Aufgabe 1

a) Anweisungsüberdeckung

(1)

zeichen = 'h'
alphabet = {a, b, c, d, e, f, g, h, i, j, k}

(2)

zeichen = 'h'
alphabet = {}

Zuerst werden Zeilen 1-4 durchlaufen. Im ersten Schleifendurchlauf ist die erste Entscheidung war, im zweiten die zweite und schließlich die dritte. Um das "return -1" zu testen ist ein eigener Testfall (2) notwendig, da dort die Schleife garnicht erst betreten werden darf. Das wird erreicht, indem eine Kette übergeben wird, welche en gesuchten Buchstaben nicht enthält.

b) Zweigüberdeckung

(1)

zeichen = 'h'
alphabet = {a, b, c, d, e, f, g, h ,i, j, k}

im 1. Lauf: 1. if true
im 2. Lauf: 2. if true
im 3. Lauf: 1. if true
im 4. Lauf: 3. if true

(2)

zeichen = 'd'
alphabet = {a, b, c, d, e, f, g, h ,i, j, k}

im 1. Lauf: 2. if true
im 2. Lauf: 1. if true
im 3. Lauf: 3. if true

(3)

zeichen = 'h'
alphabet = {}

Im Test (1) wurde das Zeichen bewusst so gewählt, dass die Zweige wie aufgezeigt durchlaufen werden, erst wird rechts der mitte weiter gesucht, anschließend links. Der zweite Fall verhält sich zum ersten gegensätzlich, es wird also zunächst links weiter gesucht und anschließend rechts, womit insgesamt eine Zweigüberdeckung innerhalb der Schleife erreicht wird. Testfall (3) betritt die Schleife nicht und deckt somit hier den letzten Zweig innerhalb der Funktion ab.

c) du-Ketten:

1 [erstes, erstes = 0, while (erstes <= letztes)]
2 [erstes, erstes = 0, final int mitte = erstes + ((letztes - erstes) / 2)]
3 [erstes, erstes = mitte+1, while (erstes <= letztes)]
4 [erstes, erstes = mitte+1, final int mitte = erstes + ((letztes - erstes) / 2)]

5 [letztes, letztes = alphabet.length-1, while (erstes <= letztes)]
6 [letztes, letztes = alphabet.length-1, final int mitte = erstes + ((letztes - erstes) / 2)]
7 [letztes, letztes = mitte - 1, while (erstes <= letztes)]
8 [letztes, letztes = mitte - 1, final int mitte = erstes + ((letztes - erstes) / 2)]

9 [mitte, final int mitte = erstes + ((letztes - erstes) / 2), alphabet[mitte] < zeichen]
10 [mitte, final int mitte = erstes + ((letztes - erstes) / 2), alphabet[mitte] > zeichen]
11 [mitte, final int mitte = erstes + ((letztes - erstes) / 2), erstes = mitte + 1]
12 [mitte, final int mitte = erstes + ((letztes - erstes) / 2), letztes = mitte - 1]
13 [mitte, final int mitte = erstes + ((letztes - erstes) / 2), return mitte]

14 [zeichen, final char zeichen, alphabet[mitte] < zeichen]
15 [zeichen, final char zeichen, alphabet[mitte] > zeichen]

16 [alphabet, final char[] alphabet, int letztes = alphabet.length - 1]
17 [alphabet, final char[] alphabet, alphabet[mitte] < zeichen]
18 [alphabet, final char[] alphabet, alphabet[mitte] > zeichen]

d) Test für du-Ketten

(1)

zeichen = 'h'
alphabet = {a, b, c, d, e, f, g, h ,i, j, k}
Ketten: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18

Hier reicht sogar ein einziger Test, um alle du-Ketten zu durchlaufen. Die Idee hier bei ist, dass während der Ausführung einmal rechts (Ketten 3, 4, 11, 14, ...) von der Mitte und ein weiteres mal links (7,8,12,15,...) davon weiter gesucht werden muss. Somit ergibt sich, dass auch alle du-Ketten einmal durchlaufen werden.