ЧЕБЫШЕВСКИЙ СБОРНИК
Том 19. Выпуск 2

УДК 519.68:159.955 DOI 10.22405/2226-8383-2018-19-2-431-445

Пользовательские рекурсивные функции в Maxima[1]

Есаян Альберт Рубенович — доктор педагогических наук, профессор, Институт стратегии развития образования РАО.
e-mail: esayanalbert@mail.ru
Добровольский Николай Михайлович — доктор физико-математических наук, профессор, заведующий кафедрой алгебры, математического анализа и геометрии Тульского государственного педагогического университета им. Л. Н. Толстого.
e-mail: dobrovol@tsput.ru

Аннотация

Мы рассмотрим проблему деления прямоугольного параллелепипеда на конечное число непересекающихся кубов для некоторых жадных алгоритмов. Сформулированные задачи решаются серией блок-функций с прямой и косвенной (взаимной) рекурсией, написанных на языке программирования свободной программной системы *Maxima*. Все построенные функции проверяются контрольными вычислениями. Заметим, что на попарно различные кубы разделить прямоугольный параллелепипед невозможно.

Язык программирования системы *Maxima* используется исходя из следующих соображений. Постановки решаемых в данной статье задач вполне понятны и студенту, и школьнику. С рекурсией они также знакомы. Так что дело лишь в выборе языка программирования для реализации предлагаемых алгоритмов. И здесь язык системы *Maxima* вполне уместен. Дело в том, что в последнее время школы и вузы по многим причинам из многочисленных математических пакетов вынуждены выбирать для использования свободно распространяемое программное обеспечение. Лидерами среди таких пакетов являются кроссплатформенные системы *Maxima* и *GeoGebra*. Поэтому разговор об особенностях создания пользовательских рекурсивных функций на языке программирования Maxima своевременен и полезен.

Ключевые слова: прямоугольный параллелепипед, куб, прямая рекурсия, взаимная рекурсия, свободное программное обеспечение, *Maxima*, *GeoGebra*.

Библиография: 14 названий.

Для цитирования:
А. Р. Есаян, Н. М. Добровольский. Пользовательские рекурсивные функции в Maxima // Чебышевский сб. 2018. Т. 19, вып. 2. С. 431–445.

[1]Статья подготовлена в рамках государственного задания ФГБНУ «Институт стратегии развития образования Российской академии образования» (проект № 27.6122.2017/БЧ).

CHEBYSHEVSKII SBORNIK
Vol. 19. No. 2

UDC 519.68:159.955 DOI 10.22405/2226-8383-2018-19-2-431-445

User recursive functions in Maxima

Esayan Albert Rubenovich — doctor of pedagogical sciences, professor, Institute of Educational Development Strategy RAO.
e-mail: esayanalbert@mail.ru
Dobrovolsky Nikolai Mihailovich — doctor of physical and mathematical sciences, professor, head of the department algebra, calculus and geometry of the Tula State L. N. Tolstoy Pedagogical University.
e-mail: dobrovol@tsput.ru

Abstract

We consider the problem of dividing a rectangular parallelepiped on finite number of disjoint cubes for some greedy algorithms. The formulated problems are solved by a series of block-functions with direct and indirect (mutual) recursion, written in the programming language of the free *Maxima* software system. All constructed functions are checked by control calculations. Note that it is impossible to divide a rectangular parallelepiped into pairs of different cubes.

The programming language of system *Maxima* is used for the following reasons. Statements of the problems solved in this article are quite clear to both the student of secondary school and student of higher education institution. They are also familiar with recursion. So it is only a matter of choosing a programming language for the implementation of the proposed algorithms. And here the language of the *Maxima* system is quite appropriate. The fact is that in recent years, schools and universities for many reasons, from the many mathematical packages are forced to choose to use freely distributed software. The leaders among such packages are cross-platform *Maxima* and *GeoGebra* systems. Therefore, the talk about the features of creating user-defined recursive functions in the *Maxima* programming language is timely and useful.

Keywords: rectangular parallelepiped, cub, direct recursion, mutual recursion, free software, *Maxima*, *GeoGebra*.

Bibliography: 14 titles.

For citation:
A. R. Esayan, N. M. Dobrovol'skii, 2018, "User recursive functions in Maxima" , *Chebyshevskii sbornik*, vol. 19, no. 2. pp. 431–445.

1. Введение

В последнее время школы и вузы по многим причинам вынуждены переходить на свободно распространяемое программное обеспечение (СПО) [12]. Среди математических пакетов несомненными лидерами здесь являются кроссплатформенные системы *Maxima* (*WxMaxima*) [5, 6, 13] и *GeoGebra*. И хотя в Интернете по *Maxima* имеются немало доступных учебно-методических разработок, ряд важных вопросов описаны в них недостаточно полно или не совсем правильно. Например, это касается рекурсии, используемой в информатике как практически ориентированное знание с изящным и эффективным инструментарием для реальных вычислений. В данной статье затрагивается весьма важный и интересный вопрос об особенностях создания пользовательских рекурсивных функций в *Maxima* и работе с ними. Делается это на различных вариантах наглядной и интересной геометрической задачи о разрезании прямоугольного параллелепипеда на кубы, используя те или правила.

2. Постановка задач

Ниже дается постановка и программное решение нескольких задач на вырезание кубов из прямоугольного параллелепипеда. Пусть P – прямоугольный параллелепипед со сторонами x, y и z, длины которых являются натуральными числами. Не ограничивая общности можно считать $0<x \le y \le z$. Во всех предлагаемых задачах требуется составить рекурсивную программу-функцию, по которой подсчитывается, сколько всего кубов с целыми сторонами можно вырезать из P, если руководствоваться теми или иными правилами. Эти правила и конкретизируются в приведенных ниже задачах (A1-A4, B1-B5), причем в любом случае на каждом шаге вырезается куб с наибольшей возможной стороной, то есть речь всегда идет о некотором жадном алгоритме [11].

