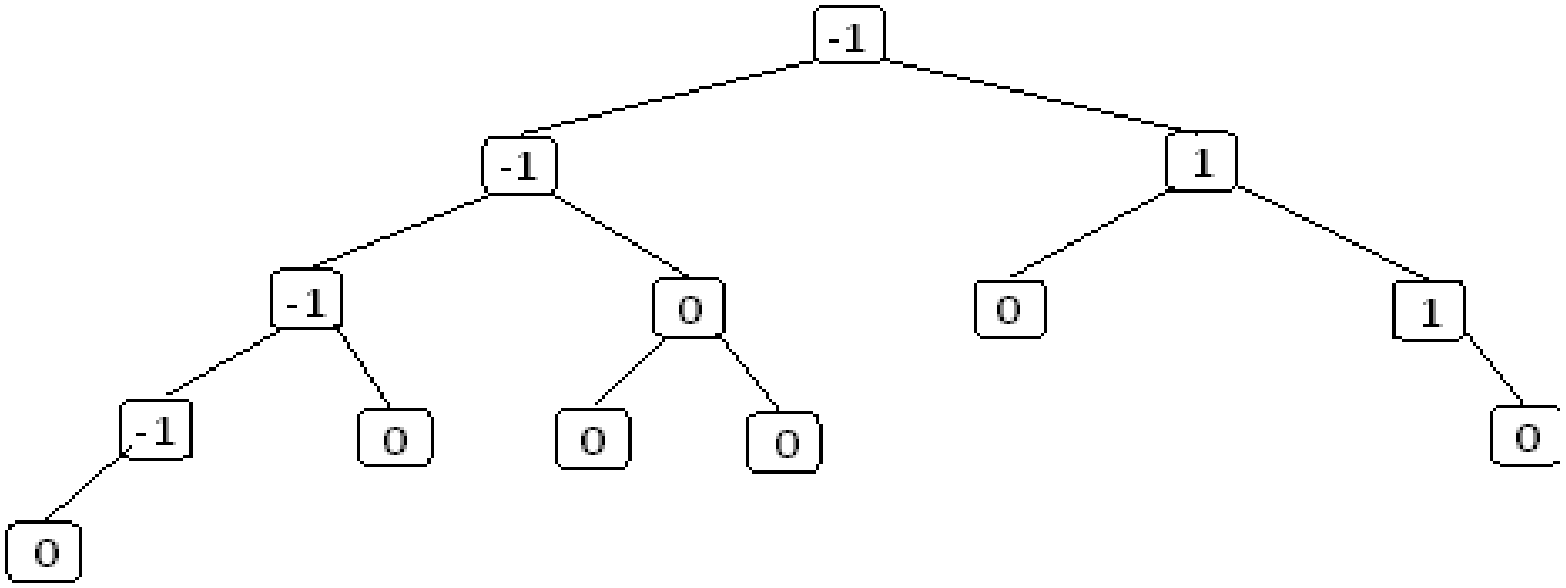
Курс «Алгоритмы и алгоритмические языки»
1 семестр 2013/2014

