Задачка 15. Василий и скидки

Введение

Программист Василий пишет для соседнего магазина программу, автоматизирующую кассовые операции. Кассир вводит в программу отпускаемые покупателю товары, а программа считает конечную стоимость и печатает чек на кассовом аппарате. Например, покупатель может прийти с таким набором товаров:

Пример 1

Номер п/п	Наименование товара	Цена за штуку	Скидка на штуку	Количество	Итого
1	Бутылка кефира	33,98	0,00	2	67,96
2	Полбатона	17,81	0,00	3	53,43
3	Валенки	291,71	0,00	5	1 458,55
4	Ушанка	19,99	0,00	7	139,93
Итого:					**1 719,87**

Покупателям неудобно отсчитывать копейки, поэтому администрация магазина попросила Василия округлять конечную сумму вниз до ближайшего целого. В приведенном примере нужно сделать покупателю скидку в 87 копеек, чтобы конечная стоимость составила ровно 1719 рублей. Однако по бухгалтерским причинам скидка должна быть распределена по строкам так, чтобы скидка на штуку в каждой строке выражалась целым числом копеек. Например, так:

Пример 1 (со скидкой)

Номер п/п	Наименование товара	Цена за штуку	Скидка на штуку	Количество	Итого
1	Бутылка кефира	33,98	0,42	2	67,12
2	Полбатона	17,81	0,01	3	53,40
3	Валенки	291,71	0,00	5	1 458,55
4	Ушанка	19,99	0,00	7	139,93
Итого:					**1 719,00**

Такое распределение не всегда возможно. Например:

Пример 2

Номер п/п	Наименование товара	Цена за штуку	Скидка на штуку	Количество	Итого
1	Бутылка кефира	333,34	0,00	3	1 000,02
Итого:					**1 000,02**

В этом случае нужно назначить скидку в 2 копейки, однако это невозможно сделать. Даже если на единственную строку назначить скидку в 1 копейку за штуку, то суммарная скидка составит уже 3 копейки. Поэтому Василию приходится разбивать строку на две. Выбрав произвольную строку с количеством q, можно разбить ее на две строки с количествами 1 и $(q - 1)$. В этом случае заведомо будет возможным распределить всю скидку на одну строку с количеством 1:

Пример 2 (со скидкой)

Номер п/п	Наименование товара	Цена за штуку	Скидка на штуку	Количество	Итого
1	Бутылка кефира	333,34	0,02	1	333,32
2	Бутылка кефира	333,34	0,00	2	666,68
Итого:					**1 000,00**

Однако Василий хочет избежать разделения строк всегда, когда это возможно. Исходя из этого, он определяет требования к алгоритму поиска возможного распределения.

Постановка задачи

Алгоритм должен принимать на вход массив количеств товаров (целых чисел) и сумму скидки для распределения. Сумму для простоты предлагается измерять в копейках, тогда она должна быть целочисленной и не может превышать 99. Итак, на вход алгоритма подается уравнение:

$$\sum_{i=1}^{N} \alpha_i x_i = B,$$

где

$$B \in \mathbb{Z}, 0 \le B \le 99,$$
$$\forall i : \alpha_i \in \mathbb{N},$$
$$\forall i : x_i \in \mathbb{Z}, x_i \ge 0.$$

Для примера 1 уравнение имеет вид $2x_1 + 3x_2 + 5x_3 + 7x_4 = 87$ и имеет решение $x_1 = 42$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0$.

Для примера 2 уравнение имеет вид $3x_1 = 2$ и целочисленных решений не имеет.

Василий создал интерфейс, объявляющий метод для решения такого уравнения:

```
1 public interface EquationSolver {
2     @NotNull
3     int[] solve(int number, @NotNull int[] factors) throws NoSolutionException;
4 }
```

Здесь *B* обозначено как `number`, а коэффициенты α_i — как `factors`. Метод должен возвращать любое из решений (если их несколько) и должен выбрасывать `NoSolutionException` тогда и только тогда, когда решений не существует.

