

# Алгоритмы и структуры данных

## Лекция 8.2

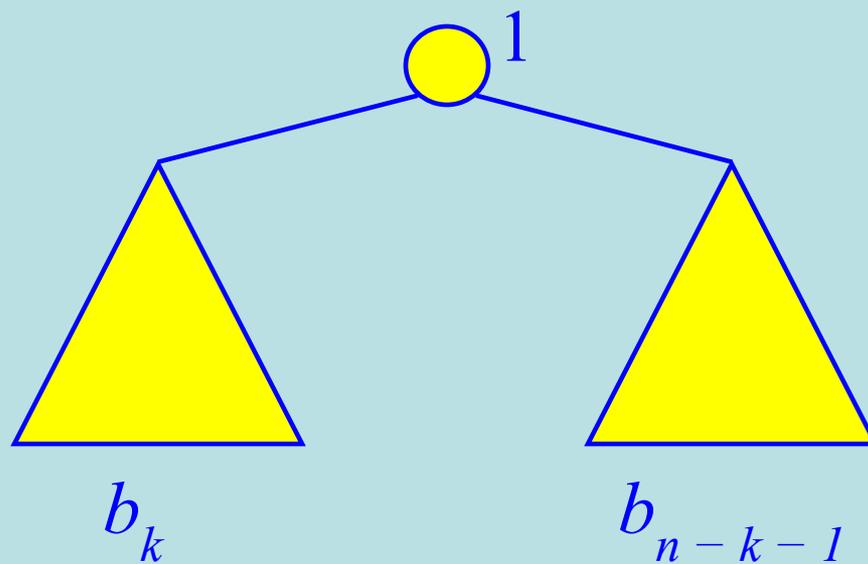
### Часть 2

0. дополнить – числа Каталана

1. Случайные бинарные деревья поиска (БДП) (продолжение)
2. Операции вращения в БДП
3. Случайные БДП с рандомизацией

Число  $b_n$   
структурно различных бинарных деревьев с  $n$  узлами

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-k-1} = b_0 b_{n-1} + b_1 b_{n-2} + \dots + b_{n-2} b_1 + b_{n-1} b_0, \quad b_0 = 1, b_1 = 1.$$



$$k \in 0..(n-1)$$

Это рекуррентное уравнение с точностью до обозначений совпадает с рекуррентным уравнением, получающимся при подсчете числа расстановки скобок в произведении  $n$  сомножителей (см. далее)

Начальное условие  $b_0 = 1$ . Далее

$$b_1 = b_0 b_0 = 1,$$

$$b_2 = b_0 b_1 + b_1 b_0 = 2,$$

$$b_3 = b_0 b_2 + b_1 b_1 + b_2 b_0 = 5.$$

$$b_4 = b_0 b_3 + b_1 b_2 + b_2 b_1 + b_3 b_0 = 14.$$

Оказывается [7, с. 393], что решением этого рекуррентного уравнения являются так называемые

**числа Каталана**  $b_k = C_k$ ,

где  $C_k = (2k \mid k) / (k+1)$ ,

а запись  $(n \mid m)$  обозначает биномиальный коэффициент

$$(n \mid m) = n! / (m! (n - m)!).$$

См. также 1.6.10 и 1.7.4 в книге



При больших значениях  $k$  удобно использовать формулу Стирлинга

$$k! \approx (2\pi)^{1/2} k^{k+\frac{1}{2}} e^{-k}$$

Тогда для чисел Каталана при больших значениях  $n$  справедливо

$$C_n \approx \frac{4^n}{n\sqrt{\pi n}}$$

т. е. число структурно различных бинарных деревьев есть экспоненциальная функция от  $n$ .

## Несколько первых чисел Каталана

$n$	0	1	2	3	4	5	6	7	8	9	10
$C_n$	1	1	2	5	14	42	132	429	1430	4862	16 796

**Конец отступления про числа Каталана**

# Случайные БДП

bst3.cpp

Пусть во входном потоке находится последовательность элементов, по которой функция *SearchAndInsert* строит БДП:

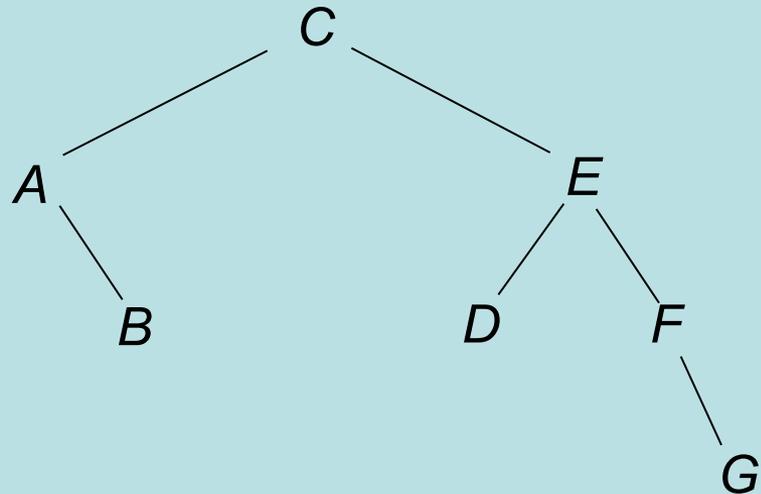
```
b = Create();  
while (infile2 >> c)  
{ SearchAndInsert (c, b);  
}
```

