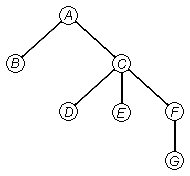
*Деревом* будем называть конечное множество T, состоящее из одного или более узлов, таких что:  
   а) Имеется один специальный узел, называемый *корнем* данного дерева.  
   б) Остальные узлы (исключая корень) содержатся в m >= 0 попарно непересекающихся подмножествах T1, T2, …, Tm, каждое из которых в свою очередь является деревом. Деревья T1, T2, …, Tm называются *поддеревьями* данного дерева.

Это определение является рекурсивным. Если коротко, то дерево это множество, состоящее из корня и присоединенных к нему поддеревьев, которые тоже являются деревьями. Дерево определяется через само себя. Однако данное определение осмысленно, так как рекурсия конечна. Каждое поддерево содержит меньше узлов, чем содержащее его дерево. В конце концов, мы приходим к поддеревьям, содержащим всего один узел.



Узлы, не содержащие поддеревьев, называются *концевыми узлами* или *листьями*. Множество непересекающихся деревьев называется *лесом*. Например, лес образуют поддеревья, исходящие из одного узла.

Если каждая вершина дерева имеет не более двух потомков, то такое дерево называется *бинарным* или *двоичным*.

**Обходы деревьев**

Во всех алгоритмах, связанных с древовидными структурами встречается одна и та же идея, а именно идея *прохождения* или *обхода дерева*. Это такой способ посещения узлов дерева, при котором каждый узел проходится точно один раз. При этом получается линейная расстановка узлов дерева. В частности существует три способа: можно проходить узлы в прямом, обратном и синтаксическом порядке. Будем рассматривать бинарные деревья.

Структуру бинарного дерева будем описывать следующим образом:

type ukaz = ^tree;

tree = record

mark: integer;

left: ukaz;

right: ukaz;

end;

**Прямой обход (префиксный, в глубину «сверху вниз»)**

Алгоритм:

1. Начать с корня дерева.
2. Пометитьтекущую вершину.
3. Совершить прямой обход левого поддерева.
4. Совершить прямой обход правого поддерева.

Реализация:

procedure preorder(p: ukaz; k: integer);

begin

p^.mark:= k;

if p^.left <> nil then preorder(p^.left, k+1);

if p^.right <> nil then preorder(p^.right, k+1);

end;

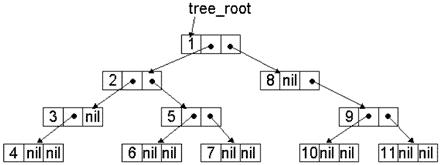
begin

...

preorder(root, 1); {Вызов из тела программы}

...

end.



**Обратный обход (постфиксный, в глубину «снизу вверх»)**

Алгоритм:

1. Начать с корня дерева.
2. Совершить обратный обход левого поддерева.
3. Совершить обратный обход правого поддерева.
4. Пометить текущую вершину.

Реализация:

procedure postorder(p: ukaz; k: integer);

begin

if p^.left <> nil then postorder(p^.left, k+1);

if p^.right <> nil then postorder(p^.right, k+1)

p^.mark:= k;

end;

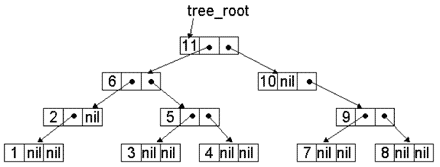
begin

...

postorder(root,1); {Вызов из тела программы}

...

end.



**Синтаксический обход (инфиксный)**

Алгоритм:

1. Начать с корня дерева.
2. Совершить прямой обход левого поддерева.
3. Пометить текущую вершину.
4. Совершить прямой обход правого поддерева.

Реализация:

procedure syntorder(p: ukaz; k: integer);

begin

if p^.left <> nil then syntorder(p^.left, k+1);

p^.mark:= k;

if p^.right <> nil then syntorder(p^.right, k+1);

end;

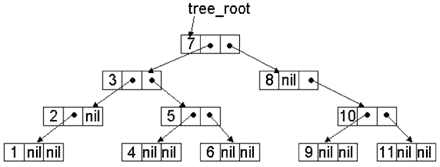
begin

...

syntorder(root,1); {Вызов из тела программы}

...

end.



**Обход в ширину**

Последовательность обхода:

1. Пометить вершину 0-го уровня (корень дерева).
2. Пометить все вершины 1-го уровня.
3. Пометить все вершины 2-го уровня.
4. ...

Алгоритм:

1. Занести в очередькорень дерева.
2. Пока очередь не станет пустой, повторять следующие действия.
3. Пометить и удалить первый элемент из головы очереди.
4. Добавить в хвост очереди всех потомков удаленной вершины.

Реализация:

Для простоты реализации пополним структуру дерева полем next: ukaz, которое будет служить для связки очереди.

procedure BFS(root: ukaz);

var head, tail: ukaz;

k: integer;

begin

head:= root;

tail:= root;

k:= 0;

repeat

tail^.next:= head^.left;

if head^.left <> nil then tail:= tail^.next;

tail^.next:= head^.right;

if head^.right <> nil then tail:= tail^.next;

inc(k);

head^.mark:= k;

head:= head^.next

until head = nil;

end;

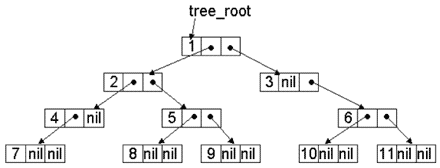
begin

...

BFS(root); {Вызов из тела программы}

...

end.



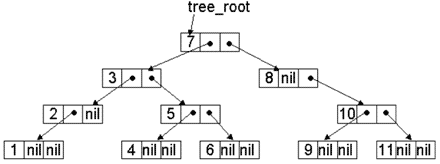
**Двоичное дерево поиска**

*Двоичное дерево поиска* для множества чисел S - это размеченное бинарное дерево, каждой вершине которого сопоставлено число из множества S, причем все пометки удовлетворяют следующим условиям:

1. Существует ровно одна вершина, помеченная любым числом из множества S.
2. Все пометки левого поддерева строго меньше, чем пометка текущей вершины.
3. Все пометки правого поддерева строго больше, чем пометка текущей вершины.

Если выражаться простым языком, то структура дерева двоичного поиска подчиняется простому правилу: «если больше - направо, если меньше – налево**»**.

Например, для набора чисел 7, 3, 5, 2, 8, 1, 6, 10, 9, 4, 11 получится такое дерево



Для того чтобы правильно учесть повторения чисел, можно ввести дополнительное поле, которое будет хранить количество вхождений для каждого числа.

С помощью двоичного дерева поиска можно организовать эффективный способ поиска, который значительно эффективнее поиска по списку.

Поиск в упорядоченном дереве выполняется по следующему рекурсивному алгоритму: если дерево пусто, то искомый элемент не найден, иначе сравнить искомый ключ с ключом в корне дерева:

– если ключи совпадают, поиск завершен;

– если ключ в корне больше искомого, выполнить поиск в левом поддереве;

– если ключ в корне меньше искомого, выполнить поиск в правом поддереве.