拉勾教育
—互联网人实战大学—

深度优先搜索 + 拓扑排序：矩阵中的最长递增路径

困难/有向图、图论、深度优先搜索、拓扑排序

学习目标

- 熟练使用图结构建模解决实际问题
- 加深对图结构中深度优先搜索算法的理解
- 掌握拓扑排序的基本思想与步骤

题目描述

给定一个整数矩阵，找出最长递增路径的长度。

对于每个单元格，你可以往上，下，左，右四个方向移动。 你不能在对角线方向上移动或移动到边界外（即不允许环绕）。

输入: nums = [[**9**,9,4],
[**6**,6,8],
[**2**,**1**,1]]
输出: 4　解释: 最长递增路径为 [1, 2, 6, 9]。

输入: nums = [[**3**,**4**,**5**],
[3,2,**6**],
[2,2,1]]
输出: 4　解释: 最长递增路径是 [3, 4, 5, 6]。注意不允许在对角线方向上移动

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

节点间通路

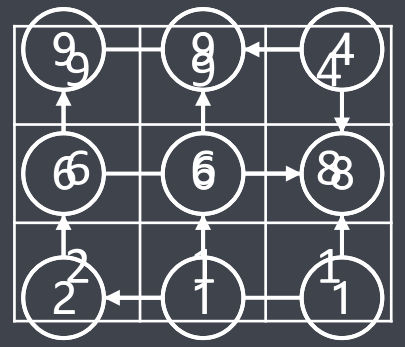
- 给定一个整数矩阵，寻找最长递增路径
- 每个单元格可以在上下左右四个方向移动
- 最长递增路径：1→2→6→9 长度为4

9	9	4
6	6	8
2	1	1

二. Choose 数据结构及算法思维选择

问题建模

- 将矩阵中每个元素当成一个节点，连接上下左右相邻的元素，我们可以得到一个图
- 由于题目要求寻找最长递增路径，我们可以只保留从小元素指向大元素的有向边
- 那么该问题也就变成了寻找有向图中的最长路径
- 由于该图中的邻接关系是固定的，我们并不用显式构建邻接表或邻接矩阵



二. Choose 数据结构及算法思维选择

数据结构选择

- 输入的数据类型为一个矩阵
- 输出的数据类型为一个整数，表示矩阵中最长的递增路径长度
- 我们虽然将矩阵转换为有向图，但是我们并不用构建真正的图结构，存储邻接矩阵或邻接表

二. Choose 数据结构及算法思维选择

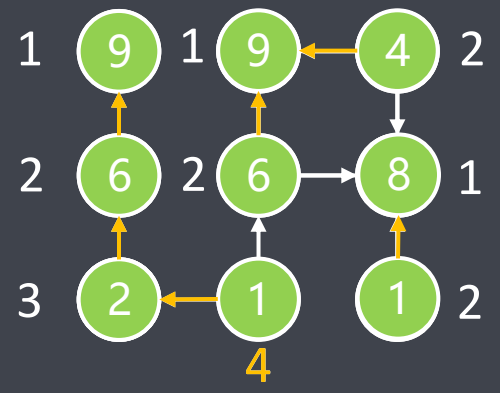
算法思维选择

- 遍历有向图中从所有节点出发的所有路径，找到其中最长的那一条，返回其长度
- 遍历的方法可以采用我们之前讲解过的广度优先搜索或深度优先搜索

三. Code 基本解法及编码实现

解题思路剖析

- 深度优先搜索
- 遍历有向图中从所有节点出发的所有路径
- 找到从每个节点出发的最长路径，找到最优的起始节点



三. Code 基本解法及编码实现

复杂度分析

- 时间复杂度
  - 图中的深度优先搜索算法，DFS需要查找图中任一节点出发的所有路径，时间复杂度为$O(|V|^{|V|})$
- 空间复杂度
  - 空间消耗方面，由于该有向图连接关系较简单，我们不需要建立邻接表或邻接矩阵来存储图的连接关系，只需要一些辅助变量，空间复杂度为O(1)

