剑指 Offer 13. 机器人的运动范围

```
class Solution:
    def __init__(self):
        self.visited = None
        self.count = 0
    def movingCount(self, m: int, n: int, k: int) -> int:
        self.visited = [[False] * n for i in range(m)]
        self.dfs(0,0,m,n,k)
        return self.count

    def dfs(self, i, j, m, n ,k):
        self.visited[i][j] = True
        self.count += 1
        directions = [[-1,0], [1, 0], [0, -1], [0, 1]]
        for di in range(4):
            newi = i + directions[di][0]
            newj = j + directions[di][1]
            if newi >= m or newi < 0 or newj >= n or newj < 0 or self.visited[newi][newj] or
self.check(newi, newj, k) == False:
                continue
            self.dfs(newi,newj, m,n,k)

    def check(self,i,j,k):
        total = 0
        while i > 0 :
            total += i % 10
            i = i // 10
        while j > 0:
            total += j % 10
            j = j // 10
        return total <= k
```

面试题 08.10. 颜色填充

```
class Solution:
    def floodFill(self, image: List[List[int]], sr: int, sc: int, newColor: int) -> List[List[int]]:
        n = len(image)
        m = len(image[0])
        self.dfs(image, n,m,sr,sc,image[sr][sc], newColor)
        return image

    def dfs(self, image, n, m, sr,sc, colour, newColor):
        image[sr][sc]= newColor
        dirs = [[-1,0], [1,0], [0, -1], [0, 1]]
        for k in range(4):
            newsr = sr + dirs[k][0]
            newsc = sc + dirs[k][1]
            if newsr >= n or newsr < 0 or newsc >= m or newsc < 0 or image[newsr][newsc] ==
newColor or image[newsr][newsc] != colour:
                continue
            self.dfs(image, n,m,newsr,newsc,colour,newColor)
```

面试题 04.01. 节点间通路

```
class Solution:
    def __init__(self):
        self.visted = None
        self.found = False
        self.Hashset = set()
    def findWhetherExistsPath(self, n: int, graph: List[List[int]], start: int, target: int) -> bool:
        self.visted = [False] * n
        self.adj = {i:set() for i in range(n)}
        for item in graph:
            self.adj[item[0]].add(item[1])
        self.dfs(start, target)
        return self.found

    def dfs(self, cur, target):
        if self.found:
            return
        if cur == target:
            self.found = True
            return
        self.visted[cur] = True
        for next in self.adj[cur]:
            if not self.visted[next]:
                self.dfs(next, target)
```

200. 岛屿数量

```
class Solution:
    def __init__(self):
        self.visited = None
        self.h = 0
        self.w = 0
        self.result = 0
    def numIslands(self, grid: List[List[str]]) -> int:
        self.h = len(grid)
        self.w = len(grid[0])
        self.visited = [[False] * self.w for i in range(self.h)]
        for i in range(self.h):
            for j in range(self.w):
                if grid[i][j] == "1" and self.visited[i][j] !=True:
                    self.result += 1
                    self.dfs(grid,i,j)
        return self.result

    def dfs(self, grid, i, j):
        directions = [[-1,0], [1,0], [0,-1], [0,1]]
        self.visited[i][j] = True
        for k in range(4):
            newi = directions[k][0] + i
            newj = directions[k][1] + j
            if newi >= 0 and newi < self.h and newj >= 0 and newj < self.w and self.visited[newi]
[newj] != True and grid[newi][newj] == "1":
                self.dfs(grid,newi, newj)
```

面试题 16.19. 水域大小

```
class Solution:
    def __init__(self):
        self.count = 0
        self.n = 0
        self.m = 0
    def pondSizes(self, land: List[List[int]]) -> List[int]:
        self.n = len(land)
        self.m = len(land[0])
        result = []
        for i in range(self.n):
            for j in range(self.m):
                if land[i][j] == 0:
                    self.count = 0
                    self.dfs(land, i, j)
                    result.append(self.count)
        result.sort()
        return result

    def dfs(self, land, i, j):
        self.count += 1
        land[i][j] = 1
        dirs = [[-1,0],[1,0], [0,-1],[0,1],[-1,-1],[1,1],[-1,1],[1,-1]]
        for d in range(8):
            newi = i + dirs[d][0]
            newj = j + dirs[d][1]
            if newi >= 0 and newi < self.n and newj >= 0 and newj < self.m and land[newi][newj] ==
0:
                self.dfs(land,newi,newj)
```

