会议室调度问题/区间合并问题

```python
import heapq

def minMeetingRooms(intervals):
    if not intervals:
        return 0

    # 按照开始时间排序
    intervals.sort(key=lambda x: x[0])

    # 初始化堆
    free_rooms = []
    heapq.heappush(free_rooms, intervals[0][1])

    for i in intervals[1:]:
        # 如果最早结束的会议室结束时间小于等于当前会议的开始时间，说明可以重用该会议室
        if free_rooms[0] <= i[0]:
            heapq.heappop(free_rooms)

        # 新的会议室
        heapq.heappush(free_rooms, i[1])

    return len(free rooms)
def merge(intervals):
    if not intervals:
        return []

    # 按照起始时间排序
    intervals.sort(key=lambda x: x[0])
    merged = [intervals[0]]

    for i in intervals[1:]:
        # 如果当前区间与上一个区间有重叠，合并
        if i[0] <= merged[-1][1]:
            merged[-1][1] = max(merged[-1][1], i[1])
        else:
            merged.append(i)

    return merged

# 示例输入
print(merge([[1, 3], [2, 6], [8, 10], [15, 18]]))  # 输出 [[1, 6], [8, 10], [15, 18]]
```

区间合并问题**排序**：首先按区间的开始时间排序，接着遍历区间，如果当前区间的开始时间小于等于上一个区间的结束时间，则进行合并。

（双指针）区间交集问题两个区间 `[start1, end1]` 和 `[start2, end2]` 的交集是 `[max(start1, start2), min(end1, end2)]`，前提是交集部分有效（即 `max(start1, start2) <= min(end1, end2)`）

```python
def intervalIntersection(A, B):
    res = []
    i, j = 0, 0

    while i < len(A) and j < len(B):
        # 找到交集
        start = max(A[i][0], B[j][0])
        end = min(A[i][1], B[j][1])

        if start <= end:
            res.append([start, end])

        # 移动指针
        if A[i][1] < B[j][1]:
            i += 1
        else:
            j += 1

    return res
```

```python
def videoStitching(self, clips: List[List[int]], time: int) -> int:
    # 对 clips 按起点升序排序
    clips.sort()

    st, ed = 0, time
    res = 0

    i = 0
    while i < len(clips) and st < ed:
        maxR = 0
        # 找到所有起点小于等于 st 的片段，并记录这些片段的最大终点 maxR
        while i < len(clips) and clips[i][0] <= st:
            maxR = max(maxR, clips[i][1])
            i += 1

        if maxR <= st:
            # 无法继续覆盖
            return -1

        # 更新 st 为 maxR，并增加结果计数
        st = maxR
        res += 1

        if maxR >= ed:
            # 已经覆盖到终点
            return res

    # 如果没有成功覆盖到终点
    return -1
```

非双指针做法

```
def findOverlap(intervals):
    if not intervals:
        return []

    # 按照区间的起始位置排序
    intervals.sort(key=lambda x: x[0])
    overlap = []

    for i in range(1, len(intervals)):
        if intervals[i][0] <= intervals[i-1][1]:
            # 当前区间与前一个区间有重叠
            overlap.append([max(intervals[i][0], intervals[i-1][0]), min(intervals[i][1],

    return overlap

# 示例
intervals = [[1, 5], [2, 6], [8, 10], [7, 9]]
result = findOverlap(intervals)
print(result)  # 输出: [[2, 5], [7, 9]]
```

覆盖区间问题（视频拼接问题）（这个比较难，还不太熟悉）
对视频片段按照左端点升序排序。
选择能够到达左端点的区间，在此基础上进行不断优化，使得新的左端点不断右移。

```
def videoStitching(self, clips: List[List[int]], time: int) -> int:
        # 对 clips 按起点升序排序
        clips.sort()
        st, ed = 0, time
        res = 0
        selected_intervals = []  # 用于保存选中的区间

    i = 0
    while i < len(clips) and st < ed:
        maxR = 0
        # 找到所有起点小于等于 st 的片段，并记录这些片段的最大终点 maxR
        best_clip = None  # 记录当前选择的最佳片段
        while i < len(clips) and clips[i][0] <= st:
            if clips[i][1] > maxR:
                maxR = clips[i][1]
                best_clip = clips[i]  # 更新当前最佳片段
            i += 1

        if maxR <= st:
            # 无法继续覆盖
            return -1

        # 更新 st 为 maxR，并增加结果计数
        st = maxR
        res += 1
        selected_intervals.append(best_clip)  # 添加选中的区间

        if maxR >= ed:
```

```
            # 已经覆盖到终点
            return res, selected_intervals

    # 如果没有成功覆盖到终点
    return -1, []
```

