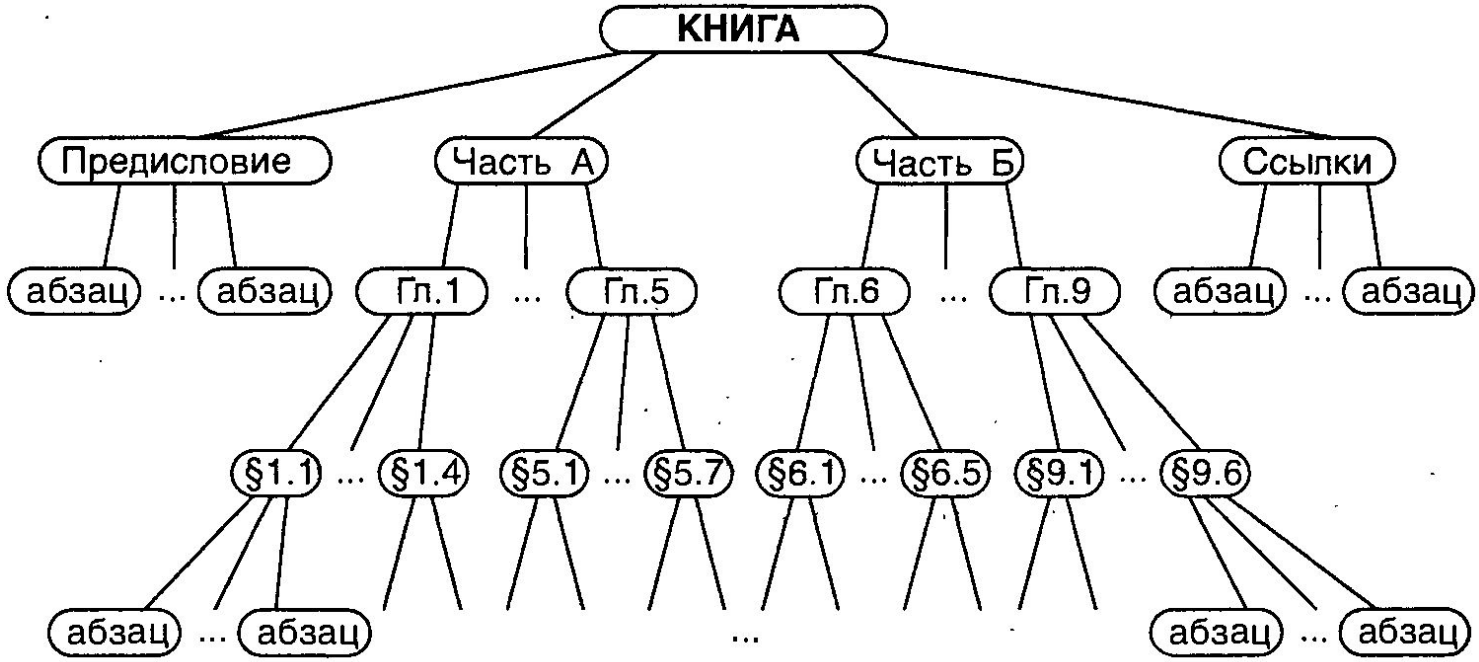
АТД «Дерево» - общее представление

Дерево — это абстрактный тип данных (АТД) для иерархического хранения элементов. За исключением элемента во главе дерева (***root***), каждый элемент структуры имеет родителя (***parent***) и ноль или более дочерних элементов (***children***).



Дерево, ассоциируемое с книгой

АТД «Дерево» - терминология (1)

Дерево (tree) *T* — это набор *узлов* ***(node),*** хранящих элементы, состоящие в отношениях «отцы и дети», со следующими свойствами:

- ***T*** имеет особый узел ***r,*** называемый корнем данного древа ***(root*** *of* ***T);***
- каждый узел ***v*** этого ***T,*** отличный от ***r,*** имеет родительский узел ***u***.

В соответствии с приведенным определением дерево не может быть пустым,

Если узел ***u*** является ***родителем*** (**parent**) узла ***v***, то ***v*** является ***дочерним (child)*** узлом ***u****.*

*У*злы, дочерние для одного родителя, называются ***сестрами/братьями (siblings).***

Узлы, имеющие один и более дочерних элементов, называются ***составными (internal),*** а не имеющие их — ***простыми (external)*** или ***листьями (leaves).***

Предок (ancestor) узла - родительский узел, либо предок родителя этого узла.

v является **потомком** узла ***u***, если ***u*** является предком ***v***.