Если прямоугольный параллелепипед P разрезан на кубы и N их количество, то $\lfloor y/x \rfloor \cdot \lfloor z/x \rfloor \le N \le x \cdot y \cdot z$ ($\lfloor a \rfloor$ в Maxima – это функция floor(a) (пол)). Что касается нахождения минимального значения Nm для N, то в общем виде решение такой задачи, по-видимому, дело не простое. По крайней мере нам оно неизвестно. А результаты вычислений по приводимым ниже рекурсивным функциям для подсчета N по специальным правилам можно рассматривать как получение оценки сверху для Nm при конкретных P.

Отметим, что жадные алгоритмы разрезания P не всегда дают наименьшее значение для N. Пусть, например, рассматривается P со сторонами $6 \times 6 \times 5$. При разметке основания P, приведенной на рис. 1, разрезать P можно на $2 \times 2 + 3 \times 3 = 13$ кубов. Если же использовать жадный алгоритм, каждый раз вырезая куб с наибольшим возможным ребром, то получим $1 + 6 \times 5 + 5 \times 5 = 56$ кубов.

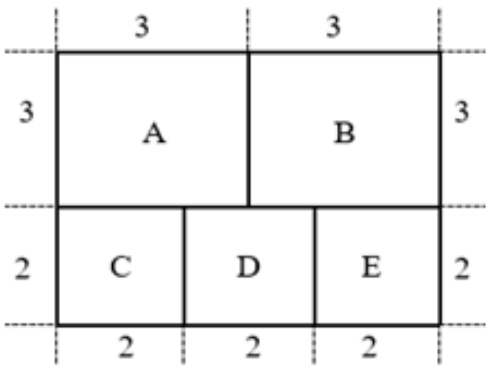

Рис. 1: Схема для разрезания на кубы параллелепипеда размером $5 \times 6 \times 6$

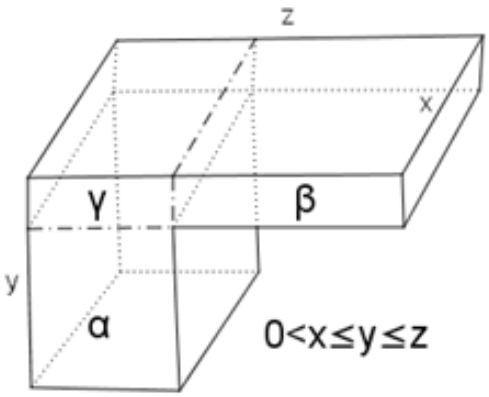

Рис. 2: Вырезание кубов из параллелепипеда размером $x \times y \times z$

Задача А1. *Разрезания тела рис. 2 на 3 части.* Составить рекурсивную программу-функцию для подсчета общего количества кубов, получаемых вырезанием их из прямоугольного параллелепипеда, если руководствоваться такими правилами:

1. параллелепипед и первый вырезаемый из него куб должны иметь общее ребро. Далее вырезается наибольшее возможное количество таких же кубов, как и в первый раз, если, конечно, это возможно. После выполнения первого пункта в общем случае от параллелепипеда останется тело в виде "уголка" (буквы "Г"), показанное на рис. 2. В нем части α, β и γ – прямоугольные параллелепипеды, причем части α и γ, части β и γ, или все три части α, β и γ могут отсутствовать;
2. при наличии всех трех частей оставшееся тело разрежем на три части α, β и γ. При отсутствии только α или только β тело уже является прямым параллелепипедом. При отсутствии и α, и β нет и тела;
3. если во втором пункте получены параллелепипеды, то к каждому из них рекурсивно применим процедуры пунктов 1 и 2, в противном случае вырезать кубы уже не из чего и процесс завершен.

Решение. См. ниже функции cutting и cut1

Задача В1. *Разрезания тела рис. 2 на 3 части.* Решим задачу А1, но результат должен быть возвращен в виде матрицы с двумя строками. В первой из них должны быть размещены длины сторон последовательно вырезаемых кубов, а во второй – их количество. Например, по параллелепипеду со сторонами $a = 3$, $b = 6$ и $c = 8$ должны получить матрицу $\begin{pmatrix} 3 & 2 & 1 \\ 4 & 3 & 12 \end{pmatrix}$.

Решение. См. ниже функцию cut1.

Задача А2. *Разрезания тела на 2 части по принципу "не у́же".* Составить рекурсивную программу-функцию для подсчета общего количества кубов, получаемых вырезанием их из прямоугольного параллелепипеда, если руководствоваться такими правилами:

1. см. пункт 1 из А1;
2. при наличии обеих частей α и β исходный параллелепипед разрежем на две части так, что часть γ добавляется к α, если α "не у́же" β ($\mathrm{mod}(z, x) \geq \mathrm{mod}(y, x)$), иначе – часть γ добавляется к β. При наличии только α или только β тело уже является параллелепипедом. При отсутствии и α, и β нет и тела;
3. см. пункт 3 из А1.

Решение. См. ниже функцию cut2.

Задача B2. *Разрезания тела на 2 части по принципу "не у́же".* Решим задачу A2, но результат должен быть возвращен в виде матрицы с двумя строками. В первой из них должны быть размещены длины сторон последовательно вырезаемых кубов, а во второй – их количество.

Решение. См. ниже функцию cut2m.

