Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгоритмы и алгоритмические языки
Лекция 22

27 ноября 2019 г.

Построение двоичного дерева поиска

Постановка задачи. Пусть имеется множество K из m ключей: $K = \{k_0, k_1, \ldots, k_{m-1}\}$.

Разобьём K на три подмножества K_1, K_2, K_3 такие, что $|K_2| = 1, |K_1| \geqslant 0, |K_3| \geqslant 0$. $K_2 = \{k\}$ и $\forall l \in K_1 : l < k, \forall r \in K_3 : k < r$.

Далее аналогично разбиваем множества K_1, K_2, K_3, пока не кончатся ключи.

Пример. $K = \{15, 10, 1, 3, 8, 12, 4\}$. Первое разбиение: $\{1, 3, 4\}, \{8\}, \{15, 10, 12\}$. Второе разбиение: $\{\{1\}\{3\}\{4\}\}\{8\}\{\{10\}\{12\}\{15\}\}$. Получилось полностью сбалансированное двоичное дерево.

Определение. Дерево называется *полностью сбалансированным* (*совершенным*), если длина пути от корня до любой листовой вершины одинакова

Построение двоичного дерева поиска

Постановка задачи. Пусть имеется множество K из m ключей: $K = \{k_0, k_1, \ldots, k_{m-1}\}$.

Разобьём K на три подмножества K_1, K_2, K_3 такие, что $|K_2| = 1, |K_1| \geqslant 0, |K_3| \geqslant 0$. $K_2 = \{k\}$ и $\forall l \in K_1 : l < k, \forall r \in K_3 : k < r$.

Далее аналогично разбиваем множества K_1, K_2, K_3, пока не кончатся ключи.

Пример. $K = \{15, 10, 1, 3, 8, 12, 4\}$. Первое разбиение: $\{1, 3, 4\}, \{8\}, \{15, 10, 12\}$. Второе разбиение: $\{\{1\}\{3\}\{4\}\}\{8\}\{\{10\}\{12\}\{15\}\}$. Получилось полностью сбалансированное двоичное дерево.

Определение. Дерево называется *полностью сбалансированным* (*совершенным*), если длина пути от корня до любой листовой вершины одинакова и все внутренние вершины имеют двоих сыновей.

Построение двоичного дерева поиска

Пусть h — высота полностью сбалансированного двоичного дерева. Тогда число вершин m должно быть равно:

$$m = 1 + 2 + 2^2 + \ldots + 2^{h-1} = 2^h - 1,$$

откуда $h = \log_2(m+1)$.

Если все m ключей известны заранее, их можно отсортировать за $O(m \log_2 m)$, после чего построение сбалансированного дерева будет тривиальной задачей.

Если дерево строится по мере поступления ключей, то возможны все варианты: от линейного дерева с высотой $O(m)$ до полностью сбалансированного дерева с высотой $O(\log_2 m)$.

Пусть `T = root, left, right` — двоичное дерево; тогда $h_T = \max(h_{left}, h_{right}) + 1$.

Деревья Фибоначчи

Числа Фибоначчи возникли в решении задачи о кроликах, предложенном в XIII веке Леонардо из Пизы, известным как Фибоначчи.

Задача о кроликах: пара новорожденных кроликов помещена на остров. Каждый месяц любая пара дает приплод — также пару кроликов. Пара начинает давать приплод в возрасте двух месяцев. Сколько кроликов будет на острове в конце n-го месяца?

В конце первого и второго месяцев на острове будет одна пара кроликов: $f_1 = 1, f_2 = 1$.

В конце третьего месяца родится новая пара, так что $f_3 = f_2 + f_1 = 2$.

По индукции можно доказать, что для $n \geqslant 3$ $f_n = f_{n-1} + f_{n-2}$.

