С авторами можно связаться по адресу: mail@avrorasystems.com, "для Шалыто".

Статья опубликована в журнале "Мир ПК", 2003. №1. С.152–155.

Задача о ходе коня

Анатолий Шалыто, Никита Туккель, Никита Шамгунов

В статье рассматривается классическая задача о ходе коня, для решения которой разработаны итеративная, рекурсивная и автоматная программы. Приводятся результаты исследования этих программ. Построенная автоматная программа позволяет получить те же результаты, что и итеративная, но является более понятной.

Решение одного любопытного вопроса,
который, кажется, не подчиняется
никакому исследованию.

Л. Эйлер

К одному из наиболее интересных разделов программирования относятся задачи из области искусственного интеллекта, в которых решение ищется методом "проб и ошибок" [1,2]. При этом имеет место перебор при поиске решения, продолжительность которого может быть сокращена за счет применения тех или иных эвристических правил — методов оптимизации.

Класс алгоритмов, позволяющий решать подобные задачи, в англоязычной литературе называется "backtracking algorithms" ("алгоритмы с возвратом"). Такие алгоритмы применяются в тех случаях, когда не подходят более "направленные" методы [3].

Исследуем одну из наиболее известных задач этого класса — задачу о ходе коня [3–5]. Она состоит в том, чтобы найти обход доски размером N×M конем, перемещающимся по правилам шахматной игры. Если такой обход существует, то каждое поле посещается только один раз (выполняется N×M–1 шагов). Проанализируем методы оптимизации и исследуем работу итеративной, рекурсивной и автоматной программ.

Методы оптимизации

Рассмотрим следующие методы оптимизации:

– определение клеток, обход из которых невозможен (оптимизация 1);

– выявление заблокированных клеток (оптимизация 2);

– применение правила Варнсдорфа (оптимизация 3);

– использование различных массивов ходов коня (оптимизация 4).

1. Определение клеток, обход из которых невозможен

Если количество клеток доски нечетно (число белых и черных клеток отличается на единицу), то обход из некоторых клеток не существует. Отметим, что путь коня проходит по клеткам, чередующимся по цвету. Если общее число клеток доски нечетно, то первая и

последняя клетки пути, пройденного конем, будут одного и того же цвета. Таким образом, обход будет существовать только тогда, когда он начнется клеткой того цвета, который имеют наибольшее число клеток.

Выполнение этого условия проверяется следующей функцией:

```
int solution_impossible()
{
  // Если произведение сторон доски нечетно и сумма координат начальной позиции
  // нечетна, то решения не существует.
  return ((size_x*size_y) % 2 != 0) && ((start_x+start_y) % 2 != 0) ;
}
```

Однако приведенное правило не охватывает всех клеток, для которых обхода не существует. Так, для доски размером 3×7, помимо тех клеток, для которых выполняется приведенное правило, обход невозможен также из клетки (2,4).

2. Выявление заблокированных клеток

Если при очередном ходе образуется клетка, куда можно войти, но откуда нельзя выйти, то такой ход недопустим, за исключением предпоследнего в обходе. Данный метод оптимизации позволяет значительно сократить число возвратов, в том числе и при совместном использовании с правилом Варнсдорфа.

Развитием этого метода является определение групп заблокированных клеток, связанных друг с другом, но отрезанных от остальной части доски. В рассматриваемой программе определяются группы из двух заблокированных клеток, что значительно уменьшает количество возвратов для небольших досок, а при использовании вместе с правилом Варнсдорфа — и для больших (например, размером 100×100 клеток).

3. Применение правила Варнсдорфа

Среди многих эвристических методов [5], используемых для сокращения перебора, правило Варнсдорфа является наиболее простым. В соответствии с ним при обходе доски коня следует каждый раз ставить на то поле, из которого он может сделать наименьшее число ходов по еще не пройденным полям. Если таких полей несколько, то можно выбрать любое из них, что может, однако, завести коня в тупик и потребовать возврата [5]. Отметим, что наилучшим образом правило Варнсдорфа работает для угловых полей.

4. Использование различных массивов ходов коня

Ходы коня могут быть заданы, например, в виде следующего массива:

```
int possible_moves_sh[][2] = {
{-1, -2}, {-2, -1}, {-2,  1}, { 1, -2},
{-1,  2}, { 2, -1}, { 1,  2}, { 2,  1} };
```

Каждый его элемент определяет один возможный ход коня и содержит изменения его координат на доске при совершении хода. При использовании различных массивов для ходов коня количество возвратов может различаться. В программе применяются пять эвристически выбранных массивов, содержащих возможные ходы коня в различном порядке. Также задается максимальное число возвратов (GOOD_RET), и когда оно будет достигнуто, поиск пути начинается заново с использованием уже другого массива. При поиске обхода с применением последнего массива количество возвратов ограничивается значением MAX_RET. Если при совместном использовании всех предложенных методов оптимизации установить значение GOOD_RET равным единице, то для досок, близких к квадратным, можно строить обходы без единого возврата для всех клеток, из которых существует обход.

