DOI: 10.32517/2221-1993-2022-21-6-55-67

М. С. Долинский
Гомельский государственный университет имени Франциска Скорины, г. Гомель, Беларусь

УСКОРЕНИЕ РЕКУРСИВНЫХ РЕШЕНИЙ ПРИ ПОМОЩИ МЕМОИЗАЦИИ*

Аннотация

В статье на примере решения двух задач проиллюстрирована методика изучения темы «Ускорение рекурсивных решений при помощи мемоизации» при подготовке школьников к олимпиадам по информатике. Изучение основано на последовательном решении усложняющихся задач. Для каждой задачи приводятся следующие материалы: условие задачи, идея решения с предложением придумать самостоятельно реализацию, решение на языке программирования Pascal. Серьезной технической основой является разработанная под управлением автора инструментальная система дистанционного обучения (http://dl.gsu.by), которая позволяет: предложить ученику условие задачи; отправить решение на проверку; получить от системы вердикт — правильное или неправильное решение; для неправильных решений указывается номер теста, на котором решение не прошло. Ученик может взять тест (входные и выходные данные), на котором не прошло его решение, разобраться, в чем ошибка в его программе, исправить и послать решение повторно. Кроме того, для каждой задачи есть ссылка по ней на тему в форуме, где можно задать вопрос по решению этой задачи и/или почитать ответ, если вопросы уже задавались ранее.

Ключевые слова: рекурсия, мемоизация, олимпиады по информатике, инструментальная система дистанционного обучения.

1. Введение

Подготовка школьников к олимпиадам по информатике и программированию требует от учащихся значительного объема теоретических знаний и практических навыков решения задач. Поэтому в мире компьютерной книжной и журнальной литературы регулярно появляются книги и статьи как по общей теории алгоритмов (см., например, [5, 10, 11, 13, 14, 16, 18–20, 23]), так и на отдельные актуальные темы: динамическое программирование [12, 27, 28], графы [15, 22, 23, 25, 26, 29], хеши [24 , 30], рекурсия [1, 2–4, 17]. Автор вносит свою лепту в разработку темы «Рекурсия» [5–10] специальным подходом, связанным с ранним началом обучения. Как следствие, уже в пятом-шестом классах появляются ученики, которым приходится объяснять такие темы, как «Рекурсия». Поэтому приходится модифицировать изложение в сторону более простого и наглядного объяснения и более медленного продвижения по учебному материалу, явно выделяя и обозначая все этапы этого продвижения. Введение в решение задач с помощью рекурсии изложено в работе [5]. Решение задач с помощью рекурсивной генерации таких комбинаторных объектов, как: множество всех подмножеств, сочетания, перестановки, перестановки с повторениями, размещения, описано в работе [6].

* Материалы к статье можно скачать на сайте ИНФО: http://infojournal.ru/journals/school/school_06-2022/

Контактная информация

Долинский Михаил Семенович, канд. тех. наук, доцент, доцент кафедры математических проблем управления и информатики, Гомельский государственный университет имени Франциска Скорины, г. Гомель, Беларусь; *адрес:* 246000, Республика Беларусь, г. Гомель, ул. Советская, д. 104; *e-mail:* dolinsky@gsu.by

M. S. Dolinsky,
Francisk Skorina Gomel State University, Gomel, Belarus

SPEEDING UP RECURSIVE SOLUTIONS USING MEMOIZATION

Abstract

In the article, using the example of solving two problems, the methodology for studying the theme "Speeding up recursive solutions using memoization" is illustrated in preparing schoolchildren for Olympiads in informatics. The study is based on the sequential solution of increasingly complex problems. For each problem the following materials are given: the formulation of the problem, the idea of a solution with a proposal to come up with an implementation on their own, the solution in the Pascal programming language. Distance learning system (http://dl.gsu.by) is the effective technical base for teaching. The system allows to offer for a student a formulation of the problem; to submit the solution for review; to get a verdict from the system — a correct or incorrect solution; for incorrect solution, the number of the test on which the solution did not pass is indicated. A student can take a test (input and output data), on which his solution did not pass, figure out what the error is in his program, correct and send the solution again. In addition, for each problem there is a link on it to the topic in the forum at site, where you can ask a question onsolving this problem and / or read the answer if the questions have already been asked before.

Keywords: recursion, memoization, programming training, Olympiads in informatics, distance learning tools.

В работе [10] описано решение рекурсивных задач «по определению». В работе [8] описано решение задач рекурсивной генерацией чисел. Решение игровых задач с помощью рекурсии описывается в работе [9]. Работа [7] предлагает материал по решению задач с помощью основанного на рекурсии метода «разделяй и властвуй». В данной статье описывается использование мемоизации для ускорения рекурсивных решений.

Проверка решений осуществляется автоматически на сайте DL.GSU.BY. Вдумчивым читателям предлагается после чтения условия задачи/подзадачи попытаться решить ее самостоятельно. Все задачи на указанном сайте находятся в учебном курсе «Олимпиады по информатике».

