

**《算法分析与技术》实验报告**

学生姓名： 8208221512 陈康坤

指导老师： 李 敏

2024年06月

目录

[实验一：一和零问题 3](#_Toc5415)

[（一）实验目的 3](#_Toc1029)

[（二）实验内容 3](#_Toc8986)

[（三）算法实现 4](#_Toc17087)

[（四）实验测试结果 5](#_Toc27992)

[实验二:删除给定值的叶子节点 6](#_Toc4193)

[（一）实验目的 6](#_Toc16506)

[（二）实验内容 6](#_Toc8072)

[（三）算法实现 6](#_Toc17278)

[（四） 实验测试结果 7](#_Toc29481)

[实验三:不同路径 III 8](#_Toc13047)

[（一） 实验目的 8](#_Toc11035)

[（二）实验内容 9](#_Toc7420)

[（三）算法实现 9](#_Toc30113)

[（四） 实验测试结果 10](#_Toc26956)

[实验总结 11](#_Toc15404)

[（一）实验1： 一和零 11](#_Toc1492)

[（二）实验2： 删除给定值的叶子节点 11](#_Toc15921)

[（三）实验3： 不同路径 III 11](#_Toc141)

[（四）总体总结 12](#_Toc13517)

[1. 动态规划： 12](#_Toc11519)

[2. 递归： 12](#_Toc12485)

[3. 深度优先搜索： 12](#_Toc7885)

# 实验一：一和零问题

## （一）实验目的

本实验的目的是利用动态规划算法解决一个优化问题，即在给定的二进制字符串数组中，找到一个最多包含m个0和n个1的最大子集的长度。通过本实验，我们将理解和应用动态规划的基本概念和技巧。

## （二）实验内容

给你一个二进制字符串数组 strs 和两个整数 m 和 n 。

请你找出并返回 strs 的最大子集的长度，该子集中 最多 有 m 个 0 和 n 个 1 。

如果 x 的所有元素也是 y 的元素，集合 x 是集合 y 的 子集 。

示例 1：

输入：strs = ["10", "0001", "111001", "1", "0"], m = 5, n = 3

输出：4

解释：最多有 5 个 0 和 3 个 1 的最大子集是 {"10","0001","1","0"} ，因此答案是 4 。

其他满足题意但较小的子集包括 {"0001","1"} 和 {"10","1","0"} 。{"111001"} 不满足题意，因为它含 4 个 1 ，大于 n 的值 3 。

示例 2：

输入：strs = ["10", "0", "1"], m = 1, n = 1

输出：2

解释：最大的子集是 {"0", "1"} ，所以答案是 2 。

提示：

1 <= strs.length <= 600

1 <= strs[i].length <= 100

strs[i] 仅由 '0' 和 '1' 组成

1 <= m, n <= 100

## （三）算法实现

设计思想

我们使用动态规划来解决这个问题。我们构建一个三维的动态规划数组 dp，其中 dp[i][j][k] 表示在前 i 个字符串中，最多包含 j 个0和 k 个1时的最大子集长度。对于每个字符串，我们有两个选择：选或不选。如果不选，当前的状态等于前一个状态。如果选，则需要考虑减去当前字符串的0和1的数量，并加1。

