

Assignment #3: Stack, DP & Backtracking

Updated 2226 GMT+8 Sep 22, 2025

2025 fall, Complied by 林奕妃、环境科学与工程学院

说明:

1. 解题与记录:

对于每一个题目, 请提供其解题思路 (可选), 并附上使用Python或C++编写的源代码 (确保已在OpenJudge, Codeforces, LeetCode等平台上获得Accepted)。请将这些信息连同显示“Accepted”的截图一起填写到下方的作业模板中。 (推荐使用Typora <https://typora.io> 进行编辑, 当然你也可以选择Word。) 无论题目是否已通过, 请标明每个题目大致花费的时间。

2. 提交安排: 提交时, 请首先上传PDF格式的文件, 并将.md或.doc格式的文件作为附件上传至右侧的“作业评论”区。确保你的Canvas账户有一个清晰可见的本人头像, 提交的文件为PDF格式, 并且“作业评论”区包含上传的.md或.doc附件。
3. 延迟提交: 如果你预计无法在截止日期前提交作业, 请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。

请按照上述指导认真准备和提交作业, 以保证顺利完成课程要求。

1. 题目

1078: Bigram分词 (15分钟)

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

思路: 首先按照空格将输入的字符串分割成单词列表, 随后遍历查找符合要求的第三个词并将其记录下来即可。

代码:

```
from typing import List

class Solution:
    def findOccurrences(self, text: str, first: str, second: str) -> List[str]:
        """
        找出文本中跟在 "first second" 模式后的所有第三个词 "third".
        """
        # 1. 将文本按空格分割成单词列表
        words = text.split(' ')
        # 用于存储结果的列表
        result = []
        # 获取单词列表的长度
        length = len(words)
        if length < 3:
            return result
        for i in range(length - 2):
            if words[i] == first and words[i + 1] == second:
                result.append(words[i + 2])
        return result
```

```

n = len(words)

# 2. 遍历单词列表，寻找 "first second third" 模式
# 循环范围必须确保 words[i+2] 不会越界
for i in range(n - 2):
    # 检查当前窗口是否匹配 "first second"
    if words[i] == first and words[i+1] == second:
        # 如果匹配，将第三个词添加到结果列表中
        result.append(words[i+2])

# 3. 返回收集到的所有 "third" 词
return result

```

代码运行截图 (至少包含有"Accepted")

通过 30 / 30 个通过的测试用例

 Yifei Lin 提交于 2025.10.08 14:39

 官方题解

 写题解



面向在校学生的专享特惠

完成认证享 7 折 Plus 会员，享受更多学业及职业成长帮助



① 执行用时分布

ⓘ

0 ms | 击败 100.00% 🎉

复杂度分析

② 消耗内存分布

17.58 MB | 击败 36.93%

283. 移动零 (15分钟)

stack, two pointers, <https://leetcode.cn/problems/move-zeroes/>

思路：采用双指针，将整个数组遍历一遍，把非零数字排列完成后剩余位置填充上0即可。

代码：

```

from typing import List

class Solution:
    def moveZeroes(self, nums: List[int]) -> None:

        # slow 指针标记下一个非零元素应该放置的位置。
        slow = 0

        # fast 指针遍历整个数组，寻找非零元素。
        for fast in range(len(nums)):
            if nums[fast] != 0:
                nums[slow], nums[fast] = nums[fast], nums[slow]
                slow += 1

```

代码运行截图 (至少包含有"Accepted")

通过 75 / 75 个通过的测试用例

 Yifei Lin 提交于 2025.10.08 14:47

 官方题解

 写题解



面向在校学生的专享特惠

完成认证享 7 折 Plus 会员，享受更多学业及职业成长帮助



① 执行用时分布

7 ms | 击败 46.95%

 复杂度分析

②

消耗内存分布

18.65 MB | 击败 34.94%

20.有效的括号 (20分钟)

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

思路：如果字符串长度为奇数，不可能有效。初始化空栈并创建映射关系，遍历输入字符实现左右括号的对应。

代码：

```
class Solution:
    def isValid(self, s: str) -> bool:
        # 如果字符串长度为奇数，不可能有效
        if len(s) % 2 != 0:
            return False

        stack = []

        # 创建一个哈希表来存储括号的映射关系
        # key 是右括号，value 是对应的左括号
        mapping = {')': '(', ']': '[', '}': '{'}

        # 遍历字符串中的每一个字符
        for char in s:
            # 如果是右括号
            if char in mapping:
                # 检查栈是否为空，如果为空则说明没有对应的左括号
                # 或者栈顶的左括号与当前右括号不匹配
                # pop() 会移除并返回列表的最后一个元素
                # 如果栈为空，stack.pop() 会报错，所以先判断 stack 是否为空
                top_element = stack.pop() if stack else '#'

                if mapping[char] != top_element:
                    return False
            else:
                stack.append(char)
```

```
# 如果是左括号，就压入栈中
stack.append(char)

# 遍历结束后，如果栈为空，说明所有括号都正确匹配了
# 如果栈不为空，说明有未闭合的左括号
return not stack
```

代码运行截图 (至少包含有"Accepted")

通过 102 / 102 个通过的测试用例

 Yifei Lin 提交于 2025.10.08 14:59

 官方题解

 写题解



尊享面试 100 题

专为会员定制面试题单，涵盖完整知识架构与更多会员面试真题，精心布局刷题顺序查...