Задача A3. *Разрезания тела на 2 части по принципу "у́же".* Составить рекурсивную программу-функцию для подсчета общего количества кубов, получаемых вырезанием их из прямоугольного параллелепипеда, если руководствоваться такими правилами:

1. см. пункт 1 из A1;
2. при наличии обеих частей α и β исходный параллелепипед разрежем на две части так, что часть γ добавляется к α, если α "у́же" β ($\mathrm{mod}(z,\, x) < \mathrm{mod}(y,\, x)$), иначе – часть γ добавляется к β. При наличии только α или только β тело уже является параллелепипедом. При отсутствии и α, и β нет и тела;
3. см. пункт 3 из A1.

Решение. См. ниже функцию cut3.

Задача B3. *Разрезания тела на 2 части по принципу "у́же".* Решим задачу A3, но результат должен быть возвращен в виде матрицы с двумя строками. В первой из них должны быть размещены длины сторон последовательно вырезаемых кубов, а во второй – их количество.

Решение. См. ниже функцию cut3m.

Задача A4. *Случайные разрезания тела на 2 части.* Составить рекурсивную программу-функцию для подсчета общего количества кубов, получаемых вырезанием их из прямоугольного параллелепипеда, если руководствоваться такими правилами:

1. см. пункт 1 из A1;
2. при наличии обеих частей α и β исходный параллелепипед разрежем на две части, добавляя γ к α или γ к β случайным образом с помощью генератора псевдослучайных чисел. При наличии только α или только β тело уже является параллелепипедом. При отсутствии и α, и β нет и тела.
3. см. пункт 3 из A1.

Решение. См. ниже функцию cut4.

Задача B4. *Случайные разрезания тела на 2 части.* Решим задачу A4, но результат должен быть возвращен в виде матрицы с тремя строками. В первой из них должны быть размещены длины сторон последовательно вырезаемых кубов, во второй – их количество, а в третьей – тип разрезания получившегося тела (например, 1, если разрезание тела проводится по принципу "не у́же" (см. A2, B2) и 0 – если по принципу "у́же" (см. A3, B3).

Решение. См. ниже функцию cut4m.

Задача B5. *Многократные случайные разрезания на 2 части.* Написать функцию, по которой при конкретных характеристиках генератора случайных чисел задача B4 решается n

раз и среди полученных результатов вычислений отыскивается вариант разрезания прямоугольного параллелепипеда на наименьшее количество кубов.

Решение. См. ниже функцию find_min.

3. Решение задач A1 и B1

Будем считать, что все приведенные ниже пользовательские функции и соответствующие им контрольные примеры размещены в одном *Maxima*-документе и, следовательно, все они доступны без каких-либо ухищрений в любой точке документа. Рассматривая конкретные задачи, мы приводим соответствующие фрагменты этого документа с ячейками ввода и вывода, имеющими сплошную нумерацию.

Задача A1 может быть решена любой из приведенными ниже блок-функций cutting(x, y, z) и cut1(x, y, z), где x, y, z - длины сторон исходного параллелепипеда P. Эти функции являются рекурсивными с прямой рекурсией и симметрическими, то есть при обращении к ним возможны любые перестановки аргументов. Относительно рекурсивной триады для данного случая отметим следующее. *Параметрами задачи* естественно считаются длины сторон параллелепипеда P – x, y, z. *Базой рекурсии* для обеих функций являются вырожденные параллелепипеды, в которых в процессе рекурсивных вызовов длина по крайней мере одной стороны становится равной 0. *Декомпозиция* четко прописана в постановке задачи и определяется теми тремя параллелепипедами, которые получаются после вырезания одного или более кубов с наибольшим возможным ребром. Фактически cutting и cut1 устроены одинаково и cut1 представляет собой лишь компактный вариант кода для cutting.

/* *Код функции* cutting (*разрезание тела на* 3 *части*). */

```
(%i1) cutting(x, y, z):=block([r],
        if x*y*z=0 then 0
        else ([x, y, z]:sort([x, y, z]), r:floor(y/x)*floor(z/x),
         if mod(y, x)=0 and mod(z, x)=0
             then r
        elseif mod(y, x)=0 and mod(z, x)\neq 0
             then r+cutting(x, y, mod(z, x))
    elseif mod(y, x)\neq 0 and mod(z, x)=0
             then r+cutting(x, mod(y, x), z)
        else (r+cutting(x, y-mod(y, x), mod(z, x))+
             cutting(x, mod(y, x), z-mod(z, x))+
             cutting(x, mod(y, x), mod(z, x)))))$
```

/* *Контрольные примеры* */

```
(%i2) [cutting(5,5,5), cutting(2, 3, 5), cutting(37, 59, 752)];
(%o2) [1, 16, 7329]
```

/* *Код функции* cut1 (*компактный вариант кода для* cutting). */

```
(%i3) cut1(x, y, z):=block(
         if x*y*z=0 then 0
         else ([x, y, z]:sort([x, y, z]),
          floor(y/x)*floor(z/x)+
              cut1(x, y-mod(y, x), mod(z, x))+
              cut1(x, mod(y, x), z-mod(z, x))+
              cut1(x, mod(y, x), mod(z, x))))$
```

/* *Контрольные примеры* */

```
(%i4) [cut1(1, 1, 1), cut1(7, 7, 7), cut1(6, 4, 3),
         cut1(5, 7, 9), cut1(7, 4, 3), cut1(60, 4, 30)];
(%o4) [1, 1, 20, 72, 32, 165]
```

Задача B1 может быть решена приведенной ниже рекурсивной симметрической блок-функцией cut1m. Рекурсия в cut1m устроена так же, как и в cut1, но здесь при формировании результата вычислений в каждой рекурсивной ветви запоминаются размеры вырезаемых кубов и их количество и делается это с помощью функции addcol. При завершении любой рекурсивной ветви к результату вычислений добавляются лишние столбцы из нулей. Чтобы результат был более компактным эти ненужные столбцы целесообразно удалить, а все столбцы с одинаковыми элементами первой строки (с одинаковыми ребрами кубов) заменить одним столбцом, просуммировав соответствующие элементы второй строки. Сделать это можно с помощью приведенной ниже функции pretty.