Обход без единого возврата из каждой клетки не удается получить для "вытянутых" досок (например, 14×3) и для больших досок, например для доски 100×100 клеток.

Итеративная программа

Полный перебор

Идея алгоритма для итеративной программы заключается в следующем:

– на каждом шаге ищется фрагмент пути, начинающийся из текущей клетки и не включающий уже пройденные;

– при совершении хода из массива возможных ходов извлекается очередной элемент, который приводит в незанятую клетку, находящуюся в пределах доски;

– если ход невозможен, то происходит возврат в предыдущую клетку (отмена хода);

– поиск пути считается успешным тогда, когда номер текущего хода станет равным количеству клеток на доске. Если из начальной клетки перебраны все возможные ходы, то пути не существует.

Ниже приведена структура функции, выполняющей перебор:

```
void find_path()
{
  for( move_num = 1 ; move_num <= max_moves ; )
  {
    if( make_move() )       // Сделать ход.
      move_num++ ;
    else                    // Ход невозможен.
      if( move_num > 1 )
      {
        undo_move() ;       // Отменить ход.
        move_num-- ;
      }
      else
        return ;            // Обход невозможен.
  }
}
```

Номер использованного варианта хода для каждого из ходов запоминается в массиве, размер которого выбирается в соответствии с размером доски.

Описанный алгоритм осуществляет перебор вариантов и находит решение, если оно имеется. Отсутствие решения приводит к полному перебору всех вариантов.

Для иллюстрации выполнения возвратов в табл. 1 приведен протокол обхода доски размером 3×4 из клетки с координатами (2,4). В конце протокола приведен полученный в результате обхода путь.

Таблица 1. Протокол обхода доски размером 3×4 из клетки (2,4)

```
Начальный ход (2, 4)      Отмена хода 4 из (2, 1)   Ход 3 в (1, 1)
Ход 2 в (3, 2)            Ход 4 в (3, 4)            Ход 4 в (2, 3)
Ход 3 в (1, 3)            Ход 5 в (2, 2)            Ход 5 в (3, 1)
Ход 4 в (2, 1)            Ход 6 в (1, 4)            Ход 6 в (1, 2)
Ход 5 в (3, 3)            Ход 7 в (3, 3)            Ход 7 в (3, 3)
Ход 6 в (1, 4)            Ход 8 в (1, 2)            Ход 8 в (1, 4)
Ход 7 в (2, 2)            Ход 9 в (3, 1)            Ход 9 в (2, 2)
Ход 8 в (3, 4)            Ход 10 в (2, 3)           Ход 10 в (3, 4)
Отмена хода 8 из (3, 4)   Ход 11 в (1, 1)           Ход 11 в (1, 3)
Отмена хода 7 из (2, 2)   Отмена хода 11 из (1, 1)  Ход 12 в (2, 1)
Отмена хода 6 из (1, 4)   Отмена хода 10 из (2, 3)
Ход 6 в (1, 2)            Отмена хода 9 из (3, 1)   Путь из (2, 4),
Ход 7 в (3, 1)            Отмена хода 8 из (1, 2)   возвратов 19:
Ход 8 в (2, 3)            Ход 8 в (2, 1)            3    12   5
Ход 9 в (1, 1)            Отмена хода 8 из (2, 1)
Отмена хода 9 из (1, 1)   Отмена хода 7 из (3, 3)   6    9    2
Отмена хода 8 из (2, 3)   Отмена хода 6 из (1, 4)
Отмена хода 7 из (3, 1)   Отмена хода 5 из (2, 2)   11   4    7
Отмена хода 6 из (1, 2)   Отмена хода 4 из (3, 4)
Отмена хода 5 из (3, 3)   Отмена хода 3 из (1, 3)   8    1    10
```

Для некоторых клеток программа работает чрезвычайно медленно уже при небольших размерах доски. Например, для доски 6×6 при старте из клетки (5,2) поиск пути требует более 290 миллионов возвратов.

