Assignment #3: Stack, DP & Backtracking

Updated 2226 GMT+8 Sep 22, 2025

2025 fall, Complied by 胡孝齐 物理学院

1. 题目

1078: Bigram分词

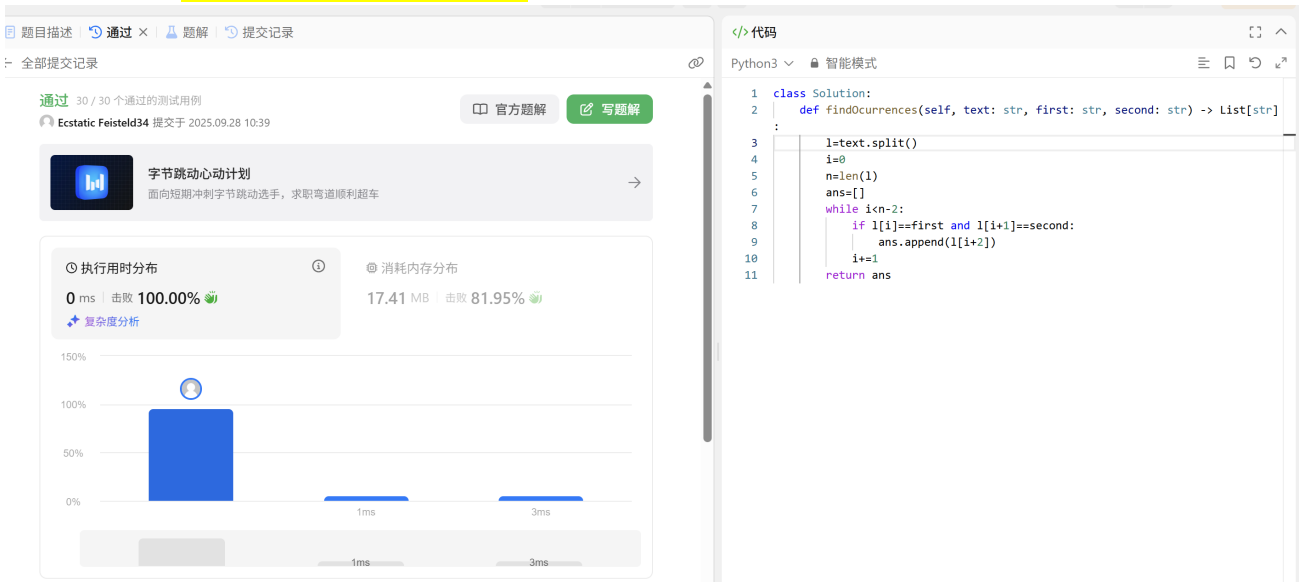
https://leetcode.cn/problems/occurrences-after-bigram/

思路： 将字符串分为列表，然后依次寻找符合第一和第二个词的情况，输出第三个词。

代码：

```
class Solution:
    def findOcurrences(self, text: str, first: str, second: str) -> List[str]:
        l=text.split()
        i=0
        n=len(l)
        ans=[]
        while i<n-2:
            if l[i]==first and l[i+1]==second:
                ans.append(l[i+2])
            i+=1
        return ans
```

代码运行截图 （至少包含有"Accepted"）



283.移动零

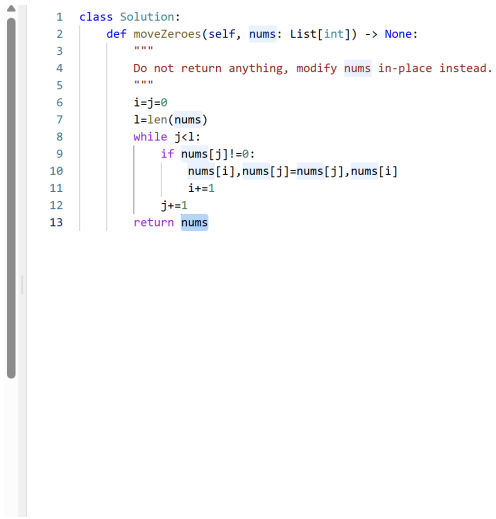
stack, two pinters, https://leetcode.cn/problems/move-zeroes/

思路： 双指针，100热题之一，核心思想是确保左指针左侧均非0，右指针与左指针之间为0，当左指针处为0时，右指针向右移动寻找非零数，然后交换填入左侧。 另外，鉴于只有右指针用于判断，左右指针之间并无影响，故可以使用 nums[i]=nums[j]替代，只要最后将索引(i,j]赋值为0即可，这样其实就和复制列表一样简单明了，只是需要一个i指针指向需要添加数的位置。

代码：

```
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        i=j=0
        l=len(nums)
        while j<l:
            if nums[j]!=0:
                nums[i],nums[j]=nums[j],nums[i]
                i+=1
            j+=1
        return nums
```

