Graph Coverage in Practice

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5522
Department of Computer Science
Idaho State University

ROAR

Outcomes

At the end of Today's Lecture you will be able to:

- Understand how to apply graph testing in Practice

The Steps

1. Pick a coverage criterion and review the code
2. Using a tool to read and analyze the code
3. Generate the CFG
4. Number the CFG (optional)
5. Identify test paths
6. Design Test Cases
7. Create Tests
8. Verify

Step 1 - Criterion Selection

- We will analyze a piece of code from the book
  - Language: Java
  - Size: 59 LOC
  - NOM: 2
- The Graph Coverage Criteria will be EPC
  - for ease of analysis

Step 2 - The Code

```
public class PatternIndex
{

   public static void main (String[] argv)
   {
      if (argv.length != 2)
      {
         System.out.println
            ("java PatternIndex Subject Pattern");
         return;
      }
      String subject = argv[0];
      String pattern = argv[1];
      int n = 0;
      if ((n = patternIndex (subject, pattern)) == -1)
         System.out.println
         ("Pattern string is not a substring of the subject string");
      else
         System.out.println
         ("Pattern string begins at character " + n);
   }
```

Idaho State University
Computer Science

Step 2 - The Code

```java
/**
 * Find index of pattern in subject string
 *
 * @param subject String to search
 * @param pattern String to find
 * @return index (zero-based) of first occurrence of pattern in subject; -1 if not found
 * @throws NullPointerException if subject or pattern is null
 */
public static int patternIndex (String subject, String pattern)
{
   final int NOTFOUND = -1;
   int  iSub = 0, rtnIndex = NOTFOUND;
   boolean isPat  = false;
   int subjectLen = subject.length();
   int patternLen = pattern.length();

   while (isPat == false && iSub + patternLen - 1 < subjectLen)
   {
      if (subject.charAt(iSub) == pattern.charAt(0))
      {
         rtnIndex = iSub; // Starting at zero
         isPat = true;
         for (int iPat = 1; iPat < patternLen; iPat ++)
         {
            if (subject.charAt(iSub + iPat) != pattern.charAt(iPat))
            {
               rtnIndex = NOTFOUND;
               isPat = false;
               /* MB: isPat = true; */
               break;  // out of for loop
            }
         }
```

ROAR

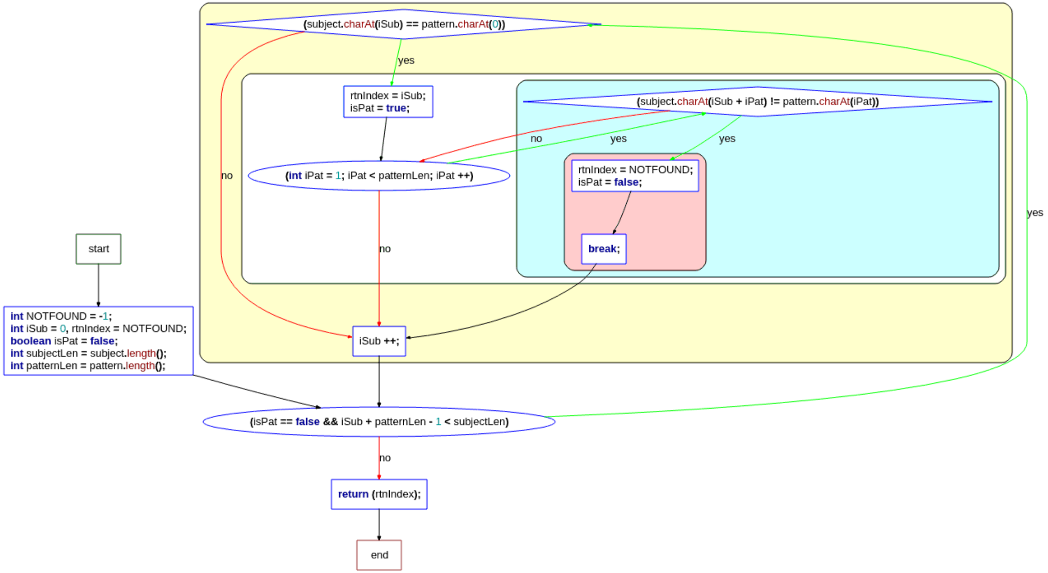
Step 2 - Analyzing

- For the Analysis we will use the following
  - SciTools Understand
  - http://scitools.com
- The tool is not free, but is free for educational use
- Can do many things but we will focus on CFG's for now

