Дмитриева Марина Валерьевна

РЕШЕНИЕ ЗАДАЧ МЕТОДОМ ПЕРЕБОРА С ВОЗВРАТОМ

Один из интересных разделов программирования посвящен решению задач из области «искусственного интеллекта». К ним относятся задачи автоматизации процесса доказательства, поиска путей в лабиринте, расстановки на шахматной доске различных фигур, реализации стратегий компьютерных игр и многие другие. Большинство из этих задач можно решать методом, который получил название метода перебора с возвратом или метода поиска с возвратом.

Поиск с возвратом – это метод перебора всех возможных решений для нахождения так называемого множества *полных решений*. Каждое *частичное решение* определяет некоторое подмножество *полных решений*.

Метод перебора с возвратом основан на том, что при поиске частичного решения многократно делается попытка расширить так называемое *текущее частичное решение*. Если расширение невозможно, то на текущем шаге поиска происходит возврат (откат) к предыдущему, более короткому текущему частичному решению и делается попытка его расширить, но уже другим способом.

Итак, сущность метода перебора с возвратом состоит в расширении найденного текущего частичного решения до тех пор, пока это возможно, а когда текущее частичное решение расширить нельзя, то при возврате пытаться сделать другой выбор по расширению частичного решения, если это возможно.

Метод перебора с возвратом можно рассматривать как процесс поиска решения задачи, при котором постепенно строится и просматривается, а также при необходимости обрезается дерево подзадач. Процесс решения задачи подразделяется на отдельные подзадачи, которые часто естественным образом описываются рекурсивными методами.

Идею метода поиска с возвратом продемонстрируем на примере решения задачи поиска выхода из лабиринта

О ЛАБИРИНТАХ

Слово *лабиринт* греческое и в переводе означает ходы в подземельях. Задачи о лабиринте относятся к глубокой древности. Часто человек, попавший в лабиринт, не мог уже из него выйти, спасти человека

могло либо чудо, либо случай. Древние задачу о лабиринте считали неразрешимой.

Примером искусственных лабиринтов являются шахты рудников или так называемые катакомбы. Словом лабиринт чаще всего обозначают именно искусственное, чрезвычайно сложное сооружение, об устройстве таких лабиринтов слагались легенды. Лабиринт строится из большого числа аллей или галерей, перекрестков и тупиков, в которых можно долго блуждать в тщетных попытках выхода.

ЛЕГЕНДА О МИНОТАВРЕ

В легенде о Минотавре упоминается лабиринт, построенный на острове Крит для царя Миноса. В центре лабиринта жило чудовище Минотавр, и никто из попавших туда не мог выйти обратно, становясь, в конце концов, жертвой чудовища.

Жители Афин каждый год должны были приносить в дар Минотавру семь юношей и семь девушек. Тезей не только убил Минотавра, но и вышел из лабиринта, не заблудившись в нем, с помощью нити из клубка царевны Ариадны. Словосочетание «Нить Ариадны» трактуется как способ, дающий выход из самого безвыходного положения.

ПОИСК ПУТИ В ЛАБИРИНТЕ

Применим метод поиска с возвратом для решения задачи поиска выхода из лабиринта. Предположим, что требуется попасть из некоторого квадрата (входа) в некоторый заданный квадрат (выход) путем последовательного перемещения по соединенным смежным квадратам, которые и образуют простой лабиринт.

Множество полных решений содержит все возможные пути от входа к выходу.

Поиск с возвратом от входа к выходу осуществляет движение по лабиринту в соответствии со следующими двумя правилами:

– из текущего квадрата можно переходить в еще не исследованный соседний квадрат;

– если все соседние квадраты уже исследованы, то нужно вернуться на один квадрат назад по пройденному пути.

Первое правило говорит о том, как расширять исследуемый путь (продолжить частичное текущее решение), если это возможно. Второе правило говорит о том, как выходить из тупика.

Каждый построенный путь определяет множество всех тех возможных путей по лабиринту, начальным отрезком которого он является. Удлинение исследуемого пути «сужает» связанное с ним множество возможных путей по лабиринту. Сокращение (возврат назад) – расширяет множество возможных путей.

РЕАЛИЗАЦИЯ АЛГОРИТМА ПОИСКА ПУТИ В ЛАБИРИНТЕ

Напишем программу, которая определяет путь в лабиринте методом перебора

с возвратом. Рассмотрим самый простой вариант устройства лабиринта и простые правила перемещения по нему.