Лекция 20

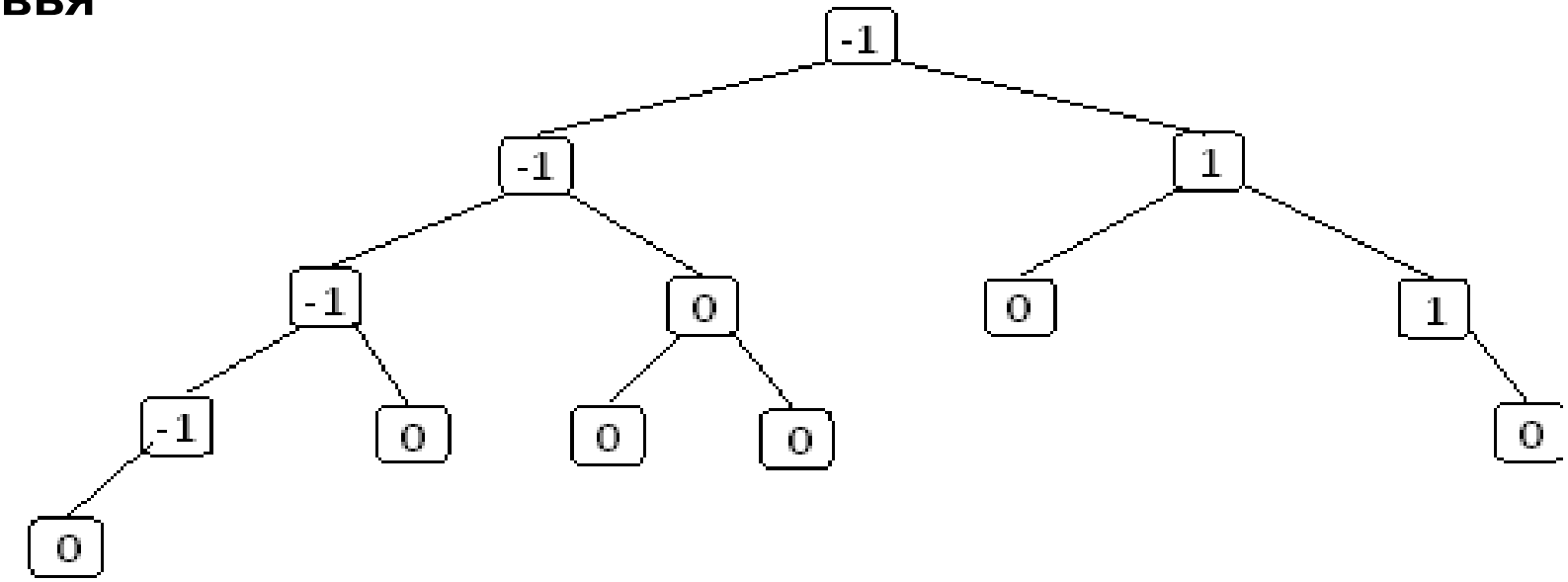
АВЛ-деревья

- 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.



Пример АВЛ-дерева.

АВЛ-деревья



- В узлах дерева записаны значения *показателя сбалансированности* (*balance factor*), определяемого по формуле:

  $$balance\ factor = height(right\ subtree) - height(left\ subtree)$$

  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).

АВЛ-деревья

- Типичная структура узла АВЛ-дерева:

```
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;    //показатель баланса
  int       height;     //высота поддерева
};
```

АВЛ-деревья.

- Базовые операции над АВЛ-деревьями.

```
avltree makeempty (avltree t);      //удалить дерево
avltree find (key_t x, avltree t);  //поиск по ключу
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

- Удалить дерево:

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

- Поиск по ключу:

```
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

- Минимальный и максимальный ключи:

```
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

- ***Поддержка балансировки АВЛ-дерева при выполнении операции включения ключей***

  Рассматриваемое дерево состоит из корневой вершины r и левого (L) и правого (R) поддеревьев, имеющих высоты h_L и h_R соответственно.

  Для определенности будем считать, что новый ключ включается в поддерево L.

- h_L **не изменяется** $\Rightarrow$ не изменяются соотношения между h_L и h_R $\Rightarrow$ свойства АВЛ-дерева сохраняются.

- h_L **увеличивается на 1** $\Rightarrow$ возможны три случая:

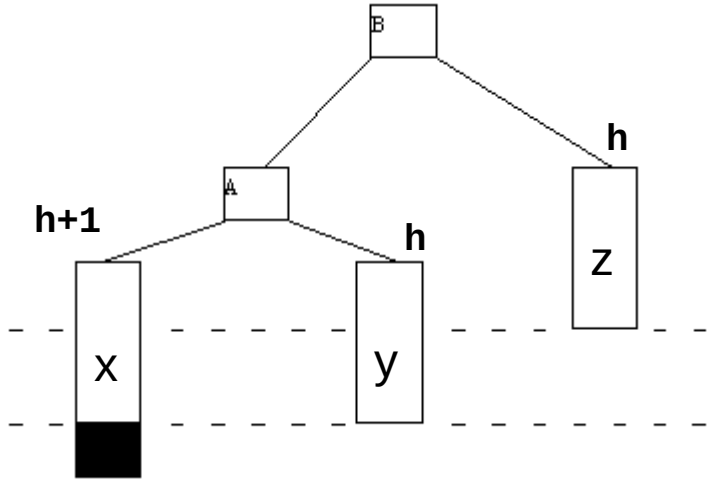
  (1) $h_L = h_R \Rightarrow$ после добавления вершины L и R станут разной высоты, но свойство сбалансированности сохранится

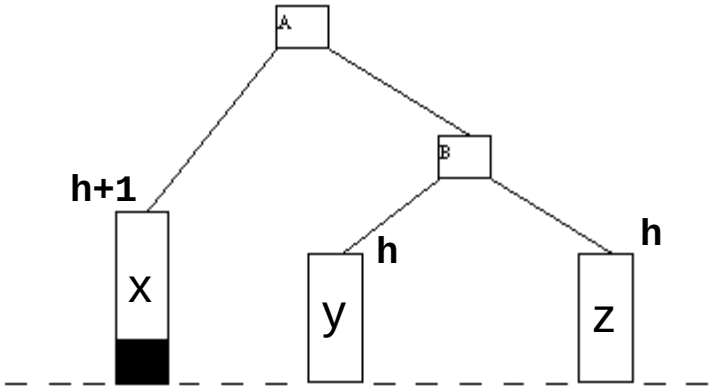
  (2) $h_L < h_R \Rightarrow$ после добавления новой вершины L и R станут равной высоты, т.е. сбалансированность общего дерева даже улучшится

  (3) $h_L > h_R \Rightarrow$ после включения ключа сбалансированность нарушится, и *потребуется перестройка дерева*.

Включение узла в АВЛ-дерево

- (3а) **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 .



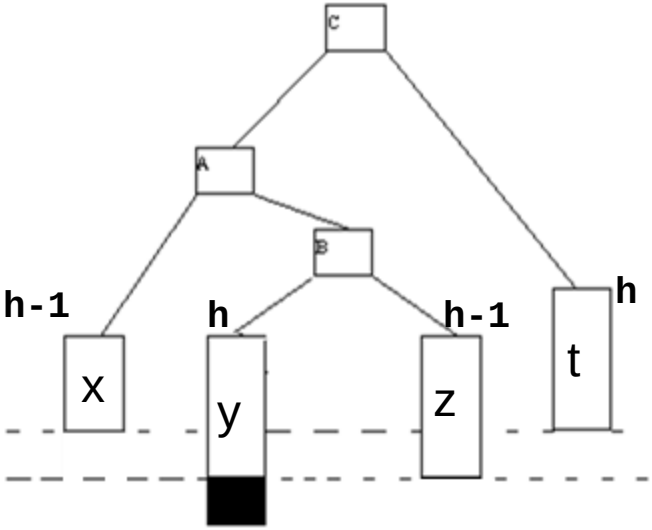


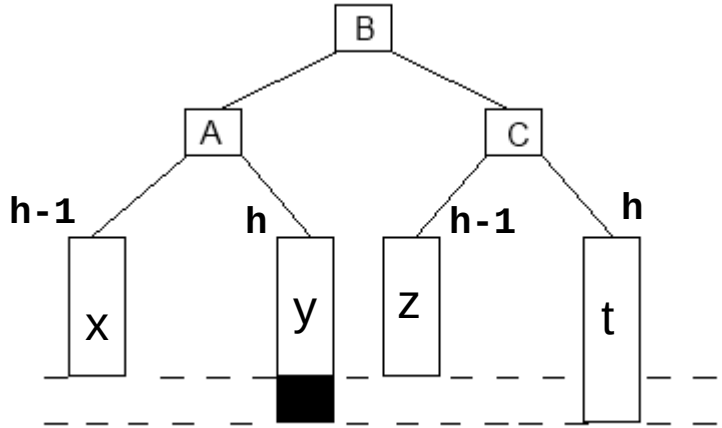
Преобразование, разрешающее ситуацию (3а) (однократный поворот $\boldsymbol{RR}$):

Делаем узел A корневым узлом поддерева, в результате правое поддерева с корнам в узле B «опускается» и разность высот становится равной 0

Включение узла в АВЛ-дерево

- (3b) **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 (3b) (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 корневыми узлами B и C.

Построение АВЛ-дерева

- Высота поддерева с корнем в узле P.

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

- Выбор более длинного поддерева