/* *Код функции* cut1m (*разрезание тела на* 3 *части*). */

```
(%i5) cut1m(x, y, z):=block([ma],
          if x*y*z=0 then ma:matrix([0], [0])
          else ([x, y, z]:sort([x, y, z]),
           ma:matrix([x], [floor(y/x)*floor(z/x)]),
               ma:addcol(ma, cut1m(x, y-mod(y, x), mod(z, x))),
               ma:addcol(ma, cut1m(x, mod(y, x), z-mod(z, x))),
               ma:addcol(ma, cut1m(x, mod(y, x), mod(z, x)))))$
```

/* *Контрольные примеры* */

```
(%i9) a:3$ b:5$ c:7$
      [cutting(a, b, c), cut1(a, b, c), ta:cut1m(a, b, c),
         sum(ta[2][k], k, 1, length(ta[2]))];
```

$$(\%o9)\ [32, 32, \begin{bmatrix} 3 & 1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 2 & 9 & 0 & 0 & 0 & 3 & 0 & 12 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \end{bmatrix}, 32]$$

```
(%i10) [a*b*c, sum(ta[2][k]*ta[1][k]^3, k, 1, length(ta[2]))];
(%o10) [105, 105]
```

Инварианты. Проверку правильности вычислений по функциям cut1m (а также по cut2m, cut3m, pretty, ...) можно проводить по инварианту, каковым в данном случае является объем – суммарный объем всех полученных кубов должен быть равен объему исходного параллелепипеда P. Иными словами, если a, b и c – длины сторон исходного параллелепипеда, а ma – матрица результата с n столбцами, то обязано выполняться условие

$$a \cdot b \cdot c = \sum_{i=1}^{n} ma[2,k] \cdot ma[1,k]^3.$$

В терминах *Maxima* это условие можно заменить кодом

$$\mathrm{is}(\mathrm{equal}(a * b * c, (ma[1]^3).ma[2])),$$

возвращающим логическое значение *true*. Если же возвращается *false*, то, повидимому, код соответствующей функции нуждается в правке.

/* *Код функции* pretty *для компактного вывода по* cut1m, ...) */

```
(%i11) pretty(x, y, z, name):=block([cu, lon, le, tt, sho],
          cu:name(x, y, z), lon:cu[1],
          sho:sort(unique(lon), ordergreatp),
          sho:matrix(sho, 0*sho), le:length(sho[1]),
```

```
          for k:1 thru le do (tt:sho[1, k], su:0,
            for i:1 thru length(lon) do
              if cu[1, i]=tt then su:su+cu[2, i],
            sho[2, k]:su), submatrix(sho, le))$
```

/* *Контрольные примеры* */

```
(%i15) a:4$ b:5$ c:6$
       [cut1(a, b, c), cut1m(a, b, c), pretty(a, b, c, cut1m)];
```

$$(\%o15)\ [29, \begin{bmatrix} 4 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 4 & 0 & 0 & 0 & 16 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 4 & 2 & 1 \\ 1 & 4 & 2 \end{bmatrix}]$$

```
(%i16) lis:[pretty(1, 1, 1, cut1m), pretty(7, 7, 7, cut1m),
       pretty(6, 4, 3, cut1m), pretty(5, 7, 9, cut1m),
       pretty(7, 4, 3, cut1m), pretty(60, 4, 30, cut1m)];
```

$$(\text{lis})\ [\begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 7 \\ 1 \end{bmatrix}, \begin{bmatrix} 3 & 1 \\ 2 & 18 \end{bmatrix}, \begin{bmatrix} 5 & 4 & 2 & 1 \\ 1 & 1 & 8 & 62 \end{bmatrix}, \begin{bmatrix} 3 & 1 \\ 2 & 30 \end{bmatrix}, \begin{bmatrix} 4 & 2 \\ 105 & 60 \end{bmatrix}]$$

/* *Инварианты* */

```
(%i17) [is(equal((lis[1][1]^3).lis[1][2], 1*1*1)),
        is(equal((lis[2][1]^3).lis[2][2], 7*7*7)),
        is(equal((lis[3][1]^3).lis[3][2], 6*4*3)),
        is(equal((lis[4][1]^3).lis[4][2], 5*7*9)),
        is(equal((lis[5][1]^3).lis[5][2], 7*4*3)),
        is(equal((lis[6][1]^3).lis[6][2], 60*4*30))];

(%o17) [true, true, true, true, true, true]
```

4. Решение задач A2 и B2

Задача A2 может быть решена рекурсивной симметрической блок-функцией cut2(x, y, z), где x, y и z - длины сторон исходного параллелепипеда P. В cut2, как и в cut1, *базой рекурсии* являются вырожденные параллелепипеды, в которых в процессе рекурсивных вызовов длина по крайней мере одной из сторон становится равной 0. *Декомпозиция* прописана в постановке задачи и определяется теми двумя параллелепипедами, которые получаются после вырезания одного или более кубов с наибольшим возможным ребром.

