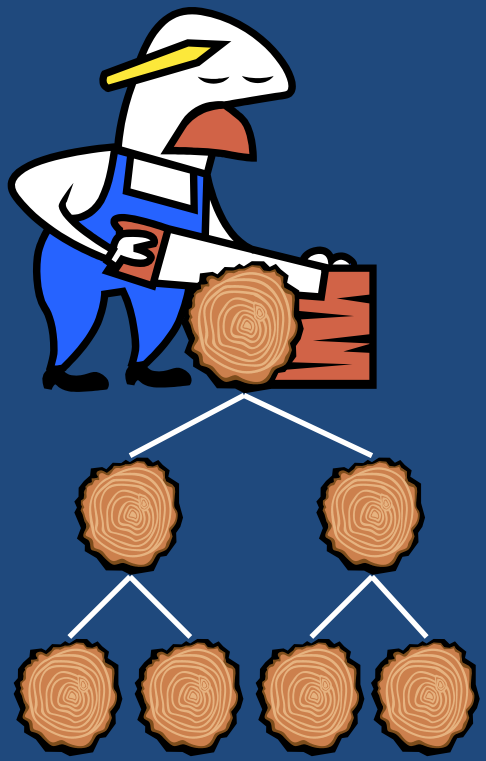


БИНАРНЫЕ ДЕРЕВЬЯ

Всякое дерево, не приносящее плода доброго, срубают и бросают в огонь.

Ев. от Матфея, 19:30

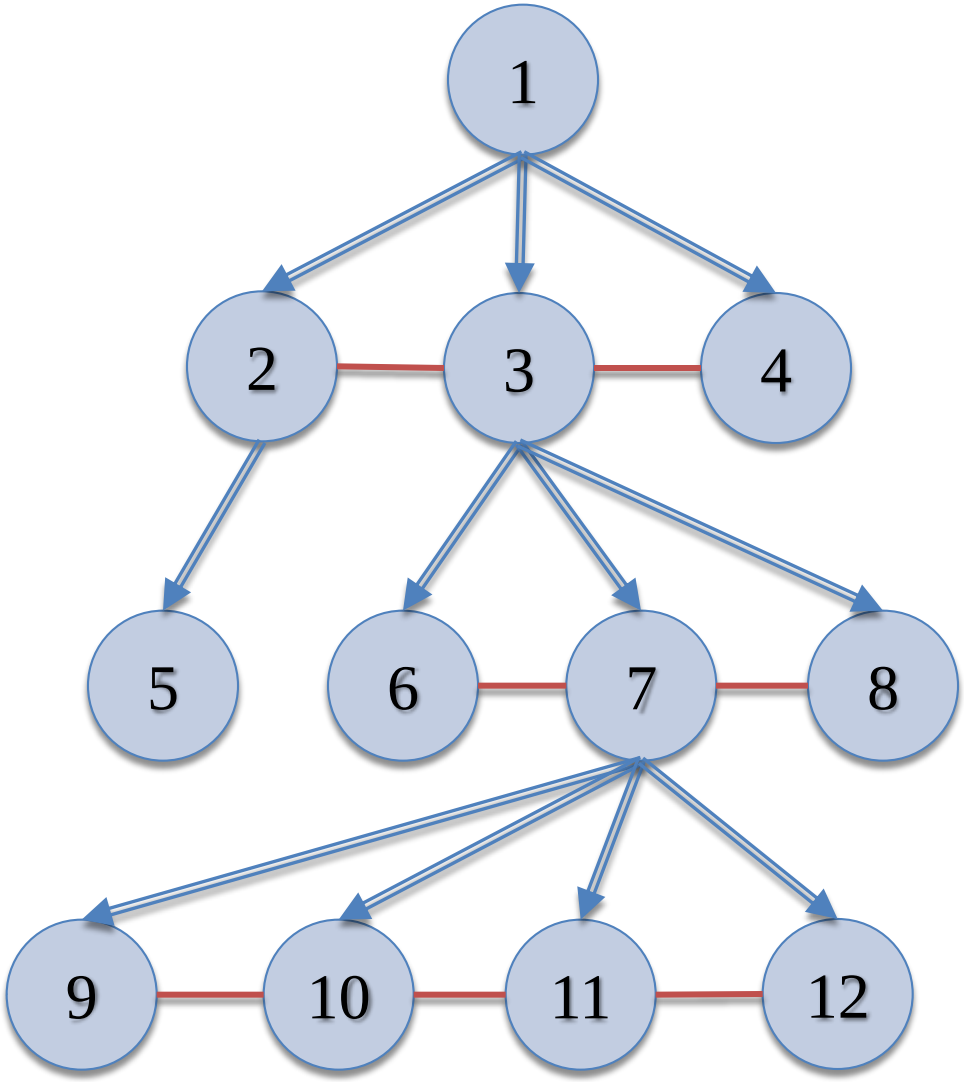
Содержание

- Деревья и бинарные деревья
- Бинарные деревья поиска

Деревья и бинарные деревья

- *Дерево* – связный ациклический граф.
  - *Граф* – совокупность непустого множества вершин и множества пар вершин.
  - *Связность* – между любой парой вершин существует по крайней мере один путь.
  - *Ацикличность* – между любой парой вершин существует только один путь.
- *Бинарное дерево* – упорядоченное дерево, в котором с каждой вершиной связаны не более двух вершин.

