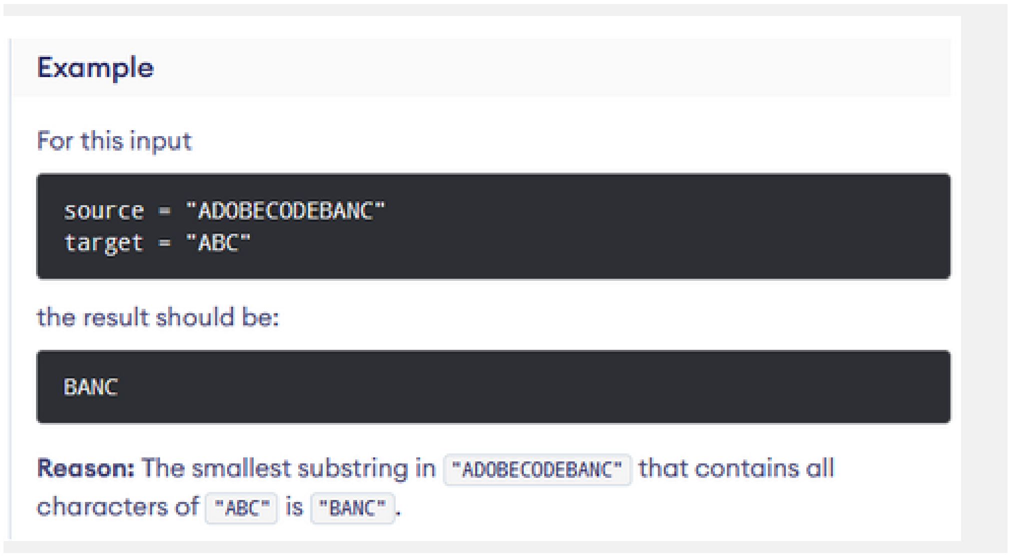
# MY APPROACH

For this challenge, could use challenge which focussed on smallest window of String within another String.



**Example**

For this input

```
source = "ADOBECODEBANC"
target = "ABC"
```

the result should be:

```
BANC
```

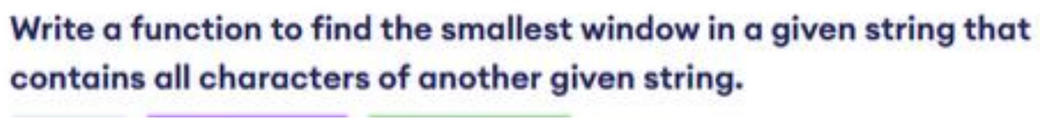
**Reason:** The smallest substring in `"ADOBECODEBANC"` that contains all characters of `"ABC"` is `"BANC"`.

## 1) All characters in String Y = BDCAB      FOUND IN STRING X

Y = BDCAB  and check if it appears in  X=ABCBDAB
If there is a single/multiple exact match (i.e. no unmatched characters) of entire String X, we would simply take the
length of Y...
And take indexes of String X (start => end)

## 2) In any other scenarios, the inclusion code will fail... Since we have to accommodate for exclusions,,,

## In the challenge completed:



**Write a function to find the smallest window in a given string that contains all characters of another given string.**

We would have discarded target string (equivalent to String X) if it contained an unmatched character. In this challenge we have the opportunity to acknowledge these. It is totally irrelevant on the occurrences or at which position. But if there are zero character matches in String X and String Y,  String X does not qualify for subsequence.

At the point where it does not find a match, we would include a counter for unMatched…
Once it has processed entire String Y, it would simply perform:
Length Y – unmatched character occurrences..
We would need to maintain the first index and last index match on String X
The longest common subsequence would be in which unmatched is minimum.

I can also maintain the permutation aspect of re-arranging String Y.