Результаты работы программы с оптимизацией

Из работы [5] известно, что для досок размером N×M и M×N:

– не существует ни одного обхода при N, M < 3; N = 3, M = 5, 6; N = M = 4;

– имеется как минимум одна клетка, из которой возможен обход доски при N = 3, M = 4; N = 3, M ≥ 7; N ≥ 4, M ≥ 5.

Переходя к изложению полученных результатов отметим, что они относятся к перечисленным выше методам оптимизации и эвристически выбранным массивам вариантов для хода коня. При применении других методов оптимизации [5] и иных массивов получаемые результаты могут различаться.

Рассмотрим результаты обхода некоторых досок. Для них приводятся таблицы, в которых указывается количество возвратов, выполненное при нахождении обхода из соответствующей клетки. При отсутствии обхода в клетке указывается символ "N". В случае, если количество возвратов превысило задаваемую в программе величину (MAX_RET = 400000), поиск пути прекращается, а в соответствующей клетке выводится сокращение "LIM". В таблицах, построенных для метода оптимизации, использующего различные массивы вариантов хода коня, в клетке дополнительно указывается обозначение массива (от 'A' до 'E'), при применении которого обход был получен без возвратов (GOOD_RET = 1). При этом, если результат был получен для первого массива, то соответствующий ему символ 'A' не выводится.

Также в таблицах приводится один из полученных путей.

Результаты, полученные для доски размером **5×5**, приведены в табл. 2.

Таблица 2. Результаты работы программы для доски 5×5

Оптимизации 1	Оптимизации 1–3
33951 N 212310 N 4543 N 52061 N 123374 N 69216 N 23411 N 32116 N 214142 N LIM N 1300 N 325867 N 30565	0 N 127 N 0 N 0 N 0 N 127 N 0 N 0 N 0 N 0 N 0 N 127 N 0
Оптимизации 1 и 2	Оптимизации 1–4
950 N 4676 N 256 N 1596 N 3057 N 2102 N 629 N 1417 N 4296 N 7463 N 94 N 4426 N 381	0 N B0 N 0 N 0 N 0 N B0 N 0 N 0 N 0 N 0 N 0 N D0 N 0
Оптимизации 1 и 3	Путь из (5, 5), возвратов 0:
0 N 1709 N 0 N 0 N 0 N 1709 N 0 N 0 N 0 N 0 N 0 N 1709 N 0	5 14 19 24 3 20 9 4 13 18 15 6 23 2 25 10 21 8 17 12 7 16 11 22 1

Результаты, полученные для доски размером **7×7**, приведены в табл. 3.

Таблица 3. Результаты работы программы для доски 7×7

Оптимизации 1	Оптимизации 1–3
29032 N LIM N LIM N LIM N LIM N LIM N 172170 N 19396 N 19396 N LIM N LIM N 29032 N LIM N LIM N LIM N 39629 N 14807 N LIM N LIM N LIM N LIM N 6415 N 33488 N 386809 N LIM	8339 N 0 N 0 N 0 N 0 N 0 N 0 N 0 N 0 N 3 N 0 N 0 N 0 N 5972 N 0 N 14204 N 0 N 0 N 0 N 5972 N 0 N 0 N 0 N 0 N 0
Оптимизации 1 и 2	Оптимизации 1–4
1426 N 17178 N 293292 N 17178 N LIM N 302303 N 3353 N 966 N 966 N LIM N 133190 N 1426 N LIM N 28390 N 70852 N 2052 N 980 N 30757 N 9903 N LIM N 24272 N 491 N 681 N 5118 N 108858	B0 N 0 N 0 N 0 N 0 N 0 N 0 N 0 N 0 N B0 N 0 N 0 N 0 N B0 N 0 N D0 N 0 N 0 N 0 N B0 N 0 N 0 N 0 N 0 N 0
Оптимизации 1 и 3	Путь из (7, 7), возвратов 0:
291565 N 0 N 0 N 0 N 0 N 0 N 0 N 0 N 0 N 375 N 0 N 0 N 0 N LIM N 0 N LIM N 0 N 0 N 0 N LIM N 0 N 0 N 0 N 0 N 0	9 6 11 40 29 4 27 12 39 8 5 26 43 30 7 10 35 44 41 28 3 36 13 38 25 48 31 42 19 16 47 34 45 2 23 14 37 18 21 24 49 32 17 20 15 46 33 22 1

Для классической шахматной доски размером **8×8** получены следующие результаты:

– для оптимизаций 1 и 2 в большинстве клеток превышено предельное значение числа возвратов (MAX_RET). Минимальное количество возвратов в клетке (5,5) — 2502;

– для оптимизаций 1 и 3 во всех клетках, кроме (4,1) и (6,4), ноль возвратов. В указанных клетках 9 и 79 возвратов, соответственно;

– для оптимизаций 1–3 в клетке (6,4) три возврата. В остальных клетках ноль возвратов;

– для оптимизаций 1–4 во всех клетках ноль возвратов. При этом в клетке (6,4) был использован второй массив возможных ходов.

