详细设计

- 创建一个长度为2^{length}的$boolean$数组$ifCheck$,其中$ifCheck[i]$代表i对应的选取方案是否被**检查过**
- $i\ from\ 1\ to\ 2^{length}-1$,对应二进制的范围从$0b00001\ to\ 0b11111$
  - 如果i对应的方案被检查过,继续下一轮迭代
  - 初始化一个HashSet,表示当前方案对应的满减券集合$subSet$(实际上是个List)set的**子集**
  - $index \leftarrow i$
  - $j\ from\ 0\ to\ length$ 遍历i代表队二进制数字的每一位(位数从右往左递增,$1_{第4位}\ 0_{第3位}\ 1_{第2位}\ 0_{第1位}\ 0_{第0位}$)
    - 如果$index\ \&\ 1 == 1$,表示第j位上的数字为1
      - 满减券$set[j]$存在于i代表的满减券组合中,将$set[j]$添加到$subSet$中
    - $index$ 向右移动一位
  - 例如, i 为 $0b10100$,遍历完之后,subSet 为 $\{set[2], set[4]\}$
  - $ifCheck[i] \leftarrow true$,i代表的满减券组合方案检查完毕
  - 计算满减限额之和是否小于商品总金额,如果大于
    - 假设 i 为 $0b10100$,计算的满减限额 > 商品总金额, 那么$0b10101, 0b10110, 0b10111$,..,由于**使用了更多的券**,所以这些方案对应的**满减限额也必然 > 商品总金额**(前提是满减券累加门槛成立)
    - $reversedI \leftarrow i\ \hat{}\ ((1 << length) - 1)$ 获得i的反码, reversel 为 $0b01011$
    - $k \leftarrow reversedI$,当$k > 0$ // **k的实际含义是, 没有被分配的满减券$set[0], set[1], set[3]$的可能的排列组合,而已经被分配的$set[2], set[4]$对应的位数始终为0,不参与分配**, 例如$0b1_{第3位}\ 1_{第1位}\ 1_{第0位}$
      - $subI \leftarrow i|k$ // 例如, i代表$0b10100$($set[2], set[4]$), k代表$0b01011$($set[0], set[1], set[3]$),subI表示选取$set[2], set[4], set[0], set[1], set[3]$
      - $ifCheck[subI] \leftarrow true;$ //因为i$set[2], set[4]$不是可行方案,所以在i的基础上,多选了券$set[0], set[1], set[3]$更不可能是可行方案,这种方案此时已经认为检查过,发现不可行
      - $k \leftarrow k - 1$
      - $k = k\ \&reversedI$ // **保证在k递减的过程中,reversedI为0的位置(代表i已经选择的方案),k始终为0,reversedI为1的位置,随着k的减少,可能变为1或0**
      - continue
    - 否则,如果小于等于,把子集存储起来,继续执行
  - 返回所有组合

```
    public static Set<Set<Integer>> getSubset(int[] set){
        Set<Set<Integer>> result = new HashSet<>();
        int length = set.length;
        int num = length == 0 ? 0 : (1 << length);
        boolean[] ifVisited = new boolean[1 << length]; // 判断某种组合是否被判断过,
默认为false
        for (int i = 1; i < num; i++) {
            // 判断当前方案是否判断过
            if(ifVisited[i]){
                continue;
            }
```

```
            // 该方案之前没有判断过
            Set<Integer> subSet = new HashSet<>();
            int index = i;
            for (int j = 0; j < length; j++) {
                if((index & 1) == 1){
                    subSet.add(set[j]);
                }
                index >>= 1;
            }
            // 当前方案检查完毕
            ifVisited[i] = true;
            // eg: "10100"{0, 2}
            // 满减限额之和是否小于商品总金额,如果大于,continue
            if(!limitValueSumLtTotalAmount(subSet)){
                // i对应的方案不可行
                // 枚举当前方案的其他子集
                // (1 << length) - 1 --> 011111
                // i ^ ((1 << length) - 1)
                // --> i的二进制位子上 和 (1 << length) - 1: 相同为0,不同为1 --> 相当
于取反码 01011
                int reversedI = i ^ ((1 << length) - 1);
                // k 从 reversedI 逐渐减少到1 eg: 01011 --> 01010 --> 01001 -->
01000 --> ...
                for(int k = reversedI; k > 0;){
                    int subI = i | k; // 这些也是不可行的方案
                    ifVisited[subI] = true;
                    k = k - 1;// k每次自减1
                    k = k & reversedI; // 保证在k递减的过程中,reversedI为0的位置,k始
终为0
                }
                continue;
            }
            // 小于等于,继续执行
            result.add(subSet);
        }
        return result;
    }
```