БДП, построенное таким способом, называется случайным БДП

# Случайные бинарные деревья поиска

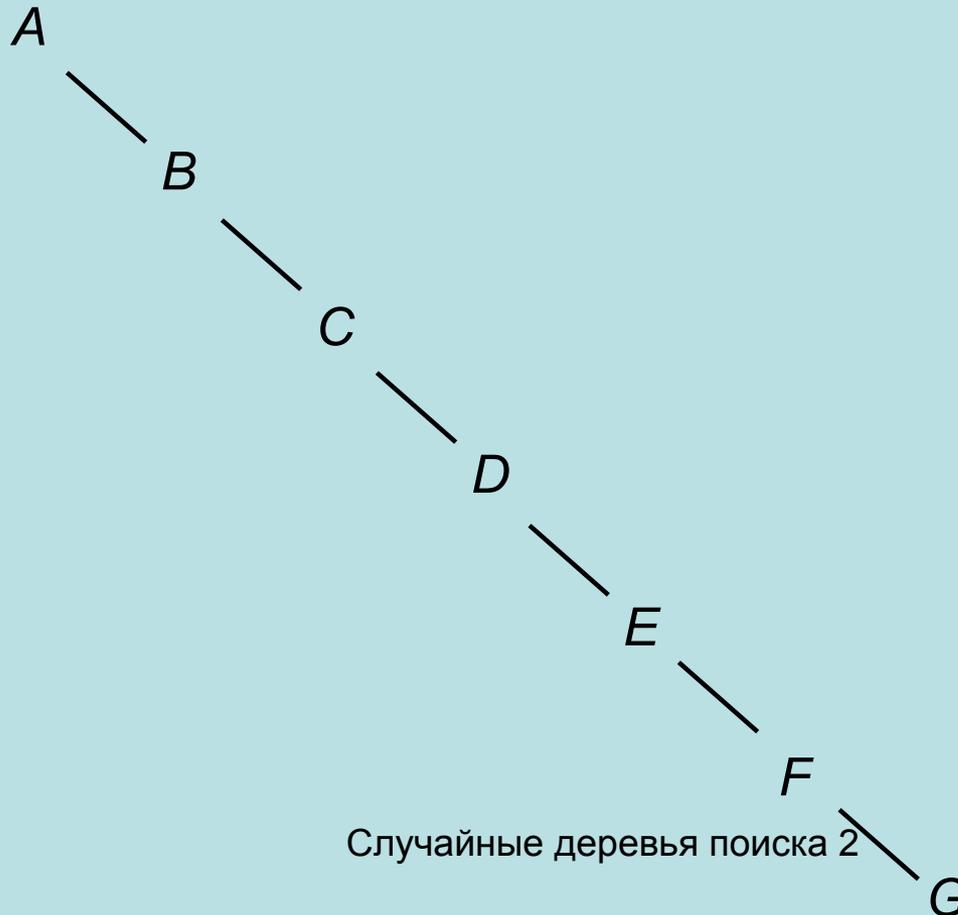
Входная последовательность (например, из файла):

*C*    *E*    *A*    *F*    *B*    *D*    *G*



# Случайные бинарные деревья поиска

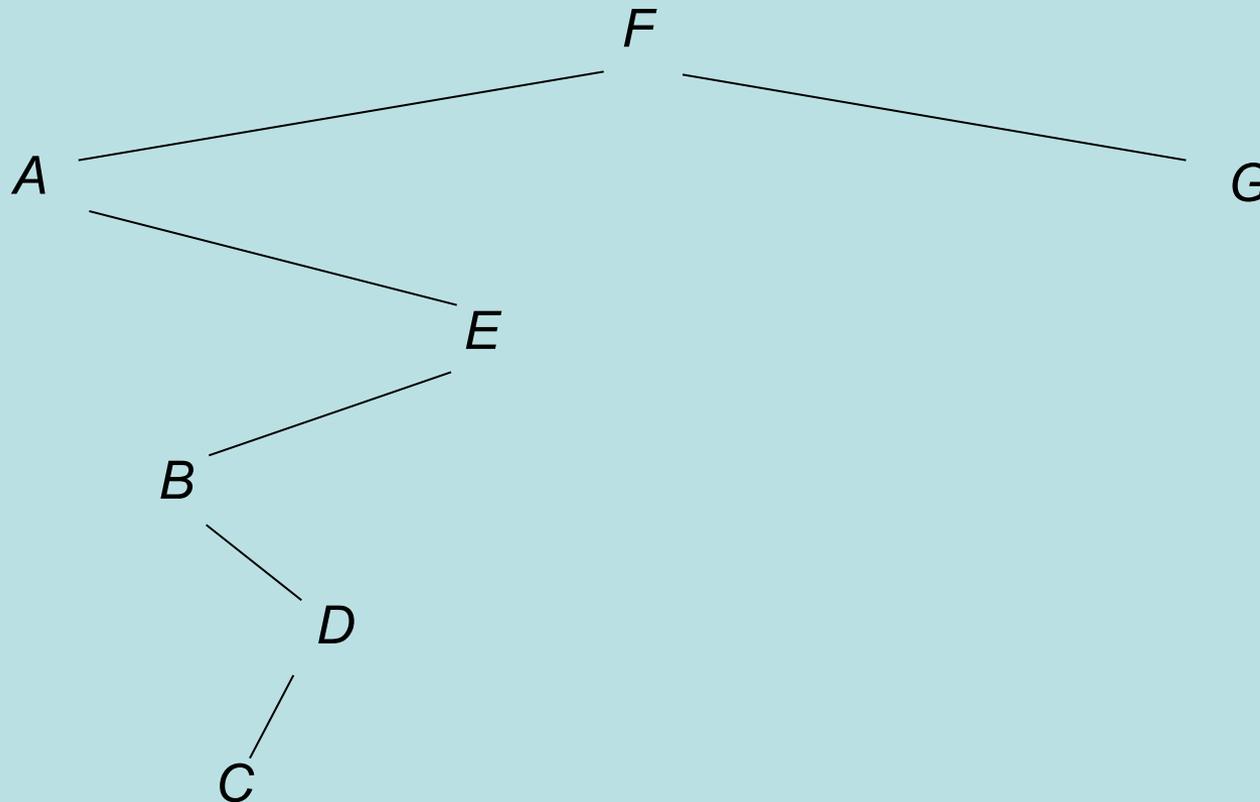
Входная последовательность (например, из файла):



