

## Лекция 1

### Исчерпывающий поиск в комбинаторных алгоритмах

1. Темы лекций, лаб. и курсовой работ
2. Поиск с возвратением. Общая схема (алгоритм).
3. Задача о ферзях.
4. Оценка сложности. Метод Монте-Карло.
5. Другие способы программирования.

## Темы лекций

1. Поиск с возвратением. Задача о ферзях. Оценка методом Монте-Карло.
2. Метод ветвей и границ. Общая схема. Задача коммивояжёра (ЗК).
3. Метод ветвей и границ. ЗК (продолжение). Приближённые решения.
4. Динамическое программирование. Идея и общая схема. Оптимальное умножение матриц.
5. Динамическое программирование. Оптимальные БДП. Хорошие БДП. Аналогии.

## Продолжение

6. Графы и структуры данных. Задачи связности.
7. Остовные деревья графа. Алгоритм Краскала. Алгоритм ЯПД.
8. Непересекающиеся подмножества.
9. Обходы графа. Алгоритм Борувки для МОД.
10. МОД как приближение в ЗК. Двусвязные компоненты (применение обхода в глубину).
11. ПВГ в орграфах. Топологическая сортировка. Сильно связные компоненты.
12. Кратчайшие пути в графе (1).
13. Кратчайшие пути в графе (2).

# Лабораторные работы и курсовая работа

## Задание 1.

Алгоритмы сортировки, частичного упорядочения, хеширования.

## Задание 2.

Перебор с возвратом (Backtracking).

## Задание 3.

Метод ветвей и границ

## Задание 4.

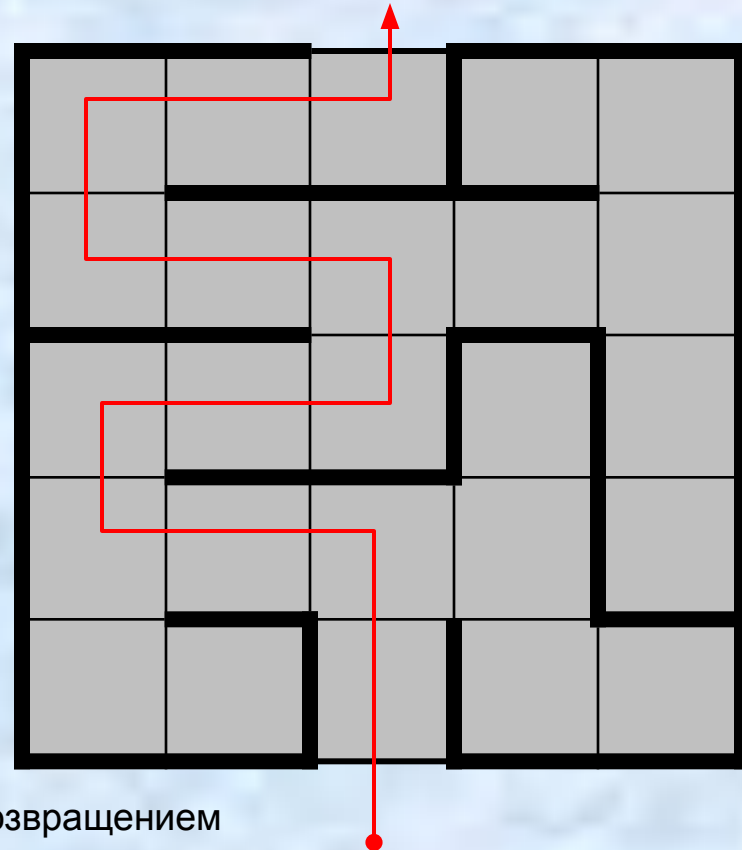
Динамическое программирование.

Курсовая работа (КР): Алгоритмы на графах.

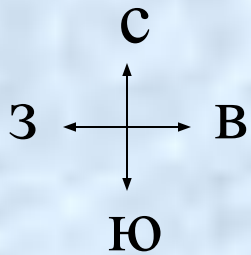
# Исчерпывающий поиск в комбинаторных алгоритмах

Поиск с возвратением =  
= Перебор с возвратом =  
= Backtracking

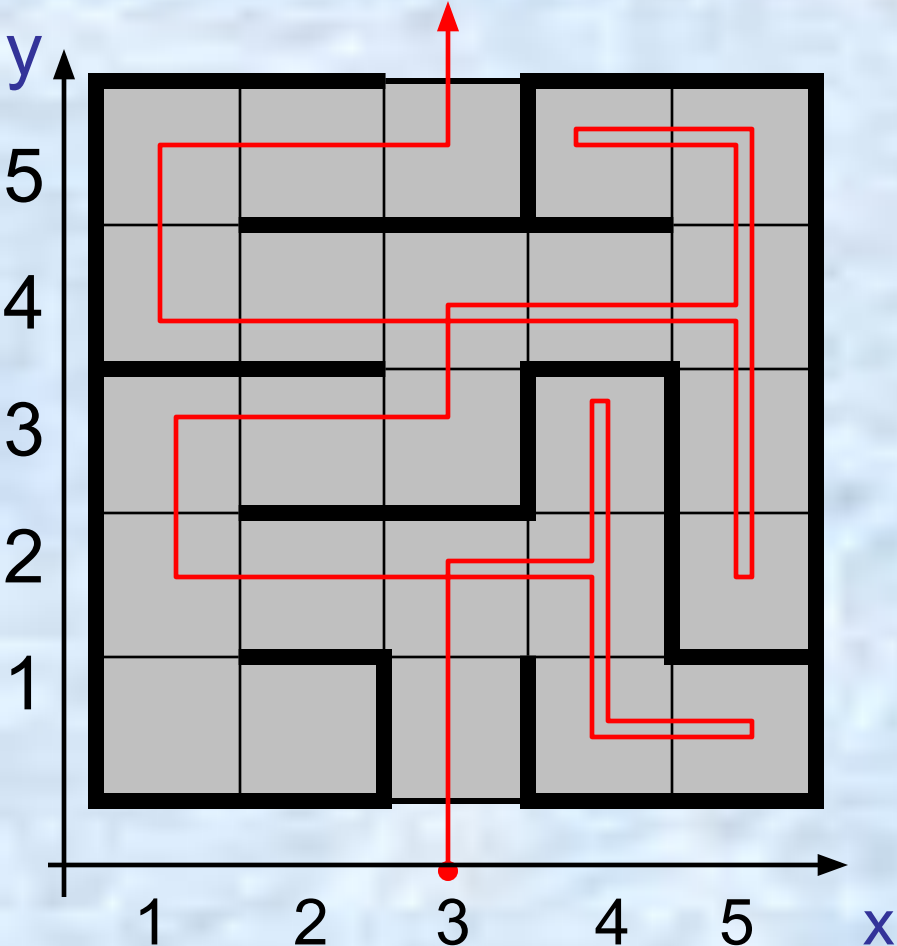
Пример.  
Поиск пути в  
лабиринте



# Стратегия поиска



С → В → Ю → З



Направление = (с, в, ю, з)

Ход = (x, y, Направление)

(3,1,с)

(3,2,с)

(4,2,в)

(4,3,с)

(4,2,ю)

(4,1,ю)

(5,1,в)

(4,1,з)

(3,2,з)

(2,2,з)

(1,2,з)

...

(x,y) - целевая клетка,  
Направление - в ц.к.

## Общий алгоритм

Решение вида  $(a_1, a_2, a_3, \dots, a_n)$

$n$  - конечно, но, вообще говоря, заранее не известно

$\forall a_i \in A_i;$

$A_i$  - конечное линейно упорядоченное множество

Исчерпываем все элементы множества  $A_1 \times A_2 \times A_3 \times \dots \times A_i$

$i = 0 \rightarrow ()$

$i = 1; S_1 \subset A_1; a_1 \in S_1; () \rightarrow (a_1)$

$i = 2; S_2 \subset A_2; a_2 \in S_2; (a_1) \rightarrow (a_1, a_2)$

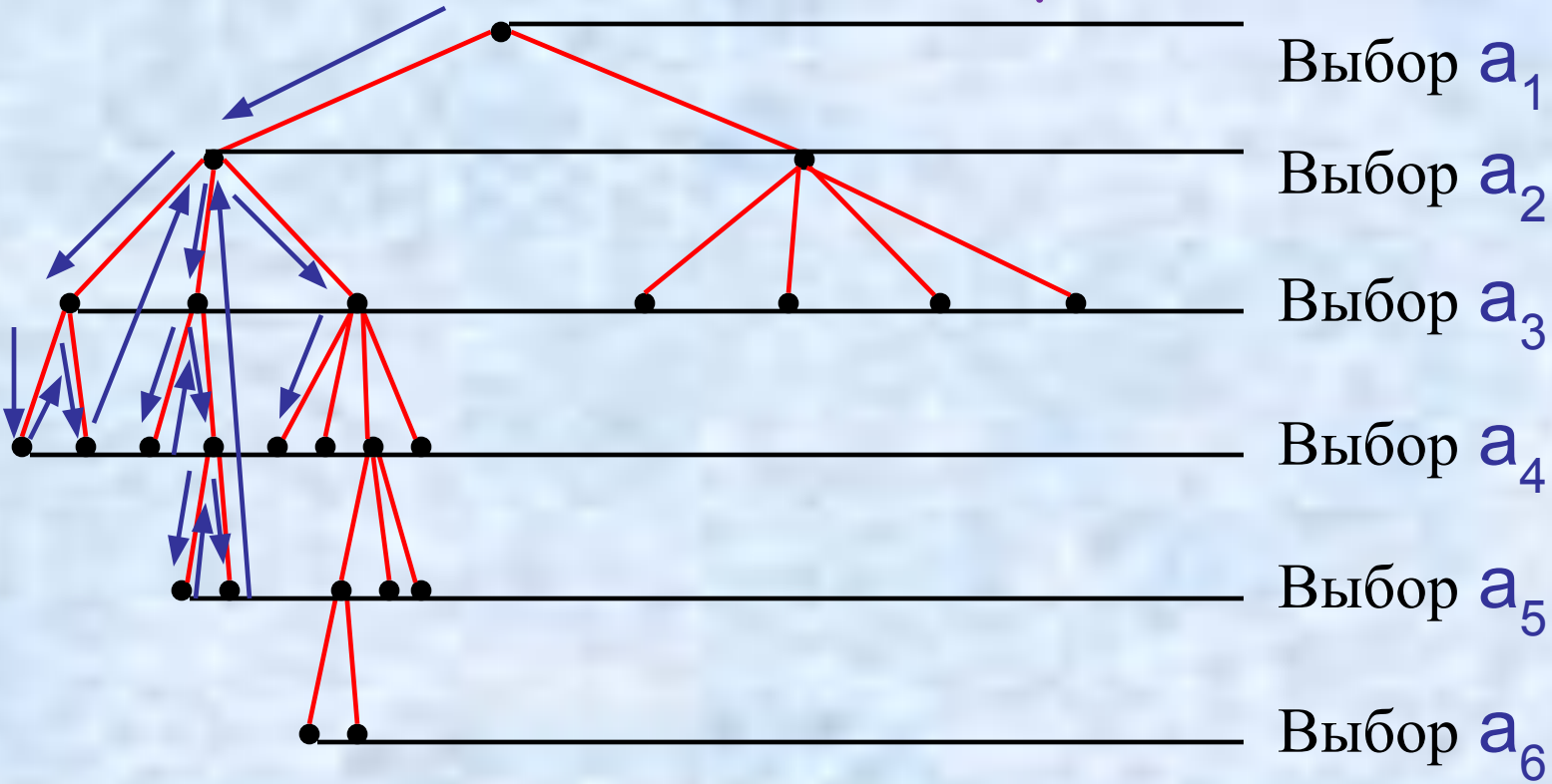
...