基本步骤

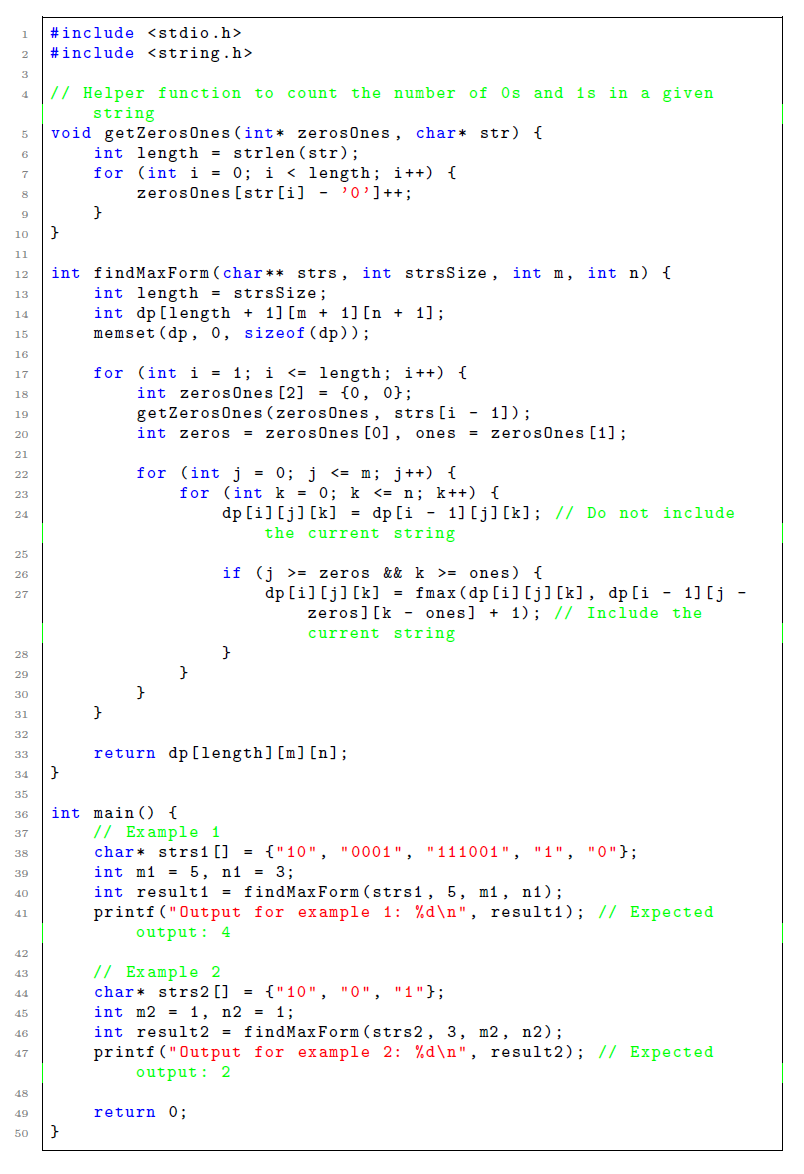
初始化动态规划数组 dp，全部设置为0。

遍历每个字符串，计算其中0和1的数量。

更新动态规划数组 dp，对于每个可能的0和1的数量，如果当前字符串可以加入，则更新状态。

最终结果存储在 dp[length][m][n] 中。

核心代码



## （四）实验测试结果

实验测试结果

示例1

输入：strs = ["10", "0001", "111001", "1", "0"], m = 5, n = 3

输出：4

解释：最多有5个0和3个1的最大子集是 {"10", "0001", "1", "0"}，所以答案是4。

示例2

输入：strs = ["10", "0", "1"], m = 1, n = 1

输出：2

解释：最大的子集是 {"0", "1"}，所以答案是2

