链滴

微软算法面试题「判断麻将是否和牌」应该如何做？

作者：washmore
原文链接：https://ld246.com/article/1505384764750
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

最近逛知乎的时候看到了这样一个题目,于是随便搜了一下,这是微软的一道面试题,题目如下:

1. 这个题目要求提供最终代码(C#)

2. 该最终代码必须可以编译,运行,并实现以下的业务功能

3. 限制时间一个小时, 包括阅读文档和提交代码的时间

业务功能:

给定若干张的麻将牌 (假设只有 万 一种类型,没有条和筒)

最终胡牌必须满足以下条件

所有的牌必须连成顺子或者3张 即:123 或者111

最后还要有一对, 例如 11

方法签名如下:

```
 bool Test( int []  cards)

{
      //这里是你的代码

}
```

传入参数例如 { 1, 1 , 2 , 3} 代表传入2张一万,一张2万,一张3万

返回参数是true 就代表胡牌, false 代表不能胡牌

之前其实恰好有随手写过这样一个demo,而且考虑了红中赖子的情况,不过没有给出具体思路,代码也得很凌乱...详见:麻将胡牌算法之爆破法

这里按照题目的要求,不考虑红中赖子的因素,简化代码,另一并给出使用穷举法解题的思路:

3N+2,首先,要明确的是,一副手牌如果胡牌,必定遵循3N+2的手牌牌型,即n(可以为0)组顺子(暗刻)+一将(掌门)...

首先是外围代码,我们创建对应的麻将花色枚举类和操作类:

```
public enum 麻将 {
   //红中(0, 0),
  一饼(1, 1),
   二饼(1, 2),
   三饼(1, 3),
   四饼(1, 4),
   五饼(1, 5),
   六饼(1, 6),
   七饼(1, 7),
   八饼(1, 8),
   九饼(1, 9),
   一条(2, 1),
   二条(2, 2),
```

```
    三条(2, 3),
    四条(2, 4),
    五条(2, 5),
    六条(2, 6),
    七条(2, 7),
    八条(2, 8),
    九条(2, 9),
    一万(3, 1),
    二万(3, 2),
    三万(3, 3),
    四万(3, 4),
    五万(3, 5),
    六万(3, 6),
    七万(3, 7),
    八万(3, 8),
    九万(3, 9);

    private int 花色;
    private int 点数;

    public int get花色() {
        return 花色;
    }

    public void set花色(int 花色) {
        this.花色 = 花色;
    }

    public int get点数() {
        return 点数;
    }

    public void set点数(int 点数) {
        this.点数 = 点数;
    }

    麻将(int 花色, int 点数) {
        this.花色 = 花色;
        this.点数 = 点数;
    }

    public static 麻将 获取指定牌型麻将(int 花色, int 点数) {
        for (麻将 pai : 麻将.values()) {
            if (pai.花色 == 花色 && pai.点数 == 点数) {
                return pai;
            }
        }
        throw new IllegalArgumentException("没有这样的牌型");
    }
}

public class 牌局 {
    public List<麻将> majiangLeft = new ArrayList<>();
```

```
    private 牌局() {

    }

    public static 牌局 开始牌局() {
        return new 牌局();
    }

    public void 洗牌() {
        if (CollectionUtils.isNotEmpty(majiangLeft)) {
            throw new IllegalArgumentException("一局牌只能洗牌一次!");
        }
        Arrays.stream(麻将.values()).forEach(m -> {
            majiangLeft.add(m);
            majiangLeft.add(m);
            majiangLeft.add(m);
            majiangLeft.add(m);
        });
    }

    public 麻将 发牌() {
        if (majiangLeft.size() == 0) {
            throw new IllegalArgumentException("已经没有剩余牌了");
        }
        return majiangLeft.remove(new Random().nextInt(majiangLeft.size()));
    }

    public 麻将 发指定牌(麻将 m) {
        for (麻将 leftM : majiangLeft) {
            if (leftM == m) {
                majiangLeft.remove(m);
                return m;
            }
        }
        throw new IllegalArgumentException("没有这种牌了");
    }
```

以及开局洗牌,发牌等逻辑:

```
牌局 station = 牌局.开始牌局();
station.洗牌();
List<麻将> 手牌 = new ArrayList<>();
IntStream.rangeClosed(1, 14).forEach(i -> 手牌.add(station.发牌()));
```

