Assignment #C: 图 (2/4)

Updated 2329 GMT+8 Nov 24, 2025

2025 fall, Complied by 杨浩、化院

1. 1. 题目

1.1 M909.蛇梯棋

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

思路:

- bfs,遍历next时添加有蛇或梯的末端以及正常移动的最远点。
- 判定终点：1.正常到达2.蛇或梯的终点

代码:

```
class Solution:
    def snakesAndLadders(self, board: List[List[int]]) ->
int:
        def where(num,n):
            x=num//n
            y=num%n
            if x%2==1:
                y=n-y-1
            return x,y
        board = board[::-1]
        n = len(board)
        finished = [False] * (n*n)
        deq = deque([[0,0]])
        while deq:
            curr,cnt = deq.popleft()
            if n*n-1-curr<=6:
                return cnt+1
            if finished[curr]:
                continue
            finished[curr] = True
            maxi=None
            for delta in range(1,7):
                x,y = where(curr+delta,n)
                if board[x][y]!=-1:
                    if board[x][y]==n*n:
                        return cnt+1
                    deq.append([board[x][y]-1,cnt+1])
                    continue
                else:
                    maxi=delta
            if maxi:
```

```
                    deq.append([curr+maxi,cnt+1])
        return -1
```

Fence 1

代码运行截图 （至少包含有"Accepted"）

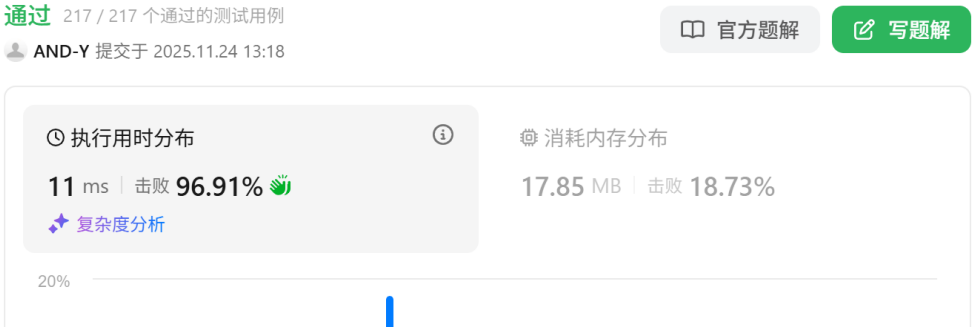


Figure 1

1.2 sy382: 有向图判环 中等

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

思路：

- dfs染色法

代码：

```
from collections import defaultdict
def dfs(i):
    if color_list[i]==1:
        return False
    if color_list[i]==2:
        return True
    color_list[i]=1
    for j in dic[i]:
        if not dfs(j):
            return False
    color_list[i]=2
    return True

n,m=map(int,input().split())
dic=defaultdict(list)
for i in range(m):
    u,v=map(int,input().split())
    dic.setdefault(u,[]).append(v)
color_list=[0 for _ in range(n)]
for i in range(n):
    if color_list[i]==2:
        continue
```

```
    else:
        if not dfs(i):
            print('Yes')
            exit()
print('No')
```

Fence 2

代码运行截图 （至少包含有"Accepted"）

```
    dic.setdefault(u,[]).append(v)
color_list=[0 for _ in range(n)]
for i in range(n):
    if color_list[i]==2:
```

测试输入 提交结果 历史提交

完美通过 查看题解

100% 数据通过测试 详情

运行时长: 0 ms

Figure 2

1.3 M28046: 词梯

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

思路：

- 构造桶代替邻接表（理论上一个单词至多有`25**4`个邻居，但显然不会有这么多）来描述图
- 由于题目保证了最短路径若存在必唯一，直接对搜索完成的桶进行标记来剪枝，大幅降低搜索次数。
- 如果没有这个保证，且要求求出所有最短路径，数据量还足够大，如LeetCode的126. 单词接龙 II，题目会变得异常困难且恶心。

代码：

```
import sys
from collections import deque
from collections import defaultdict
def build_buckets(word):
    for j in range(len(word)):
        bucket = f'{word[:j]}_{word[j+1:]}'
        buckets.setdefault(bucket, set()).add(word)

