STRINGS-1

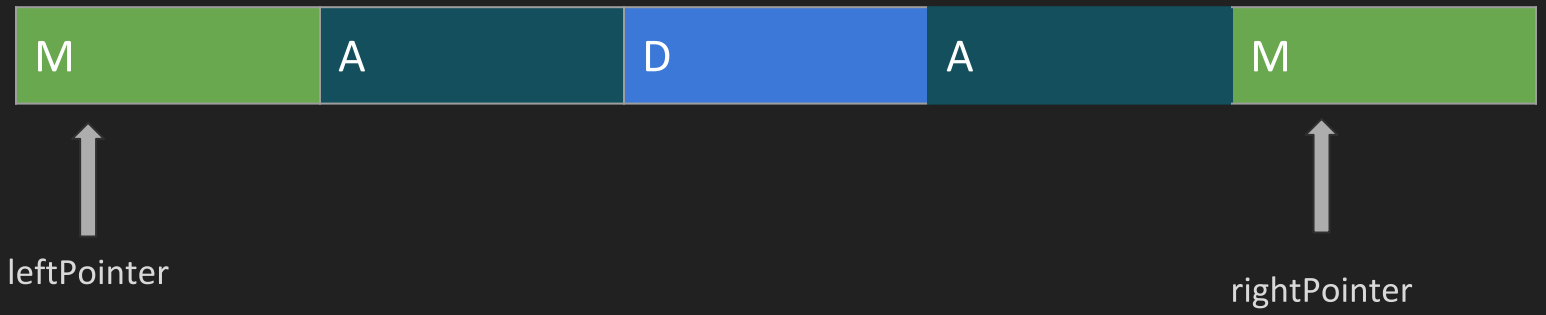
What we have already done in strings ? [RECAP]

1. Check if two strings are valid anagrams. (SESSION 1)
2. Check if we can group anagrams together. (SESSION 1)
3. Find indexes of anagram of a pattern string and a given text string. (SESSION 1) [SLIDING WINDOW TECHNIQUE]

What is a palindrome ?

STRINGS WHEN REVERSED STAY EXACTLY THE SAME.

Example : NAMAN, MADAM, “02-02-2020”, “borrow or rob”



Question: Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

YOU HAVE 10 MINUTES

EXAMPLE:

INPUT: “naman”,”normal”

OUTPUT: True, False

Link: https://leetcode.com/problems/valid-palindrome/

SOLUTION

```python
# Valid Palindrome

class Solution:
    def isPalindrome(self, s: str) -> bool:
        s = ''.join(filter(str.isalnum, s))
        left = 0
        right = len(s) - 1
        s = s.lower()
        print("This is s:",s)
        while left < right:
            if(s[left] != s[right]):
                return False
            left+=1
            right-=1
        return True
```

TIME COMPLEXITY - O(N) and SPACE COMPLEXITY - O(1). Constant space

Question: Given a non-empty string s, you may delete at most one character. Judge whether you can make it a palindrome.

YOU HAVE 15 MINUTES

EXAMPLE:

INPUT: “aba”, “abca”

OUTPUT: True, True (just remove c)

Link: https://leetcode.com/problems/valid-palindrome-ii/

What property or rule must hold for the characters of a string, for the string to be capable of being a palindrome ?

What property or rule must hold for the characters of a string, for the string to be capable of being a palindrome ?

IF THERE IS ONLY ONE CHARACTER IN THE STRING THAT OCCURS ODD NUMBER OF TIMES, THEN THE CHARACTERS MAYBE REARRANGED TO CREATE A PALINDROME

EXAMPLE : AAB, AABBBBCCC, LLMMNN -> CAN BE REARRANGED TO CREATE PALIN

ABC, LLMMOI -> CANNOT GIVE A PALINDROME

DOES THIS FACT HELP WITH valid-palindrome-ii question on LEETCODE ?

What property or rule must hold for the characters of a string, for the string to be capable of being a palindrome ?