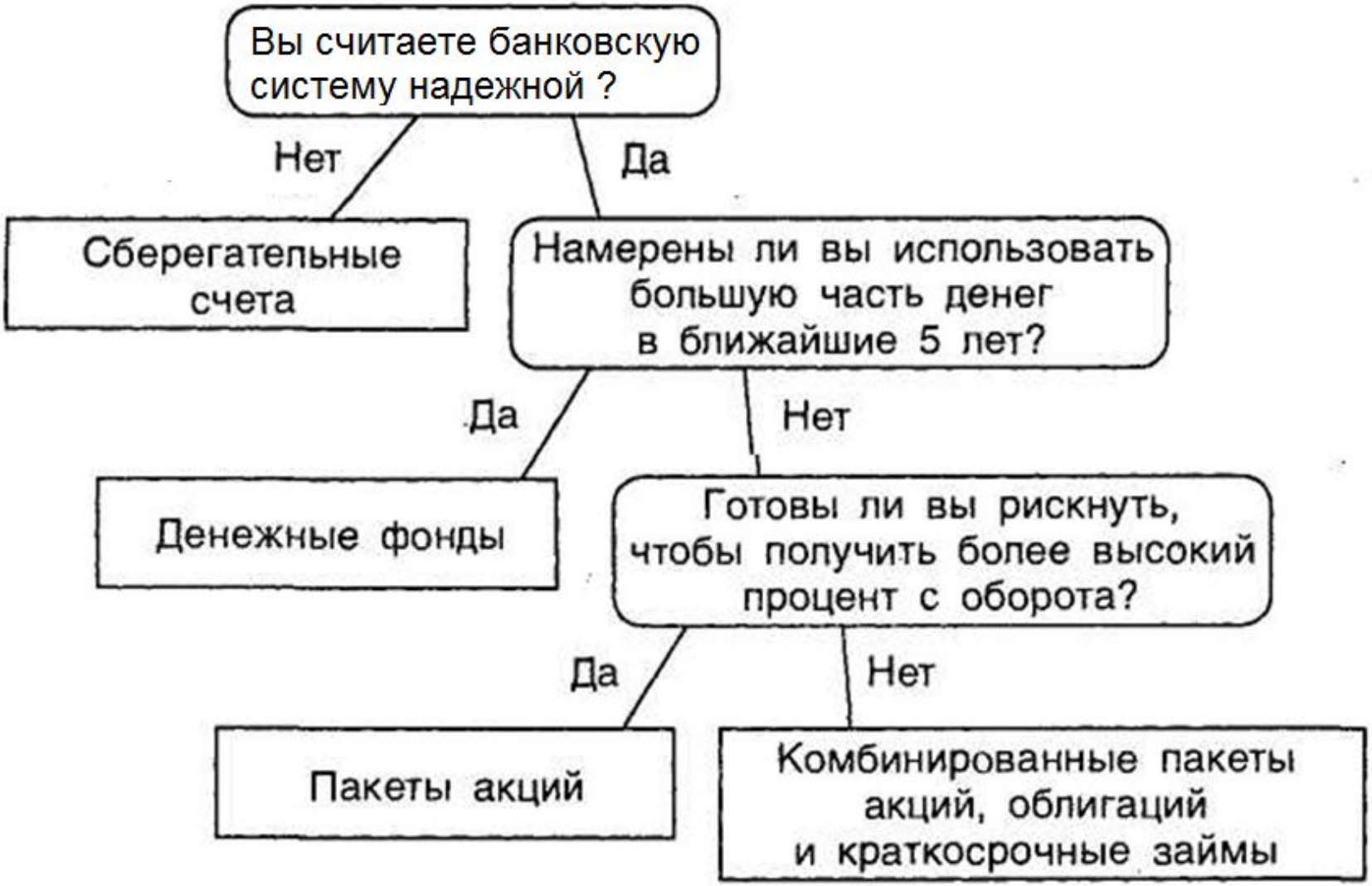
Ответвление (subtree) от дерева, корнем которого является узел ***v***, это

АТД «Дерево» - терминология (2)

Дерево является ***упорядоченным (ordered),*** если дочерние элементы каждого из узлов упорядочены, то есть каждый из элементов можно определить как первый, второй, третий и т.д. Обычно изображаются слева направо.

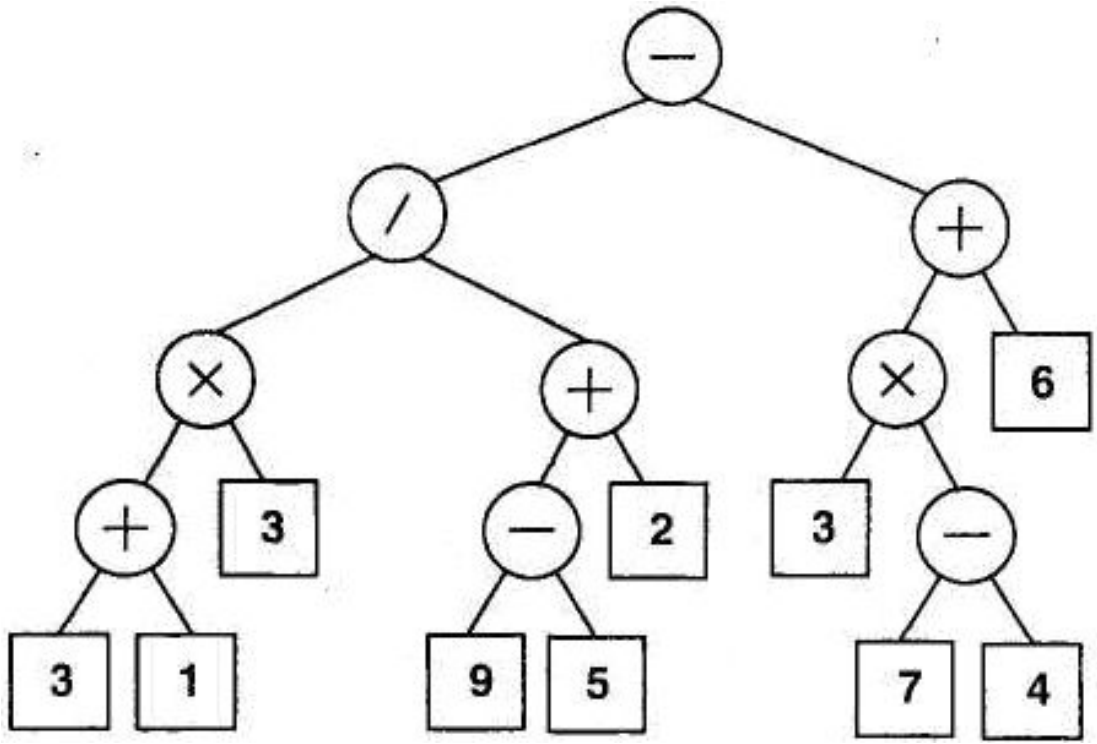
Бинарным деревом (binary tree) называется упорядоченное дерево, в котором каждый из узлов имеет максимум два дочерних элемента. Бинарное дерево считается **правильным *(proper),*** если каждый узел не содержит ни одного или содержит два дочерних элемента. Дочерние элементы в таких узлах называют ***«правый»*** и ***«левый» (left child*** *и* ***right child).*** Ответвление, берущее начало из левого или правого элемента составного узла ***v***, будет называться соответственно ***левым*** или ***правым ответвлением (left subtree*** *и* ***right subtree)*** узла ***v***.

