

题目描述

给定一组数字，表示扑克牌的牌面数字，忽略扑克牌的花色，请按如下规则对这一组扑克牌进行整理：

步骤1. 对扑克牌进行分组，形成组合牌，规则如下：

- 当牌面数字相同张数大于等于4时，组合牌为“炸弹”；
- 3张相同牌面数字 + 2张相同牌面数字，且3张牌与2张牌不相同，组合牌为“葫芦”；
- 3张相同牌面数字，组合牌为“三张”；
- 2张相同牌面数字，组合牌为“对子”；
- 剩余没有相同的牌，则为“单张”；

步骤2. 对上述组合牌进行由大到小排列，规则如下：

- 不同类型组合牌之间由大到小排列规则：“炸弹”>“葫芦”>“三张”>“对子”>“单张”；
- 相同类型组合牌之间，除“葫芦”外，按组合牌全部牌面数字加总由大到小排列；
- “葫芦”则先按3张相同牌面数字加总由大到小排列，3张相同牌面数字加总相同时，再按另外2张牌面数字加总由大到小排列；
- 由于“葫芦”>“三张”，因此如果能形成更大的组合牌，也可以将“三张”拆分为2张和1张，其中的2张可以和其“三张”重新组合成“葫芦”，剩下的1张为“单张”

步骤3. 当存在多个可能组合方案时，按如下规则排序取最大的一个组合方案：

- 依次对组合方案中的组合牌进行大小比较，规则同上；
- 当组合方案A中的第n个组合牌大于组合方案B中的第n个组合牌时，组合方案A大于组合方案B；

输入描述

第一行为空格分隔的N个正整数，每个整数取值范围[1,13]，N的取值范围[1,1000]

输出描述

经重新排列后的扑克牌数字列表，每个数字以空格分隔

用例

输入	1 3 3 3 2 1 5
输出	3 3 3 1 1 5 2
说明	无

输入	4 4 2 1 2 1 3 3 3 4
输出	4 4 4 3 3 2 2 1 1 3
说明	无

题目解析

我的解题思路如下：


首先，将给定牌中，炸弹，三张，对子，单子先统计出来，即先不处理葫芦。

统计逻辑很简单，就是看某个牌面的数量：

- >=4，那么该牌面可以组成炸弹
- ==3，那么该牌面可以组成三张
- ==2，那么该牌面可以组成对子
- ==1，那么该牌面可以组成单张

统计完后，我们就可以先对炸弹进行排序，排序规则是：全部牌面数字加总由大到小排列

接着可以组合葫芦了，组合逻辑如下：



伏城之外 已关注

0

6

24

专栏目录

已订阅

题目解析

我的解题思路如下：

首先，将给定牌中，炸弹，三张，对子，单子先统计出来，即先不处理葫芦。

统计逻辑很简单，就是看某个牌面的数量：

- ≥ 4 ，那么该牌面可以组成炸弹
- $= 3$ ，那么该牌面可以组成三张
- $= 2$ ，那么该牌面可以组成对子
- $= 1$ ，那么该牌面可以组成单张

统计完后，我们就可以先对炸弹进行排序，排序规则是：全部牌面数字加总由大到小排列

接着可以组合葫芦了，组合逻辑如下：

首先，需要先对三张、对子按照加总降序

然后，选取一个最大的三张，并比较第二大的三张的牌面和第一大的对子的牌面

- 如果对子牌面大，则直接组合最大的三张和最大对子为忽略
- 如果第二大三张牌面大，则拆分该三张为一个对子，一个单张，其中对子和最大的三张组合成葫芦，单张加入前面统计的单张数组

