Assignment #5: cs201 Mock Exam寒露第三天

Updated 1913 GMT+8 Oct 10, 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://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

E29952: 咒语序列（30分钟）

Stack, http://cs101.openjudge.cn/practice/29952/

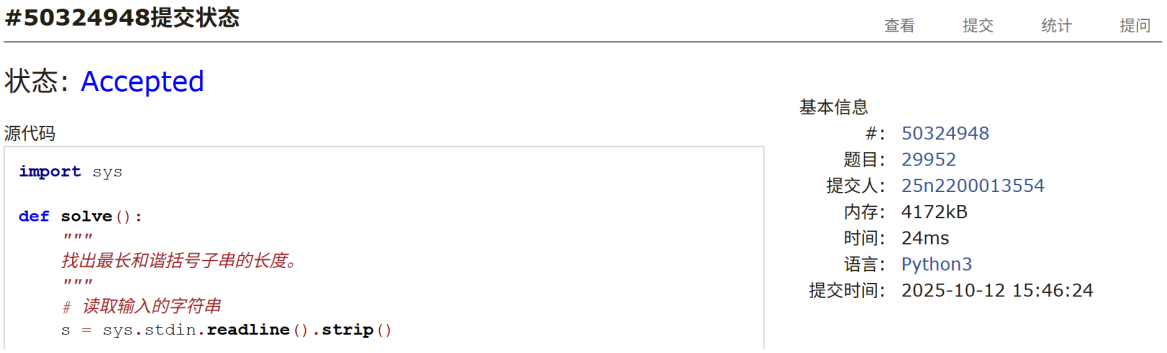
思路：为左括号找到匹配的右括号组成序列，找到无法被匹配的右括号作为断点，在断点间找到最长的序列。

代码：

```
import sys

def solve():
    """
    找出最长和谐括号子串的长度。
    """
    # 读取输入的字符串
    s = sys.stdin.readline().strip()

    # max_len 用于记录最长和谐子串的长度
    max_len = 0
```

```
    # 栈用于存储索引，初始放入-1作为基准
    stack = [-1]

    # 遍历字符串
    for i, char in enumerate(s):
        if char == '(':
            # 如果是左括号，将其索引压入栈中
            stack.append(i)
        else: # 如果是右括号
            # 弹出一个元素
            stack.pop()

            if not stack:
                # 如果弹出后栈为空，说明当前的右括号是“多余的”
                # 将它的索引压入，作为新的基准点
                stack.append(i)
            else:
                # 如果栈不为空，计算当前有效长度
                # 长度 = 当前索引 - 新的栈顶索引
                current_len = i - stack[-1]
                max_len = max(max_len, current_len)

    print(max_len)

# 调用函数执行
solve()
```

代码运行截图（至少包含有"Accepted"）



#50324948提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
import sys

def solve():
    """
    找出最长和谐括号子串的长度。
    """
    # 读取输入的字符串
    s = sys.stdin.readline().strip()
```

基本信息

#: 50324948
题目: 29952
提交人: 25n2200013554
内存: 4172kB
时间: 24ms
语言: Python3
提交时间: 2025-10-12 15:46:24

M01328: Radar Installation（30分钟）

greedy, http://cs101.openjudge.cn/practice/01328/

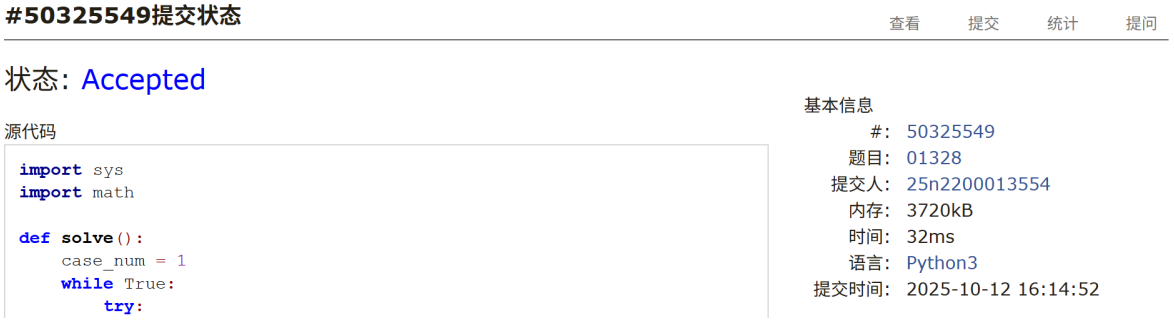
思路：雷达位于x轴上，如果岛域到x轴的距离大于雷达的探测范围则该岛域无法覆盖。因为雷达位于x轴上，把问题从将找最少的圆形区域，覆盖所有给定的点的二维转化为在一维直线上（x轴），有n个给定的区间[L1, R1], [L2, R2], ...（对应可以覆盖到n个岛的各自的雷达放置范围）。需要选择最少的点，使得每个区间都至少包含一个你选择的点。按区间的右端点升序排序后将点放置在右端点，让其能涉及更多的范围。

代码：

