Государственное образовательное учреждение высшего профессионального образования
Санкт-Петербургский национальный исследовательский университет
Информационных Технологий, Механики и Оптики
Факультет инфокоммуникационных технологий

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

По алгоритмам и структурам данных
На тему: «Рекурсия»

Проверил(а):

_____

Дата: «___»_____201__г.

Оценка: _____

Работу выполнил(а):

Никончук А.П.
Студент(ка) группы К3242
Дневного отделения

_____

2018 г.

Цель:
Получить начальные знания о рекурсивных алгоритмах
Задание:

1. Освоить синтаксис нового языка программирования;
2. Разобрать принципы работы рекурсии;
3. Реализовать рекурсию.

Задание согласно варианту:
Решить задачу о ходе коня с использованием эвристического алгоритма Варнсдорфа.

Реализация:

1. Описание принципа работы алгоритма

Задача состоит в нахождении маршрута шахматного коня, проходящего через все поля доски по одному разу. За один ход конь преодолевает две клетки по горизонтали или вертикали в зависимости от выбранного направления и ещё одну перпендикулярную текущей позиции.
Известность задачи развила множество решений нахождения, среди которых по известности выделяют решение методом Вандермонда и методом Эйлера, так как он первым обратил внимание на математические решения и написал книгу «Решение одного любопытного вопроса, который, кажется, не подчиняется никакому исследованию». Метод Эйлера состоит в том, что сначала конь двигается по произвольному маршруту, пока не исчерпает все возможные ходы. Затем оставшиеся непройденными клетки добавляются в сделанный маршрут, после специальной перестановки его элементов. Обычно алгоритм составляет генерацию всех обходов один за другим. Для ускорения обхода желательно применить метрику для упрощения обхода. Проблема в том, чтобы найти эту формулу или правило, которое позволяет нам удовлетворить критериям тура и таким образом завершить его. В качестве критериев тура, помимо основных, выставляется замкнутость решения (последняя клетка должна быть в шаге от первой). Для данных цели и задач фактор не критичен.
Метод Вандермонда решает её арифметические и аналогичен методу Эйлера, позволяющего находить маршруты коня только для досок чётной размерности. Каждый столбец и строка шахматной доски нумеруется, соответственно для фиксированного размера доски, а затем различными вариантами манипуляций доска разбивается на несколько меньших квадратов. Решения между ними соединяясь складываюся в цельное решение поиска маршрута.
Основное достоинство методов Эйлера и Вандермонда заключается в том, что они помогают нам завершить путь коня в тех случаях, когда мы двигались без всякой системы и попали в тупик - дальше идти некуда, а еще остались непройденные поля.
Алгоритм Варнсдорфа включает в себя рассмотрение валентности каждой из следующих возможных вершин (ходов рыцаря) и выбор того квадрата (если решение складывается из квадратов), который имеет наименьшую степень, так что любой квадрат, который может быть изолирован, будет использован до того, как произойдет изоляция и достижение тупика. То есть при обходе доски конь следует на то поле, с которого можно пойти на минимальное число ещё не пройденных полей. Если таких полей несколько, то можно пойти на любое из них. Второе условие составляет неточность - произвольный выбор поля может завести коня в тупик. Однако на практике вероятность попадания в тупик невелика даже при вольном пользовании второй частью правила Варнсдорфа. Метод является достаточно эффективным не только для обычной шахматной доски, но и для других досок, на которых вообще имеется решение эадачи.

2. Код программы на 2 языках программирования

2.1. Python

В первом варианте инерпретации эвристического алгоритма хода коня при выборе следующего хода происходит двойное углубление, т. е. от текущей клетки выбираются существующие на поле возможные ходы, затем от них следдующим циклом (обхода drows)