Для доски размером **9×9** для клеток, из которых существует обход, он достигается без единого возврата назад для оптимизаций 1–3 (для 1–4, естественно, тоже).

Для доски размером **50×50** получены следующие результаты:

– для оптимизаций 1–3 в большинстве клеток ноль возвратов, а количество возвратов в остальных клетках варьируется от единиц до значений, превышающих предельное значение;

– для оптимизаций 1–4 во всех клетках ноль возвратов.

Для доски размером **77×77**, для клеток, из которых существует обход, получены следующие результаты:

– для оптимизаций 1–3 в большинстве клеток ноль возвратов, а количество возвратов в остальных клетках варьируется от единиц до значений, превышающих предельное значение;

– для оптимизаций 1–4 во всех клетках ноль возвратов.

Для доски размером **100×100** получены следующие результаты:

– для оптимизаций 1–3 в большинстве клеток ноль возвратов, а количество возвратов в остальных клетках варьируется от единиц до значений, превышающих предельное значение;

– для оптимизаций 1–4 во всех клетках, за исключением клеток (94,24) и (94,79), ноль возвратов.

Как отмечено выше, рассматриваемые методы оптимизации для неквадратных досок не столь эффективны. Результаты, полученные для доски размером **14×3**, приведены в табл. 4.

Таблица 4. Результаты работы программы для доски 14×3

Оптимизации 1–4												
B0	B0	E997	E45274	B0	E0	0	B0	D0	0	E17	E626	0	0
B0	E390577	E56	0	0	C0	E16	E10378	B0	0	0	E1006	E67	0
B0	B0	E997	E45274	C0	B0	C0	B0	C0	B0	E17	E626	0	0
Путь из (14, 3), возвратов 0:													
20	17	14	11	26	23	32	9	30	7	36	39	2	5
15	12	19	22	33	10	25	28	35	40	3	6	37	42
18	21	16	13	24	27	34	31	8	29	38	41	4	1

Рекурсивная программа

Наиболее известное решение для задачи обхода конем — рекурсивное [2]. При этом структура функции, выполняющей перебор, имеет следующий вид:

```
int find_path( int cur_x, int cur_y, int move_num )
{
  desk_state[cur_x][cur_y] = move_num ; // Запомнить ход.
  if( move_num > max_moves ) return 1 ; // Проверить завершение обхода.
  // Проверить каждый возможный ход из текущей клетки.
  for( int i = 0 ; i < 8 ; i++ )
  {
    int next_x = cur_x + possible_moves[i][0] ; // Определить следующее поле.
    int next_y = cur_y + possible_moves[i][1] ;

    if(    move_possible( next_x, next_y )
        && find_path( next_x, next_y, move_num+1 )) return 1 ;
  }

  // Возврат.
  desk_state[cur_x][cur_y] = 0 ;
  back_ret++ ;
  return 0 ;
}
```

В этой программе могут быть использованы все виды оптимизаций, описанные выше.

Автоматная программа

Если первые две программы для решения этой задачи вполне традиционны [2], то автоматные к таковым не относятся.

Отметим, что автоматная программа может быть формально построена по описанной выше итеративной программе с помощью метода, изложенного в работе [7]. Автоматная программа может быть также формально построена и по приведенной рекурсивной программе с использованием метода, предложенного в работе [8].

Кроме того, можно создать автоматную программу путем непосредственного построения автомата по условиям задачи. На рис. 1, 2 для этого автомата приведены соответствующие схема связей и граф переходов автомата Мили.

Рис. 1. Схема связей автомата для задачи о ходе коня

Этот автомат использует функции и переменные, определенные в итеративной программе. Поэтому в автоматной программе применяются все рассмотренные методы оптимизации, а получаемые с ее помощью результаты совпадают с результатами работы итеративной программы.