IF THERE IS ONLY ONE CHARACTER IN THE STRING THAT OCCURS ODD NUMBER OF TIMES, THEN THE CHARACTERS MAYBE REARRANGED TO CREATE A PALINDROME

EXAMPLE : AAB, AABBBBCCC, LLMMNN -> CAN BE REARRANGED TO CREATE PALIN

ABC, LLMMOI -> CANNOT GIVE A PALINDROME

DOES THIS FACT HELP WITH valid-palindrome-ii question on LEETCODE ? NO!

What property or rule must hold for the characters of a string, for the string to be capable of being a palindrome ?

IF THERE IS ONLY ONE CHARACTER IN THE STRING THAT OCCURS ODD NUMBER OF TIMES, THEN THE CHARACTERS MAYBE REARRANGED TO CREATE A PALINDROME

EXAMPLE : AAB, AABBBBCCC, LLMMNN -> CAN BE REARRANGED TO CREATE PALIN

ABC, LLMMOI -> CANNOT GIVE A PALINDROME

DOES THIS FACT HELP WITH valid-palindrome-ii question on LEETCODE ? NO!

THE ARRANGEMENT ALSO MATTERS HERE!

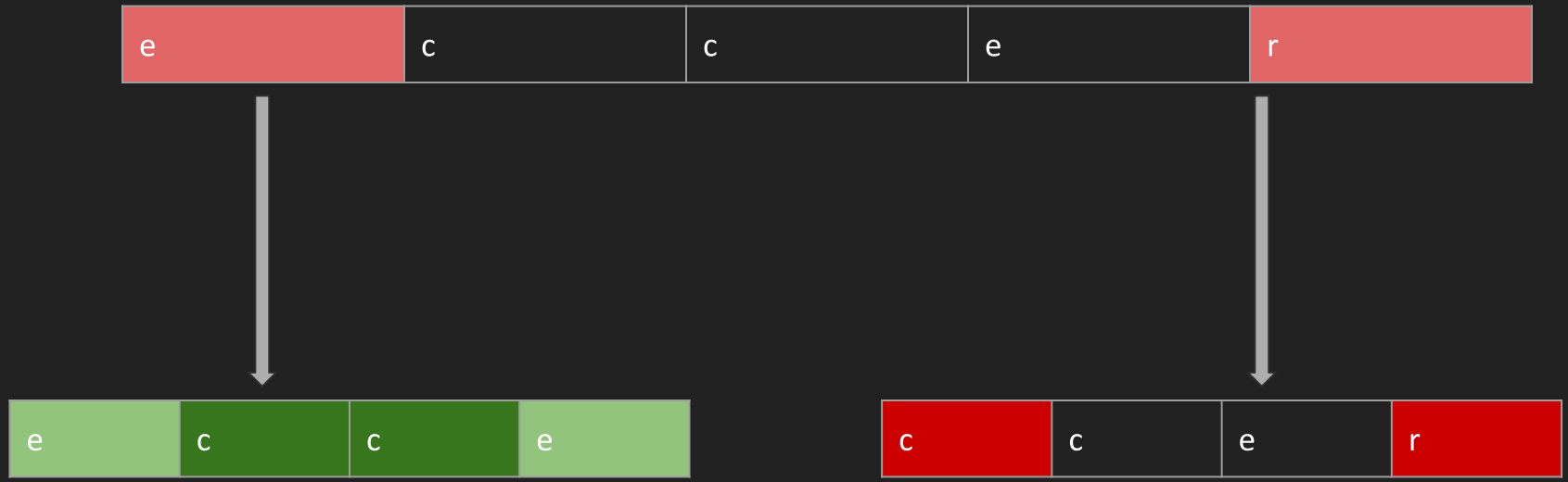
GREEDY BASED SOLUTION

ITERATE TO CHECK IF STRING IS PALINDROME, TILL CHARACTER MISMATCH

CHECK IF STR[left, right -1] or STR[left-1,right] are PALINDROME

IF ANY OF THE TWO ARE PALINDROME -> WORKS

IN SINGLE PASS (LINEAR TIME)