Деревья Фибоначчи

```
int fib (int n) {
  if (n == 1 || n == 2)
    return 1;
  else {
    int g, h, fb;
    g = h = 1;
    for (int k = 2; k < n; k++) {
      fb = g + h;
      h = g;
      g = fb;
    }
    return fb;
  }
}
```

Деревья Фибоначчи

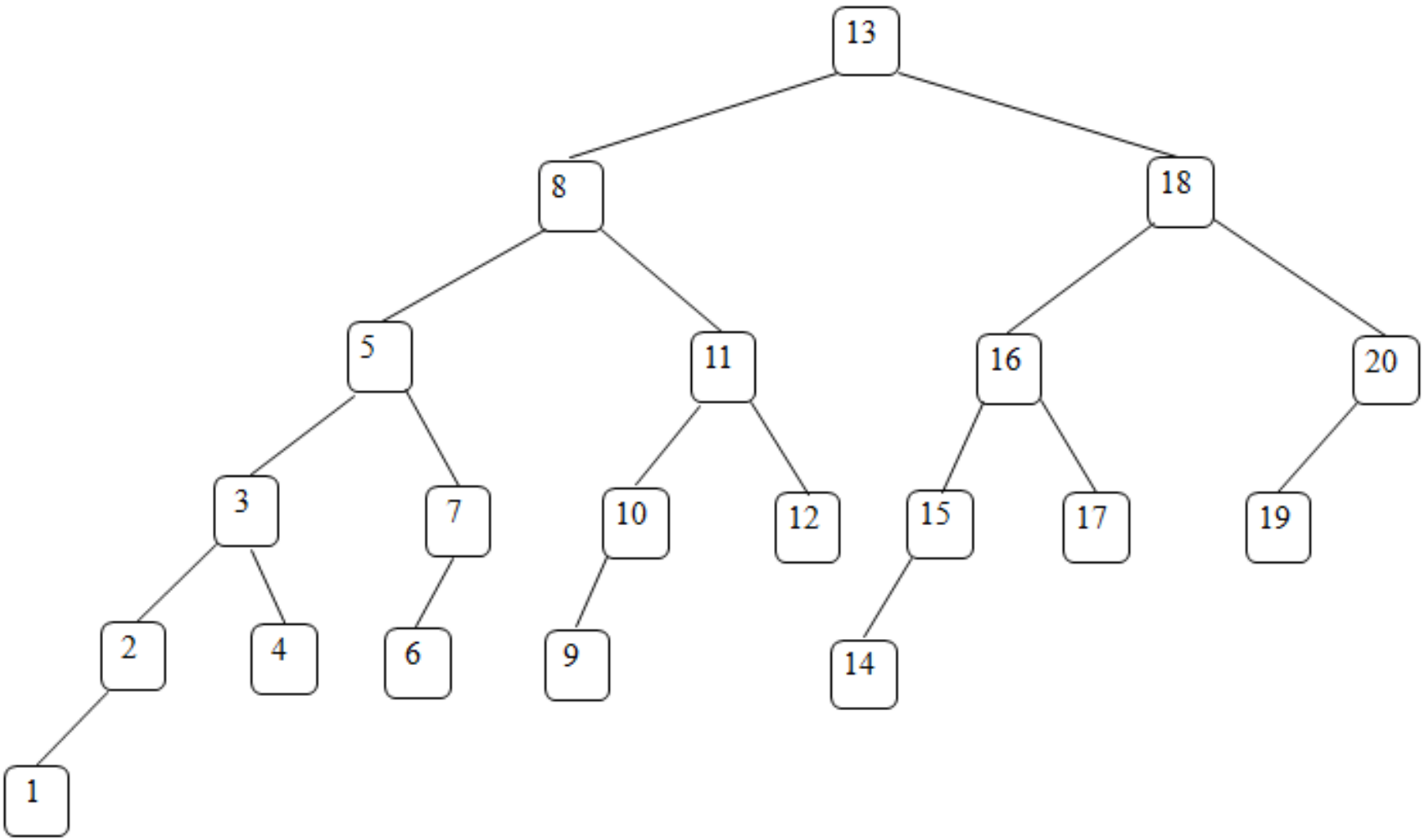
Определение дерева Фибоначчи:

- Пустое дерево — это дерево Фибоначчи с высотой $h = 0$.
- Двоичное дерево, левое и правое поддерево которого есть деревья Фибоначчи с высотами соответственно $h - 1$ и $h - 2$ (либо $h - 2$ и $h - 1$), есть дерево Фибоначчи с высотой h.

Из определения следует, что в дереве Фибоначчи значения высот левого и правого поддерева отличаются ровно на 1.