# 实验二:删除给定值的叶子节点

## （一）实验目的

**本实验的目的是通过递归算法删除二叉树中所有值为目标值的叶子节点，并在删除叶子节点后继续检查其父节点是否也成为新的目标叶子节点，直到不能继续删除。**

## （二）实验内容

**1. 输入：一个二叉树的根节点 `root` 和一个整数 `target`。**

**2. 输出：经过处理后的二叉树的根节点，其中所有值为 `target` 的叶子节点及其父节点被删除，直至不能继续删除。**

## （三）算法实现

### 设计思想

**通过递归的方法遍历整棵树，从子节点向根节点进行删除操作。对于每一个节点，先处理其左子树和右子树，再根据当前节点是否成为叶子节点以及其值是否为目标值来决定是否删除该节点。**

### 基本步骤

**1. 如果当前节点为空，返回 `nullptr`。**

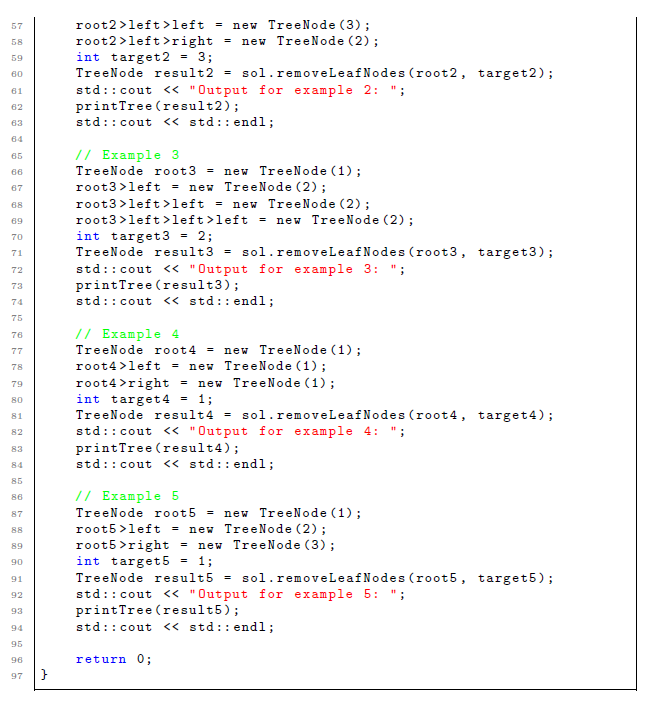
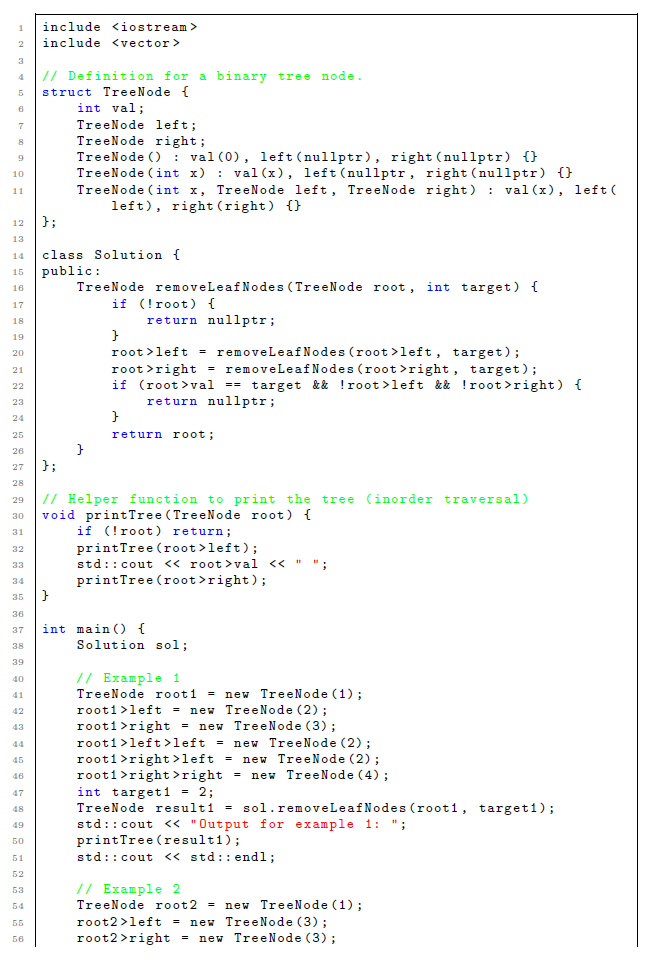
**2. 递归处理当前节点的左子树和右子树。**