$i = k; S_k \subset A_k; a_k \in S_k; (a_1, \dots, a_{k-1}) \rightarrow (a_1, \dots, a_{k-1}, a_k)$

При  $S_k = \emptyset$  **backtrack** и новый выбор  $a_{k-1} \in S_{k-1}$ ;




# Обход дерева



Прямой порядок обхода дерева. Тупики.



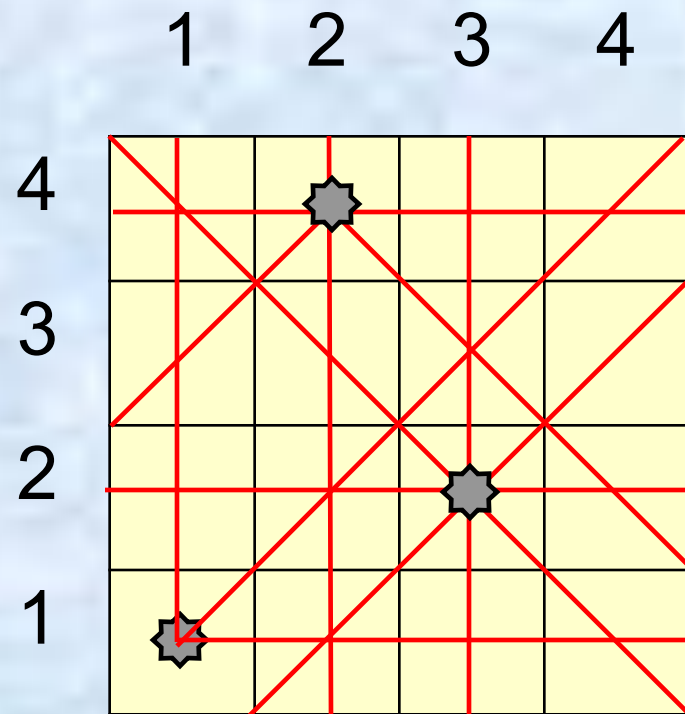
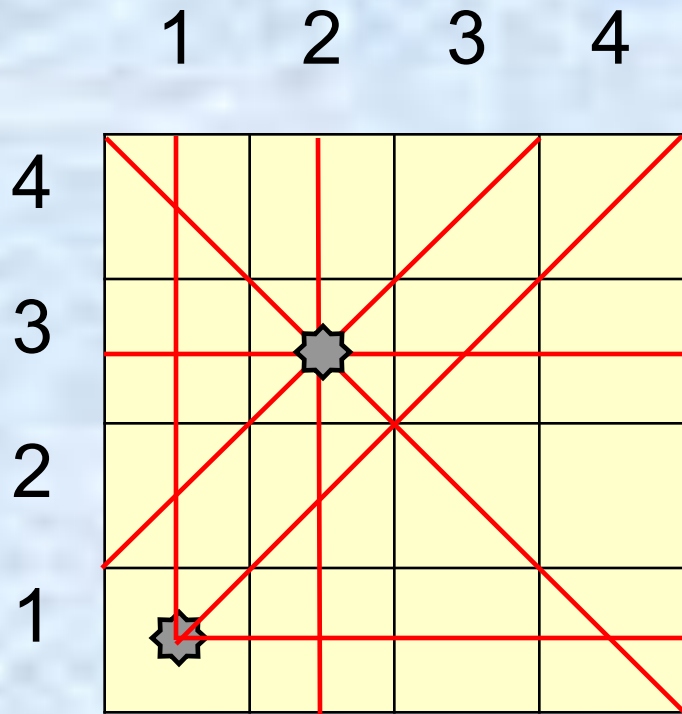
## Общий алгоритм

```
S1 = A1; k = 1; count = 0;
while (k > 0) { // пока не все решения найдены
  while (Sk != ∅) {
    // продвижение вперёд
    
  }
  // end while продвижения вперёд
  k--; // backtrack
} // все решения найдены; count - число обследованных узлов
```

Пример задачи,  
решаемой алгоритмом по этой схеме

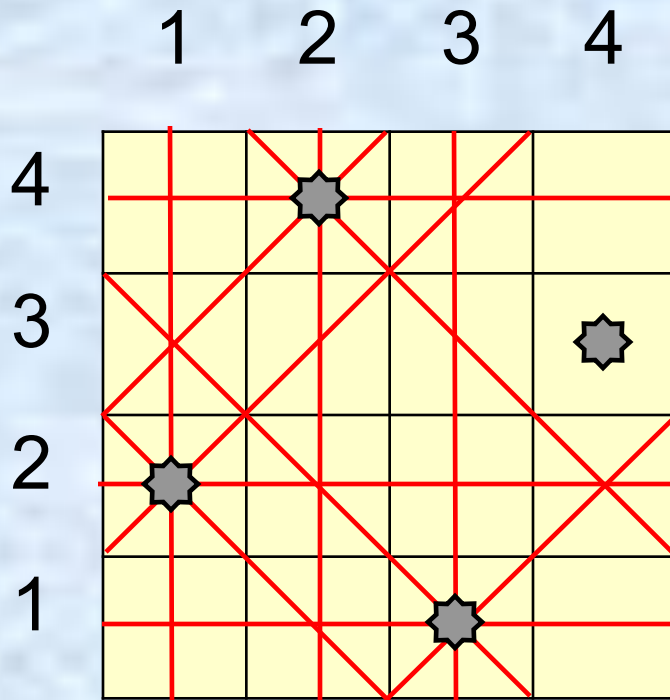
# Задача о ферзях

На шахматной доске размера  $n \times n$  расставить максимальное число не атакующих друг друга ферзей.



$n = 4$

## Продолжение



Решение =  $(a_1, a_2, a_3, a_4)$

$a_i$  - номер горизонтали  
на  $i$ -ой вертикали

Решение =  $(2, 4, 1, 3)$

## Решение $(a_1, a_2, \dots, a_n)$

Ферзи  $i$  и  $k$  атакуют друг друга:

- $i = k$  - ферзи на одной вертикали
- $a_i = a_k$  - ферзи на одной горизонтали
- $|a_i - a_k| = |i - k|$  - ферзи на одной диагонали

Наращивание (продолжение) решения

$$(a_1, a_2, \dots, a_{k-1}) \cdot a_k = (a_1, a_2, \dots, a_{k-1}, a_k)$$

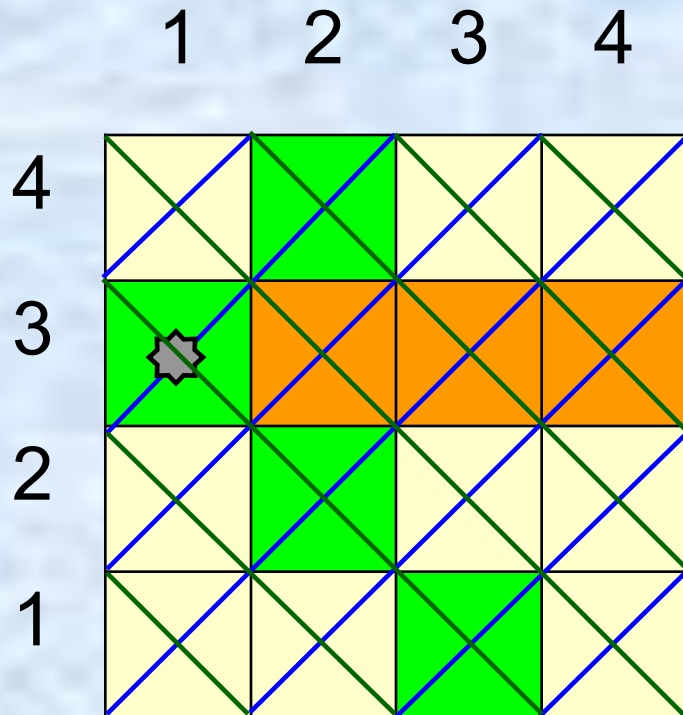
$A_k = (1, 2, \dots, n)$  - номера клеток вертикалей.

Множество  $S_k$  явно не формируется,  
но, выбирая очередного кандидата  $a_k \in A_k$ ,  
проверяем  $a_k \in S_k$

Используется  $s_k$  - нижняя граница в  $A_k$ ,  
т.о. кандидаты в  $S_k$  выбираются из множества  
 $(s_k, s_k+1, \dots, n)$ , т.е.  $S_k \subset (s_k, s_k+1, \dots, n)$ .

Обсудить альтернативу.

# Альтернативное представление $S_k$



Горизонталей =  $n$

Диагоналей =  $2(2n - 1)$



## Проверка s[k]

```
bool noQueen (uns s, uns k)
// ферзь не может быть поставлен в строку s столбца k
{ bool Flag = true;
  uns i = 1;
  while ((i<k) && Flag) {
    // Flag= 'ферзи [1..i) не атакуют поле <k,s>'
    // атакует ли ферзь из i-го столбца поле <k,s>?}
    Flag = !( (a[i]==s) || (abs(a[i]-s)== k-i) );
    i++;
  } //end - while
  return (!Flag);
} // end noQueen
```

# Нахождение очередного свободного поля $s[k]$

```
/* найти следующее наименьшее значение  $s[k]$ ,  
начиная с текущего  $s[k]$ ;  
если такового нет, то  $s[k]=n+1$   
*/  
while (( $s[k] \leq n$ ) && noQueen ( $s[k], k$ ))  $s[k]++$ ;
```

# Реализация

```
void queen1(const uns n)
{ pos s;
/*s[k] - наименьший элемент множества Sk
неопробованных (допустимых) значений
*/
uns count = 0; // счётчик обследованных
// узлов дерева поиска
uns countS = 0; // счётчик найденных
решений
a[1] = 1; s[1] = 1;
uns k = 1;
```

