题目描述

斗地主起源于湖北十堰房县，据说是一位叫吴修全的年轻人根据当地流行的扑克玩法"跑得快"改编的，如今已风靡整个中国，并流行于互联网上。

牌型：单顺，又称顺子，最少5张牌，最多12张牌(3…A)不能有2，也不能有大小王，不计花色。

例如： 3-4-5-6-7-8，7-8-9-10-J-Q，3-4-5-6-7-8-9-10-J-Q-K-A

可用的牌 3<4<5<6<7<8<9<10<J<Q<K<A<2<B(小王)<C(大王)，每种牌除大小王外有四种花色

(共有13×4+2张牌)

输入：

1. 手上有的牌
2. 已经出过的牌(包括对手出的和自己出的牌)

输出：

- 对手可能构成的最长的顺子(如果有相同长度的顺子，输出牌面最大的那一个)，
- 如果无法构成顺子，则输出 NO-CHAIN。

输入描述

输入的第一行为当前手中的牌

输入的第二行为已经出过的牌

输出描述

最长的顺子

用例

输入	3-3-3-3-4-4-5-5-6-7-8-9-10-J-Q-K-A 4-5-6-7-8-8-8
输出	9-10-J-Q-K-A
说明	无

输入	3-3-3-3-8-8-8-8 K-K-K-K
输出	NO-CHAIN
说明	剩余的牌无法构成顺子

题目解析

本题我的解题思路分为两步：

1. 求出对手的牌
2. 基于对手的牌，求最长顺子

首先，对手的牌 = 总牌 - 我的牌 - 已打出的牌

这里主要难点在于，如何记录牌面对应的牌数量。我的思路是：

定义一个数组count，将数组count的索引和牌面关联（定义一个字典mapToV），数组count的元素值就是对应牌面的数量。

这样可以得出一个数组：

```
// count每个索引值对应一个牌面值，count元素值就是对应牌面的数量
// 牌面值              3  4  5  6  7  8  9  10 J  Q  K  A    2  B  C
// 索引值              3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
int[] count = {0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4, 1, 1};
```

然后，就可以很简单的完成：对手的牌 = 总牌 - 我的牌 - 已打出的牌

比如用例1，对手的牌就可以表示为：

```
int[] count = {0, 0, 0, 1, 1, 2, 2, 0, 3, 3, 3, 3, 3, 3, 0, 4, 1, 1};
```

接下来我们可以定义一个 L 指针，作为顺子的左边界，L指针的运动范围是count数组的索引3~索引10。

> 因为，顺子只能由牌面3~牌面A组成，因此左边界起始位置是牌面3，即索引3。而顺子至少要由5张牌组成，因此，左边界的结束位置是牌面10，即索引10，对应的顺子是10,J,Q,K,A。

之后，定义一个临时右边界指针R，区间[L,R]之间就是顺子的范围，R的从L位置开始扫描：

- 如果count[R] >= 1，则可以加入顺子范围，之后R++
- 如果count[R] == 0，则顺子中断，此时，我们要看[L, R-1]的长度是否大于等于5，如果是，则是顺子，否则就不是顺子。

当顺子发生中断，则下一次L的扫描位置，应该是R+1，比如下面标红的范围，L=3，R=6，当R=7时，顺子中断，则下个顺子从L=4位置开始扫描的话，依旧不能组成顺子，因此我们应该让下个顺子的L直接跳到R+1=8的位置开始扫描。

```
int[] count = {0, 0, 0, 1, 1, 2, 2, 0, 3, 3, 3, 3, 3, 3, 0, 4, 1, 1};
```

最后，将最长的顺子输出即可。

需要注意的是，我们可以在上面过程中，实时保存最长顺子，当遇到同长度的顺子时，必然是后面的顺子更优。

JavaScript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 my = lines[0].split("-");
    const used = lines[1].split("-");
    console.log(getResult(my, used));

    lines.length = 0;
  }
});

function getResult(my, used) {
  // 牌面值 映射为 count列表索引值
  const mapToV = new Map([
    ["3", 3],
    ["4", 4],
    ["5", 5],
    ["6", 6],
    ["7", 7],
    ["8", 8],
    ["9", 9],
    ["10", 10],
    ["J", 11],
    ["Q", 12],
    ["K", 13],
    ["A", 14],
    ["2", 16],
    ["B", 17],
    ["C", 18],
  ]);
```

```
/* count每个索引值对应一个牌面值，count元素值就是对应牌面的数量
   牌面值              3  4  5  6  7  8  9  10 J  Q  K  A     2  B  C
   索引值              3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 */
const count = [0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4, 1, 1];

// count列表索引值 映射为 牌面值
const mapToK = new Map([
  [3, "3"],
  [4, "4"],
  [5, "5"],
  [6, "6"],
  [7, "7"],
  [8, "8"],
  [9, "9"],
  [10, "10"],
  [11, "J"],
  [12, "Q"],
  [13, "K"],
  [14, "A"],
  [16, "2"],
  [17, "B"],
  [18, "C"],
]);

// 总牌数 减去 自己手中牌数
for (let k of my) {
  count[mapToV.get(k)] -= 1;
}

// 总牌数 减去 已打出去的牌数
for (let k of used) {
  count[mapToV.get(k)] -= 1;
}

```

```
  let ans = "NO-CHAIN";
  let maxLen = 0;

