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

РЕШЕНИЕ ИГРОВЫХ ЗАДАЧ С ПОМОЩЬЮ РЕКУРСИИ*

Аннотация
В статье описана методика изучения темы «Решение игровых задач с помощью рекурсии» 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 руководством 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 ее и послать решение повторно. Кроме того, для каждой задачи на сайте 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, олимпиады по информатике, инструментальная система дистанционного обучения.
DOI: 10.32517/2221-1993-2021-20-9-43-50

Введение

Подготовка школьников к олимпиадам по информатике и программированию требует значительного объема теоретических знаний и практических навыков решения задач. Одно из базовых умений в такой подготовке — использование рекурсии. Тема достаточно регулярно поднимается в компьютерной книжной и журнальной литературе как с теоретической точки зрения (см., например, [1–4, 10, 12–15]), так и в применении к различным задачам [9, 16]. Особо можно выделить тему «Графы» [11, 17–20], в которой рекурсивные алгоритмы необходимо применять для решения практически всех задач. Автор вносит свою лепту в разработку темы «Рекурсия» [5–8] собственным 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]. В статье [8] представлены решения рекурсивных задач «по определению». В работе [7] описано решение задач рекурсивной генерацией чисел.

В настоящей статье предлагается материал по решению игровых задач с помощью рекурсии, включающий условия задач и примеры их решения (задачи взяты из олимпиад по программированию разных лет).

Проверка решений осуществляется автоматически на сайте DL.GSU.BY. Вдумчивым читателям предлагается после чтения условия задачи попытаться решить ее самостоятельно, а если решить не удалось, аналогичную попытку рекомендуется предпринять перед чтением текста решения каждой задачи, попытавшись вначале воспользоваться приведенными предварительно рекомендациями по алгоритму решения задачи.

* Материалы к статье можно скачать на сайте ИНФО: http://infojournal.ru/journals/school/school_09-2021/

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

SOLVING GAME PROBLEMS USING RECURSION

Abstract
The article describes the methodology to game problems of Olympiads in informatics using recursive numbers generation. The study is based on the sequential solution of increasingly complex tasks. For each task, the following materials are given: the condition of the task, the idea of the solution with a proposal to come up with an independent implementation, the solution in the Pascal programming language. Distance learning system DL.GSU.BY is the effective technical base for teaching. It allows offer the student a condition of the task; send the solution for review; get a verdict from the system — a correct or incorrect solution; for incorrect solutions, 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 on solving this problem and / or read the answer if the questions have already been asked before.
Keywords: recursion, Olympiads in informatics, distance learning tools.

Задача 1. Checkers (USACO 2008 December Bronze)

Коровы хотят, чтобы вы помогли им определить результат их игры. Задана шашечная доска размером $N \times N$ ($4 \le N \le 30$). Определите оптимальное множество ходов, чтобы закончить игру на следующем ходе. Шашки ходят только по квадратикам, отмеченным знаком «+», а «сбивают», перепрыгивая через шашку противоположного цвета. «Перепрыгнутая» шашка удаляется. Смотрите пример ниже ($N = 8$):

```
- + - + - + - +
+ - + - + - + -
- + - K - + - +
+ - + - + - + -
- o - o - + - +
+ - K - + - + -
- o - + - + - +
+ - K - + - K -
```

Символами K помечены шашки Бесси. Сейчас ее ход.

Решение данной задачи заключается в том, что нижняя левая шашка K должна сбить все три шашки противника, тем самым завершая игру.

Исходное

```
- + - + - + - +
+ - + - + - + -
- + - K - + - +
+ - + - + - + -
- o - o - + - +
+ - K - + - + -
- o - + - + - +
+ ->K<- + - K -
```

После шага 1

```
- + - + - + - +
+ - + - + - + -
- + - K - + - +
+ - + - + - + -
- o - o - + - +
>K<- K - + - + -
- + - + - + - +
+ - + - + - K -
```

После шага 2

```
- + - + - + - +
+ - + - + - + -
- + - K - + - +
+ ->K<- + - + -
- + - o - + - +
+ - K - + - + -
- + - + - + - +
+ - K - + - K -
```

После шага 3

```
- + - + - + - +
+ - + - + - + -
- + - K - + - +
+ - + - + - + -
- + - + - + - +
+ - K ->K<- + -
- + - + - + - +
+ - K - + - K -
```

Шаги будут сделаны по следующим клеткам:

```
  1 2 3 4 5 6 7 8
1 - + - + - + - +
2 + - + - + - + -
3 - + - K - + - +
4 + - * - + - + -
5 - o - o - + - +
6 * - K - * - + -
7 - o - + - + - +
8 + - K - + - K -
```

```
          R C
начало:   8 3
шаг:      6 1
шаг:      4 3
шаг:      6 5
```

Напишите программу, которая определяет уникальную последовательность ходов, завершающую игру (если такая последовательность существует). На доске всегда будет как минимум по одной шашке черного и белого цветов.