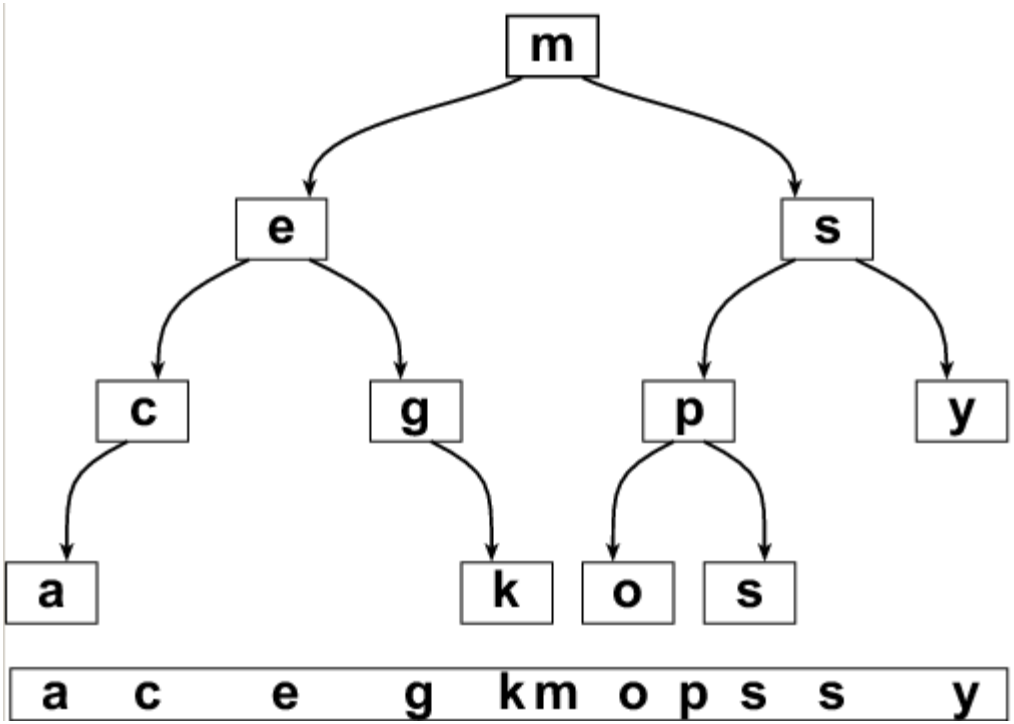
Преобразование дерева в бинарное дерево



1. Соединить братьев каждого уровня.
2. Стереть все вертикальные связи, кроме связей каждого отца к первому сыну.

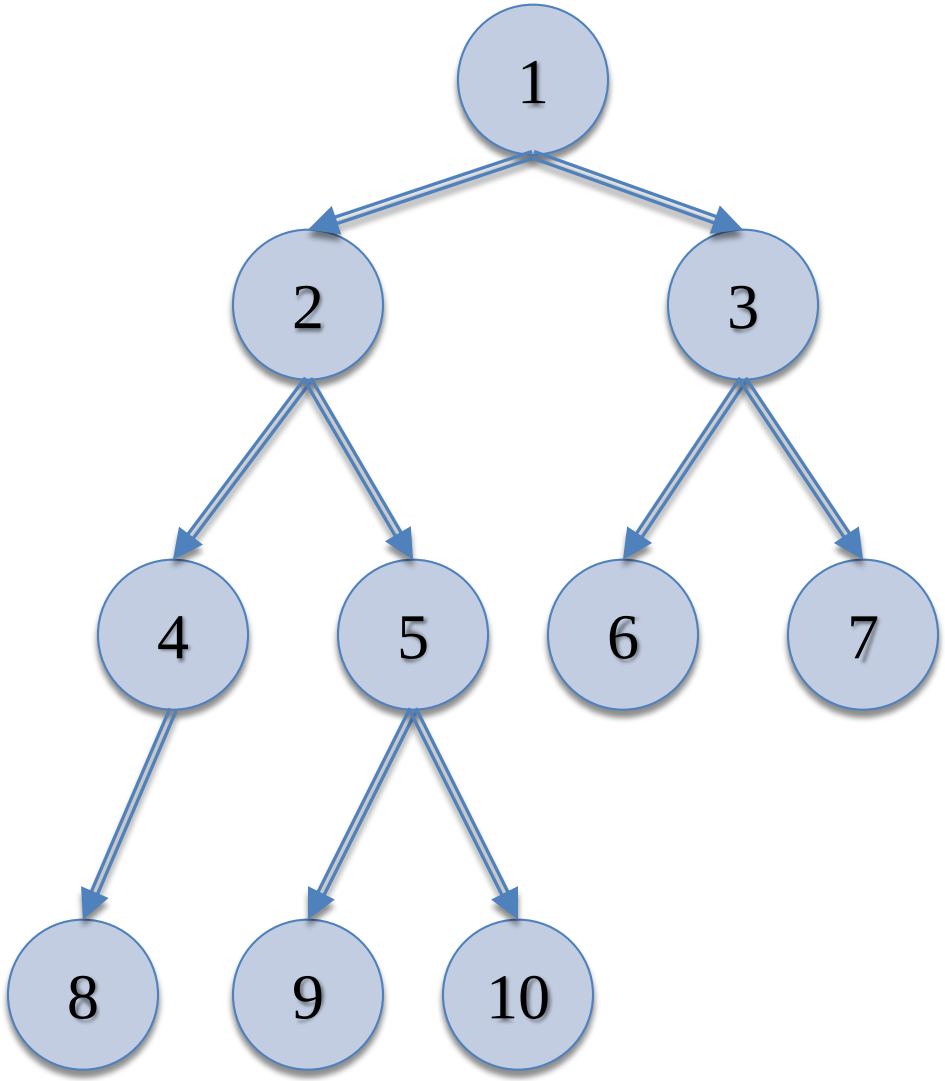
Бинарное дерево

- <дерево> ::= (<данное> <лев. поддерево> <прав. поддерево>) |
  <пусто>
  <лев. поддерево> ::= <дерево>
  <прав. поддерево> ::= <дерево>




```
(m
  (e
    (c
      (a NIL NIL)
      NIL )
    (g
      NIL
      (k NIL NIL)
    )
  )
  (s
    (p (o NIL NIL) (s NIL NIL) )
    (y NIL NIL)
  )
)
```