```

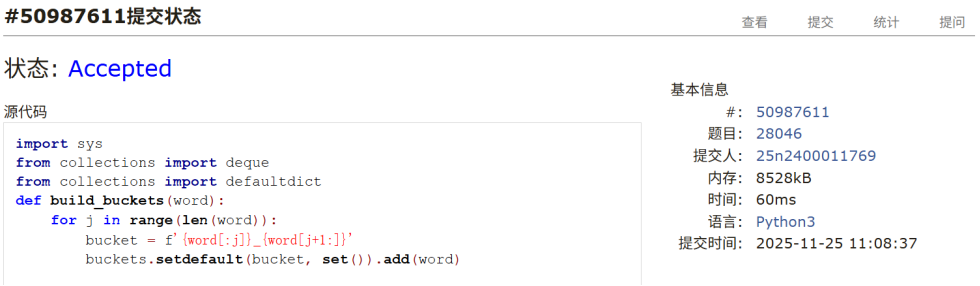
```python


data=sys.stdin.read()
data=list(data.split())
index=0
n=int(data[index])
index+=1
buckets=defaultdict(set)
for i in range(n):
    word=data[index]
    index+=1
    build_buckets(word)
del_list=[]
for bucket in buckets:
    if len(buckets[bucket]) == 1:
        del_list.append(bucket)
for bucket in del_list:
    buckets.pop(bucket)
begin,end=data[index],data[index+1]
finish_set=set()
queue=deque([[begin]])
while queue:
    path=queue.popleft()
    word=path[-1]
    for i in range(len(word)):
        bucket=f'{word[:i]}_{word[i+1:]}'
        if bucket not in finish_set:
            for new in buckets[bucket]:
                if new == end:
                    path.append(new)
                    print(*path)
                    exit()
                if new != word:
                    queue.append(path+[new])

            finish_set.add(bucket)
print('NO')
```

Fence 3

代码运行截图 （至少包含有"Accepted"）



#50987611提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys
from collections import deque
from collections import defaultdict
def build_buckets(word):
    for j in range(len(word)):
        bucket = f'{word[:j]}_{word[j+1:]}'
        buckets.setdefault(bucket, set()).add(word)
```

基本信息

#: 50987611
题目: 28046
提交人: 25n2400011769
内存: 8528kB
时间: 60ms
语言: Python3
提交时间: 2025-11-25 11:08:37

Figure 3

1.4 M433.最小基因变化

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

思路:

- 策略与前一题词梯一致

代码