Формат ввода.

Строка 1: одно целое число N.

Строки 2..N+1: строка i содержит N символов (каждый символ — один из «–», «+», «K», «o»), она представляет строку i заданной позиции.

Пример ввода (файл chkr.in):

```
8
-+-+-+-+
+-+-+-+-
-+-K-+-+
+-+-+-+-
-o-o-+-+
+-K-+-+-
-o-+-+-+
+-K-+-K-
```

Формат вывода.

Строки 1..?:

- если решения нет, выводите «impossible»;
- если решение есть, выводите ответ в несколько строк. Каждая строка содержит два целых числа, разделенных пробелом, — положение вашей шашки перед очередным ходом.

Принимается любая выигрывающая последовательность ходов.

Пример вывода (файл chkr.out):

```
8 3
6 1
4 3
6 5
```

Идея решения.

Посчитаем, сколько шашек противника («o») надо сбить. Для каждого поля, на котором стоит наша шашка («K»), рекурсивно проверим, можем ли мы с него сбить все шашки противника. Если ни с одного не сможем — ответ «impossible».

Главная программа может выглядеть так:

```
readln(N);
for i:=1 to N do
  readln(s[i]);
ko:=0;
for i:=1 to N do
  for j:=1 to N do
    if s[i,j]='o' then inc(ko);
for i:=1 to N do
  for j:=1 to N do
    if s[i,j]='K'
    then
      begin
        A[1]:=i;
        B[1]:=j;
        Rec(i,j,1);
      end;
assign(output,'chkr.out'); rewrite(output);
writeln('impossible');
```

Рекурсивная процедура *Rec*(i, j, 1) проверяет, можно ли с клетки (i, j) сбить все *ko* шашек противника, если уже одно поле (начальное) запомнено, следующим образом.

Если уже все *ko* шашек противника сбиты, вызываем процедуру вывода ответа.

Иначе проверяем все возможные четыре диагональные направления:

(i, j) — где стоим;

(ti, tj) — шашка соперника, которую будем сбивать;

(ni, nj) — место, куда попадем после текущего сбития.

Если можем сбить, дополняем путь (массивы *A* и *B*):

```
procedure Rec(i,j,h: longint);
var
  ti,tj,ni,nj,k: longint;
begin
  if h=ko+1 then Print_And_Halt;
  for k:=1 to 4 do
    begin
      ti:=i+step[k,1]; tj:=j+step[k,2];
      ni:=i+step[k,3]; nj:=j+step[k,4];
      if (ti>1) and (ti<N) and
         (tj>1) and (tj<N) and
         (s[ti,tj]='o')    and
         (s[ni,nj]='+')
      then
        begin
          s[ti,tj]:='+';
          inc(h); A[h]:=ni; B[h]:=nj;
          Rec(ni,nj,h);
          s[ti,tj]:='o';
          h:=1;
        end;
    end;
end;
```