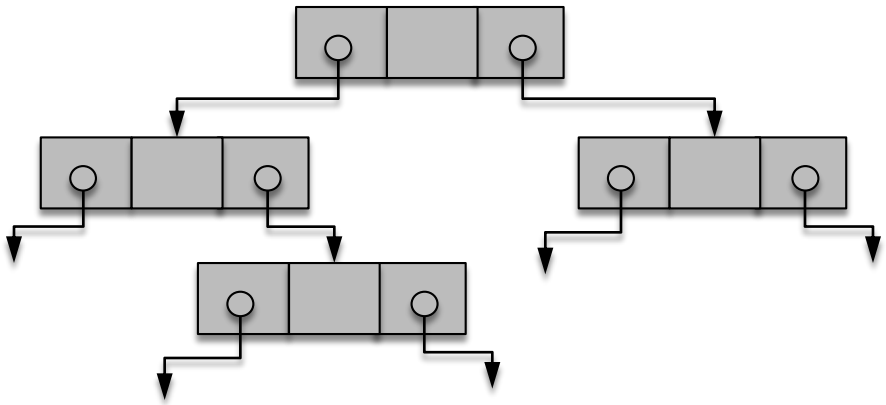
Бинарное дерево: терминология



- Корень, листья
- Отец, сыновья, братья
- Предок, потомок
- Высота

Реализация бинарного дерева


```
type
    TInfo=…;{ Тип информации в вершине }
    PNode=^TNode; { Указатель на вершину }
    TNode=record
        Info: TInfo;              { Информационное поле }
        Left, Right: PNode; { Указатели на поддеревья }
    end;
```



Реализация бинарного дерева


```
unit BinTree;
interface
type    … { Интерфейс (определение) структуры данных }
{  Интерфейс методов (подпрограмм обработки структуры данных) }
procedure Init;
procedure Destroy;
function  Root: PNode;
…
implementation
var TheRoot: PNode; { Реализация структуры данных }
{ Реализация методов }
…
end.
```

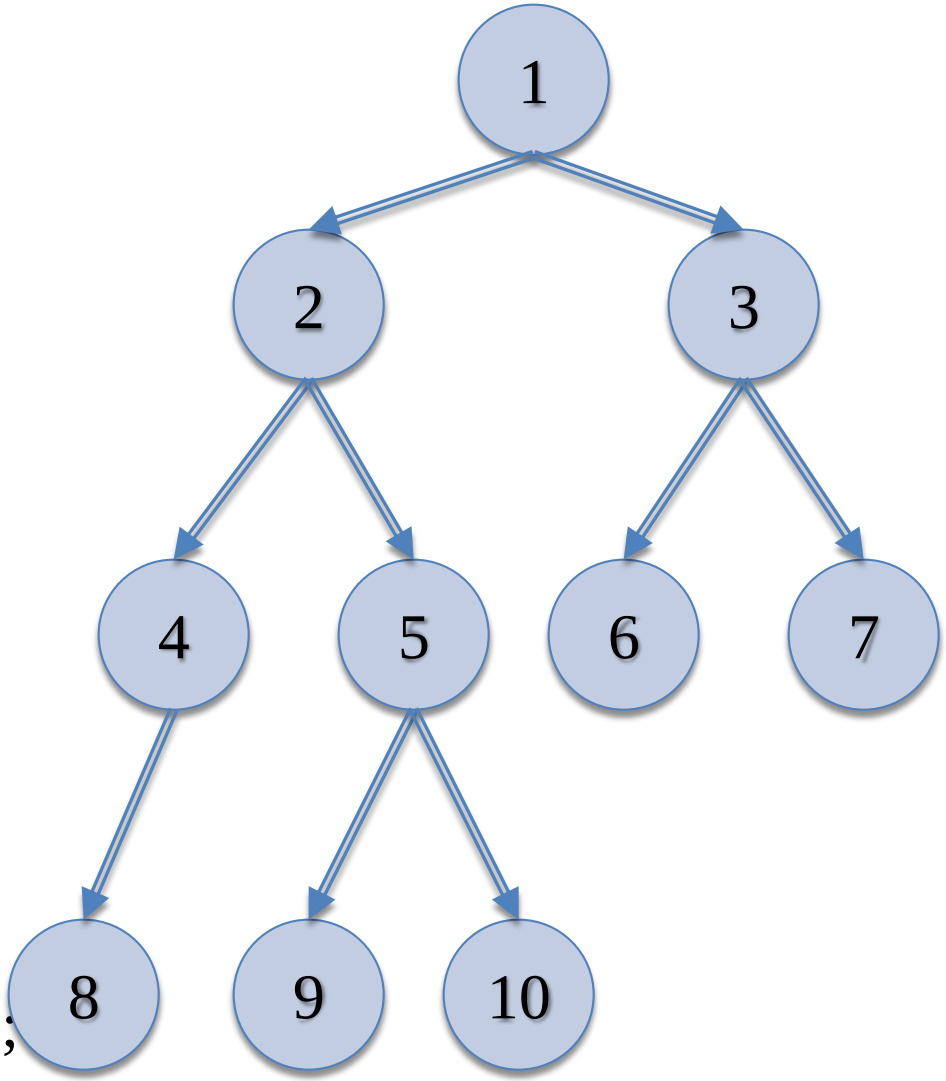
Создание бинарного дерева


```
procedure Init;
begin
    TheRoot:=NIL;
end;
procedure AppendLeft(L: PNode; E: TInfo);
var N: PNode;
begin
    New(N);
    N^.Info:=E;
    N^.Left:=NIL; N^.Right:=NIL;
    L^.Left:=N;
end;
```

```
function Root: PNode;
begin
    Result:=TheRoot;
end;
procedure MakeRoot(E: TInfo);
begin
    New(TheRoot);
    TheRoot^.Info:=E;
    TheRoot^.Left:=NIL;
    TheRoot^.Right:=NIL;
end;
```