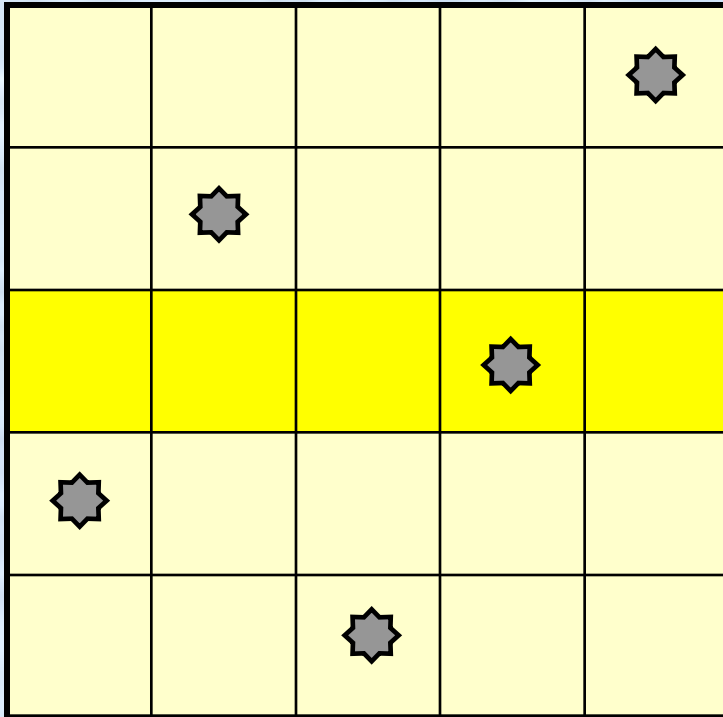
```

while (k>0) {
    while ((k>=1) && (s[k]<=n)) {
        a[k]= s[k]; s[k]++;
        // найти следующее наименьшее значение s[k]
        while ((s[k]<=n) && noQueen (s[k],k)) s[k]++;
        count++;
        if (k==n) {countS++; ...} //решение найдено - фиксировать !
        else {// переход к следующей вертикали
            k++;
            s[k]= 1;
            //найти следующее наименьшее значение s[k],
            while ((s[k]<=n) && noQueen (s[k],k)) s[k]++;
        }
    }
    k--; // backtrack
}

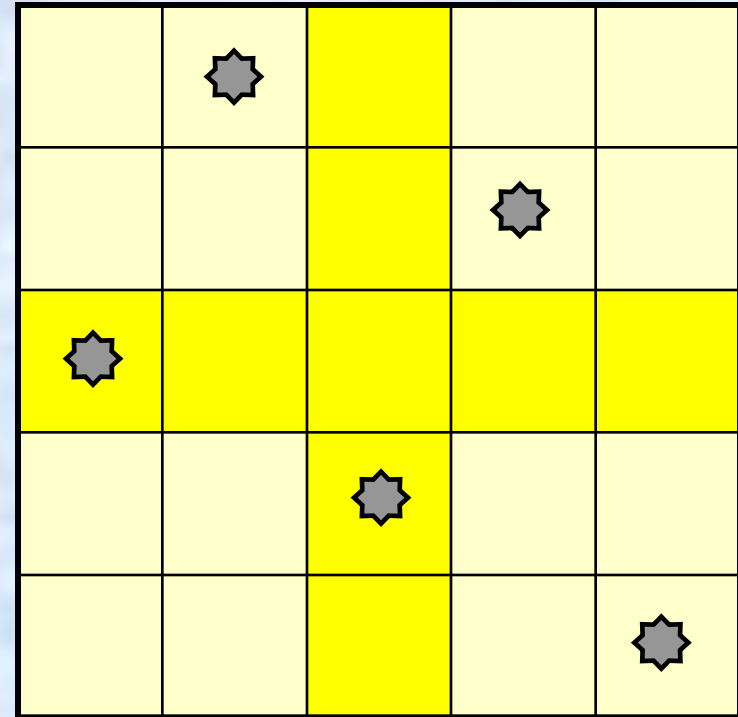
```

# Демонстрация

# Усовершенствования: пояснения к инициализации Вращения и отражения

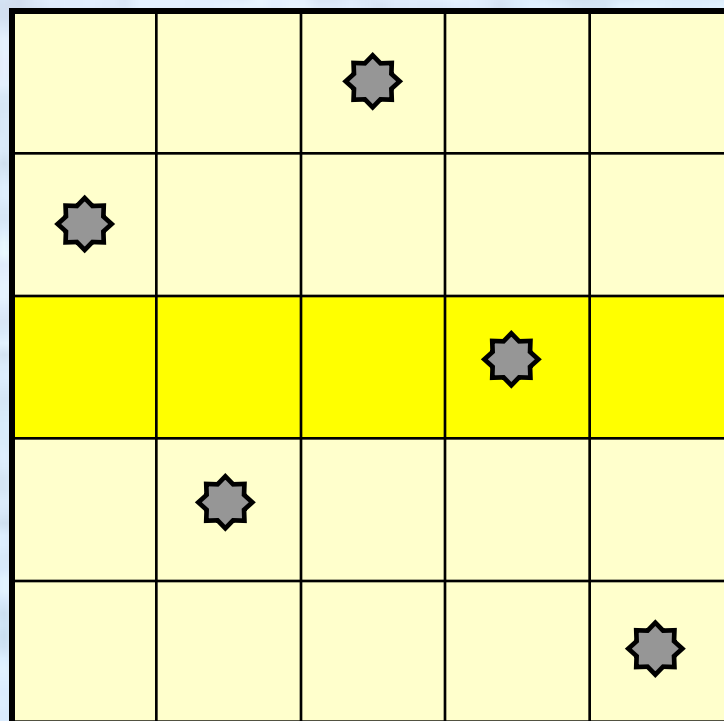
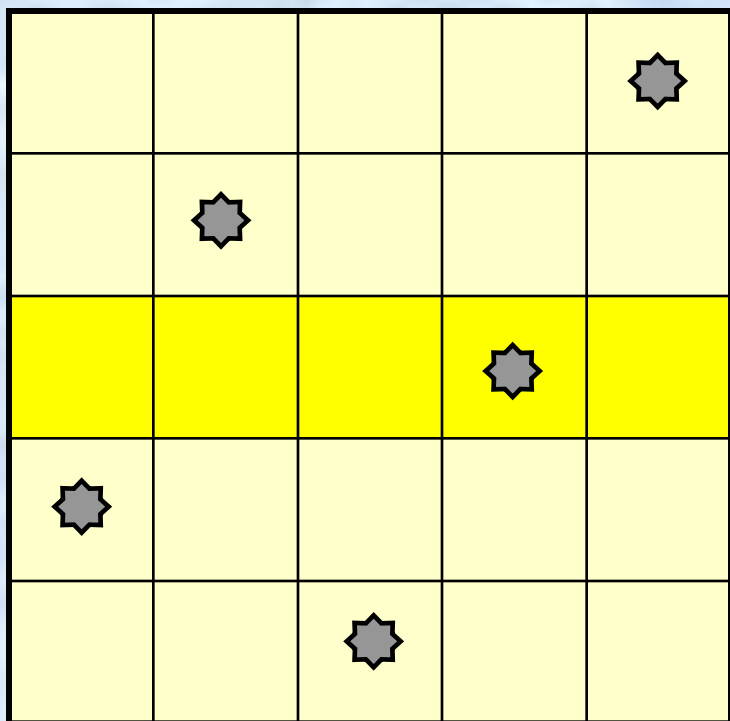


2 4 1 3 5



3 5 2 4 1

# Усовершенствования: пояснения к инициализации Вращения и отражения

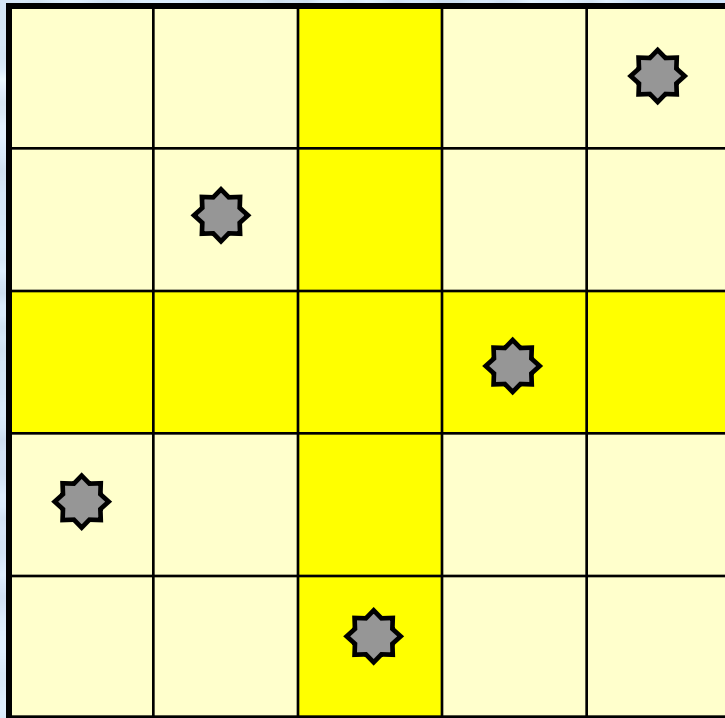


2 4 1 3 5

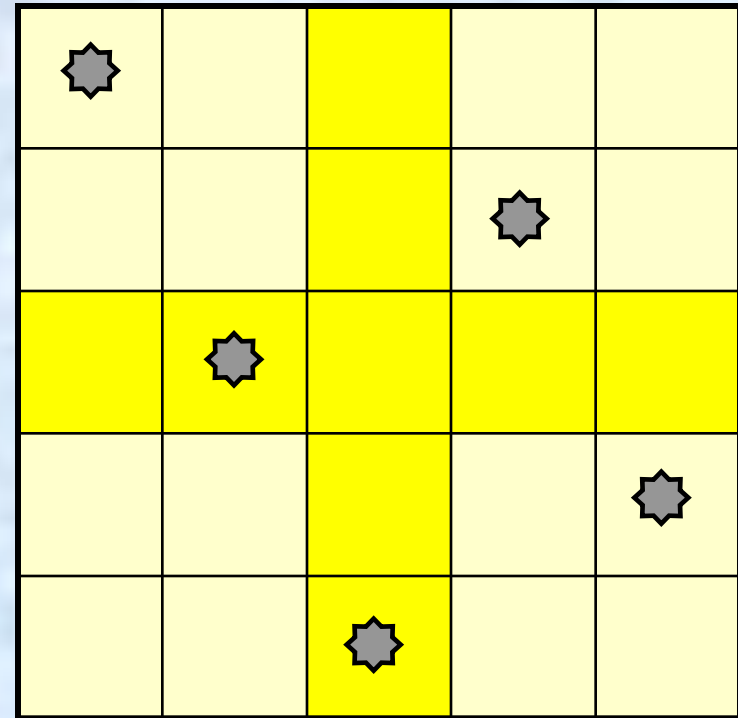
4 2 5 3 1



# Усовершенствования: пояснения к инициализации Вращения и отражения



2 4 1 3 5



5 3 1 4 2

Отсечение и склеивание ветвей

## Усовершенствования

```
void queen1(const uns n)
{pos s; //s[k] - наименьший элемент множества Sk
//неопробованных (допустимых) значений
uns count = 1; // счётчик обследованных узлов
uns countS = 0; // счётчик найденных решений
uns n_div_2 = n/2;
a[1] = 2; s[1] = 3;
uns k = 2; s[2] = 4;
while (k>0){
    while ( ((k>1) && (s[k]<=n)) || ((k==1) && (s[1]<=(n_div_2))) )
    { a[k]= s[k];
      ...
    }
}
```

