入选理由

1. 正向搜索出了好几道了，那么反向搜索你会么？有时候反向思考，有意向不到的收获。

标签

- BFS
- 反向搜索

难度

- 中等

题目地址（Shortest-Cycle-Containing-Target-Node）

https://binarysearch.com/problems/Shortest-Cycle-Containing-Target-Node

题目描述

You are given a two-dimensional list of integers graph representing a directed graph as an adjacency list. You are also given an integer target.

Return the length of a shortest cycle that contains target. If a solution does not exist, return -1.

Constraints

n, m ≤ 250 where n and m are the number of rows and columns in graph

前置知识

- BFS

思路

题目大意是给你一个邻接矩阵表示的图，让你返回**最短**的环，如果没有环则返回 -1。

回想一下，我们遍历图的时候是如何防止环的产生的？通常是使用一个集合 visited，记录已经访问过的节点。每次遇到一个新的节点，我们都检查其是否在 visited 中存在。如果存在，我们就不再进行处理（跳过），这样就避免了环的影响。

因此检测环的存在也是一样的思路。我们也可使用一个集合记录访问过的节点。如果再次访问到了已经访问过的节点，那么说明有环。

同时为了记录**最短**的环，我们不妨进行 BFS，这样当我们遇到一个环的时候，就一定最短的，直接返回 BFS 遍历的层即可，这提示我们使用带层信息的 BFS。而如果使用 dfs，在极端情况下性能会很差。

而题目要求的是返回的最短的 **包含 target**的环。与其从各个点作为搜索起点，并在检测到环的时候再判断环中是否有 target，我们不妨**从 target 开始搜索，这样检测到环就不用判断环中是否有 target 了**，这是一个小技巧。 即从题目的 target 开始反向搜索，而不是从起点找到 target。

关键点

- 反向搜索，即从 target 开始搜索。而不是从图中所有的节点开始搜。

代码

下面代码使用的就是一个标准带层的 BFS 模板。

代码支持 Python3:

```
class Solution:
    def solve(self, graph, target):
        q = collections.deque([target])
        visited = set()
        steps = 0
        while q:
            for i in range(len(q)):
                cur = q.popleft()
                visited.add(cur)
                for neighbor in graph[cur]:
                    if neighbor not in visited:
                        q.append(neighbor)
                    elif neighbor == target:
                        return steps + 1
            steps += 1
        return -1
```

令 v 节点数, e 为边数。

复杂度分析

- 时间复杂度：$O(v + e)$
- 空间复杂度：$O(v)$

上一页 下一页