STRINGS:

---

1. **Pattern Matching:**

Using bruteforce: Takes tc: O(nm)

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:

        n= len(haystack)
        m=len(needle)
        if n<m:
            return -1
        i=0
        while(i<n):
            j=0
            k=i
            while(j<m and k<n):
                if haystack[k]==needle[j]:
                    k+=1
                    j+=1
                else:
                    break
            if j==m:
                return i
            i+=1
        return -1
```

Or one more way of bruteforce. Find every substring of length needle and compare with needle that whether it is same or not.

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        n= len(haystack)
        m=len(needle)
        if n<m:
            return -1
        for i in range(n):
            temp=haystack[i:i+m]
            if temp==needle:
                return i
        return -1
```

KMP Algorithm:
TC: O(n+m)
Time complexity to make the lps (longest prefix suffix) is O(2*m)

lps[i] = the longest proper prefix of pat[0..i] which is also a suffix of pat[0..i].

Examples of lps[] construction:

For the pattern “AAAA”, lps[] is [0, 1, 2, 3]

For the pattern “ABCDE”, lps[] is [0, 0, 0, 0, 0]

For the pattern “AABAACAABAA”, lps[] is [0, 1, 0, 1, 2, 0, 1, 2, 3, 4, 5]

For the pattern “AAACAAAAAC”, lps[] is [0, 1, 2, 0, 1, 2, 3, 3, 3, 4]

For the pattern “AAABAAA”, lps[] is [0, 1, 2, 0, 1, 2, 3]

```python
def strStr(self, haystack: str, needle: str) -> int:
    n= len(haystack)
    m=len(needle)
    if n<m:
        return -1
    lps=[0]*m
    prevLPS=0
    i=1
    while(i<m):
        if needle[i]==needle[prevLPS]:
            lps[i]=prevLPS+1
            prevLPS+=1
            i=i+1
        elif prevLPS==0:
            lps[i]=0
            i=i+1
        else:
            prevLPS= lps[prevLPS-1]
    i=0
    j=0
    while(i<n):
        if haystack[i]==needle[j]:
            i+=1
            j+=1
        else:
            if j==0:
                i=i+1
            else:
                j=lps[j-1]
        if j==len(needle):
            return i-j
    return -1
```

Z-Algorithm : Finds in linear time:

```
// str = pat#patronpaties
// z[] = 00003000030000
```

Here we make a z array

How to fill it. First make a temp string = pattern_string

What is Z Array?

For a string str[0..n-1], Z array is of same length as string. An element Z[i] of Z array stores length of the longest substring starting from str[i] which is