# Подсчет вариантов

$n=8$

Все возможные способы  $C(n^2|n) \approx 4,4*10^9$

В каждом столбце один  $n^n \approx 1,7*10^7$

+ В каждой строке один  $n! \approx 4,0*10^4$

На каждой диагонали не более одного **2056**

Усовершенствования (вращения и отражения)

**801**

## Результаты

**количество ферзей = 4**

р е ш е н и я :

1 :: 2 4 1 3

всего вершин = 4

**количество ферзей = 5**

р е ш е н и я :

1 :: 2 4 1 3 5

2 :: 2 5 3 1 4

всего вершин = 11

**количество ферзей = 6**

р е ш е н и я :

1 :: 2 4 6 1 3 5

2 :: 3 6 2 5 1 4

всего вершин = 54

**количество ферзей = 7**

р е ш е н и я :

1 :: 2 4 1 7 5 3 6

2 :: 2 4 6 1 3 5 7

3 :: 2 5 1 4 7 3 6

4 :: 2 5 3 1 7 4 6

5 :: 2 5 7 4 1 3 6

6 :: 2 6 3 7 4 1 5

7 :: 2 7 5 3 1 6 4

8 :: 3 1 6 2 5 7 4

9 :: 3 1 6 4 2 7 5

10 :: 3 5 7 2 4 6 1

11 :: 3 6 2 5 1 4 7

12 :: 3 7 2 4 6 1 5

13 :: 3 7 4 1 5 2 6

всего вершин = 164

количество ферзей = 8

решения :

1 :: 2 4 6 8 3 1 7 5  
2 :: 2 5 7 1 3 8 6 4  
3 :: 2 5 7 4 1 8 6 3  
4 :: 2 6 1 7 4 8 3 5  
5 :: 2 6 8 3 1 4 7 5  
6 :: 2 7 3 6 8 5 1 4  
7 :: 2 7 5 8 1 4 6 3  
8 :: 2 8 6 1 3 5 7 4  
9 :: 3 1 7 5 8 2 4 6  
10 :: 3 5 2 8 1 7 4 6  
11 :: 3 5 2 8 6 4 7 1  
12 :: 3 5 7 1 4 2 8 6  
13 :: 3 5 8 4 1 7 2 6  
14 :: 3 6 2 5 8 1 7 4  
15 :: 3 6 2 7 1 4 8 5  
16 :: 3 6 2 7 5 1 8 4  
17 :: 3 6 4 1 8 5 7 2  
18 :: 3 6 4 2 8 5 7 1  
19 :: 3 6 8 1 4 7 5 2  
20 :: 3 6 8 1 5 7 2 4  
21 :: 3 6 8 2 4 1 7 5

22 :: 3 7 2 8 5 1 4 6  
23 :: 3 7 2 8 6 4 1 5  
24 :: 3 8 4 7 1 6 2 5  
25 :: 4 1 5 8 2 7 3 6  
26 :: 4 1 5 8 6 3 7 2  
27 :: 4 2 5 8 6 1 3 7  
28 :: 4 2 7 3 6 8 1 5  
29 :: 4 2 7 3 6 8 5 1  
30 :: 4 2 7 5 1 8 6 3  
31 :: 4 2 8 5 7 1 3 6  
32 :: 4 2 8 6 1 3 5 7  
33 :: 4 6 1 5 2 8 3 7  
34 :: 4 6 8 2 7 1 3 5  
35 :: 4 6 8 3 1 7 5 2  
36 :: 4 7 1 8 5 2 6 3  
37 :: 4 7 3 8 2 5 1 6  
38 :: 4 7 5 2 6 1 3 8  
39 :: 4 7 5 3 1 6 8 2  
40 :: 4 8 1 3 6 2 7 5  
41 :: 4 8 1 5 7 2 6 3  
42 :: 4 8 5 3 1 7 2 6

всего вершин = 801

количество ферзей = 9

решения:

1 :: 2 4 1 7 9 6 3 5 8  
2 :: 2 4 7 1 3 9 6 8 5  
3 :: 2 4 8 3 9 6 1 5 7  
4 :: 2 4 9 7 3 1 6 8 5  
5 :: 2 4 9 7 5 3 1 6 8  
6 :: 2 5 7 1 3 8 6 4 9  
7 :: 2 5 7 4 1 3 9 6 8  
8 :: 2 5 7 9 3 6 4 1 8  
9 :: 2 5 7 9 4 8 1 3 6  
10 :: 2 5 8 1 3 6 9 7 4  
11 :: 2 5 8 1 9 6 3 7 4  
12 :: 2 5 8 6 9 3 1 4 7  
13 :: 2 5 8 6 9 3 1 7 4  
14 :: 2 5 9 4 1 8 6 3 7  
15 :: 2 6 1 3 7 9 4 8 5  
16 :: 2 6 1 7 4 8 3 5 9  
17 :: 2 6 1 7 5 3 9 4 8  
18 :: 2 6 1 9 5 8 4 7 3  
19 :: 2 6 3 1 8 4 9 7 5  
20 :: 2 6 9 3 5 8 4 1 7  
21 :: 2 7 5 1 9 4 6 8 3  
22 :: 2 7 5 8 1 4 6 3 9  
23 :: 2 7 9 6 3 1 4 8 5  
24 :: 2 8 1 4 7 9 6 3 5  
25 :: 2 8 5 3 9 6 4 1 7  
26 :: 2 8 6 9 3 1 4 7 5  
27 :: 2 9 5 3 8 4 7 1 6  
28 :: 2 9 6 3 5 8 1 4 7  
29 :: 2 9 6 3 7 4 1 8 5  
30 :: 2 9 6 4 7 1 3 5 8  
31 :: 3 1 4 7 9 2 5 8 6  
32 :: 3 1 6 8 5 2 4 9 7  
33 :: 3 1 7 2 8 6 4 9 5

34 :: 3 1 7 5 8 2 4 6 9  
35 :: 3 1 8 4 9 7 5 2 6  
36 :: 3 1 9 7 5 2 8 6 4  
37 :: 3 5 2 8 1 4 7 9 6  
38 :: 3 5 2 8 1 7 4 6 9  
39 :: 3 5 7 1 4 2 8 6 9  
40 :: 3 5 8 2 9 6 1 7 4  
41 :: 3 5 8 2 9 7 1 4 6  
42 :: 3 5 9 2 4 7 1 8 6  
43 :: 3 5 9 4 1 7 2 6 8  
44 :: 3 6 2 7 1 4 8 5 9  
45 :: 3 6 2 9 5 1 8 4 7  
46 :: 3 6 8 1 4 7 5 2 9  
47 :: 3 6 8 1 5 9 2 4 7  
48 :: 3 6 8 2 4 9 7 5 1  
49 :: 3 6 8 5 1 9 7 2 4  
50 :: 3 6 8 5 2 9 7 4 1  
51 :: 3 6 9 1 8 4 2 7 5  
52 :: 3 6 9 2 5 7 4 1 8  
53 :: 3 6 9 2 8 1 4 7 5  
54 :: 3 6 9 5 8 1 4 2 7  
55 :: 3 6 9 7 1 4 2 5 8  
56 :: 3 6 9 7 2 4 8 1 5  
57 :: 3 6 9 7 4 1 8 2 5  
58 :: 3 7 2 4 8 1 5 9 6  
59 :: 3 7 2 8 5 9 1 6 4  
60 :: 3 7 2 8 6 4 1 5 9  
61 :: 3 7 4 2 9 5 1 8 6  
62 :: 3 7 4 2 9 6 1 5 8  
63 :: 3 7 4 8 5 9 1 6 2  
64 :: 3 7 9 1 5 2 8 6 4  
65 :: 3 7 9 4 2 5 8 6 1  
66 :: 3 8 2 4 9 7 5 1 6  
67 :: 3 8 4 7 9 2 5 1 6

.....

...  
90 :: 4 2 9 5 1 8 6 3 7  
91 :: 4 6 1 5 2 8 3 7 9  
92 :: 4 6 1 9 5 8 2 7 3  
93 :: 4 6 1 9 7 3 8 2 5  
94 :: 4 6 3 9 2 5 8 1 7  
95 :: 4 6 3 9 2 8 5 7 1  
96 :: 4 6 3 9 7 1 8 2 5  
97 :: 4 6 8 2 5 1 9 7 3  
98 :: 4 6 8 2 5 7 9 1 3  
99 :: 4 6 8 2 7 1 3 5 9  
100 :: 4 6 8 3 1 7 5 2 9  
101 :: 4 6 9 3 1 8 2 5 7  
102 :: 4 7 1 3 9 6 8 5 2  
103 :: 4 7 1 6 9 2 8 5 3  
104 :: 4 7 1 8 5 2 9 3 6  
105 :: 4 7 3 6 9 1 8 5 2  
106 :: 4 7 3 8 2 5 9 6 1  
107 :: 4 7 3 8 6 1 9 2 5  
108 :: 4 7 3 8 6 2 9 5 1  
109 :: 4 7 5 2 9 1 3 8 6  
110 :: 4 7 5 2 9 1 6 8 3  
111 :: 4 7 5 2 9 6 8 3 1  
112 :: 4 7 9 2 5 8 1 3 6  
113 :: 4 7 9 2 6 1 3 5 8  
114 :: 4 7 9 6 3 1 8 5 2  
115 :: 4 8 1 5 7 2 6 3 9  
116 :: 4 8 5 3 1 6 2 9 7  
117 :: 4 8 5 3 1 7 2 6 9  
118 :: 4 9 3 6 2 7 5 1 8  
119 :: 4 9 5 3 1 6 8 2 7  
120 :: 4 9 5 3 1 7 2 8 6  
121 :: 4 9 5 8 1 3 6 2 7

всего вершин = 2857

# Оценка сложности выполнения Метод Монте-Карло

Число исследуемых узлов дерева

$$\cong \prod_{i=1}^n \mu(A_i)$$

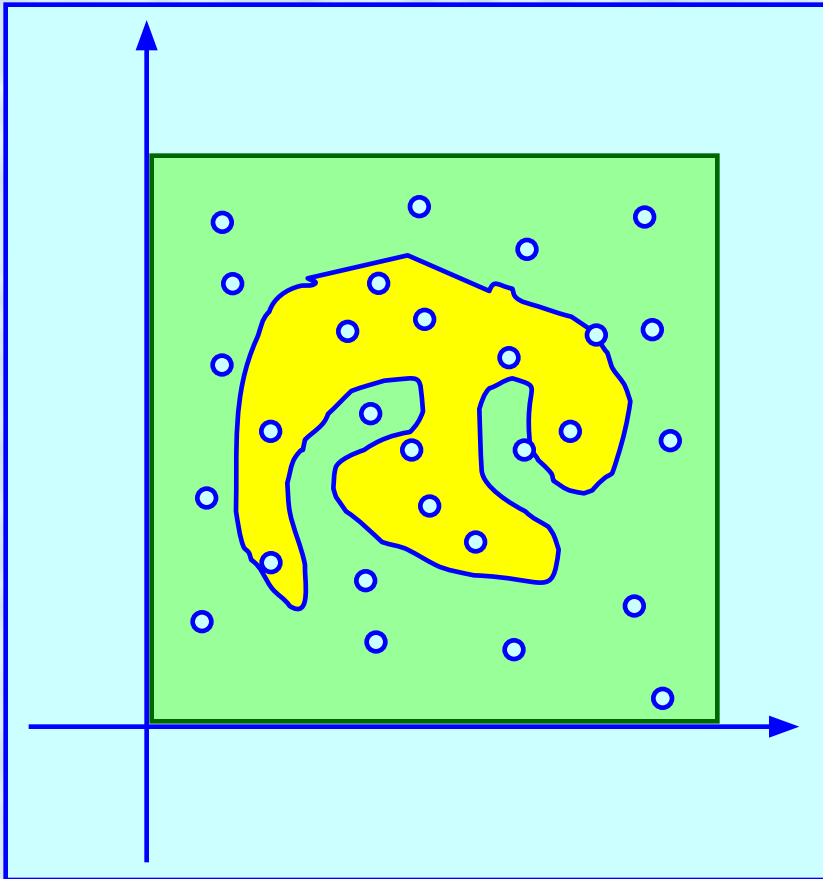
$\mu(A_i)$  - мощность множества  $A_i$

В лучшем случае  $\mu(A_i) \approx \text{Const}$

и тогда число узлов дерева  $\approx C^n$



# Метод Монте-Карло



Оценка площади  
фигуры (интеграла)