手动实现矩阵乘法：

```python
def matrix_multiply(A, B):
    # 获取 A 和 B 的维度
    m = len(A)       # A 的行数
    n = len(A[0])    # A 的列数，即 B 的行数
    p = len(B[0])    # B 的列数

    # 初始化结果矩阵 C，大小为 m x p，初始值为 0
    C = [[0] * p for _ in range(m)]

    # 计算矩阵 C 中的每一个元素
    for i in range(m):  # 遍历 A 的每一行
        for j in range(p):  # 遍历 B 的每一列
            C[i][j] = sum(A[i][k] * B[k][j] for k in range(n))

    return C
```

1. 矩阵旋转

题目：给定一个 `n x n` 的矩阵，旋转矩阵 90 度。

- **解法**：
  - 可以通过转置矩阵和反转行来完成旋转。
  - 也可以通过原地交换元素的方法进行旋转。

示例代码：

```python
def rotate(matrix):
    n = len(matrix)
    # 转置矩阵
    for i in range(n):
        for j in range(i, n):
            matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
    # 反转每一行
    for i in range(n):
        matrix[i].reverse()
```

如果给定一个已经填充好的矩阵，要求以螺旋顺序遍历它，可以按照类似的策略，从上边界、右边界、下边界和左边界依次访问元素，并每访问完一轮就更新边界。
矩阵旋转更大的度数可以通过多次旋转90度得到。

```
def generateMatrix(n: int):
    # 创建一个 n x n 的零矩阵
    matrix = [[0] * n for _ in range(n)]
    left, right = 0, n - 1  # 列的边界
    top, bottom = 0, n - 1  # 行的边界

    num = 1  # 填充的数字，从1开始

    while left <= right and top <= bottom:
        # 填充上边界（从左到右）
        for i in range(left, right + 1):
            matrix[top][i] = num
            num += 1
        top += 1  # 更新上边界
```

```
        # 填充右边界（从上到下）
        for i in range(top, bottom + 1):
            matrix[i][right] = num
            num += 1
        right -= 1  # 更新右边界

        if top <= bottom:
            # 填充下边界（从右到左）
            for i in range(right, left - 1, -1):
                matrix[bottom][i] = num
                num += 1
            bottom -= 1  # 更新下边界

        if left <= right:
            # 填充左边界（从下到上）
            for i in range(bottom, top - 1, -1):
                matrix[i][left] = num
                num += 1
            left += 1  # 更新左边界

    return matrix
```

sys.stdin.read() sys.stdoutwrite()当数据量较多较复杂时，可以优化耗时。

代码实现（处理多行输入）：

```
import sys

# 假设输入数据是多行
input_data = sys.stdin.read()

# 按行分割
lines = input_data.splitlines()

# 输出读取的内容
for line in lines:
    print(line)
```

python

```
import sys

# 读取所有输入数据
input_data = sys.stdin.read().strip()

# 按行分割输入
lines = input_data.splitlines()

# 遍历每一行，反转字符串并输出
for line in lines:
    sys.stdout.write(line[::-1] + '\n')
```