```python
import sys
import math

def solve():
    case_num = 1
    while True:
        try:
            line = sys.stdin.readline()
            if not line.strip(): # 处理案例间的空行
                line = sys.stdin.readline()

            n, d = map(int, line.split())
        except (ValueError, IndexError):
            break # 输入结束

        if n == 0 and d == 0:
            break

        possible = True
        intervals = []

        for _ in range(n):
            x, y = map(int, sys.stdin.readline().split())

            # 如果岛屿太高，雷达无法覆盖
            if y > d:
                possible = False

            # 计算雷达可以放置的 x 轴区间
            if possible:
                half_width = math.sqrt(d*d - y*y)
                left = x - half_width
                right = x + half_width
                intervals.append((left, right))

        # 如果在读取过程中发现无解，需要读完剩下的行
        if not possible:
            # 即使已经确定无解，也要把当前测试用例的剩余输入行读完
            # for _ in range(len(intervals), n):
            #     sys.stdin.readline()
            print(f"Case {case_num}: -1")
            case_num += 1
            continue

        # 按区间的右端点升序排序
        # x[1] 表示按元组的第二个元素 (right) 排序
        intervals.sort(key=lambda x: x[1])

        radar_count = 0
        last_radar_pos = -float('inf')

        for left, right in intervals:
            # 如果当前区间的左端点在前一个雷达的覆盖范围之外
            if left > last_radar_pos:
                # 需要一个新的雷达
```

```
                radar_count += 1
                # 将新雷达放在当前区间的右端点，这是最优选择
                last_radar_pos = right

        print(f"Case {case_num}: {radar_count}")
        case_num += 1

# 调用函数执行
if __name__ == "__main__":
    solve()
```

代码运行截图 （至少包含有"Accepted"）



M02754: 八皇后（1小时）

dfs, http://cs101.openjudge.cn/practice/02754/

思路：先用回溯算法找到所有皇后布局，再按大小排序，主程序读取输入，并从预先计算好的解列表中查找答案。

代码：

```
import sys

def solve_all_solutions():
    """
    使用回溯法生成8皇后问题的所有解，并按字典序排序。
    """
    solutions = []
    # queens[row] = col 表示第 row 行的皇后放在第 col 列
    queens = [-1] * 8

    def is_valid(row, col):
        """检查在 (row, col) 放置皇后是否与之前的皇后冲突"""
        for r in range(row):
            # 检查同列 或 同对角线
            # abs(row - r) == abs(col - queens[r]) 判断对角线
            if queens[r] == col or abs(row - r) == abs(col - queens[r]):
                return False
        return True
```

```
    def backtrack(row):
        """递归函数，逐行放置皇后"""
        # 终止条件：所有行都已成功放置皇后
        if row == 8:
            # 找到一个解，将其添加到 solutions 列表
            solutions.append(queens.copy())
            return