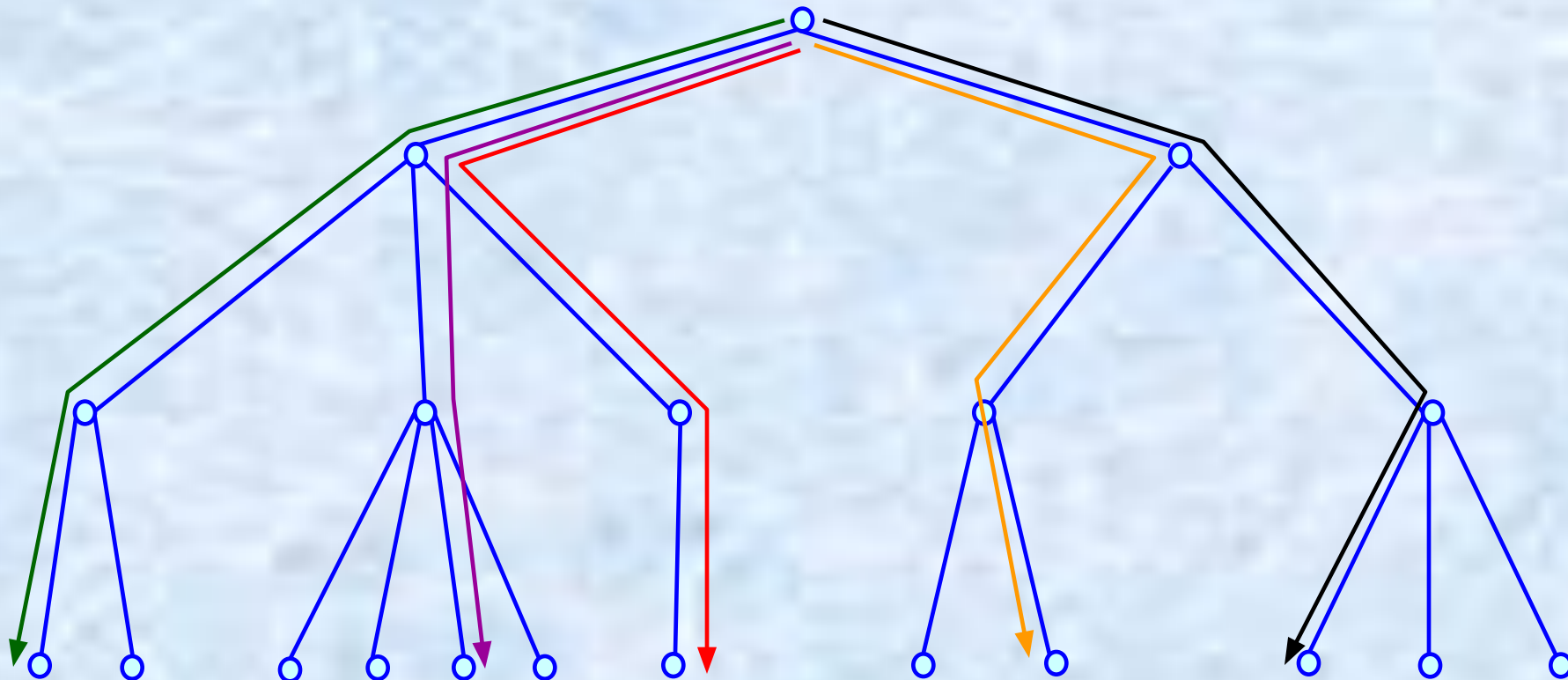
Число точек внутри

---

Общее число точек

# Оценка размеров дерева

Пример: 20 узлов, без корня 19 (количество веток)



(1)  $2+2*3+2*3*4=32$  (4)

(2)  $2+2*2+2*2*2=14$  (5)

(3)  $2+2*2+2*2*3=18$

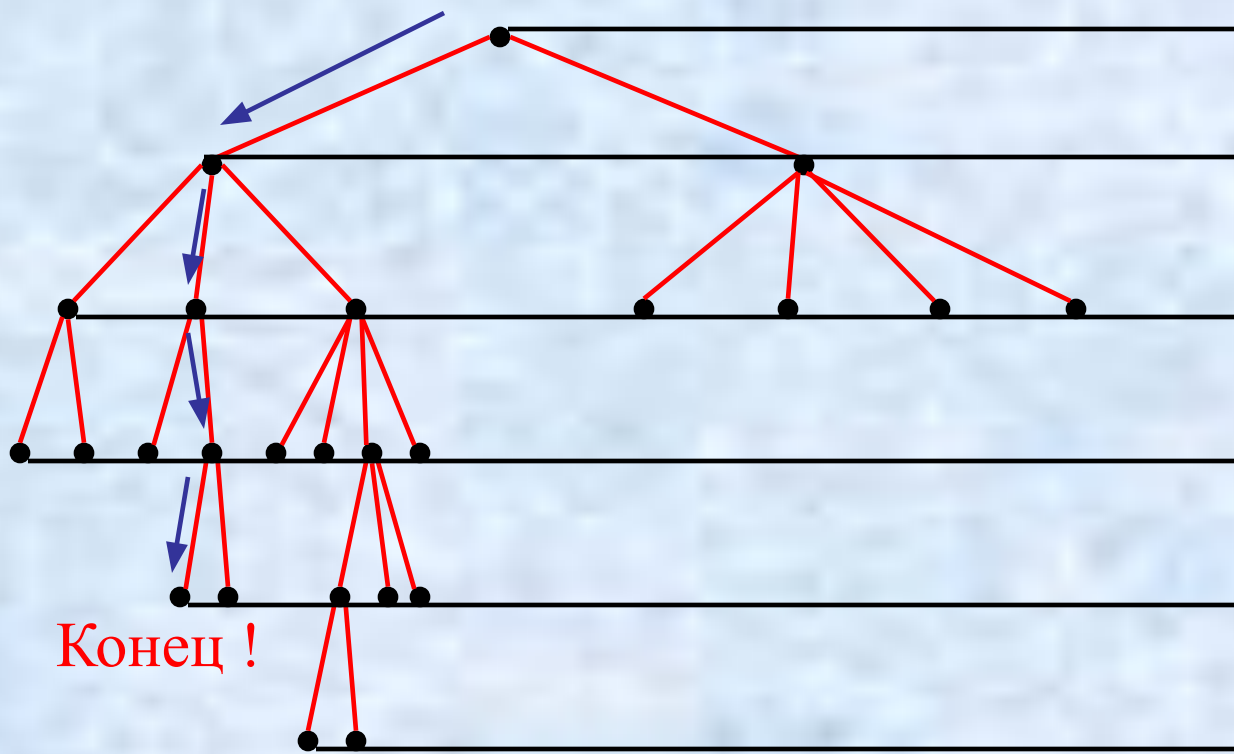
(4)  $2+2*3+2*3*2=20$

(5)  $2+2*3+2*3*1=14$

$(32+14+18)/3 = 64/3=21.3 \approx 21$

$(32+14+18+20+14)/5 = 98/5=19.6 \approx 20$

# Схема испытания



Выбор  $a_1$ :  $m_1 = |S_1|$

Выбор  $a_2$ :  $m_2 = |S_2|$

Выбор  $a_3$ :  $m_3 = |S_3|$

Выбор  $a_4$ :  $m_4 = |S_4|$

Выбор  $a_5$ :  $m_5 = |S_5|$

Выбор  $a_6$ :  $m_6 = |S_6|$

- При  $m_k = |S_k| \neq 0$  выбор  $a_k$  из  $S_k$  случайный с вероятностью  $1/m_k$ .
- При  $m_k = 0$  испытание заканчивается.

# Схема испытания

Случайная величина

$$V = m_1 + m_1 m_2 + m_1 m_2 m_3 + \dots + m_1 m_2 \dots m_L$$

Математическое ожидание

$E(V)$  = число узлов в дереве (отличных от корня)

Напоминание: для случайной величины  $x$  с исходами  $x_1, x_2, \dots, x_k$  и вероятностями  $p_1, p_2, \dots, p_k$  математическое ожидание есть

$$E(x) = \sum_{i=1}^k x_i p_i$$

$$p_i = \frac{1}{k} \Rightarrow E(x) = \frac{1}{k} \sum_{i=1}^k x_i$$

# Внимание!

На следующих слайдах 35-39 дано обоснование схемы Монте-Карло (сл.40).

Это для студентов, которые хотят понять, почему эта схема будет работать!