5.BFS

字符迷宫/简单走迷宫

Tip：当1表示障碍物时要将1作为初始化的值，否则容易发生RTE索引越界

对maze添加保护圈时，要记得两边都添加的话是2，别忘了给最上面添加保护圈。

如果RTE的话看看是不是添加入队时忘记加上了dx和dy。

也可以直接原地修改maze的值来标记其状态，如0，-1，1，2。

（最短路径问题）一般情况下第一个返回的值就是最短的，此时可以退出代码

```python
from collections import deque
n, m = map(int, input().split())
S=[]
maze = [["*" for _ in range(m + 2)] for _ in range(n + 2)]
for i in range(1,n+1):
    a=input()
    for j in range(len(a)):
        if a[j]=="S":
            S.append([i,j+1])
        maze[i][j+1]=a[j]
dir = [(1, 0), (-1, 0), (0, -1), (0, 1)]


def BFS(x, y):
    q = deque([(x, y, 0)])
    while q:
        x, y, step = q.popleft()

        for i in range(4):
            dx = dir[i][0]
            dy = dir[i][1]
            if maze[x + dx][y + dy] == ".":
                maze[x + dx][y + dy] = "*"
                q.append((x + dx, y + dy, step + 1))
            if maze[x + dx][y + dy] == "T":
                return step + 1
    return -1

a,b=S
res = BFS(a,b)
print(res)
```

```python
from collections import deque
dir = [(1, 0), (0, 1), (-1, 0), (0, -1)]
n, m = map(int, input().split())
maze = [[1 for _ in range(m+2)] for _ in range(n+2)]
for i in range(1, n + 1):
    maze[i][1:-1] = list(map(int, input().split()))
result = []
def BFS(x, y):
    q = deque([(x, y,0)])
    visit = set()
    visit.add((x, y))
    while q:
        x,y,step= q.popleft()
        if x == n and y == m:
            return step
        for i in range(4):
            dx = dir[i][0]
            dy = dir[i][1]
            if not maze[x + dx][y + dy] and (x+dx,y+dy) not in visit:
                q.append((x + dx, y + dy, step + 1))
                visit.add((x + dx, y + dy))
    return -1
res=BFS( x: 1, y: 1)
print(res)
```

传送点问题

```python
from collections import deque
dir=[(1,0),(-1,0),(0,1),(0,-1)]
n,m=map(int,input().split())
chuansongdian=[]
maze=[[1 for _ in range(m+2)]for _ in range(n+2)]
for i in range(1,n+1):
    a=list(map(int,input().split()))
    for j in range(len(a)):
        if a[j]==2:
            chuansongdian.append((i,j+1))
        maze[i][j+1]=a[j]
def BFS(x,y):
    q=deque([(x,y,0)])
    while q:
        x,y,step=q.popleft()
        if x==n and y==m:
            return step
        if (x,y)==chuansongdian[0] and maze[chuansongdian[1][0]][chuansongdian[1][1]]==2:
            q.append((chuansongdian[1][0],chuansongdian[1][1],step))
            maze[chuansongdian[1][0]][chuansongdian[1][1]] =1
        elif (x,y)==chuansongdian[1] and maze[chuansongdian[0][0]][chuansongdian[0][1]]==2:
            q.append((chuansongdian[0][0],chuansongdian[0][1],step))
            maze[chuansongdian[0][0]][chuansongdian[0][1]] =1
        for i in range(4):
            dx=dir[i][0]
            dy=dir[i][1]
            if maze[x+dx][y+dy]!=1:
                q.append((x+dx,y+dy,step+1))
                maze[x+dx][y+dy]=1
    return -1
res=BFS( x: 1, y: 1)
print(res)
```

遍历标记某块区域，如岛屿。

```python
from collections import deque

# 方向数组：上、下、左、右
directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

def bfs(grid, start_x, start_y):
    # BFS 队列初始化
    queue = deque([(start_x, start_y)])
    grid[start_x][start_y] = 0  # 将该点标记为已访问

    while queue:
        x, y = queue.popleft()

        # 访问四个方向的邻居
        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < len(grid) and 0 <= ny < len(grid[0]) and grid[nx][ny] == 1:
                grid[nx][ny] = 0  # 将该点标记为已访问
                queue.append((nx, ny))  # 将该点加入队列
```

字典的某些操作/对列表的某些操作

enumerate（列表）

```
>>> my_dict={"a":0,"b":1,"c":2,"d":3,"e":5}
... print(my_dict.values())
... print(my_dict.keys())
... print(my_dict.items())
... for i in my_dict.values():
...     print(i,end=' ')
... for i in my_dict.keys():
...     print(i,end=" ")
... for i in my_dict.items():
...     print(i,end=" ")
...
dict_values([0, 1, 2, 3, 5])
dict_keys(['a', 'b', 'c', 'd', 'e'])
dict_items([('a', 0), ('b', 1), ('c', 2), ('d', 3), ('e', 5)])
0 1 2 3 5 a b c d e ('a', 0) ('b', 1) ('c', 2) ('d', 3) ('e', 5)
```

```
>>> list=[0,1,2,3,4,5]
... for i in enumerate(list):
...     print(i)
...
(0, 0)
(1, 1)
(2, 2)
(3, 3)
(4, 4)
(5, 5)
```