207. 课程表

```
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        adjs = [set() for i in range(numCourses)]
        indegrees = [0] * numCourses
        for i in range(len(prerequisites)):
            adjs[prerequisites[i][1]].add(prerequisites[i][0])
            indegrees[prerequisites[i][0]] += 1
        zeroInDegrees = set()
        for i in range(len(indegrees)):
            if indegrees[i] == 0:
                zeroInDegrees.add(i)
        zeroInDegreesCount = 0
        while zeroInDegrees:
            coursei = zeroInDegrees.pop()
            zeroInDegreesCount += 1
            for coursej in adjs[coursei]:
                indegrees[coursej] -= 1
                if indegrees[coursej] == 0:
                    zeroInDegrees.add(coursej)
        print(indegrees)
        return zeroInDegreesCount == numCourses
```

79. 单词搜索

```
class Solution:
    def __init__(self):
        self.h = 0
        self.w = 0
    def exist(self, board: List[List[str]], word: str) -> bool:
        self.exist = False
        self.h = len(board)
        self.w = len(board[0])
        for i in range(self.h):
            for j in range(self.w):
                visited = [[False] * self.w for i in range(self.h)]
                self.dfs(board,word,i,j,0,visited)
                if self.exist:
                    return self.exist
        return self.exist

    def dfs(self, board, word,i,j,k, visited):
        if self.exist == True:
            return
        if word[k] != board[i][j]:
            return
        visited[i][j] = True
        if k == (len(word) - 1):
            self.exist = True
            return
        directions = [[-1,0], [1,0], [0, -1], [0,1]]
        for d in range(4):
            nexti = i + directions[d][0]
            nextj = j + directions[d][1]
            if nexti >= 0 and nexti < self.h and nextj >= 0 and nextj < self.w and not visited[nexti]
[nextj]:
                self.dfs(board, word, nexti, nextj, k+1, visited)
        visited[i][j] = False
```

1306. 跳跃游戏 III

```
class Solution:
    def __init__(self):
        self.visited = None
        self.reached = False
    def canReach(self, arr: List[int], start: int) -> bool:
        self.visited = [False] * len(arr)
        self.dfs(arr, start)
        return self.reached

    def dfs(self, arr, curi):
        if self.reached:
            return
        if arr[curi] == 0:
            self.reached = True
            return
        self.visited[curi] = True
        move2left = curi - arr[curi]
        if move2left >= 0 and move2left < len(arr) and self.visited[move2left] == False:
            self.dfs(arr, move2left)
        move2right = curi + arr[curi]
        if move2right >= 0 and move2right < len(arr) and self.visited[move2right] == False:
            self.dfs(arr, move2right)
```

752. 打开转盘锁

```
from queue import Queue
class Solution:
    def openLock(self, deadends: List[str], target: str) -> int:
        deadset = set()
        for d in deadends:
            deadset.add(d)
        if "0000" in deadset:
            return -1
        visited = set()
        queue = Queue()
        queue.put("0000")
        visited.add("0000")
        depth = 0
        while not queue.empty():
            size = queue.qsize()
            k = 0
            while k < size:
                node = queue.get()
                k += 1
                if node == target:
                    return depth
                newNodes = self.genNewNode(node)
                for newNode in newNodes:
                    if newNode in visited or newNode in deadset:
                        continue
                    queue.put(newNode)
                    visited.add(newNode)
            depth += 1
        return -1

    def genNewNode(self, node):
        newnodes = []
        change = [-1,1]
        for i in range(4):
            for k in range(2):
                newNode = [None] * 4
                for j in range(i):
                    newNode[j] = node[j]
                for j in range(i+1,4):
                    newNode[j] = node[j]
                newC = str((int(node[i]) + change[k] + 10) % 10)
                newNode[i] = newC
                newnodes.append("".join(newNode))
        return newnodes
```

面试题 17.22. 单词转换

```
class Solution:
```

```
def __init__(self):
    self.visited = set()
    self.found = False
    self.resultPath = []
def findLadders(self, beginWord: str, endWord: str, wordList: List[str]) -> List[str]:
    self.dfs(beginWord, endWord, [],wordList)
    return self.resultPath

def dfs(self, curWord, endWord, path, wordList):
    if self.found:
        return
    path.append(curWord)
    self.visited.add(curWord)
    if curWord == endWord:
        self.resultPath.extend(path)
        self.found = True
        return
    for i in range(len(wordList)):
        nextWord = wordList[i]
        if nextWord in self.visited or not self.isValidChange(curWord, nextWord):
            continue
        self.dfs(nextWord, endWord, path, wordList)
    path.pop()

def isValidChange(self, word1, word2):
    diff = 0
    for i in range(len(word1)):
        if word1[i] != word2[i]:
            diff += 1
    return diff == 1
```