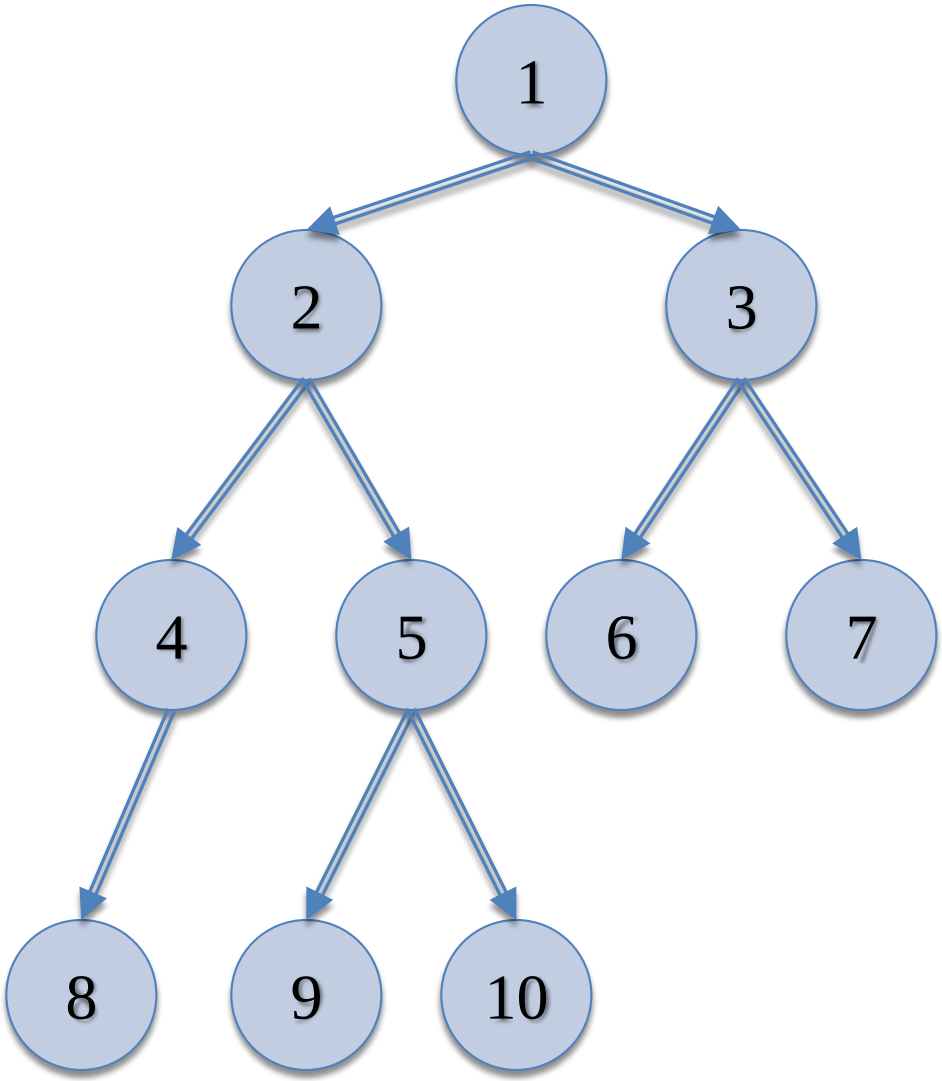
Создание бинарного дерева


```
Init;
MakeRoot(1);
AppendLeft(Root, 2);
AppendRight(Root, 3);
AppendLeft(Root^.Left, 4);
AppendRight(Root^.Left, 5);
AppendLeft(Root^.Right, 6);
AppendRight(Root^.Right, 7);
AppendLeft(Root^.Left^.Left, 8);
AppendLeft(Root^.Left^.Right, 9);
AppendRight(Root^.Left^.Right, 10);
```



