Assignment #9: Mock Exam立冬

Updated 1856 GMT+8 Nov 7, 2025

2025 fall, Complied by 林奕妃、环境科学与工程学院

> **说明：**
>
> 1. Nov月考： AC0 。考试题目都在"题库（包括计概、数算题目）"里面，按照数字题号能找到，可以重新提交。作业中提交自己最满意版本的代码和截图。
> 2. 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://typoraio.cn 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。
> 3. 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。
> 4. 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

M02255: 重建二叉树（40分钟）

http://cs101.openjudge.cn/practice/02255/

思路：如果只有一个遍历，则存在歧义。前序遍历能提供根节点信息，但无法确定左右子树。它的第一个元素永远是当前树或子树的根。中序遍历提供结构划分信息。一旦知道了根节点，中序遍历中根节点左边的所有元素都属于左子树，右边的所有元素都属于右子树。根据前序遍历和中序遍历，递归地构建后序遍历结果。同时注意到这个题目如果提供的是前序和后序的组合通常是不行的。因为它们都只提供根节点信息，而缺少了划分左右子树的关键信息。

代码

```
import sys

def get_postorder(preorder: str, inorder: str) -> str:
    # 递归的基准条件：如果遍历序列为空，说明是空树，返回空字符串。
    if not preorder:
        return ""

    # 1. 确定根节点：前序遍历的第一个字符就是根节点。
    root_val = preorder[0]
```

```
    # 2. 在中序遍历中找到根节点，划分左右子树。
    try:
        root_index_in_inorder = inorder.index(root_val)
    except ValueError:
        # 正常情况下不会发生，因为题目保证输入合法
        return ""

    # 划分出左子树和右子树的中序遍历
    inorder_left = inorder[:root_index_in_inorder]
    inorder_right = inorder[root_index_in_inorder + 1:]

    # 3. 根据左子树的节点数量，在前序遍历中划分左右子树。
    left_subtree_size = len(inorder_left)

    # 划分出左子树和右子树的前序遍历
    # preoder[0] 是根，preorder[1:1+left_subtree_size] 是左子树，剩下的是右子树
    preorder_left = preorder[1 : 1 + left_subtree_size]
    preorder_right = preorder[1 + left_subtree_size:]

    # 4. 递归地获取左、右子树的后序遍历
    postorder_left = get_postorder(preorder_left, inorder_left)
    postorder_right = get_postorder(preorder_right, inorder_right)

    # 5. 合并结果：后序遍历 = 左子树后序 + 右子树后序 + 根
    return postorder_left + postorder_right + root_val

def main():
    """
    主函数，处理多组输入直到EOF。
    """
    # sys.stdin 是一个可迭代对象，可以逐行读取输入
    for line in sys.stdin:
        # 去除行首尾的空白字符
        line = line.strip()
        if not line:
            continue

        try:
            # 按空格分割得到前序和中序遍历字符串
            preorder_str, inorder_str = line.split()
            # 调用函数计算后序遍历结果并打印
            postorder_result = get_postorder(preorder_str, inorder_str)
            print(postorder_result)
        except (ValueError, IndexError):
            # 捕获可能的分割错误，例如行中只有一个词
            continue

if __name__ == "__main__":
    main()
```

