Обучение приемам рекурсивного программирования: задача о шахматном коне

Морозов В. В.

Умение разрабатывать рекурсивные процедуры – важное и необходимое умение для участника олимпиады по программированию. Эта область программирования, несомненно, является одной из самых сложных. Вместе с тем, для успешного выступления на олимпиаде ученику необходимо «набить руку» на решении достаточного количества подобных задач. Одной из таких задач является классическая задача об обходе шахматной доски конем. Вот ее классическая формулировка[1]: «Обойти конем все поля шахматной доски, посетив каждое из них ровно 1 раз».

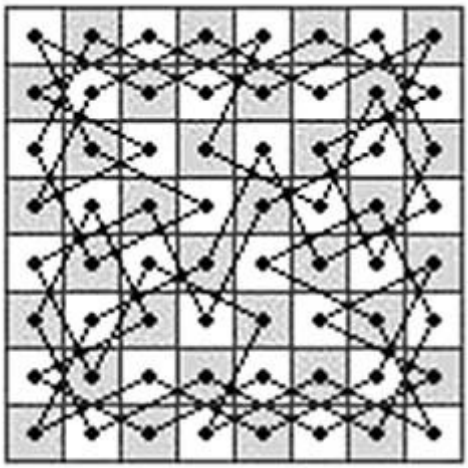

Рисунок 1

Эта задача была упомянута еще Леонардом Эйлером[2].

Постановка задачи. Написать компьютерную программу, которая находит хотя бы один из способов обхода шахматной доски конем, побывав в каждом поле не более одного раза.

Перед началом работы над составлением алгоритма решения на компьютере задачи о шахматном коне ученику следует предложить в качестве несложного упражнения разработку простых алгоритмов о шахматном коне, например:

- написать процедуру, которая в качестве входных формальных параметров будет получать координаты шахматного коня (например, строку ‘B7’), а на выходе выдает координаты полей, которые бьются этим конем;
- написать процедуру, которая в качестве входных данных будет получать координаты шахматного коня (например, строку ‘G2’), а на выходе выдает количество полей, которые бьются этим конем;

[1] (Гик Е. Я., 1983)
[2] (Задача о ходе коня, 2014)

- написать процедуру, которая в качестве входных данных будет получать координаты двух полей (например, строку ‘G2 E4’), а на выходе определяет, можно ли за один ход конем перейти из одного данного поля в другое.

После этого предлагаем решить задачу на шахматной доске с настоящим конем, пусть ученик увидит, что конь часто оказывается в тупике, пусть почувствует, насколько это трудно, пусть поймет ценность результата, который еще предстоит получить. Для Если для классической доски 8×8 ученик не найдет решение, а скорее всего так и случится, пусть попробует сделать это для меньших размеров доски, например 5×5.Было бы очень хорошо, если бы ученик после нескольких попыток обойти доску шахматным конем сам предположил, что эту задачу необходимо решать именно с помощью рекурсивного алгоритма.

После выполнения этих предварительных упражнений приступаем к обсуждению алгоритма.

Каким образом будем организовывать хранение данных? Предлагаем ученикам для обсуждения два варианта.

- Можно объявить матрицу размером n×n. Если в поле еще не было коня, то в соответствующей ячейке будем хранить 0, а если на k-м ходу в поле побывал конь, то в это поле впишем номер хода k. Текущую координату только что поставленного коня будем запоминать. Таким образом, когда задача будет решена, вся доска n×n будет заполнена числами от 1 до n^2, демонстрируя весь маршрут коня.
- Можно объявить массив из n^2 элементов, и в i-й элемент массива мы будем записывать координаты поля, где был конь на i-м ходу.

Первый вариант позволяет использовать двумерную матрицу как наглядную модель шахматной доски. Второй вариант менее нагляден и несколько сложнее в реализации, и не дает выигрыша в расходовании памяти. Предполагаем, что ученики выберут первый вариант организации хранения данных.

Затем обсуждаем с учениками алгоритм обхода доски конем. Будем писать рекурсивную процедуру, которая на вход получает доску с уже посещенными конем k полями, возможно, что количество посещенных полей k=0, а на выходе эта процедура должна попытаться сделать очередной k+1-й ход коня всеми доступными способами. Причем после каждого удачного хода процедура будет вызывать себя рекурсивно, и пытаться сделать конем очередной ход в очередное поле.