/* *Код функции* cut2. */
/* *Разрезание тела на* 2 *части по принципу* “*не у́же*”. */

```
(%i18) cut2(x, y, z):=block([r],
         if x*y*z=0 then 0
         else ([x, y, z]:sort([x, y, z]), r:floor(y/x)*floor(z/x),
              if mod(z, x)\geq mod(y, x) then
                 (r:r+cut2(x, y, mod(z, x))+
                 cut2(x, mod(y, x), z-mod(z, x)))
              else
                 (r:r+cut2(x, y-mod(y, x), mod(z, x))+
                  cut2(x, mod(y, x), z))))$
```

/* *Контрольные примеры* */

```
(%19) [cutting(2, 5, 7), cut1(2, 5, 7), cut2(2, 5, 7),
        cutting(3, 4, 11), cut1(3, 4, 11), cut2(3, 4, 11)];
(%o19) [19, 19, 19, 47, 47, 40]
```

Задача B2 может быть решена приведенной ниже рекурсивной симметрической блок-функцией cut2m. Рекурсия в cut2m устроена так же, как и в cut2. Для компактной записи результата вычислений можно воспользоваться функцией pretty.

```
/* Код функции cut2m. */
/* Разрезание тела на 2 части по принципу "не ýже". */

(%i20) cut2m(x, y, z):=block([ma],
         if x*y*z=0 then ma:matrix([0], [0])
         else ([x, y, z]:sort([x, y, z]),
               ma:matrix([x], [floor(y/x)*floor(z/x)]),
               if mod(z, x)\geq mod(y, x) then
                  (ma:addcol(addcol(ma, cut2m(x, y, mod(z, x))),
                   cut2m(x, mod(y, x), z-mod(z, x))))
               else
                   ma:addcol(addcol(ma,
                      cut2m(x, y-mod(y, x), mod(z, x))),
                      cut2m(x, mod(y, x), z))), ma)$

/* Контрольные примеры */

(%i26) a:17$ b:43$ c:49$ cut2(a, b, c);
     ta:pretty(a, b, c, cut2m);
     [(ta[1]^3).ta[2], a*b*c, is(equal((ta[1]^3).ta[2], a*b*c))];
(%o24) 740

(ta) [ 17  15  13  9  8  7  4  3   2    1   ]
     [  4   2   1  3  4  1  9  4  178  534  ],

(%o26) [35819, 35819, true]
```

5. Решение задач A3 и B3

Задача A3 может быть решена рекурсивной симметрической блок-функцией cut3(x, y, z), где x, y и z - длины сторон исходного параллелепипеда P. Заметим, что коды функций cut2 и cut3 почти одинаковы. Код cut3 получен из кода cut2 заменой в нем во вложенном *if* условия
$\mathrm{mod}(z, x) \geq \mathrm{mod}(y, x)$ на $\mathrm{mod}(z, x) < \mathrm{mod}(y, x)$

```
/* Код функции cut3. */
/* Разрезание тела на 2 части по принципу "ýже". */

(%i27) cut3(x, y, z):=block([r],
         if x*y*z=0 then 0
         else ([x, y, z]:sort([x, y, z]), r:floor(y/x)*floor(z/x),
               if mod(z, x)<mod(y, x) then
                  (r:r+cut3(x, y, mod(z, x))+
                     cut3(x, mod(y, x), z-mod(z, x)))
               else
                  (r:r+cut3(x, y-mod(y, x), mod(z, x))+
                     cut3(x, mod(y, x), z)))$

/* Контрольные примеры */

(%i31) a:17$ b:43$ c:45$ cut3(a, b, c);
(%o31) 808
```

Задача B3 может быть решена приведенной ниже рекурсивной симметрической блок-функцией cut3m. Рекурсия в cut3m устроена так же, как и в cut3. Для удаления из результата вычислений лишних столбцов с нулевыми элементами можно воспользоваться функцией pretty.

```
/* Код функции cut3m. */
/* Разрезание тела на 2 части по принципу "ýже". */
(%i32) cut3m(x, y, z):=block([ma],
          if x*y*z=0 then ma:matrix([0], [0])
          else
             ([x, y, z]:sort([x, y, z]),
             ma:matrix([x], [floor(y/x)*floor(z/x)]),
             if mod(z, x)<mod(y, x) then
                (ma:addcol(addcol(ma, cut3m(x, y, mod(z, x))),
                    cut3m(x, mod(y, x), z-mod(z, x))))
             else
                 ma:addcol(addcol(ma,
                    cut3m(x, y-mod(y, x), mod(z, x))),
                    cut3m(x, mod(y, x), z))), ma)$

/* Контрольные примеры */
(%i38) a:17$ b:43$ c:49$ cut3(a, b, c);
      ta:pretty(a, b, c, cut3m);
      [(ta[1]^3).ta[2], a*b*c, is(equal((ta[1]^3).ta[2], a*b*c))];
(%o36) 832
```

$$\text{(ta)} \begin{bmatrix} 17 & 15 & 9 & 8 & 5 & 4 & 3 & 2 & 1 \\ 4 & 2 & 5 & 5 & 1 & 17 & 12 & 127 & 659 \end{bmatrix},$$

```
(%o38) [35819, 35819, true]

(%i39) test(name):=block([],
           [name(1, 1, 1), name(7, 7, 7), name(6, 4, 3),
            name(5, 7, 9), name(7, 4, 3), name(60, 4, 30),
            name(17, 45, 437), name(171, 437, 459)])$

(%i42) test(cut1); test(cut2); test(cut3);

(%o40) [1, 1, 20, 72, 32, 165, 3188, 3348]
(%o41) [1, 1, 20, 60, 32, 165, 2754, 10766]
(%o42) [1, 1, 20, 72, 32, 165, 2821, 6814]
```

6. Решение задач A4 и B4

Задача A4 может быть решена рекурсивной симметрической блок-функцией cut4(x, y, z), где x, y и z - длины сторон исходного параллелепипеда P. Рекурсивные обращения по cut4 реализуются или по схеме cut2, или по схеме cut3 в зависимости от нечетности или четности автоматически генерируемых псевдослучайных чисел.

