1. 题目

M909.蛇梯棋

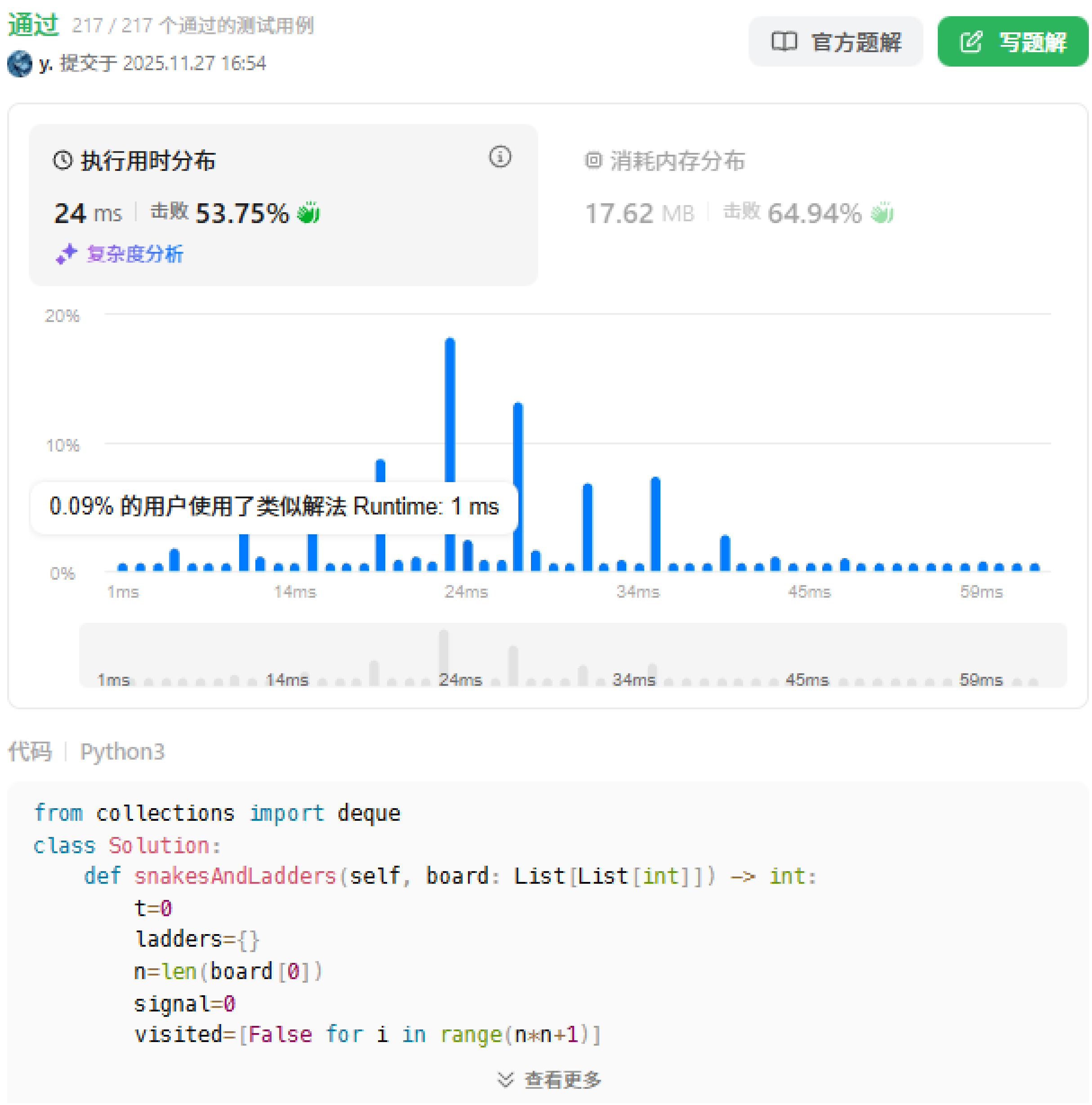
bfs, https://leetcode.cn/problems/snakes-and-ladders/

思路： bfs求最短路径，这里看似是一张图其实由于单调性直接展开到一个列上就可以计算

代码：

```
from collections import deque
class Solution:
    def snakesAndLadders(self, board: List[List[int]]) -> int:
        t=0
        ladders={}
        n=len(board[0])
        signal=0
        visited=[False for i in range(n*n+1)]
        visited[1]=True
        for i in range(n-1,-1,-1):
            if signal==0:
                for j in range(0,n):
                    t+=1
                    if board[i][j]!=-1:
                        ladders[t]=board[i][j]
                signal=1
            else:
                for j in range(n-1,-1,-1):
                    t+=1
                    if board[i][j]!=-1:
                        ladders[t]=board[i][j]
                signal=0
        q=deque()
        q.append([(1,0)])
        while q[-1]!=[]:
            l=[]
            for loc,_ in q[-1]:
                for i in range(1,7):
                    next_loc=loc+i
                    if next_loc in ladders:
                        next_loc=ladders[next_loc]
                    if next_loc==n*n:
                        return _+1
                    if visited[next_loc]==False:
                        l.append((next_loc,_+1))
                    visited[next_loc]=True
            q.append(l)
        return -1
```