проверяется слудующий «уровень». Записывается количество ходов (count_i) — работа функции chose_move.(заполяется массив для восьми всевозможных вариантов хода коня, обозначенных в drows и dcols, в которых соответственно номер строки и столбца перемещения. Числа в массивах целые от -2 до 2. расположение и описание соответсвует одному из стандартных вариантов представлений возможных ходов.) Для следующего уровня от текущей клетки высчитывается количество возможных позиций, которые при этом свободны. Что составляет другой вариант преодоления неточности второго условия (как было определено при описании готового алгоритма, так что можно назвать его максимально логичным), который состоит в том, что если при выборе хода все имеетс одинаковая степень, то алгоритм будет смотреть дальше вниз каждый из возможных путей ходов, применяющих правило наименьшей степени, пока не будет найден успешный преемник.
Количество ходов в зависимости от положения клетки на доске 8x8 можно увидеть на рисунке 1 приложения. Выбирается клетка с наименьшим количеством аргументов. В любом из случаев недотупности клетки в массиве выставляется число 9, это позволяет при выборе индекса элемента массива с наименьшим значением выбрать 8. Алгоритм может не найти решения при заполнении массива в chose_move всеми девятками — что означало бы отмену хода, а он не предполагается.
функция knight_move2 принимает в качестве аргументов обозначенные (и в оригинальном, без эвристике) строку, столбец, тур — как двумерный массив шахматной доски, move_number — как номер хода. При старте алгоритма задаются любые существующие на поле (иначе маншрут не истинный, как и задаваемые клетки), пустой (без нулей и других обозначений) двумерный массив, сгенерированный функцией desc_create (принимает один аргумент — сторону квадрата доски) и 1 в качестве номера первого хода. В главной функции knight_move2 происходит установка на поле коня (номер хода помещается в выбранную клетку)
Таким образом алгоритму не требуется дополнительная функция отмены хода, и он в действительности или находит путь, или нет, как описывают эвристику Вандермонда.

```
def chose_move(r, c, desc):
    #create mass to save move's variants
    chosing = []
    for i in range (0, len(drows)):
        ri = r + drows[i]
        ci = c + dcols[i]
        count_i = 0
        # check existence of cell on the board
        if ri>=0 and ri<len(desc) and ci>=0 and ci<len(desc):
            if desc[ri][ci] == []:
                # deepen each option on one level and choose from them the minimum
                # increasing count_i which responds count moves on board for it "i" option
                for j in range (0, len(drows)):
                    rj = ri + drows[j]
                    cj = ci + dcols[j]
                    if rj>=0 and rj<len(desc) and cj>=0 and cj<len(desc):
                        if desc[rj][cj] == []:
                            count_i += 1
                            #print rj, cj
                chosing.append(count_i)
            else:
```

```
                # if option doesn't empty its i option is assigned 9. it number more than 8, so if among i options the smallest is 8 we will be chose right i option
                chosing.append(9)
            else:
                # if oprion doesn't exist on the board
                chosing.append(9)

    # chose minimal option

    #print 'chosing ', chosing
    choice = chosing.index(min(chosing))
    #print choice
    #print('cell [', r, ',', c, ']. best of items is ', choice, '. it is row num ', r + drows[choice], 'col num', c + dcols[choice])
    return choice

#chose_move(7, 0, desc_create(8))

def knight_move2(row, col, tour, move_number):
    # find tour using Warnsdorf's rule. The rule tells us to move to the square with the smallest integer in it

    # move knight and raise move's number
    tour[row-1][col-1].append(move_number)
    move_number += 1
    #print(row, col, move_number, '\n', tour)
    # check have you reached the finish
    if move_number >= len(tour)*len(tour[0])+1:
        #print('yeah')
        return 1
    #chose the smallest move by function chose_move
    i = chose_move(row-1, col-1, tour)
    knight_move2(row + drows[i], col + dcols[i], tour, move_number)
    return tour
```

