Лекция 19

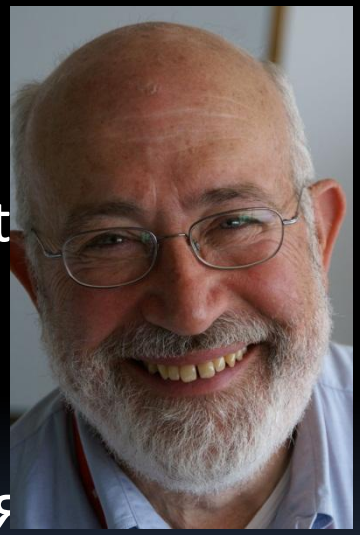
B И КРАСНО-ЧЕРНЫЕ ДЕРЕВЬЯ

План лекции

- B деревья
  - Определение
  - Вставка и удаление вершины
- Красно-черные деревья
  - Определение
  - Вставка вершины
  - Сравнение в АВЛ деревьями
  - Связь КЧ и B деревьев

B деревья

- B деревья – сбалансированные деревья для быстрого доступа к информации на устройствах с прямым доступом
- Рудольф Бэйер (R. Bayer)
- Эдвард МакКрейт (E. McCreight)
- ~1970

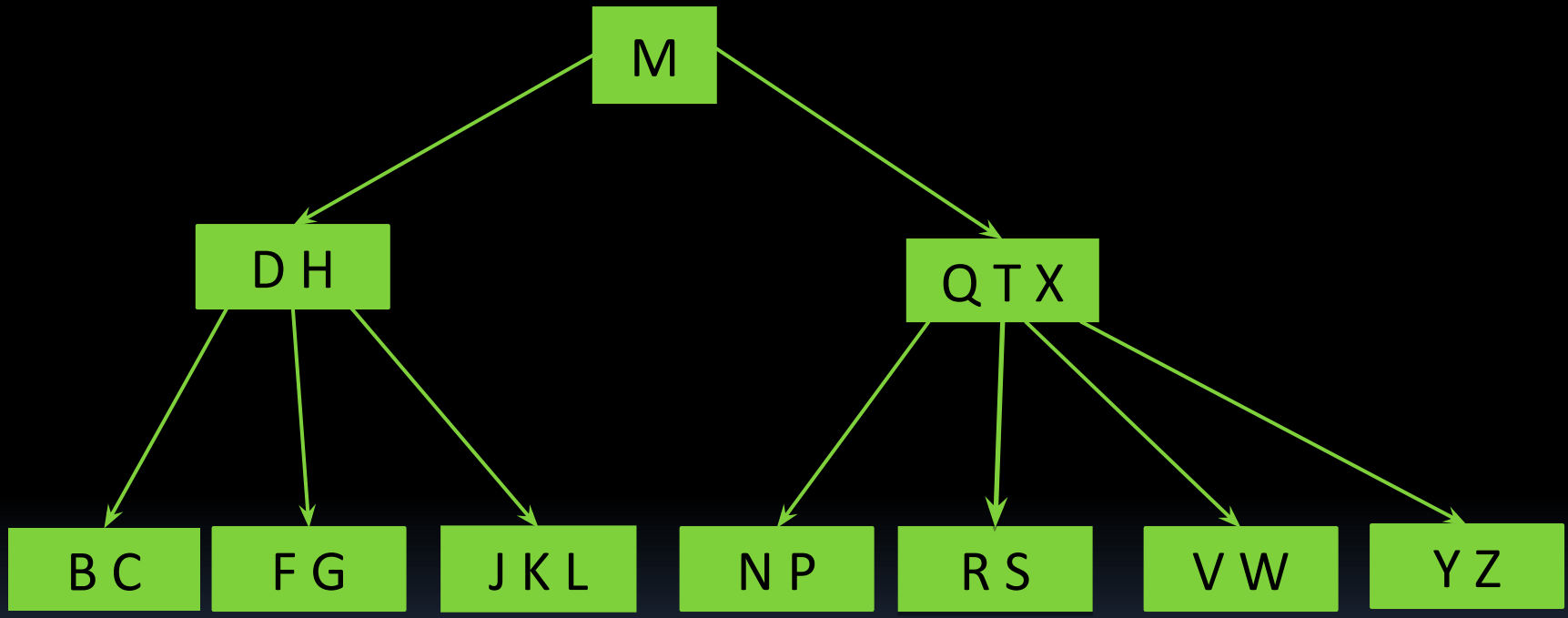


- Страничная организация памя
- Файловые системы, например, Windows NTFS
- Обработка больших массивов данных

B деревья

- Все листья находятся на одной глубине
- Существует целое число t >= 2 -- степень B дерева, что
  - Каждая вершина кроме корня имеет от t до 2*t прямых потомков
  - Корень имеет от 2 до 2*t прямых потомков
- Каждая вершина хранит ключи, разграничивающие ключи, хранящиеся в ее поддеревьях
  - Сколько ключей может хранить вершина B дерева? Корень B дерева?
- Все ключи B дерева принадлежат одному линейно упорядоченному множеству

Пример B дерева



- Какая степень этого B дерева?

Пример использования – страничная организация памяти

- Лист B дерева = физический блок памяти
  - Физическая страница памяти или кластер диска
- Совокупность внутренних вершин B дерева = «таблица трансляции адресов»
  - Хранится в специальных регистрах процессора и специальной области памяти
- Ключи = логические адреса нулевых байтов физических блоков

Пример использования – управление страничной памятью

- Поиск физического блока, хранящего байт с логическим адресом A
  - Он же «трансляция логического адреса A в физический адрес»
  - Поиск листа B дерева с ключом, равным остатку от деления A на размер физического блока
- Добавление нового физического блока в пространство логических адресов
  - Вставка листа в B дерево

B деревья -- определения

- Вершина B дерева называется полной, если число ее непосредственных потомков равно удвоенной степени B дерева
- B дерево степени 2 называется 2-3-4 деревом
  - Каждая внутренняя вершина кроме корня имеет 2, 3 или 4 потомка

Теорема о высоте B дерева

- Для любого B дерева высоты h и минимальной степени $t \geq 2$, хранящего $n \geq 1$ ключей, выполнено неравенство

$$h \leq \log_t \frac{n+1}{2}.$$

- Высота B дерева с n-вершинами есть O(log n), но основание логарифма для B дерева гораздо больше, что примерно в log t раз сокращает количество обращений к диску

- Что такое глубина вершины?
- Что такое высота (уровень) вершины?

Пример определения на Си

```
typedef struct b_tree_t {
   int n;           // количество ключей
   int *key;           // key[0]<key[1]<…<key[n-1]
   struct b_tree_t **child; // непосредств. потомки
} b_tree;
```

Обозначим x->child[i] через $C_i(x)$

Поиск в B дереве

- Дано B дерево и ключ K
- Найти вершину, содержащую K

- В каждой вершине x сравниваем K с n(x) ключами из x и продолжаем поиск в соотв. n(x)+1 потомков

Алгоритм поиска

- Поиск в B дереве похож на поиск в двоичном дереве
- Разница в том, что в вершине x мы выбираем один вариант из n(x)+1, а не из двух
- Процедура поиска получает на вход указатель x на корень поддерева и ключ k, который мы ищем в этом поддереве
- Если процедура обнаруживает в дереве ключ k, то она возвращает пару (y, i), где y - вершина, i - порядковый номер указателя, для которого $key_i(y) = k$
- Иначе операция возвращает NULL

Поиск в B дереве

```
B_tree_search(x,k)
{
  int i = 0;
  while (i < n(x) && k > key_i(x)) i++;
  if (i<n(x)&&k==key_i(x)) return(x,i);
  if (leaf(x)) return NULL;
  else
  {
   return B_tree_search(C_i(x),k);
  }
}
```

