Task 3
 Boyer-Moore Algorithm Application
Use the Boyer-Moore algorithm to write a function that finds the last occurrence of a substring in a given string and returns its index. Explain why this algorithm can outperform others in certain scenarios.
ANS:

```
package Day17;
public class BoyerMoor {
  public static int boyerMooreLastOccurrence(String text, String
pattern) {
      int textLength = text.length();
      int patternLength = pattern.length();
      if (patternLength > textLength) {
         return -1; // Pattern is longer than text, no match possible
      }
      int[] badChar = createBadCharTable(pattern);
      int lastIndex = -1;
      int shift = 0;
      while (shift <= (textLength - patternLength)) {
         int j = patternLength - 1;
         while (j >= 0 && pattern.charAt(j) == text.charAt(shift + j)) {
            j--;
         }
         if (j < 0) {
            lastIndex = shift;
            shift += (shift + patternLength < textLength) ?
patternLength - badChar[text.charAt(shift + patternLength)] : 1;
         } else {

            shift += Math.max(1, j - badChar[text.charAt(shift + j)]);
         }
      }
      return lastIndex;
  }
  private static int[] createBadCharTable(String pattern) {
      final int ALPHABET_SIZE = 256; // Assuming ASCII character
set
      int[] badChar = new int[ALPHABET_SIZE];
      for (int i = 0; i < ALPHABET_SIZE; i++) {
         badChar[i] = -1;
      }
      for (int i = 0; i < pattern.length(); i++) {
         badChar[pattern.charAt(i)] = i;
      }
      return badChar;
  }
  public static void main(String[] args) {
      String text = "ABABDABACDABABCABAB";
      String pattern = "ABABCABAB";
      int lastIndex = boyerMooreLastOccurrence(text, pattern);
      if (lastIndex != -1) {
         System.out.println("Last occurrence of pattern found at
index " + lastIndex);
      } else {
         System.out.println("Pattern not found in the text.");
      }
  }
}
```

OUTPUT:

```
Last occurrence of pattern found at index 10
```

Why Boyer-Moore Can Outperform Other Algorithms:
Efficiency with Skips:

- The Boyer-Moore algorithm can skip large sections of the text rather than checking each character sequentially. This is due to its use of heuristics (bad character and good suffix) that provide information on how far to skip ahead in the text when a mismatch occurs.
- Bad Character Heuristic:This heuristic allows the algorithm to skip over characters in the text that are not in the pattern, reducing the number of comparisons significantly.
- Good Suffix Heuristic (not implemented in this example):This heuristic allows the algorithm to skip sections of the text based on suffix matches within the pattern itself, further improving efficiency.
- Real-world Performance:In practice, the Boyer-Moore algorithm often performs better than the naive or even some other advanced string matching algorithms, especially with larger alphabets and longer patterns.