Во втором варианте алгоритм без применения эвристики. Проходится по ветвям дерева, обходом КАКИМ. Легко достигает глубины рекурсии ЧТО ЭТО. Составляет три функции: ordinary, append_ver, remove_move.
Ordinary передается двумерный массивы tour и move_nums. Второй хранит последовательно выполненные ходы, в виде: [row, col, num_var], — где row - строка, col - столбец, num_var - выбранный вариант. Выбранный вариант записывается для смены ветки дерева, т. е. в случае необходимости ссменить ход - захода коня в тупик (в конце move_nums ставится 0), если число num_var оказывается больше 1 — происходит его уменьшение и выбор ветки соседа, в ином случае он поднимается на уровень выше, предшествующий массив - move_nums[-2].
remove_move отменяет ход удаяя последний элемент - 0, поставленный в конец move_nums, и предпоследний элемент - клетку, предварительно очистив её значение.
append_var выбирает следующий ход в зависимости от ситуации - был ли удален ход.

```
def remove_move(tour, move_nums):
    # clear the cell and delete zero element and last (dead end) move
    tour[move_nums[-2][0]][move_nums[-2][1]]=[]
    move_nums.pop(-1)
```

```
    move_nums.pop(-1)
    # take and change on 1 count of variants of last move (last since end move)
    # we viseted one branch of tree and it have the dead end
    last = move_nums[-1][-1]
    move_nums[-1].pop(-1)
    move_nums[-1].append(last-1)
    return move_nums

def append_var(row, col, tour, move_nums):
    vars = []
    # find favoraible (empty) cells almong 8 of possible and correct (it is exists on the desc)
    for i in range(len(drows)-1, -1, -1):
        r = row + drows[i]
        c = col + dcols[i]
        if r>=0 and c>=0 and r<len(tour) and c<len(tour):
            if tour[r][c]==[]:
                vars.append([r,c])

   # print(row+1, col+1, vars, '  / ', len(move_nums))

    # we check how much elements does it contein. if there are 3 (more then 2) it means that move was removed
   # that ocasion has two options
    if len(move_nums[-1])>2:
        if len(vars)>=move_nums[-1][-1]-1 and move_nums[-1][-1]<1:
            # variant in which we had only one variant and it requests to remove move. let's put zero at the end
            move_nums.append(0)
        else:
            # put the number of new chosen variant in cell of move_nums
            # there is at least one variant of moving
            move_nums.append(vars[move_nums[-1][-1]-1])
    else:
        if len(vars)<1:
            # there are no variants of moving, so we put 0 in move_nums
            move_nums.append(0)
        else:
            # put the count of variants in cell of move_nums
            # there is at least one variant of moving, so we chose the last
            move_nums[-1].append(len(vars))
            move_nums.append(vars[-1])
    return move_nums

def ordinary(tour, move_nums):
    # it is a finish if lenght of move_nums hit count of the desc
    # +1 for additional sign - zero at the end
    if len(move_nums)<len(tour)*len(tour)+1:
        # if last element is zero we need to remove last move
        if move_nums[-1]!=0:
            row = move_nums[-1][0]
            col = move_nums[-1][1]
            #print('\nmain ',row, col, len(move_nums))
            if tour[row][col]==[]:
                tour[row][col].append(len(move_nums))
            move_nums = append_var(row, col, tour, move_nums)
            #print(tour)
            #print(move_nums)
        else:
```

```
            #print('else 2 \n', move_nums, len(move_nums), '\n', tour)
            move_nums = remove_move(tour, move_nums)
            #print('else 3 \n', move_nums, '\n', tour)
        #if len(move_nums)==len(tour)*len(tour):
            #print (tour)
        ordinary(tour, move_nums)
        #return tour
    return tour
```

2.2. Java

Усовершенствованный вариант эвристического алгоритма Варнсдорфа. Выделена отдельная функция для дополнительной проверки вариантов с одинаковым количеством ходов. Праверка в главном цикле функции, отсчет количества шагов с наибольшего, т. е. в случае доски 8x8 с 64 до 0, соответственно > 0. Интерпретация клеток допустимых шагов для проверки состоит из 8 отдельных массивов: {row, col}.

```
public static ArrayList<int[]> Warnsdorf(int[] comb, int horses_num, int[][] desc,
ArrayList<int[]> horseQueue)
{
    System.out.println("\n"+"Warnsdorf metod "+comb[0]+" "+comb[1]);
    horses_num++;
    desc[comb[0]][comb[1]] = horses_num;
    System.out.println("horses_num="+horses_num);
    outDesc(desc);
        horseQueue.add(new int[]{comb[0], comb[1], enstablishValue(comb[0], comb[1],
desc)});
    if (horses_num >= desc.length*desc.length) {
        return horseQueue;
    }
    int[] nextcomb = choseComb(comb[0], comb[1], desc, horses_num);
    //System.out.println("\nin Warnsdorf row="+comb[0]+" col="+comb[1]);
    //outComb(nextcomb);
    Warnsdorf(nextcomb, horses_num, desc, horseQueue);
    return horseQueue;
}
public static int[] choseComb(int row, int col, int[][] desc, int horses_num)
{
    int[][] vars = new int[8][3];
    int[] forChoseMn = new int[8];
    for (int i=0; i<steps.length; i++){
        int r = row + steps[i][0];
        int c = col + steps[i][1];
        System.out.print("\nchoseComb "+"r="+r+" c="+c);
        if (r>=0 && r<desc.length && c>=0 && c<desc.length){
            System.out.print("  exists \n");
            if (desc[r][c]==0){
                int count_i = enstablishValue(r, c, desc);
                if (horses_num==desc.length*desc.length-1){
                    forChoseMn[i] = r;
                } else {
                    forChoseMn[i] = (count_i);
                }
                vars[i] =  new int[]{r, c, count_i};
            }
        }
        //else {
            //vars[i] = new int[]{0, 0, 0};
            //forChoseMn[i] = 0;
        //}
    }
    System.out.print("\nvars");
    outDesc(vars);
    System.out.print("vars\n");
```

```
    outComb(forChoseMn);
    int mn = mn(forChoseMn);
    int index = takeIndex(forChoseMn, mn);
    return vars[index];
}
public static int enstablishValue(int row, int col, int[][] desc)
{
    int count_i = 0;
    System.out.print("step0: "+steps[0][0]+steps[0][1]);
    System.out.print("\tstep1: "+steps[1][0]+steps[1][1]);
    System.out.print("\tstep2: "+steps[2][0]+steps[2][1]);
    System.out.print("\tstep3: "+steps[3][0]+steps[3][1]);
    System.out.print("\tstep4: "+steps[4][0]+steps[4][1]);
    System.out.print("\tstep5: "+steps[5][0]+steps[5][1]);
    System.out.print("\tstep6: "+steps[6][0]+steps[6][1]);
    System.out.print("\tstep7: "+steps[7][0]+steps[7][1]);
    System.out.print("\n");
    for (int i=0; i<steps.length; i++){
        int r = row + steps[i][0];
        int c = col + steps[i][1];
        System.out.print("\n    enstablish "+"r="+r+" c="+c);
        if (r>=0 && c>=0 && r<desc.length && c<desc.length){
            if (desc[r][c] == 0){
                count_i++;
                System.out.print(" exists count_i+1="+count_i);
            }
        }
    }
    return count_i;
}
```