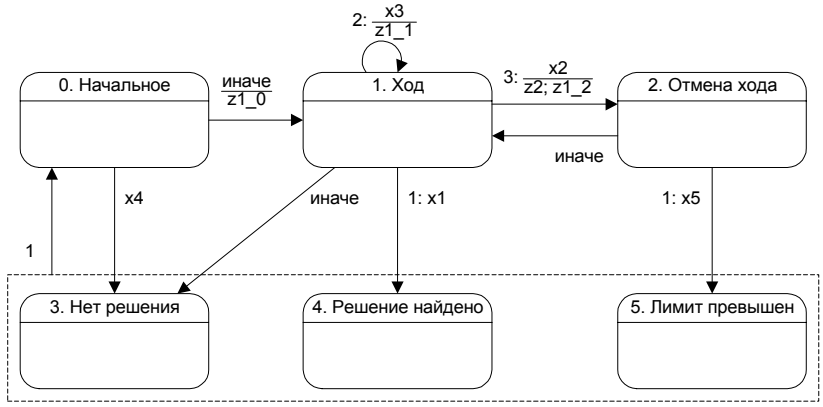

Рис. 2. Граф переходов автомата для задачи о ходе коня

Упрощенный текст функции, реализующей этот автомат, приведен ниже:

```
void find_path_switch( int limit )
{
  y = 0 ;

  do
    switch( y )
    {
      case 0:
        if( x4())                     y = 3 ;
        else
        { z1_0() ;                    y = 1 ; }
      break ;
      case 1:
        if( x1())                     y = 4 ;
        else
        if( x3()) { z1_1() ; }
        else
        if( x2()) { z2() ; z1_2() ;   y = 2 ; }
        else                          y = 3 ;
      break ;
      case 2:
        if( x5(limit))                y = 5 ;
        else
                                      y = 1 ;
      break ;
      case 3:                         y = 0 ; break ;
      case 4:                         y = 0 ; break ;
      case 5:                         y = 0 ; break ;
    }
  while( y < 3 ) ;
}
```

Заключение

Совместное применение достаточно простых и известных методов оптимизации позволило резко сократить перебор, благодаря чему удается относительно быстро находить пути в досках весьма большой размерности. Так, на компьютере, оснащенном процессором Pentium II 400-МГц, поиск обхода из каждой клетки доски размером 200×200 занял около 20 минут (на поиск одного обхода — около 0,03 с). При этом для большинства клеток обход выполняется без единого возврата назад. В программе, наряду с рассмотренными, могли бы использоваться и другие методы оптимизации [5]. Однако на досках очень большого размера, например 2000×2000 клеток, нахождение даже одного пути занимает значительное время и при применении методов оптимизации, позволяющих строить обходы без единого возврата.

Как было отмечено выше, итеративная, рекурсивная и автоматная программы позволяют получать одинаковые результаты, однако автоматная более понятна за счет явного выделения состояний.

Полный текст программы, с помощью которой выполнены эксперименты, приведен совместно с данной статьей на сайтах **http://is.ifmo.ru** и **http://www.softcraft.ru**.

Настоящая работа была выполнена на кафедре "Компьютерные технологии" Санкт-Петербургского государственного института точной механики и оптики (технического университета). Авторы благодарны студенту этой кафедры Григорьеву А.Э., который принимал участие в начале этой работы.

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту №02-07-90114 "Разработка технологии автоматного программирования".

Литература

1. Бобак И. Алгоритмы: AI-поиск //Программист. 2002. №7.

2. Вирт Н. Алгоритмы + структуры данных = программы. М.: Мир: Мир, 1985.

3. Бобак И. Алгоритмы: "возврат назад" и "разделяй и властвуй" //Программист. 2002. №3.

4. Гарднер М. Математические новеллы. М.: Мир, 1974.

5. Гик Е. Шахматы и математика. М.: Наука, 1983.

6. Шалыто А.А., Туккель Н.И. Реализация автоматов при программировании событийных систем //Программист. 2002. №4.

7. Шалыто А.А. Туккель Н.И. Реализация вычислительных алгоритмов на основе автоматного подхода //Телекоммуникации и информатизация образования. 2001. №6.

8. Шалыто А.А., Туккель Н.И., Шамгунов Н.Н. Ханойские башни и автоматы // Программист. 2002. №8.

ОБ АВТОРАХ

Шалыто Анатолий Абрамович — ученый секретарь Федерального научно-производственного центра (ФНПЦ) — Федерального государственного унитарного предприятия (ФГУП) "НПО "Аврора"", профессор кафедры "Компьютерные технологии" СПбГИТМО (ТУ). С ним можно связаться по адресу: mail@avrorasystems.com ("для Шалыто").

Туккель Никита Иосифович — инженер-программист ФНПЦ — ФГУП "НПО "Аврора"". С ним можно связаться по адресу: cynical@mail.ru.

Шамгунов Никита Назимович — трехкратный чемпион Урала по программированию, двухкратный финалист чемпионатов мира по программированию ACM, аспирант СПбГИТМО (ТУ).