代码运行截图 （至少包含有"Accepted"）

#50775994提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def get_postorder(preorder: str, inorder: str) -> str:
    # 递归的基准条件：如果遍历序列为空，说明是空树，返回空字符串。
    if not preorder:
        return ""
```

基本信息

#: 50775994
题目: 02255
提交人: 25n2200013554
内存: 3592kB
时间: 21ms
语言: Python3
提交时间: 2025-11-10 00:46:43

M02774: 木材加工（20分钟）

http://cs101.openjudge.cn/practice/02774/

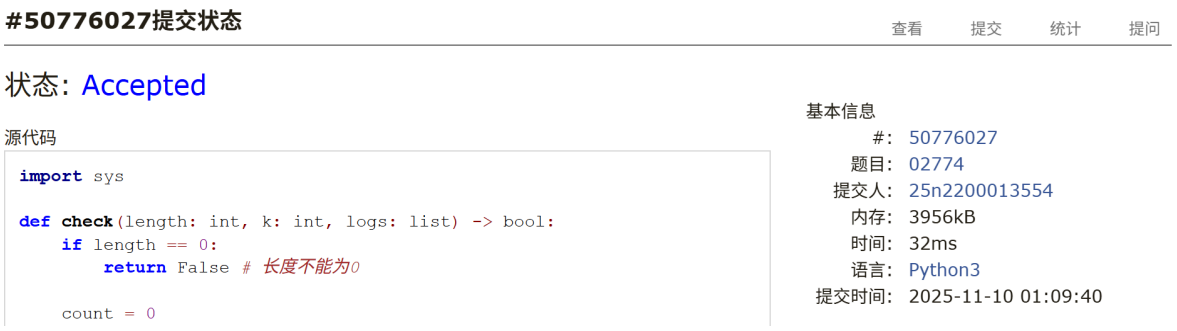
思路：这个题目和之前作业1中的袋子里最少数目的球以及月度开销问题有相似之处，可以使用二分查找利用单调性找到满足要求的最大长度。下界为1上界为最长的那根原木。用一个变量来记录我们找到的满足条件的最大长度，初始值为0。

代码

```
import sys

def check(length: int, k: int, logs: list) -> bool:
    if length == 0:
        return False # 长度不能为0

    count = 0
    for log_len in logs:
        count += log_len // length
        # 提前剪枝，如果已经满足条件，可以提前退出以优化
        if count >= k:
            return True

    return count >= k

def solve():
    try:
        # 读取第一行 N 和 K
        line1 = sys.stdin.readline().strip()
        if not line1:
            return
        n, k = map(int, line1.split())

        # 读取 N 根原木的长度
        logs = []
        for _ in range(n):
            logs.append(int(sys.stdin.readline().strip()))

        # 二分查找的范围
        # left 的最小可能答案是 1
        # right 的最大可能答案不会超过最长的木头长度
        left = 1
        # right 的上界可以设为 10001，或者 logs 中的最大值
```

```
        right = max(logs) if logs else 0
        ans = 0

        while left <= right:
            mid = left + (right - left) // 2

            # mid 可能是0，如果 right 初始为0，而 left 为1，mid会是0
            # 我们的 check 函数处理了 length=0 的情况，但逻辑上 mid 应该从1开始
            if mid == 0:
                # 避免 mid 为 0 导致死循环或除零错误
                # 实际上因为 left 从 1 开始，一般不会出现 mid=0
                left = 1
                continue

            if check(mid, k, logs):
                # 如果 mid 长度可行，我们尝试寻找更大的答案，所以向右搜索
                ans = mid
                left = mid + 1
            else:
                # 如果 mid 长度不可行，我们必须减小长度，所以向左搜索
                right = mid - 1

        print(ans)

    except (IOError, ValueError):
        return

if __name__ == "__main__":
    solve()
```

代码运行截图 （至少包含有"Accepted"）



#50776027提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def check(length: int, k: int, logs: list) -> bool:
    if length == 0:
        return False # 长度不能为0

    count = 0
```

基本信息

#: 50776027
题目: 02774
提交人: 25n2200013554
内存: 3956kB
时间: 32ms
语言: Python3
提交时间: 2025-11-10 01:09:40

M02788: 二叉树（2） （30分钟）

http://cs101.openjudge.cn/practice/02788/

思路：题目为完全二叉树，其满足对于任意一个节点 p，它的左孩子是 2p，右孩子是 2p + 1。因此可以用可以用一个循环来模拟这个逐层计算，直至达到题目所给的结点范围。