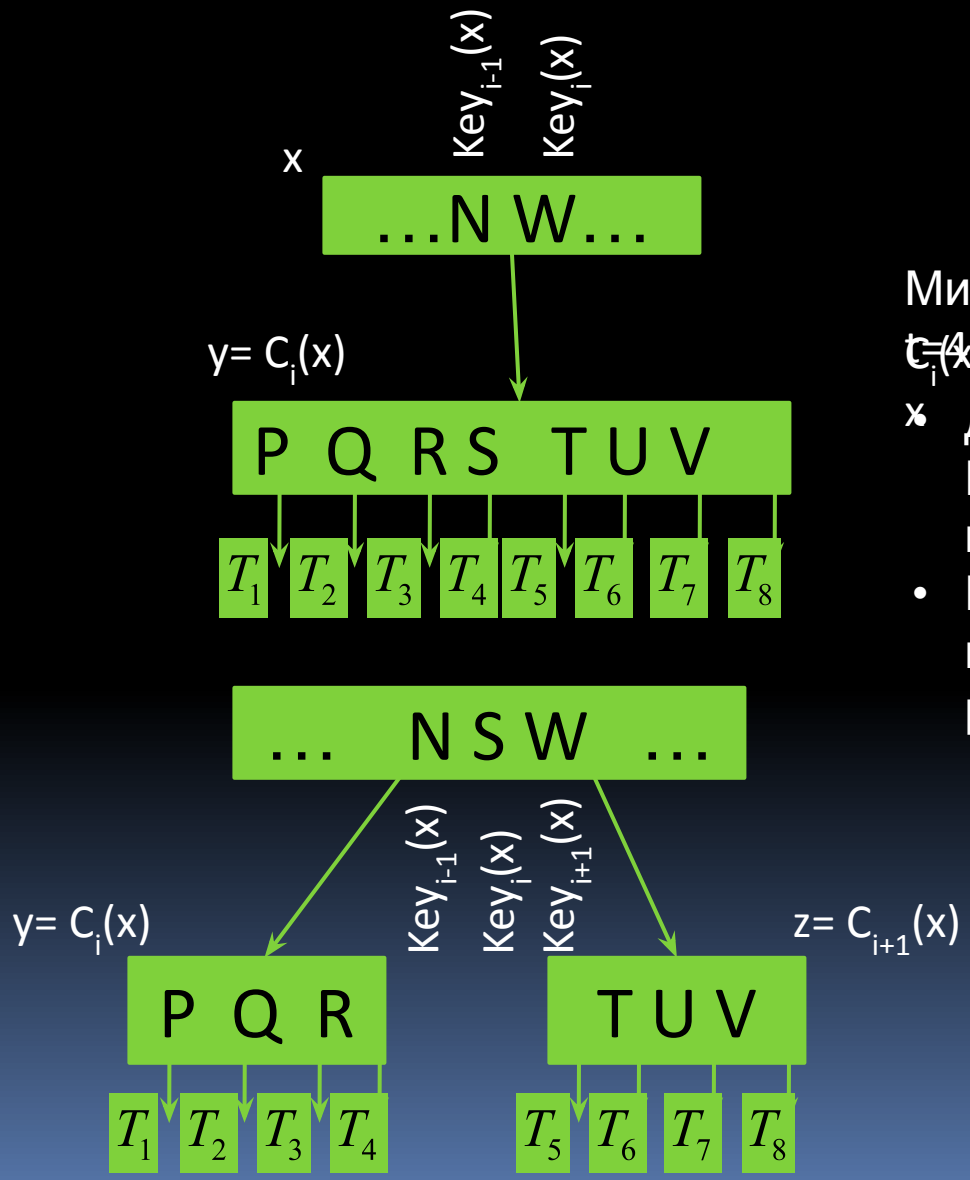
Добавление элемента в B дерево

- Процедура B_tree_insert (T, k) – добавляет элемент k в B дерево T, пройдя один раз от корня к листу
- На это требуется время O(h), если высота дерева равна h
- По ходу дела с помощью процедуры B_tree_Split_child разделяются вершины, которые являются полными и которые имеют неполного родителя
- В результате, доходим до неполного листа, куда и добавляем новый элемент

Разбиение вершины B дерева

- Добавление элемента в B дерево – более сложная операция по сравнению с бинарными деревьями
- Ключевым местом является разбиение полной (с 2t-1 ключами) вершины на две вершины, имеющие по t-1 ключей в каждой
- При этом ключ-медиана $key_{t1}(y)$ отправляется к родителю x вершины y и становится разделителем двух полученных вершин
- Это возможно, если вершина x неполна
- Если y – корень, то высота дерева увеличивается на 1

Разбиение вершины B дерева



Минимальная степень $t=4$

$C_i(x)$- указатель на i-го ребенка в x

- Делим вершину y на две: y и z
  Ключ медиана *S* вершины *y* переходит к ее родителю *x*
- Ключи, больше *S*, переписываются в нового ребенка z вершины x

```
// Входные данные
// неполная внутренняя вершина x, число i и
// полная вершина y: y = C_i(x)
// (считаем, что x и y уже в ОП)
B_tree_SPLIT_Child (x, i, y)
{
   // z – создать узел;(файл, отвести место)
   leaf(z) = leaf(y);
   n(z) = t-1;
   for(j = 0; j < t-1; j++) key_j(z) = key_j+t(y);
   if (!leaf(y))
    for(j = 0; j < t; j++) C_j(z) = C_j+t(y);
   n(y) = t-1;
```

```
    for (j = n(x)+1; j ≤ i; j--) C_{j+1}(x) = C_j(x);
    C_{i+1}[x] = z;
    for (j = n(x); j ≤ i; j--) key_{j+1}(x) = key_j(x);
    key_i(x) = key_j(y);
    n(x) = n(x)+1;
    //  Переписать вершины: y, z, x
}

// Вершина y имела 2t детей
// после разбиения в ней осталось t детей
// Остальные t детей стали детьми новой вершины z
```

```
// добавление в дерево с корнем
B_tree_insert (T, k)
{
   r = root(T);
   if (n(r)== 2t-1) {
    // s = выделяем память/файл для нового узла;
    root(T)= s;     //он становится корнем      leaf(s)= 0;
    n(s)= 0;
    C1(s)= r;
    B_tree_split_child (S, 1, r);
    B_tree_insert_nonfull (s, k);//добавляет
   } else
    // элемент в k в поддерево с корнем в неполной вершине
    B_tree_insert_nonfull (r, k);
}
```