代码运行截图（至少包含有"Accepted"）



通过 217 / 217 个通过的测试用例

y. 提交于 2025.11.27 16:54

执行用时分布

24 ms | 击败 53.75%

消耗内存分布

17.62 MB | 击败 64.94%

0.09% 的用户使用了类似解法 Runtime: 1 ms

代码 | Python3

```
from collections import deque
class Solution:
    def snakesAndLadders(self, board: List[List[int]]) -> int:
        t=0
        ladders={}
        n=len(board[0])
        signal=0
        visited=[False for i in range(n*n+1)]
```

sy382: 有向图判环 中等

dfs, topological sort, https://sunnywhy.com/sfbj/10/3/382

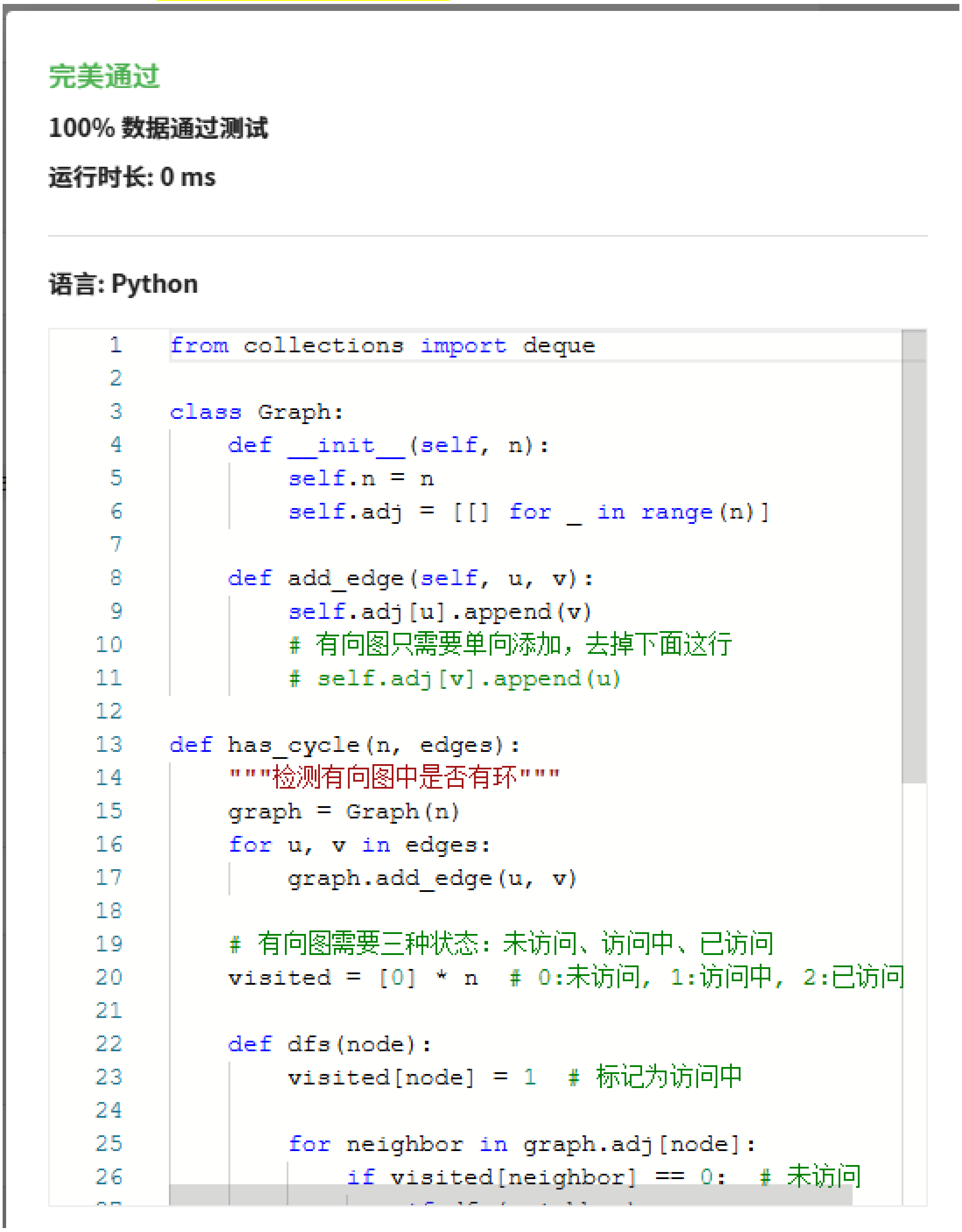
思路： 建图，使用dfs搜索满足要求的环，注意记忆化已经过的点

代码：

```
from collections import deque

class Graph:
    def __init__(self, n):
        self.n = n
        self.adj = [[] for _ in range(n)]

    def add_edge(self, u, v):
        self.adj[u].append(v)
        # 有向图只需要单向添加，去掉下面这行
```

```python
        # self.adj[v].append(u)

def has_cycle(n, edges):
    """检测有向图中是否有环"""
    graph = Graph(n)
    for u, v in edges:
        graph.add_edge(u, v)

    # 有向图需要三种状态：未访问、访问中、已访问
    visited = [0] * n  # 0:未访问, 1:访问中, 2:已访问

    def dfs(node):
        visited[node] = 1  # 标记为访问中

        for neighbor in graph.adj[node]:
            if visited[neighbor] == 0:  # 未访问
                if dfs(neighbor):
                    return True
