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，
- 第二行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。

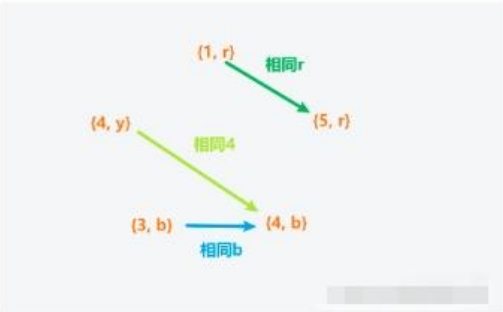
用例

输入	1 4 3 4 5 r y b b r
输出	3
说明	如果打（1, r）-> (5, r)，那么能打两张。 如果打（4，y) -> (4, b) -> (3, b)，那么能打三张。

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

题目解析

本题其实就是求{num, color}顶点的联通关系，相同num或者相同color的可以联通，因此题目用例得到如下图



可以发现，{4,y}、{4,b}、{3,b} 构成的连通图的顶点数最多，有三个，即为题解。

因此我们可以使用并查集处理本题。

关于并查集概念，可以看华为机试 - 发广播_伏城之外的博客-CSDN博客

一般并查集问题都会直接给出图中顶点之间的连接关系，而本题需要我们自己分析连接关系，分析规则即为：相同num或者相同color的顶点可以联通。

分析连接关系时，我们需要将每一个顶点和其余顶点进行连接判断。

Java算法源码

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

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

    String[] nums = sc.nextLine().split(" ");
    String[] colors = sc.nextLine().split(" ");

    System.out.println(getResult(nums, colors));
  }

  public static int getResult(String[] nums, String[] colors) {
    int n = nums.length;

    UnionFindSet ufs = new UnionFindSet(n);

    for (int i = 0; i < n; i++) {
      for (int j = i + 1; j < n; j++) {
        if (nums[i].equals(nums[j]) || colors[i].equals(colors[j])) {
          ufs.union(i, j);
        }
      }
    }

    HashMap<Integer, Integer> count = new HashMap<>();
    for (int i = 0; i < n; i++) {
      int f = ufs.find(i);
      count.put(f, count.getOrDefault(f, 0) + 1);
    }

    return count.values().stream().max((a, b) -> a - b).orElse(1);
  }
}

```

```java
class UnionFindSet {
  int[] fa;
  int count;

  public UnionFindSet(int n) {
    this.count = n;
    this.fa = new int[n];
    for (int i = 0; i < n; i++) this.fa[i] = i;
  }

  public int find(int x) {
    if (x != this.fa[x]) {
      return (this.fa[x] = this.find(this.fa[x]));
    }
    return x;
  }

  public void union(int x, int y) {
    int x_fa = this.find(x);
    int y_fa = this.find(y);

    if (x_fa != y_fa) {
      this.fa[y_fa] = x_fa;
      this.count--;
    }
  }
}
```

JS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 === 2) {
    const nums = lines[0].split(" ");
    const colors = lines[1].split(" ");

    console.log(getMaxCount(nums, colors));

    lines.length = 0;
  }
});
```

```
22
23 function getMaxCount(nums, colors) {
24   const n = nums.length;
25
26   const ufs = new UnionFindSet(n);
27
28   for (let i = 0; i < n; i++) {
29     for (let j = i + 1; j < n; j++) {
30       if (nums[i] === nums[j] || colors[i] === colors[j]) {
31         ufs.union(i, j);
32       }
33     }
34   }
35
36   const count = {};
37   for (let i = 0; i < n; i++) {
38     const f = ufs.find(i);
39     count[f] ? count[f]++ : (count[f] = 1);
40   }
41
42   return Object.values(count).sort((a, b) => b - a)[0];
43 }
44
```

```
45 // 并查集
46 class UnionFindSet {
47   constructor(n) {
48     this.fa = new Array(n).fill(0).map((_, i) => i);
49     this.count = n;
50   }
51
52   find(x) {
53     if (x !== this.fa[x]) {
54       return (this.fa[x] = this.find(this.fa[x]));
55     }
56     return x;
57   }
58
59   union(x, y) {
60     const x_fa = this.find(x);
61     const y_fa = this.find(y);
62
63     if (x_fa !== y_fa) {
64       this.fa[y_fa] = x_fa;
65       this.count--;
66     }
67   }
68 }
```

Python算法源码

```python
# 输入获取
nums = input().split()
colors = input().split()


# 并查集
class UnionFindSet:
    def __init__(self, n):
        self.fa = [idx for idx in range(n)]
        self.count = n

    def find(self, x):
        if x != self.fa[x]:
            self.fa[x] = self.find(self.fa[x])
            return self.fa[x]
        return x

    def union(self, x, y):
        x_fa = self.find(x)
        y_fa = self.find(y)

        if x_fa != y_fa:
            self.fa[y_fa] = x_fa
            self.count -= 1


# 算法入口
def getResult():
    n = len(nums)

    ufs = UnionFindSet(n)

    for i in range(n):
        for j in range(i+1, n):
            if nums[i] == nums[j] or colors[i] == colors[j]:
                ufs.union(i, j)

    count = {}
    for i in range(n):
        f = ufs.find(i)
        count[f] = count.get(f, 0) + 1

    return max(count.values())


# 算法调用
print(getResult())
```