按照上面规则组合葫芦，直到三张用完。

注意上面逻辑是三张用完结束，而不是对子用完，因为还有一种情况就是对子先用完了，但是三张还有多个，此时我们要继续拆分小的三张来组合大三张为葫芦。

组合完葫芦后。

我们就可以对单张进行加总降序排序了，因为组合葫芦过程中，很可能产生新的单张。

最后，依次将统计并排序后的炸弹、葫芦、三张、对子、单张，打印出来

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    const arr = line.split(" ").map(Number);
11    console.log(getResult(arr));
12  });
13
14  function getResult(arr) {
15    const card = {};
16
17    // 统计各种牌面的数量
18    for (let num of arr) {
19      card[num] ? card[num]++ : (card[num] = 1);
20    }
21
22    // 统计组合，4代表炸弹，3+2代表葫芦，3代表三张，2代表对子，1代表单张
23    const combine = {
24      4: [],
25      "3+2": [],
26      3: [],
27      2: [],
28      1: [],
29    };
30
31    // 首先将初始组合统计出来
32    for (let num in card) {
33      switch (card[num]) {
34        case 3:
35          combine[3].push(num);
36          break;
37        case 2:
38          combine[2].push(num);
39          break;
40        case 1:
41          combine[1].push(num);
42          break;
43        default:
44          combine[4].push([num, card[num]]); // 由于炸弹可能有4张以上相同牌面组成，因此既需要统计牌面num，也需要统计牌的
45      }
46    }
47
48    // 炸弹排序，按照牌面值总和大小排序，总和越大，越靠前，牌面总和 = 牌面值 * 牌数
49    combine[4].sort((a, b) => b[0] * b[1] - a[0] * a[1]);
50
51    // 三张排序，牌面值越大，三张越大
52    combine[3].sort((a, b) => b - a);
```

```
1  /* JavaScript Node ACH模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  rl.on("line", (line) => {
10    const arr = line.split(" ").map(Number);
11    console.log(getResult(arr));
12  });
13
14  function getResult(arr) {
15    const card = {};
16
17    // 统计各种牌面的数量
18    for (let num of arr) {
19      card[num] ? card[num]++ : (card[num] = 1);
20    }
21
22    // 统计组合, 4代表炸弹, 3+2代表葫芦, 3代表三张, 2代表对子, 1代表单张
23    const combine = {
24      4: [],
25      "3+2": [],
26      3: [],
27      2: [],
28      1: [],
29    };
30
31    // 首先将初始组合统计出来
32    for (let num in card) {
33      switch (card[num]) {
34        case 3:
35          combine[3].push(num);
36          break;
37        case 2:
38          combine[2].push(num);
39          break;
40        case 1:
41          combine[1].push(num);
42          break;
43        default:
44          combine[4].push([num, card[num]]); // 由于炸弹可能有4张以上相同牌面组成, 因此既需要统计牌面num, 也需要统计牌数
45      }
46    }
47
48    // 炸弹排序, 按照牌面值总和大小排序, 总和越大, 越高前, 牌面总和 = 牌面值 * 牌数
49    combine[4].sort((a, b) => b[0] * b[1] - a[0] * a[1]);
50
51    // 三张排序, 牌面值越大, 三张越大
52    combine[3].sort((a, b) => b - a);
53
54    // 对子排序, 牌面值越大, 对子越大
55    combine[2].sort((a, b) => b - a);
56
57    // 尝试组合出葫芦
58    while (combine[3].length) {
59      // 如果对子用完, 三张还有一个, 那么可以直接结束循环
60      if (combine[2].length === 0 && combine[3].length === 1) break;
61
62      // 选取一个最大的三张
63      const san_top = combine[3].shift();
64
65      let tmp;
66      // 如果第二大的三张的牌面, 比最大的对子牌面大, 或者没有对子了, 则可以拆分三张, 组合出葫芦
67      if (
68        combine[2].length === 0 ||
69        (combine[3].length >= 1 && combine[3][0] > combine[2][0])
70      ) {
71        tmp = combine[3].shift();
72        // 拆分三张为对子的话, 会多出一个单张
73        combine[1].push(tmp);
74      } else {
75        // 如果对子牌面比三张大, 则不需要拆分三张, 直接使用对子组合出葫芦
76        tmp = combine[2].shift();
77      }
78      combine["3+2"].push([san_top, tmp]); // 葫芦元素含义: [三张牌面, 对子牌面]
79    }
80
81    // 处理完葫芦后, 就可以对单张进行排序了 (因为组合葫芦的过程中, 可能产生新的单张, 因此单张排序要在葫芦组合得到后进行)
82    combine[1].sort((a, b) => b - a);
83
84    // ans存放题解
85    const ans = [];
86
87    // 首先将炸弹放到ans中
88    for (let card of combine[4]) {
89      const [score, count] = card;
90      ans.push(...new Array(count).fill(score));
91    }
92  }
```