Добавление элемента в неполную вершину

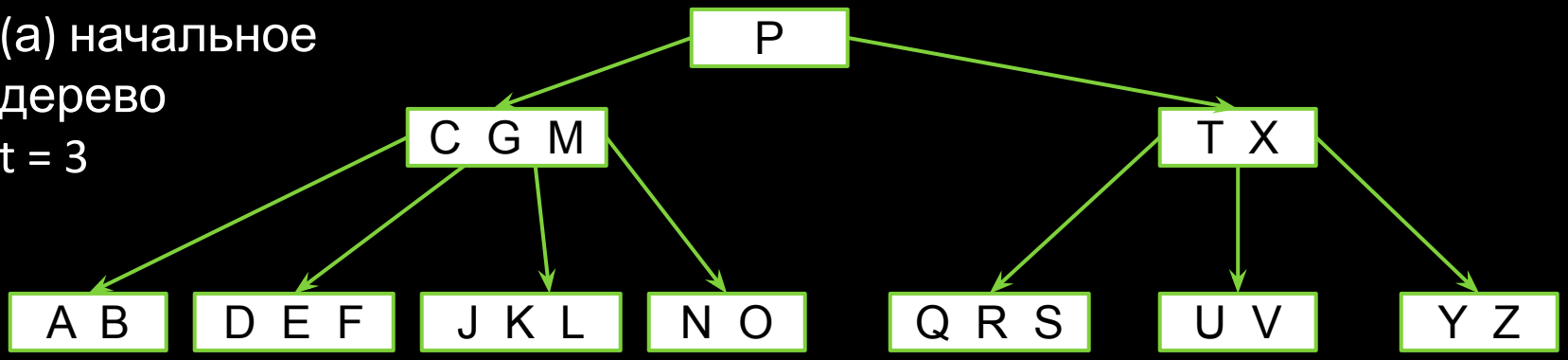
- B_tree_insert_nonfull (r, k) рекурсивно вызывает себя, при необходимости, выполнив разделение
- Если вершина x – лист, то ключ k в него добавляется
- Иначе k добавляется к поддереву, корень которого является ребенком x
- Для этого определяется нужный ребенок вершины x
- Если ребенок – полная вершина, то он разделяется

```
B_tree_insert_nonfull(x, k)
{
   i = n(x);
   if (leaf(x)) { // ключ вставляется в лист
   while (i ≥ 0 && k < key_i(x)){
       key_{i+1}(x)=key_i(x);
       i--;
   }
       key_{i+1}(x) = k;
   n(x) = n(x)+1;
   } else {
   // поиск нужного ребенка
   while( i ≥ 0 && k < key_i(x)) i--;
```

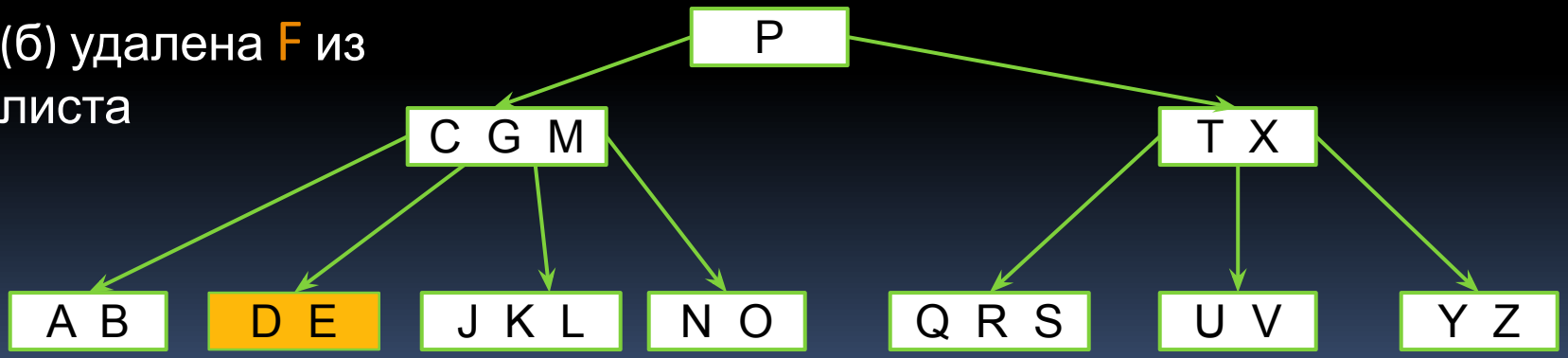
```
        i = i+1;
        if (n(C_i(x)) == 2t-1) {
        // если ребенок–полная вершина
        B_tree_split_child (x, i, C_i(x));
        // разделение
           if (k > key_i(x)) i = i+1;
        }
        B_tree_ insert_nonfull (C_i(x), k);
}
```