2. Понятие мемоизации

Мемоизация (*англ.* memoization от memory — память и optimization — оптимизация) — в программировании сохранение результатов выполнения функций для предотвращения повторных вычислений. Это один из способов оптимизации, применяемый для увеличения скорости выполнения компьютерных программ. Перед вызовом функции проверяется, вызывалась ли функция ранее; если не вызывалась, то функция вызывается и результат ее выполнения сохраняется; если вызывалась, то используется сохраненный результат.

3. Задача «Клеточки» (Гомельская городская олимпиада для I—IX классов, 2012 год)

Задано клетчатое поле размером $1 \times N$ клеток. Необходимо определить количество способов раскрасить это поле в два цвета так, чтобы никакие из K рядом находящихся клеток не были закрашены одним цветом (каждая клетка может быть окрашена только в один из двух цветов).

Формат ввода.
Два целых числа — N и K.

Ограничения.
$2 \le N, K \le 50$.

Формат вывода.
Одно число — количество способов раскраски.

Пример ввода	*Пример вывода*
5 3	16

Идея решения задачи.
Идея решения — рекурсия с мемоизацией.

Запоминать (мемоизировать) будем количество способов раскрасить поле из *Current* клеточек, заполнив одним цветом *Fill* клеточек (в массиве $F[1..50,0..50]$).

Если элемент массива уже вычислен, рекурсия не вызывается, а ответ берется из массива F. Иначе рекурсия вызывается, а вычисленный рекурсией ответ сохраняется в массиве F.

Рекурсивные вычисления таковы:

$$Rec(Current, Fill) = \begin{cases} 0, \text{ если } Fill \ge K \\ 1, \text{ если } Current > N \\ Rec(Current + 1, Fill + 1) + \\ + Rec(Current + 1, 1) \end{cases}$$

То есть значение вычисляется как количество способов, если мы раскрасим тем же цветом, плюс количество способов, если начнем красить другим цветом.

Полный текст решения приведен в листинге 1.

```
var
  F      : array [1..50,0..50] of int64;
  n,k,i,j: longint;

function Rec(Current,Fill: longint): int64;
var
  Res: int64;
begin
  if Fill>=k            then begin Rec:=0; exit; end;
  if Current>N          then begin Rec:=1; exit; end;
  if F[Current,Fill]<>0 then begin Rec:=f[Current,Fill]; exit; end;
  Res:=Rec(Current+1,Fill+1) + Rec(Current+1,1);
  F[Current,Fill]:=Res;
  Rec            :=Res;
end;

begin
  readln(n,k);
  for i:=1 to n do
    for j:=0 to k do f[i,j]:=0;
  writeln(Rec(1,0));
end.
```

Листинг 1

4. Задача «Трапеции» (Хорватская национальная олимпиада, 2017 год)

Мы можем создать гексагональный пазл размера n, поделив правильный шестиугольник на равные треугольники, проведя на одинаковых расстояниях $2n - 1$ параллельных линий между каждой из трех пар противоположных сторон шестиугольника. Некоторые из треугольников этого пазла заштрихованы и должны быть накрыты кусками пазла. Каждый кусок — это трапеция, которая состоит из трех равносторонних треугольников, приставленных сторона к стороне. Эти куски бывают шести различных цветов, обозначенных числами от 1 до 6. В нашем распоряжении есть неограниченное количество кусков каждого цвета.

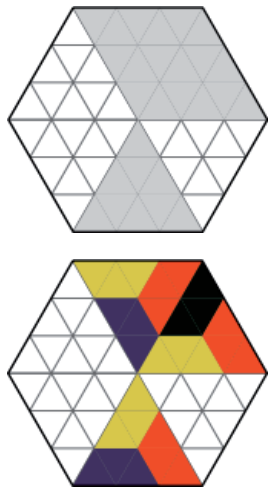
Цель пазла — положить куски на шестиугольник так, чтобы выполнялось следующее:

1) каждый кусок покрывает ровно три заштрихованных треугольника;
2) каждый треугольник накрывается ровно одним куском;
3) два куска одного цвета не касаются по стороне (но могут касаться по углу).

Определите, существует ли решение, если существует, выведите одно.

Время на прохождение каждого теста — одна секунда.

Ниже для примера приведены пазл размера $n = 3$ и решение задачи:



Формат ввода.

Первая строка ввода содержит положительное целое число n — размер пазла.

Следующие $2n$ строк описывают строки пазла сверху вниз. Каждая из этих строк содержит строку символов, которая описывает строку пазла слева направо. Цифра 0 означает заштрихованный треугольник, а символ точки «.» — незаштрихованный треугольник. Вы можете полагать, что хотя бы один треугольник заштрихован.

Формат вывода.

Если пазл невозможно решить, выведите в первой строке слово «nemoguce» («невозможно» по-хорватски). Заштрихованные треугольники должны быть обозначены цифрами от 1 до 6 вместо цифры 0. Эти цифры обозначают цвет соответствующего треугольника.

Пример ввода	3 .000000 ...000000 .....000000 .....0..... ...000... .00000.	1 .0. 0.0	2 0000. 0000000 ..00.0. .0000
Пример вывода	.111224 ...332442 .....311122 .....1..... ...112... .33322.	nemoguce	1222. 1133111 ..31.2. .1122

Решение.

Вернемся к рисунку из условия задачи.

Здесь $n = 3$, т. е. сторона шестиугольника равна 3, и, как видно из рисунка, шестиугольник составлен из шести рядов треугольников, в каждом из которых находится соответственно 7, 9, 11, 11, 9 и 7 треугольников.

Для случаев $n = 1$ и $n = 2$ соответствующий рисунок предлагается самостоятельно нарисовать заинтересованному читателю.

Дополним таблицу из формулировки задачи:

Размер стороны, n	3	1	2
Количество треугольников в рядах (сверху вниз)	7 9 11 11 9 7	3 3	5 7 7 5
Пример ввода	3 .000000 ...000000 .....000000 .....0..... ...000... .00000.	1 .0. 0.0	2 0000. 0000000 ..00.0. .0000
Пример вывода	.111224 ...332442 .....311122 .....1..... ...112... .33322.	nemoguce	1222. 1133111 ..31.2. .1122

Простейший подход к решению рассматриваемой задачи — рекурсивный перебор всех возможных приставлений трапеций (листинг 2).

```
begin
  InputData;
  Row:=1;
  Found0(Row,Column);
  DFS(Row,Column);
  writeln('nemoguce');
  close(input); close(output);
end.
```

Листинг 2

Функция *Found*0 (листинг 3) ищет первую ячейку с символом «0», начиная со строки *Row*, позиции 1.

Если ячеек с символом «0» больше нет, значит, все поле заполнили, выводим построенный ответ и останавливаемся. Если все варианты перебрали и ответ не нашли — выводим слово «nemoguce».

```
function Found0(var Row,Column: longint):
boolean;
var
  i,j: longint;
begin
  for i:=Row to 2*N  do
  for j:=1   to L[i] do
    if s[i,j]='0'
      then begin
             Row:=i;
             Column:=j;
             Found0:=true;
             exit;
           end;
  Found0:=false;
end;
```

Листинг 3

Процедура *DFS* (листинг 4) пытается приложить все варианты расположения трапеций.

На вводе в каждую строку добавляем по два пробела, чтобы не проверять отдельно границы и не менять четность столбца, от которой много зависит в этой задаче. А именно существуют описанные далее варианты прикладывания трапеции из трех треугольников в зависимости от номера строки и четности столбца (i — номер строки, j — номер столбца (номер элемента в строке)).

В трапецию включаются текущая позиция (i, j), а также еще две позиции — $(i1, j1)$, $(i2, j2)$, которые определяются смещением относительно (i, j) (+/–1 — вниз/вверх по строкам или влево/вправо по столбцам). Введем обычные обозначения: R — вправо, L — влево, D — вниз, U — вверх.

Тогда вне зависимости от строки и столбца можно брать вариант RR: взяли два треугольника вправо, первый из них характеризуется смещением (0, 1), второй — (0, 2).

В зависимости от строк и столбцов имеем такую таблицу:

	Нечетный столбец ($Odd(j)$)	Четный столбец (not $Odd(j)$)
$i < n$	R&D : (0, 1) (1, 1) DL : (1, 1) (1, 0) DR : (1, 1) (1, 2)	RD : (0, 1) (1, 2)
$i = n$	R&D : (0, 1) (1, 0) DL : (1, 0) (1, –1) DR : (1, 0) (1, 1)	RD : (0, 1) (1, 1)
$i = n + 1$	RD : (0, 1) (1, 0)	R&D : (0, 1) (1, –1) DL : (1, –1) (1, –2) DR : (1, –1) (1, 0)
$i > n + 1$	RD : (0, 1) (1, 0)	R&D : (0, 1) (1, –1) DL : (1, –1) (1, –2) DR : (1, –1) (1, 0)

Процедура *Check*($i0, j0, i1, j1, i2, j2$) (листинг 5):

- проверяет, стоят ли символы «0» на позициях $(i1, j1)$, $(i2, j2)$ (на позиции $(i0, j0)$ ноль стоит по построению);
- перебирает цвет C от 1 до 6 (до первого подходящего).

Функция *Bad* определяет, можно ли цветом C красить позиции $(i1, j1)$ и $(i2, j2)$ так, чтобы не получилось

```
procedure DFS(i,j: longint);
begin
{1RR} Check(i,j,i,j+1,i,j+2);

      if (i<N) and       Odd(j)  then begin
{1R&D}                                 Check(i,j,i,  j+1,i+1,j+1);
{1DR}                                  Check(i,j,i+1,j+1,i+1,j+2);
{1DL}                                  Check(i,j,i+1,j+1,i+1,j  );
                                     end;
{1RD} if (i<N) and (not Odd(j)) then Check(i,j,i,j+1,i+1,j+2);

      if (i=N) and       Odd(j)  then begin
{1R&D}                                 Check(i,j,i,  j+1,i+1,j  );
{1DL}                                  Check(i,j,i+1,j  ,i+1,j-1);
{1DR}                                  Check(i,j,i+1,j  ,i+1,j+1);
                                     end;
{1RD} if (i=N) and (not Odd(j)) then Check(i,j,i,j+1,i+1,j+1);

{2RD} if (i> N) and      Odd(j)  then Check(i,j,i,j+1,i+1,j);
      if (i> N) and (not Odd(j)) then begin
{2R&D}                                 Check(i,j,i,  j+1,i+1,j-1);
{2DR}                                  Check(i,j,i+1,j-1,i+1,j);
{2DL}                                  Check(i,j,i+1,j-1,i+1,j-2);
                                     end;
end;
```