Покажем, что

$E(V)$  = число узлов в дереве

1) функция на дереве  $T$  (не случайная)

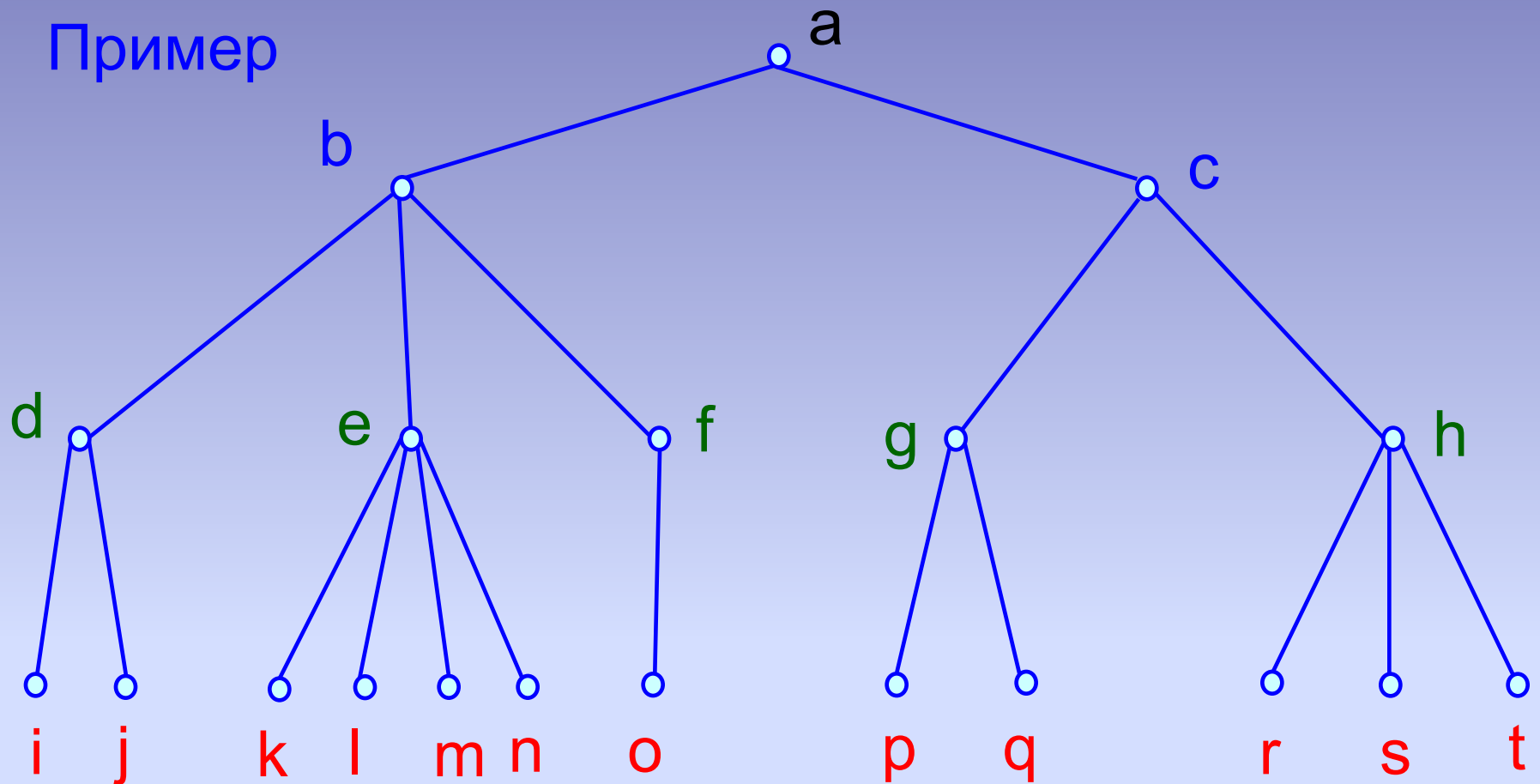
$$\mu(x) = \begin{cases} 1, & \text{если } x = \text{root}(T) \\ v \times \mu(\text{отец}(x)), & \text{если } x \neq \text{root}(T) \end{cases}$$

где  $v$  - число братьев  $x$ , включая самого  $x$   
(т.е. число сыновей узла  $\text{отец}(x)$  )

Пусть путь от корня к узлу  $x$  есть  $v_1, v_2, \dots, v_j$ , тогда

$$\begin{aligned} \mu(x) &= \mu(v_j) = v_j \times \mu(v_{j-1}) = v_j \times v_{j-1} \times \mu(v_{j-2}) = \dots = \\ &= v_j \times v_{j-1} \times \dots \times v_1 \times \mu(v_1) = v_j \times v_{j-1} \times \dots \times v_1 \end{aligned}$$

# Пример



$$\mu(a) = 1, \mu(b) = \mu(c) = 2, \mu(d) = \mu(e) = \mu(f) = 2 \cdot 3 = 6,$$

$$\mu(g) = \mu(h) = 4, \mu(i) = \mu(j) = 12, \mu(k) = \mu(l) = \mu(m) = \mu(n) = 24,$$

$$\mu(o) = 6, \mu(p) = \mu(q) = 8, \mu(r) = \mu(s) = \mu(t) = 12$$

2) Функция «индикатор», описывающая случайность

$$I(x) = \begin{cases} 1, & \text{если узел } x \text{ пройден при испытании} \\ 0, & \text{если узел } x \text{ не пройден при испытании} \end{cases}$$

Случайное событие = «узел  $x$  пройден»,

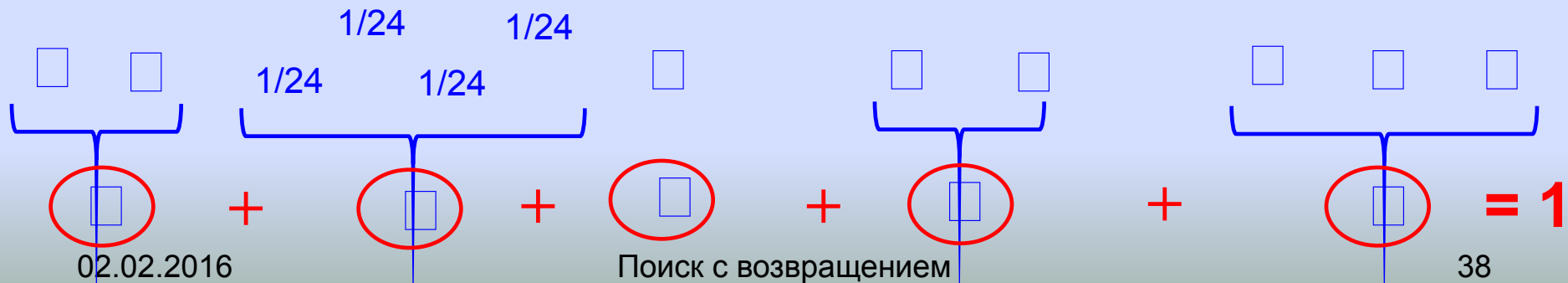
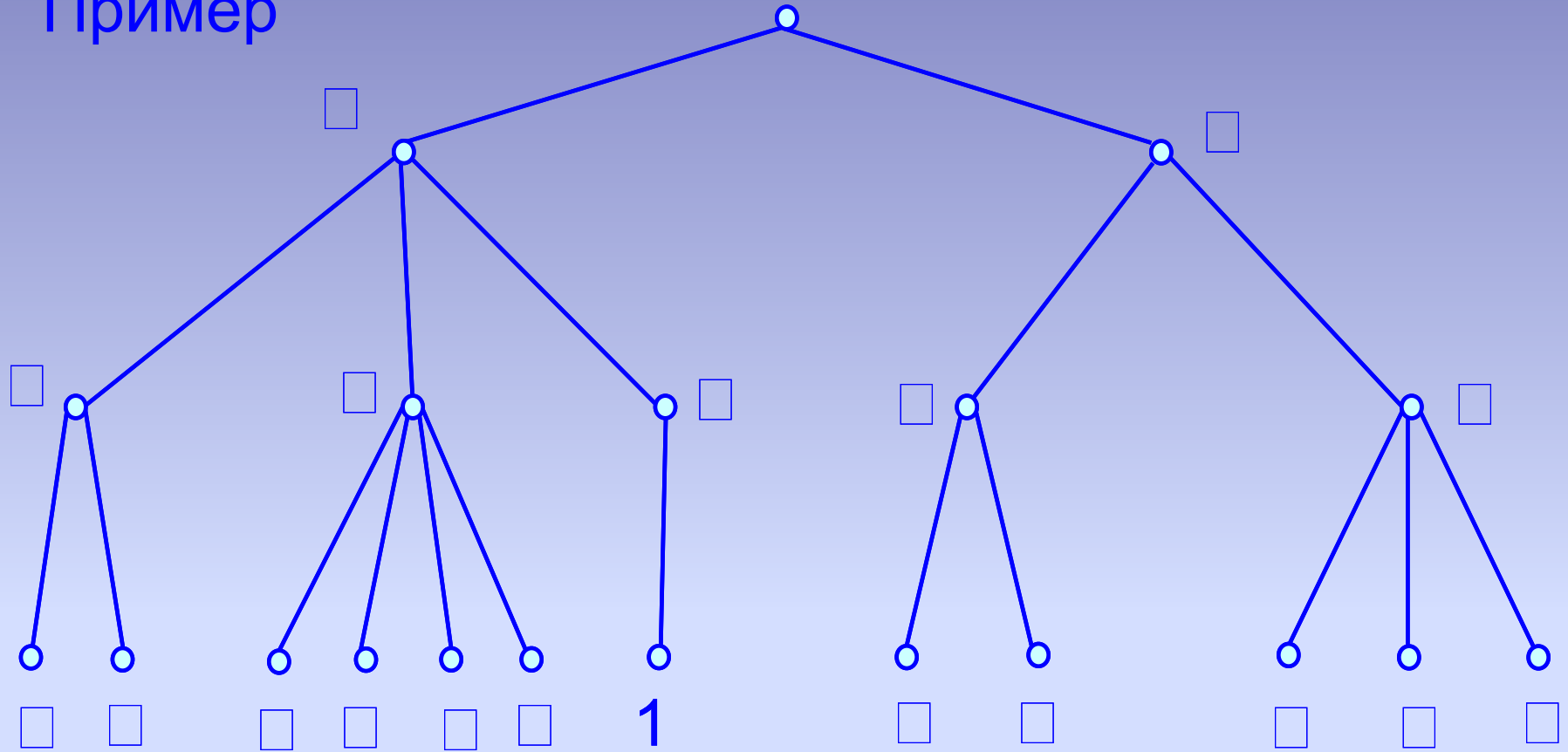
а  $I(x)$  – случайная величина  $\in \{0,1\}$

Вероятность дойти до узла  $x = v_j$  есть

$$(1/m_1) \times (1/m_2) \times \dots \times (1/m_j)$$



# Пример



Итак, покажем, что

$E(V)$  = число узлов в дереве

$$V = m_1 + m_1 m_2 + \dots + \prod_{i=1}^L m_i = \sum_{\substack{x \in T \\ x \neq \text{root}(T)}} \mu(x) I(x)$$

$$E(V) = \sum_{\substack{x \in T \\ x \neq \text{root}(T)}} \mu(x) E(I(x)) = \sum_{\substack{x \in T \\ x \neq \text{root}(T)}} \mu(x) \frac{1}{\mu(x)} =$$

= число узлов в дереве, поскольку

$$E(I(x)) = \Pr(I(x) = 1) \cdot 1 + \Pr(I(x) = 0) \cdot 0 =$$

$$= \Pr(I(x) = 1) = \frac{1}{\mu(x)}$$

# Общий алгоритм

```
// Монте-Карло
SV = 0; // M - число испытаний
for (i = 1; i <= M; i++) {
    k = 1; S1 = A1; m1 = |S1|;
    Sum = 0; Prod = 1;
    while (mk ≠ 0) {
        // продвижение вперёд
        Prod* = mk;
        Sum+ = Prod;
        ak = случайный выбор очередного элемента из Sk;
        k ++;
        вычислить Sk и mk;
    }
    SV := SV + Sum; // добав. рез. очередного испытания
} // end - for
V = SV / M;
```

```

begin { MonteCarlo }
  Randomize; n_div_2 := n div 2;  all := 0;
for iExp:=1 to nExp do
  begin { очередное испытание }
    m_k := n_div_2 - 1;  num := Random ( m_k ) + 1;
    a[1] := 1+num;  k := 2;  prod := m_k;  sum := prod;
    FormSk ( k, m_k, S_k );
    while m_k<>0 do
      begin
        prod := prod*m_k;  sum := sum + prod;
        num := Random ( m_k ) + 1;  a[k] := S_k[num];
        k := k + 1;  FormSk ( k, m_k, S_k );
      end {while};
      all := all + sum
    end {for};
  v := all/nExp
end { MonteCarlo };