Пример бинарного дерева – дерево решений



Пример бинарного дерева, представляющего арифметическое выражение

((((3 + 1) × 3) / (9 - 5) + 2)) - ((3 × (7 - 4)) + 6)



АТД «Дерево»

В АТД «дерево» «узлы» будут представлены **позициями**

Для «Дерева» определены следующие группы методов:

- ***методы доступа (accessor method)***
- ***методы запроса (query methods)***
- ***общие методы (generic method)***
- ***методы обновления (update methods)***

АТД «Дерево» - методы доступа

Root(): возвращает корень дерева.
Input: нет; ***Output:*** позиция.
Parent(*v*): возвращает родителя узла *v*; ошибка, если *v* является корнем.
Input: позиция; ***Output:*** позиция.
Children(*v*): возвращает итератор дочерних элементов узла *v*.
Input: позиция; ***Output:*** итератор объектов позиций.
Если дерево *T* упорядочено, то итератор Children(*v*) обеспечивает доступ к дочерним элементам узла *v* в определенном порядке. Для простого узла *v* Children(*v*) – пустой итератор.

АТД «Дерево» - методы доступа

- IsInternal(*v*): проверяет, является ли *v* составным. ***Input:*** позиция; ***Output:*** логическое значение.
- IsExternal(*v*): проверяет, является ли *v* простым. ***Input:*** позиция; ***Output:*** логическое значение.
- IsRoot(*v*): проверяет, является ли *v* корнем. ***Input:*** позиция; ***Output:*** логическое значение.

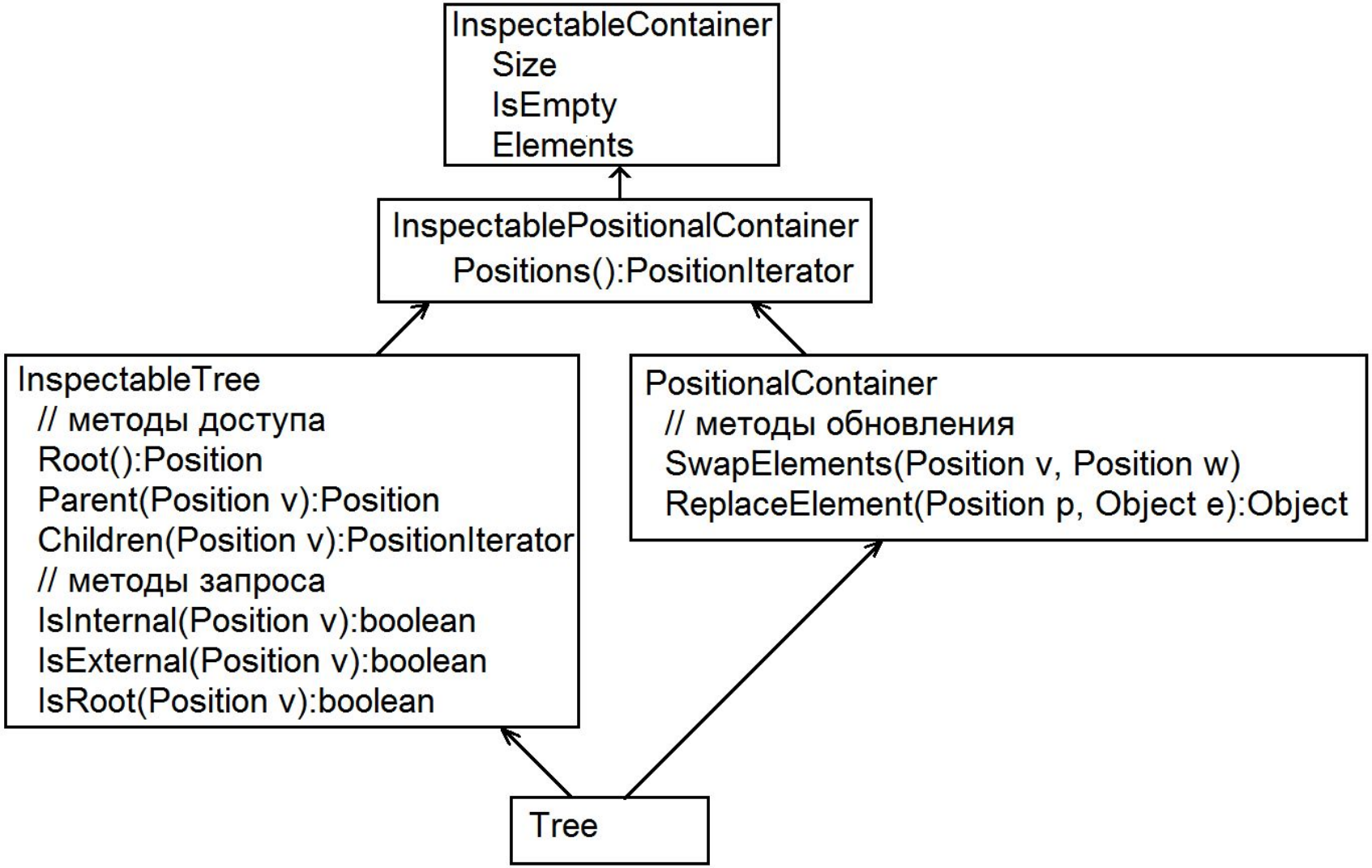
АТД «Дерево» - общие методы

- Size(): возвращает количество узлов в дереве.
  Input: нет; ***Output:*** целое число.
- Elements(): возвращает итератор всех элементов, хранимых в узлах дерева.
  Input: нет; ***Output:*** итератор объектов.
- Positions(): возвращает итератор всех узлов дерева.
  Input: нет; ***Output:*** итератор позиций.