        # 遍历当前行的所有列
        for col in range(8):
            if is_valid(row, col):
                # 做出选择
                queens[row] = col
                # 进入下一层决策
                backtrack(row + 1)
                # 回溯是隐式的，因为下一轮循环会覆盖 queens[row] 的值

    # 从第 0 行开始回溯
    backtrack(0)

    # 将解转换为题目要求的皇后串格式
    # queens 中的列是 0-7，题目要求是 1-8，所以要 +1
    solution_strings = []
    for sol in solutions:
        s = "".join(map(lambda x: str(x + 1), sol))
        solution_strings.append(s)

    # 按数值大小（即字典序）排序
    solution_strings.sort()

    return solution_strings

# --- 主程序 ---
def main():
    # 1. 一次性生成所有解并排序
    all_queen_solutions = solve_all_solutions()

    try:
        # 读取测试用例的数量
        num_cases = int(sys.stdin.readline())

        # 循环处理每个测试用例
        for _ in range(num_cases):
            # 读取 b
            b = int(sys.stdin.readline())

            # 2. 从预计算的列表中直接查找答案
            # b 是 1-indexed，列表是 0-indexed，所以用 b-1
            print(all_queen_solutions[b - 1])

    except (ValueError, IndexError):
        # 处理可能的空输入或格式错误
        return

# 调用主函数
if __name__ == "__main__":
```

```
    main()
```

代码运行截图 （至少包含有"Accepted"）

#50324535提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def solve_all_solutions():
    """
    使用回溯法生成8皇后问题的所有解，并按字典序排序。
    """
    solutions = []
    # queens[row] = col 表示第 row 行的皇后放在第 col 列
    queens = [-1] * 8
```

基本信息

#: 50324535
题目: 02754
提交人: 25n2200013554
内存: 3656kB
时间: 31ms
语言: Python3
提交时间: 2025-10-12 15:24:23

M25570: 洋葱（25分钟）

matrices, http://cs101.openjudge.cn/practice/25570/

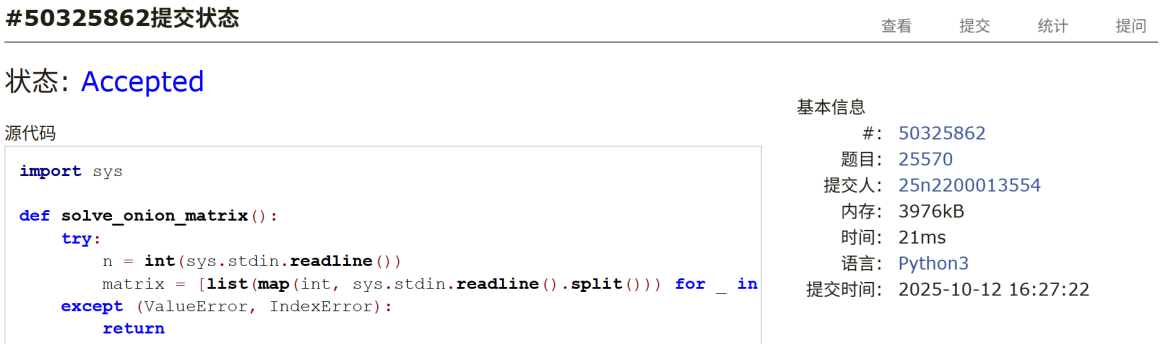
思路：循环求解每一层。分成四个边求解，比较出最大值。如果层数是单数要注意处理最里面层只有一个数字的情况。

