算法分析和复杂性理论

第 11 次作业

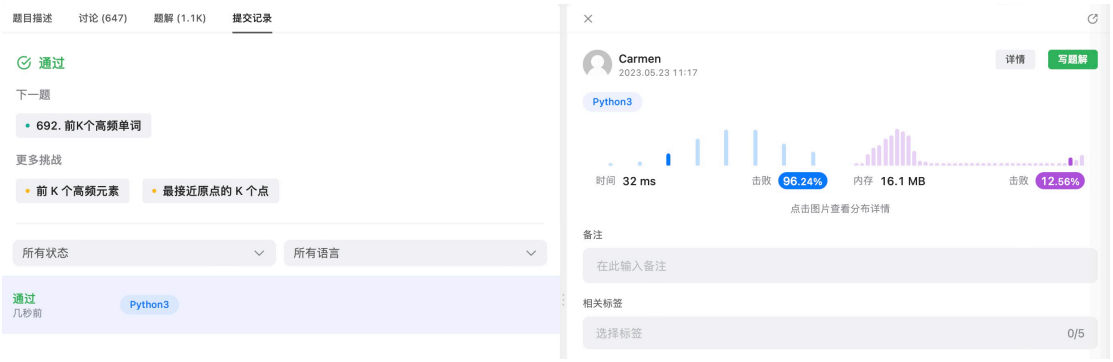
张瀚文 2201212865

1 前 K 个高频单词（692）

题目描述：给定一个单词列表 words 和一个整数 k ，返回前 k 个出现次数最多的单词。返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率， 按字典顺序排序。

解题思路：我们可以预处理出每一个单词出现的频率，然后依据每个单词出现的频率降序排序，最后返回前 k 个字符串即可。

运行截图：



测试代码：

```
class Solution:
    def topKFrequent(self, words: List[str], k: int) -> List[str]:
        # 注意：字典序是正序，出现次数为倒序
        return [w for w,_ in sorted(Counter(words).items(),key=lambda x:(-x[1],x[0]))[:k]]
```

2 K 站中转内最便宜的航班（787）

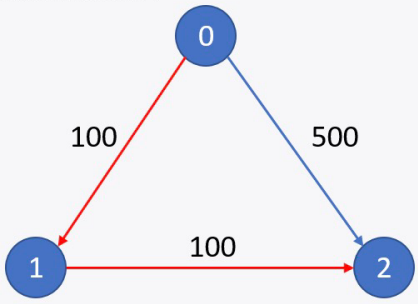
题目描述：

有 `n` 个城市通过一些航班连接。给你一个数组 `flights` ，其中 `flights[i] = [from_i, to_i, price_i]` ，表示该航班都从城市 `from_i` 开始，以价格 `price_i` 抵达 `to_i` 。

现在给定所有的城市和航班，以及出发城市 `src` 和目的地 `dst` ，你的任务是找到出一条最多经过 `k` 站中转的路线，使得从 `src` 到 `dst` 的 **价格最便宜** ，并返回该价格。 如果不存在这样的路线，则输出 `-1` 。

示例 1：

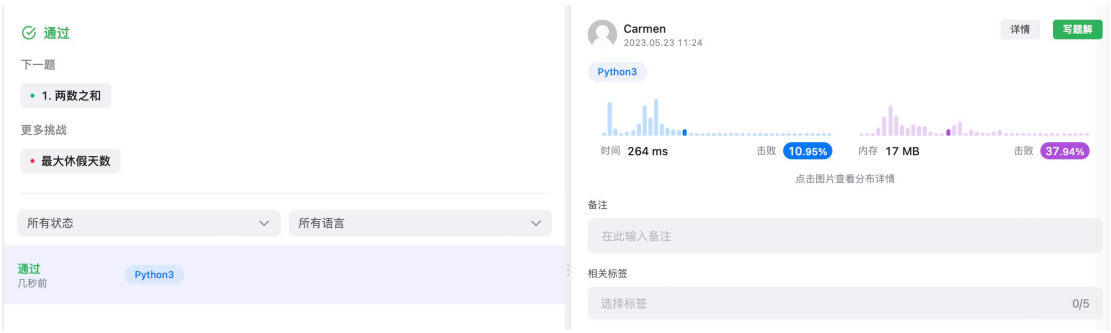
输入：
n = 3, edges = [[0,1,100],[1,2,100],[0,2,500]]
src = 0, dst = 2, k = 1
输出： 200
解释：
城市航班图如下



从城市 0 到城市 2 在 1 站中转以内的最便宜价格是 200，如图中红色所示。

解题思路： 用 f[t][i] 表示通过恰好 t 次航班，从出发城市 src 到达城市 i 需要的最小花费。在进行状态转移时，我们可以枚举最后一次航班的起点 j 。

运行截图：



测试代码：

```
class Solution:
    def findCheapestPrice(self, n: int, flights: List[List[int]],
src: int, dst: int, k: int) -> int:
        f = [[float("inf")] * n for _ in range(k + 2)]
```

```
        f[0][src] = 0
        for t in range(1, k + 2):
            for j, i, cost in flights:
                f[t][i] = min(f[t][i], f[t - 1][j] + cost)

        ans = min(f[t][dst] for t in range(1, k + 2))
        return -1 if ans == float("inf") else ans
```

3 最短的桥（934）

题目描述：

给你一个大小为 `n x n` 的二元矩阵 `grid`，其中 `1` 表示陆地，`0` 表示水域。

岛 是由四面相连的 `1` 形成的一个最大组，即不会与非组内的任何其他 `1` 相连。`grid` 中 **恰好存在两座岛** 。

你可以将任意数量的 `0` 变为 `1`，以使两座岛连接起来，变成 **一座岛** 。

返回必须翻转的 `0` 的最小数目。

示例 1：

输入： grid = [[0,1],[1,0]]
输出： 1

解题思路： 题目中求最少的翻转 0 的数目等价于求矩阵中两个岛的最短距离，因此我们可以广度优先搜索来找到矩阵中两个块的最短距离。首先找到其中一座岛，然后将其不断向外延伸一圈，直到到达了另一座岛，延伸的圈数即为最短距离。广度优先搜索时，我们可以将已经遍历过的位置标记为 −1 。

运行截图：

测试代码：

```
class Solution:
    def shortestBridge(self, grid: List[List[int]]) -> int:
```

```
        n = len(grid)
        for i, row in enumerate(grid):
            for j, v in enumerate(row):
                if v != 1:
                    continue
                island = []
                grid[i][j] = -1
                q = deque([(i, j)])
                while q:
                    x, y = q.popleft()
                    island.append((x, y))
                    for nx, ny in (x + 1, y), (x - 1, y), (x, y + 1),
(x, y - 1):
                        if 0 <= nx < n and 0 <= ny < n and grid[nx][ny]
== 1:
                            grid[nx][ny] = -1
                            q.append((nx, ny))

                step = 0
                q = island
                while True:
                    tmp = q
                    q = []
                    for x, y in tmp:
                        for nx, ny in (x + 1, y), (x - 1, y), (x, y + 1),
(x, y - 1):
                            if 0 <= nx < n and 0 <= ny < n:
                                if grid[nx][ny] == 1:
                                    return step
                                if grid[nx][ny] == 0:
                                    grid[nx][ny] = -1
                                    q.append((nx, ny))
                    step += 1
```