```

```
procedure FormSk ( k: Nat; var m_k: Nat0; var S_k: pos );  
{ формирует "множество" (вектор) S_k возможных ходов и  
его мощность m_k; если S_k пусто, то m_k=0 }  
var s: Nat;  
begin  
  m_k := 0;  
  for s:=1 to n do  
    if not NoQueen( k, s) then  
      begin { можно ставить }  
        m_k := m_k + 1;  
        S_k[m_k] := s  
      end;  
  end {FormSk};
```

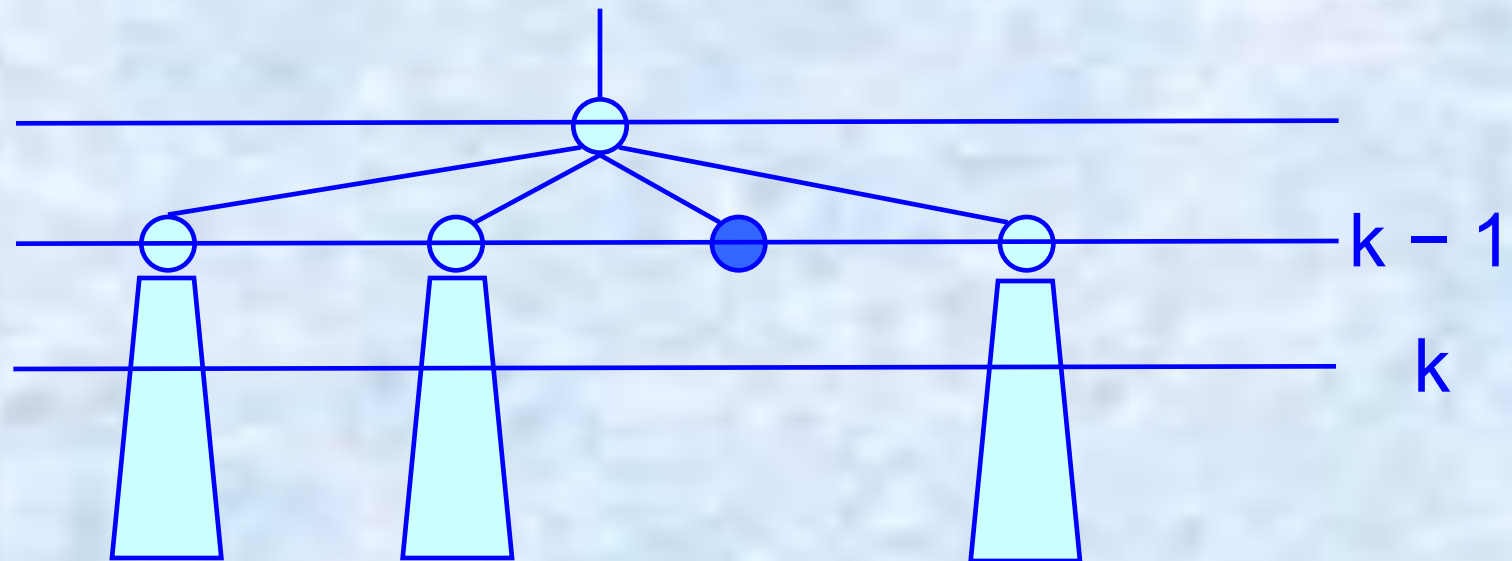
## См. файлы с результатами

- Queen
- Queen\_re

# Backtracking.

## Другие способы программирования

### 1. Рекурсивный подход



```
void backtrack (sequence a, int k);
```

```
// a = (a1, a2, ..., ak-1) - частичное решение
```

```

void backtrack (sequence a, int k)
// a = (a1, a2, ..., ak-1) - частичное решение
{
  if (a - решение) {φ фиксировать a;}
  else {
    ВЫЧИСЛИТЬ Sk;
    for (∀ b ∈ Sk) backtrack ( postfix (a, b), k+1 );
  }
} // end - backtrack

```

```

/*Старт:*/ k = 1; a = Create; backtrack (a, k);

```



## 2. Макрокоманды

Уменьшение «накладных расходов»  
(все решения одной длины  $n$ )

### Макрокоманда

$CODE_k$ : вычислить  $S_k$ ;

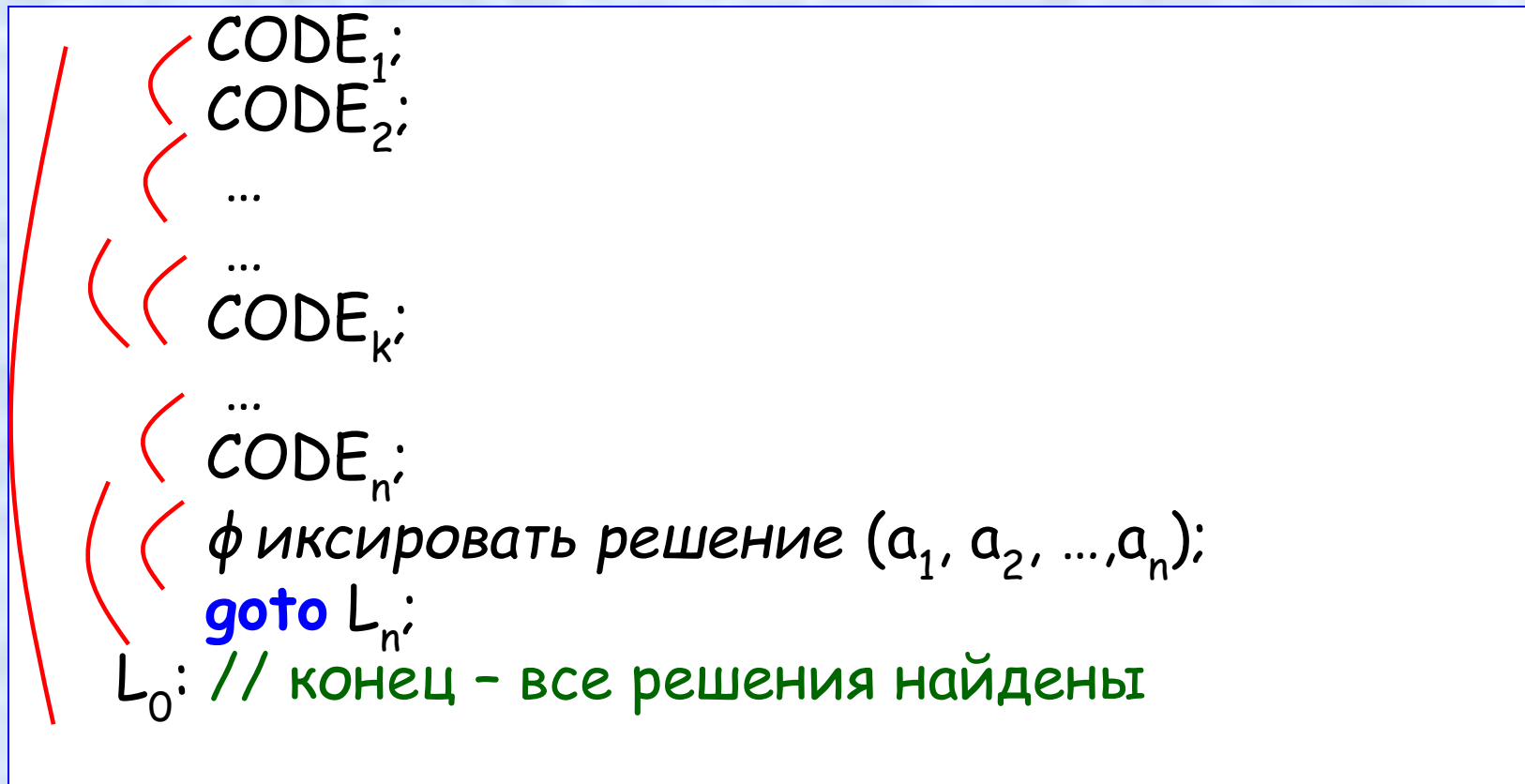
$L_k$ : **if**  $S_k = \emptyset$  **then goto**  $L_{k-1}$ ;

$a_k$  = очередной элемент из  $S_k$ ;

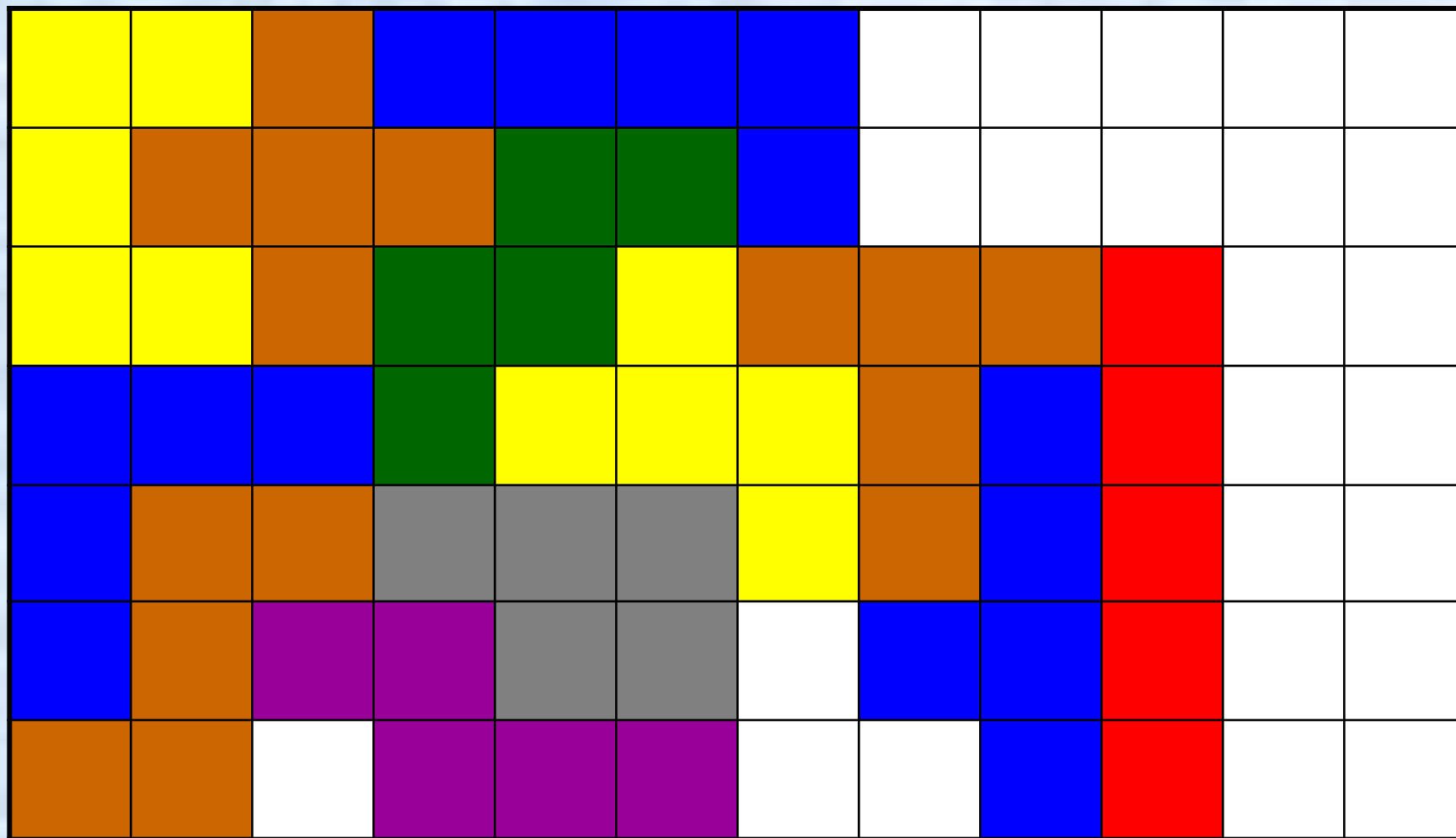
$S_k := S_k - \{a_k\}$ ;