Далее представлен полный текст решения:

```
const
  step: array [1..4,1..4]
    of longint = ((1,1,2,2),(1,-1,2,-2),
                  (-1,1,-2,2),(-1,-1,-2,-2));
var
  s       : array [1..30] of string;
  i,j,N,ko: longint;
  A,B     : array [1..450] of longint;

procedure Print_And_Halt;
var
  i: longint;
begin
  assign(output,'chkr.out'); rewrite(output);
  for i:=1 to ko+1 do
    writeln(a[i],' ',b[i]);
  close(output);
  halt;
end;

procedure Rec(i,j,h: longint);
var
  ti,tj,ni,nj,k: longint;
begin
  if h=ko+1 then Print_And_Halt;
  for k:=1 to 4 do
    begin
      ti:=i+step[k,1]; tj:=j+step[k,2];
      ni:=i+step[k,3]; nj:=j+step[k,4];
      if (ti>1) and (ti<N) and
         (tj>1) and (tj<N) and
         (s[ti,tj]='o')    and
         (s[ni,nj]='+')
      then
        begin
          s[ti,tj]:='+';
          inc(h); A[h]:=ni; B[h]:=nj;
          Rec(ni,nj,h);
          s[ti,tj]:='o';
          h:=1;
        end;
    end;
end;

begin
  assign(input,'chkr.in'); reset(input);
  readln(N);
  for i:=1 to N do
    readln(s[i]);
  ko:=0;
  for i:=1 to N do
    for j:=1 to N do
      if s[i,j]='o' then inc(ko);
  for i:=1 to N do
    for j:=1 to N do
      if s[i,j]='K'
      then
        begin
          A[1]:=i;
          B[1]:=j;
          Rec(i,j,1);
        end;
  assign(output,'chkr.out'); rewrite(output);
  writeln('impossible');
  close(input); close(output);
end.
```

Задача 2. Шашки (9-я интернет-олимпиада 2007 года, Санкт-Петербург)

Сема — любитель игр на шахматной доске. Больше всего он любит играть в шашки. Сема решил написать программу, которая будет играть в шашки. По его задумке она будет показывать, какие шашки можно взять на данном ходе. Ваша задача — реализовать эту функцию. Напомним, что взять можно только по диагонали одним из четырех способов:

После того как белая шашка перемещается на пустое поле, черная шашка снимается с доски и считается взятой. При этом, если после перемещения белой шашки у нее вновь появляется возможность взять, то она продолжает свой ход. Аналогичны правила и для черных шашек. Отметим, что в рассматриваемом варианте игры в шашки отсутствует понятие «дамка», т. е. возможности шашки по взятию не зависят от того, доходила она до последней горизонтали или нет.

Формат ввода.

В первых восьми строках входного файла записаны по восемь символов из множества {«.», «B», «W»}, ко-

торые обозначают пустое поле, черную шашку, белую шашку соответственно. Во входном файле не более 12 шашек каждого цвета. Все шашки расположены либо на черных, либо на белых полях.

Формат вывода.

В выходной файл выдайте поля, на которых стоят шашки, которые можно взять, если ходят белые или черные. Используйте формат вывода, аналогичный показанному в примерах. Отсортируйте поля сначала по первой координате (измеряется по вертикали), а при равенстве первых — по второй (измеряется по горизонтали). Учитывайте, что первый символ первой строки входного файла соответствует полю (1, 1), последний символ первой строки — полю (1, 8), первый символ последней строки — полю (8, 1), последний символ последней строки — (8, 8).

Пример ввода	*Пример вывода*	*Пояснения*
.W.W.W.W W.W.W.W. .W.W.W.W .... B.B.B.B. .B.B.B.B B.B.B.B.	White: 0 Black: 0	Никакая шашка не может брать

Пример ввода	*Пример вывода*	*Пояснения*
.W.W.W.W W.W.W.W. .....W.. W.W...W. .B.B.... B...B.B. .B.B.B.B B.B...B.	White: 3 (5, 2), (5, 4), (7, 4) Black: 1 (4, 3)	Белая шашка, стоящая на (4; 1), может взять сначала черную шашку на поле (5; 2) и, переместившись на поле (6; 3), взять черную шашку, стоящую на поле (5; 4). Также с поля (6; 3) можно взять еще одну черную шашку, стоящую на поле (7; 4), при этом придется двигаться другим путем. Черные же могут взять лишь белую шашку, которая стоит на поле (4; 3)

Идея решения.

Идея аналогична идее решения задачи 1.

Для каждой шашки нужно рекурсивно выяснить, какие шашки противника она может побить.

Опишем решение более детально.

Главная программа выглядит следующим образом:

```
for i:=1 to 8 do
  readln(s[i]);
for i:=1 to 8 do
  for j:=1 to 8 do
    B[i,j]:='.';
for i:=1 to 8 do
  for j:=1 to 8 do
    case s[i,j] of
      'B' : Rec(i,j,'W');
      'W' : Rec(i,j,'B');
    end;
Calculate(kW,kB);
Print('White: ','B',KB);
Print('Black: ','W',kW);
```

Вводим исходное поле в массив строк S.

