line, 88) – line 66
Test needs to have on second or later lines: t3.
viii. (Stut(), curWord, 71) → (checkDupes(), curWord, 86) – line 66
Test needs to find a word, and then a delimiter: t2.
ix. (Stut(), lastdelimit, 70) → (checkDupes(), lastdelimit, 83) – line 66
Test needs to find a word, and then a delimiter: t2.
x. (checkDupes(), curWord, 94) → (Stut(), curWord, 71) – line 66
Test needs multiple words: t2.
xi. (Stut(), linecnt, 74) → (checkDupes(), line, 88) – line 75
Test needs to have on second or later lines: t3.
xii. (Stut(), curWord, 71) → (checkDupes(), curWord, 86) – line 75
Test needs to stutter: t2.
xiii. (Stut(), lastdelimit, 70) → (checkDupes(), lastdelimit, 83) – line 75
Test needs to be multiline and end a line with a non-delimeter: t3.
xiv. (checkDupes(), curWord, 94) → (Stut(), curWord, 71) – line 75
Test needs a line that ends with a non delimiter: t1.

Question3:

In the following CFG, if I have the following two test paths, do they have edge and node coverage? Can only using one of them suffice for testing the function?



T1: [1, 2, 3, 4, 6, 3, 5, 7, 2, 3, 5, 7, 8]
T2: [1, 2, 3, 5, 7, 2 , 3, 4, 6, 5, 7, 8]
Yes, they both have edge and node coverage.

No, both are needed, Example, in loop 2, 3, 5, 7, 2 an instruction may be executed causing an error in loop 3, 4, 6, resulting in an error in the final result.