```

54 // 对子排序，牌面值越大，对子越大
55 combine[2].sort((a, b) => b - a);
56
57 // 尝试组合出葫芦
58 while (combine[3].length) {
59     // 如果对子用完，三张还有一个，那么可以直接结束循环
60     if (combine[2].length === 0 && combine[3].length === 1) break;
61
62     // 选取一个最大的三张
63     const san_top = combine[3].shift();
64
65     let tmp;
66     // 如果第二大的三张的牌面，比最大的对子牌面大，或者没有对子了，则可以拆分三张，组合出葫芦
67     if (
68         combine[2].length === 0 ||
69         (combine[3].length >= 1 && combine[3][0] > combine[2][0])
70     ) {
71         tmp = combine[3].shift();
72         // 拆分三张为对子的话，会多出一个单张
73         combine[1].push(tmp);
74     } else {
75         // 如果对子牌面比三张大，则不需要拆分三张，直接使用对子组合出葫芦
76         tmp = combine[2].shift();
77     }
78     combine["3+2"].push([san_top, tmp]); // 葫芦元素含义：[三张牌面，对子牌面]
79 }
80
81 // 处理完葫芦后，就可以对单张进行排序了（因为组合葫芦的过程中，可能产生新的单张，因此单张排序要在葫芦组合得到后进行）
82 combine[1].sort((a, b) => b - a);
83
84 // ans存放题解
85 const ans = [];
86
87 // 首先将炸弹放到ans中
88 for (let card of combine[4]) {
89     const [score, count] = card;
90     ans.push(...new Array(count).fill(score));
91 }
92
93 // 然后将葫芦放大ans中
94 for (let card of combine["3+2"]) {
95     const [san, er] = card;
96     ans.push(...new Array(3).fill(san), ...new Array(2).fill(er));
97 }
98
99 // 之后将三张放到ans中
100 for (let san of combine[3]) {
101     ans.push(...new Array(3).fill(san));
102 }
103
104 // 接着将对子放到ans中
105 for (let er of combine[2]) {
106     ans.push(...new Array(2).fill(er));
107 }
108
109 // 最后是单张放到ans中
110 ans.push(...combine[1]);
111
112 return ans.join(" ");
113 }

```

Java算法源码

2023.1.2 根据网友指正，发现 [Java版本](#) 代码存在问题，即LinkedList的push方法是头插，这将会导致葫芦的排序出错，换成add方法即可改正。

```

1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         String str = sc.nextLine();
8         Integer[] arr = Arrays.stream(str.split(" ")).map(Integer::parseInt).toArray(Integer[]::new);
9
10        System.out.println(getResult(arr));
11    }
12
13    public static String getResult(Integer[] arr) {
14        HashMap<Integer, Integer> card = new HashMap<>();
15
16        // 统计各种牌面的数量
17        for (Integer num : arr) {
18            if (card.containsKey(num)) {
19                int val = card.get(num);
20                card.put(num, ++val);
21            } else {
22                card.put(num, 1);
23            }
24        }
25    }

```