Будем считать, что лабиринт представлен матрицей, состоящей из M строк и N столбцов, содержащей нули и единицы. Если значение элемента матрицы равно нулю, то возможен переход к этой клетке, если же значение 1, то переход запрещен.

Требуется определить, существует ли путь от верхнего левого угла этой матрицы (элемента $A[1, 1]$) к правому нижнему углу (элементу $A[m, n]$). В рассматриваемом случае входом в лабиринт является элемент $A[1, 1]$, выходом – элемент $A[m, n]$.

Путь должен состоять из шагов, определяемых следующим правилом: от элемента $A[i, j]$ можно перейти к элементу $A[i + 1, j]$ или $A[i, j + 1]$, если значения этих элементов равны нулю. Таким образом, из каждой клетки возможен шаг либо вниз, либо вправо.

Задача решается просто, если M и N невелики, в этом случае все возможные пути можно просто перебрать. Например, при $M = N = 2$ существует всего два пути от элемента $A[1, 1]$ к элементу $A[2, 2]$, и легко проверить, удовлетворяют ли они всем условиям задачи.

Поскольку каждый шаг сводит задачу к той же задаче меньшей размерности, напишем рекурсивную функцию для решения задачи.

Функция должна выдавать ответ – существует ли путь, удовлетворяющий всем поставленным условиям, из элемента $A[i, j]$ в конечный пункт – элемент $A[M, N]$, и если он существует, то записать его для последующего вывода.

Если $i = N$ и $j = M$, то это означает, что искомый путь уже найден, в противном случае надо попробовать сделать один шаг в одном из двух возможных направлений, а затем (рекурсивно) проверить, существует ли путь в правый нижний угол из элемента, в котором мы окажемся после этого шага.

Если путь существует, то он должен содержать $m + n - 2$ шагов. Записывать путь будем в одномерном массиве P. Для каждого удачно выполненного шага следует указать направление, в котором этот шаг сделан. Такой информацией могут служить значения скалярного типа.

Опишем рекурсивную функцию **`SearchPath`**, проверяющую, существует ли путь от элемента $A[i, j]$ к элементу $A[m, n]$, то есть к выходу.

Функция зависит от двух параметров i и j, определяющих место в лабиринте, до которого путь уже построен. Результатом работы функции является значение **`true`**, если путь из точки (i, j) найден, и **`false`**, если пути из этой точки найти не удалось.

Тривиальный случай будет иметь место тогда, когда значение i равно M, значение j равно N. Это означает, что искомый путь уже найден, мы находимся в точке выхода из лабиринта.

Если шаг из точки (i, j) сделать нельзя, а точка (M, N) еще не достигнута, то пути из этой точки не существует. Требуется сделать шаг назад, то есть вернуться к точке, из которой попали в точку (i, j).

При поиске пути сначала делается шаг вниз (если это возможно) и проверяется, существует ли путь из этой новой точки до точки выхода из лабиринта. Длина искомого пути уже на единицу меньше. Если путь из этой точки найден, то требуется запомнить сделанный шаг (вниз), и задача решена.

Если же после шага вниз требуемый путь не был найден, то можно попытаться сделать шаг вправо. Опять, как и в предыдущем случае, длина искомого пути будет на единицу меньше. Если путь найден, то шаг (вправо) следует запомнить. Последовательность выполнения шагов безразлична. Можно сначала пытаться делать шаг вправо, а потом, если путь найти не удалось, делать шаг вниз. Найденные пути, естественно, могут отличаться.

Если же путь не найден, то надо вернуться к той точке, из которой пришли в точку $A[i, j]$, и продолжить поиск. Описание функции **`SearchPath`** приведено в листинге 1.

Функция **`posdown(i,j)`** определяет, возможен ли шаг вниз из точки (i, j), и выдает значение **`true`**, если шаг сделать можно. Функция **`posright (i,j)`** определяет, возможен ли шаг вправо. Для работы программы надо предусмотреть ввод реального размера лабиринта и сам лабиринт.

Для определения, существует ли путь из точки (1,1) следует выполнить вызов функции **`SearchPath (1,1)`**. Если путь существует, то требуется его вывести, если путь не найден, то выдать сообщение об этом. В листинге 2 приведен результат работы программы для небольшого лабиринта, задаваемого пользователем.