Листинг 4

```
procedure Check(i0,j0,i1,j1,i2,j2: longint);
var
  Row,c: longint;
begin
  if (s[i1,j1]<>'0') or (s[i2,j2]<>'0') then exit;
  for c:=1 to 6 do
    begin
      if Bad(C,i0,j0,i1,j1,i2,j2) then continue;
      s[i0,j0]:=d[C]; s[i1,j1]:=d[C]; s[i2,j2]:=d[C];
      Row:=i0;
      if Found0(Row,Column)
        then DFS(Row,Column)
        else PrintHalt;
      s[i0,j0]:='0'; s[i1,j1]:='0'; s[i2,j2]:='0';
      break;
    end;
end;
```

Листинг 5

соприкосновения по стороне с ранее поставленной трапецией такого же цвета.

Если подходящий цвет найден, то рисуем им трапецию ($i0, j0$), ($i1, j1$), ($i2, j2$) и, если еще есть ячейки с «0», запускаем DFS от нее, иначе выводим ответ и останавливаемся.

После выхода из рекурсии возвращаем символы «0» на позициях ($i0, j0$), ($i1, j1$), ($i2, j2$) и выходим из цикла по цвету.

Функция *Bad* (листинг 6) проверяет с помощью *Bad*3 каждую из ячеек ($i0, j0$), ($i1, j1$), ($i2, j2$) по отдельности — можно ли ее покрасить цветом *C*.

Функция *Bad*3 (листинг 7) рассматривает соседние треугольники по таким правилам:

- левый и правый треугольники — со смещениями (0, –1) и (0, 1) от текущей позиции;
- верхний и нижний треугольники по следующим правилам в зависимости от строки и четности столбца:

	Нечетный столбец (*Odd*(j))	Четный столбец (not *Odd*(j))
$i < n$	D: (1, 1)	U: (–1, –1)
$i = n$	D: (1, 0)	U: (–1, –1)
$i = n + 1$	U: (–1, 0)	D: (1, –1)
$i > n + 1$	U: (–1, +1)	D: (1, –1)

Полный текст решения приведен в листинге 8.

```
function Bad(C,i0,j0,i1,j1,i2,j2: longint): boolean;
begin
  Bad:=True;
  if Bad3(C,i0,j0) then exit;
  if Bad3(C,i1,j1) then exit;
  if Bad3(C,i2,j2) then exit;
  Bad:= false;
end;
```

Листинг 6

```
function Bad3(C,i,j: longint): boolean;
begin
  Bad3:=(s[i,j-1]=d[C]) or (s[i,j+1]=d[c]) or
        ((i<N)   and     Odd(j) and ((s[i+1,j+1]=d[c])) or
         (i<N)   and not Odd(j) and ((s[i-1,j-1]=d[c])) or
         (i=N)   and     Odd(j) and ((s[i+1,j  ]=d[c])) or
         (i=N)   and not Odd(j) and ((s[i-1,j-1]=d[c])) or
         (i=N+1) and     Odd(j) and ((s[i-1,j  ]=d[c])) or
         (i=N+1) and not Odd(j) and ((s[i+1,j-1]=d[c])) or
         (i>N+1) and     Odd(j) and ((s[i-1,j+1]=d[c])) or
         (i>N+1) and not Odd(j) and ((s[i+1,j-1]=d[c])));
end;
```

Листинг 7

```
const
  MaxN=5;
  d : string = '123456';
var
  s: array [0..2*MaxN+1] of string;
  L: array [1..2*MaxN+1] of longint;
  N,Row,Column,Color : longint;

procedure InputData;
var
  i,j: longint;
  t  : string;
begin
  readln(N);
  for i:=1 to 2*N do
    begin
      readln(t);
      s[i]:='  '+t+'  ';
    end;
  s[0]:='  ';
  for i:=1 to N+2 do s[0]:=s[0]+' ';
  s[2*N+1]:='  ';
  for i:=1 to N+2 do s[2*N+1]:=s[2*N+1]+' ';
  for i:=1 to 2*N do L[i]:=length(s[i]);
end;

function Found0(var Row, Column: longint): boolean;
var
  i,j: longint;
begin
  for i:=Row to 2*N    do
    for j:=1 to L[i] do
      if s[i,j]='0'
        then begin
               Row:=i;
               Column:=j;
               Found0:=true;
               exit;
             end;
  Found0:=false;
end;

procedure PrintHalt;
var
  k: longint;
begin
  for k:=1 to 2*N do
    begin
      delete(s[k],1,2);
      writeln(s[k]);
    end;
  close(input); close(output);
  halt;
end;

procedure DFS(i,j: longint); forward;
```

Листинг 8

```
function Bad3(C,i,j: longint): boolean;
begin
  Bad3:=(s[i,j-1]=d[C]) or (s[i,j+1]=d[c]) or
        ((i<N)   and     Odd(j) and ((s[i+1,j+1]=d[c])) or
         (i<N)   and not Odd(j) and ((s[i-1,j-1]=d[c])) or
         (i=N)   and     Odd(j) and ((s[i+1,j  ]=d[c])) or
         (i=N)   and not Odd(j) and ((s[i-1,j-1]=d[c])) or
         (i=N+1) and     Odd(j) and ((s[i-1,j  ]=d[c])) or
         (i=N+1) and not Odd(j) and ((s[i+1,j-1]=d[c])) or
         (i>N+1) and     Odd(j) and ((s[i-1,j+1]=d[c])) or
         (i>N+1) and not Odd(j) and ((s[i+1,j-1]=d[c])));
end;

function Bad(C,i0,j0,i1,j1,i2,j2: longint): boolean;
begin
  Bad:=True;
  if Bad3(C,i0,j0) then exit;
  if Bad3(C,i1,j1) then exit;
  if Bad3(C,i2,j2) then exit;
  Bad:= false;
end;

procedure Check(i0,j0,i1,j1,i2,j2: longint);
var
  Row,c: longint;
begin
  if (s[i1,j1]<>'0') or (s[i2,j2]<>'0') then exit;
  for c:=1 to 6 do
    begin
      if Bad(C,i0,j0,i1,j1,i2,j2) then continue;
      s[i0,j0]:=d[C]; s[i1,j1]:=d[C]; s[i2,j2]:=d[C];
      Row:=i0;
      if Found0(Row,Column)
        then DFS(Row,Column)
        else PrintHalt;
      s[i0,j0]:='0'; s[i1,j1]:='0'; s[i2,j2]:='0';
      break;
    end;
end;

procedure DFS(i,j: longint);
begin
{1RR} Check(i,j,i,j+1,i,j+2);
      if (i<N) and       Odd(j)  then begin
{1R&D}                                  Check(i,j,i,  j+1,i+1,j+1);
{1DR}                                   Check(i,j,i+1,j+1,i+1,j+2);
{1DL}                                   Check(i,j,i+1,j+1,i+1,j  );
                                      end;
{1RD} if (i<N) and (not Odd(j)) then Check(i,j,i,j+1,i+1,j+2);

      if (i=N) and       Odd(j)  then begin
{1R&D}                                  Check(i,j,i,  j+1,i+1,j  );
{1DL}                                   Check(i,j,i+1,j  ,i+1,j-1);
{1DR}                                   Check(i,j,i+1,j  ,i+1,j+1);
                                      end;
{1RD} if (i=N) and (not Odd(j)) then Check(i,j,i,j+1,i+1,j+1);

{2RD} if (i> N) and      Odd(j)  then Check(i,j,i,j+1,i+1,j);
      if (i> N) and (not Odd(j)) then begin
{2R&D}                                  Check(i,j,i,  j+1,i+1,j-1);
{2DR}                                   Check(i,j,i+1,j-1,i+1,j);
{2DL}                                   Check(i,j,i+1,j-1,i+1,j-2);
                                      end;
end;
```

Листинг 8 (продолжение)

```
begin
  InputData;
  Row:=1;
  Found0(Row,Column);
  DFS(Row,Column);
  writeln('nemoguce');
  close(input); close(output);
end.
```

Листинг 8 (окончание)

Такое решение проходит все предлагаемые (организаторами олимпиады) для данной задачи тесты, кроме двух:

Ввод	5 00000000000 00.0000000000 0000.0000000000 000000.0000000000 00000000.0000000000 000000000.000000000 000000000.0000000 000000000.00000 000000000.000 000000000.0	5 00000000000 00.0000000000 000..0000000000 000000..000000000 00000000..000000000 000000000.000000000 000000000.0000000 000000000.00000 000000000.000 000000000.0
Вывод	12221112221 11.3332223311 1222.1114443222 211333.2221113331 12233111.3332224411 114431222.111333422 224311333.2221112 233222111.33322 111333222.112 222111333.1	11222111222 21.1112221112 122..3331113322 211333..222443111 12233111..111422233 114431222.222111443 224311333.3332224 233222111.11133 111333222.223 222111333.2

Причем первый тест проходит за 5 секунд (а по ограничениям задачи надо уложиться в 1 секунду), а вот второму тесту не хватает и 30 секунд.

Идея решения с мемоизацией.

Для ускорения используем мемоизацию по профилю.

Заведем массив *Was*[*Row*, *P*], все элементы которого изначально равны *false*, и по мере рекурсивного обхода будем заносить туда *true* после того, как убедимся рекурсивным перебором, что для строки *Row* из профиля *P* невозможно заполнить шестиугольник трапециями.

Пусть мы нашли очередной ноль в позиции (*i*, *j*). Зафиксируем с помощью функции *Profile*(*i*, *j*) состояние заполнения шестиугольника трапециями. Тогда при входе в рекурсию пишем проверку:

```
if Was[i,Profile(i,j)] then exit;
```

То есть, если мы уже пытались заполнять с такого профиля и не получилось, не будем повторять эту работу.

А перед выходом из рекурсии будем помечать тот факт, что заполнить не получилось:

```
Was[i,Profile(i,j)]:=true;
```

То есть рекурсивный перебор станет выглядеть так, как представлено в листинге 9.

Представленная в листинге 10 функция вычисления профиля *Profile*(*i*, *j*) анализирует как профиль:

- текущую строку *Row* с позиции *j* — это будут младшие биты профиля;
- следующую строку *Row*+1 (если она есть) с позиции 1 до позиции *j* включительно — это будут старшие биты профиля:

Применение данного подхода ускоряет решение, все тесты проходятся за 0,1 секунды.

Однако в двух тестах, которые представлены в таблице ниже, получается неверный ответ «nemoguce»,

```
procedure DFS(i,j: longint);
begin
  if Was[i,Profile(i,j)] then exit;

{1RR} Check(i,j,i,j+1,i,j+2);

      …

      if (i> N) and (not Odd(j)) then begin
{2R&D}                                  Check(i,j,i,  j+1,i+1,j-1);
{2DL}                                   Check(i,j,i+1,j-1,i+1,j-2);
{2DR}                                   Check(i,j,i+1,j-1,i+1,j);
                                      end;

  Was[i,Profile(i,j)]:=true;
end;
```

Листинг 9

```
function Profile(i,j: longint): longint;
var
  t,p: longint;
begin
  p:=0;
  for t:=j to L[i] do
    if (s[i,t]>='1') and (s[i,t]<='6')
      then p:=p or (1 shl (4*N+2-(t-3)));
  if i<2*N
    then begin
           for t:=3 to j do
             if (s[i+1,t]>='1') and (s[i+1,t]<='6')
               then p:=p or (1 shl (4*N+1-(t-3)));
         end;
  Profile := p;
end;
```

Листинг 10

хотя на самом деле решение существует — см. вторую строку таблицы:

Ввод	5 00000000000 0000000000000 000000000000000 00000000000000000 00.0.0.0.0.0.0.0.0. 000000000000000000 00000000000000000 000000000000000 0000000000000 00000000000	5 00000000000 0000000000000 000000000000000 00000000000000000 00.....0.0.0.0.0.0. 000000000000000000 00.00000000000000 000000000000000 0000000000000 00000000000
Вывод	11122211122 2223331222123 111444113331133 12233112211223311 11.2.3.1.2.1.2.3.1. 2221112221112221122 33322233322233142 111333111444344 2221112211122 33344423332	11122211122 2223331222123 111444113331133 22333112211223311 12.....1.2.1.2.3.1. 1122211122211122211 22.22233322233321 211144411144422 2221112233311 33344424441

Можно предположить, что имеется ошибка в вычислении профиля с помощью функции *Profile*(*i*, *j*).

На самом деле, для корректной мемоизации нужно запоминать профиль для позиции (*i*, *j*), а не для строки *i*. Такое решение берет все тесты, но за время 1,3 секунды на половине тестов. В задаче же установлен лимит в 1 секунду.

Переход от строк к числам не принес принципиального изменения по времени выполнения.

Анализ приводит к пониманию того, что для ускорения надо чуть подправить функцию вычисления профиля для строки: вместо формирования профиля по заполненным позициям (≥'1' и ≤'6') формировать профиль по незаполненным позициям текущей и следующей строк (*s*[*i*, *t*]='0'), *s*[*i*+1, *t*]='0') (листинг 11).

Получаем *полное решение*, которое проходит все тесты за 0,1 секунды (листинг 12).

5. Заключение

В данной статье на примере решения двух задач рассмотрена методика изучения темы «Ускорение рекурсивных решений при помощи мемоизации», предлагающая, по мнению автора, наиболее простой способ постепенного осознания учащимися механизма рекурсии и способа решения задач с ее помощью. Методика включает

```
function Profile(i,j: longint): longint;
var
  t,p: longint;
begin
  p:=0;
  if i<2*N
    then begin
           for t:=3 to j do
             if s[i+1,t]='0'
               then p:=p or (1 shl (4*N+2-(t-3)));
         end;
  for t:=j to L[i] do
    if s[i,t]='0'
      then p:=p or (1 shl (4*N+1-(t-3)));
      Profile := p;
end;
```

Листинг 11

```
const
  MaxN=5;
  d: string = '123456';
  MaxW=8*1024*1024;
var
  s: array [0..2*MaxN+1] of string;
  L: array [1..2*MaxN+1] of longint;
  N,Row,Column: longint;
  Was: array [1..2*MaxN,0..MaxW] of boolean;

procedure InputData;
var
  i,j: longint;
  t  : string;
begin
  readln(N);
  for i:=1 to 2*N do
    begin
      readln(t);
      s[i]:='  '+t+'  ';
    end;
  s[0]:='  ';
  for i:=1 to N+2 do s[0]:=s[0]+' ';
  s[2*N+1]:='  ';
  for i:=1 to N+2 do s[2*N+1]:=s[2*N+1]+' ';
  for i:=1 to 2*N do L[i]:=length(s[i]);
  for i:=1 to 2*N do
    for j:=0 to MaxW do Was[i,j]:=false;
end;

function Found0(var Row, Column: longint): boolean;
var
  i,j: longint;
begin
  for i:=Row to 2*N do
  for j:=1 to L[i] do
    if s[i,j]='0'
      then begin
             Row:=i; Column:=j; Found0:=true; exit;
           end;
  Found0:=false;
end;

procedure PrintHalt;
var
  k: longint;
begin
  for k:=1 to 2*N do
    begin
      delete(s[k],1,2); writeln(s[k]);
    end;
  close(input); close(output); halt;
end;

procedure DFS(i,j: longint); forward;
```

Листинг 12

```
function Bad3(C,i,j: longint): boolean;
begin
  Bad3:=(s[i,j-1]=d[C]) or (s[i,j+1]=d[c]) or
        ((i<N)   and     Odd(j) and ((s[i+1,j+1]=d[c])) or
         (i<N)   and not Odd(j) and ((s[i-1,j-1]=d[c])) or
         (i=N)   and     Odd(j) and ((s[i+1,j  ]=d[c])) or
         (i=N)   and not Odd(j) and ((s[i-1,j-1]=d[c])) or
         (i=N+1) and     Odd(j) and ((s[i-1,j  ]=d[c])) or
         (i=N+1) and not Odd(j) and ((s[i+1,j-1]=d[c])) or
         (i>N+1) and     Odd(j) and ((s[i-1,j+1]=d[c])) or
         (i>N+1) and not Odd(j) and ((s[i+1,j-1]=d[c])));
end;

function Bad(C,i0,j0,i1,j1,i2,j2: longint): boolean;
begin
  Bad:=True;
  if Bad3(C,i0,j0) then exit;
  if Bad3(C,i1,j1) then exit;
  if Bad3(C,i2,j2) then exit;
  Bad:= false;
end;

procedure Check(i0,j0,i1,j1,i2,j2: longint);
var
  Row,c: longint;
begin
  if (s[i1,j1]<>'0') or (s[i2,j2]<>'0') then exit;
  for c:=1 to 6 do
    begin
      if Bad(C,i0,j0,i1,j1,i2,j2) then continue;
      s[i0,j0]:=d[C]; s[i1,j1]:=d[C]; s[i2,j2]:=d[C];
      Row:=i0;
      if Found0(Row,Column)
        then DFS(Row,Column)
        else PrintHalt;
      s[i0,j0]:='0'; s[i1,j1]:='0'; s[i2,j2]:='0';
      break;
    end;
end;

function Profile(i,j: longint): longint;
var
  t,p: longint;
begin
  p:=0;
  if i<2*N
    then begin
           for t:=3 to j do
             if s[i+1,t]='0'
               then p:=p or (1 shl (4*N+2-(t-3)));
         end;
  for t:=j to L[i] do
    if s[i,t]='0'
      then p:=p or (1 shl (4*N+1-(t-3)));
  Profile := p;
end;
```

Листинг 12 (продолжение)

```
procedure DFS(i,j: longint);
begin
  if Was[i,Profile(i,j)] then exit;
{1RR} Check(i,j,i,j+1,i,j+2);
      if (i<N) and       Odd(j)  then begin
{1R&D}                                  Check(i,j,i,  j+1,i+1,j+1);
{1DL}                                   Check(i,j,i+1,j+1,i+1,j  );
{1DR}                                   Check(i,j,i+1,j+1,i+1,j+2);
                                      end;
{1RD} if (i<N) and (not Odd(j)) then Check(i,j,i,j+1,i+1,j+2);
      if (i=N) and       Odd(j)  then begin
{1R&D}                                  Check(i,j,i,  j+1,i+1,j  );
{1DL}                                   Check(i,j,i+1,j  ,i+1,j-1);
{1DR}                                   Check(i,j,i+1,j  ,i+1,j+1);
                                      end;
{1RD} if (i=N) and (not Odd(j)) then Check(i,j,i,j+1,i+1,j+1);

{2RD} if (i> N) and      Odd(j)  then Check(i,j,i,j+1,i+1,j);
      if (i> N) and (not Odd(j)) then begin
{2R&D}                                  Check(i,j,i,  j+1,i+1,j-1);
{2DL}                                   Check(i,j,i+1,j-1,i+1,j-2);
{2DR}                                   Check(i,j,i+1,j-1,i+1,j);
                                      end;
  Was[i,Profile(i,j)]:=true;
end;

begin
  InputData;
  Row:=1;
  Found0(Row,Column);
  DFS(Row,Column);
  writeln('nemoguce');
  close(input);  close(output);
end.
```

Листинг 12 (окончание)

в себя последовательность задач в порядке возрастания сложности, снабженных, где необходимо, предварительными общими пояснениями и последующими полными решениями предлагаемых задач. В статье предложены две таких задачи на применение мемоизации. Серьезной технической основой является разработанная под управлением автора инструментальная система дистанционного обучения (http://dl.gsu.by), которая позволяет максимально автоматизировать процесс предъявления условий задач и проверки их решений.

Список источников

1. *Баррон Д.* Рекурсивные методы в программировании. М.: Мир,1974. 79 с.
2. *Бердж В.* Методы рекурсивного программирования. М.: Машиностроение, 1983. 256 с.
3. *Головешкин В. А., Ульянов В. В.* Теория рекурсии для программистов. М.: Физматлит, 2006. 296 с.
4. *Дасгупта С., Пападимитриу Х., Вазирани У.* Алгоритмы. М.: МЦНМО, 2019. 320 с.
5. *Долинский М. С.* Введение в решение задач с помощью рекурсивных процедур и функций // Информатика в школе. 2016. № 9. С. 49–56. EDN: XEQEZR.
6. *Долинский М. С.* Генерация комбинаторных объектов с помощью рекурсивных процедур и функций // Информатика в школе. 2019. № 4. С. 59–63. DOI: 10.32517/2221-1993-2019-18-4-59-63. EDN: TCJKQY.
7. *Долинский М. С.* Рекурсивное решение задач с помощью метода «разделяй и властвуй» // Информатика в школе. 2022. № 2. С. 58–64. DOI: 10.32517/2221-1993-2022-21-2-58-64. EDN: JOEVGG.
8. *Долинский М. С.* Решение задач рекурсивной генерацией чисел // Информатика в школе. 2021. № 1. С. 46–51. DOI: 10.32517/2221-1993-2021-20-1-46-51. EDN: OIHIZJ.
9. *Долинский М. С.* Решение игровых задач с помощью рекурсии // Информатика в школе. 2021. № 9. С. 43–50. DOI: 10.32517/2221-1993-2021-20-9-43-50. EDN: XQIAMP.
10. *Долинский М. С.* Решение рекурсивных задач по определению // Информатика в школе. 2020. № 2. С. 60–66. DOI: 10.32517/2221-1993-2020-19-2-60-66. EDN: EAYQZS.
11. *Златопольский Д. М.* 1400 задач по программированию. М.: ДМК Пресс, 2019. 192 с.
12. *Луридас П.* Алгоритмы для начинающих. Теория и практика для разработчика. М.: ЭКСМО, 2018. 608 с.
13. *Пароджанов В. Д.* Дружелюбные алгоритмы, понятные каждому. М.: ДМК Пресс, 2014. 464 с.
14. *Потопахин В. В.* Искусство алгоритмизации. М.: ДМК Пресс, 2013. 320 с.

15. *Рафгарден Т.* Совершенный алгоритм. Графовые алгоритмы и структуры данных. СПб.: Питер, 2020. 256 с.

16. *Рафгарден Т.* Совершенный алгоритм. Жадные алгоритмы и динамическое программирование. СПб.: Питер, 2020. 256 с.

17. *Рафгарден Т.* Совершенный алгоритм. Основы. СПб.: Питер, 2019. 256 с.

18. *Рубио-Санчес М.* Введение в рекурсивное программирование. М.: ДМК Пресс, 2019. 436 с.

19. *Скиена С.* Алгоритмы. Руководство по разработке. СПб.: BHV, 2011. 720 с.

20. *Солтис М.* Введение в анализ алгоритмов. М.: ДМК Пресс, 2019. 278 с.

21. *Стивенс Р.* Алгоритмы. Теория и практическое применение. М.: ЭКСМО, 2016. 544 с.

22. *Шень А.* Программирование: теоремы и задачи. М.: МЦНМО, 2017. 320 с.

23. *Castro R., Lehmann N., Pérez J., Subercaseaux B.* Wavelet trees for competitive programming // Olympiad in Informatics. 2016. Vol. 10. P. 19–37.

24. *Do P. T., Pham B. T., Than V. C.* Latest algorithms on particular graph classes // Olympiad in Informatics. 2020. Vol. 14. P. 21–35.

25. *Erdősné Németh A.* Teaching graphs for contestants in lower-secondary-school-age // Olympiad in Informatics. 2017. Vol. 11. P. 41–53.

26. *Erdősné Németh A., Zsakó L.* The place of the dynamic programming concept in the progression of contestants' thinking // Olympiad in Informatics. 2016. Vol. 10. P. 61–72.

27. *Forisek M.* Towards a better way to teach dynamic programming // Olympiad in Informatics. 2015. Vol. 9. P. 45–55.

28. *Manev K.* Tasks on graphs // Olympiad in Informatics. 2008. Vol. 2. P. 90–104.

29. *Manev K., Nikolov N., Markov M.* Reconstruction of trees using metric properties // Olympiad in Informatics. 2011. Vol. 5. P. 82–91.

30. *Pachocki J., Radoszewskij J.* Where to use and how not to use polynomial string hashing // Olympiad in Informatics. 2013. Vol. 7. P. 90–100.