19 第六讲 作业记录（二）

笔记本: 浙江大学《数据结构》
创建时间: 2025/4/19 18:13 **更新时间:** 2025/4/19 18:22
作者: panhengye@163.com
URL: https://pintia.cn/problem-sets/1873565885118418944/exam/problems/type/7?...

06-图2 Saving James Bond - Easy Version

【提交结果】

提交结果

题目	用户	提交时间
06-图2	飞翔的小师弟	2025/04/19 18:00:24
编译器	**内存**	**用时**
Python (python3)	3132 / 65536 KB	19 / 400 ms
状态	**分数**	**评测时间**
答案正确	25 / 25	2025/04/19 18:00:24

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample 有不成功的分支，连续几次到岸；有可以到岸但跳不过去的，多连通	3132	16	答案正确	7 / 7
1	sample 都可以跳到，但不到岸	3124	16	答案正确	6 / 6
2	最小N	2876	15	答案正确	3 / 3
3	最小跳，人工乱序	3004	16	答案正确	3 / 3
4	最大N最小跳，4象限对称，人工乱序	3008	19	答案正确	3 / 3
5	都能到岸，但够不着	2880	17	答案正确	3 / 3

【代码记录】

```
import math


def first_jump(distance, isle_r, locations_croc):

    """
    当Bond站在小岛中心时，尝试往岛外跳跃。

    :param locations_croc: 所有鳄鱼的位置。
    :param isle_r: 表示小岛的半径。
    :param distance: 表示最大跳跃距离的数值。
```

```
    :return: 返回起点坐标的列表，如果没有可以跳的位置就返回空列表。
    """

    start = []
    for x,y in locations_croc:
         # 判断鳄鱼是否在Bond从岛上可跳到的范围内
        if math.sqrt(x*x + y*y) <= isle_r + distance:
            start.append((x, y))

    return start

def DFS(location, vertex, distance, visited=None):
    """
    当Bond已经跳上了鳄鱼的脑袋，尝试跳上岸。递归实现深度优先搜索。

    :param location: Bond当前所在的位置坐标。
    :param vertex: 所有鳄鱼的位置列表。
    :param distance: 表示最大跳跃距离的数值。
    :param visited: 已访问的位置集合。
    :return: 如果可以跳上岸返回 True，否则返回 False。
    """
    # 初始化visited集合
    if visited is None:
        visited = set()

    # 如果当前位置可以直接跳上岸，返回True
    if is_safe(location, distance):
        return True

    # 标记当前位置为已访问
    visited.add(location)

    # 递归探索所有可能的下一步位置
    for next_pos in vertex:
        # 如果该位置没有被访问过且可以从当前位置跳到
        if next_pos not in visited and jump_nearby(distance, location, next_pos):
            # 递归探索从next_pos出发的路径
            if DFS(next_pos, vertex, distance, visited):
                return True

    # 所有可能的路径都无法到达岸边
    return False

def is_safe(location, distance):

    """
    判断当前位置是否可以跳上岸。

    :param location: 包含 x 和 y 坐标的可迭代对象（如元组或列表），表示当前位置。
    :param distance: 表示最大跳跃距离的数值。
    :return: 如果可以跳上岸返回 True，否则返回 False。
    """

    x, y = location

    if x + 50 <= distance or 50 - x <= distance:
        return True
    elif y + 50 <= distance or 50 - y <= distance:
        return True
    else:
        return False

def jump_nearby(distance, v, w):

    """
    判断是否能够跳上邻近的鳄鱼。

    :param v: 当前Bond所在的位置。
    :param w: 邻近的一条鳄鱼的位置。
    :param distance: 表示最大跳跃距离的数值。
```

```
    :return: 如果可以跳上鳄鱼返回 True，否则返回 False。
    """

    if math.sqrt((v[0] - w[0])**2 + (v[1] - w[1])**2) <= distance:
        return True
    else:
        return False

def main():

    # 处理输入数据
    num_croc, D = map(int, input().split())
    locations_croc = [tuple(map(int, input().split())) for _ in range(num_croc)]
    radius = 15 / 2 # 根据题意：The central island is a disk centered at (0,0)
with the diameter of 15

    # 首次跳跃
    loc_of_Bond = first_jump(D, radius, locations_croc)

    # 如果已经踩在了鳄鱼脑袋上，就继续跳跃
    result = False # 默认Bond逃不出去
    if loc_of_Bond:
        for loc in loc_of_Bond: # 注意存在多个起点
            if DFS(loc, locations_croc, D):
                result = True
                break

    # 给出最终判断
    print("Yes" if result else "No")

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

【整体思路】

DFS算法

```
void DFS ( Vertex V )
{ visited[V] = true;
  for ( V 的每个邻接点 W )
     if ( !visited[W] )
         DFS(W);
}
```

```
int DFS ( Vertex V )
{ visited[V] = true;
  if ( IsSafe(V) ) answer = YES;
  else {
    for ( each W in G )
      if (!visited[W] && Jump(V,W)) {
         answer = DFS(W);
         if (answer==YES) break;
      }
  }
  return answer;
}
```

伪代码如上图所示。
整个脚本需要设计如下函数：

- first_jump：
  - 小岛本身是有半径的
  - 对于Bond来说，他初次跳跃的选择可能不唯一，因此需要收集所有可能的坐标
- DFS：这里使用递归的思路实现，也可以用堆栈来实现
- is_safe：什么是“安全”呢？有四种边界条件，如（x + 50 <= D），满足任何一个都可以
- jump_nearby：根据距离公式判断是否能够跳跃
- main
  - 首行输入值要单独处理
  - 首次跳跃后要加上判断条件，如果没有鳄鱼可以跳，直接返回No
  - 由于要考虑多个起点的情况，所以增加判断条件，如果已经找到出路了就不用考虑后续起点

【关键点解析1】

```
# 初始化visited集合
if visited is None:
    visited = set()
```

- visited是后续判断所需要的关键条件
- 用if语句做分支设计
  - 初始化为空集合
  - 后续迭代过程中不断传递该集合