# Случайные бинарные деревья поиска

Входная последовательность (например, из файла):

*F A E B G D C*



# Среднее время поиска в случайных деревьях

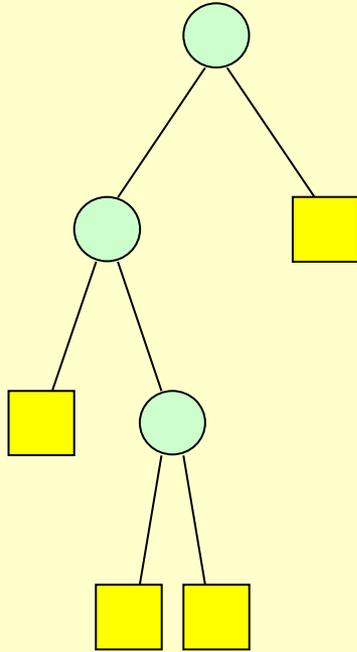


Рис. 2.2. Расширенное бинарное дерево из трех элементов

Полезные характеристики бинарных деревьев.

*Расширенное* бинарное дерево получено из исходного заменой пустых поддеревьев на узлы специального типа, которые будем называть *внешними узлами* или *листьями* в отличие от остальных узлов исходного дерева, которые назовем *внутренними*.

Расширенное дерево *строго бинарное*. Такие деревья фактически уже рассматривались в задаче кодирования. Было показано, что в расширенном бинарном дереве из  $n$  внутренних узлов имеется ровно  $n + 1$  внешний узел.

Определим 2 понятия: *длину внешних путей*  $E(T)$  дерева  $T$  и *длину внутренних путей*  $I(T)$ .

Обозначим уровень узла  $b$  или длину пути от корня до него как  $l(b)$ . При этом  $l(\text{корень}) = 0$ . Тогда длина внешних путей  $E(T)$  определяется как

$$E(T) = \sum_{i=1}^{n+1} l(\boxed{i})$$

Длина внутренних путей  $I(T)$  определяется аналогично как

$$I(T) = \sum_{i=1}^n l(\textcircled{i})$$

Для пустого дерева  $E(T) = 0$  и  $I(T) = 0$ .

Оказывается, величины  $E(T)$  и  $I(T)$   
связаны соотношением  
 $E(T) = I(T) + 2n$ .

Доказательство проведем по индукции.

Пусть в дереве  $T$  из  $n$  внутренних узлов левое поддереве есть  $T_l$ , а число внутренних узлов в нем есть  $n_l$ . Аналогично для правого поддерева –  $T_r$ ,  $n_r$ . Очевидно,  $n_l + n_r = n - 1$ . Тогда

$$E(T) = E(T_l) + E(T_r) + (n + 1),$$

$$I(T) = I(T_l) + I(T_r) + (n - 1).$$

Рассмотрим разность  $D(T) = E(T) - I(T)$ . Вычитая из рекуррентного соотношения для  $E(T)$  рекуррентное соотношение для  $I(T)$ , получим

$$D(T) = D(T_l) + D(T_r) + 2.$$

Теперь покажем по индукции, что  $D(T) = 2n$ . Действительно, так как по индуктивному предположению  $D(T_l) = 2n_l$  и  $D(T_r) = 2n_r$ , то  $D(T) = 2n_l + 2n_r + 2 = 2(n_l + n_r + 1) = 2n$ , что и завершает доказательство.