АТД «Дерево» - методы обновления

- SwapElements(*v*,*w*): меняет местами элементы, хранимые в узлах *v* и *w*.
  Input: две позиции; ***Output:*** отсутствует.
- ReplaceElement(*v*,*e*): замещает на *e* и возвращает элемент, хранившийся в узле *v*.
  Input: позиция и ее объект; ***Output:*** объект

Структура интерфейсов для АТД «Дерево»



Основные алгоритмы над деревьями

Предварительные допущения:

- Методы доступа Root() и Parent() выполняются за $O(1)$ времени.
- Метод доступа Children(v) требует $O(c_v)$ времени, где c_v — количество дочерних элементов v.
- Методы запросов IsInternal(v), IsExternal(v) и IsRoot(v) также выполняются за $O(1)$ времени.
- Общие методы SwapElements(v) и ReplaceElement(v,e) требуют $O(1)$ времени.
- Общие методы Elements() и Positions(), возвращающие итераторы, выполняются за $O(n)$ времени, где n — количество узлов в дереве.
- Для итераторов, возвращаемых методами Elements(), Positions() и Children(v), методы HasNext(), NextObject() или NextPosition() выполняются за $O(1)$ времени каждый.

Основные алгоритмы над деревьями – глубина узла

Глубина узла *v* - количество предков *v*, исключая сам *v*.

Рекурсивное определение:

- если *v* — корень, то его глубина равна 0;
- иначе глубина *v* равна 1+глубина родителя *v* .

```
public static int depth(InspectableTree T,
                        Position v)
{   if (T.IsRoot(v)) return 0;
    else return 1 + depth(T, T.Parent(v));
}
```

Время выполнения depth(*T,v*) равно $O(1 + d_v)$, где d_v - глубина узла *v* дерева *T. В худшем случае* - $O(n)$.

Основные алгоритмы над деревьями - высота

Высота узла *v* дерева *T:*

- если *v* является простым узлом, то высота *v* равна 0;
- Иначе высота *v* равна 1 + максимальная высота дочернего элемента узла *v*.

Высота дерева *T* равна высоте корня *T.*

Утверждение. Высота дерева *T* равна максимальной глубине простого узла дерева *T*.

Основные алгоритмы над деревьями – высота 1

```
public static int height1(InspectableTree T)
{   int h = 0;
    PositionIterator positer = T.Positions();
    while (positer.HasNext())
    {   Position v = positer.NextPosition();
        if (T.isExternal(v)) h = Math.Max(h, depth(T, v));
    }
    return h;
}
```

Основные алгоритмы над деревьями – высота 2

```
public static int height2(InspectableTree T, Position v)
{   if (T.IsExternal(v)) return 0;
    else
    {   int h = 0;
        PositionIterator children = T.Children(v);
        while (children.HasNext())
          h = Math.Max(h, height2(T, children.NextPosition()));
        return 1 + h;
    }
}
```

Время выполнения height2 для корня дерева T равно $\boldsymbol{O}(n)$, где n — количество узлов T.

Проход дерева

Проход (traversal) – систематическая процедура, в ходе которой каждый узел дерева обрабатывается ровно 1 раз.

В первую очередь рассмотрим:

- прямой проход;
- обратный проход.

Прямой проход (preorder)

Алгоритм preorder(T,v):
    выполнить "обращение" к узлу v
     for для каждого узла w, дочернего к v **do**
          выполнить preorder(T,w)

```
public static String preorderPrint(InspectableTree T, Position v)
{   String s=v.GetElement().ToString();
   PositionIterator children = T.Children(v);
   while (children.HasNext())
      s += "" + preorderPrint(T, children.NextPosition());
   return s;
}
```

Вычислительная сложность – O(n)

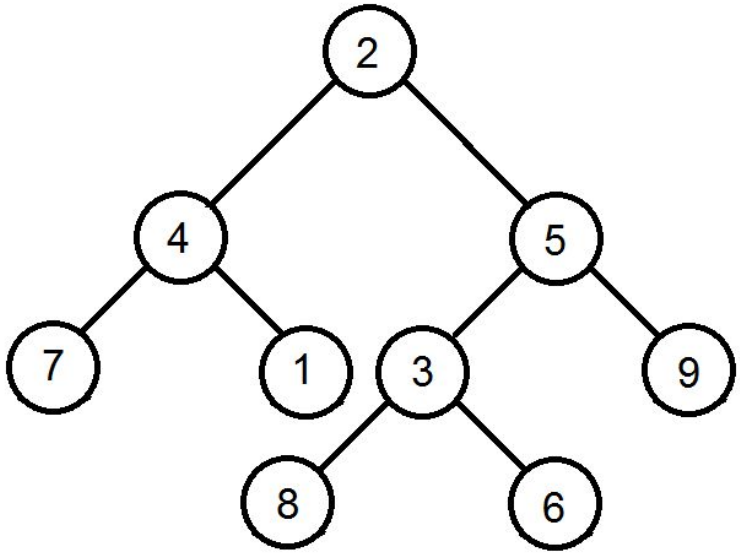
Обратный обход (postorder)

Алгоритм postorder(r,v):
    for для каждого узла w, дочернего к v **do**
        выполнить postorder(T,w)
    выполнить «обращение» к узлу v

```
public static String postorderPrint(InspectableTree T, Position v)
{   String s = "";
    PositionIterator children = T.Children(v);
    while (children.HasNext())
        s += postorderPrint(T, children.NextPosition()) + ""; s +=
     v.Element();
    return s;
}
```

Вычислительная сложность – O(n)

Прямой и обратный проходы



Прямой: 2 4 7 1 5 3 8 6 9

Обратный: 7 1 4 8 6 3 9 5 2

Когда требуется прямой или обратный проход?

