# Задание бинарных деревьев с помощью массивов

Обходы деревьев

# Содержание



# Задание полного бинарного дерева

В виде массива проще всего представить полное бинарное дерево, так как оно всегда имеет строго определенное количество вершин на каждом уровне, кроме последнего уровня (например, число узлов в дереве, имеющем три уровня, может быть от 4 до 7).

Для задания полного дерева используем три массива: key (для хранения ключей), left (для хранения ссылок на левых потомков вершин), и right (для хранения ссылок на правых потоков вершин).

Пронумеруем вершины полного дерева так, чтобы левый помок любой i – ой вершины имел адрес 2i, а правый потомок любой i – ой вершины имел адрес 2i + 1.

При задании дерева воспользуемся прямым левым обходом. Сначала задаем количество вершин в нашем дереве (n), формируем корень дерева.

Назад Меню

Для каждой i – ой вершины, начиная с корня, проверяем:

1) если 2i <= n, значит, у данной вершины есть левый потомок, т.е. в массив left на i – ое место записываем адрес 2i. Переходим на вершину с адресом 2i. Задаем число, хранящееся в данной вершине и пункт 1) иначе, если 2i > n, значит, у данной вершины нет левого потомка, т.е. в массив left на i – ое место запишем 0 и пункт 2);

2) если 2i+1 < n, значит, у данной вершин есть правый потомок, т.е. в массив right на i – ое место записываем адрес 2i+1. Переходим на вершину с адресом 2i+1. Задаем число, хранящееся в данной вершине и пункт 1) иначе, если 2i+1 > n , значит, у данной вершины нет правого потомка, т.е. в массив right на i – ое место запишем 0. Переходим на предыдущую вершину, для которой условия пункта 1) уже проверены, а условия пункта 2) еще нет.

Программа задания полного дерева на языке Паскаль

Назад

Меню

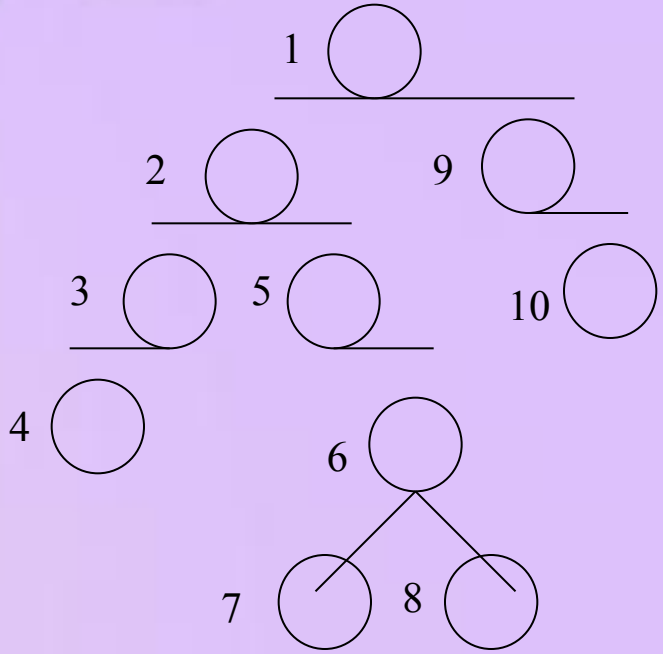
# Задание произвольного бинарного дерево

Часто для решения той или иной задачи требуется задать не полное, а произвольное (неполное) дерево.

Пусть нужно задать произвольное дерево

(см. рис.):

Значения, хранящиеся в узлах дерева, не имеют принципиального значения, т.е. их можно сгенерировать случайным образом. Для нумерации вершин используем прямой левый обход, поэтому вершины пронумеруем соответственно ( см. рис.).




