数据结构与算法B 作业7： bfs、 🌲

> **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://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。
> 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

M23555: 节省存储的矩阵乘法

implementation, matrices, http://cs101.openjudge.cn/practice/23555

要求用节省内存的方式实现，不能还原矩阵的方式实现。

思路：
在做这题时，我想到既然要用节省内存的方式来实现，那么在完成矩阵乘法的三重遍历时，就必须判断对应的index是否非零，同时如果非零，要迅速调用对应的值。因此，我想到了用dict，将index储存为键，将矩阵对应位置上的数储存为值，这样判断和调用都是o(1)复杂度，十分省时，又节省内存。

代码：

```
def main():
    n, nums_m1, nums_m2 = [int(x) for x in input().split()]
    m1 = []
    m2 = []
    dict_m1 = {}
    dict_m2 = {}
    # 输入并储存行号/列号的index
    for i in range(nums_m1):
        temp = [int(x) for x in input().split()]
        if temp[0] in dict_m1.keys():
```

```
            dict_m1[temp[0]].append(i)
        else:
            dict_m1[temp[0]] = [i]
        m1.append(temp)
    for i in range(nums_m2):
        temp = [int(x) for x in input().split()]
        if temp[1] in dict_m2.keys():
            dict_m2[temp[1]].append(i)
        else:
            dict_m2[temp[1]] = [i]
        m2.append(temp)
    ans = []
    # 检测i、j是否在对应字典中存在，如果存在，再进行遍历
    for i in range(n):
        for j in range(n):
            temp = 0
            if i in dict_m1.keys() and j in dict_m2.keys():
                for k in range(n):
                    i_k = None
                    j_k = None
                    for i1 in dict_m1[i]:
                        if m1[i1][1] == k:
                            i_k = i1
                            break
                    for j1 in dict_m2[j]:
                        if m2[j1][0] == k:
                            j_k = j1
                            break
                    if i_k is not None and j_k is not None:
                        temp += m1[i_k][2] * m2[j_k][2]
            if temp:
                ans.append([i, j, temp])
    for i in ans:
        print(' '.join([str(x) for x in i]))

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

代码运行截图 （至少包含有"Accepted"）

#50524443提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
def main():
    n, nums_m1, nums_m2 = [int(x) for x in input().split()]
    m1 = []
    m2 = []
    dict_m1 = {}
    dict_m2 = {}
    # 输入并储存行号/列号的index
    for i in range(nums_m1):
        temp = [int(x) for x in input().split()]
        if temp[0] in dict_m1.keys():
            dict_m1[temp[0]].append(i)
        else:
            dict_m1[temp[0]] = [i]
        m1.append(temp)
    for i in range(nums_m2):
        temp = [int(x) for x in input().split()]
        if temp[1] in dict_m2.keys():
            dict_m2[temp[1]].append(i)
        else:
            dict_m2[temp[1]] = [i]
        m2.append(temp)
    ans = []
    # 检测i、j是否在对应字典中存在，如果存在，再进行遍历
    for i in range(n):
        for j in range(n):
            temp = 0
            if i in dict_m1.keys() and j in dict_m2.keys():
                for k in range(n):
```

基本信息

#: 50524443
题目: 23555
提交人: 22n2200011816(略彷横溪)
内存: 3860kB
时间: 56ms
语言: Python3
提交时间: 2025-10-23 19:56:37