            elif visited[neighbor] == 1:  # 遇到访问中的节点，说明有环
                return True

        visited[node] = 2  # 标记为已访问
        return False

    for i in range(n):
        if visited[i] == 0:  # 只对未访问的节点开始DFS
            if dfs(i):
                return True
    return False

n, m = map(int, input().split())
edges = []
for _ in range(m):
    u, v = map(int, input().split())
    edges.append((u, v))

if has_cycle(n, edges):
    print('Yes')
else:
    print('No')
```

代码运行截图（至少包含有"Accepted"）



完美通过

100% 数据通过测试

运行时长: 0 ms

语言: Python

```python
from collections import deque

class Graph:
    def __init__(self, n):
        self.n = n
        self.adj = [[] for _ in range(n)]

    def add_edge(self, u, v):
        self.adj[u].append(v)
        # 有向图只需要单向添加，去掉下面这行
        # self.adj[v].append(u)

def has_cycle(n, edges):
    """检测有向图中是否有环"""
    graph = Graph(n)
    for u, v in edges:
        graph.add_edge(u, v)

    # 有向图需要三种状态：未访问、访问中、已访问
    visited = [0] * n  # 0:未访问, 1:访问中, 2:已访问

    def dfs(node):
        visited[node] = 1  # 标记为访问中

        for neighbor in graph.adj[node]:
            if visited[neighbor] == 0:  # 未访问
```

M28046: 词梯

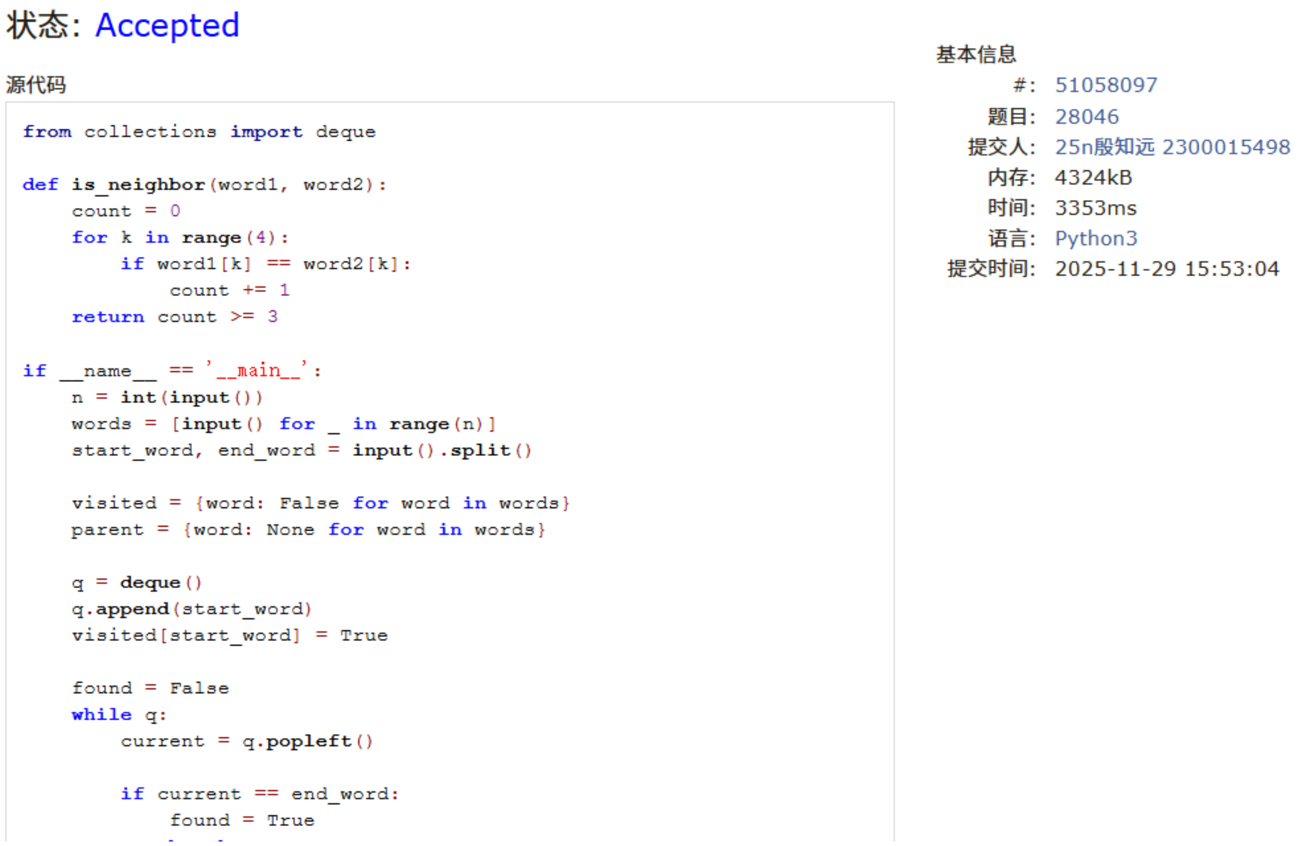
bfs, http://cs101.openjudge.cn/practice/28046/

