53、出错的或电路，考点 or 实现——深度优先搜索 DFS

题目描述

某生产门电路的厂商发现某一批次的或门电路不稳定，具体现象为计算两个二进制数的或操作时，第一个二进制数中某两个比特位会出现交换，交换的比特位置是随机的，但只交换这两个位，其他位不变。

很明显，这个交换可能会影响最终的或结果，也可能不会有影响。

为了评估影响和定位出错的根因，工程师需要研究在各种交换的可能下，最终的或结果发生改变的情况有多少种。

输入描述

第一行有一个正整数N; 其中1≤N≤1000000。
第二行有一个长为N的二进制数，表示与电路的第一个输入数，即会发生比特交换的输入数。
第三行有一个长为N的二进制数，表示与电路的第二个输入数。注意第二个输入数不会发生比特交换。

输出描述

输出只有一个整数，表示会影响或结果的交换方案个数。

用例

输入	3 010 110
输出	1
说明	原本010和110的或结果是110，但第一个输入数可能会发生如下三种交换： 1、交换第1个比特和第2个比特，第一个输入数变为100，计算结果为110，计算结果不变 2、交换第1个比特和第3个比特，第一个输入数变为010，计算结果为110，计算结果不变 3、交换第2个比特和第3个比特，第一个输入数变为001，计算结果为111，计算结果改变 故只有一种交换会改变计算结果。

输入	6 011011 110110
输出	4
说明	原本011011和110110的或结果是111111，但是第一个输入数发生如下比特交换会影响最终计算结果： 1、交换第1个比特和第3个比特，第一个输入数变为110011，计算结果变为110111 2、交换第1个比特和第6个比特，第一个输入数变为111010，计算结果变为111110 3、交换第3个比特和第4个比特，第一个输入数变为010111，计算结果变为110111 4、交换第4个比特和第6个比特，第一个输入数变为011110，计算结果变为111100 其他交换都不会影响计算结果，故输出4.

题目解析

我们假设第一个输入数是bin1，第二个输入数是bin2。

需要注意的是，本题不能去求解bin1的全排列，因为题目说，发生或运算时，bin1只有某两位会发生交换。而全排列无法保证只有两位发生交换，很可能有多位发生交换。

另外，本题的bin1,bin2的长度都为N，1≤N≤1000000，这个数量级求解全排列肯定会超时的。

因此，本题不推荐使用全排列解法。

我的解题思路如下：

二进制的或运算特点是：进行运算的两个二进制数，对应某位上数有一个为1，则结果1，否则结果为0。

因此，我们可以忽略对bin2的值为1的位的检查，因为不管bin1对应位值变为多少，结果都为1，不会造成结果改变。

只对bin2的值为0的位进行检查即可。

此时有两种情况：

bin2值为0的位，在bin1对应位的值为：

- 0：则正确结果为0，想要不同结果，则需要将bin1该位和bin1上值位1的位交换
- 1：则正确结果为1，想要不同结果，则需要将bin1该位和bin1上值为0的位交换

比如用例1中：

bin1 = "010"

bin2 = "110"

bin2值为0的位只有1个，就是第三位，而对应bin1的第三位的值为0，因此bin1需要将该位值变为1，才能产生不同结果，而变为1的交换策略只有一个，那就是其第二位和第三位交换。

比如用例2中：

bin1 = "011011"

bin2 = "110110"

即将bin1种第三位的1替换为0，有两种策略，将bin1中第六位的1替换为0，有两种策略，因此一共是2+2=4种。

但是还有一种特殊情况，即

bin1 = "1001"

bin2 = "0001"