Заводим массив символов B[1..8, 1..8], прописываем вначале все позиции символом «.». (Планируется вносить в этот массив все шашки, которые можно сбить, «B» будет обозначать черную шашку, «W» — белую.) Далее для каждой позиции на доске:

- если там стоит черная шашка, вызываем рекурсивную процедуру *Rec*, которая вносит в массив B все сбиваемые с этой позиции белые шашки;
- если там стоит белая шашка, вызываем рекурсивную процедуру *Rec*, которая вносит в массив B все сбиваемые с этой позиции черные шашки.

После завершения этой работы вызываем процедуру *Calculate*, которая считает количества сбитых белых и черных шашек (*kW* и *kB* соответственно).

```
procedure Calculate(var kW,kB: longint);
var
  i: longint;
begin
  kW:=0; kB:=0;
  for i:=1 to 8 do
    for j:=1 to 8 do
      begin
        if B[i,j]='W' then inc(kW);
        if B[i,j]='B' then inc(kB);
      end;
end;
```

Для вывода ответа два раза вызываем процедуру *Print*. Эта процедура выводит заголовок, а затем просматривает массив *B*, и если там стоит нужная шашка, то ее координаты заносятся в формируемую к выводу строку *R*.

```
procedure Print(s: string; c: char; k:
longint);
var
  i,j: longint;
  r  : string;
begin
  writeln(s,k);
  if k=0 then exit;
  R:='';
  for i:=1 to 8 do
    for j:=1 to 8 do
      if B[i,j]=c
      then R:=R+'('+chr(i+ord('0'))+', '
                 +chr(j+ord('0'))+'), ';
  delete(R,length(R)-1,2);
  writeln(R);
end;
```

Рекурсивная процедура *Rec*:

```
procedure Rec(i,j: longint; C: char);
var
  ti,tj,ni,nj,k: longint;
begin
  for k:=1 to 4 do
    begin
      ti:=i+step[k,1]; tj:=j+step[k,2];
      ni:=i+step[k,3]; nj:=j+step[k,4];
      if (ti>1) and (ti<8) and
         (tj>1) and (tj<8) and
         (s[ti,tj]=C)      and
```

```
        (s[ni,nj]='.')
      then
        begin
          B[ti,tj]:=C;
          s[ti,tj]:='.';
          Rec(ni,nj,C);
          s[ti,tj]:=C;
        end;
    end;
end;
```

получает в качестве параметров координаты (i, j) текущей шашки и цвет сбиваемых шашек C и проверяет все четыре направления, задаваемые массивом констант *step* в главной программе:

```
const
  step: array [1..4,1..4]
    of longint = ((1,1,2,2),(1,-1,2,-2),
                  (-1,1,-2,2),(-1,-1,-2,-2));
```

Первые два числа — относительные координаты сбиваемой шашки, вторые два числа — относительные координаты места, куда попадает бьющая шашка после взятия.

Если есть что бить (по всем правилам), то помечаем это место и вызываем рекурсию с позиции после взятия. После возвращения из рекурсии ставим убранную шашку туда, «где была».

Полный текст решения представлен ниже:

```
const
  step: array [1..4,1..4]
          of longint = ((1,1,2,2),(1,-1,2,-2),
                    (-1,1,-2,2),(-1,-1,-2,-2));
var
  s        : array [1..30] of string;
  B        : array [1..8,1..8] of char;
  i,j,kW,kB: longint;

procedure Calculate(var kW,kB: longint);
var
  i: longint;
begin
  kW:=0; kB:=0;
  for i:=1 to 8 do
    for j:=1 to 8 do
      begin
        if B[i,j]='W' then inc(kW);
        if B[i,j]='B' then inc(kB);
      end;
end;

procedure Print(s: string; c: char; k:
longint);
var
  i,j: longint;
  r  : string;
begin
  writeln(s,k);
  if k=0 then exit;
  R:='';
  for i:=1 to 8 do
    for j:=1 to 8 do
      if B[i,j]=c
      then R:=R+'('+chr(i+ord('0'))+', '
                  +chr(j+ord('0'))+'), ';
  delete(R,length(R)-1,2);
  writeln(R);
end;

procedure Rec(i,j: longint; C: char);
var
  ti,tj,ni,nj,k: longint;
begin
  for k:=1 to 4 do
    begin
      ti:=i+step[k,1]; tj:=j+step[k,2];
      ni:=i+step[k,3]; nj:=j+step[k,4];
      if (ti>1) and (ti<8) and
         (tj>1) and (tj<8) and
         (s[ti,tj]=C)      and
         (s[ni,nj]='.')
      then
        begin
          B[ti,tj]:=C;
          s[ti,tj]:='.';
          Rec(ni,nj,C);
          s[ti,tj]:=C;
        end;
    end;
end;

begin
  assign(input,'checkers.in'); reset(input);
  assign(output,'checkers.out');
rewrite(output);
  for i:=1 to 8 do readln(s[i]);
  for i:=1 to 8 do
    for j:=1 to 8 do
      B[i,j]:='.';
  for i:=1 to 8 do
    for j:=1 to 8 do
      case s[i,j] of
       'B' : Rec(i,j,'W');
       'W' : Rec(i,j,'B');
      end;
  Calculate(kW,kB);
  Print('White: ','B',KB);
  Print('Black: ','W',kW);
  close(input); close(output);
end.
```