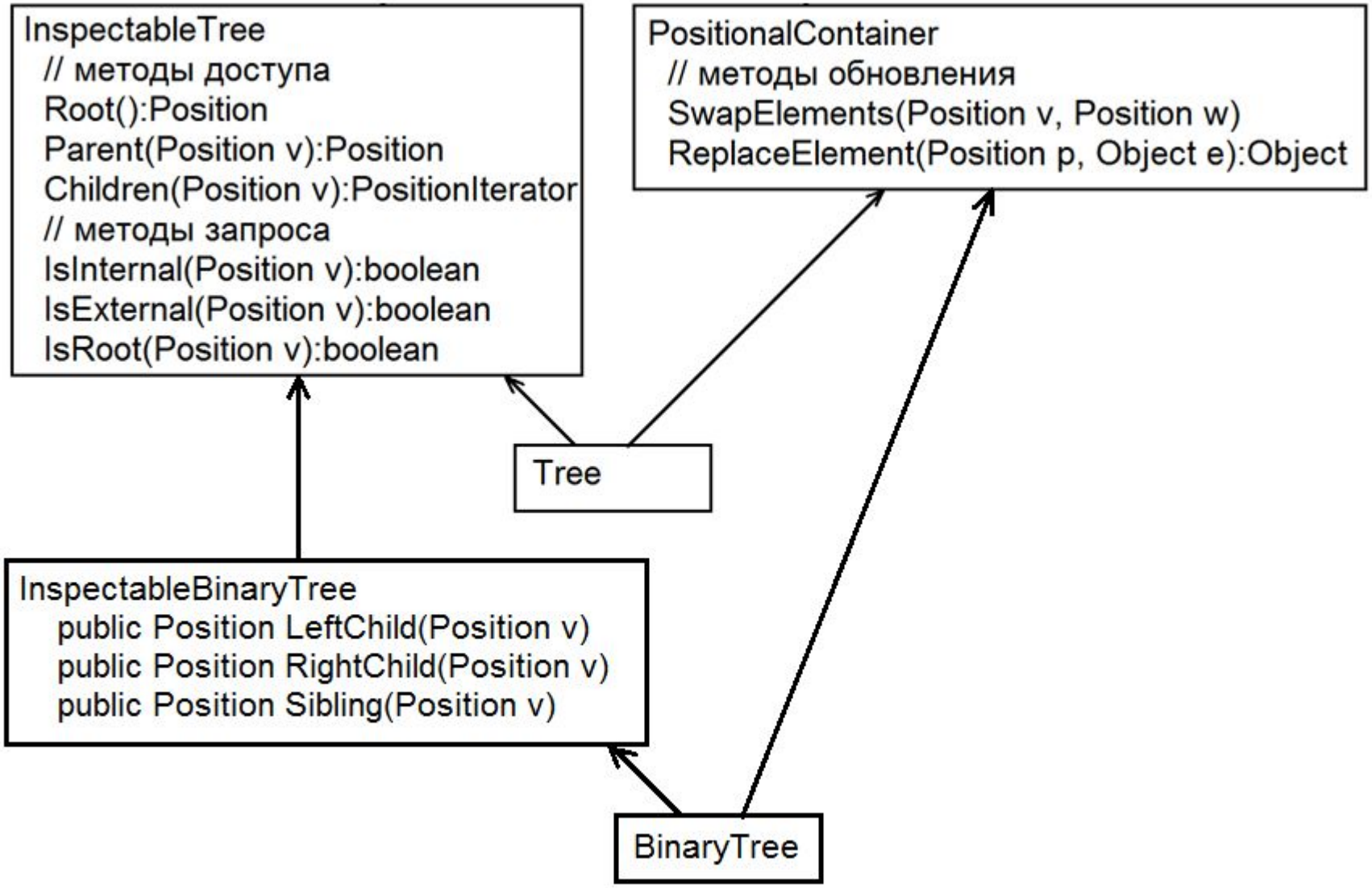
Бинарное дерево

Правильное *бинарное дерево* - упорядоченное дерево, в котором каждый составной узел имеет два дочерних элемента.

Три дополнительных метода доступа:

- LeftChild(*v*): возвращает левый дочерний элемент узла v; ошибка возникает, если *v* — простой узел.
  Input: позиция, ***Output***: позиция.
- RightChild(*v*): возвращает правый дочерний элемент узла *v*; ошибка возникает, если *v* — простой узел.
  Input: позиция, ***Output***: позиция.
- Sibling(*v*): возвращает соседний узел (брата) узла *v*; ошибка возникает, если *v* - корень.
  Input: позиция, ***Output***: позиция.

