22、斗地主顺子，考点 or 实现——数据结构/栈

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：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：3 4 5 6 7

用例

输入	2 9 J 2 3 4 K A 7 9 A 5 6
输出	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。

输入	2 9 J 10 3 4 K A 7 Q A 5 6
输出	3 4 5 6 7 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

输入	2 9 9 9 3 4 K A 10 Q A 5 6
输出	No
说明	13张牌中，无法组成顺子。

题目解析

先看一个例子：

> 3 4 5 6 6 7 7 8 9 10

这个例子该输出什么呢?

如果是优先最长顺子的话，那么应该输出：

> 3 4 5 6 7 8 9 10

如果是优先最多顺子的话，那么应该输出：

> 3 4 5 6 7
> 6 7 8 9 10

经过考友实际机试验证，本题应该优先输出最多的顺子。

我的解题策略如下：

首先定义一个长度为15的数组count，分别记录2~A扑克牌的数量，其中2~A扑克牌分别对应索引2~14。

再定义一个total变量，记录总牌数。

然后，开始找五张顺子，查找索引 i 范围依次是：3~7，4~8，5~9，6~10，...，10~14

如果对应范围的每一个count[i] 都大于 0，则对应范围可以形成五张顺子，只要形成五张顺子，则

- total -= 5
- 范围内每一个 count[i] -= 1

当把五张顺子找完后，继续检查total 是否大于 0，若total > 0，则说明还有剩余牌，此时我们外层遍历每一个顺子，内层遍历每一个剩余牌，如果剩余牌可以追加到顺子尾巴，形成新顺子，则更新顺子尾巴。

最后，打印每一个顺子即可。

注意，由于我们找五张顺子时，是按照从左到右查找的，因此顺子找出来后就是符合要求顺序的，因此不需要额外排序。

优先最多顺子解法

Java算法源码

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

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String[] cards = sc.nextLine().split(" ");
    getResult(cards);
  }

  public static void getResult(String[] cards) {
    // 总牌数
    int total = cards.length;

    // 记录每张牌的数量
    int[] count = new int[15];
    for (String card : cards) {
      int num = mapToNum(card);
      count[num]++;
    }

    // 记录顺子，顺子表示为数组，数组含义：[起始牌，结束牌]
    ArrayList<int[]> straights = new ArrayList<>();

    for (int i = 3; i <= 10; i++) {
```

```
    for (int i = 3; i <= 10; i++) {
      // 先求五张的顺子
      if (isStraight(count, i, i + 4)) {
        // 如果可以组成五张顺子，则记录该顺子
        straights.add(new int[] {i, i + 4});
        // 总牌数减5
        total -= 5;
        // 对应的牌的数量减1
        for (int j = i; j <= i + 4; j++) count[j]--;
      }
    }

    // 如果五张的顺子求解完后，还有剩余牌
    if (total > 0) {
      // 则尝试将剩余牌追加到五张顺子后面
      for (int[] straight : straights) {
        for (int i = 3; i <= 14; i++) {
          if (count[i] > 0 && i - straight[1] == 1) {
            straight[1] = i;
            count[i]--;
          }
        }
      }
    }

    // 如果没有顺子，则返回No
    if (straights.size() == 0) {
      System.out.println("No");
      return;
    }
```

```
    }

    // 如果有顺子，则打印顺子
    for (int[] straight : straights) {
      int start = straight[0];
      int end = straight[1];

      StringJoiner sj = new StringJoiner(" ");
      for (int i = start; i <= end; i++) sj.add(mapToCard(i));
      System.out.println(sj);
    }
  }

  public static boolean isStraight(int[] count, int start, int end) {
    int i = start;
    for (; i <= end; i++) {
      if (count[i] == 0) break;
    }
    return i > end;
  }

  public static int mapToNum(String card) {
    switch (card) {
      case "J":
        return 11;
      case "Q":
        return 12;
      case "K":
        return 13;
      case "A":
```

```
      case "A":
        return 14;
      default:
        return Integer.parseInt(card);
    }
  }

  public static String mapToCard(int num) {
    switch (num) {
      case 11:
        return "J";
      case 12:
        return "Q";
      case 13:
        return "K";
      case 14:
        return "A";
      default:
        return num + "";
    }
  }
}
```

JS算法源码

```
/* 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,
});

rl.on("line", (line) => {
  const cards = line.split(" ");
  getResult(cards);
});

function getResult(cards) {
  // 总牌数
  let total = cards.length;

  // 记录每张牌的数量
  const count = new Array(15).fill(0);
  for (let card of cards) {
    const num = mapToNum(card);
    count[num]++;
  }

  // straights记录顺子，其元素是顺子，顺子表示为数组：[起始牌，结束牌]
  const straights = [];

  // i 表示顺子的起始牌取值
```

```
  // i 表示顺子的起始牌取值
  for (let i = 3; i <= 10; i++) {
    // 先求五张的顺子
    if (isStraights(count, i, i + 4)) {
      // 如果可以组成五张顺子，则记录该顺子
      straights.push([i, i + 4]);
      // 总牌数减5
      total -= 5;
      // 对应的牌的数量减1
      for (let j = i; j <= i + 4; j++) count[j]--;
    }
  }

  // 如果五张的顺子求解完后，还有剩余牌
  if (total > 0) {
    // 则尝试将剩余牌追加到五张顺子后面
    for (let straight of straights) {
      for (let i = 3; i <= 14; i++) {
        if (count[i] > 0 && i - straight[1] == 1) {
          straight[1] = i;
          count[i]--;
        }
      }
    }
  }

  // 如果没有顺子，则返回No
  if (straights.length == 0) {
    console.log("No");
    console.log("No");
    return;
  }

  // 如果有顺子，则打印顺子
  for (let [start, end] of straights) {
    const ans = [];
    for (let i = start; i <= end; i++) ans.push(mapToCard(i));
    console.log(ans.join(" "));
  }
}

function isStraights(count, start, end) {
  let i = start;
  for (; i <= end; i++) {
    if (count[i] == 0) break;
  }
  return i > end;
}

function mapToNum(card) {
  switch (card) {
    case "J":
      return 11;
    case "Q":
      return 12;
    case "K":
      return 13;
    case "A":
      return 14;
```

```
      return 14;
    default:
      return parseInt(card);
  }
}

function mapToCard(num) {
  switch (num) {
    case 11:
      return "J";
    case 12:
      return "Q";
    case 13:
      return "K";
    case 14:
      return "A";
    default:
      return num + "";
  }
}
```

Python算法源码

```
# 输入获取
cards = input().split()


def mapToCard(num):
    if num == 11:
        return "J"
    elif num == 12:
        return "Q"
    elif num == 13:
        return "K"
    elif num == 14:
        return "A"
    else:
        return str(num)


def mapToNum(card):
    if card == "J":
        return 11
    elif card == "Q":
        return 12
    elif card == "K":
        return 13
    elif card == "A":
        return 14
    else:
        return int(card)
```

```
        return int(card)


def isStraights(count, start, end):
    i = start
    while i <= end:
        if count[i] == 0:
            break
        else:
            i += 1
    return i > end


# 算法入口
def getResult():
    # 总牌数
    total = len(cards)

    # 记录每张牌的数量
    count = [0] * 15
    for card in cards:
        num = mapToNum(card)
        count[num] += 1

    # 记录顺子，顺子表示为数组，数组含义：[起始牌，结束牌]
    straights = []

    for i in range(3, 11):
        # 先求五张的顺子
        if isStraights(count, i, i + 4):
            # 如果可以组成五张顺子，则记录该顺子
            straights.append([i, i + 4])
            # 总牌数减5
            total -= 5
            # 对应的牌的数量减1
            for j in range(i, i + 5):
                count[j] -= 1

    # 如果五张的顺子求解完后，还有剩余牌
    if total > 0:
        # 则尝试将剩余牌追加到五张顺子后面
        for straight in straights:
            for i in range(3, 15):
                if count[i] > 0 and i - straight[1] == 1:
                    straight[1] = i
                    count[i] -= 1

    # 如果没有顺子，则返回No
    if len(straights) == 0:
        print("No")
        return

    # 如果有顺子，则打印顺子
    for start, end in straights:
        ans = [mapToCard(num) for num in range(start, end + 1)]
        print(" ".join(ans))