Задача 3. Jigsaw Puzzles (USACO 2008 December Silver)

Коровы играют с пазлами «jigsaw». В этих пазлах имеется R ($1 \le R \le 10$) и C ($1 \le C \le 10$) столбцов, а края соединяются не формой, а буквами. Каждый квадратик пазла имеет число (последовательный номер) внутри и четыре буквы (или внешние границы, обозначенные цифрой 0 у нас). Кусок пазла может иметь не более двух границ (угловой кусок). Ниже представлено множество из шести кусков (границы помечены символами линий вместо 0), собранных одним из множества способов в пазл:

```
+---+  +---+  +---+
| 1 c  c 3 d  d 5 |
+-d-+  + a +  +-e-+

+-d-+  +-a-+  +-e-+
| 2 b  b 4 b  b 6 |
+---+  +---+  +---+
```

Заметим, что крайние буквы соседних кусков одинаковые, а границы появляются только сверху/снизу и по сторонам пазла.

Куски пазлов задаются своими последовательными номерами и списком их четырех границ в порядке обхода по часовой стрелке (граница — это либо символ от a до z, либо 0).

По заданному описанию кусков пазла найдите хотя бы один способ собрать эти куски в пазл.

Формат ввода.

Строка 1: два разделенных пробелом целых числа: R и C.

Строки 2..R×C+1: каждая строка содержит пять разделенных пробелом сущностей — последовательный номер и четыре идентификатора границы.

Пример ввода (файл jigsaw.in):

```
2 3
1 c d 0 0
2 0 d b 0
3 c 0 d a
4 b a b 0
5 d 0 0 e
6 0 0 b e
```

Пояснения к вводу.

Входной файл описывает входной пазл, при этом некоторые куски могут быть повернуты относительно приведенного решения.

Формат вывода.

Строки 1..R×C: строка $R \times (i-1)+j$ описывает кусок пазла, который необходимо разместить на позицию «строка i, колонка j» с помощью одного целого числа и четырех идентификаторов границ: верхней, правой, нижней, левой.

Пример вывода (файл jigsaw.out):

```
1 0 c d 0
3 0 d a c
5 0 0 e d
2 d b 0 0
4 a b 0 b
6 e 0 0 b
```

Пояснения к выводу.

Ответ соответствует диаграмме, приведенной в тексте задачи. Другие решения (такие, как отражения) возможны, поскольку они проверяются специально написанной программой.

Идея решения.

Основная идея решения проста.

Выполняем рекурсивный перебор — на каждой из позиций (i, j) «примерять» каждый пазл, сделав все четыре его поворота.

Если пазл подошел, то переходим к следующей позиции.

Если ни один пазл не подходит, то возвращаемся к предыдущей позиции и ставим туда другой подходящий пазл.

Основные трудности кроются в деталях, которые раскрываются в дальнейших пояснениях.

Главная программа может выглядеть так:

```
readln(R,C);
RC:=R*C;
for i:=1 to RC do
  begin
    readln(n,s);
    d:=length(s);
    p:=s[d-6]+s[d-4]+s[d-2]+s[d];
    z[n]:=p+p;
    x[n]:=0;
  end;
close(input);
for i:=1 to R do L[i,1]:='0';
for j:=1 to C do U[1,j]:='0';
Rec(1,1);
```

Здесь:

$Z[n]$ — строка, описывающая пазл с номером n четырьмя символами;

с первой позиции этой строки будет идти описание исходного пазла, со второй позиции — описание пазла, повернутого на 90 градусов, с третьей — повернутого на 180 градусов, с четвертой — повернутого на 270 градусов;

$X[n] = 1$ — признак того, что пазл n поставлен на некоторое место, $X[n] = 0$ — пазл еще не использован;

$L[i,j]$ — показывает, какой символ должен быть слева у пазла в позиции (i, j);

$U[i, j]$ — показывает, какой символ должен быть сверху у пазла в позиции (i, j).