Обходы бинарного дерева

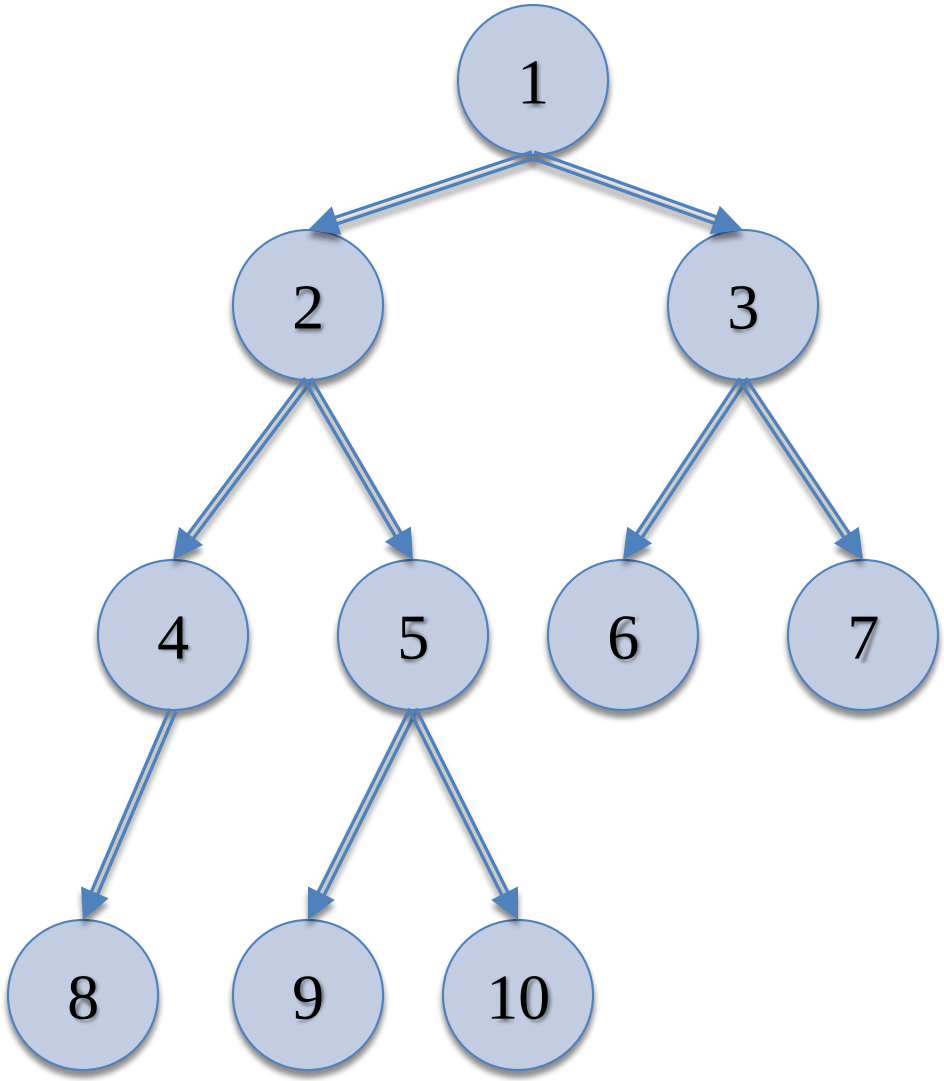
- Прямой
  - Посетить корень
  - Посетить левое поддерево
  - Посетить правое поддерево
- Обратный
  - Посетить левое поддерево
  - Посетить корень
  - Посетить правое поддерево
- Концевой
  - Посетить левое поддерево
  - Посетить правое поддерево
  - Посетить корень



Прямой обход дерева

1, 2, 4, 8, 5, 9, 10, 3, 6, 7

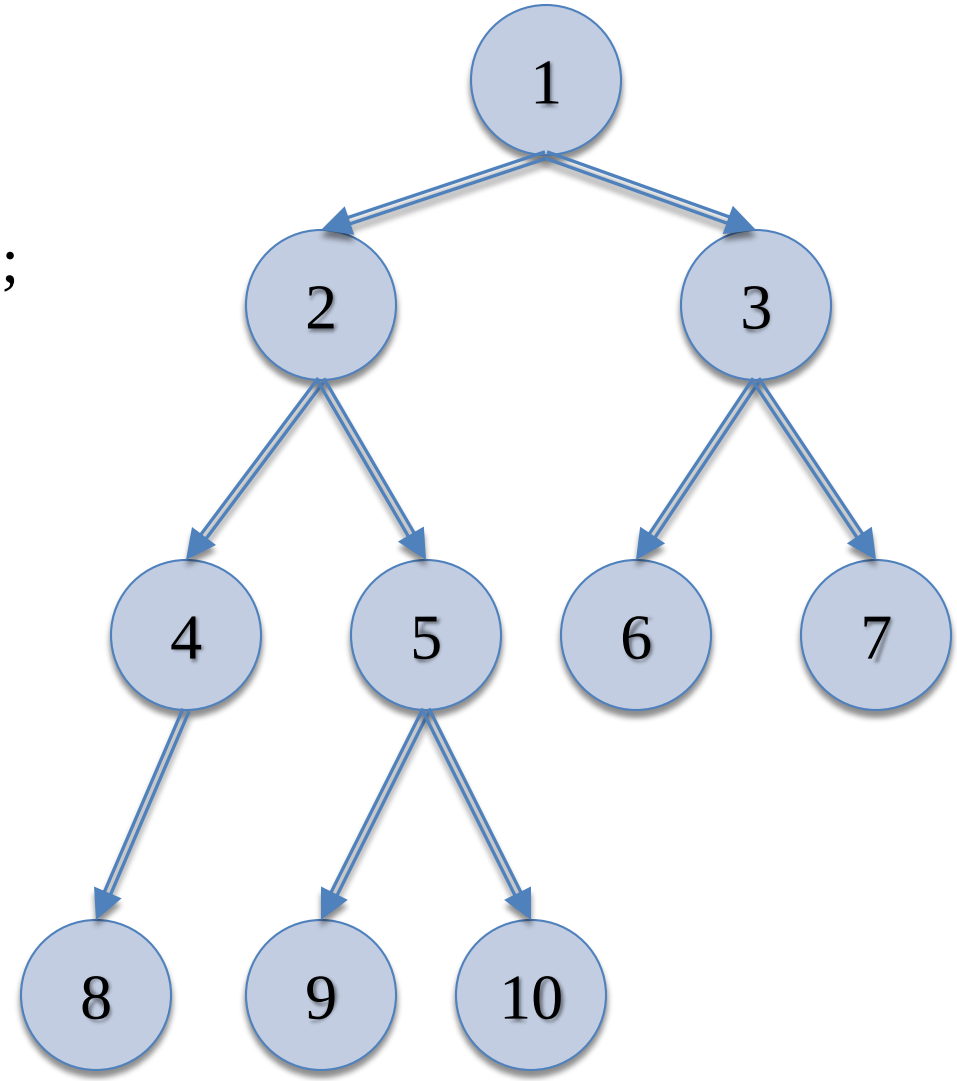
```
procedure PreOrder(Root: PNode);
begin
    if Root=Nil then break;
    Write(Root^.Info);
    PreOrder(Root^.Left);
    PreOrder(Root^.Right);
end;
```



Обратный обход дерева

8, 4, 2, 9, 5, 10, 1, 6, 3, 7

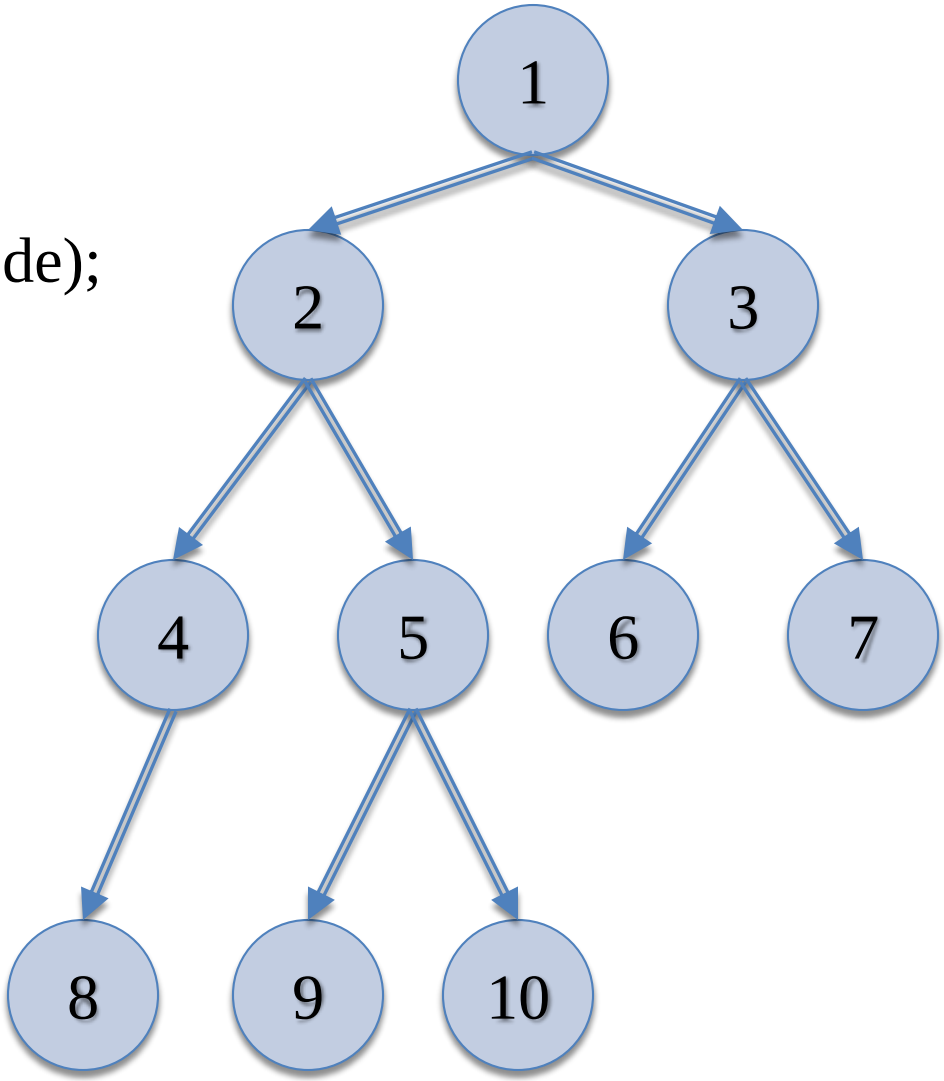
```
procedure PostOrder(Root: PNode);
begin
    if Root=Nil then break;
    PostOrder(Root^.Left);
    Write(Root^.Info);
    PostOrder(Root^.Right);
end;
```



Концевой обход дерева

8, 4, 9, 10, 5, 2, 6, 7, 3, 1

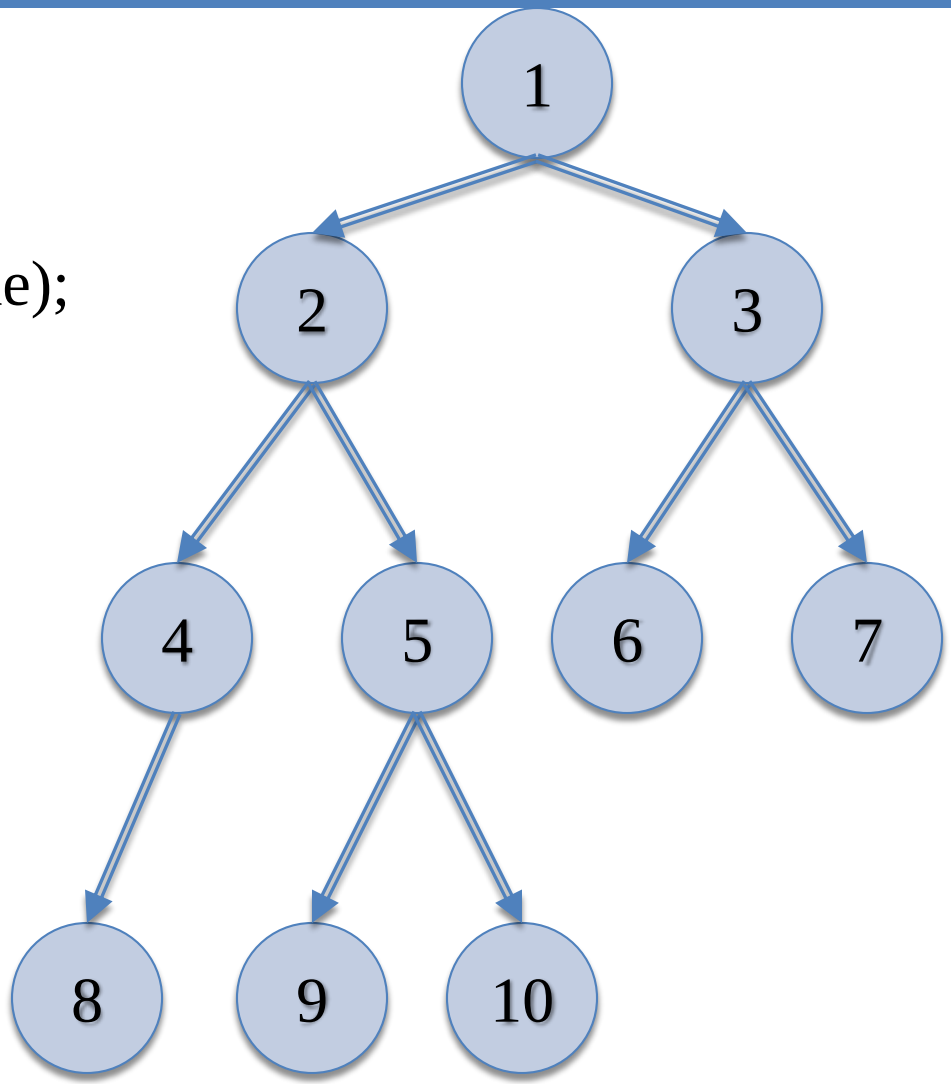
```
procedure ReverseOrder(Root: PNode);
begin
    if Root=Nil then break;
    ReverseOrder(Root^.Left);
    ReverseOrder(Root^.Right);
    Write(Root^.Info);
end;
```



Уничтожение дерева

Концевой обход с уничтожением вершины

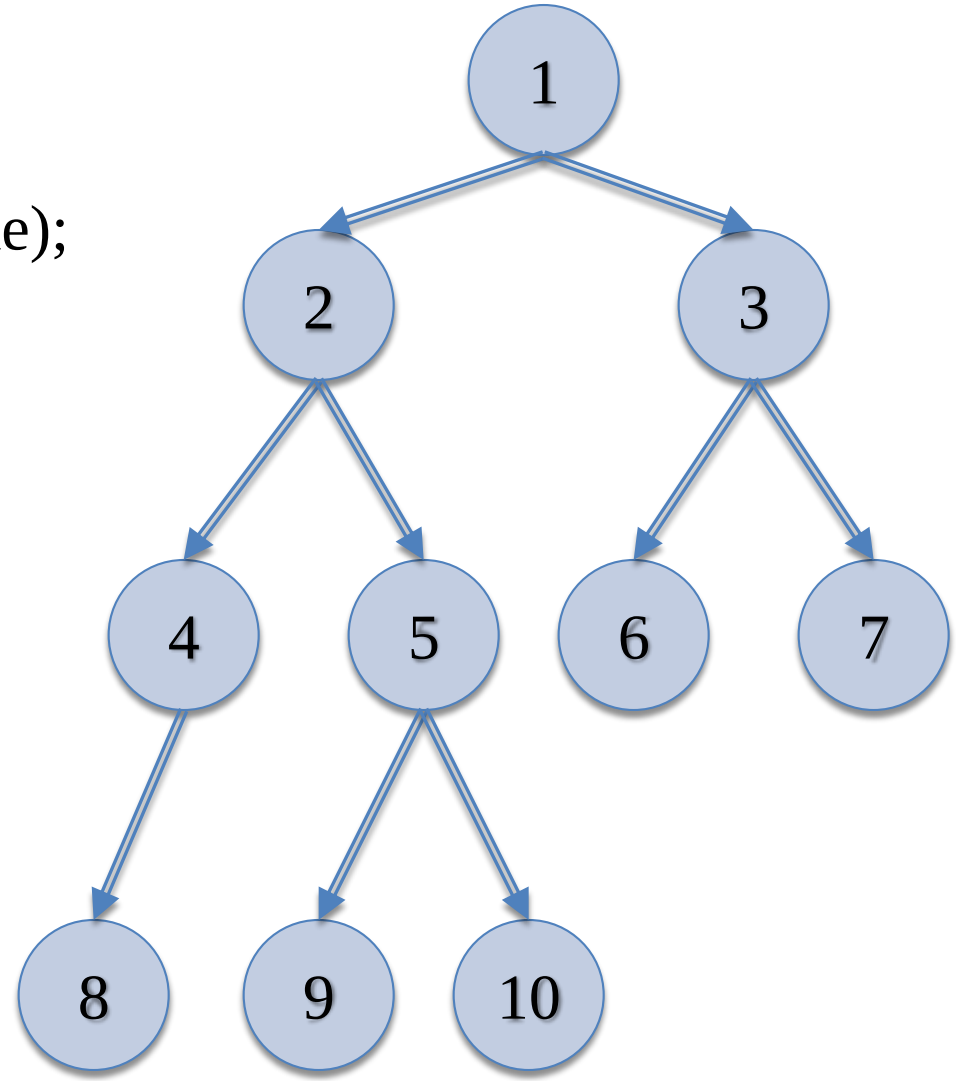
```
procedure DestroyTree(Root: PNode);
begin
    if Root=Nil then break;
    DestroyTree(Root^.Left);
    DestroyTree(Root^.Right);
    Dispose(Root);
end;
```



