

【哈希表】-斗地主之顺子

视频直播讲解: [📺 2024/09/07 真题讲解 \(2024E卷\)](#)

题目描述与示例

题目描述

在斗地主扑克牌游戏中, 扑克牌由小到大的顺序为: 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A, 2。

玩家可以出的扑克牌阵型有: 单张、对子顺子、飞机、炸弹等。其中顺子的出牌规则为: 由至少 5 张由小到大连续递增的扑克牌组成, 且不能包含 2。

例如: {3, 4, 5, 6, 7}、{3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A} 都是有效的顺子; 而 {J, Q, K, A, 2}、{2, 3, 4, 5, 6}、{3, 4, 5, 6}、{3, 4, 5, 6, 8} 等都不是顺子。

给定一个包含 13 张牌的数组, 如果有满足出牌规则的顺子, 请输出顺子。

如果存在多个顺子, 请每行输出一个顺子, 且需要按顺子的第一张牌的大小 (必须从小到大) 依次输出。如果没有满足出牌规则的顺子, 请输出 No。

输入描述

13 张任意顺序的扑克牌, 每张扑克牌数字用空格隔开, 每张扑克牌的数字都是合法的, 并且不包括大小王。比如:

2 9 J 2 3 4 K A 7 9 A 5 6

不需要考虑输入为异常字符的情况

输出描述

组成的顺子, 每张扑克牌数字用空格隔开。比如

3 4 5 6 7

示例一

输入

```
1 2 9 J 2 3 4 K A 7 9 A 5 6
```

输出

```
1 3 4 5 6 7
```

说明

13 张牌中，可以组成的顺子只有 1 组： 3 4 5 6 7

示例二

输入

```
1 2 9 J 10 3 4 K A 7 Q A 5 6
```

输出

```
1 3 4 5 6 7
2 9 10 J Q K A
```

说明

13 张牌中，可以组成 2 组顺子，从小到大分别为： 3 4 5 6 7 和 9 10 J Q K A

示例三

输入

```
1 2 9 9 9 3 4 K A 10 Q A 5 6
```

输出

```
1 No
```

说明

13 张牌中，无法组成顺子

解题思路

题意理解以及补充

题目描述非常不清楚的，对于一些特殊情况没有详细说明。只能够通过考试过程中自行理解测试并进行优化。

本篇题解最终呈现的代码能够通过 95% 的用例。

对于例子

```
1 3 4 5 6 7 4 5 6 7 8 9 10 J
```

在实际考试中，实测应该要求输出

```
1 3 4 5 6 7
2 4 5 6 7 8 9 10 J
```

而不是

```
1 3 4 5 6 7 8 9 10 J
```

这个例子说明，**当所给用例既可以凑成单个长顺子或者多个顺子的时候，应该优先凑成多个顺子。**

对于例子

```
1 3 4 5 6 7 3 4 5 6 7 A A A
```

在实际考试中，实测应该要求输出

```
1 3 4 5 6 7
2 3 4 5 6 7
```

而不是

```
1 3 4 5 6 7
```

这个例子说明，**每一张牌只可以使用一次，但如果能够凑出多个顺子需要尽量去使用。**

对于例子

```
1 3 4 5 6 7 3 4 5 6 7 8 A A
```

在实际考试中，实测应该要求输出

```
1 3 4 5 6 7 8
```

```
2 3 4 5 6 7
```

而不是

```
1 3 4 5 6 7
2 3 4 5 6 7 8
```

这个例子说明，当出现多个顺子的起始位置相等的时候，应该先输出长度更长的顺子。

上述几点，在题目中都没有说明，只能根据具体的代码通过情况来反推。

另外，由于题目指出输入的牌数一定是 13 张牌，这意味着输出的顺子数量一定只有 1 个或者 2 个（即输出的行数只有 1 行或者 2 行）

利用哈希表求下一张牌

如果顺子都是数字，那么我们处理顺子问题就非常方便。

假设某张牌对应的数字是 `num`，那么其下一张牌就是 `num+1`。

但题目有一个较难处理的地方，是牌为 J、Q、K 和 A 的情况。

为了应对字母和数字混合出现的情况，我们可以构建一个哈希表 `next_card_dic`。

```
1 next_card_dic = {str(num): str(num+1) for num in range(3, 10)}
2 next_card_dic["10"] = "J"
3 next_card_dic["J"] = "Q"
4 next_card_dic["Q"] = "K"
5 next_card_dic["K"] = "A"
```

实际上，`next_card_dic` 就是形如以下结构的哈希表

```
1 next_card_dic = {
```

```
2 '3': '4',
3 '4': '5',
4 '5': '6',
5 '6': '7',
6 '7': '8',
7 '8': '9',
8 '9': '10',
9 '10': 'J',
10 'J': 'Q',
11 'Q': 'K',
12 'K': 'A'
13 }
```

如果我们知道当前卡牌是 `card`，`card` 是顺子中的一张牌，那么下一张牌就是 `next_card_dic[card]`。

这个哈希表不大，手动构建也行。

利用哈希表统计牌数

在前面题意理解中提到，每一张牌只能够使用一次。所以我们可以用一个哈希表计数器 `card_cnt` 来统计每一张牌各有多少张，并且在凑成顺子之后减去这些牌的数量要相应减少。

```
1 from collections import Counter
2 card_cnt = Counter(cards)
```

枚举初始牌的框架

在初始化 `card_cnt` 之后，我们就可以计算顺子了。

因为最大且最短的顺子是 `10 J Q K A`，显然顺子的第一张牌的范围是 `3` 到 `10`。

我们可以枚举初始牌 `start` 的范围为 `3` 到 `10`，如果我们使用 `start` 作为顺子的初始牌，能否构建出顺子。

因此可以构建出如下的代码框架

```

1 # 设置初始牌为3，在循环中会递增
2 # 设置标记flag表示选择特定初始牌的时候，是否找到对应的顺子
3 start = 3
4 flag = True
5
6 # 枚举初始牌start，其大小不可能超过10
7 # 先枚举出长度为5的顺子
8 while start <= 10:
9     # 调用check()函数，
10    # 如果能够构建出长度为5的顺子
11    # 则ans会更新，且返回True
12    # 如果不能构建出顺子
13    # 则ans不会修改，且返回False
14    flag = check(start, card_cnt, next_card_dic, ans)
15    # 如果flag为False，说明当前start不再顺子作为初始牌使用，start递增
16    # 如果flag为True，说明start还有可能继续作为顺子的初始牌使用，start不修改
17    if flag is False:
18        start += 1

```

在后面的讲解我们会看到，`check()` 函数是用于计算特定顺子是否存在的函数。

如果以 `start` 为起始牌的顺子存在，则 `check()` 函数会返回 `True`，否则将返回 `False`。返回的结果会传参给 `flag`。

由于可能出现多个顺子均为同一个 `start` 的情况，如例子

```

1 3 4 5 6 7 3 4 5 6 7 A A A

```

要求输出两个顺子

```

1 3 4 5 6 7
2 3 4 5 6 7

```

因此如果计算出 `flag` 为 `True` 的时候，我们仍然不能排除 `start` 仍可能作为初始牌的情况。

因此只有当 `flag` 为 `False` 的时候，我们才递增 `start`。

计算特定顺子的函数

假设我们想知道，以某张牌 `start` 作为起始牌的顺子是否存在以及这个顺子是什么，我们可以构建如下的一个 `check()` 函数。

```
1 # 检查以start为初始牌的顺子是否存在
2 # card_cnt为表示当前牌剩余频率的哈希表
3 # next_card_dic为表示下一张牌的哈希表
4 # ans为储存顺子的答案列表
5 def check(start, cards_cnt, next_card_dic, ans):
6     # res储存顺子的结果，初始化为空列表
7     res = list()
8     # card表示当前牌，初始化为初始牌，取字符串形式
9     card = str(start)
10    # 严格循环5次，先找长度为5的顺子
11    for _ in range(5):
12        # 如果当前牌的张数大于0，则可以延长
13        if cards_cnt[card] > 0:
14            res.append(card)
15        # 否则退出循环
16        else:
17            break
18        # 如果当前牌不为"A"，则令card为其下一张牌
19        # 这只可能出现在start = 10的时候
20        if card != "A":
21            card = next_card_dic[card]
22    # 在退出上述循环后，如果res的长度为5
23    # 说明找到了一个长度为5的顺子，
24    if len(res) == 5:
25        # 将这些牌在card_cnt中的频率-1
26        for card in res:
27            cards_cnt[card] -= 1
28        # 将res存入ans，同时返回True表示找到了顺子
29        ans.append(res)
30        return True
31    # 如果res长度不足5，则返回False
32    return False
```

其中 `ans` 为储存最终答案的二维列表。

我们将这个以 `start` 为起始牌的顺子储存在列表 `res` 中。

5 是顺子的最小长度。

这里我们只循环 5 次的原因在于，这个顺子虽然可能不止这么长，但是为了**尽可能多地凑出更多顺子**，我们先暂时凑出长度为 5 的顺子，然后在所有顺子都考虑完毕之后，再考虑这些顺子能够进一步延长。

即对于例子

```
1 3 4 5 6 7 8 5 6 7 8 9 10 J
```

虽然其最终答案为

```
1 3 4 5 6 7 8
2 5 6 7 8 9 10 J
```

但在这一步我们必须**先多凑出顺子**，先算出两个长度为 5 的顺子

```
1 3 4 5 6 7
2 5 6 7 8 9
```

再在后续进一步延长这两个顺子得到最终答案。

顺子延长以及输出

在起始牌 `start` 的 `while` 循环遍历结束之后，我们需要再次检查 `ans` 数组中的每一个长度为 5 的顺子是否还能够使用 `card_cnt` 中的牌进行延长。

可以再次抽象出函数 `extend_res(res)`，对单个顺子 `res` 进行延长。

```
1 # 在获得所有长度为5的顺子之后，延长顺子的函数
2 def extend_res(res, cards_cnt, next_card_dic):
3     # 取顺子的最后一张牌res[-1]为end_card
4     end_card = res[-1]
5     # 如果end_card不为"A"，且其下一张牌next_card_dic[end_card]的频率大于0
6     while end_card != "A" and cards_cnt[next_card_dic[end_card]] > 0:
```

```

7         # 将下一张牌更新为end_card
8         end_card = next_card_dic[end_card]
9         # 下一张牌的频率-1
10        cards_cnt[end_card] -= 1
11        # res中加入下一张牌
12        res.append(end_card)

```

其中 `end_card` 是当前顺子 `res` 中的最后一张牌。

当 `end_card` 不为 "A"（为 "A" 则不存在下一张牌），且其下一张牌 `next_card_dic[end_card]` 的出现次数 `card_cnt[next_card_dic[end_card]]` 大于 0 时，则说明其下一张牌可以延长到当前顺子 `res` 中。

```

1  # 退出上述枚举之后，考虑ans的长度
2  # 若为0则说明不存在顺子，输出No
3  if len(ans) == 0:
4      print("No")
5  # 否则进行顺子的延长和输出
6  else:
7      # 对于ans中的每一个顺子res，都调用extend_res()函数进行延长
8      # 注意枚举的res是一维列表，所以extend_res()修改res是修改同一个对象
9      # 这个修改是对res的引用的修改，对函数外可见
10     for res in ans:
11         extend_res(res, card_cnt, next_card_dic)
12     # 按照先长度从小到大，后初始值从小到大，对res进行排序
13     ans.sort(key = lambda res: (len(res), int(res[0])))
14     # 输出每一个顺子，每个一行
15     for res in ans:
16         print(" ".join(res))

