Theory: Searching a substring in Python

54 minutes 0 / 5 problems solved

Skip this topic Start practicing

311 users solved this topic. Latest completion was 2 days ago.

One way to formulate the substring searching problem is the following: given two strings, *text* and *pattern*, we need to identify whether there is at least one occurrence of the *pattern* in the *text*. The simplest and most natural way to solve this problem is to sequentially consider all substrings of the *text* whose length is equal to the length of the *pattern* and compare them with the *pattern* itself. If at least in one case all corresponding symbols match, the *pattern* is found. If none of such attempts were successful, we should indicate that there is no *pattern* in the text. In this topic, we will consider how this simple algorithm can be implemented in Python.

§1. Implementation in Python

Below is an implementation of the simplest substring searching algorithm in Python:

```python
def contains(text, pattern):
    for i in range(len(text) - len(pattern) + 1):
        found = True

        for j in range(len(pattern)):
            if text[i + j] != pattern[j]:
                found = False
                break

        if found:
            return True

    return False
```

The function named `contains` takes two strings, `text` and `pattern`, as input and returns `True` if `text` contains `pattern` and `False` otherwise.

At each step of the outer `for` loop, we create a variable named `found` and initialize it with `True`. Then, in the inner `for` loop, we start comparing `pattern` with the current substring of `text`. If at least one of the corresponding symbols doesn't match, we set the variable `found` to `False` and break the inner loop. After the inner `for` loop is done, we check the state of the `found` variable. If it remains `True`, this means that each symbol of `pattern` matches the current substring. In this case, we return `True` indicating that `pattern` is found. Otherwise, we move to the next iteration and start considering the next substring. In case none of the comparisons were successful, that is, the outer `for` loop finishes all iterations, the function returns `False` indicating that `pattern` is not found.

§2. Usage examples

Here is how this algorithm can be used:

```python
contains("abacabad", "cab")  # True
contains("abacabad", "abacabad")  # True
contains("aba", "")  # True
contains("abacabad", "hello")  # False
```

Note that the `in` operator in Python also can be used for substring searching. It works similarly to the proposed function:

Current topic:

Searching a substring in Python ···

Topic depends on:

✓ Searching a substring ···

✓ Algorithms in Python ···

Table of contents:

↑ Searching a substring in Python

§1. Implementation in Python

§2. Usage examples

Feedback & Comments

```
"aba" in "abacabad"  # True
"ada" in "abacabad"  # False
```

A built-in method for strings called `find` also solves the same problem:

```
"hello".find("el")  # 1
"hello".find("aba")  # -1
```

Unlike the described `contains` function and the `in` operator, it returns the position of the first occurrence or -1 if no occurrences are found.

Report a typo

37 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion