ЧЕБЫШЕВСКИЙ СБОРНИК
Том 16 Выпуск 3 (2015)

УДК

ПРОСТОЙ И ОБОБЩЕННЫЙ ПОИСК ЭЛЕМЕНТОВ В ГНЕЗДОВЫХ МАССИВАХ И ИХ ЗАМЕЩЕНИЕ

С. Г. Григорьев (Москва), А. Р. Есаян (Тула)

Аннотация

В статье приводится серия пользовательских рекурсивных функций для разнообразных задач поиска и замены элементов в гнездовых массивах. Последние определяются рекурсивно так, как это сделано в системе инженерных и научных вычислений *PTC Mathcad Prime*, то есть в виде матриц, элементы которых могут быть скаляры, строки и снова гнездовые массивы. Некоторые задачи поиска рассмотрены в [1-4]. Нашей задачей являлось развитие имеющихся и создание новых средств, связанных как с обычным, так и с обобщенным поиском и замещением элементов в гнездовых массивах. Пусть A — скаляр, строка или гнездовой массив, B — гнездовой массив. Задачи с обобщенными вхождениями A в B (обобщенным поиском A в B), и замещением таких вхождений возникают тогда, когда в A или в B могут присутствовать специальные элементы, отождествляемые с любым скаляром, строкой или гнездовым массивом. В статье сформулировано 10 задач. Для каждой из них предложено одно или более решений в виде функций на языке программирования *PTC Mathcad Prime*. Все созданные функции протестированы на большом количестве примеров, но тесты приведены не полностью.

Ключевые слова: гнездовой массив, матрица, дерево, рекурсия, рекурсивная функция, поиск и замещение, обобщенный поиск, *PTC Mathcad*.

Библиография: 7 названий.

SIMPLE AND GENERALIZED ELEMENTS SEARCH IN NESTED ARRAYS AND THEIR REPLACEMENT

S. G. Grigoryev (Moscow), A. R. Esayan (Tula)

Abstract

The article provides a series of custom recursive functions for a variety of tasks of searching and replacing elements in nested arrays. The latter are defined recursively, as is done in the system of engineering and scientific calculations, *PTC Mathcad Prime*, then there are matrices whose elements can be scalars, strings, and again the nesting arrays. Some problems of the search are considered in [1-4]. Our goal was the development of existing and creation of new means associated with both simple and generalized searching and replacing elements in nested arrays. Let A be a scalar, a string, or a nested array, B — nested array. The problem with generalized occurrences of A in B (generalized searching A in B) and replaced of such occurrences arise when A or B to be included by special elements associated with any scalar, string or nested array. In the article are formulated 10 problems. For each suggested one or more solutions as functions in a programming language *PTC Mathcad Prime*. All the features are tested on a large number of examples, but the tests are not completely.

Keywords: nested array, matrix, tree, recursion, recursive function, search and replace, generalized searching, PTC Mathcad.

Bibliography: 7 titles.

1. Введение

В системе инженерных и научных вычислений *PTC Mathcad Prime* [6,7] двумерные гнездовые (вложенные) массивы определяются рекурсивно, как гнездовые матрицы, то есть матрицы, элементами которых могут быть скаляры, строки и (или) снова гнездовые матрицы. Если в гнездовом массиве все элементы скаляры или строки, то он является простым массивом. Именно так понимаются гнездовые массивы и в предлагаемой статье, где речь идет о простом и обобщенном поиске элементов в таких массивах. Все приводимые примеры функций реализованы на языке программирования *PTC Mathcad Prime* версии 3.1.

Гнездовой массив удобно интерпретировать деревом, корнем которого является сам массив, а от него идут дуги к элементам — скалярам, строкам и гнездовым массивам. От массивов снова идут дуги к их элементам — скалярам, строкам и массивам и т. д. Листьями подобного дерева являются скаляры или строки — конечные элементы, не имеющие последующих ссылок. Исходя из этого, для гнездовых массивов можно использовать многие термины, применяемые для деревьев: такие как, листья, уровни вложенности элементов, высота (максимальная глубина вложенности элементов массива) и т. п.

Обратите внимание, что гнездовые массивы определяются рекурсивно. И в силу специфики таких массивов наиболее простая и эффективная их обработка реализуется рекурсивными алгоритмами [5]. Как объекты хранения данных гнездовые массивы изучены слабо, а встроенных средств для работы с ними мало. Среди них:

- *оператор сравнения двух гнездовых массивов A и B*. Если A и B имеют одинаковую структуру и все соответствующие друг другу элементы равны, то считается, что $A{=}B$, точнее $(A{=}B){=}1$ (1 — истина). В противном случае, то есть когда структуры A и B не одинаковы или они одинаковы, но хотя бы одна пара соответствующих друг другу элементов не равны, считается, что $A{\neq}B$, то есть, $(A{=}B){=}0$ (0 — ложь);
- вытяжка из гнездового массива B элемента x, находящегося в позиции $(i1, j1)$ на первом уровне вложенности, в позиции $(i2, j2)$ на втором уровне вложенности и т. д. в позиции (ik, jk) на k уровне вложенности, где $i1$, $i2$, …, ik и $j1$, $j2$, …, jk — соответственно номера строк и номера столбцов матриц. При этом

$$\left(\left((B_{i1,j1})_{i2,j2}\right)_{\ddots}\right)_{ik,jk} = x;$$

- оператор замещения элемента первого уровня вложенности гнездового массива B числом, строкой или гнездовым массивом Z. Эту роль выполняет обычный оператор присваивания ":=": $B_{i,j} := Z$ (i — номер строки, j — номер столбца). Подчеркнем, что замещаемый элемент должен находиться на первом уровне вложенности B;
- встроенные функции *rows*, *cols*, *stack*, *augment*, *submatrix*, …, работающие лишь на первом уровне вложенности своих матричных аргументов.

В документации *PTC Mathcad Prime* предложена также специальная рекурсивная функция для нахождения высоты гнездового массива.

2. Постановка задач

Пусть A — скаляр, строка или гнездовой массив, B — гнездовой массив. Говорят, что A входит в B, если A равно B или равно какому-либо элементу B на любом его уровне вложенности. Ряд рекурсивных алгоритмов обработки гнездовых массивов обсуждается в работах [1-4]. В частности, в [1] решаются такие задачи на вхождение.

Задача *XX*. Выяснить, входит ли A в B?.

Задача *YY*. Подсчитать количество вхождений A в B на всех уровнях вложенности B.

Нашей задачей является развитие имеющихся и создание новых средств, связанных с поиском и замещением элементов в гнездовых массивах. Прежде чем формулировать решаемые в данной статье задачи, нам необходимо ввести понятие обобщенного равенства двух гнездовых массивов A и B и уточнить понятие позиции элемента в гнездовом массиве. К этому и приступим. Будем считать, что в A или в B могут быть элементы, равные любому другому элементу.

Прежде всего, надо условиться об обозначении таких "обобщенных" элементов. Если в A и (или) B нет специальных элементов NaN (Not a Number), то данный скаляр подходит для наших целей. При наличии в A или в B элементов NaN в качестве обобщенного значения может быть взято любое достаточно большое число или какая-либо строка, которые не встречаются ни в A, ни в B. Мы будем считать, что для обобщенных значений используется именно скаляр NaN.

Определение 1. *Два гнездовых массива* A *и* B *обобщенно равны, если они совпадают по структуре и все их соответствующие друг другу элементы равны, причем считается, что при любых* a∈A *и* b∈B*:* (NaN=b)*=1,* (a=NaN)*=1 и* (NaN=NaN)*=1.*

Определение 2. *Пусть* A *— скаляр, строка или гнездовой массив,* B *— гнездовой массив. Объект* A *обобщенно входит в* B*, если* A *обобщенно равно* B *или* A *обобщенно равно какому-либо элементу* B *на любом его уровне вложенности. Поиск обобщенных вхождений* A *в* B *будем называть обобщенным поиском.*

Определение 3. *Назовем позицией элемента* x *в гнездовом массиве* B *простую матрицу* po *из двух столбцов, в которой элементы последовательных строк, начиная от нулевой строки и далее, задают позиции* x *в* B *соответственно на первом, втором и т. д. уровнях вложенности. При этом, элементы нулевого и первого столбцов* po *— это номера соответственно строк и столбцов матриц.*

Ниже предлагаются решения 10 задач, связанных с поиском и замещением элементов в гнездовых массивах. В 4 первых задачах элементы NaN в массивах отсутствуют. В 6 последних задачах они могут присутствовать и используются как обобщенные элементы. В задачах, если специально не оговорено, A, Z — скаляры, строки или гнездовые массивы, B — гнездовой массив.

I. Задача IN1. Написать программу решения задачи XX.

II. Задача IN2. Написать программу решения задачи YY.

III. Задача IN3. Написать программу нахождения матрицы всех позиций вхождения A в B.

IV. Задача RIN1. Написать программу замещения всех вхождений A в B на объект Z на всех уровнях вложенности. Замещения проводить последовательно на первом, втором и т. д. уровнях.

V. Задача RIN2. Написать программу замещения в B элемента из позиции po на объект Z.

VI. Задача GEQU. Написать программу, выясняющую, являются ли гнездовые массивы A и B обобщенно равными?

VII. Задача GIN1. Написать программу, выясняющую, имеются ли обобщенные вхождения A в B?

VIII. Задача GIN2. Написать программу, подсчитывающую количество обобщенных вхождений A в B;

IX. Задача GIN3. Написать программу нахождения позиций всех обобщенных вхождений A в B.

X. Задача RGIN. Написать программу замещения всех обобщенных вхождений A в B на объект Z на всех уровнях вложенности. Замещения проводить последовательно на первом, втором и т. д. уровнях.

3. Решение задач

Все функции для решения сформулированных выше задач вместе с контрольным тестированием приведены ниже в разделах I-X фрагмента 1 многостраничного документа *PTC Mathcad*. Номера разделов на фрагменте совпадают с номерами задач и их решений. Большая часть созданных функций рекурсивна.

I. Решение задачи IN1. В дополнение к функции *in* из [1] созданы еще две рекурсивные функции $in11(A,B)$ и $in12(A,B)$ решения задачи *XX*. По ним возвращается 1 (истина), если A входит в B, и 0 (ложь), если A не входит в B.

II. Решение задачи IN2. В дополнение к функции *numx* из [1] созданы еще две рекурсивные функции $in21(A,B)$ и $in22(A,B)$ решения задачи *YY*. По ним возвращается количество вхождений A в B.

III. Решение задачи IN3. Для решения данной задачи предлагается программа из следующих двух функций: рекурсивной функции $lis(A,B,su)$ и головной функции $findpo(A,B)$.

Прежде всего, дадим пояснения к постановке задачи $IN3$. В задаче $IN2$ выдается общее количество вхождений A в B. При решении задачи $IN3$ формируется и возвращается более полная информация о подобных вхождениях. А именно, если A совпадает с B или A не входит в B, то выдается соответствующее сообщение. В противном случае, если A входит в B p раз ($p>0$), то возвращается гнездовая матрица $[ve1\ ve2\ \dots\ vep]$, каждый элемент которой есть позиция конкретного вхождения A в B. При этом, любая позиция — это простая матрица с двумя столбцами, в строке s (s=0,1,2,...) которой указываются координаты (a,b) вхождения A в B на уровне вложения $s+1$.

Рассмотрим пример. Пусть A=[7 8], $B=c$ (см. часть III фрагмента 1). Тогда по $IN3$ возвращается гнездовая матрица позиций:

$$\begin{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & \begin{bmatrix} 2 & 0 \end{bmatrix} \end{bmatrix}$$

В данном случае получаем:

1) $ve1=\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ и это означает, что спустившись в B на первом уровне в позицию (0, 0) и затем на втором уровне в позицию (0, 0) мы попадаем на элемент, совпадающий с A:

$(B_{0,\ 0})_{0,\ 0} = [7\ \ 8]$;

2) $ve2 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$ и это означает, что спустившись в B по уровням сначала в позицию $(0, 0)$, затем в позицию $(0, 1)$, и, наконец, в позицию $(0, 0)$, мы попадаем на второй элемент, который совпадает с A:

$\Big((B_{0,\ 0})_{0,\ 1}\Big)_{0,\ 0} = [7\ \ 8]$;

3) аналогично интерпретируются и строки матриц $ve3$ и $ve4$.

Несколько слов о функциях *lis*, *findpo* и их аргументах.

Функция *lis*(A,B,*su*). При обращении к *lis* аргумент *su*=[0 0 0 0 0], а возвращается вспомогательный список элементов B в виде простой матрицы *su* с 5 столбцами, строки которой имеют вид $[i\ \ j\ \ el\ \ k\ \ pri]$, где:

- *el* – очередной элемент B;
- $[i\ \ j]$ — позиция элемента *el* в B на текущем уровне вложенности;
- k — количество вхождений *el* в B;
- *pri* — признак, используемый в *findpo* при проведении вычислений. Он указывает, что элемент *el*=A уже участвовал (*pri*=1), или еще нет (*pri*=0) в построении какой-либо позиции вхождения A в B.

В возвращаемом по *lis* списке *su* начальная строка состоит из нулей.

Замечание 1. *Признак* pri *можно было бы вообще не вводить и использовать в* lis *аргумент* su *с 4 столбцами* [i j el k]. *Мы ввели этот признак лишь для того, чтобы в дальнейшем при переходе от поиска матрицы позиций вхождения* A *в* B *к поиску матрицы позиций обобщенных вхождений* A *в* B *пришлось затратить минимальные усилия.*

Функция *findpo*(A,B). По этой функции формируется однострочная гнездовая матрица результата O, каждый элемент которой является позицией A в B. Иными словами, *findpo* решает поставленную задачу. Поясним алгоритм, реализуемый функцией *findpo*:

1. при A=B выводится сообщение “Матрицы совпадают” и вычисления прекращаются. В противном случае подсчитывается общее количество p вхождений A в B. При p=0 выводится сообщение “Нет вхождений” и вычисления прекращаются, иначе выполняется пункт 2;

2. функцией *lis* создается матрица b со строками $[i\ \ j\ \ el\ \ k\ \ pri]$ (см. выше). Для конкретного гнездового массива B такая матрица приведена в части III фрагмента 1 сразу после функции *lis*. Далее в цикле *for* начинается формирование позиции *ot* очередного вхождения A в B;

3. в цикле *for* при конкретном значении его параметра t отыскивается очередная позиция *ot* вхождения A в B. Делается это так.

Сначала в цикле *while* спускаемся по 3 столбцу матрицы b до строки с номером n, в которой $b_{n,3} \neq 0$.

После этого начинает работать бесконечный цикл "*while* 1", выход из которого реализуется по *break*. В этом цикле осуществляется пробежка по b вниз, начиная со строки $m{=}n$. При этом отыскивается строка с номером m, в которой A входит в *el* и при этом $k{\neq}0$. Позиция $[i \;\; j]$ из этой строки добавляется к *ot* снизу и, кроме того, k в ней уменьшается на 1 (одно вхождение A в *el* использовано). Далее проверяется, не имеется ли совпадение A с *el* при $pri{=}0$? Если нет, то цикл "*while* 1" продолжает работу, если да, то очередная позиция *ot* сформирована, *pri* полагается равным 1 и цикл прерывается.

При выходе из бесконечного цикла найденная позиция *ot* добавляется в матрицу результата O в качестве ее очередного элемента. При завершении работы цикла *for* однострочная гнездовая матрица O оказывается полностью сформированной.

IV. Решение задачи RIN1. Результат замещений зависит от того, как их проводить. Приведем пример. Пусть в матрице [3 [3 0]] из двух элементов 3 и [3 0] производится замещение элементов [3 0] числом 3. Если замещения провести по уровням сверху вниз, то есть сначала на первом уровне, а затем на втором, то получим [3 0], если же их провести снизу вверх, то результат будет иным — [0]. Отсюда и необходимость указания в постановке задачи способа проведения замещений. Для решения задачи предлагается рекурсивная функция *rein*(A,Z,B).

V. Решение задачи RIN2. Замещение элемента из конкретной позиции *po* гнездового массива B на Z напрямую реализуется только на первом уровне вложенности. Для решения задачи в общем случае можно предложить рекурсивную функцию *repo*(*po*,Z,B).

VI. Решение задачи GEQU. Понятие обобщенного равенства двух гнездовых массивов A и B дано в определении 1. По техническим соображениям вместо создания нового оператора "обобщенного равенства" мы сформируем аналогичную ему логическую функцию *equG*(A,B), возвращающую 1 при обобщенном равенстве A и B и 0 — в противном случае. Решение задачи *GEQU* может быть реализовано следующими тремя функциями: рекурсивной функцией *obj*(B), по которой создается список всех элементов B на всех его уровнях вложенности; функцией *roco*(A,B), проверяющей, имеют ли массивы A и B на верхнем уровне одинаковое количество строк и одинаковое количество столбцов и головной функцией *equG*(A,B), решающей задачу *GEQU* обращениями к *obj* и *roco*. Заметим, что функция *equG*(A,B) создавалась для гнездовых массивов A и B, но она правильно работает как при скалярных, так и при строковых аргументах.

VII. Решение задачи GIN1. Решения данной и следующей задач на обобщенные вхождения A в B получаются заменой в соответствующих функциях ре-

шения задач на обычные вхождения оператора "="(A=B) функцией *equG*(A,B) из задачи *GEQU*. Предложенная задача решается рекурсивной функцией *gin*1.

VIII. Решение задачи GIN2. Предложенная задача решается рекурсивной функцией *gin*2.

IX. Решение задачи GIN3. Для решения *GIN*3 предлагается функция *findpoG*(A,B), построенная по аналогии с функцией *findpo*(A,B) задачи *IN*3. Сформирована она такими изменениями *findpo*:

- вместо функции *lis* создания списка элементов B с указанием количества вхождений в них A используется функция *list* создания списка элементов B с указанием количества обобщенных вхождений в них A;
- вместо оператора равенства "=" используется функция обобщенного равенства *equG* (не везде!);
- вместо функции *in*11 проверки на вхождение A в B применяется функция *gin*1 проверки на обобщенное вхождение A в B;
- вместо функции *in*21 вычисления количества вхождений A в B применяется функция *gin*2 для нахождения количества обобщенных вхождений A в B.

X. Решение задачи RGIN. Для решения задачи *RGIN* предлагается рекурсивная функция *reinG*(A,Z,B).

4. Заключение

Предложенные в статье рекурсивные функции решают основные задачи, связанные с простым и обобщенным поиском и замещением элементов в гнездовых массивах. Эти же функции можно использовать и в различных задачах поиска и замещения на графах, являющихся деревьями, поскольку последние можно представлять гнездовыми массивами.

Фрагмент 1 (1-9)

Задачи на простые и обобщенные вхождения и замещения элементов в гнездовых массивах на любых уровнях вложенности

Если специально не оговорено, то:
A - скаляр, строка, гнездовой массив;
B - гнездовой массив.

$$h := \begin{bmatrix} [6\ \text{“te”}] \\ [2 \\ [7\ 2\ 30] \\ [4\ 5\ [7\ 90]]] \\ [[7\ [5]] \\ 7 \\ [[7\ [7\ 6]]]] \end{bmatrix}$$

I. Задача *IN1*. Выяснить, входит ли *A* в *B*?

Каждая из рекурсивных функций *in*11 и *in*12 возвращает 1, если *A* входит в *B*, и 0, если *A* не входит в *B*. Для контрольного тестирования *in*11 используется гнездовой массив *h*, на верхнем уровне которого имеются 2 строки и 1 столбец. Аналогичные результаты получаются и при тестировании функции *in*12.

```
in11(A,B) := su ← (A = B)
             if IsArray(B)
                for b ∈ B
                   su ← max(su, A = b)
                   if su = 0
                      su ← if(IsArray(b), in11(A,b), 0)
```

```
in12(A,B) := su ← (A = B)
             if IsArray(B)
                for b ∈ B
                   su ← max(su, A = b)
                   if su = 0 ∧ IsArray(b)
                      su ← in12(A,b)
```

Тестирование функции in11:

$in11(7,7)=1$
$in11(7,[7])=1$
$in11([7],[7])=1$
$in11([7],7)=0$
$in11(5,h)=1$
$in11(9,h)=0$

$in11(\text{“text”},h)=0$
$in11(\text{“te”},h)=1$
$in11(\text{“dog”},h)=0$

$in11\left(\begin{bmatrix}[5]\\[7]\end{bmatrix},h\right)=0$

$in11\left(\begin{bmatrix}[7\ 2\ 30]\\[4\ 5\ [7\ 90]]\end{bmatrix},h\right)=1$

$in11\left(\begin{bmatrix}[8\ 2\ 30]\\[4\ 5\ [7\ 90]]\end{bmatrix},h\right)=0$

$in11([7\ 6],h)=1$

II. Задача *IN2*. Подсчитать количество вхождений *A* в *B*.

Каждая из рекурсивных функций *in*21 и *in*22 решает задачу *IN2*. При обращении к *in*22 третий аргумент должен быть равен 0. В нем и накапливаются вхождения *A* в *B*.

```
in21(A,B) := su ← (A = B)
             if IsArray(B)
                for b ∈ B
                   su ← su + (r ← (b = A))
                   if r = 0 ∧ IsArray(b)
                      su ← su + in21(A,b)
```

$in21(7,7)=1$
$in21([7],7)=0$
$in21(7,[7])=1$
$in21([7],[7])=1$
$in21([7\ 5],[7\ 5])=1$
$in21([7\ 5],[[7\ 5]\ [7\ 5]])=2$

Фрагмент 1 (2-9)

$$in21\left(\begin{bmatrix}5\\[7\ 6]\end{bmatrix},\begin{bmatrix}2\\7\\[[7\ 6]]\end{bmatrix}\right)=2 \qquad in21\left([7\ 8],\begin{bmatrix}[7\ 8]\\7\\[[7\ 8]\ [7\ 9\ [7\ 8]]]\end{bmatrix}\right)=4$$

```
in22(A,B,su) := su ← su + (A = B)
                if IsArray(B)
                    for b ∈ B
                        if IsArray(b)
                            su ← in22(A,b,su)
                        else
                            su ← su + (A = b)
```

$in22(7,7,0)=1$
$in22([7],7,0)=0$
$in22(7,[7],0)=1$
$in22([7],[7],0)=1$
$in22([7\ 5],[7\ 5],0)=1$
$in22([7\ 5],[[7\ 5]\ [7\ 5]],0)=2$

III. Задача *IN3*. Сформировать матрицу всех позиций вхождения *A* в *B*.

1) Для решения *IN3* предлагается программа из рекурсивной функции *lis* и головной функции *findpo*, которая обращаясь к *lis*, формирует однострочную гнездовую матрицу *O* с элементами, являющимися позициями *A* в *B*.

```
lis(A,B,su) := for i ∈ 0..rows(B)-1
                   for j ∈ 0..cols(B)-1
                       su ← stack(su,[i j B[i,j] in21(A,B[i,j]) 0])
                       if IsArray(B[i,j])
                           ot ← lis(A,B[i,j],[0 0 0 0 0])
                           su ← stack(su,submatrix(ot,1,rows(ot)-1,0,4))
```

$$d:=\begin{bmatrix}[[7\ 8]\ [7\ 8]]\\[1\ [7\ 8]]\\[7\ 8]\end{bmatrix}$$

$b:=lis([7\ 8],d,[0\ 0\ 0\ 0\ 0])$

$$b=\begin{bmatrix}0&0&0&0&0\\0&0&[[7\ 8]\ [7\ 8]]&2&0\\0&0&[7\ 8]&1&0\\0&0&7&0&0\\0&1&8&0&0\\0&1&[7\ 8]&1&0\\0&0&7&0&0\\0&1&8&0&0\\1&0&[1\ [7\ 8]]&1&0\\0&0&1&0&0\\0&1&[7\ 8]&1&0\\0&0&7&0&0\\0&1&8&0&0\\2&0&[7\ 8]&1&0\\0&0&7&0&0\\0&1&8&0&0\end{bmatrix}$$

Фрагмент 1 (3-9)

```
findpo(A,B) := if A = B
                 return ["Матрицы совпадают"]
               if (p ← in21(A,B)) = 0
                 return ["Нет вхождений"]
               [b n] ← [lis(A,B,[0 0 0 0 0]) 0]
               for t ∈ 0..p−1
                 while b_{n,3} = 0
                   n ← n+1
                 [ot s m] ← [0 0 n]
                 while 1
                   if in11(A, b_{m,2}) = 1 ∧ b_{m,3} ≠ 0
                     [ot_{s,0} ot_{s,1}] ← [b_{m,0} b_{m,1}]
                     [s b_{m,3}] ← [s+1 b_{m,3}−1]
                     if A = b_{m,2} ∧ b_{m,4} = 0
                       b_{m,4} ← 1
                       break
                   m ← m+1
                 O_{0,t} ← ot
               O
```

$findpo(d,d) = [\text{"Матрицы совпадают"}]$ $\quad findpo(99,d) = [\text{"Нет вхождений"}]$

$a := [7]$ $\quad findpo(7,a) = [[0\ 0]]$

$$b := \begin{bmatrix} [7\ 8] \\ [7\ 9\ [7\ 8]] \\ [7\ 8] \end{bmatrix} \qquad findpo([7\ 8],b) = \begin{bmatrix} [0\ 0] & \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & [2\ 0] \end{bmatrix}$$

$$c := \begin{bmatrix} [[7\ 8]\ [[7\ 8]]] \\ [7\ 9\ [7\ 8]] \\ [7\ 8] \end{bmatrix} \qquad findpo([7\ 8],c) = \begin{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} & [2\ 0] \end{bmatrix}$$

2) Проверка правильности найденных позиций.

$a_{0,0} = 7$ $\quad b_{0,0} = [7\ 8]$ $\quad (b_{1,0})_{0,2} = [7\ 8]$ $\quad b_{2,0} = [7\ 8]$

$(c_{0,0})_{0,0} = [7\ 8]$ $\quad ((c_{0,0})_{0,1})_{0,0} = [7\ 8]$ $\quad (c_{1,0})_{0,2} = [7\ 8]$ $\quad c_{2,0} = [7\ 8]$

3) Поскольку вводить многоступенчатые пары индексов трудоемко, особенно если их много, то есть смысл предложить функцию, определяющую по

Фрагмент 1 (4-9)

позиции элемента сам элемент. Это может быть представленная ниже функция elpo. Рядом с elpo показаны примеры вычислений по этой функции.

```
elpo(B, po) :=
    for k ∈ 0 .. rows(po) − 1
        [i  j] ← [po_{k,0}  po_{k,1}]
        B ← B_{i,j}
```

$$elpo\left(c, \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}\right) = [7\ \ 8] \qquad 1$$

$$elpo\left(c, \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\right) = [7\ \ 8] \qquad elpo(c, [2\ \ 0]) = [7\ \ 8] \qquad elpo\left(c, \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}\right) = [7\ \ 8]$$

IV. Задача *RIN*1. Заместить все вхождения *A* в *B* на *Z* на любом уровне вложенности. Замещения проводить последовательно на 1, 2, ... уровнях.

Для решения задачи *RIN*1 предлагается следующая рекурсивная функция *rein*:

```
rein(A, Z, B) :=
    if A = B
        return Z
    for i ∈ 0 .. rows(B) − 1
        for j ∈ 0 .. cols(B) − 1
            if B_{i,j} = A
                B_{i,j} ← Z
            else if IsArray(B_{i,j})
                B_{i,j} ← rein(A, Z, B_{i,j})
    B
```

$$xx := \begin{bmatrix} [[[\ [7\ \ 8]\ ]]\ \ ] \\ [[\ \ 1\ \ ]]\ [7\ \ 8] \\ [[[1\ \ [7\ \ 8]]]] \\ [\ [1\ \ [7\ \ 8]]\ \ [7\ \ 8] \\ [\ \ [7\ \ 8]\ \ \ [7\ \ 8] \end{bmatrix}$$

$$rein([7\ \ 8], 77, xx) = \begin{bmatrix} [[[\ \ 77\ \ ]]\ \ ] \\ [[\ \ 1\ \ ]]\ 77 \\ [[[1\ \ 77]]] \\ [\ [1\ \ 77]\ \ 77 \\ [\ \ 77\ \ \ 77 \end{bmatrix} \qquad rein\left(8, \begin{bmatrix} 0 \\ 0 \end{bmatrix}, xx\right) = \begin{bmatrix} [[[\ \ [7\ [0]]\ \ ]]\ \ ] \\ [[\ \ [7\ [0]]\ \ ]]\ [7\ [0]] \\ [[[1\ \ [7\ [0]]]]] \\ [\ [1\ \ [7\ [0]]]\ \ [7\ [0]] \\ [\ \ [7\ [0]]\ \ \ [7\ [0]] \end{bmatrix}$$

$$rein(7, 8, 7) = 8 \qquad rein(7, 8, [7]) = [8] \qquad rein\left(\begin{bmatrix} 3 \end{bmatrix}, \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \begin{bmatrix} 3 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

V. Задача *RIN*2. Заместить элемент из позиции *po* в *B* на *Z*.

Для решения задачи предлагается следующая рекурсивная функция *repo*:

Фрагмент 1 (5-9)

```
repo(po, Z, B) := k ← rows(po) − 1
                  [i  j] ← [po_{0,0}  po_{0,1}]
                  if k = 0
                      B_{i,j} ← Z
                  else
                      po ← submatrix(po, 1, k, 0, 1)
                      B_{i,j} ← repo(po, Z, B_{i,j})
                  B
```

```
c :=  [[[7 8] [[7 8]]]
       [[7 9 [7 8]]   ]
       [   [7 8]      ]]

x := repo([[0 0]  [1 2]  ], c)
          [[0 1]  [3 4]  ]
          [[0 0]         ]

x =  [[      [[1 2]]]
      [[7 8] [[3 4]]]
      [[     [ 5  ]]]
      [  [7 9 [7 8]]  ]
      [     [7 8]     ]]
```

V!. Задача *GEQU*. Выяснить, являются ли *A* и *B* обобщенно равными?

Задача решается программой из трех функций:
1) рекурсивная функция *obj*(*B*) создает список всех элементов *B* на всех его уровнях вложенности;
2) функция *roco*(*A*,*B*) проверяет, равны ли в *A* и *B* количество строк и количество столбцов на верхнем уровне вложенности;\
3) головная функция *equG*(*A*,*B*) решает задачу *GEQU*. Если *A* обобщенно равно *B*, то возвращается 1, иначе - 0.

```
obj(B) := ve ← ∞
          for b ∈ B
              ve ← if(ve = ∞, [b], stack(ve, [b]))
              if IsArray(b)
                  ve ← stack(ve, obj(b))
```

Аргументами функций *obj* и *equG* могут быть скаляры, строки, гнездовые массивы

```
roco(A, B) := (rows(A) = rows(B)) ∧ (cols(A) = cols(B))
```

```
equG(A, B) := pro ← 1
              if ¬(IsNaN(A) ∨ IsNaN(B))
                  [sp1  sp2] ← [obj(A)  obj(B)]
                  [s  k  sf  kf  pro] ← [0  0  last(sp1)  last(sp2)  roco(A, B)]
                  while (s ≤ sf) ∧ (k ≤ kf)
                      if pro = 1
                          if IsArray(sp1_s) ∧ IsArray(sp2_k)
                              pro ← roco(sp1_s, sp2_k)
                          else if (IsArray(sp1_s) ∧ (¬IsArray(sp2_k)))
                              pro ← 0
```

Фрагмент 1 (6-9)

```
if IsNaN(sp2_k)
    [pro  s] ← [1  s + rows(obj(sp1_s))]
else if (¬IsArray(sp1_s)) ∧ (IsArray(sp2_k))
    pro ← 0
    if IsNaN(sp1_s)
        [pro  k] ← [1  k + rows(obj(sp2_k))]
else
    pro ← IsNaN(sp1_s) ∨ IsNaN(sp2_k) ∨ (sp1_s = sp2_k)
    if (s > sf) ∨ (k > kf)
        pro ← 0
[s  k] ← [s + 1  k + 1]
pro
```

Контрольные примеры.

1) $g := \begin{bmatrix} 4 & \begin{bmatrix} 5 & \begin{bmatrix} 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$ $t := \begin{bmatrix} 4 & \begin{bmatrix} 5 & NaN \end{bmatrix} \end{bmatrix}$ $obj(t) = \begin{bmatrix} 4 \\ \begin{bmatrix} 5 & NaN \end{bmatrix} \\ 5 \\ \begin{bmatrix} 5 & NaN \end{bmatrix} \\ 5 \\ NaN \end{bmatrix}$ $obj(g) = \begin{bmatrix} 4 \\ \begin{bmatrix} 5 \\ \begin{bmatrix} 5 & \begin{bmatrix} 1 \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ 5 \\ \begin{bmatrix} 5 & \begin{bmatrix} 1 \end{bmatrix} \end{bmatrix} \\ 5 \\ \begin{bmatrix} 1 \end{bmatrix} \\ \begin{bmatrix} 2 \end{bmatrix} \\ 1 \\ 2 \end{bmatrix}$

$equG(g,t) = 1$

2) $x := \begin{bmatrix} \begin{bmatrix} 1 \end{bmatrix} & 4 \\ \begin{bmatrix} 2 \end{bmatrix} & \\ 3 & 5 \end{bmatrix}$ $y := \begin{bmatrix} 1 & 4 \\ \begin{bmatrix} 2 \end{bmatrix} & 5 \\ \begin{bmatrix} 3 \end{bmatrix} & \end{bmatrix}$ $z := \begin{bmatrix} 1 & 4 \\ NaN & 5 \end{bmatrix}$ $obj(z) = \begin{bmatrix} 1 \\ NaN \\ 4 \\ 5 \end{bmatrix}$ $obj(y) = \begin{bmatrix} 1 \\ \begin{bmatrix} 2 \end{bmatrix} \\ \begin{bmatrix} 3 \end{bmatrix} \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$

$equG(x,y) = 0$ $equG(x,z) = 0$ $equG(z,x) = 0$
$equG(z,y) = 1$ $equG(y,z) = 1$

3) $equG(7,7) = 1$ $equG([7],[7]) = 1$ $equG([7],7) = 0$ $equG(7,[7]) = 0$
$equG(NaN,7) = 1$ $equG(7,NaN) = 1$ $equG([7],NaN) = 1$
$equG(NaN,[7]) = 1$ $equG(NaN,NaN) = 1$ $equG(\text{“te”},\text{“te”}) = 1$

3) $B := \begin{bmatrix} 2 & \begin{bmatrix} 2 & 9 \\ \begin{bmatrix} 7 & NaN \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$

$equG\left(B, \begin{bmatrix} NaN & \begin{bmatrix} 2 & 9 \\ \begin{bmatrix} 7 & 5 \end{bmatrix} & NaN \end{bmatrix} \end{bmatrix}\right) = 1$

$equG\left(B, \begin{bmatrix} NaN & \begin{bmatrix} 2 & 9 \\ \begin{bmatrix} 6 & 5 \end{bmatrix} & NaN \end{bmatrix} \end{bmatrix}\right) = 0$

Фрагмент 1 (7-9)

VII. Задача *GIN*1. Выяснить имеются ли обобщенные вхождения *A* в *B*?

Решения всех последующих задач на обобщенные вхождения и замещения получаются заменой в соответствующих функциях решения задач на обычные вхождения и замещения оператора "=" (*A*=*B*) функцией *equG*(*A*,*B*).

Для решения задачи *GIN*1 предлагается рекурсивная функция *gin*1.

```
gin1(A,B) := su ← equG(A,B)
             if IsArray(B)
                for b ∈ B
                   su ← max(su, equG(A,b))
                   if su = 0 ∧ IsArray(b)
                      su ← gin1(A,b)
```

Контрольные вычисления для *gin*1.

$$y := \begin{bmatrix} [6\ \text{“text”}] & \\ \begin{bmatrix} 2 & \begin{bmatrix} 7 & 2 & 30 \end{bmatrix} \\ & \begin{bmatrix} 4 & 5 & [7\ 90] \end{bmatrix} \end{bmatrix} & \\ \begin{bmatrix} 7 & \begin{bmatrix} 5 \end{bmatrix} \end{bmatrix} & 7 \end{bmatrix}$$

$gin1(7,7)=1$ $gin1(7,[7])=1$ $gin1([7],[7])=1$ $gin1(11,y)=0$ $gin1(\text{“text”},y)=1$ $gin1([NaN\ \ 5],y)=0$

$gin1(7,NaN)=1$ $gin1(NaN,7)=1$ $gin1(NaN,NaN)=1$ $gin1(111,y)=0$ $gin1(\text{“dog”},y)=0$ $gin1([7\ \ NaN],y)=1$

$gin1\left(\begin{bmatrix}5\\1\end{bmatrix},y\right)=0$ $gin1\left(\begin{bmatrix}NaN\\1\end{bmatrix},y\right)=0$

$gin1\left(\begin{bmatrix}7&2&30\\4&5&[7\ 90]\end{bmatrix},y\right)=1$ $gin1\left(\begin{bmatrix}7&2&30\\4&5&NaN\end{bmatrix},y\right)=1$

$gin1\left(\begin{bmatrix}7&2&NaN\\4&5&[7\ 90]\end{bmatrix},y\right)=1$ $gin1\left(\begin{bmatrix}8&2&30\\4&5&[7\ 90]\end{bmatrix},y\right)=0$

VIII. Задача *GIN*2. Подсчитать количество обобщенных вхождений *A* в *B*.

Рекурсивная функция *gin*2 решает задачу *GIN*2, обращаясь к функции *equG*. При обращении к *gin*2 третий аргумент должен быть равен 0. В нем и накапливаются обобщенные вхождения *A* в *B*.

```
gin2(A,B,su) := su ← su + equG(A,B)
                if IsArray(B)
                   for b ∈ B
                      if IsArray(b)
                         su ← gin2(A,b,su)
                      else
                         su ← su + equG(A,b)
```

$gin2(7,7,0)=1$ $gin2([7],7,0)=0$ $gin2(7,[7],0)=1$ $gin2([7],[7],0)=1$ $gin2(7,NaN,0)=1$ $gin2(NaN,7,0)=1$ $gin2(NaN,NaN,0)=1$

$gin2([7\ \ NaN],[7\ \ 5],0)=1$ $gin2([7\ \ NaN],[7\ \ [7\ \ 11]],0)=2$

$$gin2\left(\begin{bmatrix}7&NaN\end{bmatrix},\begin{bmatrix}7&\begin{bmatrix}\begin{bmatrix}5\end{bmatrix}&\begin{bmatrix}\begin{bmatrix}6\end{bmatrix}&\begin{bmatrix}7&9&[7\ 11]\end{bmatrix}\end{bmatrix}\\7&\begin{bmatrix}\begin{bmatrix}\begin{bmatrix}7\ 8\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix},0\right)=4$$

Фрагмент 1 (8-9)

IX. Задача *GIN3*. Найти позиции всех обобщенных вхождений *A* в *B*.

Функция *findpoG* решает задачу *GIN3*, обращаясь к функции *list*.

```
list(A,B,su) := | for i ∈ 0..rows(B)−1
                |   for j ∈ 0..cols(B)−1
                |     su ← stack(su, [i  j  B_{i,j}  gin2(A, B_{i,j}, 0)  0])
                |     if IsArray(B_{i,j})
                |       ot ← list(A, B_{i,j}, [0 0 0 0 0])
                |       su ← stack(su, submatrix(ot, 1, rows(ot)−1, 0, 4))
```

```
B := [ 7   [ 7   [ [7 9 [7 11]]   6   [7 8] ]   5 ]   [7 5] ]

b := list([7  NaN], B, [0 0 0 0 0])
```

```
findpoG(A,B) := | if A = B
                |   return ["Матрицы совпадают"]
                | if (p ← gin2(A,B,0)) = 0
                |   return ["Нет вхождений"]
                | [b  n] ← [list(A,B,[0 0 0 0 0])  0]
                | for t ∈ 0..p−1
                |   while b_{n,3} = 0
                |     n ← n+1
                |   [ot  s  m] ← [0  0  n]
                |   while 1
                |     if gin1(A, b_{m,2}) = 1 ∧ b_{m,3} ≠ 0
                |       [ot_{s,0}  ot_{s,1}] ← [b_{m,0}  b_{m,1}]
                |       [s  b_{m,3}] ← [s+1  b_{m,3}−1]
                |       if equG(A, b_{m,2}) = 1 ∧ b_{m,4} = 0
                |         b_{m,4} ← 1
                |         break
                |     m ← m+1
                |   O_{0,t} ← ot
                | O
```

b =

0	0	0	0	0
0	0	7	0	0
0	1	[7 [7 [[7 9 [7 11]] 6 [7 8]]] 5]	3	0
0	0	5	0	0
1	0	[7 [[7 9 [7 11]] 6 [7 8]]]	3	0
0	0	7	0	0
0	1	[[7 9 [7 11]] 6 [7 8]]	2	0
0	0	6	0	0
1	0	[7 9 [7 11]]	1	0
0	0	7	0	0
0	1	9	0	0
0	2	[7 11]	1	0
0	0	7	0	0
0	1	11	0	0
2	0	[7 8]	1	0
0	0	7	0	0
0	1	8	0	0
0	2	[7 5]	1	0
0	0	7	0	0
0	1	5	0	0

findpoG(B,B) = ["Матрицы совпадают"]

findpoG([77 99],B) = ["Нет вхождений"]

Фрагмент 1 (9-9)

$$t := findpoG([7\ \ NaN], B) = \begin{bmatrix} [0\ 1] & [0\ 1] & [0\ 1] & \\ [1\ 0] & [1\ 0] & [1\ 0] & [0\ 2] \\ & [0\ 1] & [2\ 0] & \\ & [0\ 2] & & \end{bmatrix} \quad (B_{0,1})_{1,0} = \begin{bmatrix} [\ 6\ ] \\ [7\ [7\ 9\ [7\ 11]]] \\ [\ [7\ 8]\ ] \end{bmatrix}$$

$$((((B_{0,1})_{1,0})_{0,1})_{1,0})_{0,2} = [7\ 11] \qquad ((((B_{0,1})_{1,0})_{0,1})_{2,0} = [7\ 8] \qquad B_{0,2} = [7\ 5]$$

$$elpo(B, t_{0,1}) = [7\ 11] \qquad elpo(B, t_{0,2}) = [7\ 8] \qquad elpo(B, t_{0,3}) = [7\ 5]$$

X. Задача *RGIN*. Заместить все обобщенные вхождения *A* в *B* на *Z* на любом уровне вложенности. Замещения проводить последовательно на 1, 2, ... уровнях.