*Среднее расстояние до внешнего узла равно*

$$E(T) / (n + 1),$$

*а среднее расстояние до внутреннего узла равно*

$$I(T) / n.$$

Обозначим для заданного БДП *среднее* число сравнений при *удачном* (т. е. закончившемся во внутреннем узле) поиске как  $C(n)$ , а среднее число сравнений при *неудачном* (т. е. закончившемся во внешнем узле) поиске как  $C^*(n)$ . Если считать все исходы поиска равновероятными, то имеем

$$C(n) = \frac{I(T)}{n} + 1, \quad C^*(n) = \frac{E(T)}{n + 1}$$

Поскольку  $E(T) = I(T) + 2n$ , то отсюда следует, что в среднем по дереву

$$C^*(n) = \frac{n}{n+1} (C(n) + 1)$$

$$C(n) = \frac{n+1}{n} C^*(n) - 1$$

Эти соотношения справедливы для любого БДП.

«Средние» по всем исходам поиска для данного БДП.

# Затраты на поиск в случайном БДП

Число сравнений,

*среднее по всем  $n!$  перестановкам входных данных, т. е. по всем возможным случайным БДП.*

*(Замечание. Вариант по всем структурно различным – иной результат)*

Уже есть одно соотношение (см. слайд 15) для двух неизвестных нам величин  $C(n)$  и  $C^*(n)$ .

Это соотношение верно для *заданного* дерева в среднем по всем предъявлениям элемента для поиска.

## Для средних по всем перестановкам

- любой элемент данных находится с *равной вероятностью* на первой, второй и т. д. позиции в совокупности входных перестановок, т. е. с *равной вероятностью* он вставлялся как первый, второй и далее по порядку элемент дерева при последовательном построении дерева
- число сравнений при удачном поиске на единицу больше, чем число сравнений при том неудачном поиске элемента в дереве, после которого он был добавлен в дерево.

$$C(n) = 1 + \frac{C^*(0) + C^*(1) + C^*(2) + \dots + C^*(n-1)}{n}. \quad (##)$$

$$C(n) = \frac{n+1}{n} C^*(n) - 1 \quad (***)$$

Заменяя в (\*\*)  $C(n)$ , используя (\*\*\*), получим соотношение

$$C^*(n) \frac{n+1}{n} - 1 = 1 + \frac{1}{n} (C^*(0) + C^*(1) + \dots + C^*(n-1))$$

или в иной форме

$$\left[ \begin{array}{l} (n+1)C^*(n) = (C^*(0) + \dots + C^*(n-1)) + 2n \quad \text{для } n \\ nC^*(n-1) = (C^*(0) + \dots + C^*(n-2)) + 2n - 2 \quad \text{для } n-1 \end{array} \right.$$

---


$$C^*(n) = C^*(n-1) + \frac{2}{n+1}$$

$$C^*(n) = C^*(n-1) + \frac{2}{n+1}$$

Легко проверить, что решением этого рекуррентного соотношения является

$$C^*(n) = 2H(n+1) - 2$$

где  $H(n)$  – частичная сумма *гармонического* ряда или *гармоническое* число (см. [7, 6.3], [9, 1.2.7]), т. е.

$$H(n) = \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

или в рекуррентном виде

$$H(n) = H(n-1) + \frac{1}{n}$$

$$C(n) = 2\left(1 + \frac{1}{n}\right)H(n) - 3$$

Известно [7, 6.3], что  $H(n) = \ln n + \gamma + \frac{1}{12n^2} + \dots$

, где постоянная Эйлера  $\gamma \approx 0.577$ .

$$C^*(n) = 2\ln(n+1) + 2(\gamma - 1) + \frac{2}{12n^2} + \dots$$

$n \uparrow$

$$C^*(n) \approx 2\ln(n+1)$$

$$C(n) \approx 2\ln n$$

$$\ln n = \ln 2 \cdot \log_2 n$$

$$\ln 2 \approx 0.693$$

$$C(n) \approx C^*(n) \approx 1.386 \log_2 n$$

$$C(n) \approx C^*(n) \approx 1.386 \log_2 n$$

При поиске в случайном БДП

*среднее* число сравнений всего лишь на **39 %** больше, чем в идеально сбалансированном дереве.

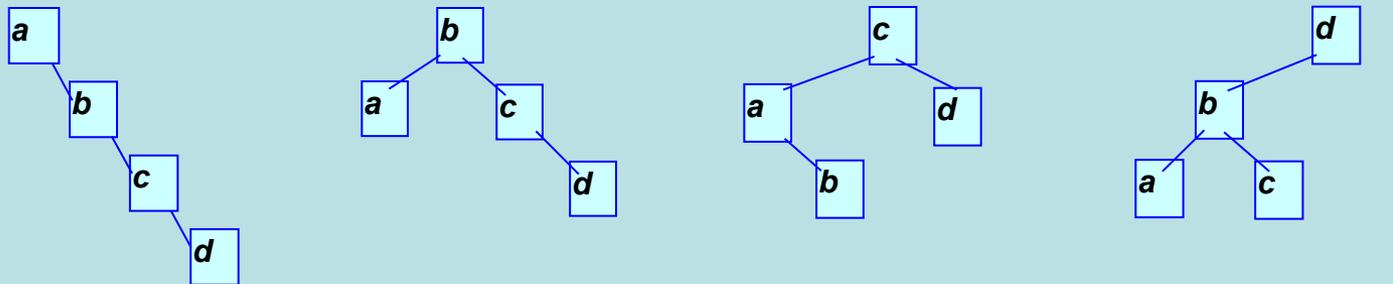
Это отражает тот факт

(см. пример на прошлой лекции «**abcd**»), что среди случайных деревьев чаще встречаются *хорошо сбалансированные* (относительно «ровные») деревья, чем вырожденные.