Несколько слов о работе с псевдослучайными числами в *Maxima*. Для ядра генератора псевдослучайных чисел по целому числу *num* можно сформировать специальную последовательность S как результат, возвращаемый функцией make_random_state (S:make_random_state(num)), причем реально вместо *num* используется число mod(num, 2^{32}). Последовательности типа S состоят из 627 32 битных слов и любую из них с помощью функции set_random_state(S) можно использовать для инициализации ядра генератора. Далее,

при каждом выполнении функции random(*ran*), где *ran* – натуральное число, будет возвращаться целое псевдослучайное число μ между 0 и *ran*-1 ($0 \leq \mu \leq ran$-1, при равномерном распределении). Если следует заново сгенерировать ту же самую последовательность псевдослучайных чисел, то для этого достаточно предварительно повторно реализовать установку set_random_state(S), а затем уже выполнять серию обращений random(*ran*). Эти действия очень полезны при отладке и тестировании программ.

```
/* Код функции cut4. */
/* Случайные разрезания тела на 2 части. */
(%i43) cut4(x, y, z):=block([r, t],
         if x*y*z=0 then 0
         else
             ([x, y, z]:sort([x, y, z]), r:floor(y/x)*floor(z/x),
             t:oddp(random(ran)),
             if (t and mod(z, x)\geq mod(y, x)) then
                (r:r+cut4(x, y, mod(z, x))+
                   cut4(x, mod(y, x), z-mod(z, x)))
             else
                (r:r+cut4(x, y-mod(y, x), mod(z, x))+
                   cut4(x, mod(y, x), z))))$

/* Контрольные примеры */
(%i50) s1:make_random_state(654321)$
       set_random_state(s1)$ ran:1101$
       a:17$ b:19$ c:77$
       [cut1(a, b, c), cut2(a, b, c), cut3(a, b, c), cut4(a, b, c)];
(%o50) [834, 657, 834, 657]
```

Задача B4 может быть решена приведенной ниже рекурсивной симметрической блок-функцией cut4m. Рекурсия в cut4m устроена так же, как и в cut4. В данном случае результат вычислений возвращается в виде матрицы с тремя строками, в столбцах которой размещаются соответственно длины сторон кубов, их количество и тип разрезания тела перед очередным рекурсивным обращением: 1 – по схеме функции cut2, 0 – по схеме функции cut3. Результат вычислений по cut4m, как и по функциям cut1m-cut3m, содержит лишние столбцы с нулевыми элементами, а также повторяющиеся элементы первой строки (длины сторон кубов). В данном случае лишние столбцы желательно удалить, но сборку столбцов с одинаковым значением элементов первой строки проводить нельзя. В последнем случае информация третьей строки будет потеряна и восстановить последовательность вырезаний кубов из P уже будет нельзя. Поэтому вместо функции pretty здесь лучше использовать функцию rem00, удаляющую только лишние столбцы (см. ниже).

```
/* Код функций cut3m и rem00 */
/* Случайные разрезания тела на 2 части. */
(%i51) cut4m(x, y, z):=block([ma, t],
         if x*y*z=0 then ma:matrix([0], [0], [0])
         else
             ([x, y, z]:sort([x, y, z]),
             t:oddp(random(ran)) and mod(z, x)\geq mod(y, x),
             ma:matrix([x], [floor(y/x)*floor(z/x)], [charfun(t)]),
             if t then
                ma:addcol(addcol(ma, cut4m(x, y, mod(z, x))),
                   cut4m(x, mod(y, x), z-mod(z, x)))
             else
```

```
                ma:addcol(addcol(ma,
                   cut4m(x, y-mod(y, x), mod(z, x))),
                      cut4m(x, mod(y, x), z))), ma)$

(%i52) rem00(ma):=block(
       for k:1 thru length(ma[1]) do
           if ma[1, k]=0 then
              (ma:submatrix(ma, k), k:k-1), ma)$
```

/* *Контрольные примеры* */

```
(%i64) a:17$ b:19$ c:95$
       s1:make_random_state(654321)$ ran:1997$
       set_random_state(s1)$
       ta1:pretty(a, b, c, cut1m); ta2:pretty(a, b, c, cut2m);
       ta3:pretty(a, b, c, cut3m); ta4:rem00(cut4m(a, b, c));
       set_random_state(s1)$
       [cut1(a, b, c), cut2(a, b, c), cut3(a, b, c), cut4(a, b, c)];
```

(ta1)
$$\begin{bmatrix} 17 & 10 & 7 & 3 & 2 & 1 \\ 5 & 1 & 3 & 24 & 376 & 435 \end{bmatrix}$$

(ta2)
$$\begin{bmatrix} 17 & 10 & 9 & 8 & 7 & 3 & 2 & 1 \\ 5 & 1 & 1 & 1 & 1 & 10 & 352 & 450 \end{bmatrix}$$

(ta3)
$$\begin{bmatrix} 17 & 10 & 7 & 3 & 2 & 1 \\ 5 & 1 & 3 & 24 & 379 & 411 \end{bmatrix}$$