3. Оценка сложности алгоритма

Осноной алгоритм прохождения по всем вешинам — экспоненциальный: $O(k^N)$, где N - число клеток на доске, а k - какая-то малая константа. Работу можно усовршенствовать совместив методы некоторых эвристик, отпимизаций. Выбранная для дальнейшего продвижения вершина посредством метода Варнсдорфа имеет наименьшее количество доступных ходов. Основная функция алгоритма warnsdorf рекурсивная (k^N, она вызывает выбор следующей клетки, который зависит от позиции предыдущей. Дополнительный метод оптимизации — массив из 8 вариантов, вместо вычисления возможного хода. При каждом выборе фактически отбрасываются ветки дерева, так как алгоритм не подразумевает возвращения для отмены хода (N*8). Также выбирается первая клетка с минимальным количеством ходов. Лучший случай — когда начальная клетка угловая (имеет два варианта ходов, её труднее достать), худший — центр (4-8 вариантов). Пусть k — некоторая константа варианта ветвления. Внутри двойного цикла выбора происходит вызов функции с другим аналогично циклом из 8 вариантов и последовательно выбор минимального значения из 8 вариантов (выбор минимума можно опутить). Тогда сложность алгоритма будет порядка N*k*(8*8+8)=N*k*2^6. $k^{N+1}-1$, где k - средний фактор ветвления для доски.

4. Пример входных данных

Для доски 5x5 получаем 2 варианта обхода доски алгоритмом ordinary без эвристики и 22 варианта обхода алгоритмом knight_move2 с эвристикой.

5. Примеры результатов работы

Рисунок 1. — несколько вариантов решений задачи хода коня Python реализации

```
 3 4
knight_move2
[[24], [3], [8], [11], [16]]

[[9], [20], [15], [2], [7]]

[[4], [23], [10], [17], [12]]

[[21], [14], [19], [6], [1, 25]]

[[], [5], [22], [13], [18]]


 4 0
knight_move2
[[3], [20], [15], [10], [5]]

[[14], [9], [4], [21], [16]]

[[19], [2], [25], [6], [11]]

[[24], [13], [8], [17], [22]]

[[1], [18], [23], [12], [7]]

ordinary
[[3], [12], [7], [18], [5]]

[[20], [17], [4], [13], [8]]

[[11], [2], [19], [6], [23]]

[[16], [21], [24], [9], [14]]

[[1], [10], [15], [22], [25]]


 4 1
```