```
static inline int max (int lhs, int rhs) {
  return lhs > rhs ? lhs : rhs;
}
```

Построение АВЛ-дерева

- Однократные повороты
  - Между узлом и его левым сыном

Функция SingleRotateWithLeft вызывается только в том случае, когда у узла К2 есть левый сын. Функция выполняет поворот между узлом (К2) 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 (все)

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

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) {
```

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

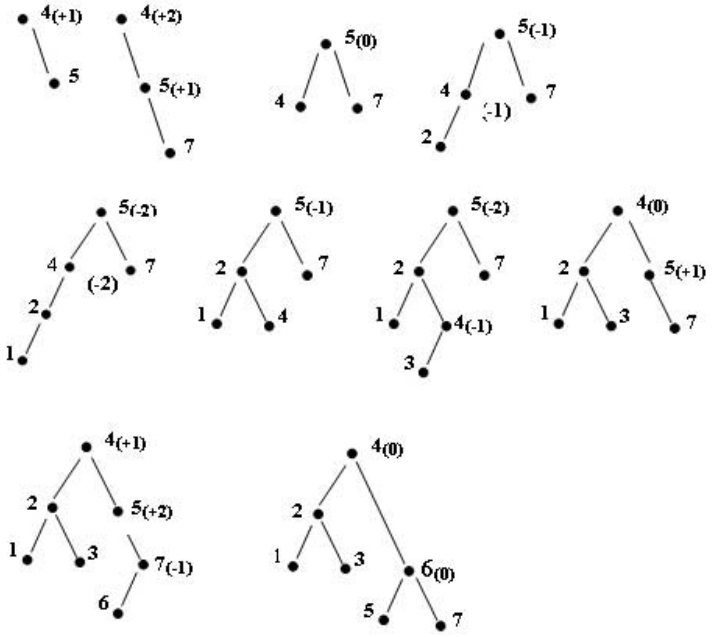
Построение АВЛ-дерева

- ***Пример***

Пусть на «вход» функции **`insert()`** последовательно поступают целые числа 4,5,7,2,1,3,6.

