488. Zuma Game

Description | Submission | Solutions

- Total Accepted: **1952**
- Total Submissions: **5283**
- Difficulty: **Hard**
- Contributors: **ckcz123**

Think about Zuma Game. You have a row of balls on the table, colored red(R), yellow(Y), blue(B), green(G), and white(W). You also have several balls in your hand.

Each time, you may choose a ball in your hand, and insert it into the row (including the leftmost place and rightmost place). Then, if there is a group of 3 or more balls in the same color touching, remove these balls. Keep doing this until no more balls can be removed.

Find the minimal balls you have to insert to remove all the balls on the table. If you cannot remove all the balls, output -1.

Examples:

```
Input: "WRRBBW", "RB"
Output: -1
Explanation: WRRBBW -> WRR[R]BBW -> WBBW -> WBB[B]W -> WW

Input: "WWRRBBWW", "WRBRW"
Output: 2
Explanation: WWRRBBWW -> WWRR[R]BBWW -> WWBBWW -> WWBB[B]WW -> WWWW -> empty

Input:"G", "GGGGG"
Output: 2
Explanation: G -> G[G] -> GG[G] -> empty

Input: "RBYYBBRRB", "YRBGB"
Output: 3
```

Explanation: RBYYBBRRB -> RBYY[Y]BBRRB -> RBBBRRB -> RRRB -> B -> B[B] -> BB[B] -> empty

Note:

1. You may assume that the initial row of balls on the table won’t have any 3 or more consecutive balls with the same color.
2. The number of balls on the table won't exceed 20, and the string represents these balls is called "board" in the input.
3. The number of balls in your hand won't exceed 5, and the string represents these balls is called "hand" in the input.
4. Both input strings will be non-empty and only contain characters 'R','Y','B','G','W'.

```
#include<iostream>
#include<stdio.h>
#include<algorithm>
#include<unordered_set>
#include<string>
using namespace std;
// cancel continuous chars with nums >= 3
void reduce(string &s)
{
    int i=0,j=0;
    int n=s.size();
    while(i<n-2)
    {
        j=i;
        while(j<n && s[j]==s[i]) j++;
        if(j-i>=3)
        {
            s.erase(i,j-i);
            n = s.size();
            i -=2;
            if(i<0) i=0;
        }else{
            i=j;
        }
    }
}

int findMinStep(string board, string hand)
{
    int minCnt = INT_MAX;
```

```
    unordered_set<char> usehand;
    for(int i=0;i<(int)hand.size();i++)//for each char in hand
    {
        // we have already found one
        if(usehand.count(hand[i])) continue;// if we have tested this char
before, skip
        usehand.insert(hand[i]);
        for(int j=0;j<(int)board.size();j++)// for each char in board
        {
            if(hand[i]!=board[j]) continue;// the main idea is insert a char
from hand next to the same char in board
            string h1 = hand;
            string s1 = board;
            s1.insert(j,1,hand[i]);
            reduce(s1);
            h1.erase(i,1);
            if(s1.size()==0) return 1;
            int subcnt = findMinStep(s1,h1);// a new state
            if(subcnt==-1) continue;
            minCnt = min(subcnt+1,minCnt);
        }
    }
    return minCnt==INT_MAX?-1:minCnt;
}


int main(int argc,char *argv[])
{

    //"RBYYBBRRB", "YRBGB"
    string board = "RBYYBBRRB", hand = "YRBGB";
    //RBYYBBRRB -> RBYY[Y]BBRRB -> RBBBRRB -> RRRB -> B -> B[B] -> BB[B] ->
empty
    int ans = findMinStep(board,hand);
    cout << ans << endl;
    return 0;
}
```