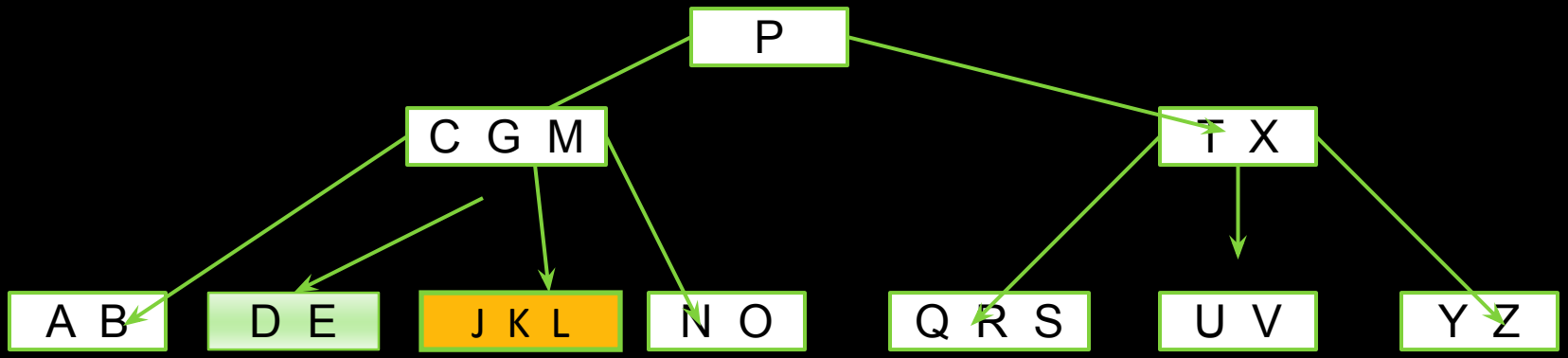
Удаление элемента из B дерева

(а) начальное дерево
t = 3

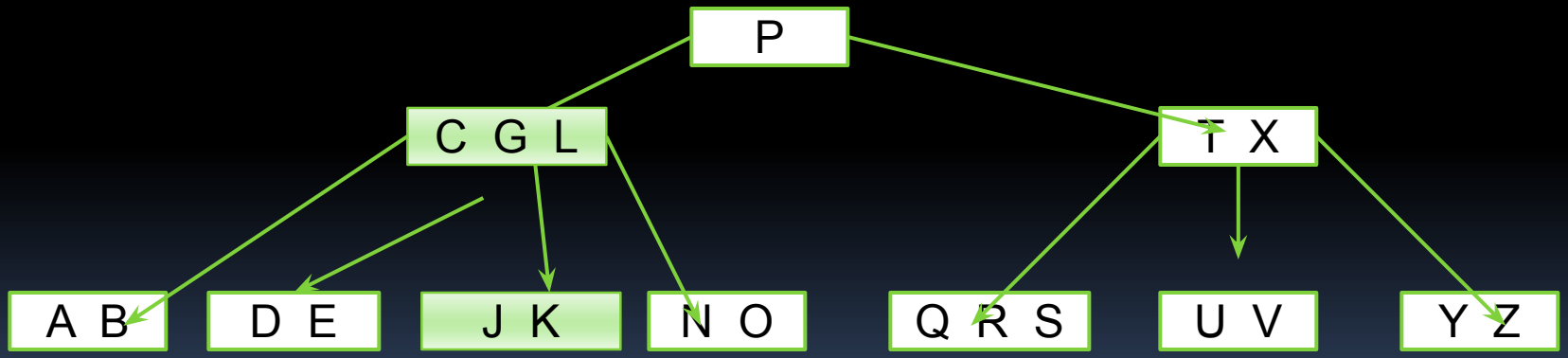


(б) удалена F из листа

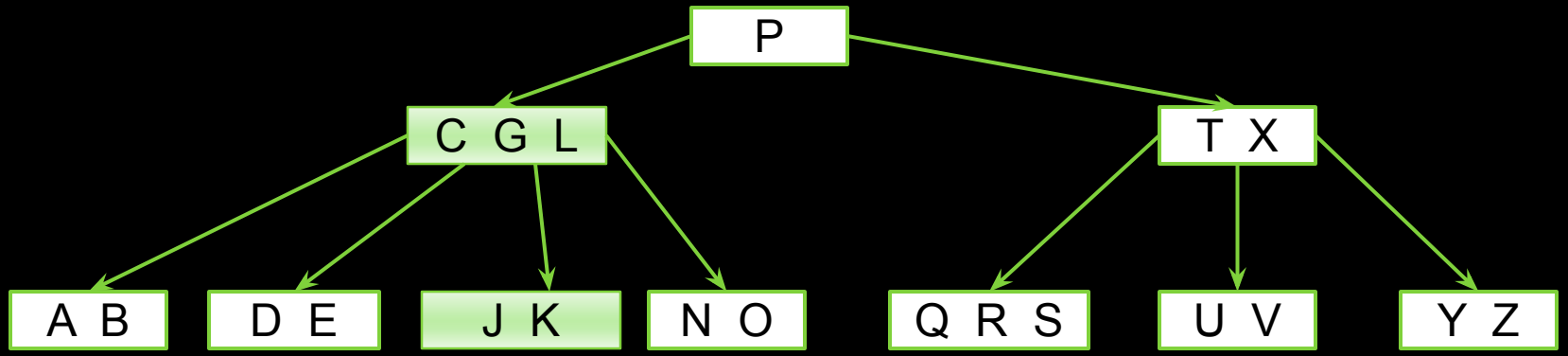




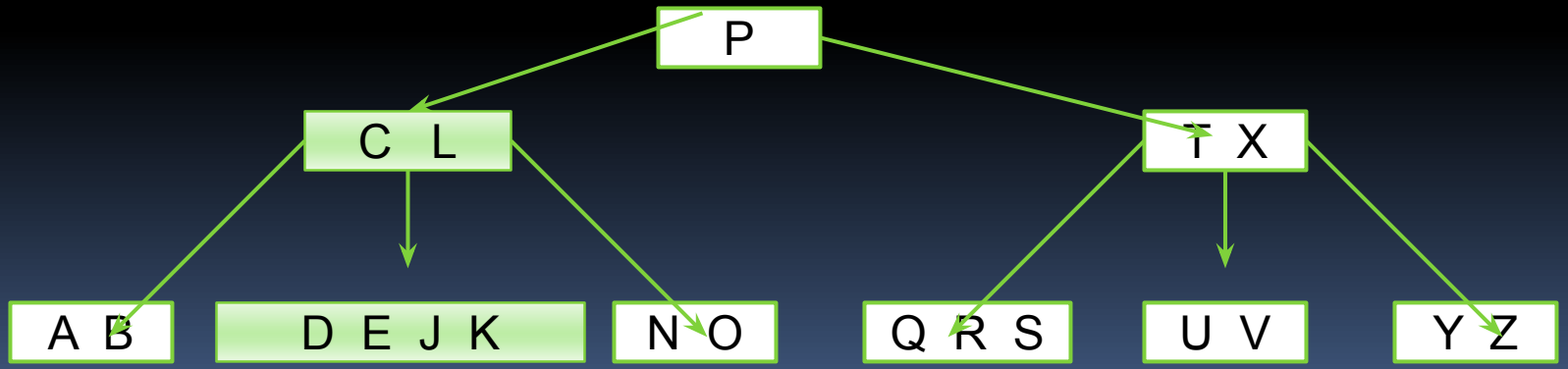
(в) удалена M из внутренней вершины, ребенок которой имеет не менее t элементов

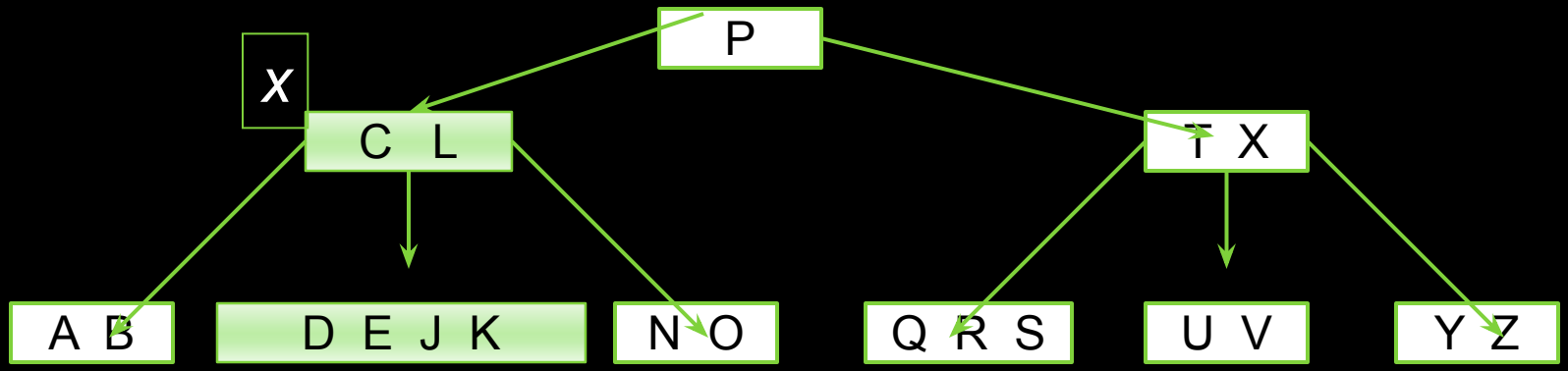


Если ребенок, следующий за удаляемым ключом, имеет не менее *t* элементов, поступаем аналогично (в)

