

Алгоритмы с возвратом

Лекция 15

Интересная область программирования — задачи так называемого «искусственного интеллекта»: ищем решение не по заданным правилам вычислений, а путем проб и ошибок. Обычно процесс проб и

ошибок разделяется на отдельные задачи, и они наиболее естественно

выражаются в терминах рекурсии и требуют исследования конечного

числа подзадач.

В общем виде весь процесс можно мыслить как процесс поиска, строящий (и обрезающий) дерево подзадач.

Во многих проблемах такое дерево поиска растет очень быстро, рост

зависит от параметров задачи и часто бывает экспоненциальным.

Иногда, используя некоторые эвристики, дерево поиска удается сократить и свести затраты на вычисления к разумным пределам.

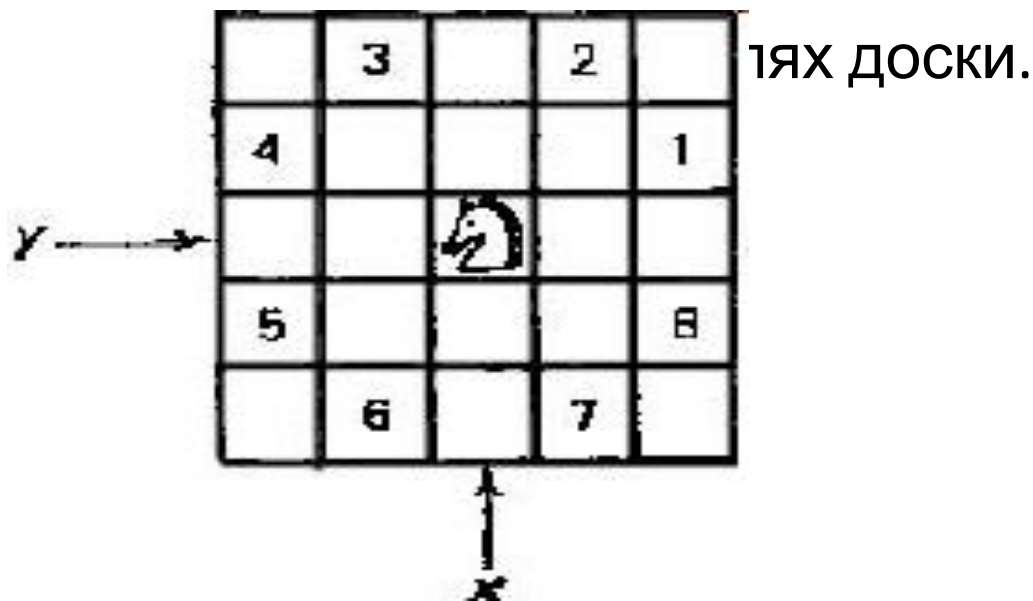
Начнем с демонстрации основных методов на хорошо известном примере — задаче о ходе коня.

Задача о ходе коня

Дана доска размером $n \times n$. Вначале на поле с координатами

(x_0, y_0) помещается конь — фигура, перемещающаяся по обычным шахматным правилам. Задача заключается в поиске последовательности ходов, при которой конь **точно**

ОДИН





Алгоритм выполнения очередного хода

```
Try(int i) {  
    инициализация выбора хода;  
    do  
        выбор очередного хода из списка возможных;  
        if (выбранный ход приемлем) {  
            запись хода;  
  
            if (доска не заполнена) {  
                Try(i+1);  
                if (неудача) отменить предыдущий ход;  
            }  
        }  
    while (неудача) && (есть другие ходы);  
}
```

Для более детального описания алгоритма нужно выбрать некоторое представление для данных. Доску можно представлять как матрицу h :

$h[x, y] = 0$ – поле (x, y) еще не посещалось

$h[x, y] = i$ – поле (x, y) посещалось на i -м ходу

Выбор параметров

Параметры должны определять начальные условия следующего хода

и результат (если ход сделан). В первом случае достаточно задавать

координаты поля (x, y) , откуда следует ход, и число i , указывающее номер хода.

Очевидно, условие «доска не заполнена» можно переписать как $i < n^2$.