Уничтожение дерева

Прямой обход с уничтожением вершины

```
procedure DestroyTree(Root: PNode);
var L, R: PNode;
begin
    if Root=Nil then break;
    L:=Root^.Left;
    R:=Root^.Right;
    Dispose(Root);
    DestroyTree(L);
    DestroyTree(R);
end;
```



Количество вершин в дереве


```
function NodeCount(R: PNode): Integer;
var cL, cR: Integer;
begin
    if R=NIL then begin
        Result:=0;
        break;
    end;
    cL:=NodeCount(R^.Left);
    cR:=NodeCount(R^.Right);
    Result:=1+cL+cR;
end;
```

Количество вершин с заданным значением


```
function Count(R: PNode; E: TInfo): Integer;
var C, cL, cR: Integer;
begin
    if R=NIL then begin
        Result:=0;
        break;
    end;
    if R^.Info=E then C:=1 else C:=0;
    cL:=Count(R^.Left, E);
    cR:=Count(R^.Right, E);
    Result:=C+cL+cR;
end;
```

Высота дерева


```
function Height(R: PNode): Integer;
begin
    if R=NIL then begin
        Result:=-1;
        break;
    end;
    hL:=Height(R^.Left);
    hR:=Height(R^.Right);
    Result:=1+max(hL,hR);
end;
```

Создание копии дерева

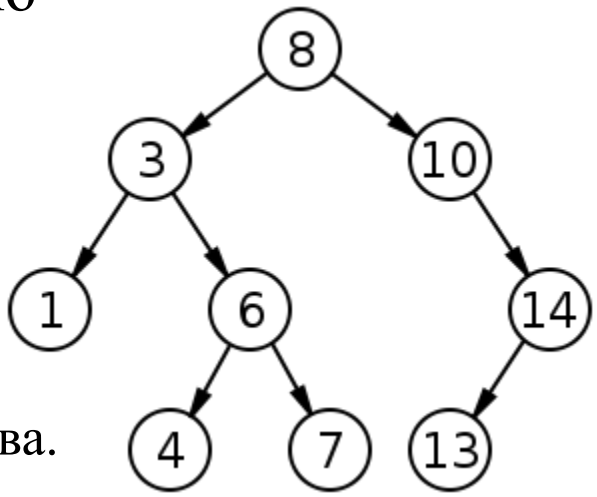

```
procedure Duplicate(R: PNode; var newR: PNode);
begin
    if R=NIL then begin
        newR:=NIL;
        break;
    end;
    New(newR);
    newR^.Info:=R^.Info;
    Duplicate(R^.Left, newR^.Left);
    Duplicate(R^.Right, newR^.Right);
end;
```

Сравнение двух деревьев


```
function AreEqual(R1, R2: PNode): Boolean;
begin
    Result:=((R1=NIL) and (R2=NIL)) or
        ((R1<>NIL) and (R2<>NIL) and (R1^.Info=R2^.Info) and
            AreEqual(R1^.Left, R2^.Left) and
                AreEqual(R1^.Right, R2^.Right));
end;
```

Бинарное дерево поиска

- Бинарное дерево поиска – бинарное дерево, вершины которого подчиняются отношению порядка и включаются в дерево в соответствии со следующим правилом:
  - Если дерево пустое, то создать дерево, корень которого хранит значение включаемой вершины.
  - Если дерево не пустое, то сравнить значение включаемой вершины со значением в корне дерева.
    - Если значение включаемой вершины меньше значения в корне, то включить вершину в левое поддерево (рекурсивно).
    - Если значение включаемой вершины больше значения в корне, то включить вершину в правое поддерево (рекурсивно).



Включение вершины в дерево поиска


```
procedure Include(var R: PNode; E: TInfo);
begin
    if R=NIL then begin
        New(R); R^.Info:=E; R^.Left:=NIL; R^.Right:=NIL;
        break;
    end;
    if E<R^.Info then
        Include(R^.Left, E)
    else
        if E>R^.Info then
            Include(R^.Right, E);
end;
```

Поиск вершины в дереве поиска


```
function Find(R: PNode; E: TInfo): Pnode;
begin
    if R=NIL then begin
        Result:=NIL;
        break;
    end;
    if E<R^.Info then
        Result:=Find(R^.Left, E)
    else
        if E>R^.Info then
            Result:=Find(R^.Right, E)
        else
            Find:=R;
end;
```

Поиск вершины в дереве поиска


```
function Find(R: PNode; E: TInfo): PNode;
begin
    while R<>NIL do begin
        if E<R^.Info then
            R:=R^.Left
        else
            if E>R^.Info then
                R:=R^.Right
            else
                break;
    end;
    Result:=R;
end;
```

Слияние деревьев поиска


```
procedure Merge(var R1: PNode; R2: PNode);
begin
    if R2=NIL then break;
    Include(R1, R2^.Info);
    Merge(R1, R2^.Left);
    Merge(R1, R2^.Right);
end;
```

Удаление вершины


```
procedure DeleteNode
        (var Root: PNode; X: TInfo);
var
   V: PNode;

procedure Descend(var R: PNode);
begin
   if R^.Right<>NIL then
      Descend(R^.Right)
   else begin
      V^.Info:=R^.Info;
      V:=R;
      R:=R^.Left;
   end;
end;
```

```
begin
   if Root=NIL then break;
   if Root^.Info>X then
      DeleteNode(Root^.Left,X)
   else
      if Root^.Info<X then
         DeleteNode(Root^.Right,X)
      else begin
         V:=Root;
         if V^.Right=NIL then
            Root:=V^.Left
         else
            if V^.Left=NIL then
               Root:=V^.Right
            else
               Descend(V^.Left);
         Dispose(V);
      end;
end;
```