Rotate String

We are given two strings, `A` and `B`.

A *shift on* `A` consists of taking string `A` and moving the leftmost character to the rightmost position. For example, if `A = 'abcde'`, then it will be `'bcdea'` after one shift on `A`. Return `True` if and only if `A` can become `B` after some number of shifts on `A`.

```
Example 1:
Input: A = 'abcde', B = 'cdeab'
Output: true

Example 2:
Input: A = 'abcde', B = 'abced'
Output: false
```

Note:

- `A` and `B` will have length at most `100`.

Solution 1

We can easily see whether it is rotated if B can be found in (A + A).
For example, with A = “abcde”, B = “cdeab”, we have

```
“abcdeabcde” (A + A)
  “cdeab” (B)
```

B is found in (A + A), so B is a rotated string of A.

C++

```cpp
bool rotateString(string A, string B) {
    return A.size() == B.size() && (A + A).find(B) != string::npos;
}
```

Java

```java
public boolean rotateString(String A, String B) {
    return A.length() == B.length() && (A + A).contains(B);
}
```

Python

```python
def rotateString(self, A, B):
    return len(A) == len(B) and B in A + A
```

written by dnuang original link here

Solution 2

```java
class Solution {
    public boolean rotateString(String A, String B) {
        return A.length() == B.length() && (A + A).contains(B);
    }
}
```

written by shawngao original link here

Solution 3

```
bool rotateString(string A, string B) {
    return (A.length() == B.length()) && ((A + A).find(B) != string::npos);
}
```

written by DDev original link here