# Операции вращения в БДП

- В любом БДП *горизонтальный порядок* узлов фиксирован\*, однако расположение узлов *по уровням дерева* зависит от способа его построения.
- Можно ли изменять относительное расположение узлов дерева *по вертикали*, сохраняя при этом инвариант дерева поиска?

\* перечисление узлов БДП «слева направо», порождаемое ЛКП-обходом, дает упорядоченную последовательность.



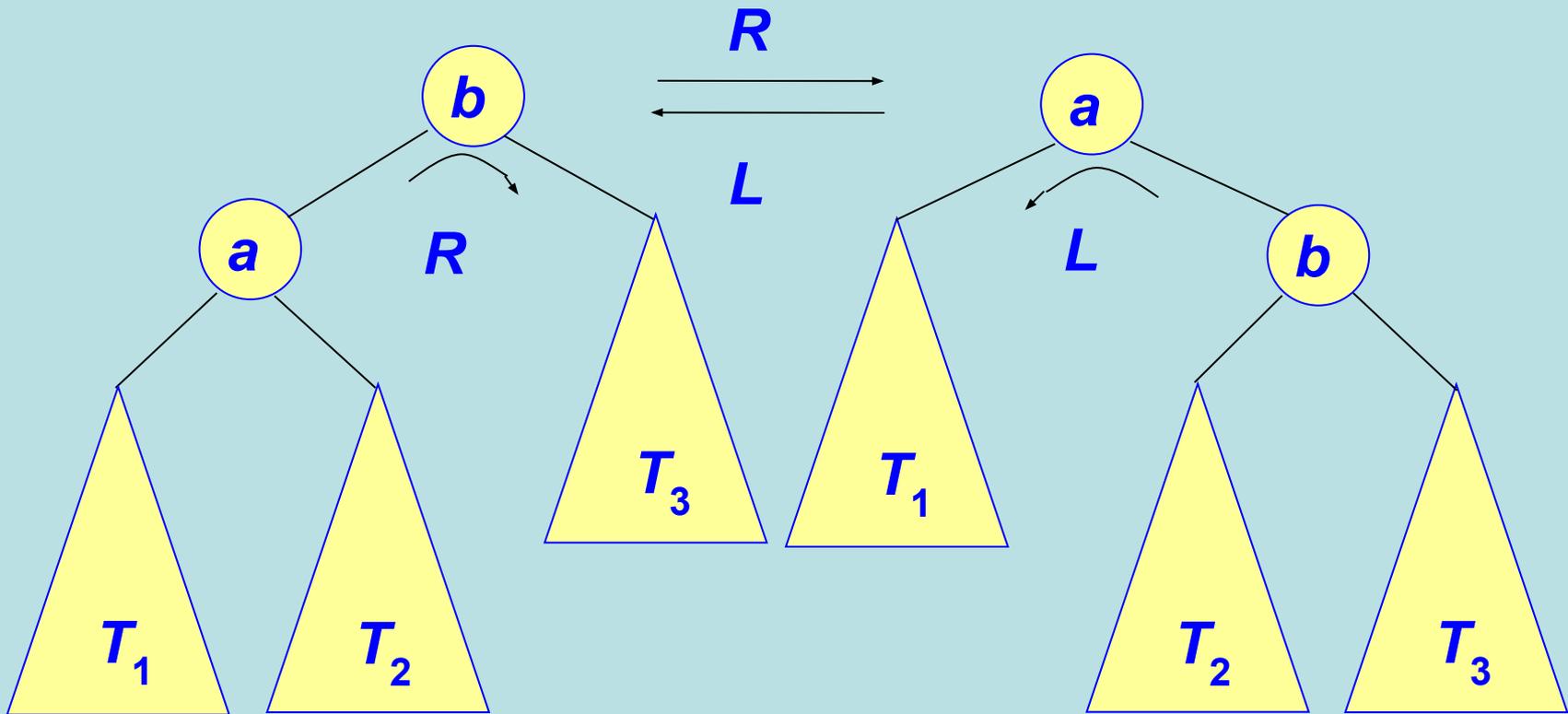


Рис. Операции правого  $R$  и левого  $L$  вращений с перекреплением в БДП

ЛКП-обходы дерева до и после операции вращения совпадают, а именно: (ЛКП( $T_1$ ),  $a$ , ЛКП( $T_2$ ),  $b$ , ЛКП( $T_3$ )).