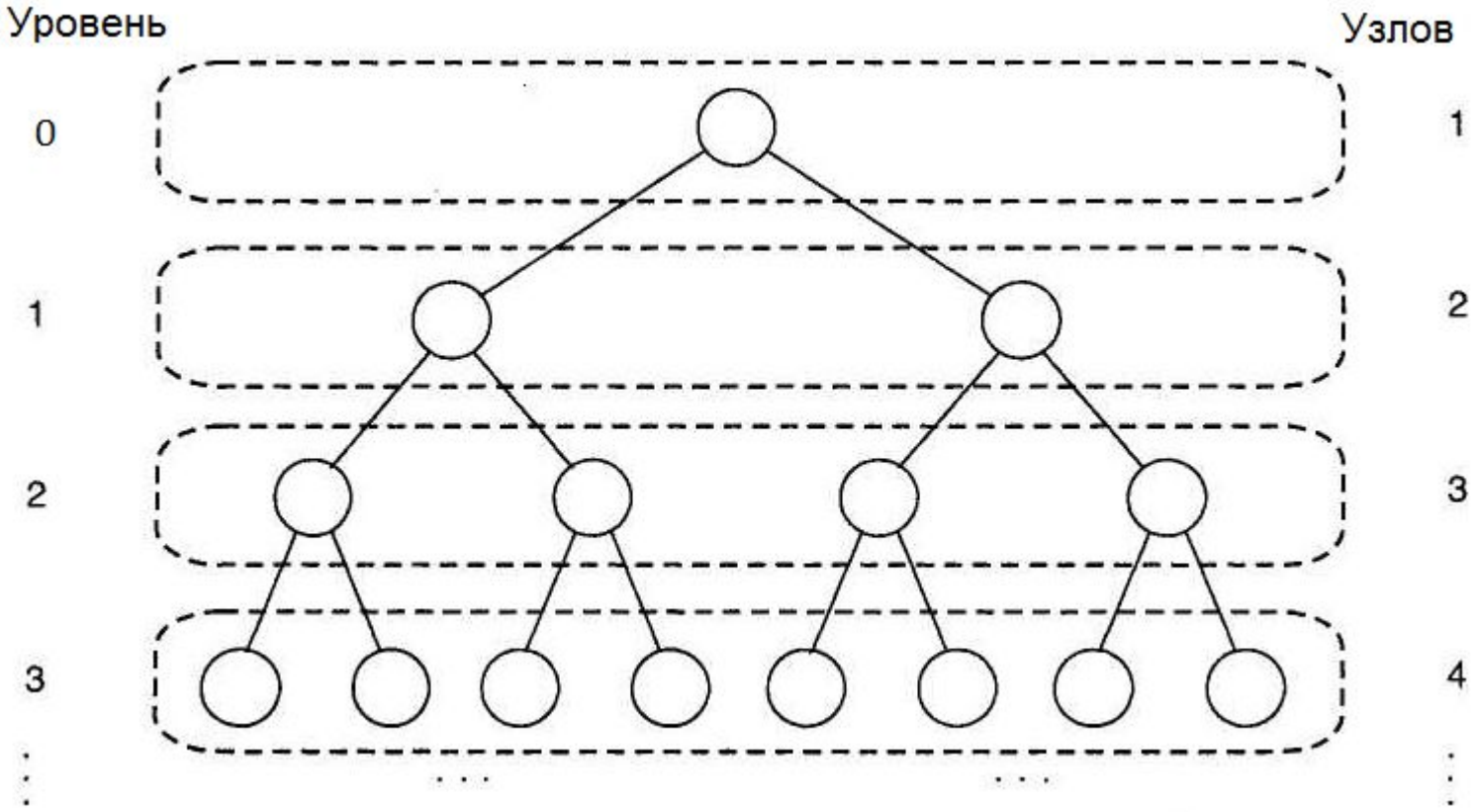
Структура интерфейсов для АТД «Бинарное дерево»



Свойства бинарного дерева

Уровень *d* дерева *T* - все узлы дерева *T,* расположенные на одной глубине *d.*

Уровень *d бинарного дерева* содержит максимум 2^d



Свойства бинарного дерева

Утверждение 6.3. Допустим, *T* является бинарным (правильным) деревом с количеством узлов *n* и высотой *h.* Тогда *T* имеет следующие свойства:

1) количество простых узлов дерева *T* - $[h+1, 2^h]$

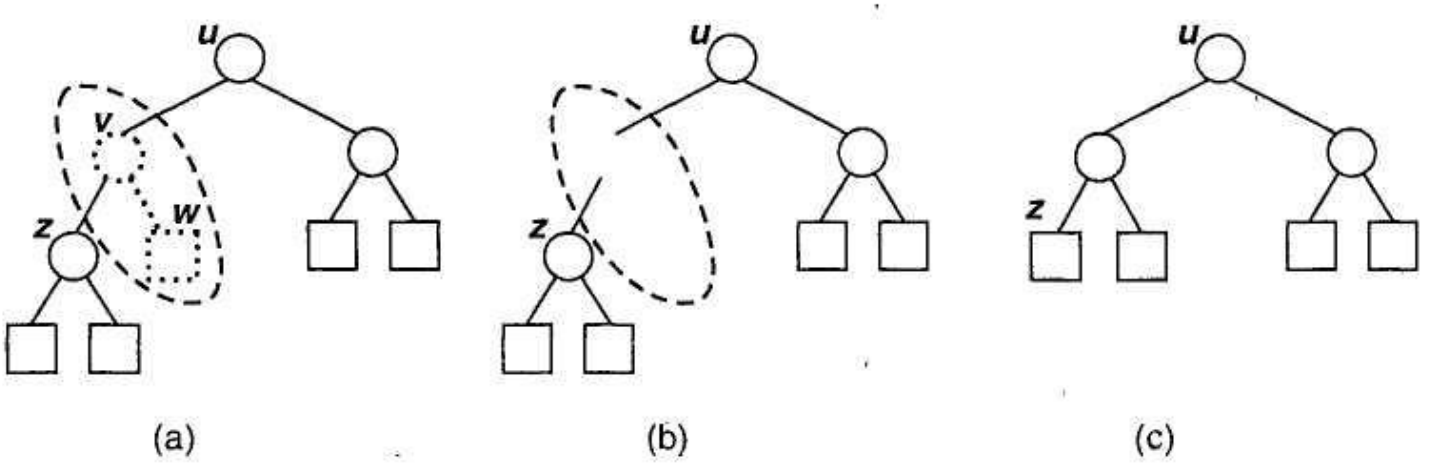
2) количество составных узлов дерева *T* - $[h, 2^h-1]$

3) общее количество *n* узлов дерева *T* - $[2^h - 1, 2^{h+1} – 1]$

4) высота дерева *T* - $[\log(n+1)-1, (n-1)/2]$

Свойства бинарного дерева

Утверждение 6.4. В бинарном (правильном) дереве T количество простых узлов на единицу больше количества составных узлов.