思路： 其实不一定要完整建图，只需建立边的关系，然后就是一个简单的路径搜索问题，查找最短路径则使用bfs

代码：

```python
from collections import deque

def is_neighbor(word1, word2):
    count = 0
    for k in range(4):
        if word1[k] == word2[k]:
            count += 1
    return count >= 3

if __name__ == '__main__':
    n = int(input())
    words = [input() for _ in range(n)]
    start_word, end_word = input().split()

    visited = {word: False for word in words}
    parent = {word: None for word in words}

    q = deque()
    q.append(start_word)
    visited[start_word] = True

    found = False
    while q:
        current = q.popleft()

        if current == end_word:
            found = True
            break

        for word in words:
            if not visited[word] and is_neighbor(current, word):
                visited[word] = True
                parent[word] = current
                q.append(word)

    if found:
        # 重建路径
        path = []
        node = end_word
        while node:
            path.append(node)
            node = parent[node]
        path.reverse()
        print(' '.join(path))
    else:
        print('NO')
```

代码运行截图 （至少包含有"Accepted"）



状态: Accepted

源代码

```
from collections import deque

def is_neighbor(word1, word2):
    count = 0
    for k in range(4):
        if word1[k] == word2[k]:
            count += 1
    return count >= 3

if __name__ == '__main__':
    n = int(input())
    words = [input() for _ in range(n)]
    start_word, end_word = input().split()

    visited = {word: False for word in words}
    parent = {word: None for word in words}

    q = deque()
    q.append(start_word)
    visited[start_word] = True

    found = False
    while q:
        current = q.popleft()

        if current == end_word:
            found = True
```

基本信息

#: 51058097
题目: 28046
提交人: 25n殷知远 2300015498
内存: 4324kB
时间: 3353ms
语言: Python3
提交时间: 2025-11-29 15:53:04