# 调用算法
getResult()
```

优先最长顺子解法（80%通过率）

Java算法源码

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    LinkedList<String> cards = new LinkedList<>(Arrays.asList(sc.nextLine().split(" ")));
    getResult(cards);
  }

  public static void getResult(LinkedList<String> cards) {
    cards.sort((a, b) -> map(a) - map(b));

    LinkedList<String> stack = new LinkedList<>();
    ArrayList<String> ans = new ArrayList<>();

    while (cards.size() > 0) {
      String bot = cards.removeFirst();

      if (stack.size() == 0) {
        stack.add(bot);
        continue;
      }

      String top = stack.getLast();
      if (map(bot) - map(top) == 1) {
        stack.add(bot);
      } else if (map(bot) == map(top)) {
        cards.add(bot);
      } else {
        if (stack.size() >= 5) ans.add(String.join(" ", stack));
        stack.clear();
        stack.add(bot);
      }
    }

    if (stack.size() >= 5) ans.add(String.join(" ", stack));

    if (ans.size() == 0) {
      System.out.println("No");
    } else {
      ans.sort((a, b) -> map(a.charAt(0) + "") - map(b.charAt(0) + ""));
      for (String an : ans) {
        System.out.println(an);
      }
    }
  }

  public static int map(String card) {
    switch (card) {
      case "J":
        return 11;
```

```
      case "Q":
        return 12;
      case "K":
        return 13;
      case "A":
        return 14;
      case "2":
        return 16;
      default:
        return Integer.parseInt(card);
    }
  }
}
```

JS算法源码

```
/* 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,
});

rl.on("line", (line) => {
  const cards = line.split(" ");
  getResult(cards);
});

function getResult(cards) {
  cards.sort((a, b) => map(a) - map(b));

  const stack = [];
  const ans = [];

  while (cards.length > 0) {
    const bot = cards.shift();

    if (stack.length == 0) {
      stack.push(bot);
      continue;
    }

    const top = stack.at(-1);
```

```
    const top = stack.at(-1);
    if (map(bot) - map(top) == 1) {
      stack.push(bot);
    } else if (map(bot) == map(top)) {
      cards.push(bot);
    } else {
      if (stack.length >= 5) ans.push(stack.join(" "));
      stack.length = 0;
      stack.push(bot);
    }
  }

  if (stack.length >= 5) ans.push(stack.join(" "));

  if (ans.length == 0) {
    console.log("No");
  } else {
    ans.sort((a, b) => map(a[0]) - map(b[0])).forEach((s) => console.log(s));
  }
}

function map(card) {
  switch (card) {
    case "J":
      return 11;
    case "Q":
      return 12;
    case "K":
      return 13;
    case "A":
```

```
    case "A":
      return 14;
    case "2":
      return 16;
    default:
      return parseInt(card);
  }
}
```

Python算法源码

```python
# 输入获取
cards = input().split()


def map(card):
    if card == "J":
        return 11
    elif card == "Q":
        return 12
    elif card == "K":
        return 13
    elif card == "A":
        return 14
    elif card == "2":
        return 16
    else:
        return int(card)


# 算法入口
def getResult():
    cards.sort(key=lambda x: map(x))

    stack = []
    ans = []

    while len(cards) > 0:
        bot = cards.pop(0)
```

```
        bot = cards.pop(0)

        if len(stack) == 0:
            stack.append(bot)
            continue

        top = stack[-1]
        if map(bot) - map(top) == 1:
            stack.append(bot)
        elif map(bot) == map(top):
            cards.append(bot)
        else:
            if len(stack) >= 5:
                ans.append(" ".join(stack))
            stack.clear()
            stack.append(bot)

    if len(stack) >= 5:
        ans.append(" ".join(stack))

    if len(ans) == 0:
        print("No")
    else:
        ans.sort(key=lambda x: map(x[0]))
        for an in ans:
            print(an)


```

```
        ans.sort(key=lambda x: map(x[0]))
        for an in ans:
            print(an)


# 调用算法
getResult()
```