M102.二叉树的层序遍历

bfs, https://leetcode.cn/problems/binary-tree-level-order-traversal/

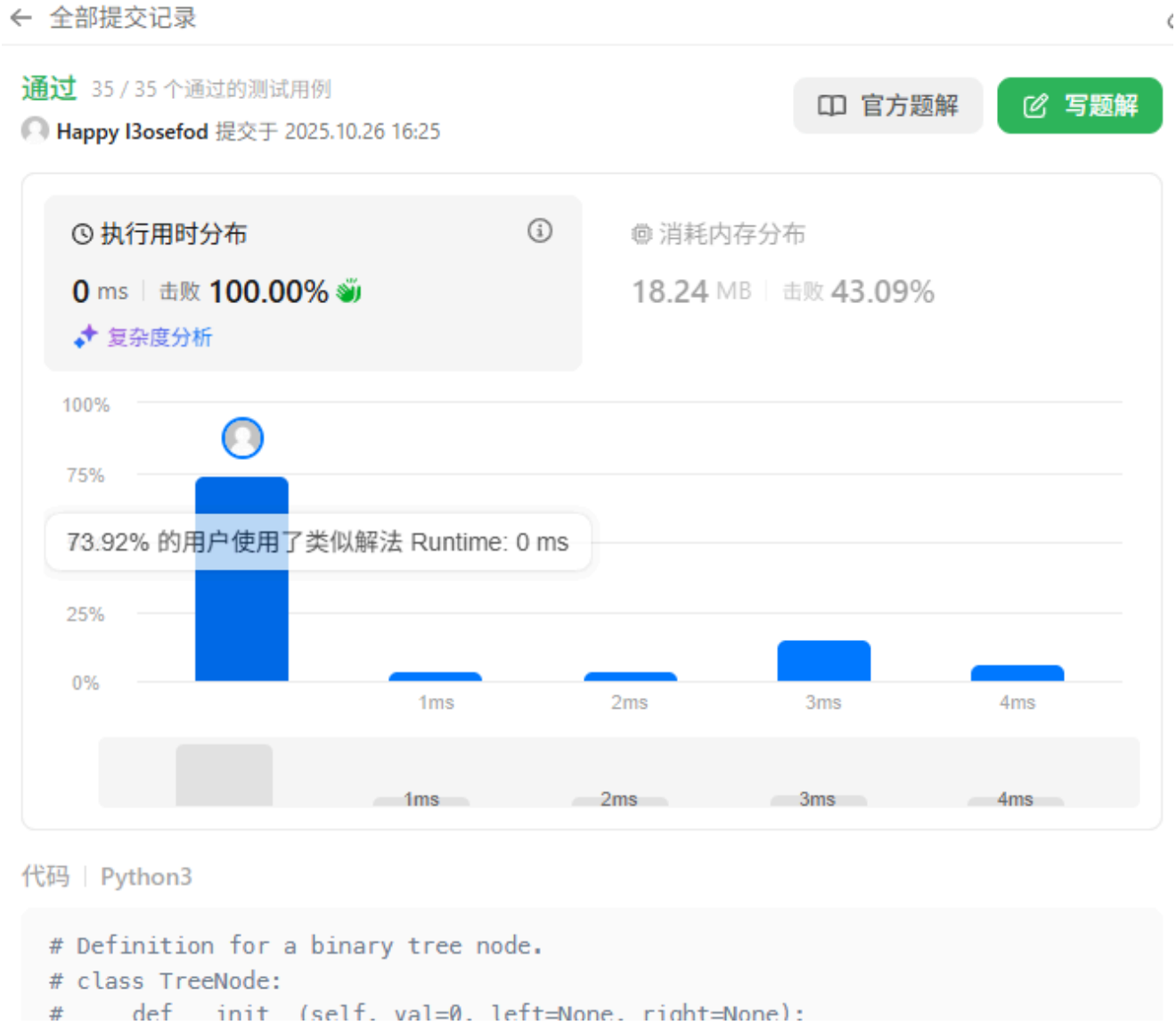
思路：
完成这道题目时，一开始想着用课件里面的层状遍历的代码，结果发现这边还需要将同层的元素放在一个列表里面（囧），后来想到用一个列表来储存当前的level中的node的剩余数量，对每一层，对queue popleft时，当前层的node数量减一，如果存在左右子节点，加入queue时顺便将下一层的node数加上去才实现。
最后看题解才发现可以直接用for循环解决每一层单独遍历（

代码：

```
class Solution:
    def levelOrder(self, root) -> list[list[int]]:
        ans = []
        if not root:
            return ans
        ans.append([])
        que = deque([root])
        level = 0
        node_in_level = [1,0]  # 列表储存层数和层中的节点
        while que:
            q = que.popleft()
            node_in_level[level] -= 1
            ans[-1].append(q.val)
```

```
            if q.left:
                que.append(q.left)
                node_in_level[level+1] += 1
            if q.right:
                que.append(q.right)
                node_in_level[level+1] += 1
            if node_in_level[level] == 0 and que:
                ans.append([])
                level += 1
                node_in_level.append(0)
        return ans
```

代码运行截图 （至少包含有"Accepted"）



M131.分割回文串

dp, backtracking, https://leetcode.cn/problems/palindrome-partitioning/

思路：
在完成这道题目时，本来没什么思路，一直想把dp和backtracking结合在一起，一直想不到怎么做，后来忙其他事情，过了几天再回来写就很快明白了：将dp和backtracking分开写就可