M433.最小基因变化

bfs, https://leetcode.cn/problems/minimum-genetic-mutation/description/

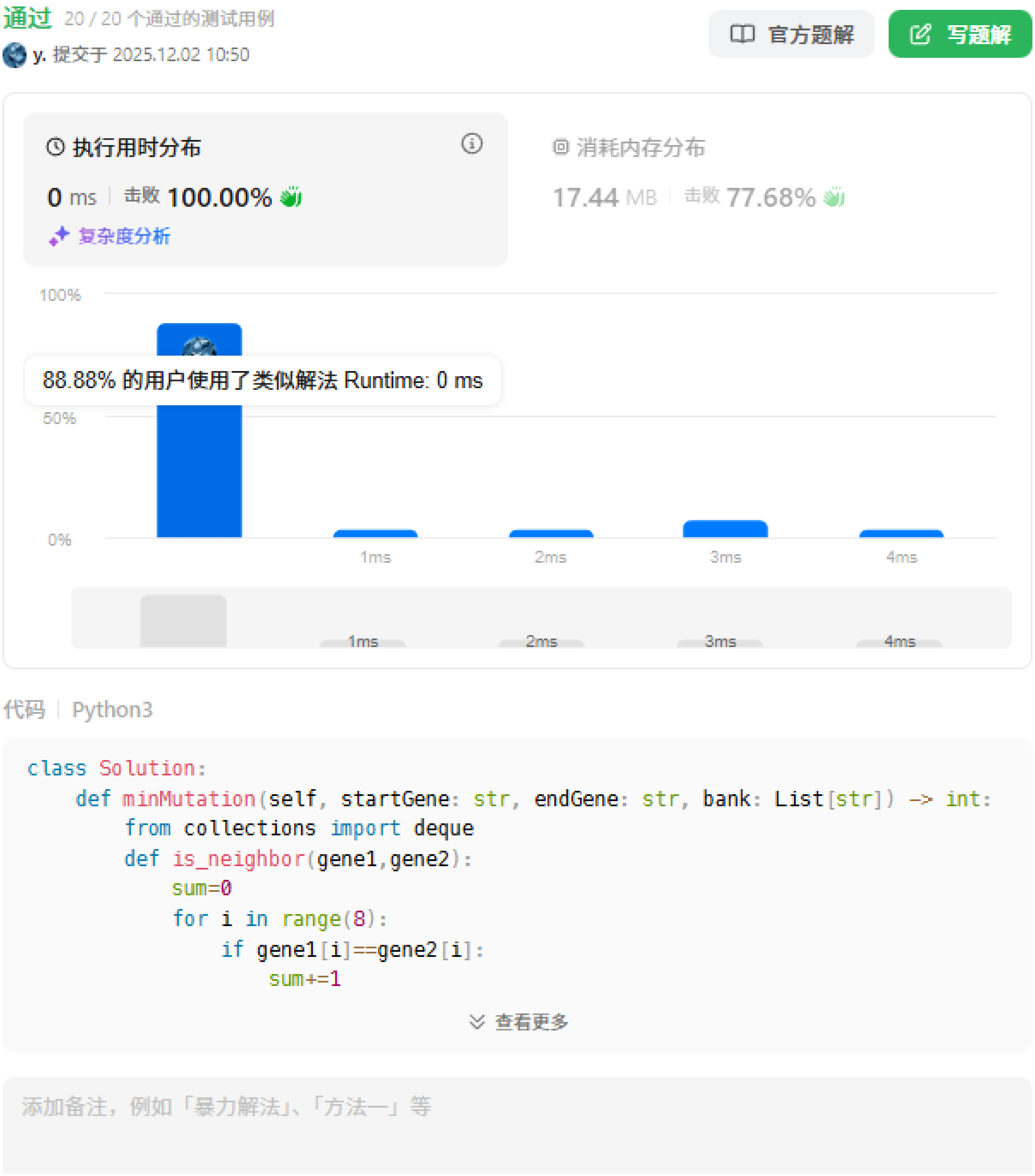
思路：

类似的题目也是，用相邻关系建图，再用bfs找最短路径

代码

```
class Solution:
    def minMutation(self, startGene: str, endGene: str, bank: List[str]) -> int:
        from collections import deque
        def is_neighbor(gene1,gene2):
            sum=0
            for i in range(8):
                if gene1[i]==gene2[i]:
                    sum+=1
            if sum==7:
                return True
            else:
                return False
        q=deque()
        q.append((startGene,0))
        n=len(bank)
        visited=[False for i in range(n)]
        while q:
            x=q.popleft()
            gene,t=x[0],x[1]
            if gene==endGene:
                return t
            for i in range(n):
                if visited[i]==False and is_neighbor(bank[i],gene):
                    q.append((bank[i],t+1))
                    visited[i]=True
        return -1
```

代码运行截图（至少包含有"Accepted"）



通过 20 / 20 个通过的测试用例

y. 提交于 2025.12.02 10:50

执行用时分布

0 ms | 击败 100.00%

消耗内存分布

17.44 MB | 击败 77.68%

88.88% 的用户使用了类似解法 Runtime: 0 ms

代码 | Python3

```
class Solution:
    def minMutation(self, startGene: str, endGene: str, bank: List[str]) -> int:
        from collections import deque
        def is_neighbor(gene1,gene2):
            sum=0
            for i in range(8):
                if gene1[i]==gene2[i]:
                    sum+=1
```