代码运行截图 （至少包含有"Accepted"）



20.有效的括号

stack, https://leetcode.cn/problems/valid-parentheses/

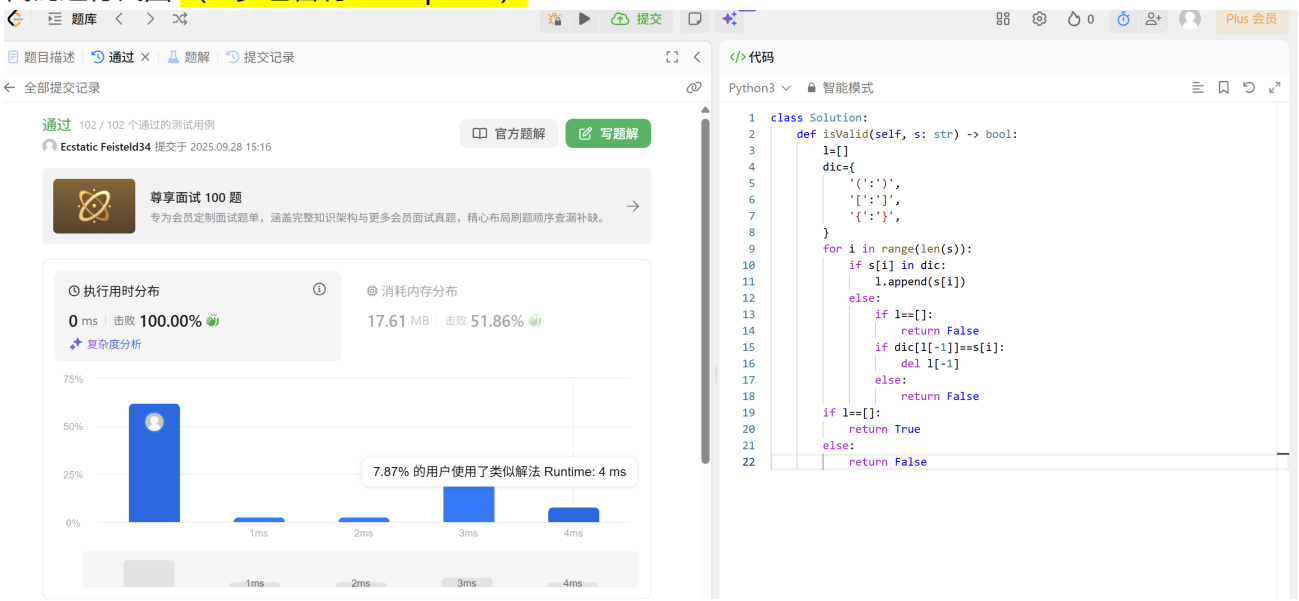
思路： 关于方法原理：将左括号按顺序放入列表中，若满足要求，则后放入的左括号对应的右括号必然先出现，否则不满足要求，同时将与右括号对应后的左括号删去。 本题若想不到这个做法就不好做了，关于如何想到这个方法，括号需要满足顺序，后出现的左括号先闭合，故考虑使用一个栈来储存当前未闭合括号，要求接下来每一个括号不能违反规则。

代码：

```
class Solution:
    def isValid(self, s: str) -> bool:
        l=[]
        dic={
            '(':')',
            '[':']',
            '{':'}',
        }
        for i in range(len(s)):
            if s[i] in dic:
                l.append(s[i])
            else:
                if l==[]:
                    return False
                if dic[l[-1]]==s[i]:
                    del l[-1]
```

```
                else:
                    return False
        if l==[]:
            return True
        else:
            return False
```

代码运行截图 （至少包含有"Accepted"）



118.杨辉三角

dp, https://leetcode.cn/problems/pascals-triangle/

思路： 使用二维列表储存杨辉三角，先将两端赋值为1，然后逐层计算。

代码：

```
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        li=[]
        for i in range(numRows):
            l=[0]*(i+1)
            l[0]=1
            l[i]=1
            for j in range(1,i):
                l[j]=li[i-1][j-1]+li[i-1][j]
            li.append(l)
        return li
```

代码运行截图 （至少包含有"Accepted"）

46.全排列

backtracking, https://leetcode.cn/problems/permutations/

思路： 首先，排列种类的统计方法为从第一个数开始，第一个数可以选n种，第二个数可以选n-1种。。。由于最终的种类数为N！，时间复杂度不低于O（N！），故使用递归。在函数中使用一个变量（first）代表当前所处的位置，每次只和后面的数进行交换可以保证不重复，每次交换完成（即选取完毕当前位置的数字）后进入一个新的分支，继续选取下一个数，然后交换回来，重新选取当前位置的数。以上是官方题解的方法，但使用交换来处理不够直观，