Назад

Меню

В отличие от полного дерева в произвольном дереве на каждом уровне (кроме нулевого уровня) может быть различное количество вершин, поэтому вид дерева нужно задавать вручную. Сделать это можно с помощью последовательности, например, 0 и 1 (можно использовать любые символы), хранящейся в текстовом файле.

Составляется данная последовательность следующим образом: идем по дереву прямым левым обходом:

- встаем на первую вершину и смотрим, слева есть вершина, значит, ставим 1 и переходим на вторую вершину;
- у второй вершины слева есть вершина, ставим 1 и переходим на третью вершину;
- у третьей вершины слева нет вершины, значит, ставим 0, справа у данной вершины тоже нет другой вершины, ставим 0 и возвращаемся на вторую вершину;
- у второй вершины справа есть вершина, ставим 1 и переходим на четвертую вершину и т.д.

Процесс закончится когда мы обойдем все вершины.

В результате в нашем файле должна быть записана следующая последовательность: 1 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 1 0 0.

Назад Меню

После этого остаётся только задать отношения «отец – сын». Для хранения левых и правых ссылок используем массивы left и right соответственно. Заполнение данных массивов происходит следующим образом:

- стоим на первой вершине и считываем из ранее заготовленного файла первый элемент, т.е. 1, а это значит, что у первой вершины есть левый потомок – вершина №2, следовательно, в массив left на первое место запишется 2;
- переходим на вторую вершину и считываем из файла второй элемент, т.е. 1, значит, что у второй вершины есть левый потомок – вершина №3, следовательно, в массив left на второе место запишется 3;
- переходим на третью вершину и считываем из файла третий элемент, т.е. 1, значит, что у третьей вершины есть левый потомок – вершина №4, следовательно, в массив left на третье место запишется 1;
- переходим на четвертую вершину и считываем из файла четвертый элемент, т.е. 0, значит, что у четвертой вершины нет левого потомка, следовательно, в массив left на четвертое место запишется 0;

Назад Меню

• остаемся на четвертой вершине и считываем следующий элемент из файла, т.е. 0, значит, что у четвертой вершины нет правого потомка, следовательно, в массив right на четвертое место запишется 0;

• возвращаемся на третью вершину и считываем следующий элемент из файла, т.е. 0, значит, что у третьей вершины нет правого потомка, следовательно, в массив right на третье место запишется 0;и т.д.

Процесс закончится, когда мы прямым левым обходом обойдем все вершины дерева.

В результате наше дерево будет задано с помощью трех массивов:

Key – в данном массиве будут храниться значения узлов дерева;

Left = [2 3 4 0 0 7 0 0 0 0] – массив правых ссылок;

Right = [9 5 0 0 6 8 0 0 10 0] – массив правых ссылок.

Программа задания произвольного дерева на языке Паскаль

Назад Меню

# Задание двоичного дерева поиска

Рассмотрим задание двоичного дерева поиска с помощью массивов.

Как и в предыдущих случаях для хранения значений узлов, левых и правых ссылок выделяем массивы key, left и right соответственно. Значения элементов, из которых нужно построить дерево поиска удобно хранить в текстовом файле.

Пусть нужно построить дерево поиска для следующего набора элементов: 20 15 32 17 13 23 28 16.

Назад Меню

- Берем первый элемент, он будет корнем нашего дерева. Для корня левую и правую ссылки объявляем равными нулю, т.е. пока кроме корня других вершин нет.
- Берем второй элемент и сравниваем его с корнем: 15<20 и левая ссылка у корня = 0 т.е. слева у 20 нет вершины, поэтому нужно эту вершину создать, записать в неё 15, обнулить левую и правую ссылки второй вершины и создать связь между этими вершинами т.е. в массив left на первое место записать 2.
- Берем третий элемент и сравниваем его с элементами нашего дерева, начиная с корня: 32>20, правая ссылка у корня = 0. Следовательно, справа от корня создаем вершину, записываем в неё 32, обнуляем левую и правую ссылки третьей вершины и создаем связь между этими вершинами, т.е. в массив right на первое место записать 3.
- Берем четвертый элемент и сравниваем его с элементами нашего дерева, начиная с корня: 17<20, но слева у 20 уже есть вершина №2 (в ней записано 15), поэтому нужно перейти на эту вершину и сравнить ее значение с взятым элементом. 17>15, правая ссылка у второй вершины = 0. Следовательно, справа от неё создаем вершину, записываем в неё 17, обнуляем левую и правую ссылки четвертой вершины и создаем связь между этими вершинами, т.е. в массив right на второе место записываем 4 и т.д.