  // l为顺子的左边界，[3,10]，即顺子的左边界值最少是count索引3，最多是count索引10
  let l = 3;
  while (l <= 10) {
    const tmp = [];
    for (let r = l; r < 16; r++) {
      // 如果对应牌数>=1，则可以组顺子
      if (count[r] >= 1) {
        tmp.push(mapToK.get(r));
      } else {
        // 如果对应牌数 == 0，则顺子中断
        // 顺子必须大于五张牌，且总是记录最长，遇到长度相同的，记录后面发现的顺子
        if (tmp.length >= 5 && tmp.length >= maxLen) {
          maxLen = tmp.length;
          ans = tmp.join("-");
        }
        // 顺子中断处+1，即为下一次顺子的起始位置
        l = r;
        break;
      }
    }
    l++;
  }

  return ans;
}
```

Java算法源码

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

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

    String[] my = sc.nextLine().split("-");
    String[] used = sc.nextLine().split("-");

    System.out.println(getResult(my, used));
  }

  public static String getResult(String[] my, String[] used) {
    // 牌面值 映射为 count列表索引值
    HashMap<String, Integer> mapToV = new HashMap<>();
    mapToV.put("3", 3);
    mapToV.put("4", 4);
    mapToV.put("5", 5);
    mapToV.put("6", 6);
    mapToV.put("7", 7);
    mapToV.put("8", 8);
    mapToV.put("9", 9);
    mapToV.put("10", 10);
    mapToV.put("J", 11);
    mapToV.put("Q", 12);
    mapToV.put("K", 13);
    mapToV.put("A", 14);
    mapToV.put("2", 16);
    mapToV.put("B", 17);
    mapToV.put("C", 18);

    // count每个索引值对应一个牌面值，count元素值就是对应牌面的数量
    // 牌面值            3  4  5  6  7  8  9  10 J  Q  K  A     2  B  C
    // 索引值            3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

```
38      int[] count = {0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4, 1, 1};
39
40      // count列表索引值 映射为 牌面值
41      HashMap<Integer, String> mapToK = new HashMap<>();
42      mapToK.put(3, "3");
43      mapToK.put(4, "4");
44      mapToK.put(5, "5");
45      mapToK.put(6, "6");
46      mapToK.put(7, "7");
47      mapToK.put(8, "8");
48      mapToK.put(9, "9");
49      mapToK.put(10, "10");
50      mapToK.put(11, "J");
51      mapToK.put(12, "Q");
52      mapToK.put(13, "K");
53      mapToK.put(14, "A");
54      mapToK.put(16, "2");
55      mapToK.put(17, "B");
56      mapToK.put(18, "C");
57
58      // 总牌数 减去 自己手中牌数
59      for (String k : my) {
60        count[mapToV.get(k)] -= 1;
61      }
62
63      // 总牌数 减去 已打出去的牌数
64      for (String k : used) {
65        count[mapToV.get(k)] -= 1;
66      }
67
```

```
68      String ans = "NO-CHAIN";
69      int maxLen = 0;
70
71      // l为顺子的左边界，[3,10]，即顺子的左边界值最少是count索引3，最多是count索引10
72      int l = 3;
73      while (l <= 10) {
74        ArrayList<String> tmp = new ArrayList<>();
75        StringJoiner sj = new StringJoiner("-");
76        for (int r = l; r < 16; r++) {
77          // 如果对应牌数>=1，则可以组顺子
78          if (count[r] >= 1) {
79            tmp.add(mapToK.get(r));
80            sj.add(mapToK.get(r));
81          } else {
82            // 如果对应牌数 == 0，则顺子中断
83            // 顺子必须大于五张牌，且总是记录最长，遇到长度相同的，记录后面发现的顺子
84            if (tmp.size() >= 5 && tmp.size() >= maxLen) {
85              maxLen = tmp.size();
86              ans = sj.toString();
87            }
88            // 顺子中断处+1，即为下一次顺子的起始位置
89            l = r;
90            break;
91          }
92        }
93        l++;
94      }
95
96      return ans;
97    }
98  }
```

Python算法源码

```python
# 输入获取
my = input().split("-")
used = input().split("-")


# 算法入口
def getResult():
    # 牌面值 映射为 count列表索引值
    mapToV = {
        "3": 3,
        "4": 4,
        "5": 5,
        "6": 6,
        "7": 7,
        "8": 8,
        "9": 9,
        "10": 10,
        "J": 11,
        "Q": 12,
        "K": 13,
        "A": 14,
        "2": 16,
        "B": 17,
        "C": 18
    }

    # count每个索引值对应一个牌面值，count元素值就是对应牌面的数量
    # 牌面值          3  4  5  6  7  8  9  10 J  Q  K  A     2  B  C
    # 索引值          3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
    count = [0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 4, 1, 1]

```

```
    # count列表索引值 映射为 牌面值
    mapToK = {
        3: "3",
        4: "4",
        5: "5",
        6: "6",
        7: "7",
        8: "8",
        9: "9",
        10: "10",
        11: "J",
        12: "Q",
        13: "K",
        14: "A",
        16: "2",
        17: "B",
        18: "C"
    }

    # 总牌数 减去 自己手中牌数
    for k in my:
        count[mapToV[k]] -= 1

    # 总牌数 减去 已打出去的牌数
    for k in used:
        count[mapToV[k]] -= 1

    ans = "NO-CHAIN"
    maxLen = 0

    # l为顺子的左边界，[3,10]，即顺子的左边界值最少是count索引3，最多是count索引10
    l = 3
    while l <= 10:
        tmp = []
        for r in range(l, 16):
            # 如果对应牌数>=1，则可以组顺子
            if count[r] >= 1:
                tmp.append(mapToK[r])
            # 如果对应牌数 == 0，则顺子中断
            else:
                # 顺子必须大于五张牌，且总是记录最长，遇到长度相同的，记录后面发现的顺子
                if len(tmp) >= 5 and len(tmp) >= maxLen:
                    maxLen = len(tmp)
                    ans = "-".join(tmp)
                # 顺子中断处+1，即为下一次顺子的起始位置
                l = r
                break
        l += 1

    return ans


# 算法调用
print(getResult())
```