最长的顺子题目描述：

斗地主起源于湖北十堰房县，据传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 4 种花色(共有 13X4 + 2 张牌)

输入 1. 手上已有的牌 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

题目描述：

斗地主起源于湖北十堰房县，据传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 4 种花色(共有 13X4 + 2 张牌)

输入 1. 手上已有的牌 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

```
package main


import (
    "fmt"
    "strings"
)


func main() {
    cards := map[string]int{}
```

```
    for {
        var s string
        n, _ := fmt.Scan(&s)
        if n == 0 {
            break
        }
        arr := strings.Split(s, "-")
        for _, card := range arr {
            cards[card] += 1
        }
    }

    r := longestShunzi(cards)
    fmt.Printf("%s\n", r)
}

func cardToint(card string) int {
    m := map[string]int{
        "3": 3,
        "4": 4,
        "5": 5,
        "6": 6,
```

```
        "7": 7,
        "8": 8,
        "9": 9,
        "10": 10,
        "J": 11,
        "Q": 12,
        "K": 13,
        "A": 14,
    }
    return m[card]
}

func intToCard(i int) string {
    m := map[int]string{
        3: "3",
        4: "4",
        5: "5",
        6: "6",
        7: "7",
        8: "8",
        9: "9",
        10: "10",
```

```
        11: "J",
        12: "Q",
        13: "K",
        14: "A",
    }
    return m[i]
}

func longestShunzi(cards map[string]int) string {
    arr := make([]bool, 15)
    for card, cnt := range cards {
        if cnt == 4 {
            arr[cardToint(card)] = true
        }
    }

    left, right := -1, -1
    for i := 14; i>=3;  {
        if arr[i] {
            i--
            continue
        }
```

```
        start := i
        for i>=3 && !arr[i] {
            i--
        }

        if start-(i+1) > right-left {
            left = i+1
            right = start
        }
    }

    if right - left < 4 {
        return "NO-CHAIN"
    }
    resArr := []string{}
    for i:=left; i<=right;i++ {
        resArr = append(resArr, intToCard(i))
    }
    return strings.Join(resArr, "-")
}
```