Let’s start with patternIndex()

Idaho State University
Computer Science

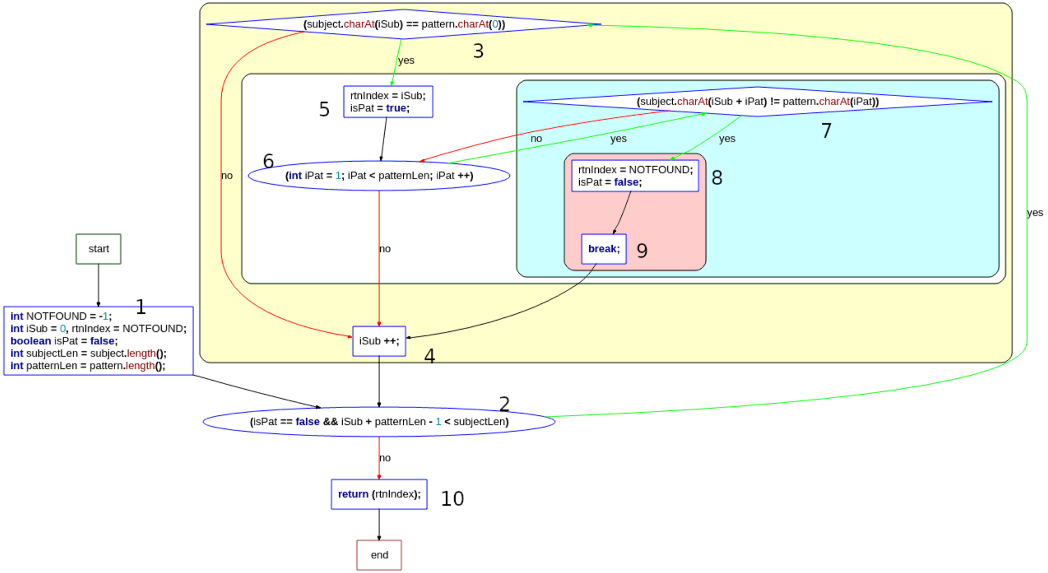
Step 3 - patternIndex() Results



start
int NOTFOUND = -1;
int iSub = 0, rtnIndex = NOTFOUND;
boolean isPat = false;
int subjectLen = subject.length();
int patternLen = pattern.length();
(subject.charAt(iSub) == pattern.charAt(0))
yes
no
rtnIndex = iSub;
isPat = true;
(int iPat = 1; iPat < patternLen; iPat ++)
no
(subject.charAt(iSub + iPat) != pattern.charAt(iPat))
no
yes
yes
rtnIndex = NOTFOUND;
isPat = false;
break;
iSub ++;
yes
(isPat == false && iSub + patternLen - 1 < subjectLen)
no
return (rtnIndex);
end

Idaho State University
Computer Science

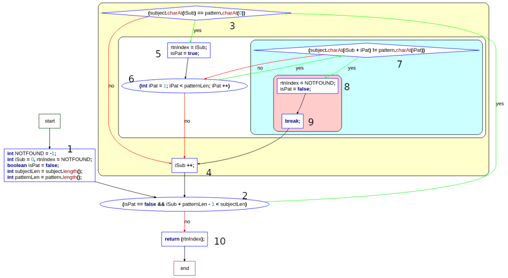
Step 4 - patternIndex()



start
1
int NOTFOUND = -1;
int iSub = 0, rtnIndex = NOTFOUND;
boolean isPat = false;
int subjectLen = subject.length();
int patternLen = pattern.length();
2
(isPat == false && iSub + patternLen - 1 < subjectLen)
3
(subject.charAt(iSub) == pattern.charAt(0))
yes
no
5
rtnIndex = iSub;
isPat = true;
6
(int iPat = 1; iPat < patternLen; iPat ++)
7
(subject.charAt(iSub + iPat) != pattern.charAt(iPat))
no
yes
yes
8
rtnIndex = NOTFOUND;
isPat = false;
9
break;
no
4
iSub ++;
yes
no
10
return (rtnIndex);
end

Idaho State University
Computer Science

Step 5 - patternIndex()



start
int NOTFOUND = -1;
int iSub = 0, rtnIndex = NOTFOUND;
boolean isPat = false;
int subjectLen = subject.length();
int patternLen = pattern.length();
1
(isPat == false && iSub + patternLen - 1 < subjectLen)
2
(subject.charAt(iSub) == pattern.charAt(0))
3
yes
no
rtnIndex = iSub;
isPat = true;
5
(int iPat = 1; iPat < patternLen; iPat ++)
6
no
(subject.charAt(iSub + iPat) != pattern.charAt(iPat))
7
no
yes
yes
rtnIndex = NOTFOUND;
isPat = false;
8
break;
9
iSub ++;
4
yes
no
return (rtnIndex);
10
end