Задача. Движение в лабиринте по четырем направлениям

Усложним предыдущую задачу. Пусть движение в лабиринте разрешено по четы-

Листинг 1. Рекурсивная процедура поиска пути из точки (*i*, *j*) в точку (*m*, *n*)

```
function SearchPath (i,j: integer): boolean;
  var sp: boolean;
begin if (i=M) and (j=N)
      then  {конечная точка пути, выход из лабиринта}
        SearchPath := true
      else
        Begin sp:= false;
              if posdown(i,j)
              then {возможен шаг вниз}
                begin sp :=  SearchPath(i+1,j);
                      if sp then P[i+j-1] := down
                end;
              if not sp and  posright (i,j)
              then {возможен шаг вправо}
                begin sp :=  SearchPath(i,j+1);
                      if sp
                      then { после шага вправо путь найден}
                        P[i+j-1] := right
                end;
                SearchPath := sp
        end
end;
```

рем направлениям: на восток, на юг, на запад и на север, в отличие от предыдущей задачи, когда движение разрешалось только на восток и на юг. Кроме того, пользователь может задавать точку входа в лабиринт и точку выхода из него. В предыдущей задаче эти точки были зафиксированы заранее. Требуется определить, существует ли путь в лабиринте от входа к выходу.

Для решения этой задачи можно использовать алгоритм перебора с возвратом: если возможно, то выполняется шаг по одному из четырех направлений. Шаг в данном направлении возможен, если не выходим за границы поля, если клетка, куда пытаемся шагнуть, проходима, и если в ней еще не были при предыдущих попытках поиска пути. Следует быть внимательным при поиске пути, чтобы не возникло ситуации «движения по кругу» или зацикливания, в результате которого, естественно, выход найти невозможно.

Далее делается попытка найти путь из новой точки. Если это сделать удалось, то задача решена. Для дальнейшего восстановления маршрута движения надо запомнить, как попали в клетку, то есть откуда в нее пришли.

Если путь из новой точки не найден, то осуществляется возврат на шаг назад и выполняется шаг в другом направлении.

Если из заданной точки сделали четыре попытки продолжить путь, и все они оказались неудачными, то объявляем клетку, в которой находимся, свободной. Возвращаемся в ту клетку, из которой попали

Листинг 2. Результат работы программы для заданного лабиринта

```
Исходный лабиринт:
   0   0   0   1   0
   1   0   0   1   1
   1   1   0   0   0
   0   0   0   1   1
   1   1   0   0   0
Найденный путь в лабиринте по шагам:
вправо
вниз
вправо
вниз
вниз
вниз
вправо
вправо
```

в рассматриваемую клетку, осуществляя возврат (откат).

Порядок выполнения шагов может быть произвольным, но найденный путь, естественно, зависит от порядка рассмотрения альтернатив продолжения пути. Договоримся о структурах данных. Данные о лабиринте хранятся в текстовом файле. В первой строке файла задаются размеры лабиринта: число строк и число столбцов. Со второй строки задается собственно лабиринт. Символ '0' обозначает проходимую клетку, символ '1' соответствует точке входа в лабиринт, символ '2' соответствует точке выхода из лабиринта. Любой другой символ обозначает непроходимую клетку. В листинге 3 представлен файл с данными о лабиринте.

Листинг 3. Данные о лабиринте, в котором разрешено движение по 4 направлениям

```
6  6
02x100
00x00x
0x00x0
000x00
0x0000
000xx0
```

Формат выходных данных представлен в листинге 4, символом '*' помечен путь в лабиринте.

Листинг 4. Найденный путь в лабиринте при движении по 4 направлениям

```
-*#**-
**#**#
*#**#-
*-*#—
*#*——
***##-
```

При поиске пути порядок альтернатив продолжения пути был следующий: восток, юг, запад, север. Можно указать путь, длина которого меньше, чем длина пути в листинге 4. Для рассматриваемого лабиринта минимальный путь представлен в листинге 5.