Деревья Фибоначчи

Дерево Фибоначчи с $h = 6$.



Деревья Фибоначчи

Теорема. Число вершин в дереве Фибоначчи F_h высоты h равно $m(h) = f_{h+2} - 1$.

Доказательство проводится по индукции:

- При $h = 0$ $m(0) = f_2 - 1 = 0$, $m(1) = f_3 - 1 = 1$.
- По определению $m(h) = m(h-1) + m(h-2) + 1$, значит, $m(h) = (f_{h+1} - 1) + (f_h - 1) + 1 = f_{h+2} - 1$, так как $f_h + f_{h+1} = f_{h+2}$.

Деревья Фибоначчи

Теорема. Пусть C_1 и C_2 таковы, что уравнение

$$r^2 - C_1 r - C_2 = 0$$

имеет два различных корня r_1 и r_2, $r1 \neq r2$.

Тогда для $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ выполняется соотношение $a_n = C_1 a_{n-1} + C_2 a_{n-2}$.

Доказательство. Раз r_1 и r_2 — корни уравнения, то $r_1^2 = C_1 r_1 + C_2$, $r_2^2 = C_1 r_2 + C_2$. Поэтому $C_1 a_{n-1} + C_2 a_{n-2} = C_1(\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + C_2(\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2}) = \alpha_1 r_1^{n-2}(C_1 r_1 + C_2) + \alpha_2 r_2^{n-2}(C_1 r_2 + C_2) = \alpha_1 r_1^{n-2} r_1^2 + \alpha_2 r_2^{n-2} r_2^2 = \alpha_1 r_1^n + \alpha_2 r_2^n = a_n$.

Деревья Фибоначчи

Теорема. Пусть C_1 и C_2 таковы, что уравнение

$$r^2 - C_1 r - C_2 = 0$$

имеет два различных корня r_1 и r_2, $r1 \neq r2$.

Тогда из $a_n = C_1 a_{n-1} + C_2 a_{n-2}$ и начальных условий a_0, a_1 следует $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ для $n = 1, 2, \ldots$.

Доказательство. Нужно повторить в обратном порядке вывод предыдущей теоремы, но и подобрать такие α_1 и α_2, чтобы $a_0 = \alpha_1 + \alpha_2, a_1 = \alpha_1 r_1 + \alpha_2 r_2$. Решая эту систему линейных уравнений, получим

$$\alpha_1 = \frac{a_1 - a_0 r_2}{r_1 - r_2}, \alpha_2 = \frac{-a_1 + a_0 r_1}{r_1 - r_2}.$$

.

Деревья Фибоначчи

Применим доказанные теоремы к числам Фибоначчи $f_n = f_{n-1} + f_{n-2}$. Уравнение $r^2 - r - 1 = 0$ имеет корни $r_{1,2} = \frac{1 \pm \sqrt{5}}{2}$.

Следовательно, $f_n = \alpha_1(\frac{1+\sqrt{5}}{2})^n + \alpha_2(\frac{1-\sqrt{5}}{2})^n$, $f_0 = \alpha_1 + \alpha_2 = 0$, $f_1 = \alpha_1(\frac{1+\sqrt{5}}{2}) + \alpha_2(\frac{1-\sqrt{5}}{2}) = 1$.

Из описанной системы получаем $\alpha_1 = \frac{1}{\sqrt{5}}$, $\alpha_2 = -\frac{1}{\sqrt{5}}$.

Отсюда $m(h) = f_{h+2} - 1 = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^{h+2} - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^{h+2} - 1$.