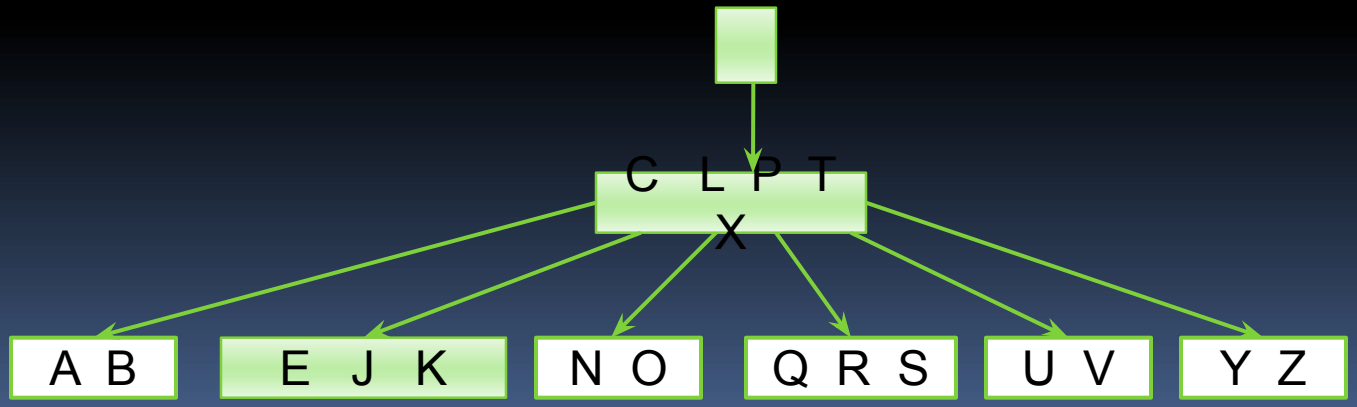


(г) удалена G, ее дети имеют по t-1 ключу

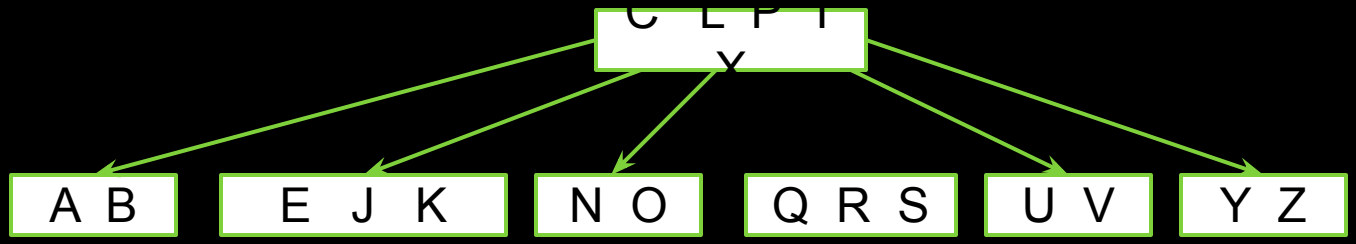




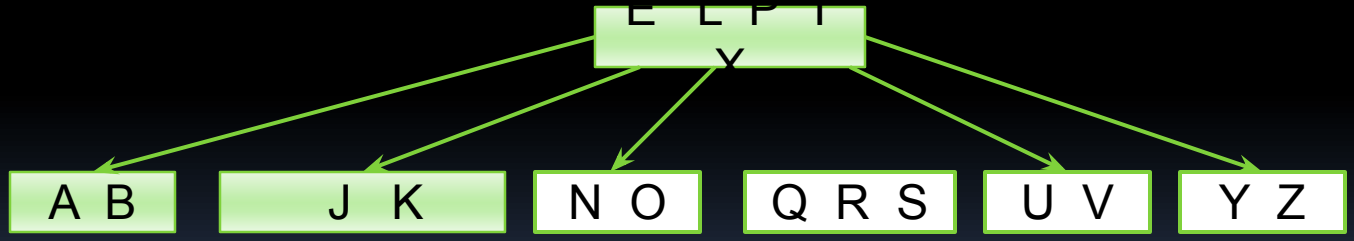
(д) удалена D, в вершине x нет ключа D и t = 2



(д’) уменьшение высоты дерева



(е) удалена
C



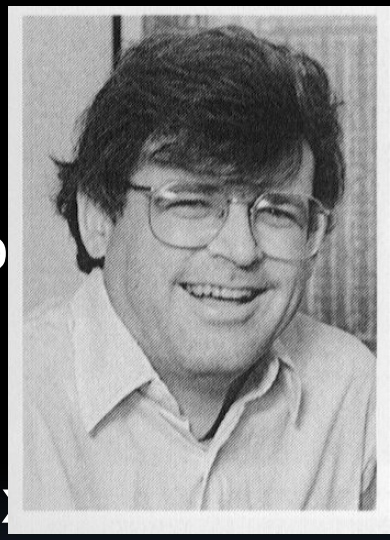
- B деревья
  - Определение
  - Вставка и удаление вершины
- Красно-черные деревья
  - Определение
  - Вставка вершины
  - Сравнение в АВЛ деревьями
  - Связь КЧ и B деревьев

Красно-чёрное дерево

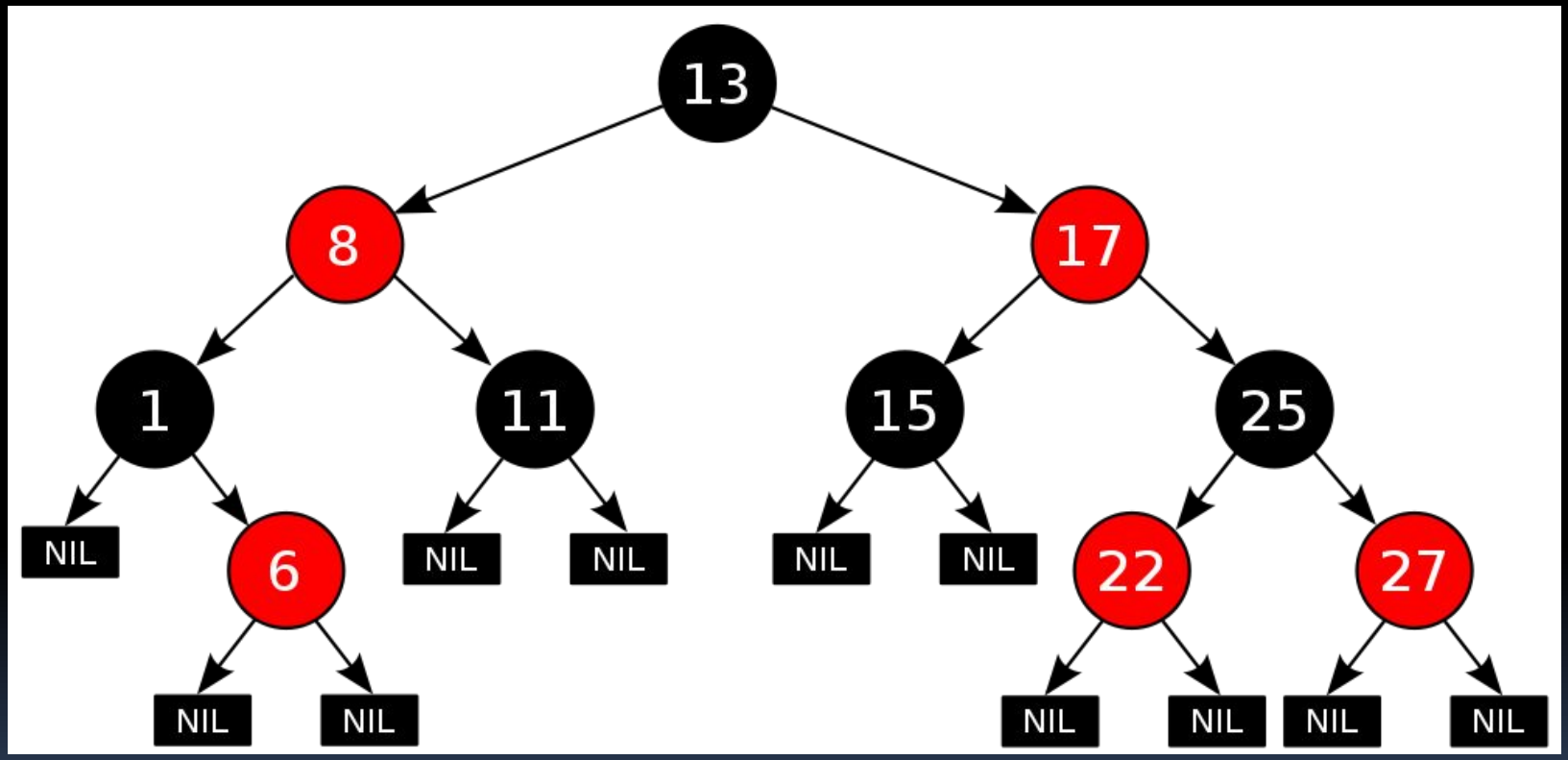
Rudolf Bayer 1972
Симметричные двоичные B деревья
Леонидас Гибас и Роберт Седжвик 1978
КЧ деревья

Красно-чёрное дерево – это дерево двоичного поиска, обладающее следующими КЧ свойствами

1. Все листья чёрные и не содержат данных
2. Все потомки красных узлов чёрные – нет двух красных узлов подряд
3. На всех путях от корня к листьям число чёрных узлов одинаково и равно чёрной высоте дерева



Пример КЧ дерева (Википедия)



Высота и число узлов в КЧ дереве

1. Если h - чёрная высота дерева, то количество узлов не менее 2^{h-1}
   - Почему?
   - Что останется от КЧ дерева, если красные вершины "втянутся" в черных предков?
   - Как выглядит двоичное дерево, у которого все листья находятся на одной глубине?
2. Если h - высота дерева, то количество узлов не менее $2^{(h-1)/2}$
3. Если количество узлов N, высота дерева не больше $2\log_2 N + 1$