代码：

```
import sys

def solve_onion_matrix():
    try:
        n = int(sys.stdin.readline())
        matrix = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]
    except (ValueError, IndexError):
        return

    max_sum = -1

    # 遍历所有层，从最外层 (layer 0) 到最内层
    num_layers = (n + 1) // 2
    for layer in range(num_layers):
        current_sum = 0

        # 这一层的子矩阵边长
        side = n - 2 * layer

        # 如果是 1x1 的中心层
        if side == 1:
            current_sum = matrix[layer][layer]
        else:
            # 计算上边和下边的和
            for j in range(side):
                current_sum += matrix[layer][layer + j] # 上边
                current_sum += matrix[layer + side - 1][layer + j] # 下边
```

```
            # 计算左边和右边的和 (不包括已经加过的角点)
            for i in range(1, side - 1):
                current_sum += matrix[layer + i][layer] # 左边
                current_sum += matrix[layer + i][layer + side - 1] # 右边

        # 更新最大和
        if max_sum == -1 or current_sum > max_sum:
            max_sum = current_sum

    print(max_sum)

solve_onion_matrix()
```

代码运行截图 （至少包含有"Accepted"）



#50325862提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def solve_onion_matrix():
    try:
        n = int(sys.stdin.readline())
        matrix = [list(map(int, sys.stdin.readline().split())) for _ in
    except (ValueError, IndexError):
        return
```

基本信息

#: 50325862
题目: 25570
提交人: 25n2200013554
内存: 3976kB
时间: 21ms
语言: Python3
提交时间: 2025-10-12 16:27:22

M29954: 逃离紫罗兰监狱（2小时）

bfs, http://cs101.openjudge.cn/practice/29954/

思路：在记录位置的基础上需要再加一个记录闪现的使用情况。故需要一个三元组。

代码

```
import sys
from collections import deque

def solve():
    # --- 1. 读取输入和初始化 ---
    try:
        R, C, K = map(int, sys.stdin.readline().split())
        grid = [sys.stdin.readline().strip() for _ in range(R)]
    except (ValueError, IndexError):
        return

    start_pos = None
    end_pos = None
    for r in range(R):
        for c in range(C):
```

```
            if grid[r][c] == 'S':
                start_pos = (r, c)
            elif grid[r][c] == 'E':
                end_pos = (r, c)

    # 队列: (row, col, remaining_blinks, distance)
    queue = deque([(start_pos[0], start_pos[1], K, 0)])

    # visited 数组: visited[r][c][k]
    visited = [[[False for _ in range(K + 1)] for _ in range(C)] for _ in
range(R)]
    visited[start_pos[0]][start_pos[1]][K] = True

    # 定义四个移动方向
    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    # --- 2. BFS 循环 ---
    while queue:
        r, c, k, dist = queue.popleft()

        # 检查是否到达终点
        if (r, c) == end_pos:
            print(dist)
            return

        # 探索相邻格子
        for dr, dc in moves:
            nr, nc = r + dr, c + dc

            # 检查边界
            if 0 <= nr < R and 0 <= nc < C:
                # 情况一：目标是走廊
                if grid[nr][nc] in ['.', 'S', 'E']:
                    if not visited[nr][nc][k]:
                        visited[nr][nc][k] = True
                        queue.append((nr, nc, k, dist + 1))

                # 情况二：目标是屏障
                elif grid[nr][nc] == '#':
                    # 检查是否有闪现次数
                    if k > 0:
                        if not visited[nr][nc][k - 1]:
                            visited[nr][nc][k - 1] = True
                            queue.append((nr, nc, k - 1, dist + 1))

    # --- 3. 无法到达 ---
    print(-1)

# 调用函数执行
solve()
```

(至少包含有"Accepted")

#50330368提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
import sys
from collections import deque

def solve():
    # --- 1. 读取输入和初始化 ---
    try:
        R, C, K = map(int, sys.stdin.readline().split())
```

基本信息

#: 50330368
题目: 29954
提交人: 25n2200013554
内存: 5732kB
时间: 182ms
语言: Python3
提交时间: 2025-10-12 20:20:06