① 执行用时分布

ⓘ

0 ms | 击败 100.00% 🏆

⭐ 复杂度分析

ⓘ 消耗内存分布

17.70 MB | 击败 36.69%

118. 杨辉三角

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

思路：初始化设置第一行为1后，从第一行开始逐行向下生成，同时每一行第一个与最后一个数字均为1。

代码：

```
from typing import List

class Solution:
    def generate(self, numRows: int) -> List[List[int]]:

        # 初始化结果列表，并放入第一行
        triangle = [[1]]

        # 从第二行开始，逐行生成
        # 我们需要生成 numRows-1 个新行
        for i in range(1, numRows):
            # 获取上一行
            previous_row = triangle[i-1]

            # 新的一行总是以 1 开始
            current_row = [1]

            # 计算中间的元素
            for j in range(1, len(previous_row)):
                current_row.append(previous_row[j] + previous_row[j-1])
```

```

# 遍历上一行的相邻元素对
for j in range(len(previous_row) - 1):
    # 新元素是上一行相邻两个元素的和
    new_element = previous_row[j] + previous_row[j+1]
    current_row.append(new_element)

    # 新的一行总是以 1 结束
    current_row.append(1)

    # 将生成的新行添加到杨辉三角中
    triangle.append(current_row)

return triangle

```

代码运行截图 (至少包含有"Accepted")

通过 30 / 30 个通过的测试用例

 Yifei Lin 提交于 2025.10.08 15:09

 官方题解

 写题解

① 执行用时分布

ⓘ

 消耗内存分布

0 ms | 击败 100.00% 🎉

17.48 MB | 击败 56.05% 🎉

 复杂度分析

100%

46.全排列 (15分钟)

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

思路：使用回溯算法罗列出所有可能性。思考了一下如果丢弃题干里数字不同的条件，在题解中看到了剪枝思想，先对数字进行排列后对相同的数字归类，在排列的时候可以跳过相同的数字。

代码

```

from typing import List

class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        # 用于存放所有排列结果的列表
        result = []
        # 用于存放当前正在构建的排列
        path = []
        # 用于记录 nums 中的数字是否已被使用
        # 长度为 len(nums)，初始值都为 False
        used = [False] * len(nums)

        def backtrack():
            # 1. 递归的终止条件
            # 当 path 的长度等于 nums 的长度时，说明找到了一个完整的排列

```

```

if len(path) == len(nums):
    # 添加 path 的一个副本到 result
    result.append(path[:])
    return

# 2. 遍历所有可能的选择
for i in range(len(nums)):
    # 如果当前数字已经被使用过，则跳过
    if used[i]:
        continue

    # 3. 做出选择
    path.append(nums[i])
    used[i] = True

    # 4. 进入下一层决策
    backtrack()

    # 5. 撤销选择（回溯）
    path.pop()
    used[i] = False

# 初始调用
backtrack()
return result

```

(至少包含有"Accepted")

通过 26 / 26 个通过的测试用例



Yifei Lin 提交于 2025.10.08 15:21

官方题解

写题解

① 执行用时分布

ⓘ

0 ms | 击败 100.00% 🏆

复杂度分析

② 消耗内存分布

17.61 MB | 击败 80.12% 🏆

75%

78. 子集 (20分钟)

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

思路：遍历，由最初的空集到生成含一个元素的子集，再遍历由子集生成新的子集。

代码

```

from typing import List

class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:

```

```

# 用于存放所有子集结果的列表
result = []
# 用于存放当前正在构建的子集
path = []

def backtrack(start_index: int):
    # 1. 将当前路径的副本添加到结果中
    # 每一个递归节点都代表一个有效的子集
    result.append(path[:])

    # 2. 从 start_index 开始，遍历所有可能的选择
    for i in range(start_index, len(nums)):
        # 3. 做出选择
        path.append(nums[i])

        # 4. 进入下一层决策
        # 传入 i + 1，确保下次选择从下一个元素开始
        backtrack(i + 1)

        # 5. 撤销选择（回溯）
        path.pop()

    # 初始调用，从索引 0 开始
backtrack(0)
return result

```

(至少包含有"Accepted")

← 全部提交记录

通过 10 / 10 个通过的测试用例

 Yifei Lin 提交于 2025.10.08 16:04

 官方题解

 写题解

① 执行用时分布

ⓘ

0 ms | 击败 100.00% 🎉

↗ 复杂度分析

ⓘ 消耗内存分布

17.56 MB | 击败 69.08% 🎉

100%

2. 学习总结和个人收获

本次作业与之前计概课作业有一些贯通之处，写起来会更轻松些。在完成作业后还可以在官方解题中学一下其他解法，觉得自己的代码简洁性还能提高，leetcode里面有一些题解写的很好，学到很多。目前我主要还是要提高代码熟练度，多进行练习抓紧复健。栈的概念与算法运用部分我不是很熟练，需要进一步巩固学习。