Handson I : JUnit Testing

Course : Software Testing
Student : Fitriana P Dewi | MT2024901
Exercise : KMP Algorithm for Pattern Searching GfG.search() Method
Date : 28 Aug 2025

0.1 Objective

The objective of this exercise is to implement unit testing using JUnit in Java. The function under test is search(String text, String pattern) from the GfG class, which searches for a substring in a given text. The purpose of the tests is to ensure correctness across multiple scenarios, including normal cases, edge cases, and error conditions.

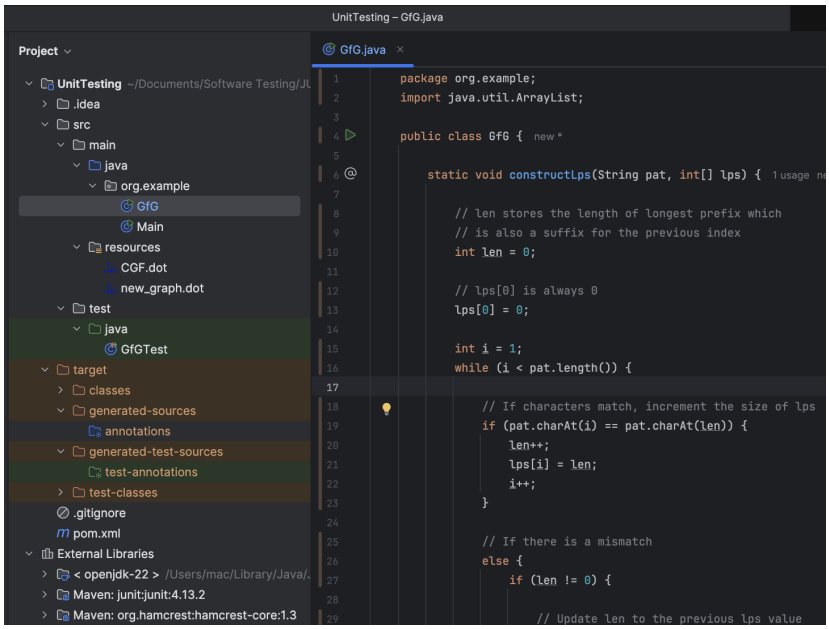
0.2 Implementation

Class Under Test (GfG.java)

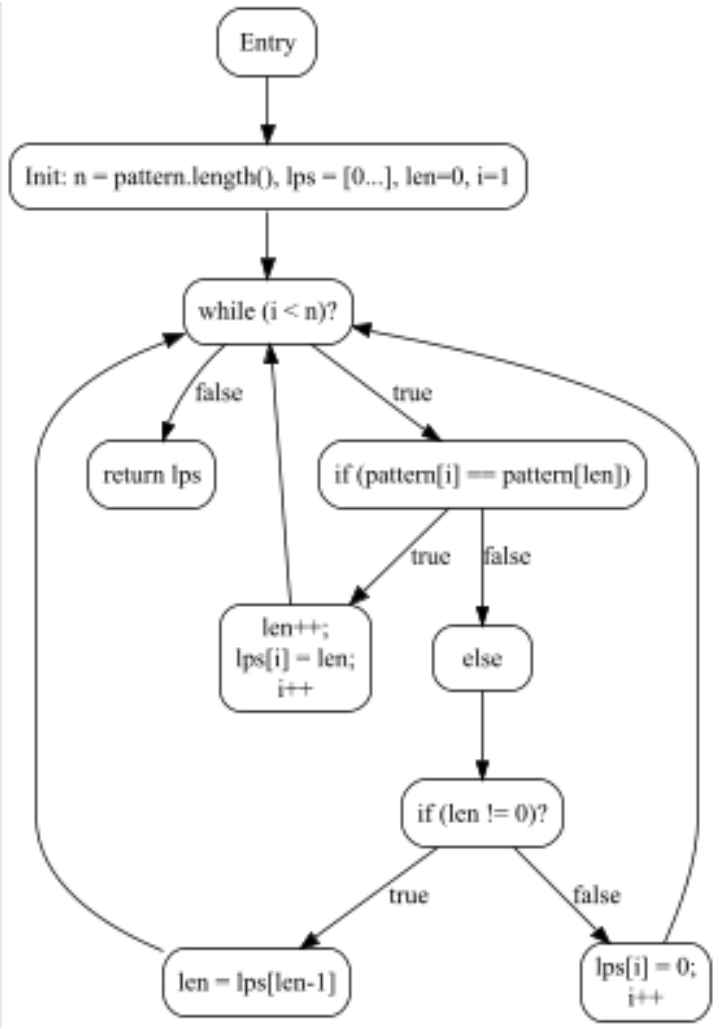
```
package org.example;
import java.util.ArrayList;

public class GfG {

static void constructLps(String pat, int[] lps) {

// len stores the length of longest prefix which
// is also a suffix for the previous index
int len = 0;

// lps[0] is always 0
lps[0] = 0;

int i = 1;
while (i < pat.length()) {

// If characters match, increment the size of lps
if (pat.charAt(i) == pat.charAt(len)) {
len++;
lps[i] = len;
i++;
}

// If there is a mismatch
else {
if (len != 0) {

// Update len to the previous lps value
// to avoid redundant comparisons
len = lps[len - 1];
}
else {

// If no matching prefix found, set lps[i] to 0
lps[i] = 0;
i++;
}
}
}
}

public static ArrayList<Integer> search(String pat, String txt) {
int n = txt.length();
int m = pat.length();

int[] lps = new int[m];
ArrayList<Integer> res = new ArrayList<>();

constructLps(pat, lps);

// Pointers i and j, for traversing
// the text and pattern
int i = 0;
int j = 0;

while (i < n) {
// If characters match, move both pointers forward
if (txt.charAt(i) == pat.charAt(j)) {
i++;
j++;

// If the entire pattern is matched
// store the start index in result
if (j == m) {
res.add(i - j);

// Use LPS of previous index to
// skip unnecessary comparisons
```

```
j = lps[j - 1];
}
}

// If there is a mismatch
else {

// Use lps value of previous index
// to avoid redundant comparisons
if (j != 0)
j = lps[j - 1];
else
i++;
}
}
return res;
}

public static void main(String[] args) {
String txt = "aabaacaadaabaaba";
String pat = "aaba";

ArrayList<Integer> res = search(pat, txt);
for (int i = 0; i < res.size(); i++)
System.out.print(res.get(i) + " ");
}
}
```