Рекурсивный перебор пазлов осуществляется процедурой *Rec*:

```
procedure Rec(i,j: longint);
var
  f,k,v,w: longint;
  p,E    : string;
begin
  for f:=1 to RC do
    if x[f]=0
    then
      begin
        p:=z[f];
        E:=L[i,j]+U[i,j];
        for k:=1 to 4 do
          if (E=copy(p,k,2)) and
             (((j<C) and (p[k+2]<>'0') or
              ((j=C) and (p[k+2]='0')))) and
             (((i<R) and (p[k+3]<>'0') or
              ((i=R) and (p[k+3]='0'))))
            then
              begin
                x[f]:=1;
                a[i,j]:=f; b[i,j]:=k;
                L[i,j+1]:=p[k+2];
                U[i+1,j]:=p[k+3];
                v:=i; w:=j;
                if Next(v,w)
```

```
                then Rec(v,w)
                else Print_Halt;
              x[f]:=0;
            end;
      end;
end;
```

Здесь:

i, j — строка и столбец позиции, на которую подбираем пазл;

f — номер текущего пазла;

p — описание текущего пазла;

E — обязательные буквы пазла (левая и верхняя стороны);

k — номер положения пазла (основное + 3 поворота).

Если пазл подходит, то присваиваем:

$a[i,j]$ — номер пазла, который подошел на позицию i,j;

$b[i,j]$ — номер положения, в котором пазл подошел;

$L[i,j+1]$ — как определился символ левой стороны следующей позиции в этой строке;

$U[i+1,j]$ — как определился символ верхней стороны следующей позиции в этой строке.

Next(v,u) — функция, которая возвращает истину, если новая позиция (v, u) находится в пределах фигуры, и ложь, если последняя фигура поставлена в позицию (R, C), в этом случае вызывается процедура *Print_Halt*, которая выводит ответ в заданном формате и завершает выполнение программы.

Полный текст программы представлен ниже:

```
const
  MaxRC = 100;
var
  L,U         : array [1..10,1..10] of char;
  z           : array [1..MaxRC] of string[11];
  x           : array [1..MaxRC] of longint;
  a,b         : array [1..10,1..10] of longint;
  R,C,RC,i,j,d,n: longint;
  s,p         : string;

procedure Print_Halt;
begin
  for i:=1 to R do
    for j:=1 to C do
      begin
        write(a[i,j]);
        for d:=1 to 4 do
          write(' ',z[a[i,j],b[i,j]+d]);
        writeln;
      end;
  close(output);
  halt;
end;

function Next(var i,j: longint): boolean;
begin
  inc(j);
  if j>C
  then
    begin j:=1; inc(i) end;
  Next := i<=R;
end;

procedure Rec(i,j: longint);
var
  f,k,v,w: longint;
  p,E    : string;
begin
  for f:=1 to RC do
    if x[f]=0
    then
      begin
        p:=z[f];
        E:=L[i,j]+U[i,j];
        for k:=1 to 4 do
          if (E=copy(p,k,2)) and
             (((j<C) and (p[k+2]<>'0') or
              ((j=C) and (p[k+2]='0')))) and
             (((i<R) and (p[k+3]<>'0') or
              ((i=R) and (p[k+3]='0'))))
            then
              begin
                x[f]:=1;
                a[i,j]:=f; b[i,j]:=k;
                L[i,j+1]:=p[k+2];
                U[i+1,j]:=p[k+3];
                v:=i; w:=j;
                if Next(v,w)
                  then Rec(v,w)
                  else Print_Halt;
                x[f]:=0;
              end;
      end;
end;

begin
  assign(input,'jigsaw.in'); reset(input);
  assign(output,'jigsaw.out'); rewrite(output);
  readln(R,C);
  RC:=R*C;
  for i:=1 to RC do
    begin
      readln(n,s);
      d:=length(s);
      p:=s[d-6]+s[d-4]+s[d-2]+s[d];
      z[n]:=p+p;
      x[n]:=0;
    end;
  close(input);
  for i:=1 to R do L[i,1]:='0';
  for j:=1 to C do U[1,j]:='0';
  Rec(1,1);
end.
```