(ta4)
$$\begin{bmatrix} 17 & 10 & 9 & 1 & 1 & 7 & 5 & 2 & 1 & 3 & 2 & 1 & 1 & 1 & 2 & 1 & 1 \\ 5 & 1 & 1 & 81 & 90 & 2 & 2 & 10 & 20 & 8 & 3 & 4 & 14 & 36 & 336 & 32 & 170 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(%o64) [844, 821, 823, 815]

```
(%i66) [sum(ta1[2][k].ta1[1][k]^3, k, 1, length(ta1[1])),
       sum(ta2[2][k].ta2[1][k]^3, k, 1, length(ta2[1])),
       sum(ta3[2][k].ta3[1][k]^3, k, 1, length(ta3[1])),
       sum(ta4[2][k].ta4[1][k]^3, k, 1, length(ta4[1])), a*b*c];
       [sum(ta1[2][k], k, 1, length(ta1[2])),
       sum(ta2[2][k], k, 1, length(ta2[2])),
       sum(ta3[2][k], k, 1, length(ta3[2])),
       sum(ta4[2][k], k, 1, length(ta4[2]))];
```

(%o65) [30685, 30685, 30685, 30685, 30685]

(%o66) [844, 821, 823, 815]

7. Решение задачи B5

В блок-функции find_min(a, b, c, n) аргументы a, b, c – длины сторон исходного прямоугольного параллелепипеда P. Данная функция рекурсивной не является, но обращается к пользовательским рекурсивным функциям cut4(a, b, c) и cut4m(a, b, c). Выполняется find_min так. Инициализируется ядро генератора случайных чисел, циклом *for* n раз вычисляется cut4(a, b, c) и среди n полученных результатов запоминается тот, который дает наименьшее количество кубов mi при разрезании P, а также сответствующий ему номер $k0$ ($1 \le$ k0 $\le n$) параметра цикла. Далее снова и тем же самым способом инициализируется ядро генератора случайных чисел, $k0$-1 раз выполняется cut4(a, b, c) и, наконец, один раз выполняется rem00(cut4m(a, b, c)). Этим обеспечивается получение соответствующей матрицы разрезаний P на mi кубов. Результат возвращается в виде списка [mi, rem00(cut4m(a, b, c))].

Управлять поиском "лучшего" разрезания конкретного прямоугольного параллелепипеда P на кубы при использовании find_min можно с помощью следующих параметров: n – аргумент find_min, S – последовательность для инициализации ядра генератора и ran – аргумент random. В контрольных примерах для P со сторонами a=27, b=29, c=100 при

$$S : make\_random\_state(654321)\$\_ran : 101\$\_n : 100\$$$

получено 1170 кубов. Соответствующие вычисления по $cut1$, $cut2$ и $cut3$ дают соответственно 1320, 1237 и 1184 куба.

```
/* Код функции find_min. */
/* Выбор минимального из нескольких вариантов. */
(%i67) find\_min(a, b, c, n):=block([mi:inf, k0],
         set_random_state (S),
         for k:1 thru n do
            (r:cut4(a, b, c), if r<mi then (mi:r, k0:k)),
         set_random_state (S1),
         for k:1 thru k0-1 do cut4(a, b, c),
         [mi, rem00(cut4m(a, b, c))])$

/* Контрольные примеры */
(%i77) S:make_random_state(654321)$ ran:101$
       a:27$ b:29$ c:100$ n:100$
       [cut1(a, b, c), cut2(a, b, c), cut3(a, b, c)];
       ta:find_min(a, b, c, n)$ ta[1]; ma:ta[2];
{%o74} [1320, 1237, 1184]
{%o76} 1170
```

$$\text{(ma)} \begin{bmatrix} 27 & 19 & 8 & 3 & 1 & 2 & 1 & 1 & 3 & 1 & 2 & 1 & 8 & 3 & \dots & 2 & 1 \\ 3 & 1 & 4 & 12 & 18 & 9 & 4 & 38 & 10 & 18 & 8 & 32 & 6 & 10 & \dots & 650 & 200 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

```
(%i78) [a*b*c, sum(ma[2, k]*ma[1, k]^3, k, 1, length(ma[2])),
       sum(ma[2, k], k, 1, length(ma[2]))];
(%o78) [78300, 78300, 1170]
```

Отметим, что если g – наибольший общий делитель длин сторон a, b, c исходного параллелепипеда P, то подсчет количества вырезаемых из P кубов, можно заменить на подсчет количества вырезаемых кубов из параллелепипеда со сторонами a/g, b/g и c/g. В *Maxima* наибольший общий делитель чисел a, b, c можно получать как первый элемент списка, возвращаемого функцией ezgcd(a, b, c) (см. пример ниже).

```
/* Контрольные примеры */
(%i84) a:500$ b:700$ c:900$
       [g:ezgcd(a, b, c)[1], cut2(a, b, c), cut2(a/g, b/g, c/g)];
       [pretty(a, b, c, cut2m), ma:pretty(a/g, b/g, c/g, cut2m)];
       [ma, matrix(ma[1]*g, ma[2])];
(%o82) [100, 60, 60]
```

$$\text{(\%o83)} \left[\begin{bmatrix} 500 & 400 & 300 & 200 & 100 \\ 1 & 1 & 1 & 6 & 51 \end{bmatrix}, \begin{bmatrix} 5 & 4 & 3 & 2 & 1 \\ 1 & 1 & 1 & 6 & 51 \end{bmatrix}\right]$$

$$\text{(\%o84)} \left[\begin{bmatrix} 5 & 4 & 3 & 2 & 1 \\ 1 & 1 & 1 & 6 & 51 \end{bmatrix}, \begin{bmatrix} 500 & 400 & 300 & 200 & 100 \\ 1 & 1 & 1 & 6 & 51 \end{bmatrix}\right]$$

8. Заключение

Обсуждаемые в статье встроенные средства *Maxima*, связанные с построением блочных функций с прямой или косвенной рекурсией, хорошо иллюстрируются на задачах о разрезании прямоугольного параллелепипеда на кубы. Эти средства легко осваиваются, просты в использовании и могут служить хорошим инструментарием при обучении рекурсии как школьников, так и студентов. В заключение заметим, что разбить куб или параллелепипед на попарно различные кубы невозможно. Суть доказательства этого утверждения, можно найти в [7, 9].