三. Code 基本解法及编码实现

Java编码实现

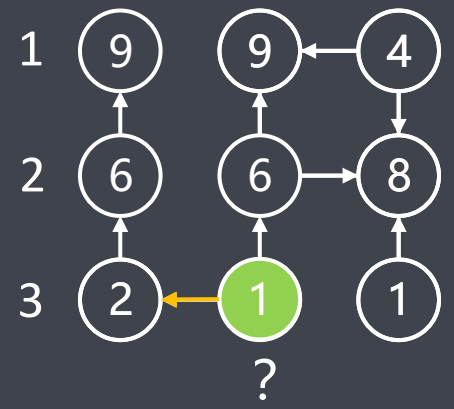
```java
public int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
public int rows, columns;
public int dfs(int[][] matrix, int row, int column) {
    int result = 1;
    for (int[] dir : dirs) {
        int newRow = row + dir[0], newColumn = column + dir[1];
        //如果邻居元素存在，且比当前元素大
        if (newRow >= 0 && newRow < rows && newColumn >= 0 &&
newColumn < columns && matrix[newRow][newColumn] > matrix[row][column])
            result = Math.max(result, dfs(matrix, newRow, newColumn) + 1);
    }
    return result;
}
public int longestIncreasingPath(int[][] matrix) {
    if (matrix.length == 0 || matrix[0].length == 0) return 0;
    rows = matrix.length;
    columns = matrix[0].length;
    int ans = 0;
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < columns; ++j)
            ans = Math.max(ans, dfs(matrix, i, j));
    return ans;
}
```

执行结果： 超出时间限制

四. Consider 思考更优解

是否存在无效代码或者无效空间消耗

是否有更好的算法思维



- 经过分析我们可以发现朴素算法中存在冗余计算
- 在寻找从当前节点1出发的最长路径时，某一次搜索我们选择了向左访问节点2
- 而此时从节点2出发的最长路径长度已经确定，那么当前搜索分支从节点1出发经过节点2的所有路径中最长的那一条长度一定是：
  - 从节点1到节点2路径的长度+从节点2出发的最长路径

无需继续扩展节点2

五. Code 最优解思路及编码实现

解题思路剖析

- 那么我们只需要额外维护一个数组记录当前已经确定最长路径的节点及其长度
- 如果搜索到达的当前节点已经确定了最长路径的长度，那么直接返回该长度，无需继续扩展该节点

五. Code 最优解思路及编码实现

复杂度分析

- 时间复杂度
  - 不难发现，在改进的深度优先算法中，每个节点只会被扩展一次，每条边也只会被访问一次，那么总的时间复杂度为O(|E|+|V|)
  - 矩阵中|V|=m*n，|E| = O(m*n)，据此时间复杂度可以写为O(m*n)
- 空间复杂度
  - 空间消耗方面我们需要存储每个元素出发的最短路径，空间复杂度为O(m*n)

五. Code 最优解思路及编码实现

Java编码实现

```java
public int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
public int rows, columns;
public int[][] longestPath; //数组longestPath记录从各个节点出发的最长路径的长度
public int dfs(int[][] matrix, int row, int column) {
    //如果该节点已经被扩展过
    if (longestPath[row][column] != 0) return longestPath[row][column];
    longestPath[row][column] = 1; //初始化为1
    for (int[] dir : dirs) {
        int newRow = row + dir[0], newColumn = column + dir[1];
        //如果邻居元素存在，且比当前元素大
        if (newRow >= 0 && newRow < rows && newColumn >= 0 && newColumn < columns &&
matrix[newRow][newColumn] > matrix[row][column])
            longestPath[row][column] =
              Math.max(longestPath[row][column], dfs(matrix, newRow, newColumn) + 1);
    }
    return  longestPath[row][column];
}
public int longestIncreasingPath(int[][] matrix) {
    if (matrix.length == 0 || matrix[0].length == 0) return 0;
    rows = matrix.length;
    columns = matrix[0].length;
    longestPath = new int[rows][columns];
    int ans = 0;
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < columns; ++j)
            ans = Math.max(ans, dfs(matrix, i, j));
    return ans;
}
```