WE CREATE TWO SUBSTRING. ONE IGNORES THE LEFT MISMATCH, ONE IGNORES THE RIGHT MISMATCH AND TAKES ALL THE REMAINING UNPROCESSED CHARACTERS. IF EITHER SUBSTRING IS PALINDROME. YOU'RE DONE.

```python
# Valid Palindrome - ii
class Solution:
    def valid(self, s: str) -> bool:
        print("S=",s)
        l=0
        r=len(s)-1
        while l < r:
            if(s[l] != s[r]):
                return False
            l+=1
            r-=1
        return True

    def validPalindrome(self, s: str) -> bool:
        l=0
        r=len(s)-1
        while l < r:
            if(s[l] != s[r]):
                return self.valid(s[l:r]) or self.valid
                    (s[l+1:r+1])
            l+=1
            r-=1
        return True
```

TIME COMPLEXITY:
O(N), where N is the length of the string.

SLIDING WINDOW

(RECAP)

Question: Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.

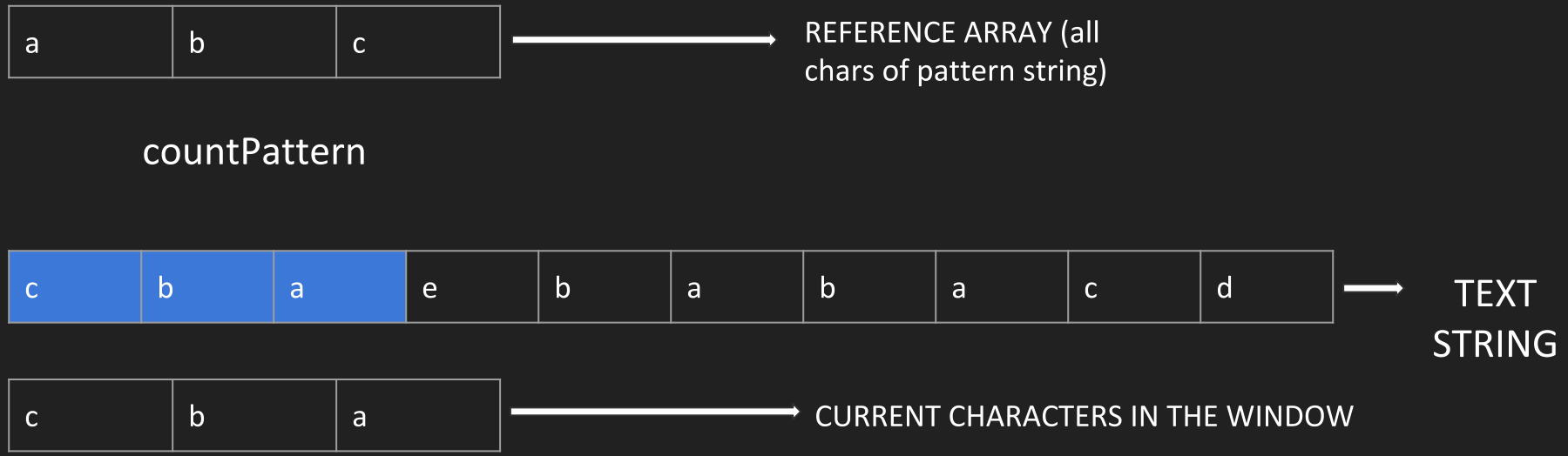
EXAMPLE:

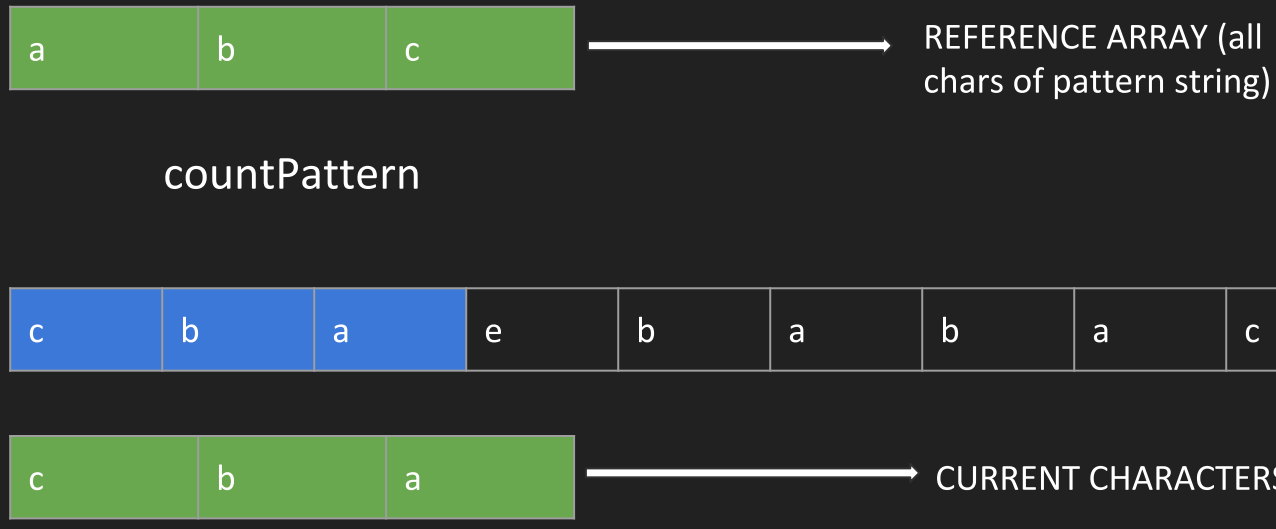
INPUT: s: "cbaebabacd" p: "abc"

OUTPUT: [0, 6]

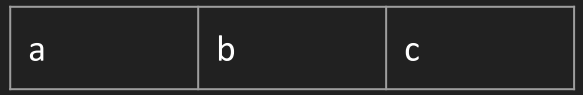
Link: https://leetcode.com/problems/find-all-anagrams-in-a-string/

The substring with start index = 0 is "cba", which is an anagram of "abc". The substring with start index = 6 is "bac", which is an anagram of "abc"


a
b
c
REFERENCE ARRAY (all chars of pattern string)
countPattern
c
b
a
e
b
a
b
a
c
d
TEXT STRING
c
b
a
CURRENT CHARACTERS IN THE WINDOW
countText


a
b
c
REFERENCE ARRAY (all chars of pattern string)
RESULT = [0]
countPattern
c
b
a
e
b
a
b
a
c
d
TEXT STRING
c
b
a
CURRENT CHARACTERS IN THE WINDOW
countText
CURRENT WINDOW AND REFERENCE ARRAY MATCH! CURRENT WINDOW STARTS FROM INDEX 0, so 0 is added to result

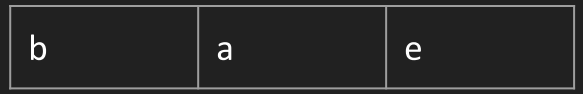
RESULT = [0]



REFERENCE ARRAY (all chars of pattern string)

countPattern

TEXT STRING



CURRENT CHARACTERS IN THE WINDOW

countText

NO MATCH

REFERENCE ARRAY (all chars of pattern string)