```

代码

Python

```

1  # 题目：【哈希表】2024E-斗地主之顺子

```

```

2 # 分值: 100
3 # 作者: 许老师-闭着眼睛学数理化
4 # 算法: 哈希表, 模拟
5 # 代码看不懂的地方, 请直接在群上提问
6
7
8 from collections import Counter
9
10
11 # 检查已start为初始牌的顺子是否存在
12 # card_cnt为表示当前牌剩余频率的哈希表
13 # next_card_dic为表示下一张牌的哈希表
14 # ans为储存顺子的答案列表
15 def check(start, cards_cnt, next_card_dic, ans):
16     # res储存顺子的结果, 初始化为空列表
17     res = list()
18     # card表示当前牌, 初始化为初始牌, 取字符串形式
19     card = str(start)
20     # 严格循环5次, 先找长度为5的顺子
21     for _ in range(5):
22         # 如果当前牌的张数大于0, 则可以延长
23         if cards_cnt[card] > 0:
24             res.append(card)
25         # 否则退出循环
26         else:
27             break
28         # 如果当前牌不为"A", 则令card为其下一张牌
29         # 这只可能出现在start = 10的时候
30         if card != "A":
31             card = next_card_dic[card]
32     # 在退出上述循环后, 如果res的长度为5
33     # 说明找到了一个长度为5的顺子,
34     if len(res) == 5:
35         # 将这些牌在card_cnt中的频率-1
36         for card in res:
37             cards_cnt[card] -= 1
38         # 将res存入ans, 同时返回True表示找到了顺子
39         ans.append(res)
40         return True
41     # 如果res长度不足5, 则返回False
42     return False
43
44
45 # 在获得所有长度为5的顺子之后, 延长顺子的函数
46 def extend_res(res, cards_cnt, next_card_dic):
47     # 取顺子的最后一张牌res[-1]为end_card
48     end_card = res[-1]

```

```

49     # 如果end_card不为"A", 且其下一张牌next_card_dic[end_card]的频率大于0
50     while end_card != "A" and cards_cnt[next_card_dic[end_card]] > 0:
51         # 将下一张牌更新为end_card
52         end_card = next_card_dic[end_card]
53         # 下一张牌的频率-1
54         cards_cnt[end_card] -= 1
55         # res中加入下一张牌
56         res.append(end_card)
57
58
59 cards = input().split()
60 # 获得当前所有13张牌的出现频率
61 card_cnt = Counter(cards)
62
63 # 构建下一张牌的哈希表next_card_dic
64 # 如果已知当前牌为card,
65 # 那么可以通过该哈希表得到在顺子中的下一张牌为next_card_dic[card]
66 next_card_dic = {str(num): str(num+1) for num in range(3, 10)}
67 next_card_dic["10"] = "J"
68 next_card_dic["J"] = "Q"
69 next_card_dic["Q"] = "K"
70 next_card_dic["K"] = "A"
71
72 # 初始化答案列表
73 ans = list()
74
75 # 设置初始牌为3, 在循环中会递增
76 # 设置标记flag表示选择特定初始牌的时候, 是否找到对应的顺子
77 start = 3
78 flag = True
79
80 # 枚举初始牌start, 其大小不可能超过10
81 # 先枚举出长度为5的顺子
82 while start <= 10:
83     # 调用check()函数,
84     # 如果能够构建出长度为5的顺子
85     # 则ans会更新, 且返回True
86     # 如果不能构建出顺子
87     # 则ans不会修改, 且返回False
88     flag = check(start, card_cnt, next_card_dic, ans)
89     # 如果flag为False, 说明当前start不再顺子作为初始牌使用, start递增
90     # 如果flag为True, 说明start还有可能继续作为顺子的初始牌使用, start不修改
91     if flag is False:
92         start += 1
93
94
95 # 退出上述枚举之后, 考虑ans的长度