Рисунок 2. — несколько вариантов решений задачи хода коня Java реализации

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

[ 0 0 2 ] --> [ 1 2 5 ] --> [ 0 4 3 ] --> [ 1 6 3 ] --> [ 3 7 3 ] --> [ 5 6 5 ] --> [ 7 7 1 ] --> [ 6 5 5 ] -->
 [ 5 7 3 ] --> [ 7 6 2 ] --> [ 6 4 4 ] --> [ 7 2 3 ] --> [ 6 0 2 ] --> [ 4 1 5 ] --> [ 2 0 2 ] --> [ 0 1 2 ]
 --> [ 1 3 5 ] --> [ 0 5 3 ] --> [ 1 7 2 ] --> [ 2 5 4 ] --> [ 0 6 2 ] --> [ 2 7 3 ] --> [ 4 6 3 ] --> [ 6 7 2
 ] --> [ 7 5 2 ] --> [ 6 3 5 ] --> [ 7 1 2 ] --> [ 5 0 3 ] --> [ 3 1 4 ] --> [ 1 0 2 ] --> [ 0 2 3 ] --> [ 2 1
 3 ] --> [ 4 0 3 ] --> [ 5 2 3 ] --> [ 7 3 2 ] --> [ 6 1 2 ] --> [ 5 3 4 ] --> [ 7 4 3 ] --> [ 6 2 3 ] --> [ 7
 0 1 ] --> [ 5 1 3 ] --> [ 3 0 3 ] --> [ 1 1 3 ] --> [ 3 2 2 ] --> [ 4 4 2 ] --> [ 3 6 3 ] --> [ 2 4 3 ] --> [
 0 3 2 ] --> [ 1 5 3 ] --> [ 0 7 1 ] --> [ 2 6 4 ] --> [ 4 5 2 ] --> [ 3 3 2 ] --> [ 1 4 2 ] --> [ 2 2 2 ] -->
 [ 3 4 2 ] --> [ 4 2 2 ] --> [ 2 3 1 ] --> [ 3 5 3 ] --> [ 4 3 1 ] --> [ 5 5 1 ] --> [ 4 7 1 ] --> [ 6 6 1 ]
 --> [ 5 4 0 ] --> finish
Process finished with exit code 0
```

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

[ 2 4 8 ] --> [ 1 6 3 ] --> [ 0 4 3 ] --> [ 1 2 4 ] --> [ 0 0 1 ] --> [ 2 1 5 ] --> [ 0 2 3 ] --> [ 1 0 2 ] -->
 [ 3 1 4 ] --> [ 5 0 3 ] --> [ 7 1 2 ] --> [ 6 3 5 ] --> [ 7 5 3 ] --> [ 6 7 2 ] --> [ 4 6 5 ] --> [ 2 7 3 ]
 --> [ 0 6 2 ] --> [ 1 4 4 ] --> [ 2 6 5 ] --> [ 0 7 1 ] --> [ 1 5 4 ] --> [ 0 3 2 ] --> [ 1 1 3 ] --> [ 2 3 3
 ] --> [ 3 5 4 ] --> [ 4 7 2 ] --> [ 6 6 3 ] --> [ 7 4 3 ] --> [ 6 2 4 ] --> [ 7 0 1 ] --> [ 5 1 4 ] --> [ 3 0
 2 ] --> [ 4 2 3 ] --> [ 5 4 2 ] --> [ 7 3 3 ] --> [ 6 1 2 ] --> [ 4 0 2 ] --> [ 3 2 4 ] --> [ 2 0 2 ] --> [ 0
 1 2 ] --> [ 1 3 3 ] --> [ 0 5 1 ] --> [ 1 7 2 ] --> [ 2 5 3 ] --> [ 3 7 2 ] --> [ 5 6 3 ] --> [ 7 7 1 ] --> [
 6 5 3 ] --> [ 4 4 2 ] --> [ 3 6 2 ] --> [ 5 7 2 ] --> [ 7 6 2 ] --> [ 5 5 2 ] --> [ 4 3 2 ] --> [ 2 2 2 ] -->
 [ 3 4 1 ] --> [ 5 3 3 ] --> [ 4 1 2 ] --> [ 3 3 2 ] --> [ 4 5 1 ] --> [ 6 4 2 ] --> [ 5 2 1 ] --> [ 6 0 1 ]
 --> [ 7 2 0 ] --> finish
Process finished with exit code 0
```

Выводы: По ходу решения задачи хода коня эвристики Варнсдорфа был выявлен алгоритм, который совпал с существующим логическим решением — углубление при столкновении с выбором равноценных клеток шахматной доски. Хотя сутью «рыцарского обхода» является посещение всех клеток, исследование всех направлений не обязательно для достижения не обязательно. Помимо этого получен навык реализации рекурсивного алгоритма на примере задачи о ходе коня с использованием эвристического алгоритма, изучен синтаксис нового языка программирования.

Список использованной литературы:

1) http://markkeen.com/knight/index.html
2) https://intellect.ml/analizu-slozhnosti-algoritmov-70

Приложение 1

Рисунок 1. — граф ходов доски 8x8