Note:
Java's default String.contains() is a brute-force algorithm.

27.2 Improving Brute-Force

Suppose the following inputs:

Text: a a a a a a h a a a a a ...
Pattern: a a a a a a a

In this case, then any pattern that includes the h is a waste of time. So, to prevent unnecessary checks, we can increment j to be past the h, instead of just adding one.

27.3 Boyer-Moore

When a mismatch occurs, see which character caused the mismatch in the text, and shift the pattern accordingly. In order to do so, we must pre-process the pattern to learn which characters are where. We will build a "Last Occurrence Table", in which we'll keep track of the index of the last occurrence of that character in the pattern.

Note:
Java has the method "getOrDefault()", which will return the index of the given character, but will return a default value if it is not found.

LastOccurrenceTable pseudocode

```
makeLOT(pattern)
  table = HashMap<char, integer>

  for j : 0 -> len(pattern) - 1
    table.put(pattern[j], j)

  return table
```

27.3.1 Boyer-Moore cases

Case 1: Mismatched character is NOT in pattern
Shift j to the first character after the mismatched one.

Case 2: Mismatched char is in pattern, but before current index
Solution: Align text char with the matching character in pattern.

Warning!
Do not forget about this third case! Prof. Faulkner says this is *extremely* common for people to miss on exams.

Case 3: Mismatched char is in pattern, but after the current index.
If our text is the word "allege", and the pattern is the word "enrage", then we need to ensure that we do not shift the pattern LEFTWARDS. So, in this case, we will only shift by 1.

27.3.2 Time Complexity

Best case time complexity for Boyer-Moore is $O(\frac{n}{m} + m)$. This means that as your pattern gets larger, Boyer-Moore becomes more and more efficient, because you can skip larger portions of the text at a time.