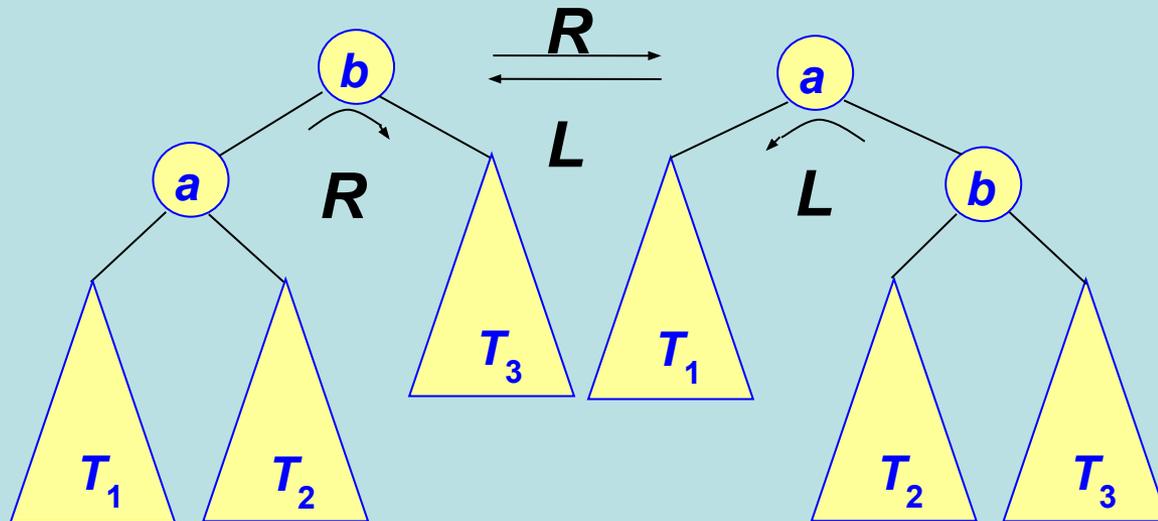
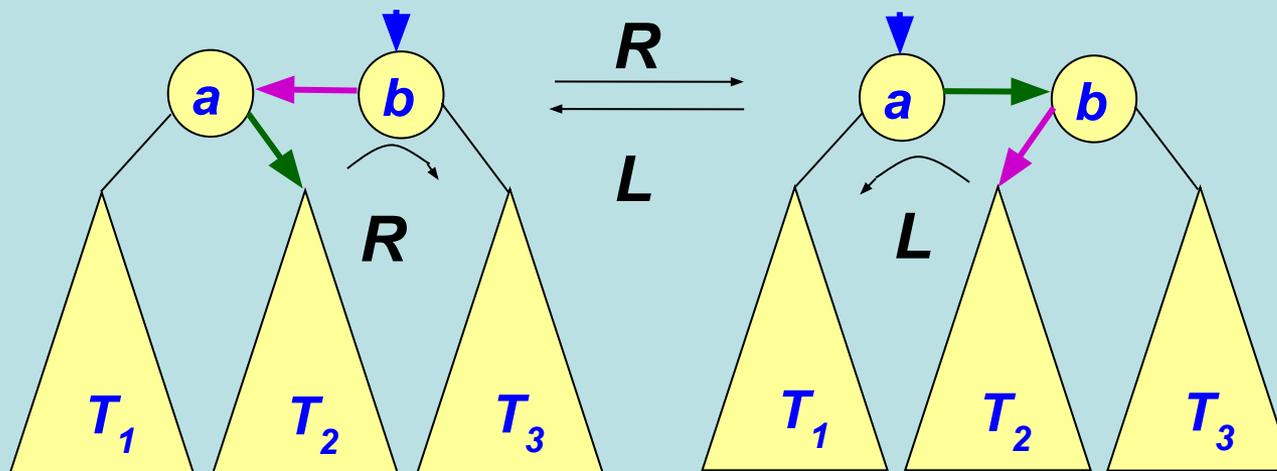


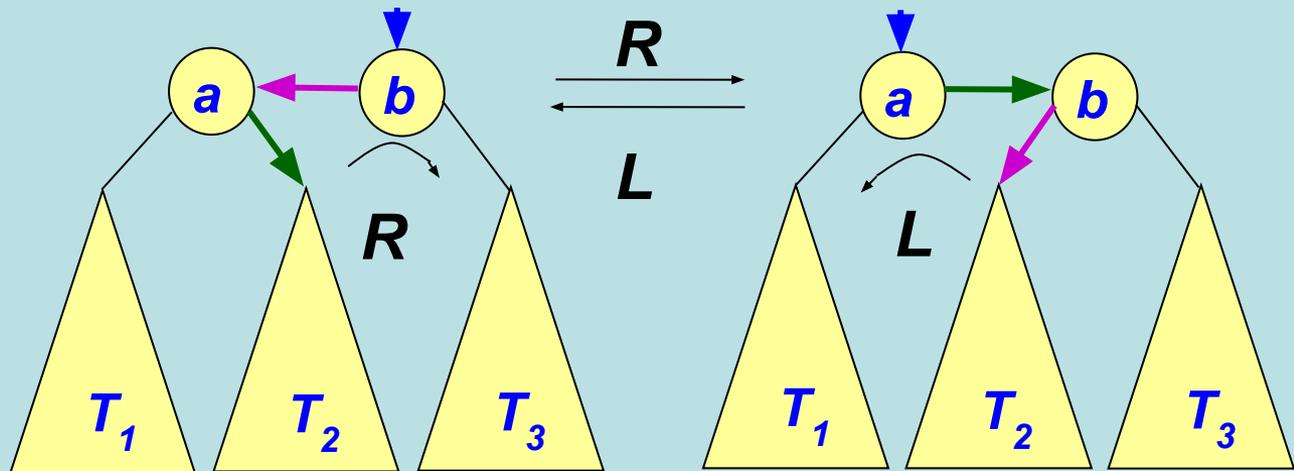
Рис. Операции правого  $R$  и левого  $L$  вращений с перекреплением в БДП