Заметим, что второе слагаемое по модулю не превосходит единицы, а, следовательно,

$$m(h) + 1 > \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^{h+2}.$$

Деревья Фибоначчи

$$m(h) + 1 > \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^{h+2}.$$

Обозначим $\gamma = \frac{1+\sqrt{5}}{2}$ и логарифмируем неравенство. Тогда

$$h + 2 < \frac{\log_2(m+1)}{\log_2 \gamma} + \frac{\log_2 \sqrt{5}}{\log_2 \gamma},$$

откуда

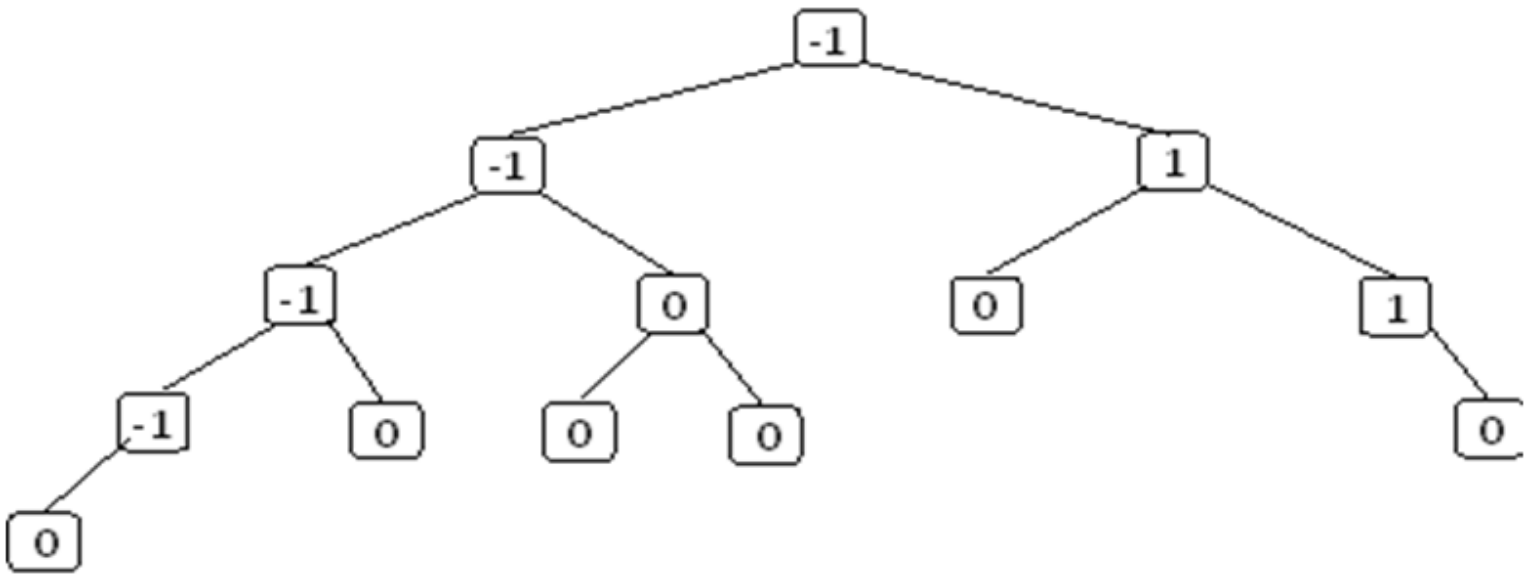
$$h < 1.44 \log_2(m+1) - 0.32.$$

Таким образом, мы доказали, что для деревьев Фибоначчи с числом вершин m количество сравнений в худшем случае не превышает $1.44 \log_2(m+1)$.

АВЛ-деревья

В АВЛ-деревьях (Адельсон-Вельский, Ландис) оценка сложности не лучше, чем в совершенном дереве, но не хуже, чем в деревьях Фибоначчи для всех операций: поиск, исключение, занесение.

АВЛ-деревом (подравненным деревом) называется такое двоичное дерево, в котором для любой его вершины высоты левого и правого поддерева отличаются не более, чем на 1.



АВЛ-деревья

В узлах дерева записаны значения показателя сбалансированности (balance factor), равного разности высот левого и правого поддеревьев. Показатель сбалансированности может иметь одно из трех значений:

–1: Высота левого поддерева на 1 больше высоты правого поддерева.

0: Высоты обоих поддеревьев одинаковы.

+1: Высота правого поддерева на 1 больше высоты левого поддерева.

У совершенного дерева все узлы имеют показатель баланса 0 (это самое «хорошее» АВЛ-дерево) а у дерева Фибоначчи все узлы имеют показатель баланса +1 (либо –1) (это самое «плохое» АВЛ-дерево).