代码

```python
import sys

def solve():
    """
    处理多组输入，直到遇到 '0 0'
    """
    # sys.stdin 是一个可迭代对象，可以逐行读取
    for line in sys.stdin:
        try:
            m, n = map(int, line.strip().split())
        except (ValueError, IndexError):
            # 忽略空行或格式不正确的行
            continue

        # 输入结束标志
        if m == 0 and n == 0:
            break

        # 如果 m 本身就大于 n，那么它的子树中没有任何节点 <= n
        if m > n:
            print(0)
            continue

        count = 0
        # left 和 right 代表以 m 为根的子树中，当前层的节点编号范围
        left, right = m, m

        # 只要当前层的最左边节点编号不大于 n，就说明这一层可能存在有效节点
        while left <= n:
            # 当前层的有效节点数是 [left, min(right, n)]
            # 节点个数 = min(right, n) - left + 1
            count += min(right, n) - left + 1

            # 更新到下一层
            # 下一层的最左节点是当前 left 的左孩子
            left = left * 2
            # 下一层的最右节点是当前 right 的右孩子
            right = right * 2 + 1

        print(count)

if __name__ == "__main__":
    solve()
```

代码运行截图 （至少包含有"Accepted"）

#50778356提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 50778356
题目: 02788
提交人: 25n2200013554
内存: 3892kB
时间: 30ms
语言: Python3
提交时间: 2025-11-10 13:22:12

源代码

```
import sys

def solve():
    """
    处理多组输入，直到遇到 '0 0'
    """
    # sys.stdin 是一个可迭代对象，可以逐行读取
```

M04081: 树的转换 （1小时）

http://cs101.openjudge.cn/practice/04081/

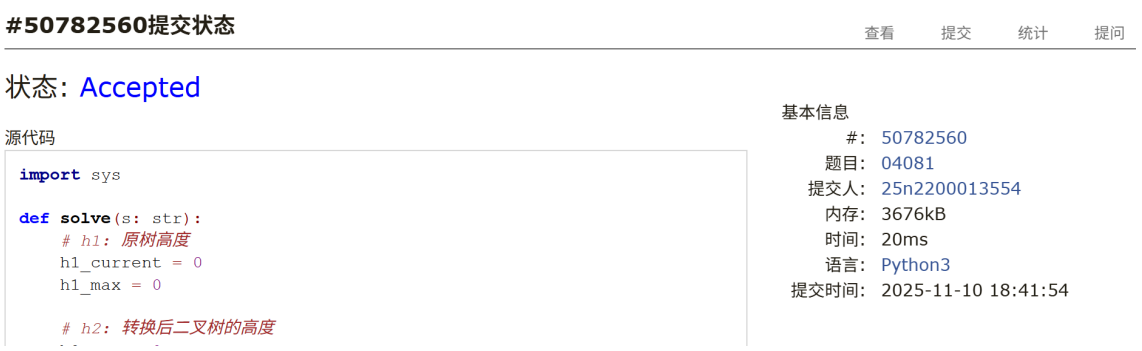
思路：'d' (down) 表示从父节点移动到一个子节点，树的当前深度加一。'u' (up) 表示从子节点返回到父节点，树的当前深度减一。因此，原树的高度 h1 就是在整个遍历过程中，当前深度所能达到的最大值。转换规则为一个节点的第一个孩子成为它在二叉树中的左孩子，一个节点的下一个兄弟成为它在二叉树中的右孩子。无论是在原树中向下走一步（到第一个孩子），还是横向走一步（到下一个兄弟），在转换后的二叉树中，深度都会加一。因此遇到d的时候如果是第一个孩子则为其父亲的二叉树深度+1，如果有记录则为其兄弟的二叉树深度+1，遇到u则从子树返回。