Операция RemoveAboveExternal(w), удаляющая простой узел и его родителя и иллюстрирующая обоснование утверждения 6.4

Прямой проход бинарного дерева

Алгоритм binaryPreorder(T, v):

выполнить обращение к узлу v

if v составной узел **then**

{рекурсивное прохождение левой ветви}

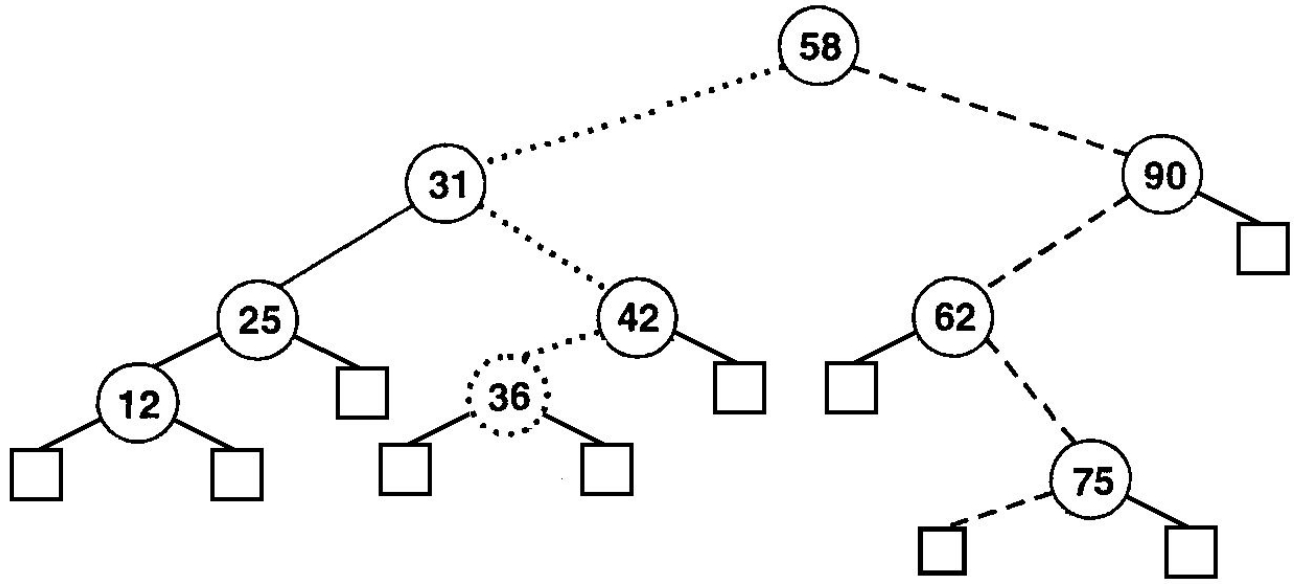
binaryPreorder(T, T.LeftChild(v))

{рекурсивное прохождение правой ветви}

binaryPreorder(T, T.RightChild(v))

Поисковые бинарные деревья

Бинарное поисковое дерево - дерево, в котором каждый составной узел *v* содержит элемент *e*, так что элементы, хранимые в левой ветви *v*, меньше или равны *e*, а элементы, хранимые в правой ветви *v*, больше или равны *e*. Простые узлы не содержат э



Время поиска в бинарном поисковом дереве [***O***(log *n*), ***O***(*n*)]

Обратный проход бинарного дерева

Алгоритм binaryPostorder(T, v):
  if v составной узел **then**
    {рекурсивное прохождение левой ветви}
    binaryPostorder(T, T.LeftChild(v))
    {рекурсивное прохождение правой ветви}
    binaryPostorder(T, T.RightChild(v))
  выполнить обращение к узлу v

Алгоритм evaluateExpression(*T*, *v*):
  if *v* составной узел, хранящий оператор *o*, **then**
    x ← evaluateExpression(T, T.LeftChild(v))
    y ← evaluateExpression(T, T.RightChild(v))
    return x *o* y
  else
    return значение, хранимое в *v*

Симметричный проход бинарного дерева

Алгоритм Inorder(*T*, *v*):

if *v* составной узел **then**

{рекурсивное прохождение левой ветви}

inorder(T, T.LeftChild(v))