Вставка узла в КЧ дерево -- схема

Чтобы вставить узел

- Находим двоичным поиском место, куда его следует добавить
- Новый узел добавляем как красный узел с двумя чёрными листьями
- После этого восстанавливаем красно-чёрные свойства -- перекрашиваем узлы и поворачиваем поддеревья, если необходимо

Вставка узла -- лист

Вставка красного узла с двумя черными NULL-потомками

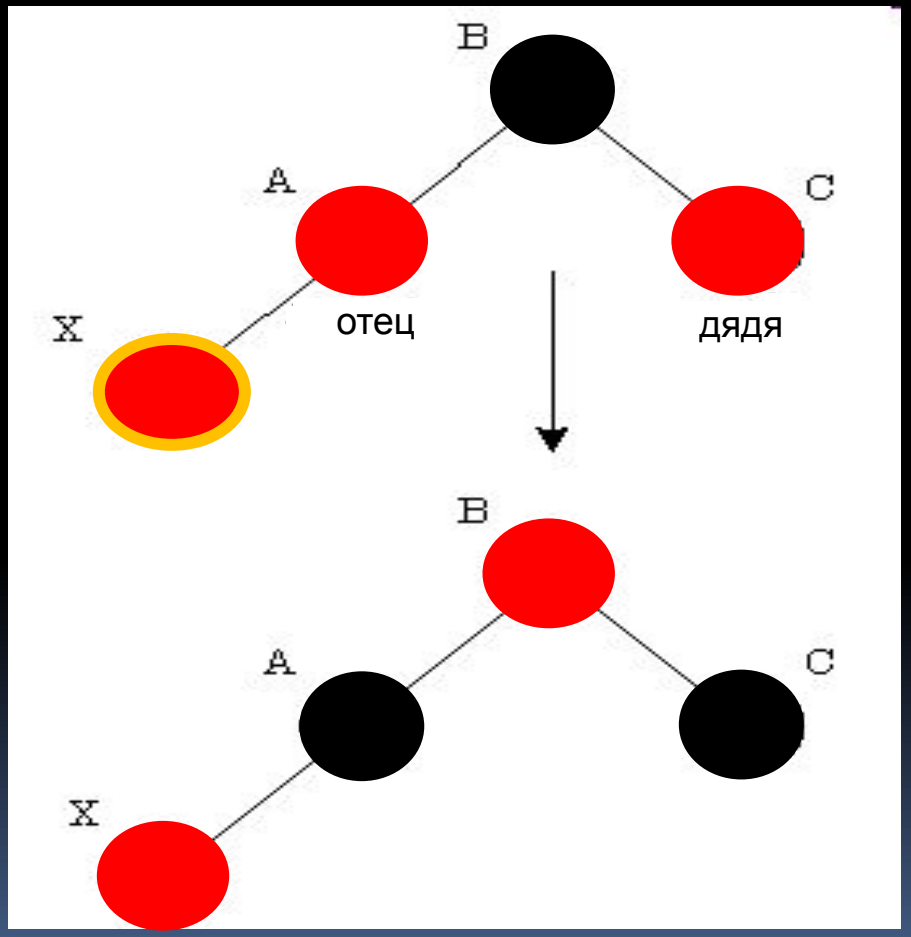
1. Все листья чёрные – сохраняется
2. Все потомки красных узлов чёрные – нет двух красных узлов подряд – может нарушиться
3. На всех путях от корня к листьям число чёрных узлов одинаково – сохраняется

Вставка узла – красные отец и дядя

- Цвет отца и дяди меняется на черный
- Цвет деда меняется на красный
- КЧ свойства (возможно) нарушились на 2 уровня выше -- повторяем уже для деда узла
- В самом конце корень красим в черный цвет
  - Если он был красным, то увеличится черная высота дерева

Вставка узла – красные отец и дядя

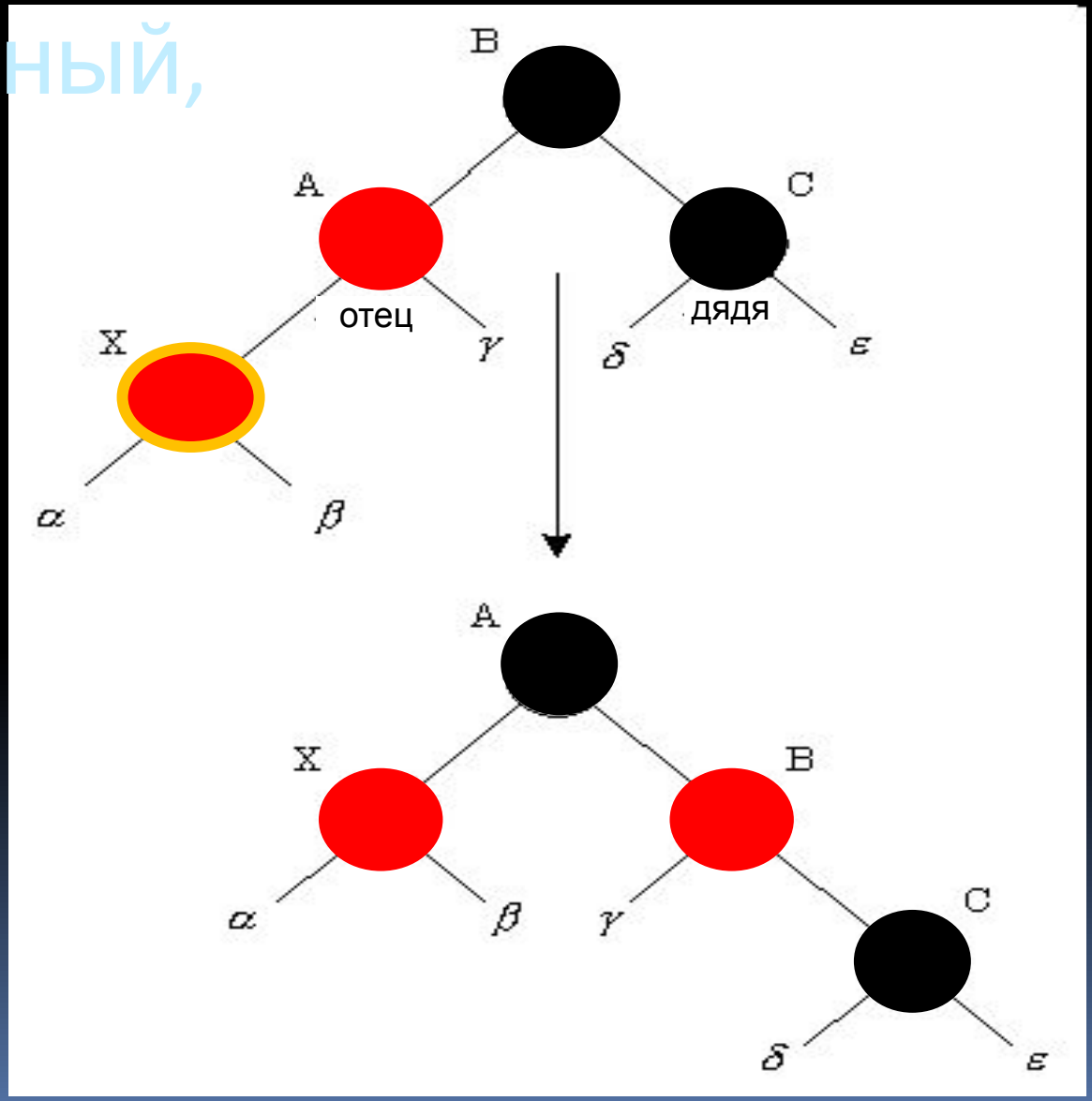
- Красно-красный конфликт устраняется перекрашиванием
- После перекраски нужно проверить деда нового узла (узел B), поскольку он может оказаться красным