↓ на корень

↓ на правое

↓ на левое





```
void rotateR (binSTree &t)
```

```
  //b - не пусто
```

```
  { binSTree x;
```

```
    if (t==NULL){cout <<  
"rotateR(NULL)" << endl; }
```

```
    else {
```

```
      x = t->lt;
```

```
      t->lt = x->rt;
```

```
      x->rt = t;
```

```
      t = x;
```

```
    }
```

```
  }
```

```
void rotateL (binSTree &t)
```

```
  //b - не пусто
```

```
  { binSTree x;
```

```
    if (t==NULL){cout <<  
"rotateL(NULL)" << endl; }
```

```
    else {
```

```
      x = t->rt;
```

```
      t->rt = x->lt;
```

```
      x->lt = t;
```

```
      t = x;
```

```
    }
```

```
  }
```

# Случайные бинарные деревья поиска с рандомизацией

В некоторых случаях полезно при добавлении нового узла сделать так, чтобы этот узел стал *корнем* БДП.

1. Построенные таким образом БДП имеют некоторые полезные свойства.
2. Операция *вставки в корень* будет использоваться в модификации случайных БДП, рассмотренной далее.

# Вставка в корень БДП

## Рекурсивный способ

1. Если вставляемый элемент больше корня БДП, то его место в правом поддереве.  
Поэтому *сначала* рекурсивно *вставим* этот элемент в качестве корня правого поддерева, *а затем* с помощью левого вращения *поднимем его в корень* дерева.
2. Если вставляемый элемент меньше корня БДП, то его место в левом поддереве.  
Поэтому *сначала* рекурсивно *вставим* этот элемент в качестве корня левого поддерева, *а затем* с помощью правого вращения *поднимем его в корень* дерева.

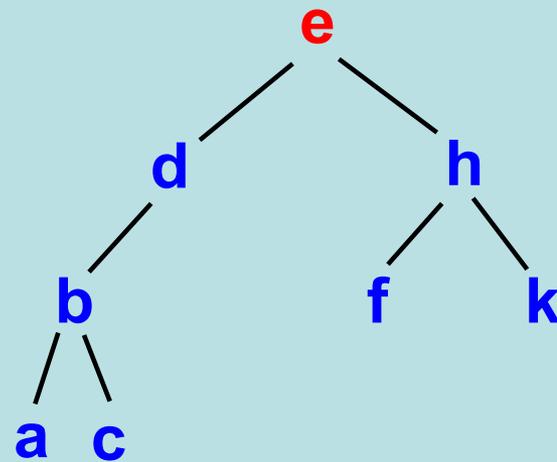
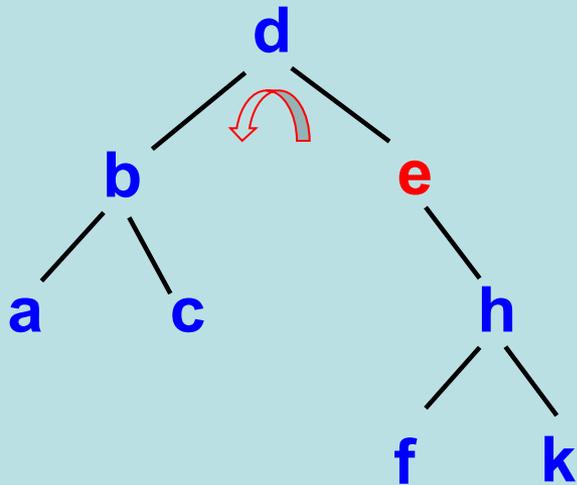
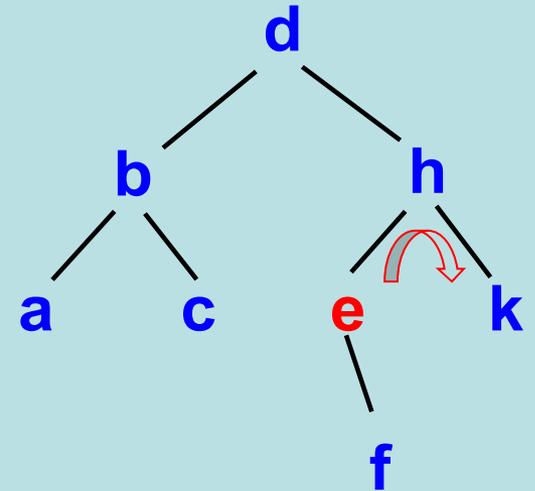
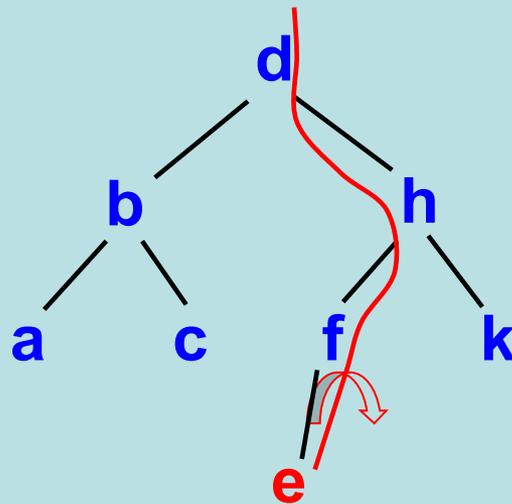
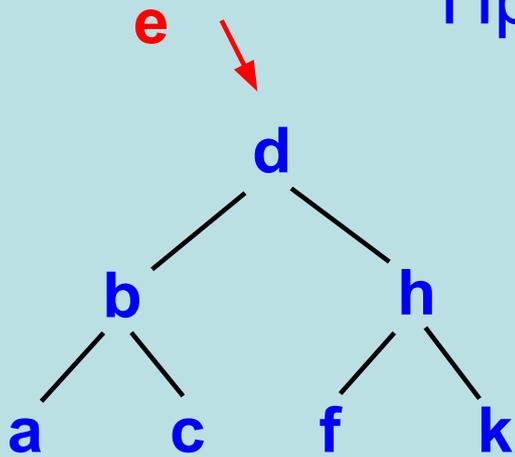
# // вставка в корень