```

```

96 # 若为0则说明不存在顺子，输出No
97 if len(ans) == 0:
98     print("No")
99 # 否则进行顺子的延长和输出
100 else:
101     # 对于ans中的每一个顺子res，都调用extend_res()函数进行延长
102     # 注意枚举的res是一维列表，所以extend_res()修改res是修改同一个对象
103     # 这个修改是对res的引用的修改，对函数外可见
104     for res in ans:
105         extend_res(res, card_cnt, next_card_dic)
106     # 按照先长度从小到大，后初始值从小到大，对res进行排序
107     ans.sort(key = lambda res: (len(res), int(res[0])))
108     # 输出每一个顺子，每个一行
109     for res in ans:
110         print(" ".join(res))

```

Java

```

1 import java.util.*;
2
3 public class Main {
4
5     // 检查以start为初始牌的顺子是否存在
6     // cardCnt为表示当前牌剩余频率的哈希表
7     // nextCardDic为表示下一张牌的哈希表
8     // ans为储存顺子的答案列表
9     public static boolean check(int start, Map<String, Integer> cardsCnt,
10 Map<String, String> nextCardDic, List<List<String>> ans) {
11         // res储存顺子的结果，初始化为空列表
12         List<String> res = new ArrayList<>();
13         // card表示当前牌，初始化为初始牌，取字符串形式
14         String card = String.valueOf(start);
15         // 严格循环5次，先找长度为5的顺子
16         for (int i = 0; i < 5; i++) {
17             // 如果当前牌的张数大于0，则可以延长
18             if (cardsCnt.getOrDefault(card, 0) > 0) {
19                 res.add(card);
20             } else {
21                 // 否则退出循环
22                 break;
23             }
24             // 如果当前牌不为"A"，则令card为其下一张牌
25             // 这只可能出现在start = 10的时候
26             if (!card.equals("A")) {
27                 card = nextCardDic.get(card);
28             }
29         }
30         ans.add(res);
31         return true;
32     }
33 }

```

```

27         }
28     }
29     // 在退出上述循环后, 如果res的长度为5
30     // 说明找到了一个长度为5的顺子
31     if (res.size() == 5) {
32         // 将这些牌在cardCnt中的频率-1
33         for (String c : res) {
34             cardsCnt.put(c, cardsCnt.get(c) - 1);
35         }
36         // 将res存入ans, 同时返回true表示找到了顺子
37         ans.add(res);
38         return true;
39     }
40     // 如果res长度不足5, 则返回false
41     return false;
42 }
43
44 // 在获得所有长度为5的顺子之后, 延长顺子的函数
45 public static void extendRes(List<String> res, Map<String, Integer>
cardsCnt, Map<String, String> nextCardDic) {
46     // 取顺子的最后一张牌res.get(res.size() - 1)为endCard
47     String endCard = res.get(res.size() - 1);
48     // 如果endCard不为"A", 且其下一张牌nextCardDic[endCard]的频率大于0
49     while (!endCard.equals("A") &&
cardsCnt.getOrDefault(nextCardDic.get(endCard), 0) > 0) {
50         // 将下一张牌更新为endCard
51         endCard = nextCardDic.get(endCard);
52         // 下一张牌的频率-1
53         cardsCnt.put(endCard, cardsCnt.get(endCard) - 1);
54         // res中加入下一张牌
55         res.add(endCard);
56     }
57 }
58
59 public static void main(String[] args) {
60     Scanner scanner = new Scanner(System.in);
61     String[] cards = scanner.nextLine().split(" ");
62
63     // 获得当前所有13张牌的出现频率
64     Map<String, Integer> cardsCnt = new HashMap<>();
65     for (String card : cards) {
66         cardsCnt.put(card, cardsCnt.getOrDefault(card, 0) + 1);
67     }
68
69     // 构建下一张牌的哈希表nextCardDic
70     // 如果已知当前牌为card,
71     // 那么可以通过该哈希表得到在顺子中的下一张牌为nextCardDic[card]