Типичная структура узла и операции в АВЛ-деревьях

```
typedef int key_t;
struct avlnode;
typedef struct avlnode *avltree;
struct avlnode {
  key_t key;              //ключ
  avltree       left;   //левое поддерево
  avltree       right;  //правое поддерево
  //    int     balance;        показательбаланса
  int           height; //высота поддерева
};

avltree makeempty (avltree t);     //удалить дерево
avltree find (key_t x, avltree t);  //поиск поключу
avltree findmin (avltree t);          //минимальный ключ
avltree findmax (avltree t);          //максимальный ключ
avltree insert (key_t x, avltree t); //вставить узел
avltree delete (key_t x, avltree t); //исключить узел
```

Реализация простейших базовых операций

```
avltree makeempty (avltree t) {
  if (t != NULL) {
    makeempty (t->left);
    makeempty (t->right);
    free (t);
  }
  return NULL;
}

avltree find (key_t x, avltree t) {
 if (t == NULL || x == t->key)
   return t;
 if (x < t->key)
   return find (x, t->left);
 if (x > t->key)
   return find (x, t->right);
}
```

Реализация простейших базовых операций

```
avltree findmin (avltree t) {
  if (t == NULL)
    return NULL;
  else if (t->left == NULL)
    return t;
  else
    return findmin (t->left);
}

avltree findmax (avltree t) {
  if (t != NULL)
    while (t->right != NULL)
      t = t->right;
  return t;
}
```

Включение узла в АВЛ-дерево

Рассматриваемое дерево состоит из корневой вершины r и левого (L) и правого (R) поддеревьев, имеющих высоты h_L и h_R соответственно. Для определённости будем считать, что новый ключ включается в поддерево L.

Если h_L **не изменяется**, то не изменяются и соотношения между h_L и h_R, и свойства АВЛ-дерева сохраняются.

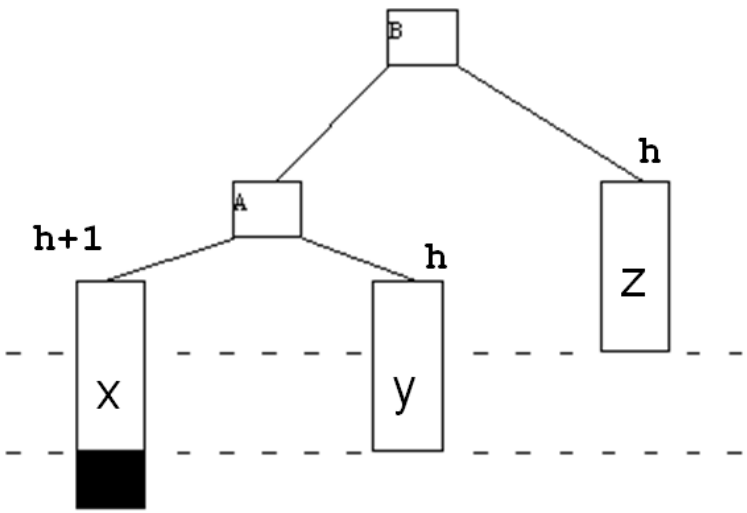
Если h_L **увеличивается на единицу**, возможны три случая:

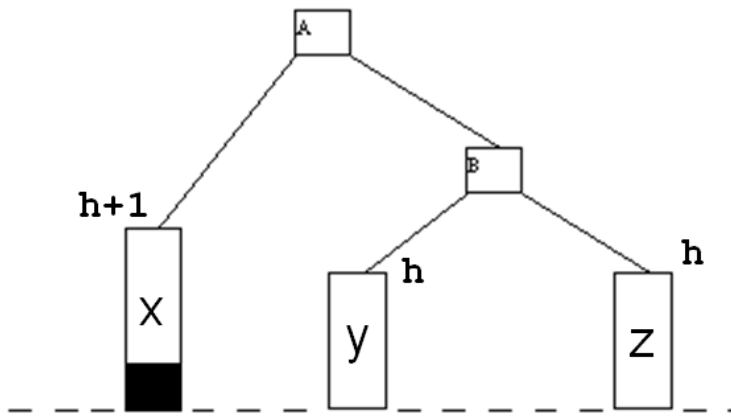
- $h_L = h_R$, тогда после добавления вершины L и R станут разной высоты, но свойство сбалансированности сохранится;
- $h_L < h_R$, тогда после добавления новой вершины L и R станут равной высоты, т.е. сбалансированность общего дерева даже улучшится;
- $h_L > h_R$, тогда после включения ключа сбалансированность нарушится, и потребуется перестройка дерева.