bst4.cpp

```
void insInRoot (binSTree &b, base x)
{if (b==NULL) {
    b = new node;
    if ( b != NULL) {
        b ->info = x;
        b ->count = 1;
    }
    else {cerr << "1 Memory not enough\n"; exit(1);}
} else
if (x < b->info) {insInRoot (b->lt, x); rotateR (b);}
else if (x > b->info) {insInRoot (b->rt, x); rotateL (b);}
else b->count++;
}
```



# Пример вставки в корень



*Здесь остановились 14 октября*

## Применение вставки в корень

В некоторых задачах частота обращений к поиску с некоторым значением ключа возрастает после добавления этого ключа в БДП.

Например, при обработке финансовых транзакций, когда актуальность старых данных постепенно уменьшается, а актуальность новых («свежих») данных велика.

## Рандомизация в случайных БДП

Идея модификации случайных БДП - чередование обычной вставки в дерево поиска и вставки в корень.

Чередование происходит **случайным (рандомизированным)** образом с использованием компьютерного генератора псевдослучайных чисел (например, функции **rand()** в C++).

Цель такого чередования - сохранить хорошие свойства случайного БДП в среднем и исключить (сделать маловероятным) появление «худшего случая» (поддеревьев большой высоты).

## Рандомизированная вставка элемента $x$ в БДП

Пусть в дереве имеется  $n$  узлов. Считаем, что после добавления еще одного узла любой узел (теперь из  $n + 1$  узла) с равной вероятностью может быть корнем дерева.

Пусть абстрактная булевская функция  $chance(k)$  выдает значение  $true$  с вероятностью  $1/k$ .

Тогда, если  $chance(n + 1) = true$ , то осуществим вставку в корень, иначе рекурсивно используем рандомизированную вставку в левое или правое поддереве в зависимости от значения ключа  $x$ .

## Набросок процедуры рандомизированной вставки в БДП

```
void randomInsert (binSTree &b, base x)
{   if (b==NULL) {
        b = new node;
        if ( b != NULL) {
            b ->info = x; b ->count = 1; b ->number = 1;
            return;
        }
        else {cerr << "1 Memory not enough\n"; exit(1);}
    }
    if (chance(b ->number + 1)) {insInRoot (b, x); return;}
    if (x < b->info) randomInsert (b->lt, x);
    else randomInsert (b->rt, x);
    b ->number ++;
}
```

- Здесь остановились!!!

## Здесь предполагалось, что

- в каждом узле дерева в поле *number* хранится количество узлов поддерева, корнем которого является данный узел
- осуществляется «чистая» вставка, т. е. заранее известно, что элемент *x* в дереве отсутствует
- процедуры *insInRoot*, *rotateR* и *rotateL* изменены так, что они при своей работе правильно корректируют поле *number* в соответствующих узлах дерева

## Выводы

Оказывается [16], что

случайные БДП с рандомизацией имеют в среднем примерно такие же характеристики, что и случайные БДП, однако в тех случаях, когда нарушается предположение о случайном порядке вставки элементов в БДП, т. е. когда характеристики случайного БДП значительно ухудшаются, деревья с рандомизацией сохраняют свои хорошие характеристики за счет «принудительной» рандомизации своей структуры.

## Выводы

Можно сказать, что в просто случайных БДП степень «случайности» регулируется порядком элементов входной последовательности узлов, а в случайных БДП с рандомизацией - генератором псевдослучайных чисел.

# Примечание

Термин «в среднем» имеет разный смысл  
в случайных БДП  
и в БДП с рандомизацией.

*Среднее по разным «ансамблям»:*

- в случайных БДП - по разным входным последовательностям
- в БДП с рандомизацией - для одной входной последовательности по значениям *Random*

КОНЕЦ ЛЕКЦИИ

КОНЕЦ ЛЕКЦИИ