链滴

《程序员的算法趣题》-Q5- 还在用现金支付吗

作者：DattyRabbit

原文链接：https://ld246.com/article/1601561816340

来源网站：链滴

前言

此篇为《程序员的算法趣题》中的入门篇第5题"还在用现金支付吗"的相关解题分析博文。

关于该系列的介绍请看：

《程序员的算法趣题》-开坑记录

题目

当下，坐公交或者地铁时大部分人都是刷卡的。不过，时至今日还在用现金支付的人还是比想象的多本题我们以安置在公交上的零钱兑换机为背景。

这个机器可以用纸币兑换到 10 日元、50 日元、100 日元和 500 日元硬币的组合，且每种硬币的数都足够多（因为公交接受的最小额度为 10 日元，所以不提供 1 日元和 5 日元的硬币）。

兑换时，允许机器兑换出本次支付时用不到的硬币。此外，因为在乘坐公交时，如果兑换出了大量的钱会比较不便，所以只允许机器最多兑换出 15 枚硬币。譬如用 1000 日元纸币兑换时，就不能兑换“100 枚 10 日元硬币”的组合（ 图5 ）。

问题

求兑换 1000 日元纸币时会出现多少种组合？注意，不计硬币兑出的先后顺序。

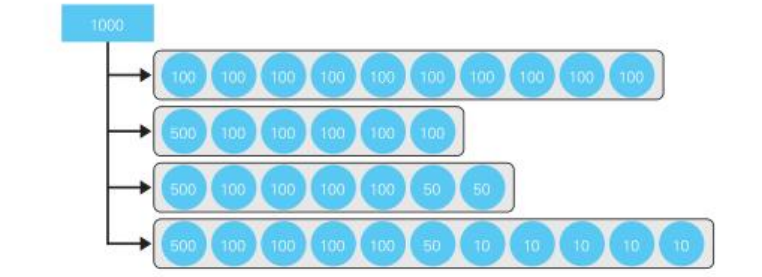
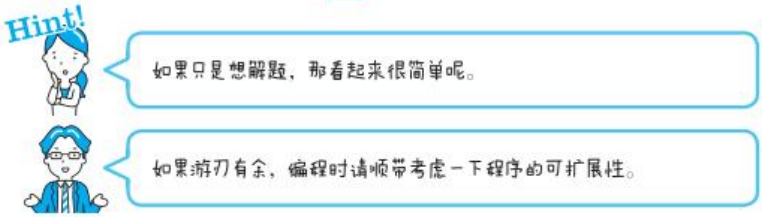


图5 兑换示例



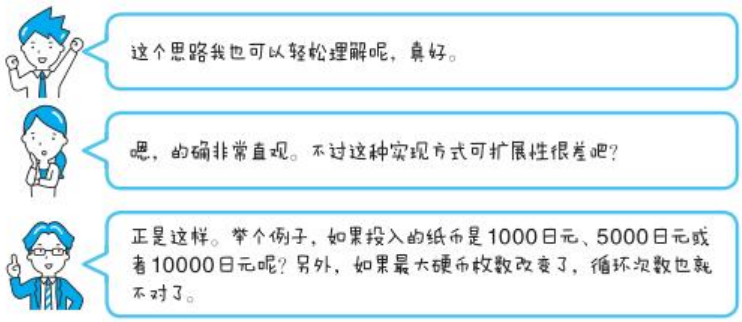
作者思路及代码实现

思路

这道题并不复杂，单纯地解开并不是什么难事。只需要把满足条件的硬币组合一一列举出来就可以了譬如想简单地使用循环来解答时，用 Ruby 就可以实现，代码如代码清单 05.01 所示。

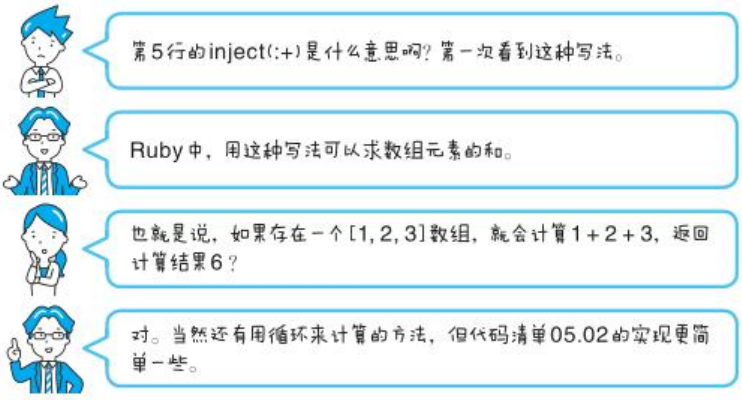
```
cnt = 0
(0..2).each{|coin500| # 500 日元硬币最多 2 枚
   (0..10).each{|coin100| # 100 日元硬币最多 10 枚
      (0..15).each{|coin50| # 50 日元硬币最多 15 枚
```

```
        (0..15).each{|coin10| # 10 日元硬币最多 15 枚
          if coin500 + coin100 + coin50 + coin10 <= 15 then
            if coin500 * 500 + coin100 * 100 + coin50 * 50 + coin10 * 10 == 1000 then
              cnt += 1
            end
          end
        }
      }
    }
}
puts cnt
```