添加备注，例如「暴力解法」、「方法一」等

M05443: 兔子与樱花

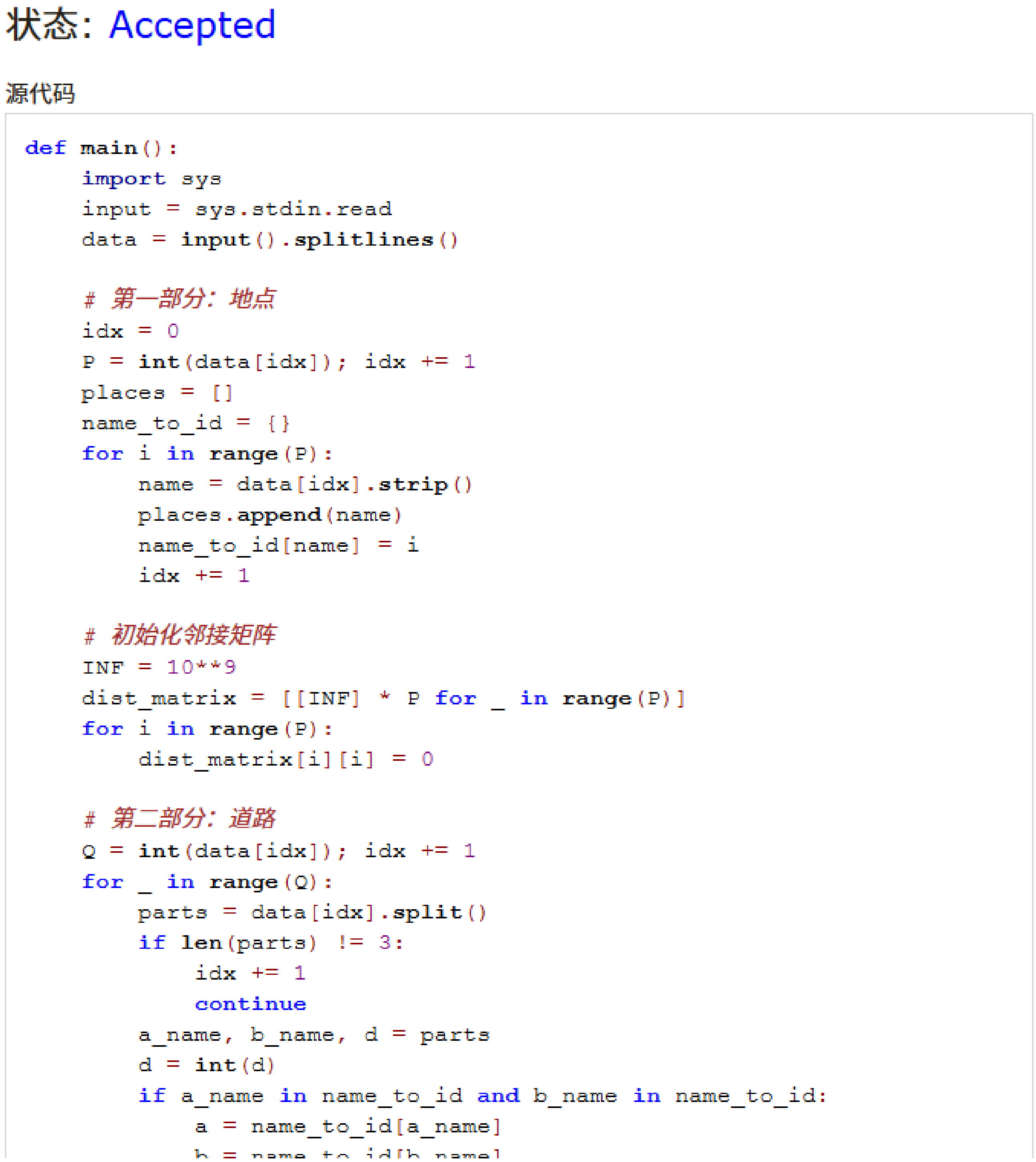
Dijkstra, http://cs101.openjudge.cn/practice/05443/

思路： Dijkstra算法查找最短加权路径，用邻接矩阵记录图，每次都用邻居的已知最短路径更新其与当前节点的路径

代码

```
def main():
    import sys
    input = sys.stdin.read
    data = input().splitlines()

    # 第一部分：地点
    idx = 0
    P = int(data[idx]); idx += 1
    places = []
    name_to_id = {}
    for i in range(P):
        name = data[idx].strip()
        places.append(name)
        name_to_id[name] = i
        idx += 1

    # 初始化邻接矩阵
    INF = 10**9
    dist_matrix = [[INF] * P for _ in range(P)]
    for i in range(P):
        dist_matrix[i][i] = 0

    # 第二部分：道路
    Q = int(data[idx]); idx += 1
    for _ in range(Q):
        parts = data[idx].split()
        if len(parts) != 3:
            idx += 1
            continue
        a_name, b_name, d = parts
        d = int(d)
        if a_name in name_to_id and b_name in name_to_id:
            a = name_to_id[a_name]
            b = name_to_id[b_name]
            # 使用最小值，以防有重复边
            if d < dist_matrix[a][b]:
                dist_matrix[a][b] = d
                dist_matrix[b][a] = d
        idx += 1

    # 第三部分：查询
    R = int(data[idx]); idx += 1
    queries = []
    for _ in range(R):
        parts = data[idx].split()
        if len(parts) == 2:
            start_name, end_name = parts
            queries.append((start_name, end_name))
        idx += 1

    # Dijkstra 函数，返回路径节点ID列表和边权重列表
    def dijkstra(start_id, end_id):
        # 如果起点和终点相同
        if start_id == end_id:
            return [start_id], []

        dist = [INF] * P
        prev = [-1] * P
        edge_weight = [0] * P  # 从前驱节点到这个节点的边的权重
        visited = [False] * P

        dist[start_id] = 0

        for _ in range(P):
            # 选择未访问的最小距离节点
            u = -1
            min_d = INF
            for i in range(P):
                if not visited[i] and dist[i] < min_d:
                    min_d = dist[i]
                    u = i

            # 如果没有找到可达节点
            if u == -1:
                break

            visited[u] = True

            # 如果找到了终点，可以提前终止（已标记visited，可以继续）
            if u == end_id:
                # 继续完成当前节点的邻居更新，确保所有可能路径都被考虑
                pass

            # 更新邻居节点
            for v in range(P):
                w = dist_matrix[u][v]
                if w < INF and not visited[v]:
                    if dist[u] + w < dist[v]:
                        dist[v] = dist[u] + w
                        prev[v] = u
                        edge_weight[v] = w

        # 检查是否可达
        if dist[end_id] >= INF:
            return None, None

        # 回溯路径
        path_nodes = []
        weights = []
        cur = end_id
        while cur != -1:
            path_nodes.append(cur)
            if prev[cur] != -1:
                weights.append(edge_weight[cur])
            cur = prev[cur]

        # 反转得到从起点到终点的顺序
        path_nodes.reverse()
        weights.reverse()

        return path_nodes, weights

    results = []
    for start_name, end_name in queries:
        sid = name_to_id.get(start_name)
        eid = name_to_id.get(end_name)

        if sid is None or eid is None:
            results.append("")  # 地点不存在的情况
            continue

        nodes, weights = dijkstra(sid, eid)

        if nodes is None:
            # 不可达
            results.append("")
            continue

        # 构造输出
        if len(nodes) == 1:
            # 起点终点相同
            results.append(places[nodes[0]])
        else:
            parts = [places[nodes[0]]]  # 起点
            for i in range(len(weights)):
                parts.append(f"->({weights[i]})->{places[nodes[i+1]]}")
            results.append("".join(parts))

    # 输出结果
    sys.stdout.write("\n".join(results))


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

代码运行截图（至少包含有"Accepted"）



状态: Accepted

源代码