```

```

72     Map<String, String> nextCardDic = new HashMap<>();
73     for (int num = 3; num <= 9; num++) {
74         nextCardDic.put(String.valueOf(num), String.valueOf(num + 1));
75     }
76     nextCardDic.put("10", "J");
77     nextCardDic.put("J", "Q");
78     nextCardDic.put("Q", "K");
79     nextCardDic.put("K", "A");
80
81     // 初始化答案列表
82     List<List<String>> ans = new ArrayList<>();
83
84     // 设置初始牌为3, 在循环中会递增
85     int start = 3;
86     boolean flag = true;
87
88     // 枚举初始牌start, 其大小不可能超过10
89     // 先枚举出长度为5的顺子
90     while (start <= 10) {
91         // 调用check()函数,
92         // 如果能够构建出长度为5的顺子
93         // 则ans会更新, 且返回true
94         flag = check(start, cardsCnt, nextCardDic, ans);
95         // 如果flag为false, 说明当前start不再顺子作为初始牌使用, start递增
96         if (!flag) {
97             start++;
98         }
99     }
100
101     // 退出上述枚举之后, 考虑ans的长度
102     // 若为0则说明不存在顺子, 输出No
103     if (ans.isEmpty()) {
104         System.out.println("No");
105     } else {
106         // 否则进行顺子的延长和输出
107         for (List<String> res : ans) {
108             extendRes(res, cardsCnt, nextCardDic);
109         }
110         // 按照先长度从小到大, 后初始值从小到大, 对res进行排序
111         ans.sort((res1, res2) -> {
112             int len1 = res1.size();
113             int len2 = res2.size();
114             if (len1 != len2) {
115                 return Integer.compare(len1, len2);
116             } else {
117                 return Integer.compare(Integer.parseInt(res1.get(0)),
Integer.parseInt(res2.get(0)));

```

```

118         }
119     });
120     // 输出每一个顺子, 每个一行
121     for (List<String> res : ans) {
122         System.out.println(String.join(" ", res));
123     }
124 }
125
126 scanner.close();
127 }
128 }
129

```

C++

```

1  #include <iostream>
2  #include <vector>
3  #include <unordered_map>
4  #include <string>
5  #include <algorithm>
6
7  using namespace std;
8
9  // 检查以start为初始牌的顺子是否存在
10 // cardsCnt为表示当前牌剩余频率的哈希表
11 // nextCardDic为表示下一张牌的哈希表
12 // ans为储存顺子的答案列表
13 bool check(int start, unordered_map<string, int>& cardsCnt,
14            unordered_map<string, string>& nextCardDic, vector<vector<string>>& ans) {
15     // res储存顺子的结果, 初始化为空列表
16     vector<string> res;
17     // card表示当前牌, 初始化为初始牌, 取字符串形式
18     string card = to_string(start);
19     // 严格循环5次, 先找长度为5的顺子
20     for (int i = 0; i < 5; i++) {
21         // 如果当前牌的张数大于0, 则可以延长
22         if (cardsCnt[card] > 0) {
23             res.push_back(card);
24         } else {
25             // 否则退出循环
26             break;
27         }
28         // 如果当前牌不为"A", 则令card为其下一张牌
29         // 这只可能出现在start = 10的时候
30         if (card != "A") {