Заключение

В данной статье представлена методика изучения темы «Решение игровых задач с помощью рекурсии» с учащимися V—VIII классов, предлагающая, по мнению автора, наиболее простой вариант постепенного осознания механизма рекурсии и способа решения задач с ее помощью. Методика включает в себя последовательность задач в порядке возрастания сложности, снабженных, где необходимо, предварительными общими пояснениями и последующими полными решениями предлагаемых

задач. Серьезной технической основой является разработанная под руководством автора инструментальная система дистанционного обучения (http://dl.gsu.by), которая позволяет максимально автоматизировать процесс предъявления условий задач и проверки их решений.

Список использованных 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.
6. *Долинский М. С.* Генерация комбинаторных объектов с помощью рекурсивных процедур и функций // Информатика в школе. 2019. № 4. С. 59–63.
7. *Долинский М. С.* Решение задач рекурсивной генерацией чисел // Информатика в школе. 2021. № 1. С. 46–51.
8. *Долинский М. С.* Решение рекурсивных задач по определению // Информатика в школе. 2020. № 2. С. 60–66.
9. *Златопольский Д. М.* 1400 задач по программированию. М.: ДМК, 2019. 192 с.
10. *Луридас П.* Алгоритмы для начинающих. Теория и практика для разработчика. М.: ЭКСМО, 2018. 608 с.
11. *Рафгарден Т.* Совершенный алгоритм. Графовые алгоритмы и структуры данных. СПб.: Питер, 2020. 256 с.
12. *Рубио-Санчес М.* Введение в рекурсивное программирование. М.: ДМК Пресс, 2019. 436 с.
13. *Скиена С.* Алгоритмы. Руководство по разработке. СПб.: BHV, 2011. 720 с.
14. *Солтис М.* Введение в анализ алгоритмов. М.: ДМК, 2019. 278 с.
15. *Стивенс Р.* Алгоритмы. Теория и практическое применение. М.: ЭКСМО, 2016. 544 с.
16. *Шень А.* Программирование: теоремы и задачи. М.: МЦНМО, 2017. 320 с.
17. *Do P. T., Pham B. T., Than V. C.* Latest algorithms on particular graph classes // Olympiad in Informatics. 2020. Vol. 14. P. 21–35.
18. *Manev K.* Tasks on graphs // Olympiad in Informatics. 2008. Vol. 2. P. 90–104.
19. *Manev K., Nikolov N., Markov M.* Reconstruction of trees using metric properties // Olympiad in Informatics. 2011. Vol. 5. P. 82–91.
20. *Erdősné Németh A.* Teaching graphs for contestants in lower-secondary-school-age // Olympiad in Informatics. 2017. Vol. 11. P. 41–53.

ПРАВИЛА ДЛЯ АВТОРОВ

Уважаемые коллеги!

Статьи для публикации в журналах «Информатика и образование» и «Информатика в школе» должны отправляться в редакцию **только через электронную форму на сайте ИНФО (раздел «Авторам → Отправка статьи»)**:

http://infojournal.ru/authors/send-article/

Обращаем ваше внимание, что для отправки статьи необходимо предварительно зарегистрироваться на сайте ИНФО (или авторизоваться — для зарегистрированных пользователей).

С требованиями к оформлению представляемых для публикации материалов можно ознакомиться на сайте ИНФО в разделе «Авторам»:

http://infojournal.ru/authors/

Обратите внимание: требования к оформлению файла рукописи — **разные** для журналов «Информатика и образование» и «Информатика в школе». При подготовке файла рукописи ориентируйтесь на требования для того журнала, в который вы представляете статью. Если вы представляете рукопись в оба журнала (для публикации в одном из изданий — на усмотрение редакции), при ее оформлении следует руководствоваться требованиями к оформлению рукописи в журнал «Информатика и образование».

Дополнительную информацию можно получить в разделе **«Авторам → Часто задаваемые вопросы»**:

http://infojournal.ru/authors/faq/

а также в редакции ИНФО:

E-mail: readinfo@infojournal.ru

Телефон: (495) 140-19-86