Цикл периода макрогенерации:

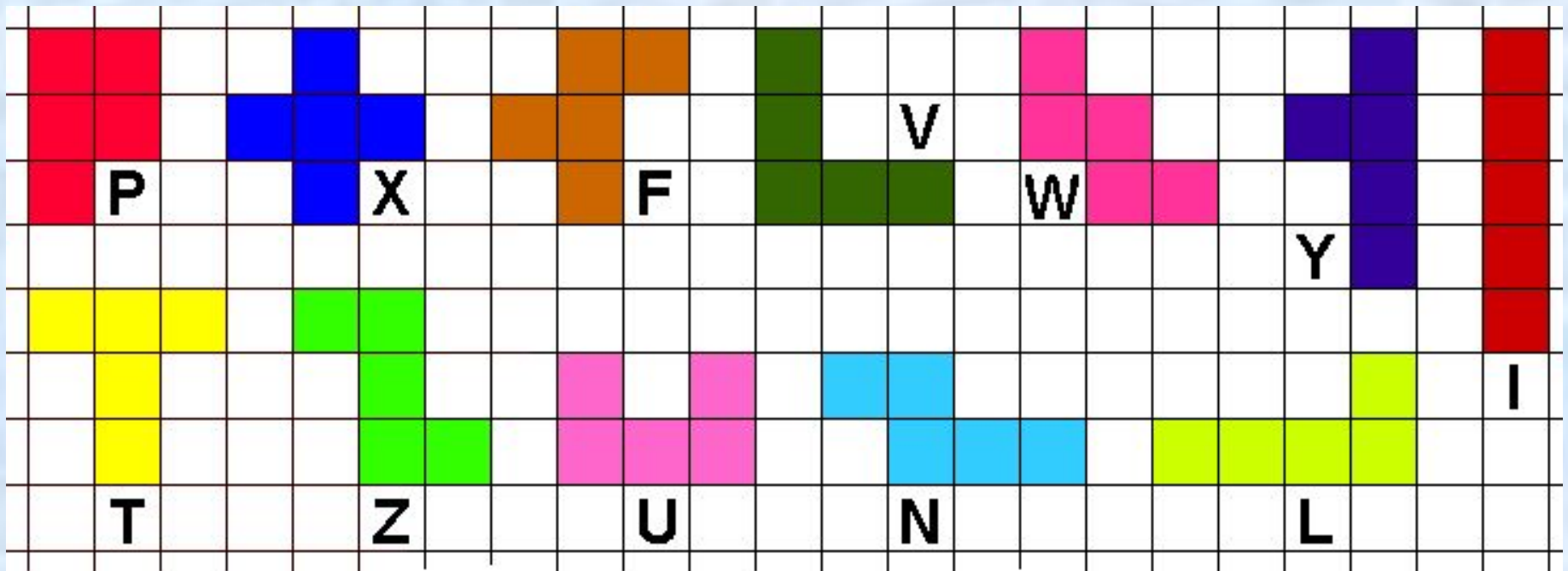
**for** ( k = 1; k<=n; k++) CODE<sub>k</sub>;

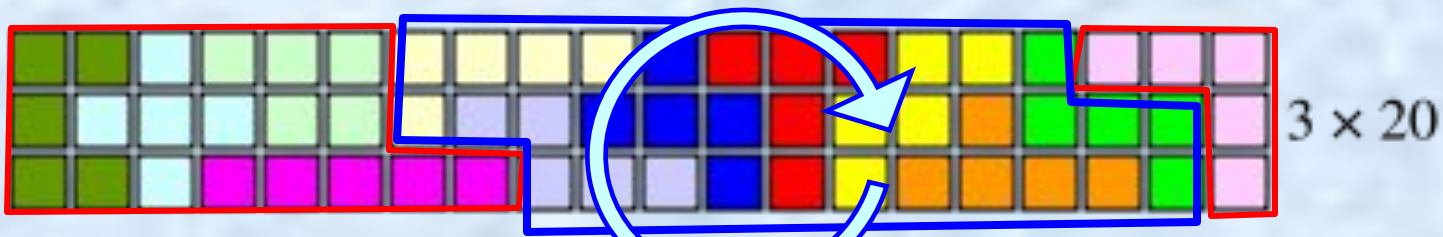
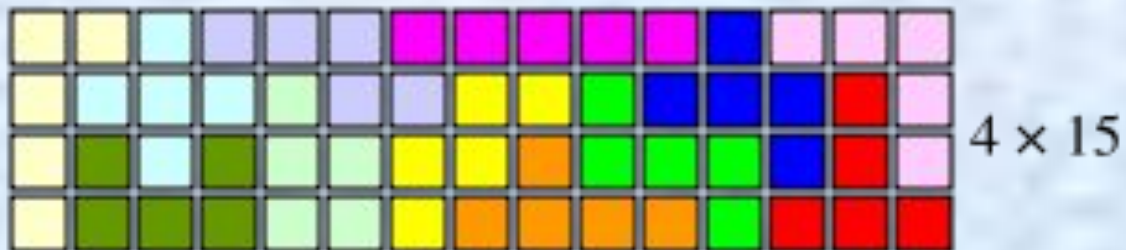
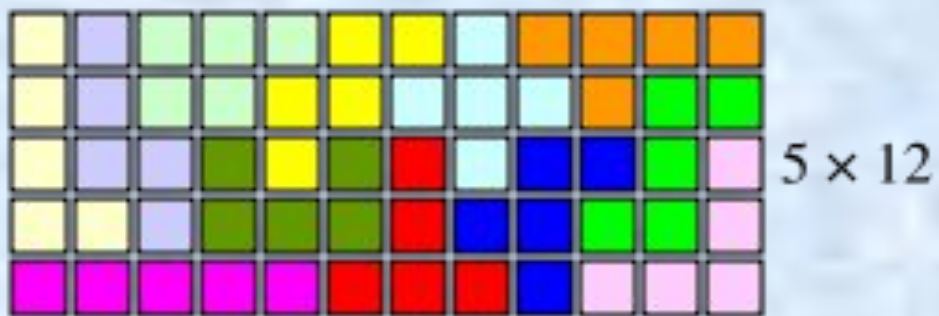
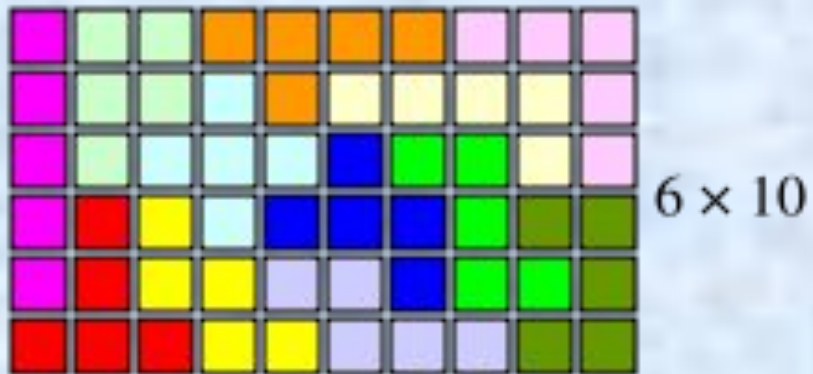


# Пентамино



# Пентамино





Для случая  $6 \times 10$  эту задачу впервые решил в 1965 году Джон Флетчер [1].

Существует ровно 2339 различных укладок пентамино в прямоугольник  $6 \times 10$ , не считая поворотов и отражений целого прямоугольника, но считая повороты и отражения его частей

(иногда внутри прямоугольника образуется симметричная комбинация фигур, поворачивая которую, можно получить дополнительные решения; для прямоугольника  $3 \times 20$ , приведённого на рисунке, второе решение можно получить поворотом блока из 7 фигур, или, иначе говоря, если поменять местами четыре фигуры, крайние слева, и одну крайнюю справа, см. предыдущий слайд).



## Продолжение

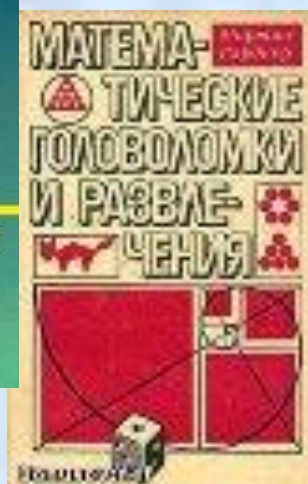
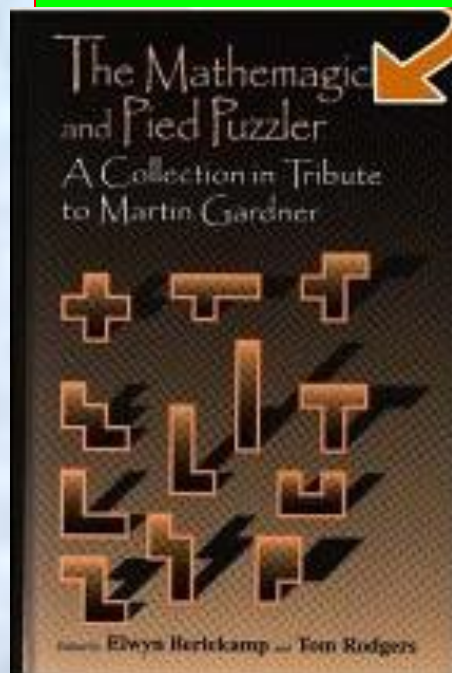
Для прямоугольника  $5 \times 12$  существует 1010 решений,

$4 \times 15$  — 368 решений,

$3 \times 20$  — всего 2 решения.

John G. Fletcher (1965). "A program to solve the pentomino problem by the recursive use of macros". *Communications of the ACM* 8, 621-623.

# Мартин Гарднер



02.02.2016

Поиск с возвратением

53



КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