代码

```
import sys

def solve(s: str):
    # h1: 原树高度
    h1_current = 0
    h1_max = 0

    # h2: 转换后二叉树的高度
    h2_max = 0
    # 栈，存储当前路径上每个节点的二叉树深度
    h2_path_stack = [0]
    # 字典，key为原树深度(父亲的深度)，value为该父亲的最后一个孩子的二叉树深度
    last_sibling_h2 = {}

    for char in s:
        if char == 'd':
            # --- 计算 h1 ---
            h1_current += 1
            h1_max = max(h1_max, h1_current)

            # --- 计算 h2 ---
            parent_h1_depth = h1_current - 1

            # 检查父节点是否已经有访问过的孩子
            if parent_h1_depth not in last_sibling_h2:
                # 这是第一个孩子
                # 它的二叉树深度 = 父亲的二叉树深度 + 1
                parent_h2_depth = h2_path_stack[-1]
                current_h2 = parent_h2_depth + 1
```

```
            else:
                # 这是兄弟节点
                # 它的二叉树深度 = 前一个兄弟的二叉树深度 + 1
                prev_sibling_h2_depth = last_sibling_h2[parent_h1_depth]
                current_h2 = prev_sibling_h2_depth + 1

            # 更新最大二叉树高度
            h2_max = max(h2_max, current_h2)

            # 记录当前节点作为其父节点的“最后一个孩子”
            last_sibling_h2[parent_h1_depth] = current_h2

            # 将当前节点的二叉树深度压入路径栈
            h2_path_stack.append(current_h2)

        elif char == 'u':
            # --- h1 回溯 ---
            # 清除我们刚刚离开的那一层(子节点层)的“最后兄弟”记录
            # 因为下一次访问这一层时，将是从一个新的父节点开始
            if h1_current in last_sibling_h2:
                del last_sibling_h2[h1_current]

            h1_current -= 1

            # --- h2 回溯 ---
            h2_path_stack.pop()

    print(f"{h1_max} => {h2_max}")


def main():
    for line in sys.stdin:
        s = line.strip()
        if s:
            solve(s)

if __name__ == "__main__":
    main()
```

代码运行截图 （至少包含有"Accepted"）



#50782560提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def solve(s: str):
    # h1: 原树高度
    h1_current = 0
    h1_max = 0

    # h2: 转换后二叉树的高度
```

基本信息

#: 50782560
题目: 04081
提交人: 25n2200013554
内存: 3676kB
时间: 20ms
语言: Python3
提交时间: 2025-11-10 18:41:54

M04117: 简单的整数划分问题（30分钟）

dfs, dp, http://cs101.openjudge.cn/practice/04117/

思路：dp[i] [j] 为：将整数 i 划分为若干个正整数之和，其中每个划分出的数都不超过 j 的方案数。讨论两种情况，如果划分中不包含整数 j，那么所有的划分部分都必须小于 j（即最大不超过 j-1）。这种情况下，方案数就等于将整数 i 划分为最大数不超过 j-1 的方案数，即 dp[i] [j-1]。如果划分中至少包含一个 j，我们可以先从 i 中拿出一个 j，剩下需要划分的整数就是 i-j。对于这剩下的 i-j，我们依然要将其划分为若干数之和，并且为了满足 n1>=n2>=... 的约束，剩下的这些数每一个都不能超过 j。所以，这种情况下的方案数就等于将 i-j 划分为最大数不超过 j 的方案数，即 dp[i-j] [j]。因此dp[i] [j] = dp[i] [j-1] + dp[i-j] [j]。

代码

```
import sys
def solve():
    """
    处理多组输入直到EOF
    """
    for line in sys.stdin:
        try:
            n = int(line.strip())
            if n == 0:  # 虽然题目说N>0，但以防万一
                continue
        except (ValueError, IndexError):
            # 忽略空行或格式不正确的行
            continue

        # dp[i][j]: 将整数 i 划分为最大数不超过 j 的方案数
        # 大小设为 (n+1) x (n+1)
        dp = [[0] * (n + 1) for _ in range(n + 1)]

        # 初始化边界条件
        # dp[0][j] = 1: 将0划分为最大数不超过j，只有一种方案(空划分)
        for j in range(n + 1):
            dp[0][j] = 1

        # 填充dp表
        # i 是要划分的整数
        for i in range(1, n + 1):
            # j 是划分中允许的最大整数
            for j in range(1, n + 1):
                # 情况1: 划分中不包含 j
                # 方案数等于将 i 划分为最大数不超过 j-1 的方案数
                dp[i][j] = dp[i][j-1]

                # 情况2: 划分中至少包含一个 j
                # 这要求 i >= j
                if i >= j:
                    # 方案数等于将 i-j 划分为最大数不超过 j 的方案数
                    dp[i][j] += dp[i-j][j]