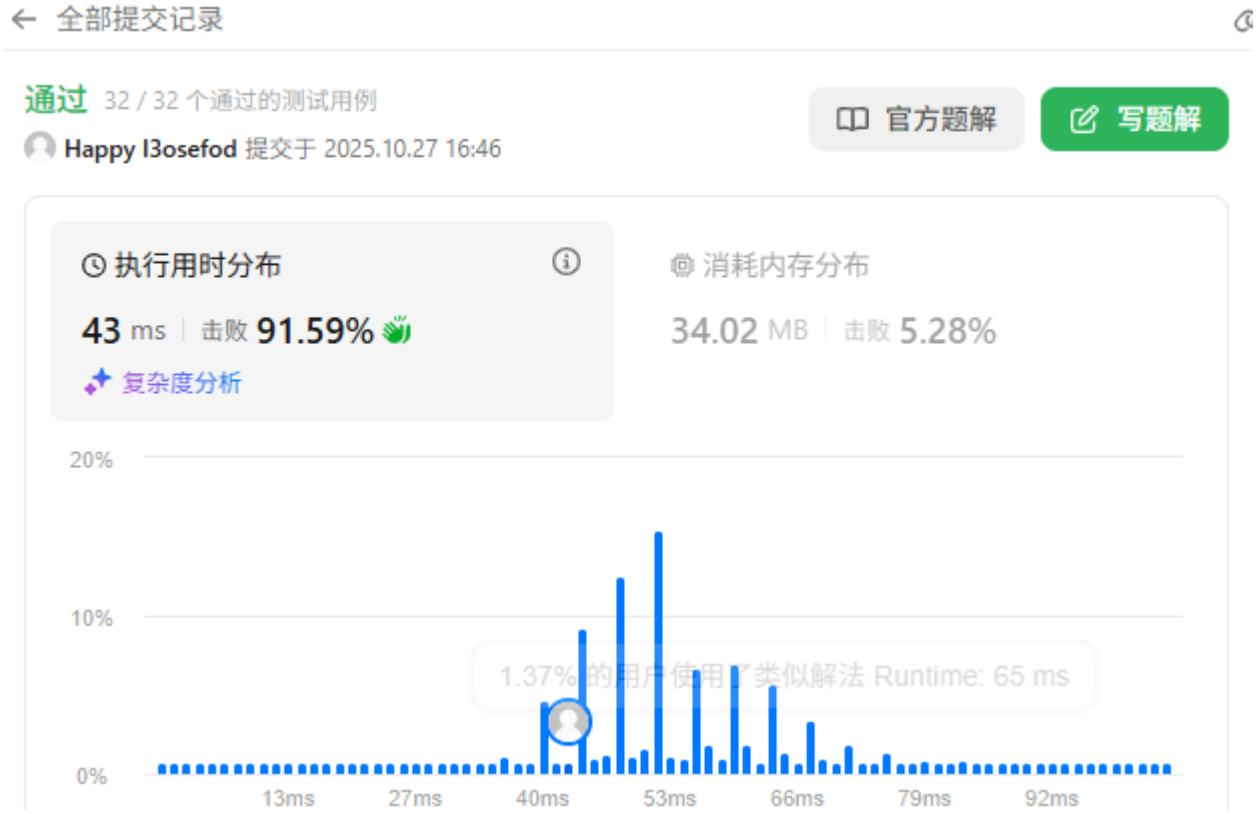
以。
首先是dp，可以先对str做一个二重遍历，获得str中所有的回文串，存储起来。此时，我想到利用set()不会重复、in的判断是O(1)复杂度的性质存储可能的回文串。当然，由于直接存储了一堆字符串，用了更多的内存；好处是节省了不少时间。
然后是backtracking部分，也是第一次这么快想出来怎么递归：如果当前切片是回文，就保存，并在剩下的字符串里面继续从左到右切片，这样就很快想出来怎么递归了。

代码：

```
class Solution:
    def partition(self, s: str) -> list[list[str]]:
        def is_palin(s):
            if s == s[::-1]:
                return True
            else:
                return False
        palin = set()
        s_len = len(s)
        for sub_len in range(1, s_len + 1):
            for j in range(0, s_len - sub_len + 1):
                if is_palin(s[j:j+sub_len]):
                    palin.add(s[j:j+sub_len])
        result = []

        def backtrack(index_left, trace):
            for i in range(index_left+1, s_len + 1):
                if s[index_left:i] in palin:
                    trace.append(s[index_left:i])
                    if i == s_len:
                        result.append(trace[:])
                    else:
                        backtrack(i, trace)
                    trace.pop()
        backtrack(0, trace=[])
        return result
```