Кроме того, если ввести две локальные переменные u и v для позиции

возможного хода, определяемого в соответствии с правилами хода

коня, то условие «ход приемлем» можно представить как

конъюнкцию условий, что новое поле находится в пределах доски

$(1 \leq u \leq n \ \&\& \ 1 \leq v \leq n)$ и еще не посещалось

$h[u, v] == 0$.

Отмена хода: $h[u, v] = 0$.

Введем локальную переменную $q1$ для результата, получим:

```
int Try(i, x, y) {
    int u,v; int q1;
    инициация выбора хода;

do <u, v> - координаты следующего хода;
    if ((1<=u) && (u<=n) && (1<=v) && (v<=n) && (h[u, v]==0)) {
        h[u, v]= i;
        if (i<n*n) {
            q1= Try(i+1, u, v);
            if (!q1) h[u, v]=0;
        }
        else q1 = 1;
    }
    while(!q1) && (есть другие ходы) ;
q = q1;
}
```


Выбор ходов

Полю с координатами (x_0, y_0) присваивается значение 1, остальные поля помечаются как свободные.

Если задана начальная пара координат x, y , то для следующего хода u, v существует максимально восемь возможных вариантов.

Получать u, v из x, y можно, если к последним добавлять разности между координатами, хранящиеся либо в массиве разностей, либо в двух массивах, хранящих отдельные разности.

Для фиксированного поля (x, y) количество ходов может варьироваться от двух до восьми.

Рассмотрим вспомогательную матрицу:

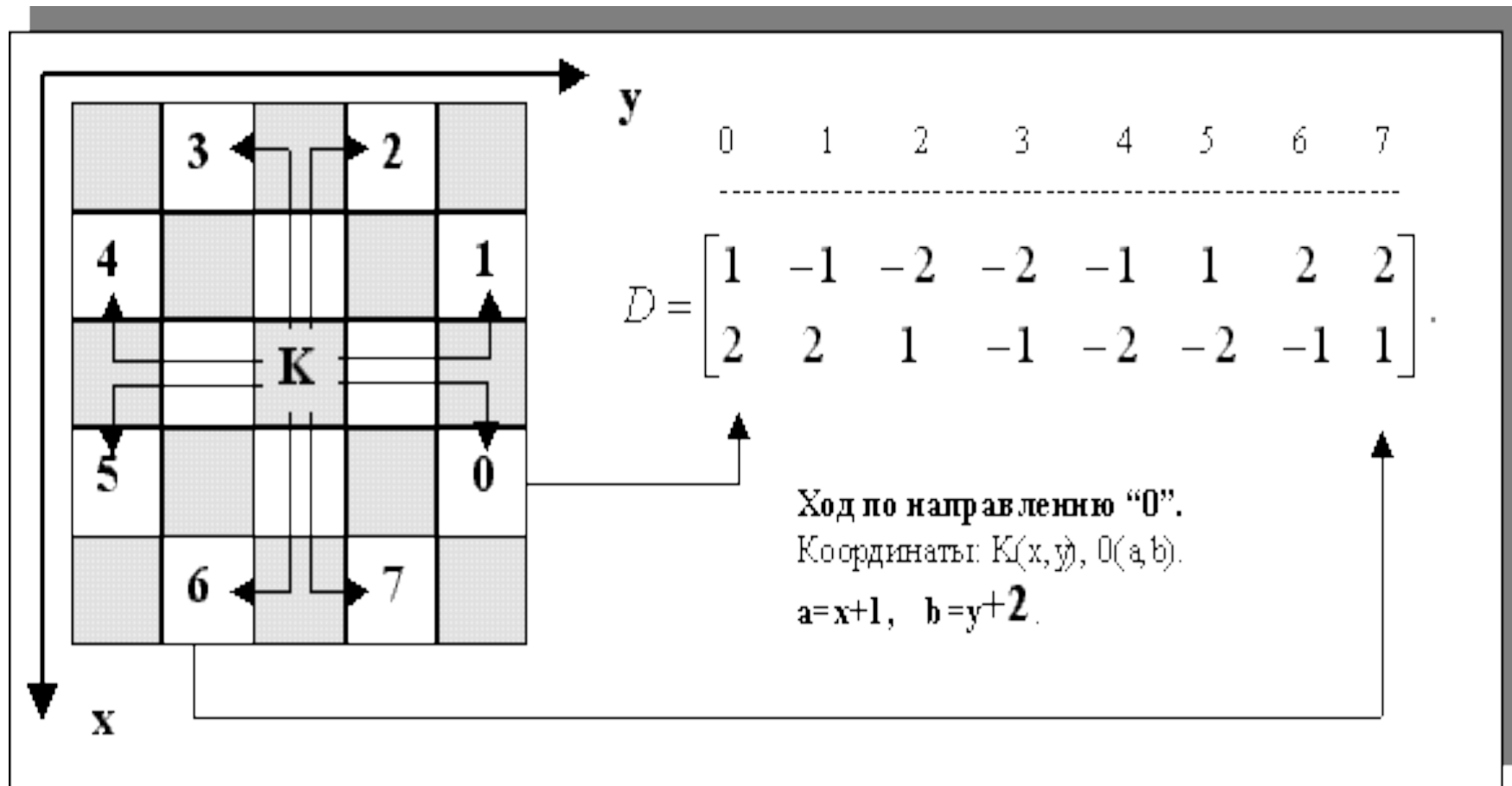
$$D = \begin{bmatrix} 1 & -1 & -2 & -2 & -1 & 1 & 2 & 2 \\ 2 & 2 & 1 & -1 & -2 & -2 & -1 & 1 \end{bmatrix}.$$