所以，我们需要设计可以更灵活地应对变化的算法（这里不考虑处理速度，单纯从可扩展性的角度出），譬如代码清单 05.02 的实现方式。

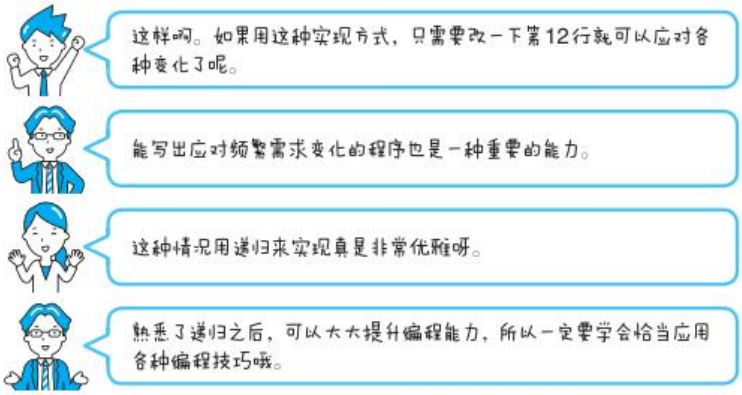
```
coins = [10, 50, 100, 500]
cnt = 0
(2..15).each do |i|
  coins.repeated_combination(i).each{|coin_set|
    cnt += 1 if coin_set.inject(:+) == 1000
  }
end
puts cnt
```



如果是这样的程序，那么即使在改变硬币面值、目标纸币面值等时，也能一眼看到要改哪些地方。这逻辑还可以利用递归来实现（代码清单 05.03）。

```
@cnt = 0
def change(target, coins, usable)
  coin = coins.shift
```

```
    if coins.size == 0 then
        @cnt += 1 if target / coin <= usable
    else
        (0..target/coin).each{|i|
            change(target - coin * i, coins.clone, usable - i)
        }
    end
end
change(1000, [500, 100, 50, 10], 15)
puts @cnt
```



答案

Column

用函数式语言学习递归

虽然用过程式语言也能学习递归，但哪怕掌握一点点函数式语言，就能对理解递归有所裨益。在函数语言里，用递归实现循环功能的做法非常普遍。也就是说，用函数式语言编程基本上离不开递归。

LISP、Scheme、Haskell 等是代表性的函数式语言，此外使用 Scala、Python 等也可以学习到函数语言的特性。可以尝试把其他语言里的循环用函数式语言的方式（不使用循环）来实现。

熟悉了这种写法后，你会发现，用递归来实现循环应该就不困难了。不过，反过来可能又会觉得把递的写法转换成循环会比较难，所以请务必多做一些这样的写法转换训练。

自己做的思路及实现

最开始看到题目的时候，是想用循环的方式实现的，但是想要做足可扩展性，满足变化的条件，想到果改变可兑换的硬币面值的条件之后，对于循环本身有多少层就不能确定。思考了良久，如何用递归解决，结果一直没想清楚递归体怎么完成，然后把这个问题放了一小会儿，去上了个大号，又一次想白了怎么去写。然后回来就一点一点的先把代码先写完，然后再去补充的注释（以往是思路清晰，先每一步要做什么的注释写好，再去写代码，这一次最初思路不是特别清晰，就先从突破口出开始写，现缺什么补什么完成的，所以就是先完成的代码，再补写的注释）。

大致思路如下：

-

使用递归方法，将待兑换的数值和可用来兑换的数组以及允许兑换的次数作为参数进行传递。

-

递归方法中会取可兑换面额的首位，将其从原数组中取出，然后进行判断

-

如果可兑换的数组为空，则用当前兑换的币值和待兑换的面值作除法，结果如果小于等于允许的最大币个数，则满足。

-

如果可兑换数组仍有数，则从0到待兑换面额除以当前兑换的币值结果来遍历，遍历中递归调用本方
，
参数为 待兑换的币值 - 当前兑换的币值 * 遍历数 作为下次的待兑换币值，
剩下的可兑换的硬币面额数组作为下次的可兑换数组，
然后 允许的最大兑换个数 - 当前遍历数 作为下次的允许的最大兑换数量。