```
from typing import List
from collections import defaultdict
from collections import deque
class Solution:
    def minMutation(self, startGene: str, endGene: str, bank: List[str]) -> int:
        def build_buckets(word):
            for i in range(8):
                bucket = f'{word[:i]}_{word[i+1:]}'
                buckets.setdefault(bucket, []).append(word)
        buckets = defaultdict(list)
        build_buckets(startGene)
        for word in bank:
            build_buckets(word)
        graph=defaultdict(list)
        for bucket in buckets:
            if len(buckets[bucket]) == 1:
                continue
            else:
                alist=buckets[bucket]
                for i in range(len(alist)):
                    for j in range(i+1, len(alist)):
                        graph.setdefault(alist[i], []).append(alist[j])
                        graph.setdefault(alist[j], []).append(alist[i])
        queue = deque([[startGene,0]])
        visited = set()
        while queue:
            pr,cnt = queue.popleft()
            for new_pr in graph[pr]:
                if new_pr == endGene:
                    return cnt+1
                if new_pr not in visited:
                    queue.append([new_pr,cnt+1])
            visited.add(pr)
        return -1
```

Fence 4

代码运行截图（至少包含有"Accepted"）

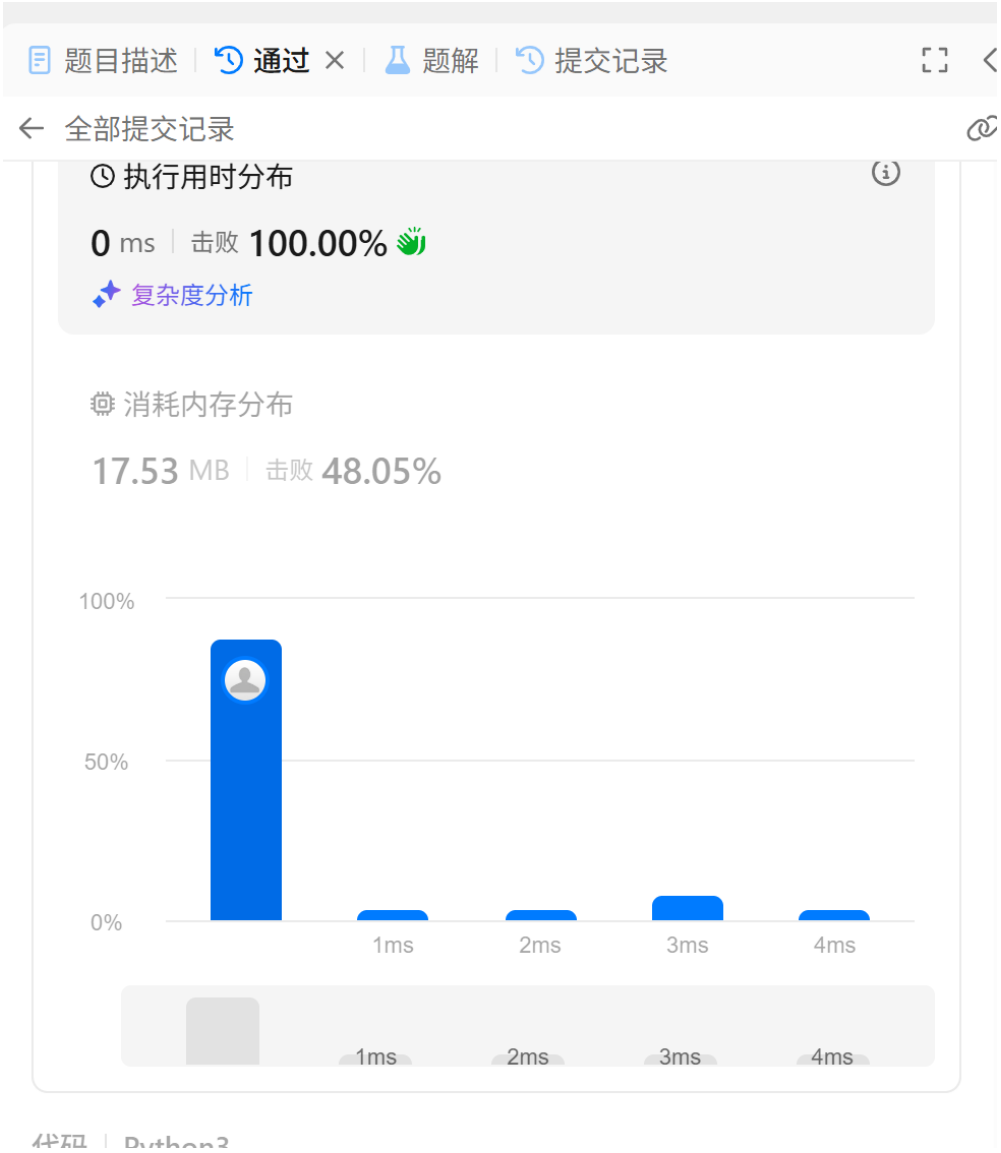


Figure 4

1.5 M05443: 兔子与樱花

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

思路:

- Dijkstra可以解决此类查询数量较少，边的数量较少的题目
- Floyd-Warshall解决查询数量多，顶点数量较少的题目
- 此题背景下两者时间复杂度和空间复杂度差不多，Dijkstra写起来要轻松一些（也有可能我Floyd-Warshall写的太少了）

代码

```
from collections import defaultdict
import heapq
```

```
import itertools

class Graph:
    def __init__(self):
        self.check_dict = None
        self.graph=defaultdict(dict)

    def add_edge(self,u,v,w):
        self.graph[u][v]=w
        self.graph[v][u]=w

    def dijkstra(self,start,end):
        heap=[]
        cnt=itertools.count()
        heapq.heappush(heap,(0,next(cnt),[start]))
        path_dic={start:0}
        finished=set()
        while heap:
            distance,count,path=heapq.heappop(heap)
            u=path[-1]
            if u in finished:
                continue
            if u==end:
                break
            finished.add(u)
            for v,w in self.graph[u].items():
                if v in finished:
                    continue
                if v in path_dic:
                    new_distance=distance+w
                    if new_distance < path_dic[v]:
                        path_dic[v]=new_distance
                        heapq.heappush(heap,
(new_distance,next(cnt),path+[v]))
                else:
                    path_dic[v]=distance+w
                    heapq.heappush(heap,
(distance+w,next(cnt),path+[v]))
        res=''
        for i in range(len(path)-1):
            res+=path[i]+f'->({self.graph[path[i]]
[path[i+1]]})->'
        res+=path[-1]
        return res

    def floyd_warshall(self):
        cnt=itertools.count()
        check_dict={}
        for pr in self.graph:
            check_dict[pr]=next(cnt)
            check_dict[check_dict[pr]]=pr
```

```
        self.check_dict=check_dict
        n=len(check_dict)//2
        dist = [[float('inf') for _ in range(n)] for _ in range(n)]
        next_node = [[-1]*n for _ in range(n)]

        for i in range(n):
            for j in range(n):
                if i == j:
                    dist[i][j] = 0
                    next_node[i][j] = j
                elif check_dict[j] in self.graph[check_dict[i]]:
                    dist[i][j] = self.graph[check_dict[i]][check_dict[j]]
                    next_node[i][j] = j