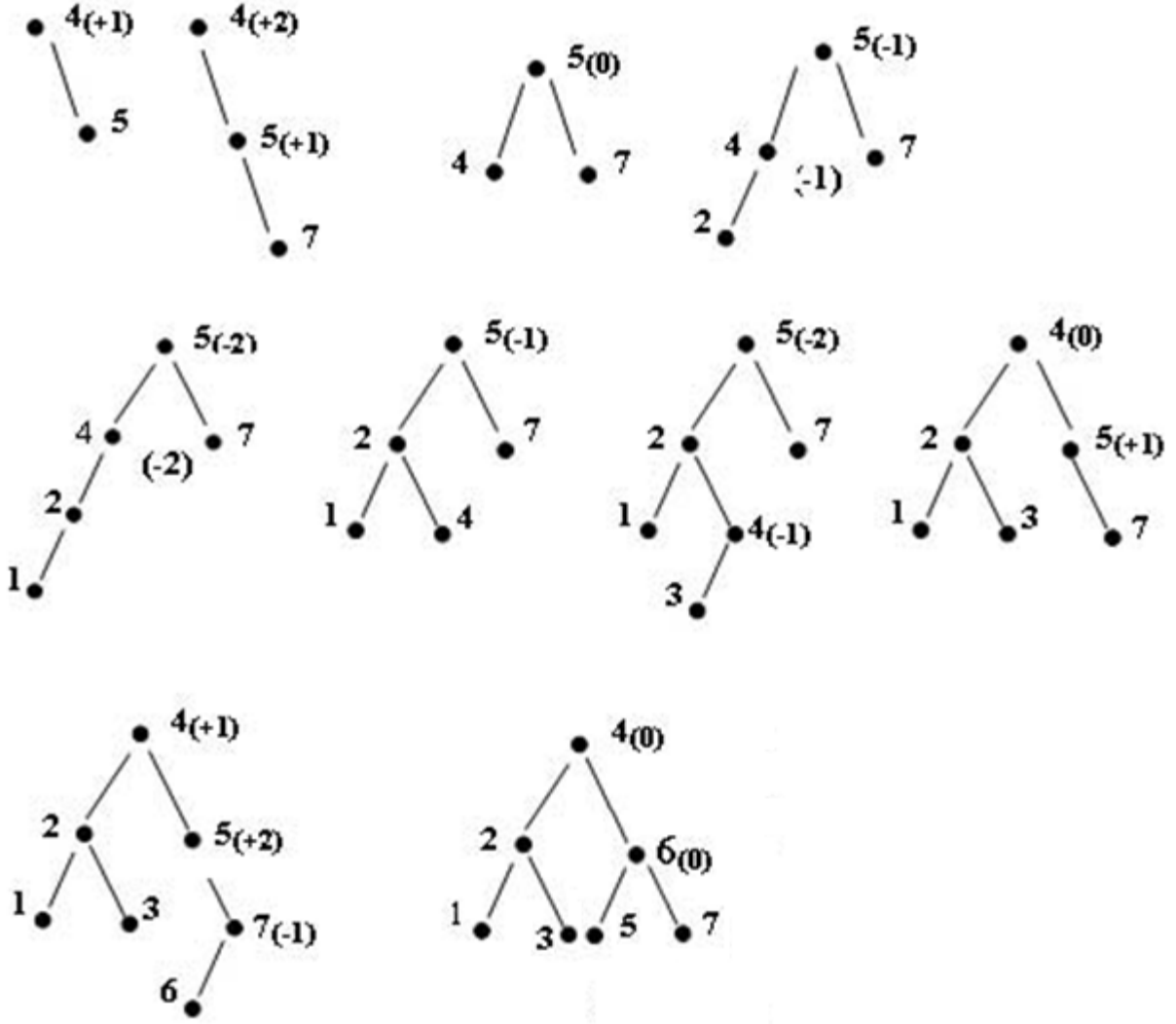
Изобразим процесс «роста» АВЛ-дерева (в скобках для части вершин указан показатель сбалансированности):

Числа в примере подобраны так, чтобы обеспечить как можно больше поворотов при минимальном числе включений.



Построение АВЛ-дерева

- ***Пример построения АВЛ-дерева***



Исключение узла из АВЛ-дерева

- Объявление функции:

```
avltree delete (key_t x, avltree t);
```

- Исключение узла из АВЛ-дерева требует балансировки дерева.

  Иными словами, в конец функции, выполняющей исключение узла, 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

(1) самого «хорошего» АВЛ-дерева, содержащего m узлов (*полностью сбалансированное дерево*)

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

(2) самого «плохого» АВЛ-дерева, содержащего m узлов (*дерево Фибоначчи*)

$$h \le 1.44 \cdot \log_2(m+1) - 0.32$$

Следовательно, для «среднего» АВЛ-дерева, содержащего m узлов оценка высоты будет где-то посередине:

$$\log_2(m+1) \le h \le 1.44 \cdot \log_2(m+1) - 0.32$$