**3. 在递归调用后检查当前节点：**

**如果当前节点是叶子节点且其值等于 `target`，删除该节点（返回 `nullptr`）。**

**否则返回当前节点。**

### 核心代码



## （四） 实验测试结果

**示例1**

**输入：`root = [1,2,3,2,null,2,4]`, `target = 2`**

**输出：`[1,null,3,null,4]`**

**示例2**

**输入：`root = [1,3,3,3,2]`, `target = 3`**

**输出：`[1,3,null,null,2]`**

**示例3**

**输入：`root = [1,2,null,2,null,2]`, `target = 2`**

**输出：`[1]`**

**示例4**

**输入：`root = [1,1,1]`, `target = 1`**

**输出：`[]`**

**示例5**

**输入：`root = [1,2,3]`, `target = 1`**

**输出：`[1,2,3]`**

**总结**

**通过本实验，我们使用递归算法成功地解决了删除二叉树中所有值为目标值的叶子节点的问题。算法正确处理了各个边界情况，并通过多组测试数据验证了其正确性。实验结果与预期一致，证明了算法的有效性。**

# 实验三:不同路径 III

## （一） 实验目的

**本实验的目的是使用深度优先搜索（DFS）算法来解决二维网格中寻找从起始点到结束点的所有不同路径的数量的问题。要求每条路径必须经过所有无障碍的方格一次且只能经过一次。**

## （二）实验内容

**1. 输入：一个二维网格 `grid`，其中：**

**`1` 表示起始方格。**

**`2` 表示结束方格。**

**`0` 表示可以走过的空方格。**

**`1` 表示无法跨越的障碍。**

**2. 输出：所有满足条件的路径的数量。**

## （三）算法实现

**设计思想**

**使用DFS算法从起始点出发，遍历每一个可以走的方格，记录路径经过的方格数。当到达结束点时，如果路径经过了所有的无障碍方格，则路径有效。为了避免重复经过同一个方格，每次经过一个方格时，将其标记为障碍（即 1），在递归返回时再恢复原状。**

**基本步骤**

**1. 初始化参数，找到起始点位置并统计所有无障碍方格的数量。**

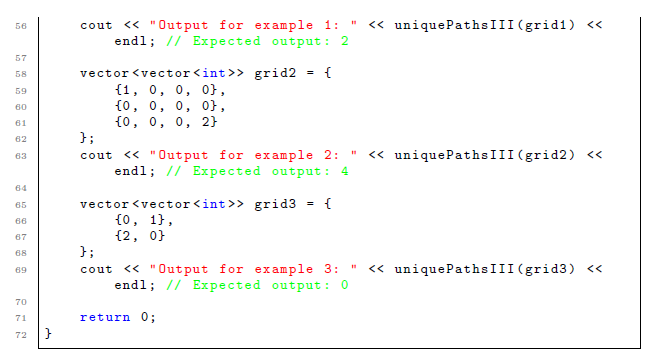
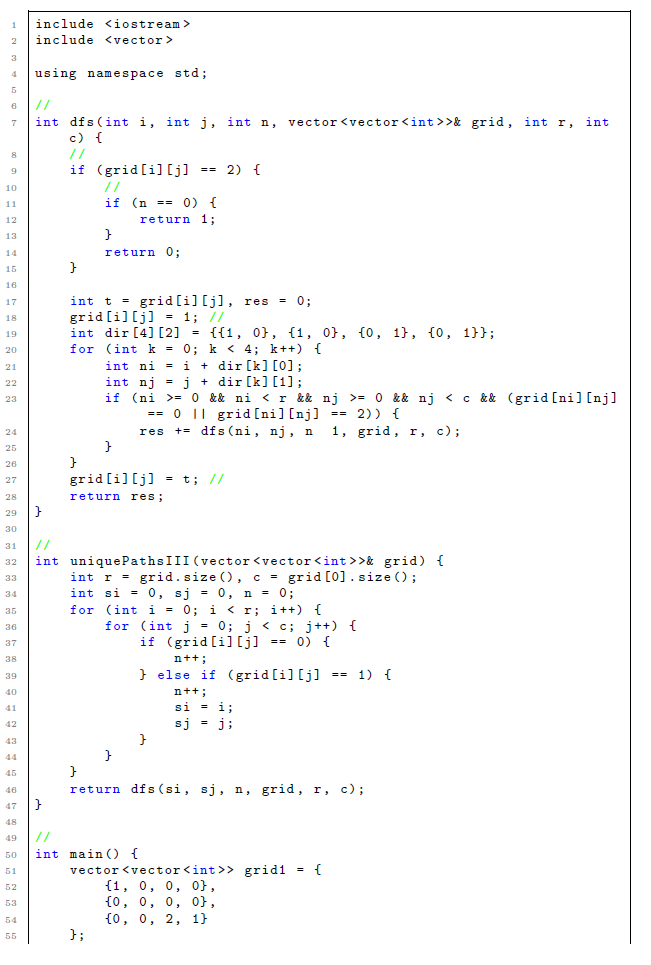
**2. 使用DFS从起始点开始遍历：**

**如果当前方格是结束方格且路径经过了所有无障碍方格，则路径有效。**

**遍历当前方格的四个方向，如果可以走（即未标记为障碍），则继续递归。**

**3. 返回所有有效路径的数量。**

**核心代码**



## （四） 实验测试结果

**示例1**

**输入：`grid = [[1,0,0,0],[0,0,0,0],[0,0,2,1]]`**

**输出：2**

**解释：有两条路径满足条件。**

**示例2**

**输入：`grid = [[1,0,0,0],[0,0,0,0],[0,0,0,2]]`**

**输出：4**

**解释：有四条路径满足条件。**

**示例3**

**输入：`grid = [[0,1],[2,0]]`**

**输出：0**

**解释：没有路径满足条件。**

# 实验总结

在这三个实验中，分别解决了不同的算法问题，包括二进制字符串子集选择、二叉树节点删除以及二维网格路径查找。每个实验都涉及不同的数据结构和算法技术，具体如下：

# （一）实验1： 一和零

算法通过动态规划实现了有效的状态转移，正确求解了最大子集长度的问题，验证了算法的正确性和有效性。

# （二）实验2： 删除给定值的叶子节点

算法通过递归实现了对二叉树的逐层处理，正确删除了所有目标值叶子节点，并在测试数据中验证了算法的正确性和有效性。

# （三）实验3： 不同路径 III

通过DFS算法有效地遍历了网格，找到了所有满足条件的路径，验证了算法在复杂路径查找问题中的应用和正确性。

# （四）总体总结

在这三个实验中，我应用了动态规划、递归和深度优先搜索等算法技术，成功解决了不同类型的数据结构问题。每个实验都展示了特定算法在特定场景中的有效性，并通过多组测试数据验证了算法的正确性。以下是各实验的关键点和学习总结：

## 1. 动态规划：

动态规划在多维状态问题中表现出色，适用于资源有限制的最优化问题。

通过构建状态转移方程，可以有效地解决复杂的子集选择问题。

## 2. 递归：

递归算法适合处理树形结构的数据，能够逐层解决问题。

通过递归函数的调用和返回，实现了对复杂条件的逐步处理。

## 3. 深度优先搜索：

DFS在路径查找和遍历问题中表现优秀，适用于需要遍历所有可能路径的情况。

通过标记和恢复状态，DFS能够有效避免重复和错误遍历。

这些实验不仅帮助我理解了不同算法的应用场景和实现细节，也提升了我们在实际问题中选择和设计算法的能力。