```

1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         String str = sc.nextLine();
8         Integer[] arr = Arrays.stream(str.split(" ")).map(Integer::parseInt).toArray(Integer[]::new);
9
10        System.out.println(getResult(arr));
11    }
12
13    public static String getResult(Integer[] arr) {
14        HashMap<Integer, Integer> card = new HashMap<>();
15
16        // 统计各种牌面的数量
17        for (Integer num : arr) {
18            if (card.containsKey(num)) {
19                int val = card.get(num);
20                card.put(num, ++val);
21            } else {
22                card.put(num, 1);
23            }
24        }
25
26        // 统计组合, 4代表炸弹, 3+2代表葫芦, 3代表三张, 2代表对子, 1代表单张
27        HashMap<String, LinkedList<Integer[]>> combine = new HashMap<>();
28        combine.put("4", new LinkedList<Integer[]>());
29        combine.put("3+2", new LinkedList<Integer[]>());
30        combine.put("3", new LinkedList<Integer[]>());
31        combine.put("2", new LinkedList<Integer[]>());
32        combine.put("1", new LinkedList<Integer[]>());
33
34        // 首先将初始组合统计出来
35        Set<Integer> cardKeys = card.keySet();
36        for (Integer num : cardKeys) {
37            switch (card.get(num)) {
38                case 3:
39                    combine.get("3").add(new Integer[] { num});
40                    break;
41                case 2:
42                    combine.get("2").add(new Integer[] { num});
43                    break;
44                case 1:
45                    combine.get("1").add(new Integer[] { num});
46                    break;
47                default:
48                    combine
49                        .get("4")
50                        .add(
51                            new Integer[] {
52                                num, card.get(num)
53                            }); // 由于炸弹可能有4张以上相同牌面组成, 因此既需要统计牌面num, 也需要统计牌数card[num]
54            }
55        }
56
57        // 炸弹排序, 按照牌面值总和大小排序, 总和越大, 越高前, 牌面总和 = 牌面值 * 牌数
58        combine.get("4").sort((a, b) -> b[0] * b[1] - a[0] * a[1]);
59
60        // 三张排序, 牌面值越大, 三张越大
61        combine.get("3").sort((a, b) -> b[0] - a[0]);
62
63        // 对子排序, 牌面值越大, 对子越大
64        combine.get("2").sort((a, b) -> b[0] - a[0]);
65
66        // 尝试组合出葫芦
67        while (combine.get("3").size() > 0) {
68            // 如果对子用完, 三张还有一个, 那么可以直接结束循环
69            if (combine.get("2").size() == 0 && combine.get("3").size() == 1) break;
70
71            // 否则, 选取一个最大的三张
72            Integer san_top = combine.get("3").removeFirst()[0];
73
74            Integer tmp;
75            // 如果此时没有对子了, 胡总和第二大的三张的牌面, 比最大的对子牌面大, 则可以拆分三张, 组合出葫芦
76            if (combine.get("2").size() == 0
77                || (combine.get("3").size() > 0
78                    && combine.get("3").get(0)[0] > combine.get("2").get(0)[0])) {
79                tmp = combine.get("3").removeFirst()[0];
80                // 拆分三张为对子的话, 会多出一个单张
81                combine.get("1").add(new Integer[] { tmp});
82            } else {
83                // 如果对子牌面比三张大, 则不需要拆分三张, 直接使用对子组合出葫芦
84                tmp = combine.get("2").removeFirst()[0];
85            }
86            combine.get("3+2").add(new Integer[] { san_top, tmp}); // 葫芦元素含义: {三张牌面, 对子牌面}
87        }

```



伏城之外 已关注

👍 0



🌟 6



🗨️ 24



专栏目录

已订阅