        # 最终答案是 dp[n][n]
```

```
        # 因为划分n，其最大部分不可能超过n
        print(dp[n][n])


if __name__ == "__main__":
    solve()
```

代码运行截图 （至少包含有"Accepted"）

#50778633提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
import sys
def solve():
    """
    处理多组输入直到EOF
    """
    for line in sys.stdin:
        try:
```

基本信息

#: 50778633
题目: 04117
提交人: 25n2200013554
内存: 3616kB
时间: 26ms
语言: Python3
提交时间: 2025-11-10 14:01:03

M04137:最小新整数（30分钟）

monotonous-stack, http://cs101.openjudge.cn/practice/04137/

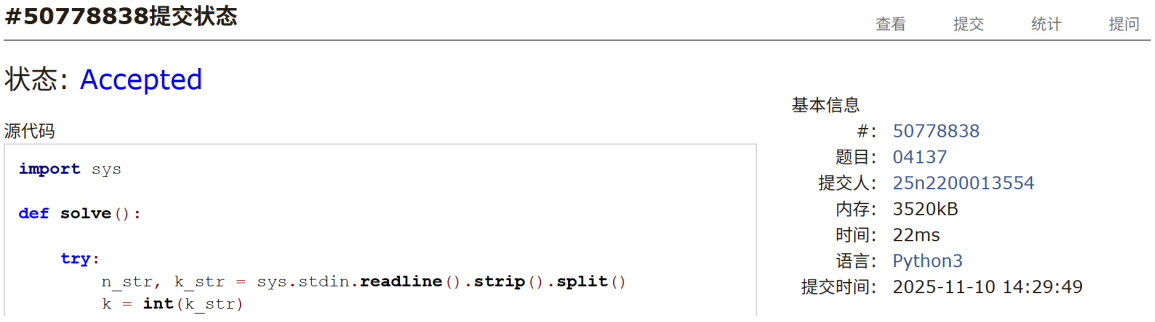
思路：尽可能地让结果数字的高位更小。从左到右构建最终的数字，使用栈来实现。

代码

```
import sys

def solve():

    try:
        n_str, k_str = sys.stdin.readline().strip().split()
        k = int(k_str)
    except (ValueError, IndexError):
        # 处理可能的空行或格式错误
        return

    # 使用列表模拟栈
    stack = []

    for digit in n_str:
        # 当栈不为空，k>0，且栈顶元素大于当前数字时，弹出栈顶
        while stack and k > 0 and stack[-1] > digit:
            stack.pop()
            k -= 1
        stack.append(digit)

    # 如果遍历完所有数字后 k 仍然大于 0，说明数字是升序的，需要从末尾删除
    if k > 0:
        stack = stack[:-k]
```

```
    # 将列表中的字符连接成字符串
    result = "".join(stack)

    # 输出结果
    print(result)


def main():
    """
    处理多组输入
    """
    try:
        t_str = sys.stdin.readline().strip()
        t = int(t_str)
        for _ in range(t):
            solve()
    except (ValueError, IndexError):
        return

if __name__ == "__main__":
    main()
```

代码运行截图 （至少包含有"Accepted"）



#50778838提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def solve():

    try:
        n_str, k_str = sys.stdin.readline().strip().split()
        k = int(k_str)
```

基本信息

#: 50778838
题目: 04137
提交人: 25n2200013554
内存: 3520kB
时间: 22ms
语言: Python3
提交时间: 2025-11-10 14:29:49

2. 学习总结和收获

本次考试由于时间冲突未能参加，只能周末自行完成相关题目。感觉考试题目难度较大，目前还未能仅仅依靠10张A4纸的cheating paper完成题目，同时由于此前未上过python语法的计概课，当程序报错时我基本上无法debug，openjudge也没有提供具体的报错情况，本人目前感觉对于本课程的上机考试非常忧虑。目前对于树的相关问题我的编程困难非常大，基本上看到题目没有什么思路，后续可能需要自行额外学习及练习一下。