```
def main():
    import sys
    input = sys.stdin.read
    data = input().splitlines()

    # 第一部分: 地点
    idx = 0
    P = int(data[idx]); idx += 1
    places = []
    name_to_id = {}
    for i in range(P):
        name = data[idx].strip()
        places.append(name)
        name_to_id[name] = i
        idx += 1

    # 初始化邻接矩阵
    INF = 10**9
    dist_matrix = [[INF] * P for _ in range(P)]
    for i in range(P):
        dist_matrix[i][i] = 0

    # 第二部分: 道路
    Q = int(data[idx]); idx += 1
    for _ in range(Q):
        parts = data[idx].split()
        if len(parts) != 3:
            idx += 1
            continue
        a_name, b_name, d = parts
        d = int(d)
        if a_name in name_to_id and b_name in name_to_id:
            a = name_to_id[a_name]
            b = name_to_id[b_name]
```

M28050: 骑士周游

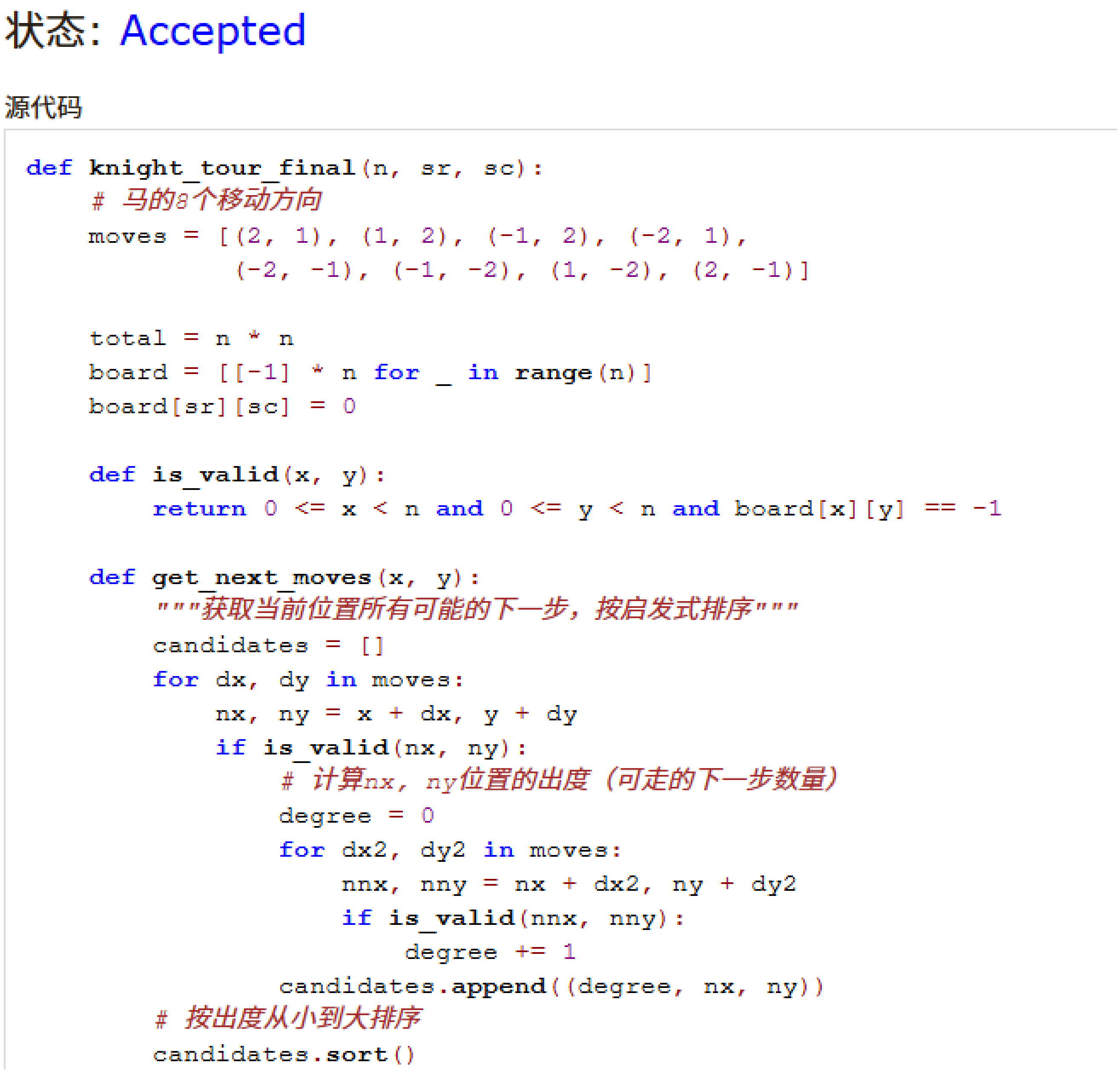
dfs, http://cs101.openjudge.cn/practice/28050/

思路： 也是经典的搜索问题，这里的关键在于启发式排序，优先选择下一步可走格子最少的路径

代码：

