482. License Key Formatting
We will cut the first redundant characters and then split all other characters by every K characters. TC is O(n)

```
class Solution:
    def licenseKeyFormatting(self, S: str, K: int) -> str:
        s_without_dash = (''.join(S.split('-'))).upper()
        ret = ''
        f_len = len(s_without_dash) % K
        ret = s_without_dash[:f_len]
        for i in range(f_len, len(s_without_dash), K):
          ret += ('-' if ret else '') + s_without_dash[i:i+K]
        return ret
```

686. Repeated String Match
We will check B in A(multiply times larger than B), and return times.

```
class Solution:
    def repeatedStringMatch(self, A: str, B: str) -> int:
      len_A = len(A)
      len_B = len(B)
      times = len(B) // len(A)
      if times * len_A < len_B:
        times += 1

      if B in A * times:
        return times
      if B in A * (times + 1):
        return times + 1
      return -1
```

844. Backspace String Compare
We will use stack to remove previous letter. TC is O(n)

```
class Solution:
    def backspaceCompare(self, S: str, T: str) -> bool:
        f_s, f_t = '', ''
        stack = []
        for i in S:
          if i == '#':
            if stack:
              stack.pop()
          else:
            stack.append(i)
        f_s = ''.join(stack)
        stack = []
```

```
        for i in T:
          if i == '#':
            if stack:
              stack.pop()
          else:
            stack.append(i)
        f_t = ''.join(stack)
        return f_s == f_t
```

66. Plus One

We will transform it to int and plus one than split it. TC is O(1)

```
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:
        if digits[-1] < 9:
          digits[-1] += 1
          return digits
        else:
          return map(int, list(str(int(''.join(map(str,digits))) + 1)))
```

392. Is Subsequence

We will accumulate i, j by 1 until j approach the end. In the end, we will check whether i is in the end of letter. TC is O(max(m, n))

```
class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        if len(s) == 0:
            return True
        if len(t) == 0:
            return False
        i, j = 0, 0
        while i < len(s) and j < len(t):
            if s[i] == t[j]:
                i += 1
            j += 1
        return True if i == len(s) else False
```