Затем Василий написал его реализацию методом простого перебора:

```
1 public class EquationSolverImpl implements EquationSolver {
2     @NotNull
3     @Override
4     public int[] solve(int number, @NotNull int[] factors) throws NoSolutionException {
5           assert number >= 0 && number <= 99;
6           for (int factor : factors) {
7                 assert factor >= 1;
8           }
9
10          // массив с перебираемыми значениями x<sub>i</sub>.
11          int[] row = new int[factors.length];
12
13          // вначале инициализируем его нулями
14          Arrays.fill(row, 0);
15
```

```
16          // далее в цикле перебираем возможные значения и проверяем, сошлось ли равенство
17          do {
18                final int sum = calculateSum(row, factors);
19                if (sum < number) {
20                      // еще можно увеличить перебираемую сейчас переменную
21                      row[0]++;
22                } else if (sum == number) {
23                      // Бинго!
24                      return row;
25                } else if (sum > number) {
26                      // уже больше, чем надо. Сбрасываем до нуля перебираемую сейчас переменную
27                      // и увеличиваем на 1 следующую. Если и она уже слишком большая, то сбрасываем и ее
28                      // и увеличиваем на 1 следующую. И так далее.
29                      int i = 0;
30                      do {
31                            row[i] = 0;
32                            i++;
33                            if (i < row.length) {
34                                  row[i]++;
35                            } else {
36                                  // уже все перебрали :(
37                                  throw new NoSolutionException();
38                            }
39                      } while (row[i] * factors[i] > number);
40                }
41          } while (true);
42    }
43
44    private int calculateSum(int[] summands, int[] factors) {
45          assert summands.length == factors.length;
46
47          int sum = 0;
48          for (int i = 0; i < summands.length; i++) {
49                sum += summands[i] * factors[i];
50          }
51          return sum;
52    }
53 }
```

Далее Василий решил оценить, сколько времени может потребоваться для выполнения такого перебора. Он написал тест, который осуществляет множество запусков реализованного метода с различными входными параметрами и выводит в консоль максимальное время его выполнения:

```
1 @RunWith(AtUnit.class)
2 public class EquationSolverTest {
3     @Unit
4     private final EquationSolver equationSolver = new EquationSolverImpl();
5
6     @Test
7     public void solveTest() {
8           List<int[]> factorsList = new ArrayList<int[]>();
9           factorsList.add(new int[]{3});
10          factorsList.add(new int[]{2, 3});
11          factorsList.add(new int[]{2, 3, 2});
12          factorsList.add(new int[]{2, 3, 4, 5, 6, 7, 8, 9, 12, 17});
13          factorsList.add(new int[]{13, 11, 17, 19});
14          factorsList.add(new int[]{5, 7});
15          factorsList.add(new int[]{3, 6, 9, 12, 15});
16          factorsList.add(new int[]{3, 5, 7, 9, 11, 13, 15, 17, 19});
17          factorsList.add(new int[]{2, 4, 6});
18          factorsList.add(new int[]{2, 4, 6, 8, 10, 12, 14, 16, 18});
```

```
19
20          long maxDelta = 0;
21
22          for (int i = 0; i < 100; i++) {
23               for (int[] factors : factorsList) {
24                    long startTime = System.currentTimeMillis();
25                    try {
26                         equationSolver.solve(i, factors);
27                    } catch (NoSolutionException ignore) {
28
29                    }
30                    long endTime = System.currentTimeMillis();
31
32                    maxDelta = Math.max(maxDelta, endTime - startTime);
33               }
34          }
35
36          System.out.println("MaxDelta is " + maxDelta + " ms.");
37     }
38 }
```

Запустив этот тест, Василий увидел, что на его машине выводимое в консоль максимальное время выполнения составляет около 1500 мс. Это время расстраивает Василия. Помогите ему оптимизировать алгоритм.