В случае, если коня ставить уже некуда, то текущая ветвь решения закрывается. Если в текущее поле поставлено число n^2, то найдено решение, которое запишем в файл.

Следует также обсудить с учениками, как компьютер будет выбирать, в какую клетку делать очередной ход. Ведь если конь достаточно далеко от краев доски, то следующий ход можно выбрать 8-ю способами, если на краю доски, то возможных ходов 4, а если конь в углу, то возможных ходов только два. Но процесс выбора хода коня хочется сделать универсальным, не зависимо от текущего положения фигуры. Договоримся нумеровать возможные ходы коня.

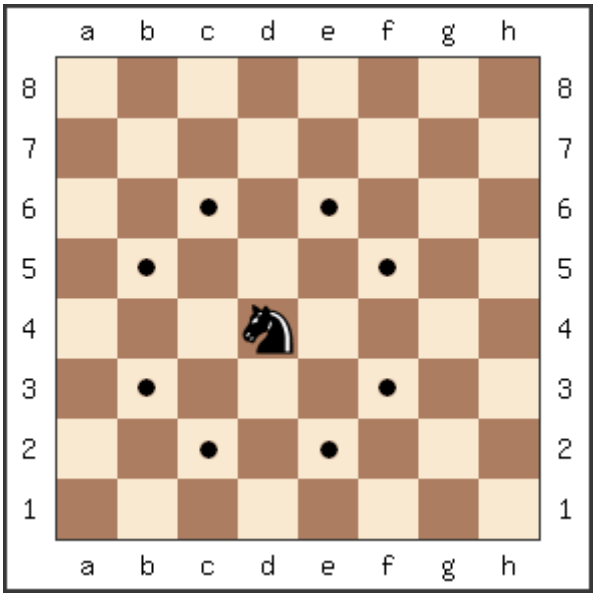

Рисунок 2

Соглашение о нумерации возможных ходов коня (пример для поля d4):

Таблица 1

№ хода	Поле
1	e6
2	f5
3	f3
4	e2
5	c2
6	b3
7	b5
8	c6

Если конь стоит близко к краю, то после некоторых ходов он может выскочить за пределы доски, поэтому мы напишем логическую функцию, которая будет определять, можно ли делать ход с данным номером, и чтобы конь оставался на доске, не выходил за пределы доски.

Итак, начинаем реализовывать предлагаемый алгоритм.

Таблица 2

`Program knight;`
`uses crt;`
`const n=8; m=n*n;`
	Объявляем константу n – размер доски.

`Type TBoard=^array[1..n,1..n] of integer;`	Поскольку мы собираемся передавать рекурсивной процедуре матрицу – объявляем для этого тип TBoard. Объясним учащимся, что поскольку массив будет передаваться как аргумент процедуре, то целесообразно использовать динамические переменные. TBoard – это не матрица, а указатель на матрицу[3]. Вместо передачи процедуре n^2 элементов массива мы будем передавать только адрес этого массива, указатель на массив. Такой подход позволит избежать переполнения стека при больших n.
`var p:TBoard; k:integer; L:boolean;` `f:text;`	Объявляем глобальные переменные: p – шахматная доска с расставленными ферзями. k – номер очередного хода коня. L – логическая переменная, флаг, признак того, что конь посетил еще не все поля. f – переменная для вывода данных в текстовый файл.

`Function Correct(x,y,i:integer):boolean;`
`begin`
`case i of`
`1: begin inc(x); dec(y,2) end;`
`2: begin inc(x,2); dec(y) end;`
`3: begin inc(x,2); inc(y) end;`
`4: begin inc(x); inc(y,2) end;`
`5: begin dec(x); inc(y,2) end;`
`6: begin dec(x,2); inc(y) end;`
`7: begin dec(x,2); dec(y) end;`
`8: begin dec(x); dec(y,2) end;`
`end;`
`correct:=(x>0)and(y>0)and(x<=n)and(`
	Нам потребуется вспомогательная функция Correct, которая на вход принимает текущее поле, где находится конь, и номер хода[4]. Функция будет возвращать логическое значение: остается ли конь на доске, если сделает ход с данным номером.

[3] (Грогоно, 1982)

[4] См. соглашение о нумерации ходов коня, Таблица 1, Рисунок 2.

y<=n) end;	

Procedure PrintBoard(p:TBoard);
  var i,j:integer;
  begin
  for i:=1 to n do
  begin
  for j:=1 to n do
  write(f,p^[i,j]:4);
  writeln(f);
  end;
  end;
	Для записи заполненной матрицы с номерами ходов в файл напишем процедуру. На вход процедура получает указатель на матрицу. Данные построчно выводятся в файл.
	Объявляем локальные переменные i, j в качестве индексов массива.