执行结果： 通过 显示详情 >

执行用时： 10 ms，在所有 Java 提交中击败了 83.22% 的用户

内存消耗： 38.5 MB，在所有 Java 提交中击败了 97.30% 的用户

五. Code 最优解思路及编码实现 II

解题思路剖析

- 如果我们再一次审视题目，我们可以有以下观察：
  - 最长递增路径的起始点一定不会大于周围四个邻居的数值
  - 否则我们可以得到一条从其邻居出发经过该起始点的更长路径
  - 也就是说有向图中最长路径的起始节点入度一定为0
- 那么我们只需要寻找从这样的起始点出发的最长路径
  - 拓扑排序

五. Code 最优解思路及编码实现 II

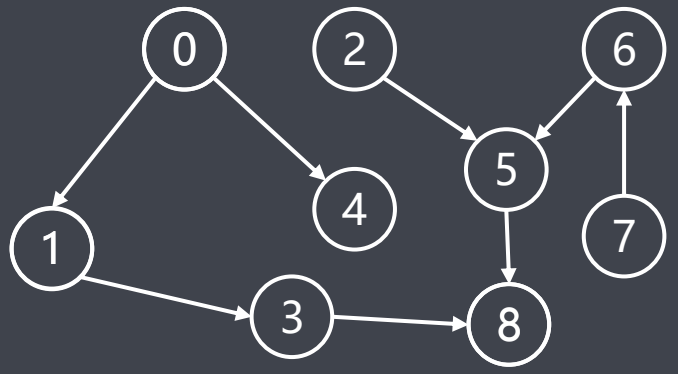
关键知识点：拓扑排序（Topological sort）

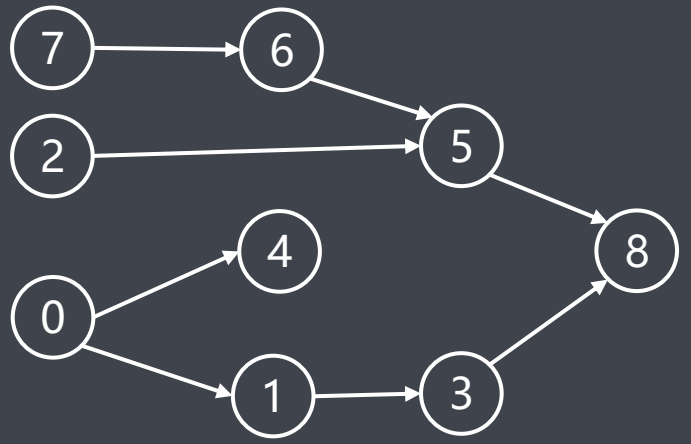
- 在计算机科学领域，有向图的拓扑排序是对其顶点的一种线性排序，使得对于图中的每个有向边 uv，u 在排序中都在v之前。
- 例如，图的顶点可以表示要执行的任务，边可以表示一个任务必须在另一个任务之前执行的约束；在这个应用中，拓扑排序只是一个有效的任务顺序。
- 当且仅当图中没有定向环时（即有向无环图），才有可能进行拓扑排序。