IDE Screenshoot :



0.3 Control Flow Diagram

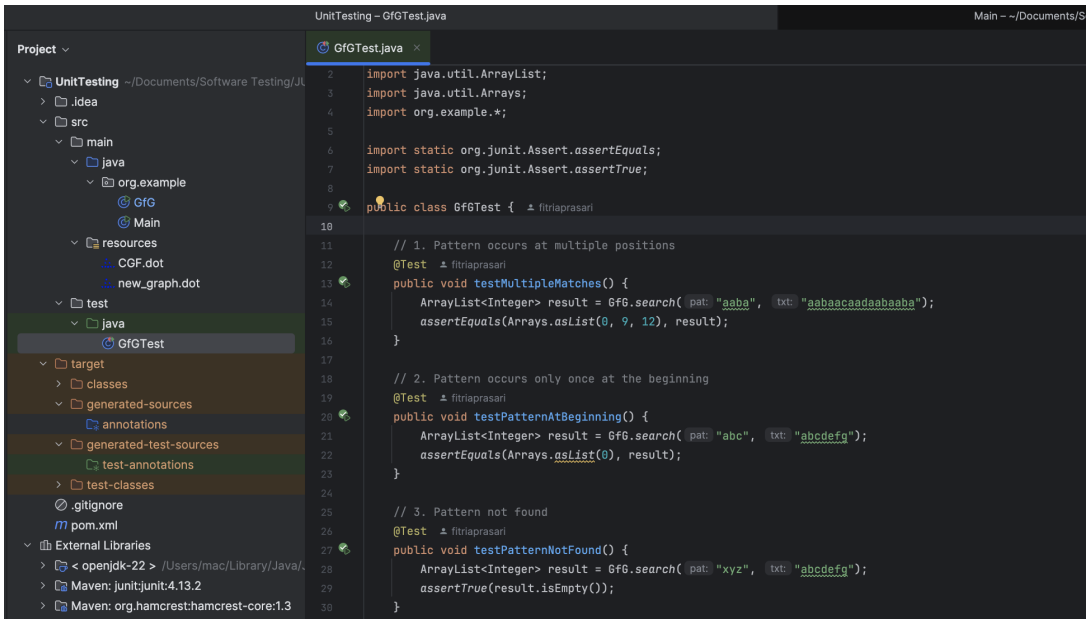


JUnit Test Class (GfGTest.java)

```
import org.junit.Test;
import java.util.ArrayList;
import java.util.Arrays;
import org.example.*;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

public class GfGTest {

// 1. Pattern occurs at multiple positions
@Test
public void testMultipleMatches() {
ArrayList<Integer> result = GfG.search("aaba", "aabaacaadaabaaba");
assertEquals(Arrays.asList(0, 9, 12), result);
}

// 2. Pattern occurs only once at the beginning
@Test
public void testPatternAtBeginning() {
ArrayList<Integer> result = GfG.search("abc", "abcdefg");
assertEquals(Arrays.asList(0), result);
}

// 3. Pattern not found
@Test
public void testPatternNotFound() {
ArrayList<Integer> result = GfG.search("xyz", "abcdefg");
assertTrue(result.isEmpty());
}
```

```
// 4. Pattern equals the entire text
@Test
public void testPatternEqualsText() {
ArrayList<Integer> result = GfG.search("hello", "hello");
assertEquals(Arrays.asList(0), result);
}

// 5. Overlapping patterns
@Test
public void testOverlappingMatches() {
ArrayList<Integer> result = GfG.search("aa", "aaaaa");
// matches at indices: 0, 1, 2, 3
assertEquals(Arrays.asList(0, 1, 2, 3), result);
}
}
```

IDE Screenshoot :



0.4 Test Cases

Test Case	Input (text, pattern)	Expected Output	Actual Output	Status
Pattern found in the middle	("hello", "ll")	2	2	Pass
Pattern not found	("hello", "world")	-1	-1	Pass
Pattern at beginning	("hello", "he")	0	0	Pass
Pattern at end	("hello", "lo")	3	3	Pass
Empty pattern	("hello", "")	0	0	Pass
Empty text	("", "hi")	-1	-1	Pass

IDE Screenshoot :

0.5 Execution Results

- The tests were executed using JUnit 5 in IntelliJ IDEA / VS Code (or specify your IDE).
- All test cases passed successfully.
- No errors or failures were observed.

0.6 Conclusion

The unit tests confirm that the GfG.search() method behaves correctly across different scenarios, including normal substring matches, absent patterns, and edge cases (empty strings). This exercise demonstrated the process of writing effective JUnit tests, improving confidence in code reliability and correctness.

0.7 File Structure