RESULT = [0]

countPattern

TEXT STRING

CURRENT CHARACTERS IN THE WINDOW

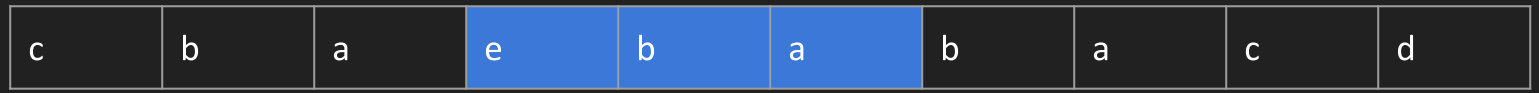
countText

NO MATCH

REFERENCE ARRAY (all chars of pattern string)

RESULT = [0]

countPattern



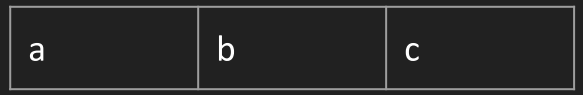
TEXT STRING

CURRENT CHARACTERS IN THE WINDOW

countText

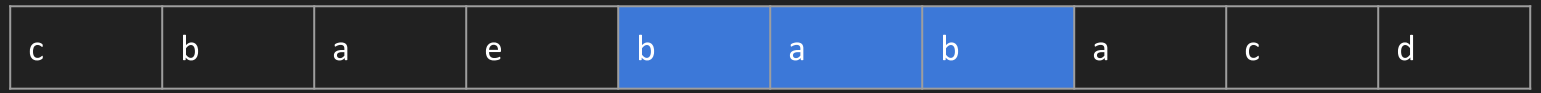
NO MATCH

RESULT = [0]

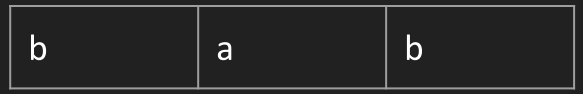


REFERENCE ARRAY (all chars of pattern string)

countPattern



TEXT STRING



CURRENT CHARACTERS IN THE WINDOW

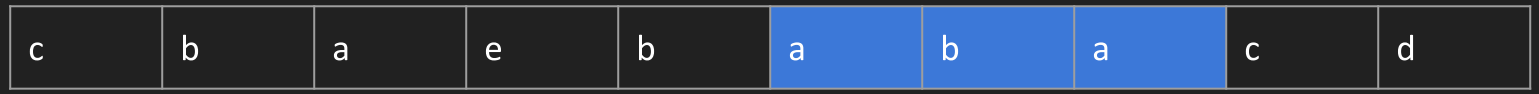
countText

NO MATCH

REFERENCE ARRAY (all chars of pattern string)

RESULT = [0]

countPattern



TEXT STRING

CURRENT CHARACTERS IN THE WINDOW

countText

NO MATCH

REFERENCE ARRAY (all chars of pattern string)

RESULT = [0,6]

countPattern

TEXT STRING

CURRENT CHARACTERS IN THE WINDOW

countText

CURRENT WINDOW AND REFERENCE ARRAY MATCH! CURRENT WINDOW STARTS FROM INDEX 6, so 6 is added to result

REFERENCE ARRAY (all chars of pattern string)

RESULT = [0,6]

countPattern

TEXT STRING

CURRENT CHARACTERS IN THE WINDOW

countText

NO MATCH

SLIDING WINDOW APPROACH

```python
# Sliding window
MAX = 256
class Solution:
    def findAnagrams(self, s: str, p: str) -> List[int]:
        result = {}
        cnt = 0
        patternLen = len(p)
        textLen = len(s)

        if patternLen == 0 or textLen == 0 or patternLen > textLen:
            return result

        countPattern = [0]*MAX
        countText = [0]*MAX

        for i in range(patternLen):
            (countPattern[ord(p[i]) ]) += 1
            (countText[ ord(s[i]) ]) += 1

        for i in range(patternLen, textLen):
            if self.compare(countPattern, countText):
                result[cnt] = i-patternLen
            cnt+=1

            (countText[ord(s[i])]) += 1
            (countText[ ord(s[i-patternLen]) ]) -= 1

        if self.compare(countPattern, countText):
                result[cnt] = textLen - patternLen
        return result
```

TIME COMPLEXITY - O(N+K) SPACE COMPLEXITY - O(K)

Question: Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

YOU HAVE 20 MINUTES

EXAMPLE:

INPUT: S=“ADOBECODEBANC”, T= ”ABC”

OUTPUT: “BANC”

Link: https://leetcode.com/problems/minimum-window-substring/

A	D	O	B	E	C	O	D	E	B	A	N	C

A	D	O	B	E	C	O	D	E	B	A	N	C

Here, we don’t know what the WINDOW SIZE is.
WINDOW SIZE we minimize!