代码运行截图 （至少包含有"Accepted"）



M200.岛屿数量

dfs, bfs, https://leetcode.cn/problems/number-of-islands/

思路：
用了bfs的方法去解。具体的思路是：对某个点，如果是岛屿，就输入进queue中，并将四个方向上为'1'的index放入queue并取为'0'，对queue中的每个元素都如是操作即可。这样就可以将连续的'1'只记为一个岛屿。

代码

```
def edge(A:list,k):
    x=len(A[0]);y=len(A)
    for i in range(y):
        A[i]=[k]+A[i]+[k]
    A.append([k]*(x+2));A.insert(0,[k]*(x+2))
    return A


class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        grid = edge(grid, 0)
        direc = [[1, 0], [-1, 0], [0, 1], [0, -1]]
        num_island = 0

        def bfs(index_x, index_y):
            que = deque([(index_x, index_y)])
            grid[index_x][index_y] = '0'
```

```
            while que:
                temp = que.popleft()
                for i in direc:
                    x1 = temp[0] + i[0]
                    y1 = temp[1] + i[1]
                    if grid[x1][y1] == '1':
                        que.append((x1, y1))
                        grid[x1][y1] = '0'
            return 1

        for i in range(1, len(grid)-1):
            for j in range(1, len(grid[0])-1):
                if grid[i][j] == '1':
                    num_island += bfs(i, j)
        return num_island
```

（至少包含有"Accepted"）

1123.最深叶节点的最近公共祖先

dfs, https://leetcode.cn/problems/lowest-common-ancestor-of-deepest-leaves/

思路：
一开始对题目的理解有问题，怎么写都写不对，后来看了题解才知道原来是“所有“最深叶节点的公共祖先，以及顺便了解了可以在递归中通过比较左右叶节点的深度来确认祖先节点。秉持这个思路，修改了代码最终ac了

代码

```
class Solution:
    def lcaDeepestLeaves(self, root: Optional[TreeNode]) ->
```

```
Optional[TreeNode]:
        blank_node = TreeNode(None, None, None)

        def preorder_traversal(parent_node, node, level):
            if node:
                node_left, level1 = preorder_traversal(node, node.left,
level + 1)
                node_right, level2 = preorder_traversal(node, node.right,
level + 1)
                if level1 == level2:
                    return node, level1
                elif level1 > level2:
                    return node_left, level1
                elif level1 < level2:
                    return node_right, level2
            else:
                return parent_node, level - 1
        ans = preorder_traversal(blank_node, root, 0)
        return ans[0]
```

M79.单词搜索

回溯，https://leetcode.cn/problems/word-search/

思路：
用了DFS的写法，根据递归的深度逐位比较临近index中的字母是否符合即可。

代码：

```
def edge(A:list,k):
    x=len(A[0]);y=len(A)
    for i in range(y):
```

```
        A[i]=[k]+A[i]+[k]
    A.append([k]*(x+2));A.insert(0,[k]*(x+2))
    return A

def backtrack(board, track, index_x, index_y, word, index_word):
    direc = [[1, 0], [-1, 0], [0, 1], [0, -1]]
    if board[index_x][index_y] == word[index_word]:
        if index_word == len(word)-1:
            return True
        else:
            track[index_x][index_y] = 1
            for i in range(4):
                dx = direc[i][0]; dy = direc[i][1]
                if track[index_x+dx][index_y+dy] == 0:
                    if backtrack(board, track, index_x+dx, index_y+dy, word,
index_word+1):
                        return True
            track[index_x][index_y] = 0
    return False

class Solution:
    def exist(self, board, word: str) -> bool:
        track = [[1 for _ in range(len(board[0])+ 2)] for _ in
range(len(board) + 2)]
        for i in range(1, len(board) + 1):
            for j in range(1, len(board[0]) + 1):
                track[i][j] = 0
        board = edge(board, 1)
        for i in range(1, len(board)-1):
            for j in range(1, len(board[0])-1):
                if board[i][j] == word[0]:
                    if backtrack(board, track, i, j, word, index_word=0):
                        return True
        return False
```

代码运行截图（至少包含有"Accepted"）

2. 学习总结和个人收获

第一次学习树这种数据结构，不过由于之前有链表题目的锻炼，对树这种结构上手快多了。同时感觉这周的DFS\BFS\DP的题目都挺有意思，还是比较有挑战性的。