可能文字有点不好理解，直接看代码吧

```
public class CashPayment {

    //硬币面额数组
    private Integer[] coinDenominations;
    //能兑换的总组合数
    private int exchangeGroupCount = 0;
    //能够允许的最大兑换硬币个数默认15
    private int maxCoinCount = 15;

    //默认构造方法，初始化硬币面额值为固定500,100,50,10
    public CashPayment(){
        coinDenominations = new Integer[]{500,100,50,10};
    }

    //带参构造方法，传入硬币面额值的数组，和允许最大兑换的硬币个数
    public CashPayment(Integer[] coinDenominations,int maxCoinCount){
        this.coinDenominations = coinDenominations;
        this.maxCoinCount = maxCoinCount;
    }


    /**
     * 获得待兑换的纸币总共能有多少总兑换组合。
     * @param banknotes
     */
    public void execute(int banknotes){
        ArrayList<Integer> denominations = new ArrayList<>();
        Collections.addAll(denominations, coinDenominations);
        String str_coinDenominations = denominations.toString();
        //创建一个数组用于暂时保存兑换时各个硬币面额的数量
        int[] exchangeCoinGroup = new int[coinDenominations.length];
```

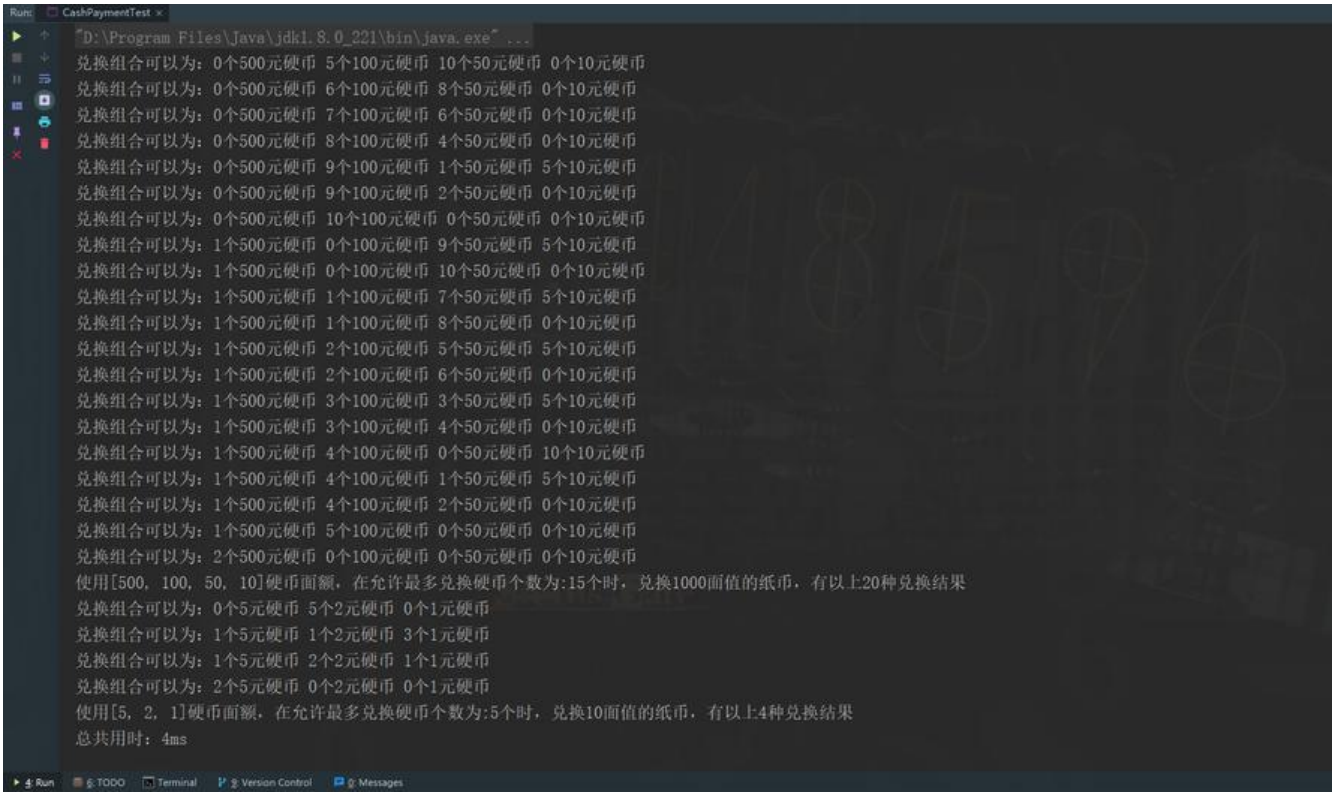
```
        exchangeCash(banknotes, denominations, maxCoinCount, exchangeCoinGroup);
        System.out.println("使用"+str_coinDenominations+"硬币面额，在允许最多兑换硬币个数为:
+maxCoinCount+"个时，兑换"+banknotes+"面值的纸币，有以上"+exchangeGroupCount+"种
换结果");
    }

    /**
     * 根据需要兑换的纸币面额，求出可以兑换的硬币组合并打印
     * @param banknotes 待兑换的面额
     * @param denominations 可兑换的硬币面额
     * @param max 最大能兑换的硬币数
     */
    public void exchangeCash(int banknotes, ArrayList<Integer> denominations, int max, int[]
xchangeCoinGroup){
        //获得当前使用的兑换币面额
        int coin = denominations.remove(0);

        //保存当前需要操作的面额数组的下标
        int currentIndex = coinDenominations.length - 1 - denominations.size();
        if(denominations.isEmpty()){
            //兑换完成
            if(banknotes / coin <= max){
                //进行当前组合面额的保存
                exchangeCoinGroup[currentIndex] = banknotes / coin;
                //输出组合，总组合数+1
                exchangeGroupCount += 1;
                //输出打印单次的兑换结果
                System.out.print("兑换组合可以为：");
                for(int i = 0; i < coinDenominations.length; i++){
                    System.out.print(exchangeCoinGroup[i]+"个"+coinDenominations[i]+"元硬币 ");
                }
                System.out.println();
            }
        }else{
            //兑换未完成，进行递归兑换
            for(int i = 0; i <= banknotes/coin; i++){
                //进行当前组合面额的保存
                exchangeCoinGroup[currentIndex] = i;
                exchangeCash(banknotes - coin * i, (ArrayList<Integer>) denominations.clone(), max
- i, exchangeCoinGroup);
            }
        }
    }
}
```