```



```

30         card = nextCardDic[card];
31     }
32 }
33 // 在退出上述循环后, 如果res的长度为5
34 // 说明找到了一个长度为5的顺子
35 if (res.size() == 5) {
36     // 将这些牌在cardsCnt中的频率-1
37     for (const string& c : res) {
38         cardsCnt[c]--;
39     }
40     // 将res存入ans, 同时返回true表示找到了顺子
41     ans.push_back(res);
42     return true;
43 }
44 // 如果res长度不足5, 则返回false
45 return false;
46 }
47
48 // 在获得所有长度为5的顺子之后, 延长顺子的函数
49 void extendRes(vector<string>& res, unordered_map<string, int>& cardsCnt,
    unordered_map<string, string>& nextCardDic) {
50     // 取顺子的最后一张牌res.back()为endCard
51     string endCard = res.back();
52     // 如果endCard不为"A", 且其下一张牌nextCardDic[endCard]的频率大于0
53     while (endCard != "A" && cardsCnt[nextCardDic[endCard]] > 0) {
54         // 将下一张牌更新为endCard
55         endCard = nextCardDic[endCard];
56         // 下一张牌的频率-1
57         cardsCnt[endCard]--;
58         // res中加入下一张牌
59         res.push_back(endCard);
60     }
61 }
62
63 int main() {
64     string line;
65     getline(cin, line);
66
67     // 将输入的牌以空格分割
68     vector<string> cards;
69     string card;
70     for (char ch : line) {
71         if (ch == ' ') {
72             cards.push_back(card);
73             card.clear();
74         } else {
75             card.push_back(ch);

```

```

76     }
77 }
78 if (!card.empty()) cards.push_back(card);
79
80 // 获得当前所有13张牌的出现频率
81 unordered_map<string, int> cardsCnt;
82 for (const string& c : cards) {
83     cardsCnt[c]++;
84 }
85
86 // 构建下一张牌的哈希表nextCardDic
87 // 如果已知当前牌为card,
88 // 那么可以通过该哈希表得到在顺子中的下一张牌为nextCardDic[card]
89 unordered_map<string, string> nextCardDic;
90 for (int num = 3; num <= 9; num++) {
91     nextCardDic[to_string(num)] = to_string(num + 1);
92 }
93 nextCardDic["10"] = "J";
94 nextCardDic["J"] = "Q";
95 nextCardDic["Q"] = "K";
96 nextCardDic["K"] = "A";
97
98 // 初始化答案列表
99 vector<vector<string>> ans;
100
101 // 设置初始牌为3, 在循环中会递增
102 int start = 3;
103 bool flag = true;
104
105 // 枚举初始牌start, 其大小不可能超过10
106 // 先枚举出长度为5的顺子
107 while (start <= 10) {
108     // 调用check()函数,
109     // 如果能够构建出长度为5的顺子
110     // 则ans会更新, 且返回true
111     flag = check(start, cardsCnt, nextCardDic, ans);
112     // 如果flag为false, 说明当前start不再顺子作为初始牌使用, start递增
113     if (!flag) {
114         start++;
115     }
116 }
117
118 // 退出上述枚举之后, 考虑ans的长度
119 // 若为0则说明不存在顺子, 输出No
120 if (ans.empty()) {
121     cout << "No" << endl;
122 } else {

```

```

123         // 否则进行顺子的延长和输出
124         for (auto& res : ans) {
125             extendRes(res, cardsCnt, nextCardDic);
126         }
127         // 按照先长度从小到大, 后初始值从小到大, 对res进行排序
128         sort(ans.begin(), ans.end(), [](const vector<string>& res1, const
vector<string>& res2) {
129             if (res1.size() != res2.size()) {
130                 return res1.size() < res2.size();
131             } else {
132                 return stoi(res1[0]) < stoi(res2[0]);
133             }
134         });
135         // 输出每一个顺子, 每个一行
136         for (const auto& res : ans) {
137             for (const string& c : res) {
138                 cout << c << " ";
139             }
140             cout << endl;
141         }
142     }
143
144     return 0;
145 }
146

```

时空复杂度

时间复杂度： $O(5N)$ 。此处 $N = 13$ ，每次调用 `check()` 函数都需要循环 5 次。可以认为是常数级别。

空间复杂度： $O(N)$ 。哈希表所占空间。可以认为是常数级别。