Zuma Game

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'.

Solution 1

```java
public class Solution {
    public int findMinStep(String board, String hand) {
        List<Character> boardList = new ArrayList<Character>();
        for (char c : board.toCharArray()) {
            boardList.add(c);
        }
        Map<Character,Integer> handMap = new HashMap<>();
        handMap.put('R',0);
        handMap.put('Y',0);
        handMap.put('B',0);
        handMap.put('G',0);
        handMap.put('W',0);
        for (char h : hand.toCharArray()) {
            handMap.put(h, handMap.get(h) + 1);
        }
        return find(boardList, handMap);
    }

    private int find(List<Character> board, Map<Character, Integer> hand) {
        cleanupBoard(board);
        if (board.size() == 0) return 0;
        if (empty(hand)) return -1;
        int count = 0;
        int min = Integer.MAX_VALUE;
        for (int i = 0; i<board.size(); i++) {
            char c = board.get(i);
            count++;
            if (i == board.size() - 1 || board.get(i+1) != c) {
                int missing = 3 - count;
                if (hand.get(c) >= missing) {
                    hand.put(c, hand.get(c) - missing);
                    List<Character> smallerBoard = new ArrayList<>(board);
                    for (int j = 0; j<count; j++) {
                        smallerBoard.remove(i-j);
                    }
                    int smallerFind = find(smallerBoard, hand);
                    if ( smallerFind != -1 ) {
                        min = Math.min(smallerFind + missing, min);
                    }
                    hand.put(c, hand.get(c) + missing);
                }
                count = 0;
            }
        }
        return (min == Integer.MAX_VALUE) ? -1 : min;
    }

    private void cleanupBoard(List<Character> board) {
        int count = 0;
        boolean cleaned = false;
        for (int i = 0; i<board.size(); i++) {
            char c = board.get(i);
            count++;
            if (i == board.size() - 1 || board.get(i+1) != c) {
                if (count >= 3) {
                    for (int j = 0; j<count; j++) {
                        board.remove(i-j);
                    }
                    cleaned = true;
                    break;
                }
                count = 0;
            }
        }
        if (cleaned) {
            cleanupBoard(board);
        }
    }

    private boolean empty(Map<Character,Integer> hand) {
        for (int val : hand.values()) {
            if (val > 0) return false;
        }
        return true;
    }
}
```

written by ruben3 original link here

Solution 2

At the first look, it is similar toCan I Win. But not like that problem, the `board` state is not relevant to the state of `hand` and this problem requires the minimum value. So, it seems that memorization might not help much. So, I just do it brute-force and stop iff it is sure there no better results than current one.

Sort `hand` string so that we easily know if there at least 2 balls with same color in hand to eliminate a single ball on board. Refer to inline comments for details.

According to the description, the number of balls in hand won't exceed `5`, to make life easier, I just return a number equal or greater than `6` when there is no way to clear board.

```
#define MAX_STEP 6
class Solution {
public:
    int findMinStep(string board, string hand) {
        sort(hand.begin(), hand.end());
        int res = helper(board, hand);
        return res > hand.size() ? -1 : res;
    }

    int helper(string b, string h) {
        if (b.empty()) return 0;
        if (h.empty()) return MAX_STEP;
        int res = MAX_STEP;
        for (int i = 0; i < h.size(); i++) {
            int j = 0;
            int n = b.size();
            while (j < n) {
                int k = b.find(h[i], j);
                if (k == string::npos) break;
                if (k < n-1 && b[k] == b[k+1]) { // 2 consecutive balls with same color on board
                    string nextb = shrink(b.substr(0, k) + b.substr(k+2)); // shrink the string until no 3 or more consecutive balls in same color
                    if (nextb.empty()) return 1; // this is the best result for current board, no need to continue
                    string nexth = h;
                    nexth.erase(i, 1); // remove the used ball from hand
                    res = min(res, helper(nextb, nexth) + 1);
                    k++;
                }
                else if (i > 0 && h[i] == h[i-1]) { // 2 balls with same color in hand
                    string nextb = shrink(b.substr(0, k) + b.substr(k+1)); // shrink the string until no 3 or more consecutive balls in same color
                    if (nextb.empty()) return 2;  // this is the best result for current board, no need to continue
                    string nexth = h;
                    nexth.erase(i, 1); // remove the used balls from hand
                    nexth.erase(i-1, 1);
                    res = min(res, helper(nextb, nexth) + 2);
                }
                j = k + 1;
            }
        }
        return res;
    }

    string shrink(string s) {
        while(s.size() > 0) {
            int start = 0;
            bool done = true;
            for (int i = 0; i <= s.size(); i++) {
                if (i == s.size() || s[i] != s[start]) {
                    if (i - start >= 3) {
                        s = s.substr(0, start) + s.substr(i);
                        done = false;
                        break;
                    }
                    start = i;
                }
            }
            if (done) break;
        }
        return s;
    }
};
```

written by Hcisly original link here

Solution 3

Just searching and memorizing...
It is not a good interview question because many codes are not related to the algorithm.

```
public class Solution {
     public int findMinStep(String board, String hand) {
         HashMap<Character, int[]> mhand = new HashMap<Character, int[]>();
         mhand.put('R', new int[]{0});
         mhand.put('Y', new int[]{0});
         mhand.put('B', new int[]{0});
         mhand.put('G', new int[]{0});
         mhand.put('W', new int[]{0});
         for(char c:hand.toCharArray()){
                 mhand.get(c)[0]++;
         }
         HashMap<String, Integer> record = new HashMap<String, Integer>();
         int min = helper(board, mhand, record);
         if(min>=10000){
             return -1;
         } else {
             return min;
         }
     }

     int helper(String board, HashMap<Character, int[]> hand, HashMap<String, Int
eger> record){
         if(board.length()==0){
             return 0;
         } else {
             int min = 10000;
                for(int i=0;i<board.length();i++){
                    if(hand.get(board.charAt(i))[0]>0){
                        hand.get(board.charAt(i))[0]--;
                        String newboard = board.substring(0,i)+board.charAt(i)+bo
ard.substring(i);
                        newboard = further(newboard);
                        String c = code(newboard, hand);
                     if(record.containsKey(c)){
                      min = Math.min(min, 1+record.get(c));
                     } else {
                      int s = helper(newboard, hand, record);
                      min = Math.min(min, 1+s);
                      record.put(c, s);
                     }
                        hand.get(board.charAt(i))[0]++;
                    }
                }
             return min;
         }
     }
     String further(String board){
         if(board.length()==0){
             return "";
         }
```

```
        int count=1;
        for(int i=1;i<board.length();i++){
            if(board.charAt(i-1)==board.charAt(i)){
                count++;
            } else {
                if(count>=3){
                    return further(board.substring(0, i-count)+board.substring(i
));
                } else {
                    count=1;
                }
            }
        }
        if(count>=3){
            return board.substring(0, board.length()-count);
        }
        return board;
    }
    String code(String board, HashMap<Character, int[]> hand){
     StringBuilder sb = new StringBuilder();
     sb.append(board);
     for(Map.Entry<Character, int[]> e: hand.entrySet()){
      sb.append(e.getKey());
      sb.append(e.getValue()[0]);
     }
     return sb.toString();
    }
}
```

written by pureklkl original link here