Вставка узла – отец красный, дядя черный

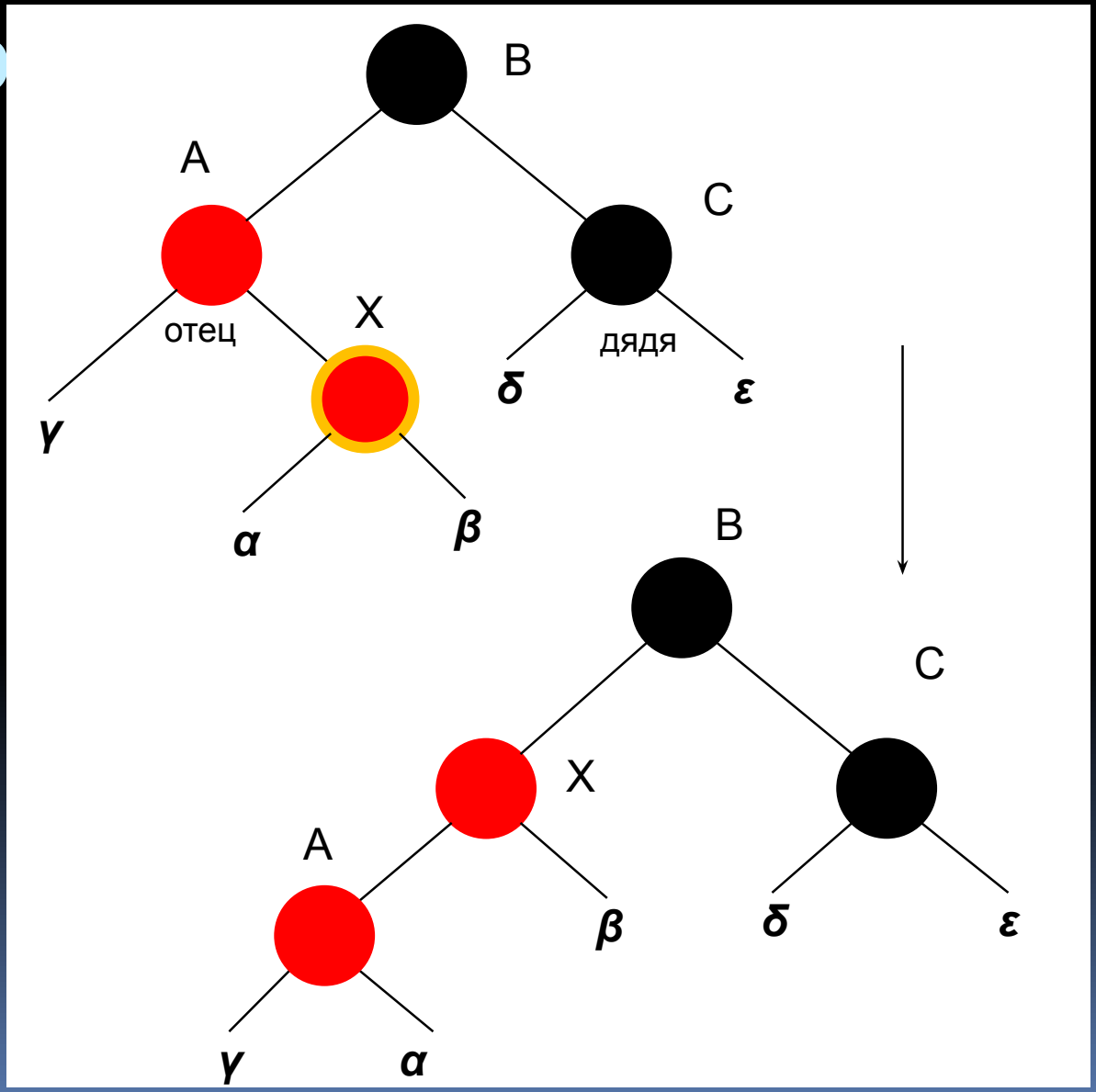
- Новый узел -- левый сын своего отца
  - Цвет отца меняется на черный
  - Цвет деда меняется на красный
  - Дерево поворачивается направо вокруг отца нового узла
  - КЧ свойство восстановлено, вставка закончена
- Новый узел -- правый сын своего отца
  - Дерево поворачивается налево вокруг отца нового узла
  - Далее см. пред. случай

Вставка узла – отец красный, дядя черный, левый сын



Вставка узла – отец красный, дядя чер правый сын

Далее как на пред. слайде



Сравнение с АВЛ деревом

Обозначим N(h) = минимальное число узлов в дереве высоты h

- N(h) для АВЛ дерева
  - N(h) = N(h − 1) + N(h − 2) + 1, N(0) = 1, N(1) = 2
  - N(h) растёт как последовательность Фибоначчи – почему?
  - Следовательно, N(h) = Θ(λ^h), где $\lambda = (\sqrt{5}+1)/2 \approx 1,62$
- N(h) для красно-чёрного дерева
  - Свойство 3 красно-чёрных деревьев ==>

$$N(h) \geq 2^{(h-1)/2} = \Theta(\sqrt{2}^h)$$

- При том же количестве узлов КЧ дерево м. б. выше АВЛ дерева, но не боле$\log \lambda / \log \sqrt{2} \approx 1,388$ раз

Сравнение с АВЛ деревом

- Поиск и вставка для АВЛ дерева м.б. быстрее, чем для КЧ дерева
  - Высота КЧ дерева м. б. на 40% больше высоты АВЛ дерева при одинаковом числе узлов
- Удаление из КЧ дерева м. б. быстрее, чем из АВЛ дерева
  - КЧ дерево – достаточно 2 или менее поворотов
  - АВЛ дерево – возможно понадобится поворот в каждом узле на пути от удаляемого листа до корня

Связь КЧ и B деревьев

- B деревья с t=2 можно перестроить в КЧ деревья так
  - Каждый узел окрашен либо в красный, либо в чёрный цвет
  - Вершина с двумя потомками черная и переносится в КЧ дерево без изменений -- почему нет вершин с одним потомком?
  - Вершина с тремя потомками черная, первый потомок черный и присоединяется непосредственно, а другие два -- через соединительный красный узел
  - Вершина с четырьмя потомками черная, черные потомки присоединяются через два красных соединительных узла
- В исходном B дереве (так как оно сбалансировано) все пути от корня до любого листа имеют одинаковую длину
- По построению очевидно, что длина любого пути в КЧ дереве возрастает не более чем в два раза

Использование в библиотеке стандартных шаблонов C++ (STL)

- АВЛ-деревья 1961 -- первые сбалансированные деревья

- Красно-чёрные деревья 1978
- Библиотека стандартных шаблонов STL языка C++ использует сбалансированные деревья для реализации множества и ассоциативного массива