Листинг 5. Путь в лабиринте минимальной длины

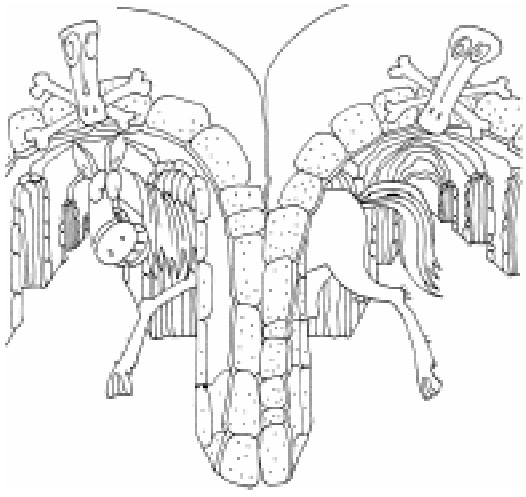
```
**#*—
*-#*-#
*#**#-
***#—
-#——
—##-
```

Задача. Нахождение пути минимальной длины

Опишите алгоритм (напишите программу) нахождения пути в лабиринте минимальной длины. Задается точка входа и точка выхода. Разрешено движение в лабиринте по четырем направлениям.

Задача. Конь и лабиринт

Рассмотрим лабиринт, содержащий $n \times n$ клеток. Как и ранее, клетка может быть «проходима» или «непроходима». По лабиринту перемещается конь, шаг коня определяется шахматными правилами. Задаются координаты точки входа в лабиринт и точки выхода из него. Требуется определить, может ли конь найти выход из лабиринта и указать путь коня, если выход найден.



Рассмотрим сначала возможные варианты решения более простых задач.

Обход конем шахматной доски

Рассмотрим доску размером $n \times n$ клеток. Конь, который ходит согласно шахматным правилам, помещается на клетку с

начальными координатами x_0 и y_0. Требуется написать программу, в результате работы которой будет осуществлен обход доски конем так, что каждая клетка будет посещаться только один раз.

Обход, если он существует, можно выполнить за $n^2 - 1$ шагов. Можно пытаться выполнить очередной ход конем или убедиться, что дальнейшее продвижение невозможно, то есть очередной ход сделать невозможно.

Доска представляется в виде двумерного массива (матрицы), содержащего n строк и n столбцов. Элементами массива будут номера ходов коня. Значение 0 будет означать, что клетка еще не посещалась, значение, равное k, будет означать, что в клетку попали на k-ом шаге.

Опишем функцию, осуществляющую обход доски конем. Функция имеет три параметра. Первый параметр i задает номер шага, второй (x) и третий (y) параметры задают текущее положение коня, то есть координаты клетки, в которой располагается конь. В результате выполнения функции выдается логическое значение **true**, если с заданной позиции удалось обойти доску полностью, и **false** в противном случае.

Требуется продолжать обход доски, если верно, что $i < n^2$. По правилам хода коня из клетки с координатами (x, y) можно сделать восемь ходов и получить текущее положение с координатами (u, v). На рисунке 1 показаны возможные ходы коня, ходы занумерованы по часовой стрелке.

Чтобы получить очередной ход следует изменить текущие координаты расположения коня на значения, хранящиеся в массивах a и b. Следующий ход коня будет получен, если выполнить операторы присваивания **u := x + a[k]**; **v := y + b[k]**.

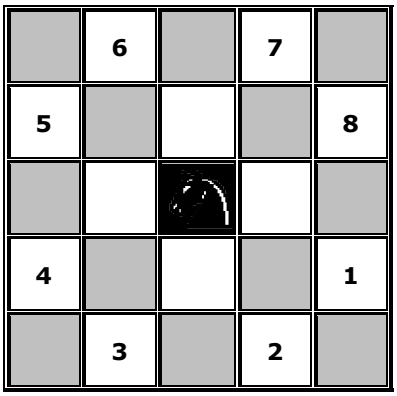


Рисунок 1. Возможные ходы коня.

Процедура **init** осуществляет формирование массивов для получения ходов коня и заполнения клеток таблицы нулевыми значениями (листинг 6).

При формировании следующей позиции коня следует проверять, не «выскочили» ли мы за пределы доски. Когда сделан следующий шаг, то осуществляется попытка обхода доски, начиная с нового шага. Если все попытки возможных ходов рассмотрены, но ни одна из них не привела к нужному решению, то осуществляется возврат (откат). Клетка объявляется свободной, и поиск решения продолжается. Функция, реализующая описанный алгоритм, приведена в листинге 7.

Если размер доски 5×5, а начало обхода осуществляется с клетки с координатами (n, n), то в результате вызова функции **kon(2,n,n)** будет сформирована матрица, содержащая номера ходов коня. В листинге 8 представлен результат работы программы.