СПИСОК ЦИТИРОВАННОЙ ЛИТЕРАТУРЫ

1. Гарднер М., Математические головоломки и развлечения. Пер. с английского Ю. Данилова. Изд. «Оникс», -М.: 1994.

2. Есаян А. Р. Вероятностная рекурсия // Изв. ТулГУ. Сер. “Математика. Механика. Информатика”. -Тула: Изд-во ТулГУ, 2000. - Т. 6. - Вып. 3. С. 56-61.

3. Есаян А. Р. Обучение алгоритмизации на основе рекурсии: Учеб. пособие для студентов пед. вузов. -Тула: Изд-во ТГПУ им. Л. Н. Толстого, 2001. -215 с.

4. Есаян А. Р. Рекурсия как общеобразовательная ценность // Образование как ценность: Сб. науч. тр. аспирантов и докторантов. Тула: Изд-во ТГПУ им. Л. Н. Толстого, 1998. С. 21-35.

5. Есаян А. Р., Чубариков В. Н., Добровольский Н. М., Якушин А. В. Maxima. Данные и графика. –Тула: Изд-во Тул. гос. пед. ун-та, 2011. -367 с.

6. Есаян А. Р., Чубариков В. Н., Добровольский Н. М., Якушин А. В. Программирование в Maxima. –Тула: Изд-во Тул. гос. пед. ун-та, 2012. -351 с.

7. Курляндчик Л., Розенблюм Г. Метод бесконечного спуска. http://ega-math.narod.ru/Quant/Descent.htm

8. Покровский В. Г. О разрезании многомерного параллелепипеда на параллелепипеды, Изв. вузов. Матем., 2001, № 10, С. 69–72; Russian Math. (Iz. VUZ), 45:10 (2001), С. 65–68

9. Brooks R. L., Smith C. A. B., Stone A. H., Tulle W. T., The Dissection of Rectangles into Squares, Duke Mathematical Journal 7, 1940, pp. 312-340.

10. Duijvestijn A. J. W., Simple perfect square of lowest order. Universität Twente 1978.

11. https://ru.wikipedia.org/wiki/`Жадный_алгоритм`

12. https://ru.wikipedia.org/wiki/`Свободное_программное_обеспечение`

13. Maxima Manual. Version 5.41.0, 2018 г. -1154 с. http://maxima.sourceforge.net/

14. Wolfram D. A. Solving Generalized Fibonacci Recurrences // Fib. Quart. 1998. – Т. 36, вып. 2.

REFERENCES

1. Gardner M., 1994, *Matematicheskie golovolomki i razvlecheniya,* Per. s anglijskogo YU. Danilova. Izd. «Oniks», M.

2. Esayan A. R., 2000, "Veroyatnostnaya rekursiya" , *Izv. TulGU. Ser. "Matematika. Mexanika. Informatika"*. Tula: Izd-vo TulGU, T. 6. Vip. 3. pp. 56-61.

3. Esayan A. R., 2001, *Obuchenie algoritmizacii na osnove rekursii*, Ucheb. posobie dlya studentov ped. vuzov. -Tula: Izd-vo TGPU im. L. N. Tolstogo, 215 P.

4. Esayan A. R., 1998, "Rekursiya kak obshheobrazovatel'naya cennost" , *Obrazovanie kak cennost': Sb. nauch. tr. aspirantov i doktorantov.* Tula: Izd-vo TGPU im. L. N. Tol-stogo, pp. 21-35.

5. Esayan A. R., Chubarikov V. N., Dobrovol'skij N. M., Yakushin A. B., 2011, *Maxima. Dannye i grafika.* Tula: Izd-vo Tul. gos. ped. un-ta, 367 P.

6. Esayan A. R., Chubarikov V. N., Dobrovol'skij N. M., Yakushin A. B., 2012, *Programmirovanie v Maxima.* Tula: Izd-vo Tul. gos. ped. un-ta, 351 P.

7. Kurlyandchik L., Rozenblyum G. *Metod beskonechnogo spuska.* http://ega-math.narod.ru/Quant/Descent.htm

8. Pokrovskij V. G., 2001, "O razrezanii mnogomernogo parallelepipeda na parallele-pipedy" , *Izv. vuzov. Matem.,* № 10, S. 69–72; *Russian Math. (Iz. VUZ)*, 45:10, S. 65–68

9. Brooks R. L., Smith C. A. B., Stone A. H., Tulle W. T., 1940, "The Dissection of Rectangles into Squares" , *Duke Mathematical Journal* 7, pp. 312-340.

10. Duijvestijn A. J. W., 1978, "Simple perfect square of lowest order" , *Universität Twente*

11. https://ru.wikipedia.org/wiki/`ZHadnyj_algoritm`

12. https://ru.wikipedia.org/wiki/`Svobodnoe_programmnoe_obespechenie`

13. Maxima Manual. Version 5.41.0, 2018 г. -1154 с. http://maxima.sourceforge.net/

14. Wolfram D. A., 1988, "Solving Generalized Fibonacci Recurrences" , *Fib. Quart.* T. 36, 2.

Получено 23.04.2018
Принято в печать 17.08.2018