A
B
C
REFERENCE ARRAY


A
D
O
B
E
C
O
D
E
B
A
N
C


A
B
C
REFERENCE ARRAY


A
D
O
B
E
C
O
D
E
B
A
N
C


A
B
C
REFERENCE ARRAY


A
D
O
B
E
C
O
D
E
B
A
N
C


A
B
C
REFERENCE ARRAY


A
D
O
B
E
C
O
D
E
B
A
N
C


A
B
C
REFERENCE ARRAY


A
D
O
B
E
C
O
D
E
B
A
N
C

A	D	O	B	E	C	O	D	E	B	A	N	C

A	D	O	B	E	C	O	D	E	B	A	N	C

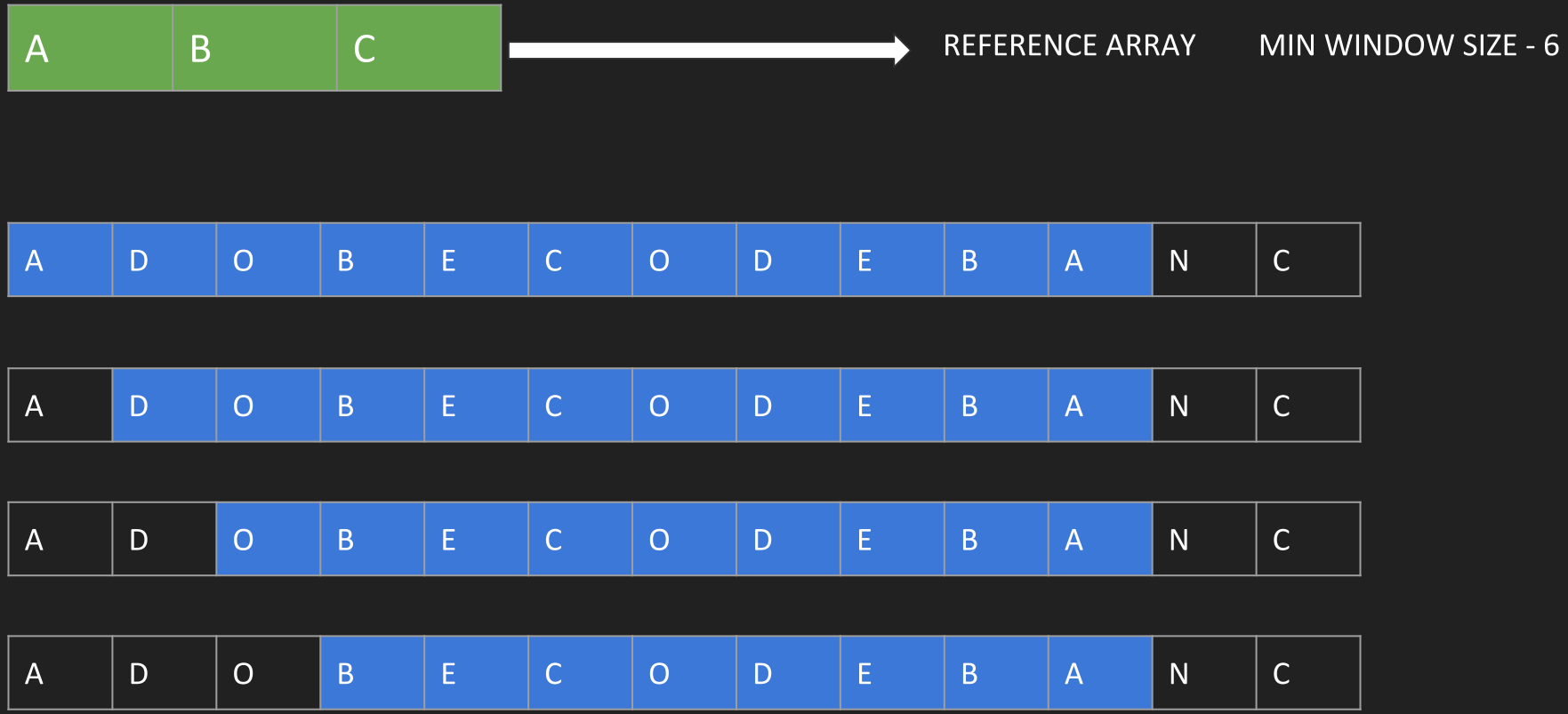
KEEP GROWING THE WINDOW, TO SEE IF YOU CAN FIND ‘A’ AGAIN

A	D	O	B	E	C	O	D	E	B	A	N	C

A	D	O	B	E	C	O	D	E	B	A	N	C

A	D	O	B	E	C	O	D	E	B	A	N	C

A	D	O	B	E	C	O	D	E	B	A	N	C

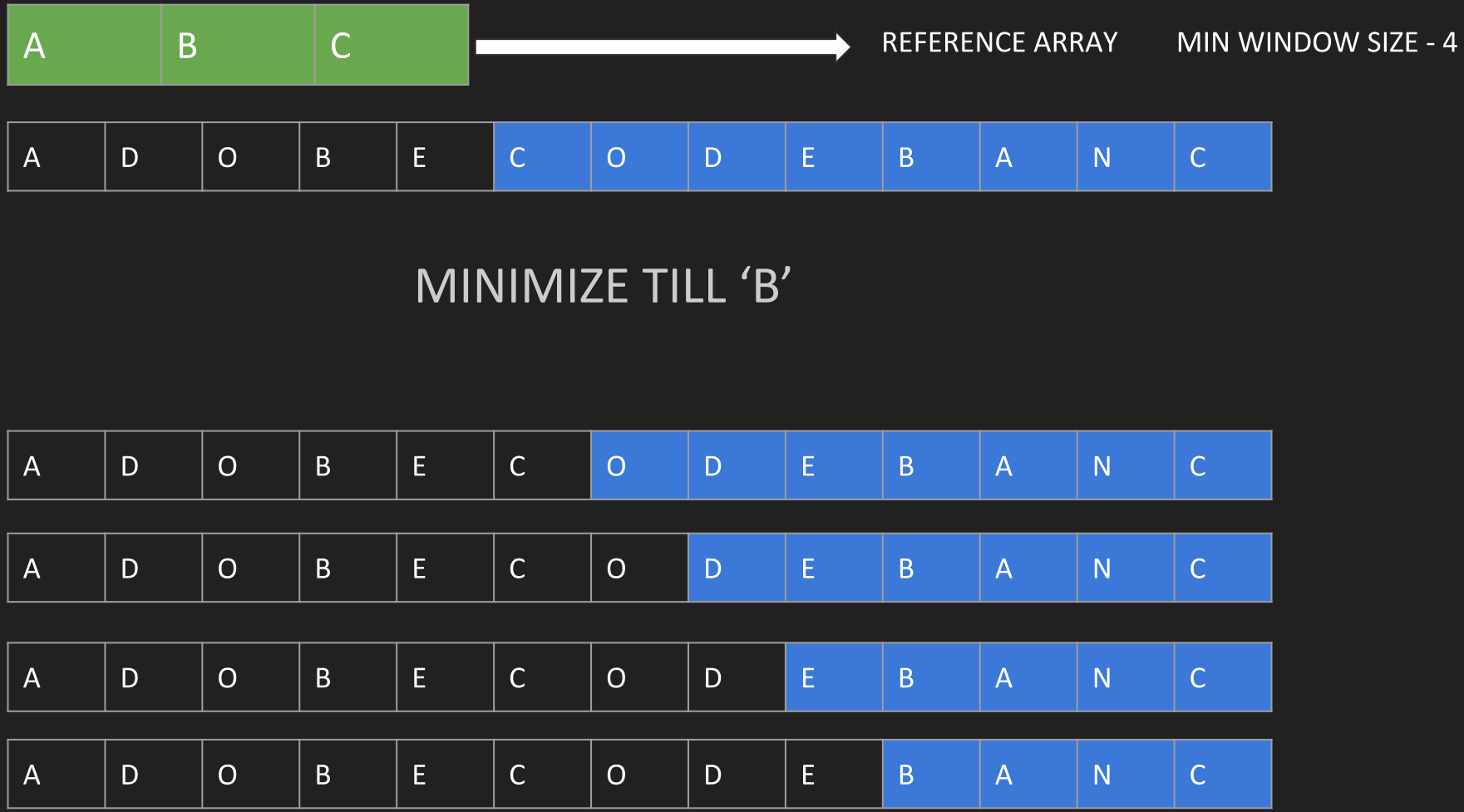


‘A’ FOUND AT LATER INDEX, MINIMIZE THE WINDOW

MINIMIZE TILL C

A	D	O	B	E	C	O	D	E	B	A	N	C

A	D	O	B	E	C	O	D	E	B	A	N	C


A
B
C
REFERENCE ARRAY
MIN WINDOW SIZE - 4
A D O B E C O D E B A N C
MINIMIZE TILL ‘B’
A D O B E C O D E B A N C
A D O B E C O D E B A N C
A D O B E C O D E B A N C
A D O B E C O D E B A N C

```python
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if(len(t) > len(s)):
            return ""

        hash_pat = [0]*256