```

52         num, card.get(num)
53     }); // 由于炸弹可能有4张以上相同牌面组成, 因此既需要统计牌面num, 也需要统计牌数card[num]
54 }
55 }
56
57 // 炸弹排序, 按照牌面值和大小排序, 总和越大, 越高前, 牌面总和 = 牌面值 * 牌数
58 combine.get("4").sort((a, b) -> b[0] * b[1] - a[0] * a[1]);
59
60 // 三张排序, 牌面值越大, 三张越大
61 combine.get("3").sort((a, b) -> b[0] - a[0]);
62
63 // 对子排序, 牌面值越大, 对子越大
64 combine.get("2").sort((a, b) -> b[0] - a[0]);
65
66 // 尝试组合出葫芦
67 while (combine.get("3").size() > 0) {
68     // 如果对子用完, 三张还有一个, 那么可以直接结束循环
69     if (combine.get("2").size() == 0 && combine.get("3").size() == 1) break;
70
71     // 否则, 选取一个最大的三张
72     Integer san_top = combine.get("3").removeFirst()[0];
73
74     Integer tmp;
75     // 如果此时没有对子了, 胡总和第二大的三张的牌面, 比最大的对子牌面大, 则可以拆分三张, 组合出葫芦
76     if (combine.get("2").size() == 0
77         || (combine.get("3").size() > 0
78             && combine.get("3").get(0)[0] > combine.get("2").get(0)[0])) {
79         tmp = combine.get("3").removeFirst()[0];
80         // 拆分三张为对子的话, 会多出一个单张
81         combine.get("1").add(new Integer[] {tmp});
82     } else {
83         // 如果对子牌面比三张大, 则不需要拆分三张, 直接使用对子组合出葫芦
84         tmp = combine.get("2").removeFirst()[0];
85     }
86     combine.get("3+2").add(new Integer[] {san_top, tmp}); // 葫芦元素含义: [三张牌面, 对子牌面]
87 }
88
89 // 处理完葫芦后, 就可以对单张进行排序了 (因为组合葫芦的过程中, 可能产生新的单张, 因此单张排序要在葫芦组合得到后进行)
90 combine.get("1").sort((a, b) -> b[0] - a[0]);
91
92 // ans存放题解
93 ArrayList<Integer> ans = new ArrayList<>();
94
95 // 首先将炸弹放到ans中
96 for (Integer[] vals : combine.get("4")) {
97     int score = vals[0];
98     int count = vals[1];
99     for (int i = 0; i < count; i++) {
100         ans.add(score);
101     }
102 }
103
104 // 然后将葫芦放入ans中
105 for (Integer[] vals : combine.get("3+2")) {
106     int san = vals[0];
107     int er = vals[1];
108     for (int i = 0; i < 3; i++) ans.add(san);
109     for (int i = 0; i < 2; i++) ans.add(er);
110 }
111
112 // 之后将三张放入ans中
113 for (Integer[] vals : combine.get("3")) {
114     for (int i = 0; i < 3; i++) ans.add(vals[0]);
115 }
116
117 // 接着将对子放入ans中
118 for (Integer[] vals : combine.get("2")) {
119     for (int i = 0; i < 2; i++) ans.add(vals[0]);
120 }
121
122 // 最后是单张放入ans中
123 for (Integer[] vals : combine.get("1")) {
124     ans.add(vals[0]);
125 }
126
127 StringJoiner sj = new StringJoiner(" ", "", "");
128 for (Integer an : ans) {
129     sj.add(an + "");
130 }
131
132 return sj.toString();
133 }
134 }

```

Python算法源码

```

1 # 输入获取
2 arr = input().split()
3
4
5 # 算法入口
6 def getResult(arr):
7     # card统计各种牌面的数量

```