```
def knight_tour_final(n, sr, sc):
    # 马的8个移动方向
    moves = [(2, 1), (1, 2), (-1, 2), (-2, 1),
             (-2, -1), (-1, -2), (1, -2), (2, -1)]
    total = n * n
    board = [[-1] * n for _ in range(n)]
    board[sr][sc] = 0

    def is_valid(x, y):
        return 0 <= x < n and 0 <= y < n and board[x][y] == -1

    def get_next_moves(x, y):
        """获取当前位置所有可能的下一步，按启发式排序"""
        candidates = []
        for dx, dy in moves:
            nx, ny = x + dx, y + dy
            if is_valid(nx, ny):
                # 计算nx, ny位置的出度（可走的下一步数量）
                degree = 0
                for dx2, dy2 in moves:
                    nnx, nny = nx + dx2, ny + dy2
                    if is_valid(nnx, nny):
                        degree += 1
                candidates.append((degree, nx, ny))
        # 按出度从小到大排序
        candidates.sort()
        return candidates

    def dfs(x, y, step):
        if step == total - 1:
            return True

        for _, nx, ny in get_next_moves(x, y):
            board[nx][ny] = step + 1
            if dfs(nx, ny, step + 1):
                return True
            board[nx][ny] = -1

        return False

    return 'success' if dfs(sr, sc, 0) else 'fail'

# 读取输入并输出结果
n = int(input())
sr, sc = map(int, input().split())
print(knight_tour_final(n, sr, sc))
```

代码运行截图 （至少包含有"Accepted"）



状态: Accepted

源代码

```
def knight_tour_final(n, sr, sc):
    # 马的8个移动方向
    moves = [(2, 1), (1, 2), (-1, 2), (-2, 1),
             (-2, -1), (-1, -2), (1, -2), (2, -1)]

    total = n * n
    board = [[-1] * n for _ in range(n)]
    board[sr][sc] = 0

    def is_valid(x, y):
        return 0 <= x < n and 0 <= y < n and board[x][y] == -1

    def get_next_moves(x, y):
        """获取当前位置所有可能的下一步，按启发式排序"""
        candidates = []
        for dx, dy in moves:
            nx, ny = x + dx, y + dy
            if is_valid(nx, ny):
                # 计算nx, ny位置的出度（可走的下一步数量）
                degree = 0
                for dx2, dy2 in moves:
                    nnx, nny = nx + dx2, ny + dy2
                    if is_valid(nnx, nny):
                        degree += 1
                candidates.append((degree, nx, ny))
        # 按出度从小到大排序
        candidates.sort()
```

2. 学习总结和个人收获

如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算2025fall每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。

这次作业难度适中，很好的巩固练习了dfs和bfs在图论上的使用，也介绍了一些关于最短路搜索的变式，启发了一些新的思考。 这周针对练习了图论内容，对如何建图，书写基础类方法， 以及类方法中一些常用的函数均有运用，对图的理解加深理解了其作为模型刻画的一些好处。