Procedure ClearBoard(var p:TBoard);
  var i,j:integer;
  begin
  for i:=1 to n do
  for j:=1 to n do
  p^[i,j]:=0;
  end;
	Напишем процедуру очистки шахматной доски. Эту процедуру придется вызывать всякий раз, когда конь окажется в тупике, и не все поля еще удалось посетить. На вход процедура получает указатель на матрицу.
	Объявляем локальные переменные i, j в качестве индексов массива.
	В цикле зануляем все элементы массива.

procedure move(var p:TBoard; x,y:integer; var L:boolean; var k:integer);	Итак, начинаем писать рекурсивную процедуру, которая делает очередной ход конем. В качестве параметров процедура получает указатель на доску p, в которой уже, возможно, сделаны несколько ходов, x, y – координаты текущего поля, логическую переменную L – признак того, что у коня еще есть ходы, номер хода k, именно это число будем записывать в поле, куда сделан очередной ход.

var i,xn,yn,k0:integer;
  MiniL:boolean;
  q:TBoard;
	Объявляем локальные переменные:
	i – переменная для хранения номера направления хода коня[5];
	xn, yn – координаты нового поля, куда сделана попытка хода;
	k0 – переменная для запоминания номера последнего удачного хода, который не привел коня в тупик;
	MiniL – вспомогательная логическая переменная;

[5] См. Таблица 1, Рисунок 2.

	q – указатель на резервную копию удачно сделанных ходов.
begin q:=p;k0:=k;	Запомнили состояние доски и номер текущего хода для отката, если ход окажется неудачным.

if L
  then
  begin
  if p^[x,y]=0
  then
  p^[x,y]:=k;
	Если не все ходы сделаны, если поле, куда конь собирается пойти, чисто, то в текущее поле записываем номер хода.

if (k>=m)
  then
  begin
  printboard(p);
  close(f);
  halt;
  end
	Если номер записанного хода достиг n^2, то задача решена, сделан последний ход. В этом случае выводим построенный маршрут движения коня в файл и заканчиваем программу[6].

else
  begin
  MiniL:=false;
  for i:=1 to 8 do
  if correct(x,y,i)
  then
	В противном случае, если еще не все ходы конем сделаны, то «оглядываемся», пытаемся сделать все допустимые ходы. Допустимость хода проверяем описанной выше логической функцией Correct. Заметим, что если изменить порядок выбора номера направления хода коня, то, вообще говоря, может получиться другое решение задачи. Например, цикл можно выполнить в обратном порядке:
	for i:=8 downto 1 do
	Или можно переназначить номер направления хода коня[7]. В результате стоит ожидать получение другого способа обхода шахматной доски конем.

begin
  case i of
  1: begin xn:=x+1; yn:=y-2 end;
  2: begin xn:=x+2; yn:=y-1 end;
  3: begin xn:=x+2; yn:=y+1 end;
  4: begin xn:=x+1; yn:=y+2 end;
  5: begin xn:=x-1; yn:=y+2 end;
	В зависимости от выбранного номера направления хода коня получаем новые координаты возможного хода коня: xn, yn.

[6] Важное замечание. Напоминаем в этом месте ученикам, что для рекурсивной процедуры во избежание зацикливания необходимо предусмотреть ситуацию, когда процедура не будет себя вызывать рекурсивно. В данном случае, если сделан последний ход колнем процедура прекращает вызывать сама себя рекурсивно.

[7] См. Таблица 1, Рисунок 2.

`    6: begin xn:=x-2; yn:=y+1 end;`
`    7: begin xn:=x-2; yn:=y-1 end;`
`    8: begin xn:=x-1; yn:=y-2 end;`
`end;`	

`if p^[xn,yn]=0`
`  then`
`    begin`
`      inc(k);`
`      move(p,xn,yn,L,k);`
`      if not(L)`
`        then`
`          MiniL:=true;`
`    end`
`end;`
	Если поле для предполагаемого хода еще не посещалось конем, то:
	Увеличиваем номер хода на 1, делаем этот удачный ход.

