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。

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

```
#include <iostream>
#include <sstream>
#include <vector>
using namespace std;

struct GameItem{
    int val;
    char color;
    GameItem(int num): val(num){}
};

class Solution {
public:
        // 排列，Time Complexity： O(n2)， Space Complexity： O(n)
        void maxTimes(vector<GameItem>& nums, vector<int>& hits, int& maxtime, int idx,int
curtime) { // nums[idx]表示出的牌
        int len = nums.size();
        for (int i = 0; i < len; i++) {
            if (i != idx && hits[i] == 0) { // i 未出牌
                if (nums[idx].val == nums[i].val || nums[idx].color == nums[i].color) {
                    hits[i] = 1;
                    curtime += 1;
                    if (curtime > maxtime) maxtime = curtime;
                    maxTimes(nums, hits, maxtime, i, curtime);
                    hits[i] = 0; // 还原
                    curtime--;
                }
            }
        }
    }

    int calculateMaxTimes(vector<GameItem>& nums) {
        int maxtime = 1, curtime = 1;
        int len = nums.size();
        vector<int> hits(len, 0); // 出牌则置 1，未出则置 0
        for (int i = 0; i < len; i++) {
            hits[i] = 1;
            maxTimes(nums, hits, maxtime, i, curtime);
```

```
                hits[i] = 0;
            }
            return maxtime;
        }
};


int main() {
    vector<GameItem> nums; // 保存牌
    string input;
    getline(cin, input);
    stringstream ss_num(input); // 输入数值序列
    string split;
    while (getline(ss_num, split, ' ')) {
        GameItem item(atoi(split.c_str()));
        nums.push_back(item);
    }


    getline(cin, input);
    stringstream ss_color(input); // 输入字母序列
    int i = 0;
    while (getline(ss_color, split, ' ')) {
        nums[i++].color = split.c_str()[0];
    }

    Solution sol;
    cout << sol.calculateMaxTimes(nums);

    return 0;
}
```