T27256: 当前队列中位数（3个小时）

backtracking, http://cs101.openjudge.cn/practice/27256/

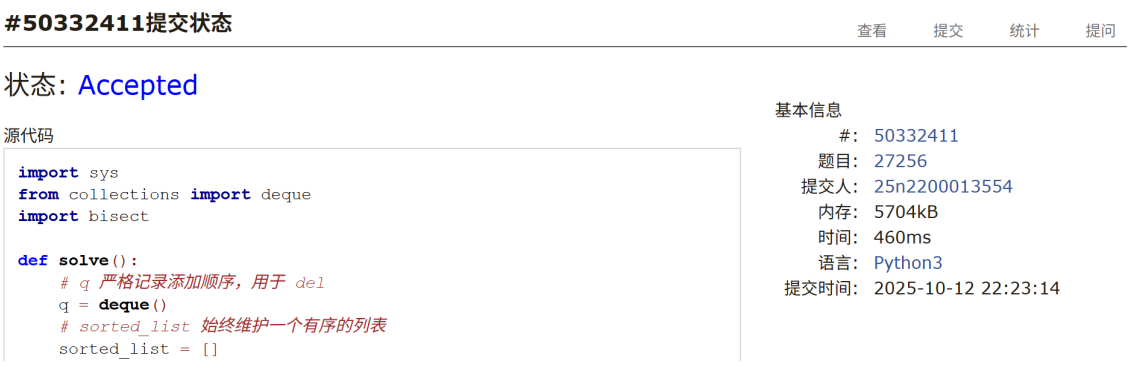
思路：同时使用两种数据结构，顺序需求——del 操作要求删除最早添加的元素，有序需求——query 操作要求快速找到中位数，需要数据保持有序。因此采用deque (双端队列)专门负责维护元素的添加顺序和sorted_list (有序列表)专门负责维护数据的有序性。该方案的思路是用 add 和 del 操作中较高的 O(k) 复杂度，来换取 query 操作极低的 O(1) 复杂度。

代码

```
import sys
from collections import deque
import bisect

def solve():
    # q 严格记录添加顺序，用于 del
    q = deque()
    # sorted_list 始终维护一个有序的列表
    sorted_list = []

    try:
        n = int(sys.stdin.readline())
    except (ValueError, IndexError):
        return

    for _ in range(n):
        line = sys.stdin.readline().strip().split()
        op = line[0]

        if op == "add":
            x = int(line[1])
            q.append(x)
            # 使用 bisect.insort 来保持列表有序
            bisect.insort(sorted_list, x)

        elif op == "del":
            if q:
                val_to_delete = q.popleft()
                # 找到 val_to_delete 在有序列表中的位置并删除
                # bisect.bisect_left 可以在 O(log k) 找到位置
                # .pop(index) 是 O(k)
```

```
                index = bisect.bisect_left(sorted_list, val_to_delete)
                sorted_list.pop(index)

        elif op == "query":
            k = len(sorted_list)
            if k % 2 == 1: # 奇数个
                median = float(sorted_list[k // 2])
            else: # 偶数个
                mid1 = sorted_list[k // 2 - 1]
                mid2 = sorted_list[k // 2]
                median = (mid1 + mid2) / 2.0

            if median == int(median):
                print(int(median))
            else:
                print(median)

solve()
```

(至少包含有"Accepted")



#50332411提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys
from collections import deque
import bisect

def solve():
    # q 严格记录添加顺序，用于 del
    q = deque()
    # sorted_list 始终维护一个有序的列表
    sorted_list = []
```

基本信息

#: 50332411
题目: 27256
提交人: 25n2200013554
内存: 5704kB
时间: 460ms
语言: Python3
提交时间: 2025-10-12 22:23:14

2. 学习总结和个人收获

这次月考我感觉非常吃力，题目难度很大。我对python的基本语法还是不熟练，在月考时出现了缩进的错误。目前我还不是很能熟练使用python语法自行书写代码。后续需要多读代码，月考前整理cheating paper感觉收获很多，希望能在后续考试中通过参考的10张A4纸资料的辅助能顺利写出月考题目。目前学习了一个月最大的感受是python把代码分装成一个一个函数，在主程序中对需要的功能进行调用。