Для поля (x, y) построим последовательность ходов:

$$(x + D_{0,k}, y + D_{1,k}) \quad (k = 0, 1, \dots, 7)$$

и отберем из них те, которые не выводят за пределы поля.

На приведен фрагмент доски. Конь K стоит в позиции (x, y) . Клетки с цифрами вокруг K - это поля, на которые конь может переместиться из (x, y) за один ход.



Правило Варнсдорфа, 1823

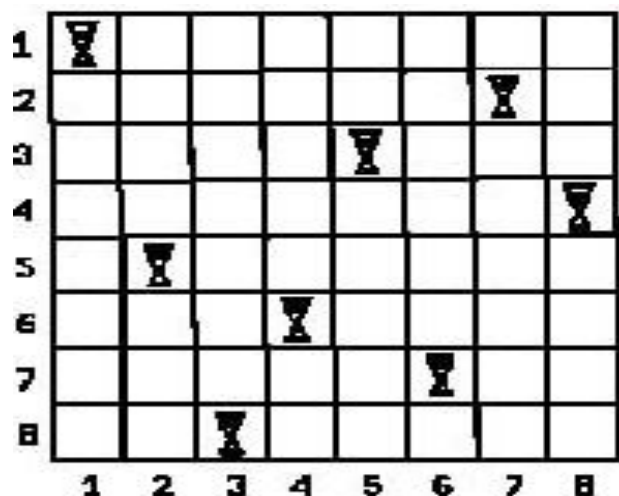
*На каждом ходу ставь коня на такое поле, из которого можно совершить наименьшее число ходов на еще не пройденные поля. Если таких полей несколько, разрешается выбирать **любое** из них.*

Долгое время не было известно, справедливо ли оно. Верно для доски от 5x5 до 76x76.

В опровержении правила Варнсдорфа для любого исходного поля доски указаны контрпримеры, построенные с помощью ЭВМ. Иными словами, с какого бы поля конь ни начал движение, следуя правилу Варнсдорфа, его можно завести в тупик до полного обхода доски.

Задача о восьми ферзях

Задача о восьми ферзях — хорошо известный пример использования методов проб и ошибок и алгоритмов с возвратами. В 1850 г. эту задачу исследовал К. Ф. Гаусс, однако полностью он ее так и не решил. Восемь ферзей нужно расставить на шахматной доске так, чтобы один ферзь не угрожал другому.



Задача о стабильных браках

Имеются два непересекающихся множества A и B . Нужно найти множество пар $\langle a, b \rangle$, таких, что $a \in A$, $b \in B$, и они удовлетворяют некоторым условиям.

Для выбора таких пар существует много различных критериев; один из них называется «правилом стабильных браков».

Пусть A — множество мужчин, а B — женщин. У каждой мужчины и

женщины есть различные предпочтения возможного партнера.

Если среди n выбранных пар существуют мужчины и женщины, не состоящие между собой в браке, но предпочитающие друг друга, а не

своих фактических супругов, то такое множество браков считается

нестабильным. Если же таких пар нет, то множество считается стабильным.

Алгоритм поиска супруги для мужчины m

Поиск ведется в порядке списка предпочтений именно этого мужчины.

```
Try(m) {
    int r;
    for (r=0; r<n; r++) {
        выбор r-ой претендентки для  $m$ ;
        if (подходит) {
            запись брака;
            if ( $m$  - не последний) Try( $m+1$ );
            else записать стабильное множество;
        }
        отменить брак;
    }
}
```

Будем использовать две матрицы, задающие предпочтительных партнеров для мужчин и женщин: *Lady* и *Man*.

ForMan [m, r] — женщина, стоящая на r -м месте в списке для мужчины m .

ForLady [w, r] — мужчина, стоящий на r -м месте в списке женщины w .

Результат — массив женщин x , где $x[m]$ соответствует партнер для мужчины m .

Для поддержания симметрии между мужчинами и женщинами для эффективности алгоритма будем использовать дополнительный

массив y : $y[w]$ — партнер для женщины w .

Предикат “подходит” можно представить в виде конъюнкции `single` и `stable`, где `stable` — функция, которую нужно еще определить.

```
Try (int m) {
    int r, w;
    for (r=0; r<n; r++) {
        w=ForMan[m, r];
        if (single[w] && stable) {
            x[m]= w; y[w]= m;
            single[w]=1;
            if (m < n) Try(m+1);
            else record set;
        }
        single[w]=0;
    }
}
```

Стабильность системы

Мы пытаемся определить возможность брака между t и w , где w стоит в списке t на r -м месте. Возможные источники неприятностей могут быть:

- 1) Может существовать женщина pw , которая для t предпочтительнее w , и для pw мужчина t предпочтительнее ее супруга.
- 2) Может существовать мужчина pt , который для w предпочтительнее t , причем для pt женщина w

1) Исследуя первый источник неприятностей, мы сравниваем ранги женщин, которых m предпочитает больше w . Мы знаем, что все эти женщины уже были выданы замуж, иначе бы выбрали ее.

```
s = 1; i = 1;
while((i < r) && s) {
    pw = ForMan[m, i];
    i := i + 1;
    if(single[pw]) {
        s = ForLady[pw, m] > ForLady[pw, y[pw]];
    }
}
```

2) Нужно проверить всех кандидатов pt , которые для w предпочтительнее «суженому». Здесь не надо проводить сравнение с мужчинами, которые еще не женаты. Нужно использовать проверку $pt < t$: все мужчины, предшествующие t , уже женаты.

Напишите проверку 2) самостоятельно!

Задача о кубике

Задано описание кубика и входная строка.

Можно ли получить входную строку, прокатив кубик?

Перенумеруем грани кубика с 123456 на 124536:

1 – нижняя;

6 – верхняя; ($1+6 = 7$)

3 – фронтальная;

4 – задняя; ($3+4 = 7$)

2 – боковая левая;

5 – боковая правая ($2+5 = 7$).

Тогда соседними для i -й будут все, кроме i -й и $(7-i)$ -й.

Попробуем построить слово, начиная со всех шести граней.

Результат (впеременной q) 1, если можно получить слово, записанное в глобальной строке w , начиная n -го символа, перекачивая кубик, лежащий g -ой гранью.

```
int chkword(g, n) {
    if((n>strlen(w)) || (w[n]== ` `))
        return 1;
    if(CB[g] != w[n]) break;
    for(i=1; i<=6; i++) {
        if((i != g) && (i+g != 7))
            q=chkwrд(i, n+1);
            if (q) return 1;
    }
}
```