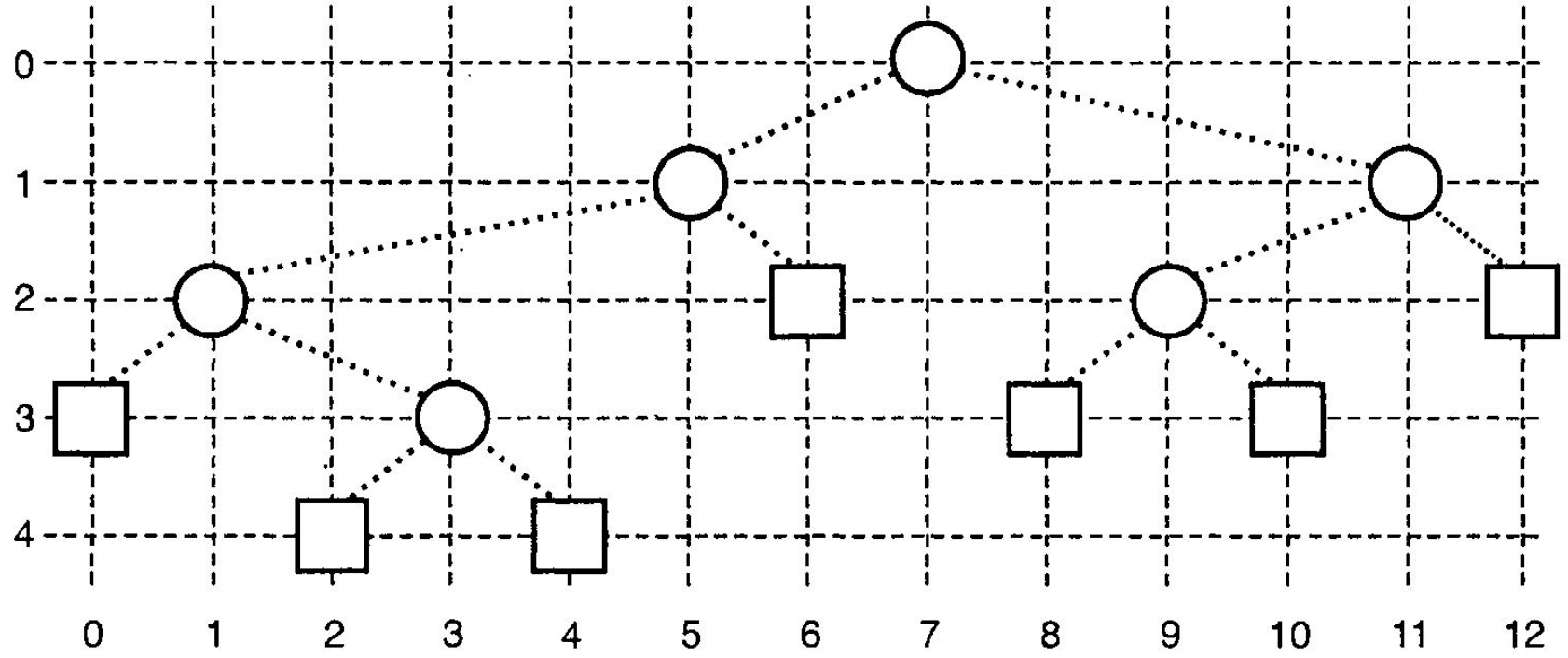
выполнить обращение к узлу v

if v составной узел **then**

{рекурсивное прохождение правой ветви}

inorder(T, T.RightChild(v))

Вычисление схемы бинарного дерева



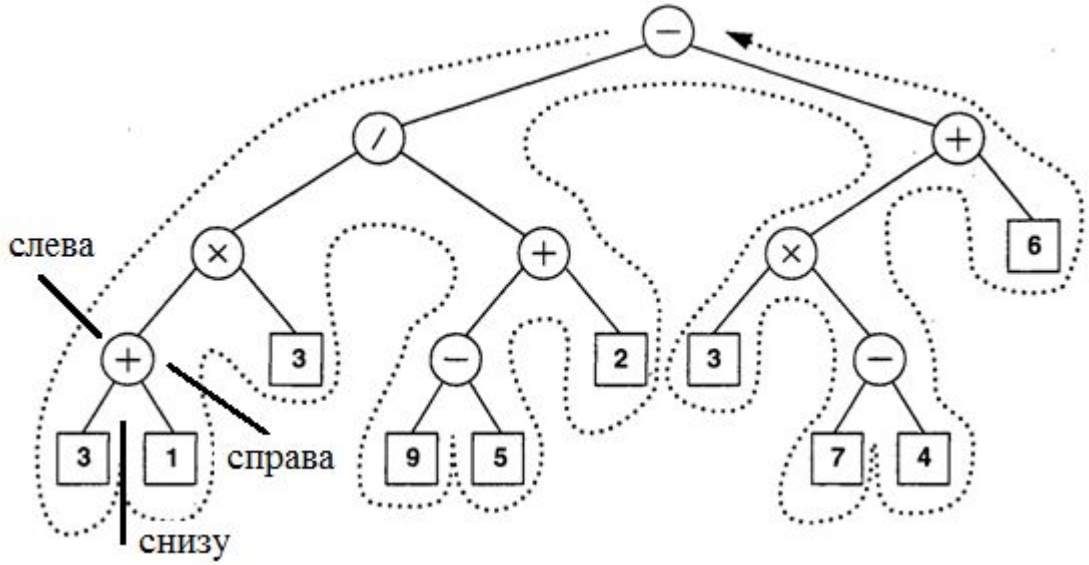
- $x(v)$ равно количеству узлов, пройденных до обращения к v при симметричном проходе дерева T;
- $y(v)$ равно глубине узла v в дереве T.

Унифицированная среда прохода дерева

Алгоритмы прохода дерева унифицируются в виде единого обобщенного подхода при отсутствии требования об одноразовом обращении к узлу. Полученный в результате метод прохода будет называться *«проходом по Эйлеру» (Euler tour traversal)*

- Каждый узел *v* дерева *T* при эйлеровом проходе будет встречаться трижды:
- «слева» (до прохода вдоль левой ветви *v*);
- «снизу» (когда окажемся между двумя ветвями *v*);
- «справа» (при проходе вдоль правой ветви *v*).
- Если узел *v* простой (пустой), то эти обращения выполняются одновременно.

Унифицированная среда прохода дерева



Алгоритм eulerTour(*T*, *v*):
выполнить обращение к узлу *v* слева
if *v* составной узел **then**
рекурсивно обойти левую ветвь узла *v* с помощью eulerTour(*T*, *T*.LeftChild(*v*))
выполнять обращение к *v* снизу
if *v* составной узел **then**
рекурсивно обойти правую ветвь узла *v* с помощью eulerTour(*T*, *T*.RightChild(*v*))
выполнять обращение к *v* справа