代码 法一：直接对nums操作，想起来比较繁复

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        ans=[]
        def backtrack(first):
            if first==len(nums)-1:
                ans.append(nums.copy())
                ##return
            for i in range(first,len(nums)):
                nums[first],nums[i]=nums[i],nums[first]
                backtrack(first+1)
                nums[first],nums[i]=nums[i],nums[first]
        backtrack(0)
        return ans
```

方法二：创建一个列表记录各个数字是否已被使用，然后每次选择未被使用过的数字填入即可，想起来很直观但千万不要忘记在当前位置重新选择其它数之前要把这个数重新标为未选。

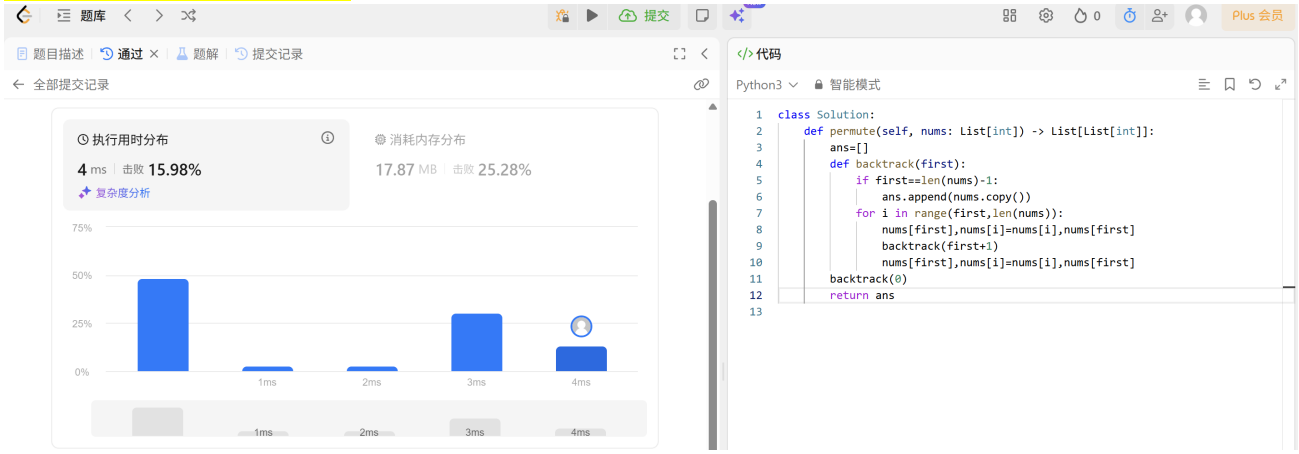
```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        n=len(nums)
        ans=[]
        path=[False]*n
        def backtrack(i,l):
            if i==n:
                ans.append(l)
                ##return
            for j in range(n):
                if not path[j]:
                    path[j]=True
```

```
                    backtrack(i+1,l+[nums[j]])
                    path[j]=False
        backtrack(0,[])
        return ans
```

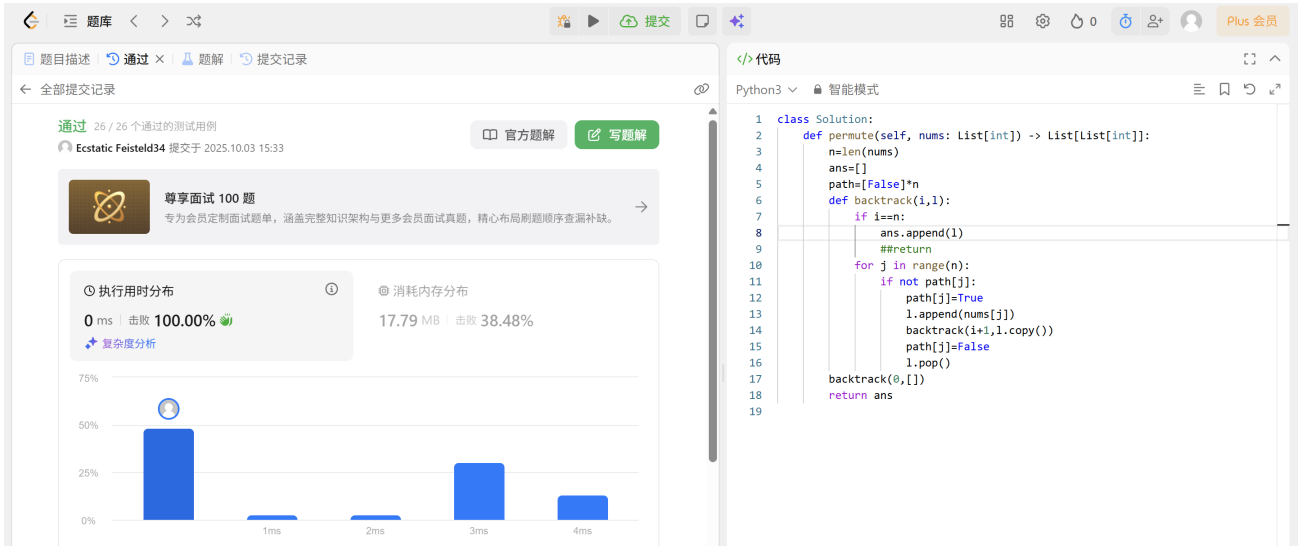
方法三:方法二中在进入下一步时使用l+[nums[j]]作为函数变量，实际上这是对列表进行了一次复制，因此最后ans.append(l)不需要copy，如果愿意的话，把copy放在backtrack(i+1,l.copy())，就可以省去ans.append(l)的copy

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        n=len(nums)
        ans=[]
        path=[False]*n
        def backtrack(i,l):
            if i==n:
                ans.append(l.copy())
                ##return
            for j in range(n):
                if not path[j]:
                    path[j]=True
                    l.append(nums[j])
                    backtrack(i+1,l)
                    path[j]=False
                    l.pop()
        backtrack(0,[])
        return ans
```