- **Edge Pair Paths**
  - 1 [1, 2, 3], (2) [1, 2, 10], (3) [2, 3, 4]
  - 4 [2, 3, 5], (5) [3, 5, 6], (6) [3, 4, 2]
  - 7 [4, 2, 10], (8) [4, 2, 3], (9) [5, 6, 4]
  - 10 [5, 6, 7], (11) [6, 7, 6], (12) [6, 7, 8]
  - 13 [7, 6, 7], (14) [7, 6, 4], (15) [7, 8, 9]
  - 16 [8, 9, 4], (17) [9, 4, 2], (18) [6, 4, 2]

- **Test Paths**
  - ❶ [1, 2, 10]
  - ❷ [1, 2, 3, 4, 2, 10]
  - ❸ [1, 2, 3, 5, 6, 4, 2, 10]
  - ❹ [1, 2, 3, 5, 6, 7, 8, 9, 4, 2, 10]

- **Test Paths Continued**
  - ❺ [1, 2, 3, 5, 6, 7, 6, 4, 2, 10]
  - ❻ [1, 2, 3, 5, 6, 7, 6, 7, 8, 9, 4, 2, 10]
  - ❼ [1, 2, 3, 5, 6, 7, 6, 4, 2, 10]

ROAR

Step 5 - patternIndex()

- **Test Paths**
  - ❶ [1, 2, 10] -> 2
  - ❷ [1, 2, 3, 4, 2, 10] -> 1, 4, 5, 7, 9, 18
  - ❸ [1, 2, 3, 5, 6, 4, 2, 10] -> 1, 4, 5, 7, 9, 10
  - ❹ [1, 2, 3, 5, 6, 7, 8, 9, 4, 2, 10] -> 1, 4, 5, 7, 10, 12, 15, 16, 17
  - ❺ [1, 2, 3, 5, 6, 7, 6, 4, 2, 10] -> 1, 4, 5, 7, 10, 11, 18
  - ❻ [1, 2, 3, 5, 6, 7, 6, 7, 8, 9, 4, 2, 10] -> 1,4, 5, 7, 10, 11, 12, 13, 15, 16, 17
  - ❼ [1, 2, 3, 5, 6, 7, 6, 4, 2, 10] -> 1, 4, 5, 7, 10, 11, 14, 18
- Note: 4 and 5 are redundant
- So we only have 4 tests to create

Step 6 - patternIndex()

- Test Case 1: [1, 2, 10]
  - Inputs:
    - `subject = ""`
    - `pattern = "a"`
  - Expected: `-1`
- Test Case 2: [1, 2, 3, 4, 2, 10]
  - Inputs:
    - `subject = "Too"`
    - `pattern = "How"`
  - Expected: `-1`

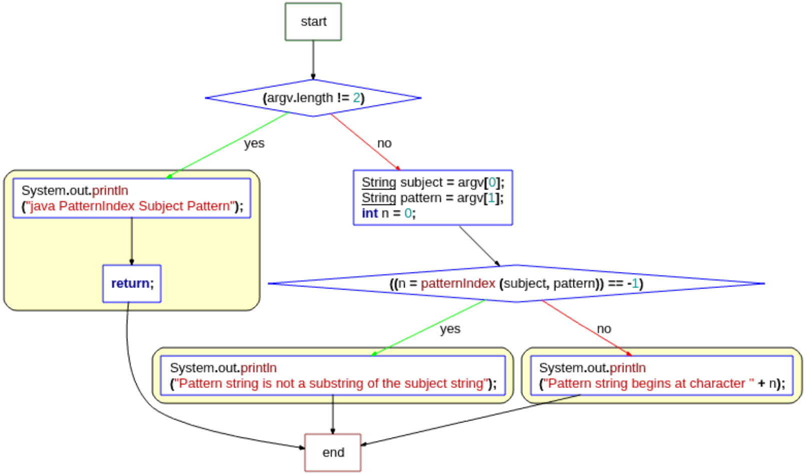
Step 6 - patternIndex()