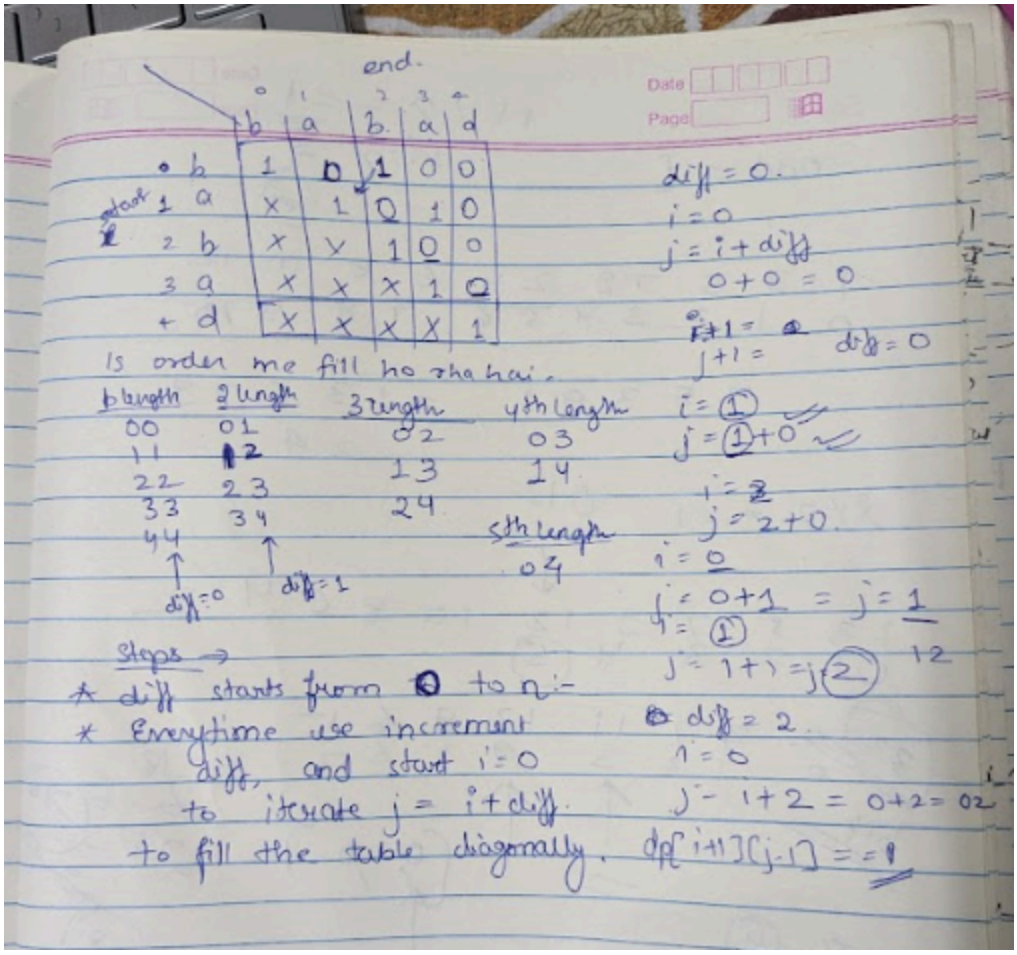
also a prefix of str[0..n-1]. The first entry of Z array is meaning less as complete string is always prefix of itself.

```
Example:
Pattern P = "aab",  Text T = "baabaa"

The concatenated string is = "aab$baabaa"

Z array for above concatenated string is {x, 1, 0, 0, 0,
                                          3, 1, 0, 2, 1}.
Since length of pattern is 3, the value 3 in Z array
indicates presence of pattern.
```

Q: Longest pallindromic substring:



end.

		0 b	1 a	2 b	3 a	4 d
0	b	1	0	1	0	0
start 1	a	x	1	0	1	0
2	b	x	x	1	0	0
3	a	x	x	x	1	0
4	d	x	x	x	x	1

Is order me fill ho raha hai.

b length	2 length	3 length	4th length	5th length
00	01	02	03	04
11	12	13	14	
22	23	24		
33	34			
44				

diff=0 diff=1

Steps →

* diff starts from 0 to n:-
* Everytime use increment diff, and start i=0 to iterate j = i+diff. to fill the table diagonally.

diff = 0.
i = 0
j = i + diff
0+0 = 0
i+1 = 0+1 = 1
j+1 = 0+1 = 1
i = 1
j = 1+0
i = 2
j = 2+0.
i = 0
j = 0+1 = j = 1
i = 1
j = 1+1 = j = 2
diff = 2.
i = 0
j = i+2 = 0+2 = 02
dp[i+1][j-1] = = 1

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        maxi=0
        n=len(s)
        dp=[[0]*n for i in range(n)]
        for diff in range(n):
            i=0
            j=i+diff
            while(j<n):
                if i==j:    #i= 0 and j= i+0   when diff=0
                    dp[i][j]=1
                elif diff==1:
                    if s[i]==s[j]:
                        dp[i][j]=1
                else:
                    if s[i]==s[j] and dp[i+1][j-1]==1:
                        dp[i][j]=1
                if dp[i][j]==1 and j-i+1 > maxi:
                    maxi= j-i+1
                    ans= s[i:j+1]
                i+=1
                j+=1
        return ans
```

Q:Reverse a string using recursion:

```python
def reverseWord(s):
    n=len(s)

    def f(s,i):
        if i==n:
            return ""
        temp= s[i]
        return f(s,i+1)+temp
    return f(s,0)
```

Q: Largest odd number in string:

```python
class Solution:
    def largestOddNumber(self, num: str) -> str:
        for i in range(len(num)-1,-1,-1):
            if int(num[i])%2!=0:
                return num[:i+1]
        return ''
```