（至少包含有"Accepted"）



为什么这个跑出来是0ms，方法三跑出来是4ms，无论如何方法三的代码不会比这个慢吧?



78.子集

backtracking, https://leetcode.cn/problems/subsets/

思路： 每个数可以选或者不选，复制列表来传递可以避免回溯

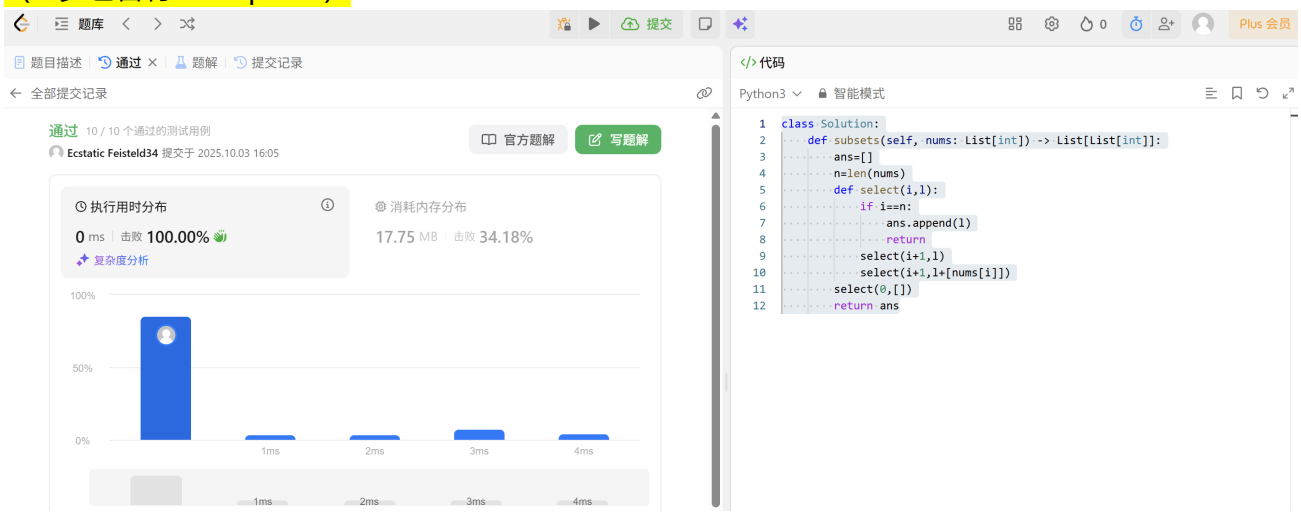
代码 方法一

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        ans=[]
        n=len(nums)
        l=[]
        def select(i):
            if i==n:
                ans.append(l.copy())
                return
            for j in range(2):
                if j==0:
                    l.append(nums[i])
                    select(i+1)
                    l.pop()
                else:
                    select(i+1)
        select(0)
        return ans
```

方法二：一开始没想到什么写，导致方法一写得很蠢。。。

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        ans=[]
        n=len(nums)
        def select(i,l):
            if i==n:
                ans.append(l)
                return
            select(i+1,l)
            select(i+1,l+[nums[i]])
        select(0,[])
        return ans
```

（至少包含有"Accepted"）



2. 学习总结和个人收获

学习了双指针，栈，以及回溯算法（深度优先搜索），并厘清了关于列表的特性，在需要固定当前列表状态时需要复制，避免随列表的改变而改变。对于何种情况需要使用什么算法有了大概的认知，譬如复杂度

为指数函数和N!时使用递归，其余可以直接遍历，在注重顺序（后进先出）时使用栈。