- Test Case 3: [1, 2, 3, 5, 6, 7, 6, 7, 8, 9, 4, 2, 10]
  - Inputs:
    - `subject = "Too"`
    - `pattern = "Toa"`
  - Expected: `-1`
- Test Case 4: [1, 2, 3, 5, 6, 7, 6, 4, 2, 10]
  - Inputs:
    - `subject = "Foo"`
    - `pattern = "Fo"`
  - Expected: `0`

Step 7 & 8 - patternIndex()

Now on to main()

Step 3 - main()



```
System.out.println
("java PatternIndex Subject Pattern");
```

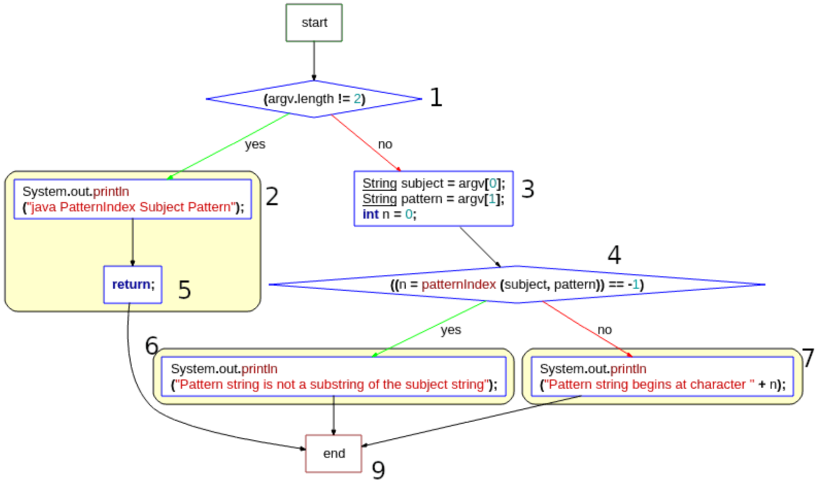
```
String subject = argv[0];
String pattern = argv[1];
int n = 0;
```

```
((n = patternIndex (subject, pattern)) == -1)
```

```
System.out.println
("Pattern string is not a substring of the subject string");
```

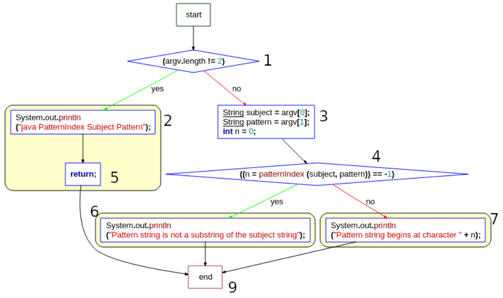
```
System.out.println
("Pattern string begins at character " + n);
```

Step 4 - main()


start
(argv.length != 2)
1
yes
no
System.out.println
("java PatternIndex Subject Pattern");
2
String subject = argv[0];
String pattern = argv[1];
int n = 0;
3
4
((n = patternIndex (subject, pattern)) == -1)
return;
5
yes
no
6
System.out.println
("Pattern string is not a substring of the subject string");
7
System.out.println
("Pattern string begins at character " + n);
end
9

Idaho State University
Computer Science

Step 5 - main()



start
(argv.length != 2)
1
yes
no
System.out.println
("java PatternIndex Subject Pattern");
2
return;
5
String subject = argv[0];
String pattern = argv[1];
int n = 0;
3
4
((n = patternIndex (subject, pattern)) == -1)
yes
no
6
System.out.println
("Pattern string is not a substring of the subject string");
System.out.println
("Pattern string begins at character " + n);
7
end
9

- **Edge-Pair Paths**
  1. [1, 2, 5]
  2. [1, 3, 4]
  3. [2, 5, 9]
  4. [3, 4, 6]
  5. [3, 4, 7]
  6. [4, 6, 9]
  7. [4, 7, 9]
- **Test Paths**
  1. [1, 2, 5, 9]
  2. [1, 3, 4, 6, 9]
  3. [1, 3, 4, 7, 9]

ROAR

Step 6 - main()

- Test Case 1: [1, 2, 5, 8, 9]
  - Input: `argv = []`
  - Expected: `"java PatternIndex Subject Pattern\n"`
- Test Case 2: [1, 3, 4, 6, 9]
  - Input: `argv = ["foo", "bar"]`
  - Expected: `"Pattern string is not a substring of the subject string\n"`
- Test Case 3: [1, 3, 4, 7, 9]
  - Input: `argv = ["foobar", "oba"]`
  - Expected: `"Pattern string begins at character 2\n"`

Step 7 & 8 - main()

Are there any questions?

ROAR