            # Floyd-Warshall算法核心
        for k in range(n):
            for i in range(n):
                for j in range(n):
                    if dist[i][k] + dist[k][j] < dist[i][j]:
                        dist[i][j] = dist[i][k] + dist[k][j]
                        next_node[i][j] = next_node[i][k]

        return dist, next_node

    def construct_shortest_path(self, start, end, next_node):

 start,end=self.check_dict[start],self.check_dict[end]
        if next_node[start][end] == -1:
            return []

        path = [self.check_dict[start]]
        current = start

        while current != end:
            current = next_node[current][end]
            path.append(self.check_dict[current])

        res=''
        for i in range(len(path)-1):

            res += path[i] + f'->({self.graph[path[i]][path[i + 1]]})->'
        res += path[-1]
        return res

```

```
94
95  if __name__=='__main__':
96      graph=Graph()
97      p=int(input())
98      name=[]
99      for i in range(p):
100         name.append(input())
101     q=int(input())
102     for i in range(q):
103         u,v,w=input().split()
104         graph.add_edge(u,v,int(w))
105     r=int(input())
106     #Dijkstra算法
107     for i in range(r):
108         start,end=input().split()
109         print(graph.dijkstra(start,end))
110
111     """Floyd算法
112     dist,next_node=graph.floyd_warshall()
113     for i in range(r):
114         start,end=input().split()
115
    print(graph.construct_shortest_path(start,end,next_node))
116     """
```

Fence 5

代码运行截图（至少包含有"Accepted"）

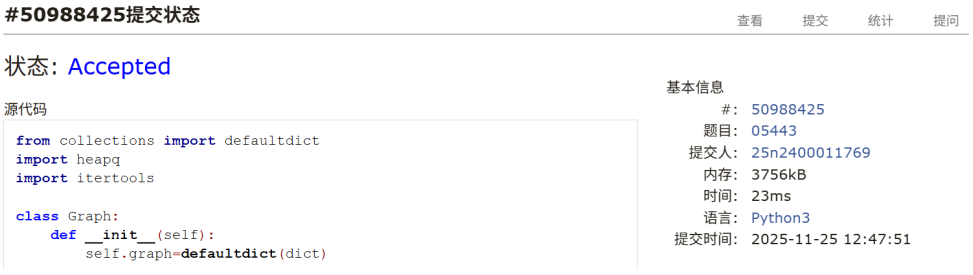


Figure 5

1.6 M28050: 骑士周游

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

思路：

- Warnsdorff 算法,优先访问合理走法最少的顶点，即先把最难到达的地方（边角）走了，好走的地方用于相隔较远的地方的迁移。
- 如果没有合格的周游，代码运行时间会相当的长，例如`n=7,sr=0,sc=1`。

代码：

```python
def dfs(x,y,n,cnt,visited_matrix):
    if cnt == n*n-1:
        return True
    for num,new_x,new_y in build_target(x,y,n,visited_matrix):
        visited_matrix[new_x][new_y] = True
        if dfs(new_x,new_y,n,cnt + 1,visited_matrix):
            return True
        visited_matrix[new_x][new_y] = False
    return False

def build_target(x,y,n,visited_matrix):
    target_list = []
    for dx,dy in delta:
        if 0<=x+dx<n and 0<=y+dy<n and not visited_matrix[x+dx][y+dy]:
            num=0
            for di,dj in delta:
                if 0<=x+dx+di<n and 0<=y+dy+dj<n and not visited_matrix[x+dx+di][y+dy+dj]:
                    num+=1
            target_list.append([num,x+dx,y+dy])
    target_list.sort()
    return target_list

n=int(input())
start_x,start_y=map(int,input().split())
visited_matrix=[[False]*n for _ in range(n)]
visited_matrix[start_x][start_y] = True
delta=[(1,2),(2,1),(-1,2),(-2,1),(1,-2),(2,-1),(-1,-2),(-2,-1)]
if dfs(start_x,start_y,n,0,visited_matrix):
    print("success")
else:
    print("fail")
```

Fence 6

代码运行截图 （至少包含有"Accepted"）

Figure 6

2. 2. 学习总结和个人收获

本周涉及了很多最短路径的题目，无权图使用BFS，有权图使用Dijkstra或Floyd-Warshall即可。BFS的题目有的可以用A *算法优化（如上周的E07218: 献给阿尔吉侬的花束），不过很多题目时间卡的不严格，并且A *算法应用范围有限，有的题目写不出合适的启发函数（例如本周的词梯）；Dijkstra可以用数组也可用堆实现，绝大分情况堆是更优的，但当图足够稠密时，数组更优。这部分题目有一定的模板性，但往往都有一些小坑，限时做仍有一定难度。