

Java-DFS 搜索-有这么一款单人卡牌游戏

题目描述：

有这么一款单人卡牌游戏，牌面由颜色和数字组成，颜色为红、黄、蓝、绿中的一种，数字为 0-9 中的一个。游戏开始时玩家从手牌中选取一张卡牌打出，接下来如果玩家手中有和他上一次打出的手牌颜色或者数字相同的手牌，他可以继续将该手牌打出，直至手牌打光或者没有符合条件可以继续打出的手牌。

现给定一副手牌，请找到最优的出牌策略，使打出的手牌最多。

输入描述：

输入为两行，第一行是每张手牌的数字，数字由空格分隔，第二张为对应的每张手牌的颜色，用 **r y b g** 这 4 个字母分别代表 4 种颜色，字母也由空格分隔。手牌数量不超过 10。

输出描述：

输出一个数字，即最多能打出的手牌的数量。

补充说明：

示例 1

输入：

1 4 3 4 5

r y b b r

输出：

3

说明：

如果打出 1r，那么下面只能再打出 5r，共打出两张牌，而按照 4y-4b-3b 的顺序则可以打出三张牌，故输出 3

示例 2

输入：

1 2 3 4

r y b l

输出：

1

说明：

没有能够连续出牌的组合，只能在开始时打出一张手牌，故输出 1

```
import java.util.Scanner;
```

```
// 注意类名必须为 Main, 不要有任何 package xxx 信息
```

```
public class Main {
```

```
    public static void main(String[] args) {  
        //      System.out.println(playPoke(new Pai[]{  
        //          new Pai(1, 'r'),  
        //          new Pai(4, 'y'),  
        //          new Pai(3, 'b'),  
        //          new Pai(4, 'b'),  
        //          new Pai(5, 'r')  
        //      }));
```

```

Scanner in = new Scanner(System.in);
String s1 = in.nextLine();
String[] split1 = s1.split(" ");
String s2 = in.nextLine();
String[] split2 = s2.split(" ");
Pai[] pais = new Pai[split1.length];
for (int i = 0; i < split1.length; i++) {
    pais[i] = new Pai(Integer.parseInt(split1[i]), split2[i].charAt(0));
}
System.out.println(playPoke(pais));
}

static class Pai {
    int num;
    char color;

    public Pai(int num, char color) {
        this.num = num;
        this.color = color;
    }
}

private static int playPoke(Pai[] pais) {
    boolean[] path = new boolean[pais.length];

    int res = 0;
    for (int i = 0; i < pais.length; i++) {
        path[i] = true;
        res = Math.max(res, process(i, pais, path) + 1);
        path[i] = false;
    }
    return res;
}

private static int process(int lastChoose, Pai[] pais, boolean[] path) {
    if (lastChoose >= pais.length) {
        return 0;
    }
    int res = 0;
    for (int i = 0; i < pais.length; i++) {
        if (path[i]) {
            continue;
        }
        if (pais[i].color == pais[lastChoose].color || pais[i].num == pais[lastChoose].num) {

```

```
        path[i] = true;
        res = Math.max(res, process(i, pais, path) + 1);
        path[i] = false;
    }
}
return res;
}
}
```