Для решения задачи *RGIN* предлагается следующая рекурсивная функция *reinG*:

```
reinG(A, Z, B) :=
    if equG(A, B)
        return Z
    for i ∈ 0..rows(B) − 1
        for j ∈ 0..cols(B) − 1
            if equG(A, B_{i,j})
                B_{i,j} ← Z
            else if IsArray(B_{i,j})
                B_{i,j} ← reinG(A, Z, B_{i,j})
    B
```

$$h := \begin{bmatrix} [[1]] & [1] & [4\ [\ 5\ ]] \\ [[2]] & [1] & [6\ [\ 1\ ]] \\ & & [NaN] \\ \text{“te”} & 6 & [[NaN]] \\ [NaN] & & [\ 2\ ] \\ & & \text{“te”} \end{bmatrix}$$

$$w1 := reinG([[1]\ [2]], [3\ \ 4], h) \qquad w2 := reinG([1\ \ NaN], [3\ \ 4], h)$$

$$w3 := reinG([4\ \ NaN], \text{“яблоко”}, h) \qquad w4 := reinG([NaN\ \ \text{“te”}], [88\ \ 99], h)$$

$$w1 = \begin{bmatrix} [[3\ 4]] & [1] & [4\ [\ 5\ ]] \\ [6\ [3\ 4]] & & \\ \text{“te”} & 6 & [[3\ 4]] \\ [3\ 4] & & \text{“te”} \end{bmatrix} \qquad w2 = \begin{bmatrix} [[3\ 4]] & [3\ 4] & [4\ [\ 5\ ]] \\ [6\ [3\ 4]] & & \\ \text{“te”} & 6 & [[3\ 4]] \\ [3\ 4] & & \text{“te”} \end{bmatrix}$$

$$w3 = \begin{bmatrix} [[1]] & [1] & \text{“яблоко”} \\ [[2]] & & \\ \text{“te”} & 6 & [[\text{“яблоко”}]] \\ \text{“яблоко”} & & [\ 2\ ] \\ & & \text{“te”} \end{bmatrix} \qquad w4 = \begin{bmatrix} [[1]] & [1] & [4\ [\ 5\ ]] \\ [[2]] & [1] & [6\ [88\ 99]] \\ [88\ 99] & 6 & [88\ 99] \end{bmatrix}$$

СПИСОК ЦИТИРОВАННОЙ ЛИТЕРАТУРЫ

1. Есаян А. Р. Гнездовые массивы и рекурсия / Н. М. Добровольский, А. Р. Есаян. Материалы XXIII межд. конф. “Алгебра, теория чисел и дискретная геометрия. Современные проблемы и приложения” Тула, 25-30 мая 2015, с. 319-321

2. Есаян А. Р. Векторизация и гнездовые массивы / А. Р. Есаян, А. В. Якушин. Материалы XXIII межд. конф. “Алгебра, теория чисел и дискретная геометрия. Современные проблемы и приложения” Тула, 25-30 мая 2015, с. 328-330

3. Есаян А. Р., Добровольский Н. М. Гнездовые массивы и рекурсия // Чебышевский сборник. 2015. Т. 16, вып. 3(55). С. 479–495.

4. Есаян А. Р., Якушин А. В. Векторизация и гнездовые массивы // Чебышевский сборник. 2015. Т. 16, вып. 3(55). С. 496–509.

5. Есаян А. Р. Обучение алгоритмизации на основе рекурсии. Тула: Изд. ТГПУ, 2001, с. 215

6. Brent Maxfield, P. E. Essential PTC Mathcad Prime 3.0. A Guide for New and Current Users, New York, Academic Press is an imprint of Elsevier, Nov. 11, 2013, p. 563

7. Hans Wessenlingh and Hans de Waard. Calculate & Communicate with Mathcad Prime 3.0, Delft Academic Press, The Netherlands, First edition 2014

REFERENCES

1. Esayan, A. R. & Dobrovolsky, N. M. 2015, "Nested arrays and recursion", *Proc. XXIII Int. Conf. “Algebra, number theory and discrete geometry. Modern problems and applications” Tula, May 25–30 2015*, pp. 319–321. (Russian)

2. Esayan, A. R. & Yakushin, A. V. 2015, "Vectorization and nested arrays", *Proc. XXIII Int. Conf. “Algebra, number theory and discrete geometry. Modern problems and applications” Tula, May 25–30 2015*, pp. 328–330. (Russian)

3. Esayan, A. R. & Dobrovolsky, N. M. 2015, "Nested arrays and recursion", *Chebyshevskii Sb.*, vol. 16, no. 3(55), pp. 479–495.

4. Esayan, A. R. & Yakushin, A. V. 2015,"Vectorization and nested arrays", *Chebyshevskii Sb.*, vol. 16, no. 3(55), pp. 496–509.

5. Esayan, A. R. 2001, "*Obuchenie algoritmizacii na osnove rekursii.*", [Teaching algorithmization based on recursion], TGPU, Tula, 215 pp. (Russian)

6. Brent Maxfield, P. E. 2013, *Essential PTC Mathcad Prime 3.0. A Guide for New and Cur-rent Users*, Academic Press, New York.

7. Wessenlingh, H. & de Waard, H. 2014, *Calculate & Communicate with Mathcad Prime 3.0*, Delft Academic Press, The Netherlands.

Московский городской педагогический университет.
Тульский государственный педагогический университет им. Л. Н. Толстого
Получено 09.07.2015