就是出现重叠交换策略的情况：

bin1第一位1替换为0，有两种策略，替换后bin1分别为：0101、0011

bin1第二位0替换为1，有两种策略，替换后bin1为：0101、1100

bin1第三位0替换为1，有两种策略，替换后bin1为：0011、1010

造成重叠的原因就是bin2值为0的位在bin1对应位上的值既有位1的，又有为0的。bin1这些位1，和位0的互相交换就会产生重叠情况。

去重策略是：6 - 1 * 2 = 4

6的含义是包含重复情况的所有交换策略，1*2的含义是bin1、bin2对应位组合为01的有一个，对应位组合为00有两个，因此重复策略为1*2个。

我的解题思路如下，找出bin2值为0的位，并统计对应位上bin1的值为0的有x个，值为1的有y个，同时统计bin1总共有多少个1（假设a个），多少个0（假设b个），然后就可以用公式：

x * a + y * b - x * y

比如用例1，x = 1，y=0，a=1，b=2，最终结果为 1 * 1 + 0 * 2 - 1 * 0 = 1

比如用例2，x = 0，y=2，a= 4, b=2，最终结果为 0 * 4 + 2 * 2 - 0 * 2 = 4

再比如自测用例：

```
4
1001
0001
```

其中 x = 2，y=1, a=2,b=2，最终结果为：2*2 + 1*2 - 2*1 = 4

其中 x = 2，y=1, a=2,b=2，最终结果为：2*2 + 1*2 - 2*1 = 4

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 3) {
    const n = lines[0] - 0;
    const bin1 = lines[1];
    const bin2 = lines[2];
    console.log(getResult(n, bin1, bin2));
    lines.length = 0;
  }
});

/**
 * @param {number} n 二进制长度
```

```
 * @param {string} bin1 可能产生错误交换的二进制
 * @param {string} bin2 不会发生错误的二进制
 */
function getResult(n, bin1, bin2) {
  // 找出bin2值为0的位，并统计对应位上bin1的值为0的有x个
  let x = 0;
  // 找出bin2值为0的位，并统计对应位上bin1的值为1的有y个
  let y = 0;
  // 统计bin1总共有多少个1
  let a = 0;
  // 统计bin1总共有多少个0
  let b = 0;

  for (let i = 0; i < n; i++) {
    if (bin1[i] == "0") {
      b++;
      if (bin2[i] == "0") x++;
    } else {
      a++;
      if (bin2[i] == "0") y++;
    }
  }

  return x * a + y * b - x * y;
}
```

Java算法源码

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = sc.nextInt();
    String bin1 = sc.next();
    String bin2 = sc.next();

    System.out.println(getResult(n, bin1, bin2));
  }

  /**
   * @param n 二进制长度
   * @param bin1 可能产生错误交换的二进制
   * @param bin2 不会发生错误的二进制
   * @return 产生错误结果的情况有几种
   */
  public static int getResult(int n, String bin1, String bin2) {
    // 找出bin2值为0的位，并统计对应位上bin1的值为0的有x个
    int x = 0;
    // 找出bin2值为0的位，并统计对应位上bin1的值为1的有y个
    int y = 0;
```

```
    // 统计bin1总共有多少个1
    int a = 0;
    // 统计bin1总共有多少个0
    int b = 0;

    for (int i = 0; i < n; i++) {
      if (bin1.charAt(i) == '0') {
        b++;
        if (bin2.charAt(i) == '0') x++;
      } else {
        a++;
        if (bin2.charAt(i) == '0') y++;
      }
    }

    return x * a + y * b - x * y;
  }
}
```

Python算法源码

```
# 输入获取
n = int(input())
bin1 = input()
bin2 = input()


# 算法入口
def getResult(n, bin1, bin2):
    """
    :param n: 二进制长度
    :param bin1: 可能产生错误交换的二进制
    :param bin2: 不会发生错误的二进制
    :return: 产生错误结果的情况有几种
    """
    # 找出bin2值为0的位，并统计对应位上bin1的值为0的有x个
    x = 0
    # 找出bin2值为0的位，并统计对应位上bin1的值为1的有y个
    y = 0
    # 统计bin1总共有多少个1
    a = 0
    # 统计bin1总共有多少个0
    b = 0

    for i in range(n):
```

```
        if bin1[i] == '0':
            b += 1
            if bin2[i] == '0':
                x += 1
        else:
            a += 1
            if bin2[i] == '0':
                y += 1

    return x * a + y * b - x * y


# 算法调用
print(getResult(n, bin1, bin2))
```