Листинг 6. Формирование начальных значений

```
{процедура init формирует данные для хода коня и очищает доску}
   procedure init;
     begin a[1] := 2;  b[1] :=1;   a[2] := 1;    b[2] := 2;
           a[3] := -1; b[3] := 2;  a[4] := -2;   b[4] := 1;
           a[5] := -2; b[5] := -1; a[6] := -1;   b[6] := -2;
           a[7] := 1;  b[7] := -2; a[8] := 2;    b[8] :=-1;
       for i := 1 to n do
         for j := 1 to n do h[i,j] := 0
     end;
```

Листинг 7. Функция обхода конем шахматной доски

```
function kon (i: integer;  {номер хода}
              x, y: integer      {положение коня}
             ): boolean;
  var k,             {число возможных вариантов ходов коня}
      u, v: integer; {текущие координаты коня}
     q1: boolean; {результат обхода после выполнения очередного шага}
 begin k := 0;       repeat k := k+1;  {номер варианта для хода коня}
       q1 := false; {вычисление  координат следующего хода коня}
       u := x + a[k]; v := y + b[k];
       if (u >=1) and (u <= n) and (v >= 1) and (v <= n)
       then  {в пределах доски}
         if h[u,v]=0
         then    {клетку не посещали или она свободна}
           begin h[u,v] := i; {записали номер хода}
             if i = nsq
             then {всю доску обошли, посетив все клетки}
               q1 := true
             else {есть "непройденные клетки"}
               begin {обход доски со следующей позиции коня}
                  q1 := kon (i+1,u,v);
                  if not q1
                  then  {возврат на предыдущий шаг}
                    h[u,v] := 0 {клетка объявляется свободной}
               end
           end
     until q1 or (k=8);
     kon := q1
   end;
```

Листинг 8. Последовательность ходов коня при обходе доски

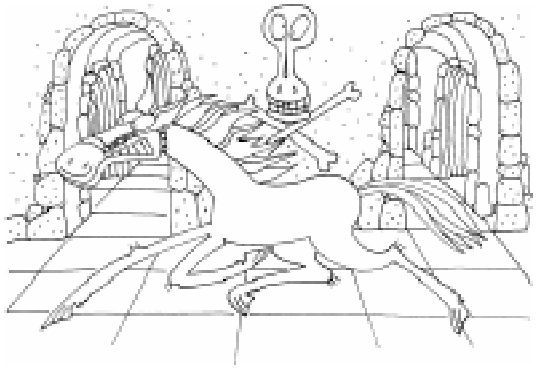
```
25 18  7 12  3
 8 13  4 17  6
21 24 19  2 11
14 9  22  5 16
23 20 15 10  1
```

Нетрудно написать программу, которая для шахматной доски размером $n \times n$, определяет те поля, начиная с которых невозможен обход доски полностью. В листинге 9 для доски размером 5×5 такие клетки помечены символом ‘1’.

Листинг 9. Клетки, начиная с которых невозможен обход конем доски полностью

```
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
1 0 1 0 1
0 1 0 1 0
```

Тем не менее, такие клетки могут рассматриваться в качестве точки входа в лабиринт, потому что при поиске пути до точки выхода не требуется осуществлять обход всей доски. Если доску рассматривать как лабиринт, то требуется следить за тем, чтобы очередной ход коня был осуществлен на клетку, которая является «проходимой». Сделаны все необходимые замечания, после которых можно приступить к решению задачи «Конь и лабиринт».



Задача. Минимальное число ходов коня к одному из выходов

Рассмотрим лабиринт, в котором задан один вход и может быть задано несколько выходов. Опишите алгоритм, позволяющий определить минимальное число ходов коня, которое требуется, чтобы попасть от точки входа в лабиринт к одному из выходов из него. Реализуйте описанный алгоритм.

ЗАДАЧИ ДЛЯ САМОСТОЯТЕЛЬНОГО РЕШЕНИЯ

Задача 1. Движение в лабиринте по четырем направлениям.

Лабиринт представлен матрицей, состоящей из *M* строк и *N* столбцов, содержащей нули и единицы. Если значение элемента матрицы равно нулю, то возможен переход к этой клетке, если же значение 1, то переход запрещен. Движение в лабиринте разрешено по четырем направлениям: на восток, на юг, на запад и на север, пользователь может задавать точку входа в лабиринт и точку выхода из него

Уровень 1. Описать алгоритм для определения, существует ли путь в лабиринте от входа к выходу.

Уровень 2. Написать программу, определяющую, существует ли путь в лабиринте от входа к выходу.

Вход показан на листинге 3.

Выход показан на листинге 4.