Включение узла в АВЛ-дерево

Новая вершина добавляется к левому поддереву поддерева *L*. В результате поддерево с корнем в узле *B* разбалансировалось: разность высот его левого и правого поддеревьев стала равной –2.

Преобразование, разрешающее ситуацию — однократный поворот RR: делаем узел *A* корневым узлом поддерева, в результате правое поддерева с корнем в узле *B* «опускается» и разность высот становится равной 0.

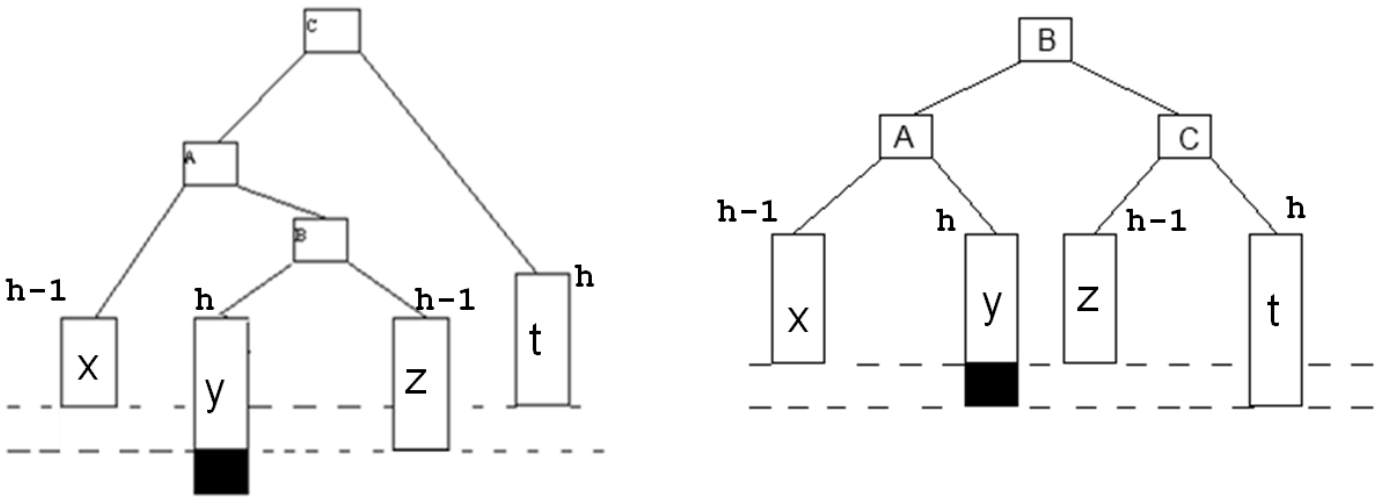




Включение узла в АВЛ-дерево

Новая вершина добавляется к правому поддереву поддерева *L*. В результате поддерево с корнем в *C* разбалансировалось: разность высот его левого и правого поддеревьев стала равной -2.

Преобразование, разрешающее ситуацию — двукратный поворот LR: «вытягиваем» узел *B* на самый верх, чтобы его поддеревья поднялись. Для этого сначала делаем левый поворот, меняя местами поддеревья с корневыми узлами *A* и *B*, а потом правый поворот, меняя местами поддеревья с корнями *B* и *C*.



Построение АВЛ-дерева

```
static inline int height (avltree p) {
        return p ? p->height : 0;
}

static inline int height (avltree p) {
        return p ? p->height : 0;
}
```

Построение АВЛ-дерева. Однократные повороты

Между узлом и его левым сыном.