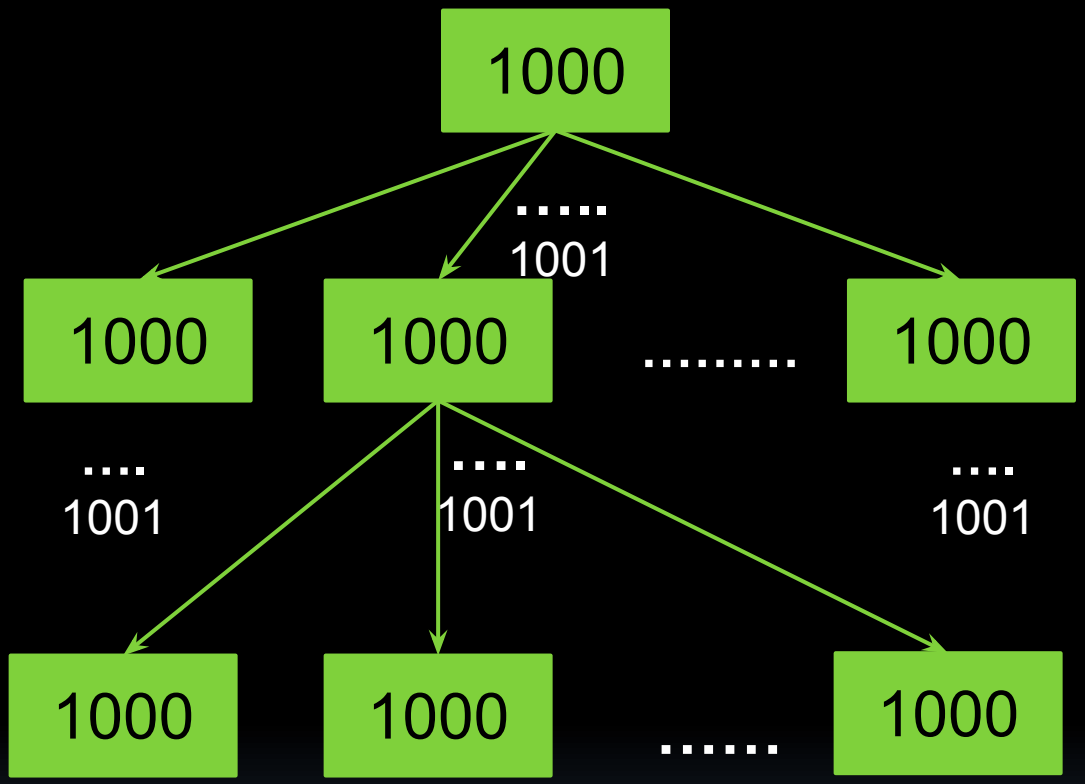
Кто автор STL и в какой стране он родился?

Заключение

- B деревья
  - Определение
  - Вставка и удаление вершины
- Красно-черные деревья
  - Определение
  - Вставка вершины
  - Сравнение в АВЛ деревьями
  - Связь КЧ и B деревьев

Удаление узла из КЧ дерева

- Если удаляемый узел красный все правила сохраняются и все прекрасно
- Если же удаляемый узел черный, требуется значительное количество кода, для поддержания дерева частично сбалансированным
- Как и в случае вставки в красно-черное дерево, здесь также существует несколько различных случаев



1 вершина –
1000 ключей

1001 вершина –
1001000 ключей

1 002 001 вершина
-1 002 001 000
ключей

При высоте 2 и размере страницы 8Кб это дерево содержит > миллиарда ключей и позволяет адресовать 8Тб данных

B деревья

- I am occasionally asked what the **B** in B-Tree means. I recall it as a lunchtime discussion that you never in your wildest dreams imagine will one day have deep historical significance. We wanted the name to be short, quick to type, easy to remember. It honored our employer, **B**oeing Airplane Company, but we wouldn't have to request permission to use the name. It suggested **B**alance. Rudolf **B**ayer was the senior researcher of the two of us. We had been admiring the elegant natural balance of AVL Trees, but for reasons clear to American English speakers, the name BM Tree was a non-starter. I don't recall one meaning standing out above the others that day. Rudolf is fond of saying that the more you think about what the **B** could mean, the more you learn about B-Trees, and that is good. (2012)

- У таких деревьев, как правило, только корень находится в ОП, остальное дерево – на диске
- Диск разбит на сектора (дорожки на сектора)
- Обычно записывают или считывают сектор целиком
- Время доступа, чтобы подвести головку к нужному месту на диске, может быть достаточно большим
- Как только головка диска установлена, запись или чтение происходит довольно быстро
- Часто получается, что обработка прочитанного занимает меньше времени, чем поиск нужного сектора
- Важно количество обращений к диску!

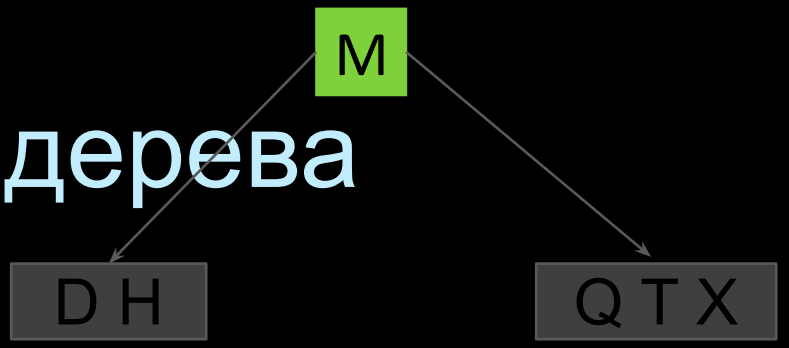
Определение B дерева 1/3

- В каждой вершине x хранятся
  - n - количество ключей, в данной вершине
  - сами ключи $k_0 \le k_1 \le \ldots \le k_{n-1}$ в неубывающем порядке
  - булевское значение leaf[x], истинное, если вершина x - лист
- Если x – внутренняя вершина, то она также содержит n(x)+1-указателей: C_0, C_1,…, $C_{n(x)}$ на ее детей

Определение B дерева 2/3

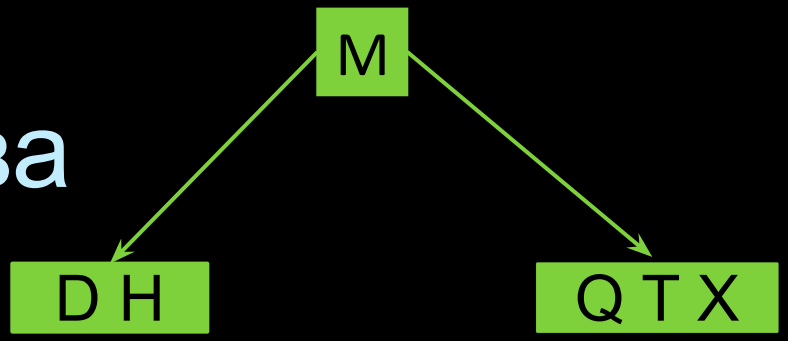
- Ключи $key_i[x]$ служат границами, разделяющими значения ключей в поддеревьях:
  $k_0 \le key_0[x] \le k_1 \le key_2[x] \le \ldots \le key_{n[x]-1}[x] \le K_{n[x]}$,
  где k_i - множество ключей, хранящихся в поддереве с корнем $C_i[x]$

Создание корня B дерева




```
B_tree *B = (B_tree*) malloc (sizeof(*B));
B->n = 1;
B->key = (int*) malloc (B->n*sizeof(int));
B->key[0] = 'M';
B->child = NULL;
```

Создание B дерева




```
B->child = (B_tree**)malloc(sizeof(B_tree*)*2);
B->child[0]=(B_tree*)malloc(sizeof(B_tree));
B->child[1]=(B_tree*)malloc(sizeof(B_tree));
x=B->child[0];
x->n=2;
x->key=(int*)malloc(x->n*sizeof(int));
x->key[0]='D';
x->key[1]='H';
X->child=NULL;
// Аналогичные действия для вершины: QTX
// Как это сделать цивилизованно?
```

- Возможна реализация, где каждая вершина является отдельным файлом
- В общем случае имеются операции
  - Disk_READ(x) – чтение с диска
  - Disk_Write(x) – запись на диск