        hash_str = [0]*256
        for c in t:
            hash_pat[ord(c)-ord('a')]+=1

        start = 0
        start_index = -1;
        min_index = sys.maxsize
        count = 0

        for it in range(0,len(s)):
            c2 = s[it]
            i = ord(c2)-ord('a')
            hash_str[i]+=1
            if(hash_pat[i] != 0 and hash_pat[i] >= hash_str[i]):
                count+=1
            if(count == len(t)):
                j=ord(s[start]) - ord('a')
                while(hash_str[j] == 0 or hash_str[j] > hash_pat[j]):
                    if(hash_str[j] > hash_pat[j]):
                        hash_str[j]-=1
                    start+=1
                    j=ord(s[start]) - ord('a')
                if it - start + 1 < min_index:
                    min_index = it - start + 1;
                    start_index = start

        if(start_index == -1):
            return ""
        return s[start_index:start_index+min_index]
```

TIME COMPLEXITY- O(N+K). where N is the size of ‘S’ and K is the size of ‘T’

Auxiliary Question: Given a string, find the length of the longest substring without repeating characters.

EXAMPLE:

INPUT: s: "abcabcbb"

OUTPUT: 3

https://leetcode.com/problems/longest-substring-without-repeating-characters/

The answer is "abc", with the length of 3.

```python
# Longest substring w/o repeating characters
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        if(len(s) <= 0):
            return 0

        cur_sum = 1
        max_sum = 1
        visited = {}
        visited[s[0]] = 0
        for i in range(1,len(s)):
            if(s[i] not in visited):
                cur_sum+=1
            elif(i - visited[s[i]] > cur_sum):
                cur_sum+=1
            else:
                max_sum = max(max_sum,cur_sum)
                cur_sum = i - visited[s[i]]
            visited[s[i]] = i

        max_sum = max(max_sum,cur_sum)
        return max_sum
```

TIME COMPLEXITY- O(N), where N is the size of the string.