Уровень 3.

а) Разработать алгоритм и

б) написать программу

для нахождения в лабиринте пути минимальной длины от входа к выходу.

Задача 2. «Конь и лабиринт».

Рассмотрим лабиринт, содержащий $n \times n$ клеток. Как и ранее, клетка может быть «проходима» или «непроходима». По лабиринту перемещается конь, шаг коня определяется шахматными правилами. Задаются координаты точки входа в лабиринт и точки выхода из него. Требуется определить, может ли конь найти выход из лабиринта, и указать путь коня, если выход найден.

Уровень 1. Описать алгоритм для определения, существует ли путь конем в лабиринте от входа к выходу.

Уровень 2. Написать программу, определяющую, существует ли путь конем в лабиринте от входа к выходу, и выводящую путь коня, если выход найден.

Вход аналогичен входу, показанному на листинге 3.

Выход аналогичен выходу, показанному на листинге 8.

Уровень 3.

а) Разработать алгоритм и

б) написать программу

для нахождения в лабиринте пути конем минимальной длины от входа к выходу.

Задача 4. Визуализация алгоритмов на лабиринтах.

Уровень 3.

В задачах 1 и 2

а) нарисовать путь (использовать графику),

б) визуализировать алгоритм поиска.

Задача 5. Расстановка слонов.

Требуется найти такую расстановку 8 слонов на шахматной доске, при которой каждое поле будет находиться под ударом одного из них. Первые три фигуры расставляет пользователь.

Уровень 1. Описать алгоритм для нахождения такой расстановки.

Уровень 2. Написать программу нахождения такой расстановки.

Вход аналогичен входу, показанному на листинге 9.

Выход аналогичен выходу, показанному на листинге 9.

Задача 6. Расстановка ладей.

На шахматной доске поставить 8 ладей так, чтобы ни одна из них не угрожала другой. Первые три фигуры расставляет пользователь.

Уровень 1. Описать алгоритм для нахождения такой расстановки.

Уровень 2. Написать программу нахождения такой расстановки.

Вход аналогичен входу, показанному на листинге 9.

Выход аналогичен выходу, показанному на листинге 9.

Задача 7. Расстановка восьми ферзей.

На шахматной доске поставить 8 ферзей так, чтобы ни один из них не угрожал другому. Пользователю разрешается поставить одного ферзя на одно из полей *a*1–*a*8.

Уровень 1. Описать алгоритм для нахождения такой расстановки.

Уровень 2. Написать программу нахождения такой расстановки.

Вход аналогичен входу, показанному на листинге 9.

Выход аналогичен выходу, показанному на листинге 9.

Уровень 3. Написать программу, которая находит все варианты расстановки 8 ферзей на шахматной доске.

Задача 8. Расстановка пяти ферзей.

На шахматной доске найти такую расстановку 5 ферзей, при которой каждое поле будет находиться под ударом. На шахматную доску пользователю разрешается поместить одну фигуру.

Уровень 1. Описать алгоритм для нахождения такой расстановки.

Уровень 2. Написать программу нахождения такой расстановки.

Вход аналогичен входу, показанному на листинге 9.

Выход аналогичен выходу, показанному на листинге 9.

Задача 9. Расстановка двенадцати коней.

Расставить на шахматной доске 12 коней таким образом, что каждое поле шахматной доски находится под ударом одного из коней.

Уровень 1. Описать алгоритм для нахождения такой расстановки.

Уровень 2. Написать программу нахождения такой расстановки.

Вход аналогичен входу, показанному на листинге 9.

Выход аналогичен выходу, показанному на листинге 9.

Задача 10. Визуализация алгоритмов на шахматной доске.

Уровень 3.

В задачах 5–9 визуализировать расстановки фигур, применяя для вывода графику.

Литература

1. *Дейкстра Э.* Дисциплина программирования: Пер. с англ. М., 1978.
2. *Вирт Н.* Алгоритмы + структуры данных = программы: Пер. с анг. М.: Мир, 1985.
3. *Дмитриева М.В., Кубенский А.А.* Элементы современного программирования. СПб., 1991.
4. *Скиена С.С., Ревилла* М.А. Олимпиадные задачи по программированию. Руководство по подготовке к соревнованиям/ Пер. с англ. М.: КУДИЦ-ОБРАЗ, 2005.

Дмитриева Марина Валерьевна, доцент кафедры информатики математико-механического факультета Санкт-Петербургского государственного университета.