得到14张手牌

```
[八饼, 八饼, 八饼, 九饼, 九饼, 四条, 五条, 五条, 六条, 六条, 七条, 二万, 三万, 四万]
```

然后,我们将相同牌放到一起,然后排号顺序:

```
List<List<麻将>> 分组牌 = 手牌.stream().collect(Collectors.groupingBy(x -> x.get花色() + ":" + x
get点数())).values()
.stream().sorted((l1, l2) -> {
    if (l1.get(0).get花色() == l2.get(0).get花色()) {
```

```
        return l1.get(0).get点数() - l2.get(0).get点数();
    }
    return l1.get(0).get花色() - l2.get(0).get花色();
}).collect(Collectors.toList());
```

得到摆好的手牌:

```
[[八饼, 八饼, 八饼], [九饼, 九饼], [四条], [五条, 五条], [六条, 六条], [七条], [二万], [三万], [四万]]
```

如果没有牌型没有任何对子,那么肯定无法胡牌,如果包含对子,我们就看剩下的能否全部组成顺子或者刻;

```
boolean 是否胡牌 = false;
//验证将军
if (分组牌.stream().filter(l -> CollectionUtils.size(l) >= 2).count() > 0) {
    for (int i = 0; i < 分组牌.size(); i++) {
        List<麻将> t = 分组牌.get(i);
        if (t.size() >= 2) {
            List> 分组牌副本 = 深度复制(分组牌);
            分组牌副本.get(i).remove(0);
            分组牌副本.get(i).remove(0);
            if (验证3N(分组牌副本)) {
                是否胡牌 = true;
                break;
            }
        }
    }
} else {
    System.out.println("没有掌门,无法胡牌...");
}
```

上面的代码中有体现出,会不停的尝试多种对子作为掌门的情况,直到能判定成功胡牌为止!

其中深度复制方法用于让原始数据(引用型)可以重复使用不被影响;

我们看看如何处理3N的:

```
private boolean 验证3N(List> 分组牌副本) {
    List> trimList = 分组牌副本.stream().filter(l -> l.size() > 0).collect(Collectors.toList());
    if (trimList == null || trimList.size() == 0) {
        return true;
    }
    List<麻将> check = trimList.get(0);
    if (check.size() > 3) {
        if (!处理顺子(trimList)) {
            return false;
        }
    } else if (check.size() == 1 || check.size() == 2) {
        if (!处理顺子(trimList)) {
            return false;
        }
    } else if (check.size() == 3) {
        trimList.get(0).removeAll(check);
    }
```

```
    return 验证3N(trimList);
}
```

注意,当check.size() > 3,则只可能check.size() ==4,如果当前玩家准备14子胡牌,那么肯定不能算为杠则其中一定有一个牌要用来做顺子;

再看check.size() == 1 || check.size() == 2 时,由于不满三张,无法组成暗刻,因此也只有可能和旁边的组成顺子;

由于会递归调用此方法,只有当以上两种组合完全排除后,剩下的则一定要组成暗刻check.size() == 3,整体移除三张;

最后如果所有的牌都被成功移除,满足

```
    if (trimList == null || trimList.size() == 0) {
        return true;
    }
```

表示此手牌不是顺子就是暗刻,成功胡牌,我们看一下处理顺子的具体逻辑

```
public static boolean 处理顺子(List> trimList) {
    if (trimList.size() < 3) {
        return false;
    }
    麻将 first = trimList.get(0).get(0);
    麻将 second = trimList.get(1).get(0);
    麻将 third = trimList.get(2).get(0);
    if (!(first.get花色() == second.get花色() && first.get花色() == third.get花色()//
  && first.get点数() == second.get点数() - 1 && first.get点数() == third.get点数() - 2)) {
        return false;
    }
    trimList.get(0).remove(0);
    trimList.get(1).remove(0);
    trimList.get(2).remove(0);
    return true;
}
```

关键逻辑就是判断三张牌是否同花色并且点数连续...

以上就是使用递归法判断麻将手牌是否胡牌的全解,代码写的有点狗屎,有丁点兴趣的童鞋可在github找到,传送门:**微软面试题_判断麻将是否胡牌**