Функция SingleRotateWithLeft вызывается только в том случае, когда у узла K2 есть левый сын. Функция выполняет поворот между узлом (K2) и его левым сыном, корректирует высоты поддеревьев, после чего возвращает новый корень.

```
static avltree SingleRotateWithLeft (avltree k2) {
  avltree k1;
  /* выполнение поворота */
  k1 = k2->left;
  k2->left = k1->right;          /* k1 != NULL */
  k1->right = k2;
  /* корректировка высот переставленных узлов */
  k2->height = max (height (k2->left),
                    height (k2->right)) + 1;
  k1->height = max (height (k1->left), k2->height) + 1;
  return k1; /* новый корень */
}
```

Построение АВЛ-дерева. Однократные повороты

Между узлом и его правым сыном.

Эта функция вызывается только в том случае, когда у узла K1 есть правый сын. Функция выполняет поворот между узлом (K1) и его правым сыном, корректирует высоты поддеревьев, после чего возвращает новый корень.

```
static avltree SingleRotateWithRight (avltree k1) {
  avltree k2;
  k2 = k1->right;
  k1->right = k2->left;
  k2->left = k1;
  k1->height = max (height (k1->left)
                    height (k1->right)) + 1;
  k2->height = max (height (k2->right), k1->height) + 1;
  return k2; /* новый корень */
}
```

Построение АВЛ-дерева. Двойные повороты

LR-поворот.

Эта функция вызывается только тогда, когда у узла K3 есть левый сын, а у левого сына K3 есть правый сын. Функция выполняет двойной поворот LR, корректирует высоты поддеревьев, после чего возвращает новый корень.

```
static avltree DoubleRotateWithLeft (avltree k3) {
  /* Поворот между K1 и K2 */
  k3->left = SingleRotateWithRight (k3->left);
  /* Поворот между K3 и K2 */
  return SingleRotateWithLeft (k3);
}
```

Построение АВЛ-дерева. Двойные повороты

RL-поворот.

Эта функция вызывается только в том случае, когда у узла K1 есть правый сын, а у правого сына узла K1 есть левый сын. Функция выполняет двойной поворот RL, корректирует высоты поддеревьев, после чего возвращает новый корень.

```
static avltree DoubleRotateWithRight (avltree k1) {
  /* Поворот между K3 и K2 */
  k1->right = SingleRotateWithLeft (k1->right);
  /* Поворот между K1 и K2 */
  return SingleRotateWithRight(k1);
}
```

Построение АВЛ-дерева. Вставить новый узел

```
avltree insert (key_t x, avltree t) {
  if (t == NULL) {
    /* создание дерева с одним узлом */
    t = malloc (sizeof (struct avlnode));
    if (!t)
      abort();
    t->key = x;
    t->height = 1;
    t->left = t->right = NULL;
  }
  else if (x < t->key) {
    t->left = insert (x, t->left);
    if (height (t->left) - height (t->right) == 2) {
```