```

1 # 输入获取
2 arr = input().split()
3
4
5 # 算法入口
6 def getResult(arr):
7     # card统计各种牌面的数量
8     card = {}
9     for num in arr:
10         if card.get(num) is None:
11             card[num] = 1
12         else:
13             card[num] += 1
14
15     # combine统计组合, 4代表炸弹, 3+2代表葫芦, 3代表三张, 2代表对子, 1代表单张
16     combine = {
17         "4": [],
18         "3+2": [],
19         "3": [],
20         "2": [],
21         "1": []
22     }
23
24     # 首先将初始组合统计出来
25     for num in card.keys():
26         if card[num] == 3:
27             combine["3"].append(num)
28         elif card[num] == 2:
29             combine["2"].append(num)
30         elif card[num] == 1:
31             combine["1"].append(num)
32         else:
33             # 由于炸弹可能有4张以上相同牌面组成, 因此需要统计牌面num, 也需要统计牌数card[num]
34             combine["4"].append([num, card[num]])
35
36     # 炸弹排序, 按照牌面值总和大小排序, 总和越大, 越高前, 牌面总和 = 牌面值 * 牌数
37     combine["4"].sort(key=lambda x: -x[0] * x[1])
38
39     # 三张排序, 牌面值越大, 三张越大
40     combine["3"].sort(reverse=True)
41
42     # 对子排序, 牌面值越大, 对子越大
43     combine["2"].sort(reverse=True)
44
45     # 尝试组合出葫芦
46     while len(combine["3"]) > 0:
47         # 如果对子用完, 三张还有一个, 那么可以直接结束循环
48         if len(combine["2"]) == 0 and len(combine["3"]) == 1:
49             break
50
51         # 选取一个最大的三张
52         san_top = combine["3"].pop(0)
53
54         tmp = None
55
56         # 如果第二大三张的牌面, 比最大的对子牌面大, 或者没有对子了, 则可以拆分三张, 组合出葫芦
57         if len(combine["2"]) == 0 or (len(combine["3"]) >= 1 and combine["3"][0] > combine["2"][0]):
58             tmp = combine["3"].pop(0)
59             # 拆分三张为对子的话, 会多出一个单张
60             combine["1"].append(tmp)
61         else:
62             # 如果对子牌面比三张大, 则不需要拆分三张, 直接使用对子组合出葫芦
63             tmp = combine["2"].pop(0)
64
65         combine["3+2"].append([san_top, tmp]) # 葫芦元素含义: [三张牌面, 对子牌面]
66
67     # 处理完葫芦后, 就可以对单张进行降序了 (因为组合葫芦的过程中, 可能产生新的单张, 因此单张排序要在葫芦组合得到后进行)
68     combine["1"].sort(reverse=True)
69
70     # ans存放题解
71     ans = []
72
73     # 首先将炸弹放到ans中
74     for score, count in combine["4"]:
75         ans += [score] * count
76
77     # 然后将葫芦放入ans中
78     for san, er in combine["3+2"]:
79         ans += [san] * 3 + [er] * 2
80
81     # 之后将三张放入ans中
82     for san in combine["3"]:
83         ans += [san] * 3
84
85     # 接着将对子放入ans中
86     for er in combine["2"]:
87         ans += [er] * 2
88
89     # 最后是单张放入ans中
90     for dan in combine["1"]:
91         ans += [dan]
92

```

复制

```

55
56     # 如果第二大的三张的牌面, 比最大的对子牌面大, 或者没有对子了, 则可以拆分三张, 组合出葫芦
57     if len(combine["2"]) == 0 or (len(combine["3"]) >= 1 and combine["3"][0] > combine["2"][0]):
58         tmp = combine["3"].pop(0)
59         # 拆分三张为对子的话, 会多出一个单张
60         combine["1"].append(tmp)
61     else:
62         # 如果对子牌面比三张大, 则不需要拆分三张, 直接使用对子组合出葫芦
63         tmp = combine["2"].pop(0)
64
65         combine["3+2"].append([san_top, tmp]) # 葫芦元素含义: [三张牌面, 对子牌面]
66
67     # 处理完葫芦后, 就可以对单张进行降序了 (因为组合葫芦的过程中, 可能产生新的单张, 因此单张排序要在葫芦组合得到后进行)
68     combine["1"].sort(reverse=True)
69
70     # ans存放题解
71     ans = []
72
73     # 首先将炸弹放到ans中
74     for score, count in combine["4"]:
75         ans += [score] * count
76
77     # 然后将葫芦放大ans中
78     for san, er in combine["3+2"]:
79         ans += [san] * 3 + [er] * 2
80
81     # 之后将三张放到ans中
82     for san in combine["3"]:
83         ans += [san] * 3
84
85     # 接着是对子放到ans中
86     for er in combine["2"]:
87         ans += [er] * 2
88
89     # 最后是单张放到ans中
90     for dan in combine["1"]:
91         ans += [dan]
92
93     return " ".join(ans)
94
95
96 # 算法调用
97 print(getResult(arr))

```