然后再来一个测试类

```
public class CashPaymentTest {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        CashPayment cashPayment1 = new CashPayment();
        cashPayment1.execute(1000);
        CashPayment cashPayment2 = new CashPayment(new Integer[]{5,2,1}, 5);
        cashPayment2.execute(10);
```

```
        long end = System.currentTimeMillis();
        System.out.println("总共用时：" + (end-start) +"ms");
    }
}
```

测试结果如下



不同思路的对比

作者在解题思路中一共给出了三种思路。分别聊一下吧。

第一个方法就是最简单的，也是适应性最差的一种，但凡有参数改动就需要改动代码本身，而且循环套次数太多，可读性和可扩展性都不好，所以基本就是作者用来给初心者引路的一种方式。

然后作者给出的第二种思路的写法让我第一眼看上去不太明白，因为不是很懂ruby的一些内置函数，以除了看作者给出提示的inject(:+)之外，还去看了下repeated_combination()的功能，虽然在查之根据方法语义以及整段代码已经猜出了功能是根据传入参数来将对应的数组元素进行排列组合，并得所有组合的数组。但是查一查验证下，也是应该的操作。

这里不禁感慨下ruby的内置函数库还挺厉害的，java当中应该是没有类似的工具方法（也可能是因为己没见过）。通过这样两个函数就较为轻松的完成了解题，并且还可以做到一个比较好的适应性，改参数都只需要改动参数本身，而不需要去调整代码，还是蛮有意思的。

然后作者给出的第三个方法就与我写的一样，不过我为了测试写得对不对，要看到结果也要看到过程就多加了一个记录兑换硬币组合详情的数组，用于结果的输出，方便调试，也让最后的结果更直观可一些。除此之外，自己的那部分代码好像也没啥亮点了
oy

总结

这一节多少还是花了点时间，一个是用在思考怎么写递归体，另一个是为了方便调试，判断结果是否确，我自己加了一个记录状态的部分，然后因为初期脑子大概有坑，就自己挖了个坑跳进去，然后后大部分的时间都在找坑填平的过程中浪费掉了sob

总结下来就是看到作者给出的第二个思路，觉得还是眼前一亮，因为自己当时是没想到去写一个类似repeated_combination()的方法，去把数组元素按照组合个数参数去进行组合。然后再判断每种组合否能满足要求。自己从头到尾只有一个正序的思路，就是用待兑换的钱去换硬币，把剩余需要兑换的直换到0，才算完。所有根据完事后看作者的思路，感觉还是学到了不少新的解决思路。

不过个人还是更爱递归一点，但是也如作者所说，要多锻炼循环和递归两种写法的相互转化，不然是易出现把递归的写法转换成循环比较难的感觉。

PS:自己以前比较喜欢看日漫，所以也学习过日文，但是感觉对于日本的一些文化习俗确实是不甚了（虽然对于一般人来说已经算比较了解的了，毕竟是没去过日本）。通过读这本书，没想到还可以了一些当地的常识啥的（这个公交车上的零钱兑换机以前还真不知道），有这种意外收获还是挺不错哈。