Построение АВЛ-дерева. Вставить новый узел

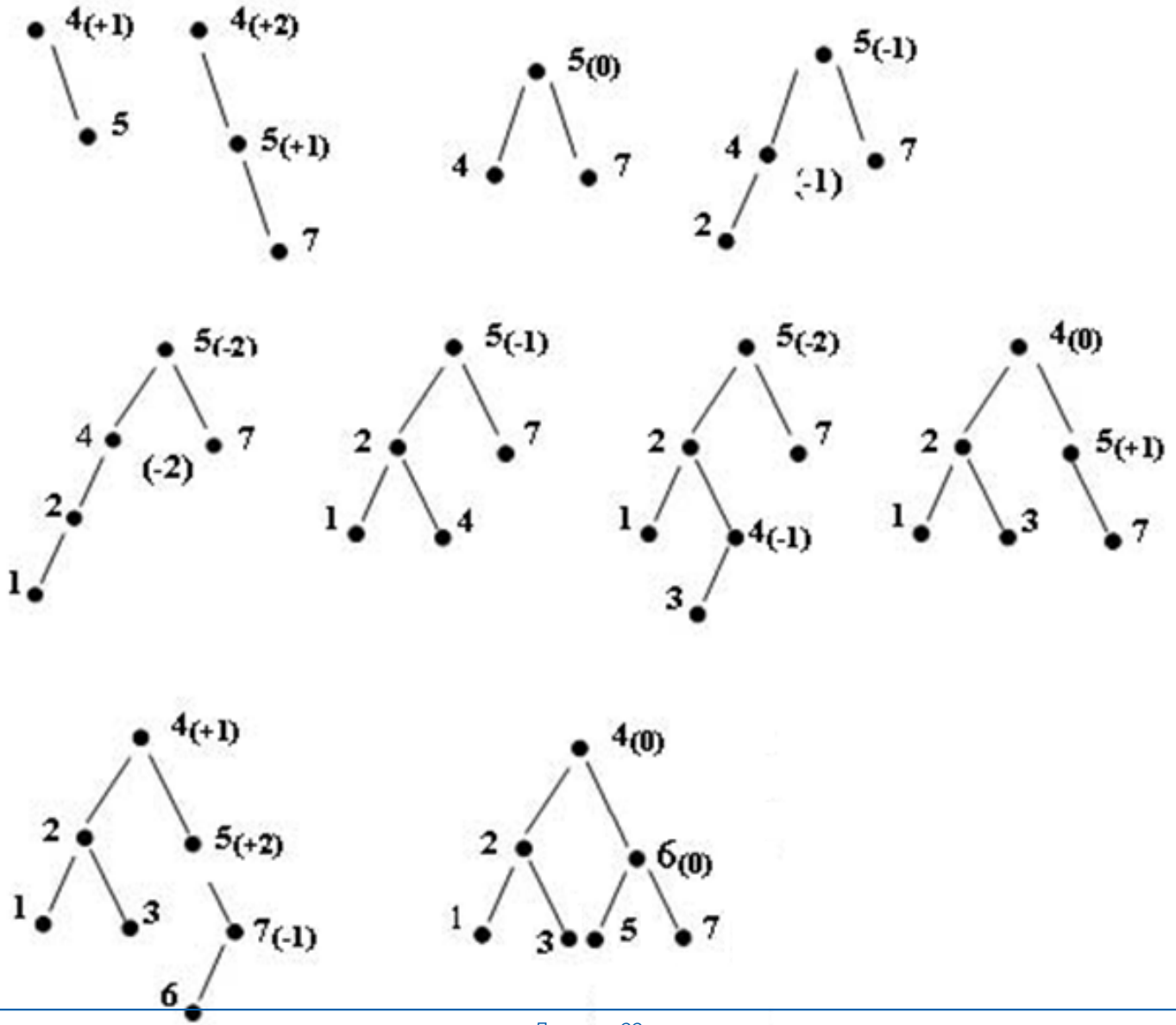
```
    if (x < t->left->key)
      t = SingleRotateWithLeft (t);
    else
      t = DoubleRotateWithLeft (t);
    }
  }
  else if (x > t->key) {
    t->right = insert (x, t->right);
    if (height (t->right) – height (t->left) == 2) {
      if (x > t->right->key)
        t = SingleRotateWithRight (t);
      else
        t = DoubleRotateWithRight (t);
    }
  }
```

Построение АВЛ-дерева. Вставить новый узел

```
  /* иначе x уже в дереве */
  t->height = max (height (t->left),
                   height (t->right)) + 1;
  return t;
}
```

Пример построения АВЛ-дерева

Последовательно вставляем целые числа 4,5,7,2,1,3,6.



Удаление узла из АВЛ-дерева

Удаление узла из АВЛ-дерева требует балансировки дерева. Иными словами, в конец функции, выполняющей удалениеузла, необходимо добавить вызовы функций SingleRotateWithRight(T), SingleRotateWithLeft(T), DoubleRotateWithRight(T) и DoubleRotateWithLeft(T).

Возможны случаи вращения, не встречавшиеся при вставке.

Может оказаться необходимым выполнить несколько вращений.

Оценка сложности

Ранее были получены оценки высоты самого «хорошего» АВЛ-дерева, содержащего m узлов (полностью сбалансированное дерево):

$$h = O(\log_2(m+1)),$$

и самого «плохого» АВЛ-дерева, содержащего m узлов (дерево Фибоначчи):

$$h < 1.44 \log_2(m+1) - 0.32.$$

Следовательно, для «среднего» АВЛ-дерева, содержащего m узлов:

$$\log_2(m+1) \leqslant h \leqslant 1.44 \log_2(m+1) - 0.32.$$