五. Code 最优解思路及编码实现 II

关键知识点：拓扑排序（Topological sort）

- 对于下面的有向图，我们可以进行拓扑排序得到：

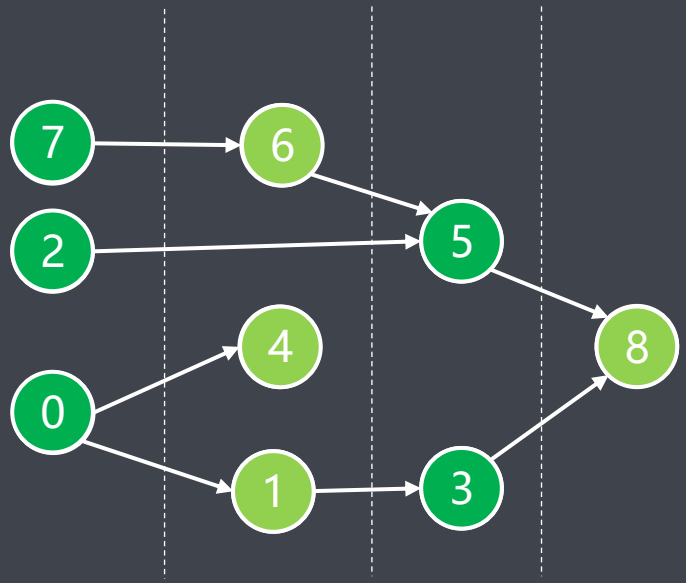




五. Code 最优解思路及编码实现 II

关键知识点：拓扑排序（Topological sort）

- 我们可以将拓扑排序后的有向图划分为4个部分
- 其中每一个部分的节点都不依赖后面部分的节点
- 对于该图来说，最长路径就是划分的数目4
- 那么我们怎么得到拓扑排序的结果呢?
- 首先第一部分的节点一定是入度为0的节点
- 然后我们删除这些节点的出边，再找出剩下节点中入度为0的节点，如此往复就可以完成拓扑排序



五. Code 最优解思路及编码实现 II

复杂度分析

- 时间复杂度
  - 拓扑排序也仅需要访问每条边、每个节点一次，总的时间复杂度亦为O(|E|+|V|)
  - 矩阵中|V|=m*n，|E| = O(m*n)，时间复杂度也可以写为O(m*n)
- 空间复杂度
  - 空间消耗方面我们只需要一些临时变量，均为常数级的变量空间，空间复杂度为O(1)

五. Code 最优解思路及编码实现 II

Java编码实现

```
public int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
public int longestIncreasingPath(int[][] matrix) {
    if (matrix.length == 0 || matrix[0].length == 0) return 0;
    rows = matrix.length;
    columns = matrix[0].length;
    int[][] indegrees = new int[rows][columns]; //统计每个节点的入度
    for (int i = 0; i < rows; ++i)
      for (int j = 0; j < columns; ++j)
        for (int[] dir : dirs) {
            int newRow = i + dir[0], newColumn = j + dir[1];
            if (newRow >= 0 && newRow < rows && newColumn >= 0 && newColumn < columns && matrix[newRow]
[newColumn] < matrix[i][j])  ++indegrees[i][j];
        }
    Queue<int[]> queue = new LinkedList<int[]>();
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < columns; ++j)
            //将入度为0的节点加入队列
            if (indegrees[i][j] == 0) queue.offer(new int[]{i, j});
    int ans = 0;
    while (!queue.isEmpty()) {
        ++ans;
        int size = queue.size();
        for (int i = 0; i < size; ++i) {
            int[] cell = queue.poll();
            int row = cell[0], column = cell[1];
            for (int[] dir : dirs) {
                int newRow = row + dir[0], newColumn = column + dir[1];
                if (newRow >= 0 && newRow < rows && newColumn >= 0 && newColumn < columns && matrix[new
Row][newColumn] > matrix[row][column]) {
                    --indegrees[newRow][newColumn];
                    if (indegrees[newRow][newColumn] == 0)   queue.offer(new int[]{newRow, newColumn});
                }
            }
        }
    }
    return ans;
}
```

执行结果： 通过 显示详情 >

执行用时： 18 ms , 在所有 Java 提交中击败了 17.90% 的用户

内存消耗： 38.6 MB , 在所有 Java 提交中击败了 95.62% 的用户

六. Change 变形延伸

拉勾教育
—互联网人实战大学—

题目变形

1. 若要求寻找最长递减路径?

延伸扩展

- 拓扑排序广泛应用于实际生活中
  - 产品生产流程图：一个较大的工程往往被划分成许多子工程，有些子工程必须在其它有关子工程完成之后才能开始，但有些子工程则可以安排在任何时间开始
  - 类似的任务还有课程学习流程图

总结

- 熟练使用图结构建模解决实际问题
- 加深对图结构中深度优先搜索算法的理解
- 掌握拓扑排序的基本思想与步骤

课后练习

1. 朋友圈（leetcode 547/中等）
2. 课程表（leetcode 207/中等）
3. 课程表 II（leetcode 210/中等）
4. 项目管理（leetcode 1203/困难）

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容