`   if not(MiniL)`
`     then`
`       begin`
`         p:=q;`
`         k:=k0-1;`
`         p^[x,y]:=0`
`       end;`
`   end`
` end`
`   else`
`     p^[x,y]:=0;`
`end;`
	Если конь оказался в тупике, возможных ходов для коня не осталось, значит, сделанный ход неудачен. Откатываем доску к последнему удачному состоянию: уменьшаем номер хода, возвращаем доку к предыдущему сохраненному состоянию.

`begin`
`  new(p);`
`  ClearBoard(p);`
	Тело программы. Готовим указатель на доску. Очищаем доску перед началом работы.

`  L:=true;`
`  k:=1;`
`  assign(f,'h:/kon.txt');`
`  rewrite(f);`
	Устанавливаем флаг L = true, что означает, что еще не все ходы сделаны.
	Номер первого хода устанавливаем равным 1.
	Готовим файл для записи ответа.

`  move(p,1,1,L,k);` `end.`	Делаем первый ход конем. Замечание: если указать начальное поле иное, то получится другой способ обхода шахматной доски конем.

Результат работы программы при n=5.

Таблица 3

Начальная координата (1; 1)	Начальная координата (3; 3)
1 16 21 10 25 20 11 24 15 22 17 2 19 6 9 12 7 4 23 14 3 18 13 8 5	21 6 11 16 23 12 17 22 5 10 7 20 1 24 15 2 13 18 9 4 19 8 3 14 25
Начальная координата (1; 1), перебор направлений против часовой стрелки	Начальная координата (3; 3), перебор направлений против часовой стрелки.
1 16 11 6 3 10 5 2 17 12 15 22 19 4 7 20 9 24 13 18 23 14 21 8 25	19 8 3 14 25 2 13 18 9 4 7 20 1 24 15 12 17 22 5 10 21 6 11 16 23

Результат работы программы при n=6.

Таблица 4

Начальная координата (1; 1)	Начальная координата (3; 4)
1 34 29 18 31 36 28 19 32 35 22 17 33 2 21 30 11 6 20 27 12 7 16 23 3 8 25 14 5 10 26 13 4 9 24 15	18 29 20 33 16 31 21 10 17 30 23 34 28 19 22 1 32 15 9 2 11 24 35 6 12 27 4 7 14 25 3 8 13 26 5 36

Начальная координата (1; 1), перебор направлений против часовой стрелки	Начальная координата (3; 4), перебор направлений против часовой стрелки.
1 6 9 26 3 36 8 27 2 5 10 25 31 16 7 12 35 4 28 21 30 17 24 11 15 32 19 22 13 34 20 29 14 33 18 23	26 17 8 3 24 15 7 2 25 16 9 4 18 27 6 1 14 23 33 30 21 12 5 10 28 19 32 35 22 13 31 34 29 20 11 36

Результат работы программы при n=7.

```
 1  36  29  40  43  34  21
28  39  42  35  22  31  44
37   2  23  30  41  20  33
24  27  38  19  32  45  14
 3  18  25  10  15  48   7
26  11  16   5   8  13  46
17   4   9  12  47   6  49
```

Результат работы программы при n=8.

```
 1  54  39  48  59  44  31  50
38  47  56  53  32  49  60  43
55   2  33  40  45  58  51  30
34  37  46  57  52  25  42  61
 3  20  35  24  41  62  29  14
36  23  18  11  26  15   8  63
19   4  21  16   9   6  13  28
22  17  10   5  12  27  64   7
```

Список иллюстраций

Предметный указатель

Список литературы

1. Гик, Е. (2009). *Математика на шахматной доске.* Москва: Мир энциклопедий Аванта +.

2. Гик, Е. Я. (1983). *Шахматы и математика.* Москва: Наука.

3. Грогоно, П. (1982). *Программирование не языке Паскаль.* Москва: Мир.

4. *Задача о восьми ферзях.* (13 1 2014 г.). Получено 15 9 2014 г., из https://ru.wikipedia.org: https://ru.wikipedia.org/w/index.php?title=%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D0%B2%D0%BE%D1%81%D1%8C%D0%BC%D0%B8_%D1%84%D0%B5%D1%80%D0%B7%D1%8F%D1%85&stable=1

5. *Задача о ходе коня*. (5 8 2014 г.). Получено 17 9 2014 г., из Википедия: https://ru.wikipedia.org/wiki/%C7%E0%E4%E0%F7%E0_%EE_%F5%EE%E4%E5_%EA%EE%ED%FF