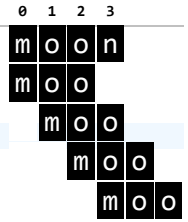
More Efficient Substrings



Here's an illustration of the loop we just wrote, using it to count the number of times that `moo` appears in `moon`.

▶ *Watch the video*

You'll notice that the last two iterations of the loop are not necessary, since we're comparing `moo` to `on`, and then a last time to `n`.

You may have encountered this situation in Java, where you'd solve it by changing the loop like this:

```
for (size_t i = 0, len = s1.size() - (s2.size() - 1); i < len; ++i)
```

Now, your code correctly works for the example shown here, only completing the first two repetitions, and skipping the two extraneous ones at the end.

However, what if the string `s1` contained `m` instead of `moon`. In Java, the expression `s1.size() - (s2.size() - 1)` would result in a negative number (`1 - (3 - 1)->-1`), and the loop would work correctly. In C++, **that's not the case**, since the types are **unsigned** numbers, so the result "wraps around" to a very large number, and your program crashes.

That means, to get this to work correctly for **any** input in C++, you need to add a **loop guard** like this:

```
size_t len1 = s1.size(), len2 = s2.size();
if (len2 <= len1) {
    for (size_t i = 0, len = len1 - (len2 - 1); i < len; ++i) {
        if (s1.substr(i, len2) == s2) { count++; }
    }
}
```