Процесс закончится, когда мы переберем все элементы файла.

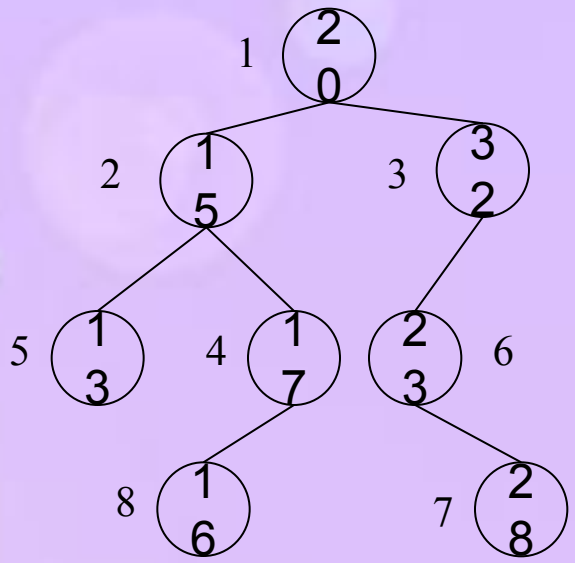
Назад Меню

В результате получится следующее двоичное дерево поиска (см. рис.), заданное с помощью массивов:

Key = [20 15 32 17 13 23 28 16];

Left = [2 5 6 8 0 0 0 0];

Right = [3 4 0 0 0 7 0 0].




Программа задания двоичного дерева поиска на языке Паскаль

Назад

Меню

# Обходы дерева

Для того чтобы просмотреть информационные поля всех вершин построенного дерева, найти нужный элемент в дереве, отсортировать элементы дерева и во многих других задачах необходимо совершить так называемый **обход дерева (посетить каждую его вершину, причем производить какие-то действия с каждой вершиной можно только 1 раз)**.

В случае, когда бинарное дерево пусто, оно обходится без выполнения каких-либо действий. Чтобы обойти непустое дерево существует несколько алгоритмов.

Рекурсивные алгоритмы прохождения каждого дерева по каждому из способов включают 3 одинаковых процедуры, где нужно найти корень поддерева, левое поддерево текущего корня и правое поддерево текущего корня. Направление обхода однозначно определяет последовательность выполнения указанных процедур.

Назад Меню

# Алгоритмы обхода дерева:

Прямой левый обход

Прямой правый обход

Обратный левый обход

Обратный правый обход

Внутренний левый обход

Внутренний правый обход

По уровням слева на право сверху вниз

По уровням справа налево сверху вниз

По уровням слева на право снизу вверх

По уровням справа налево снизу вверх

Назад

Меню

# Прямой левый обход дерева

**Алгоритм прямого левого обхода дерева:**

1. Посетить корень;
2. Обойти левое поддерево;
3. Обойти правое поддерево.

Процедура прохождения дерева прямым левым обходом на языке Паскаль:

```
procedure obxod_1(i:integer);
begin
if i<>0 then  begin  write(key[i]:4); obxod_1(left[i]);  obxod_1(right[i]); end;
end;
```

Список обходов Назад Меню

# Прямой правый обход дерева

**Алгоритм прямого правого обхода дерева:**

1. Посетить корень;
2. Обойти правое поддерево;
3. Обойти левое поддерево.

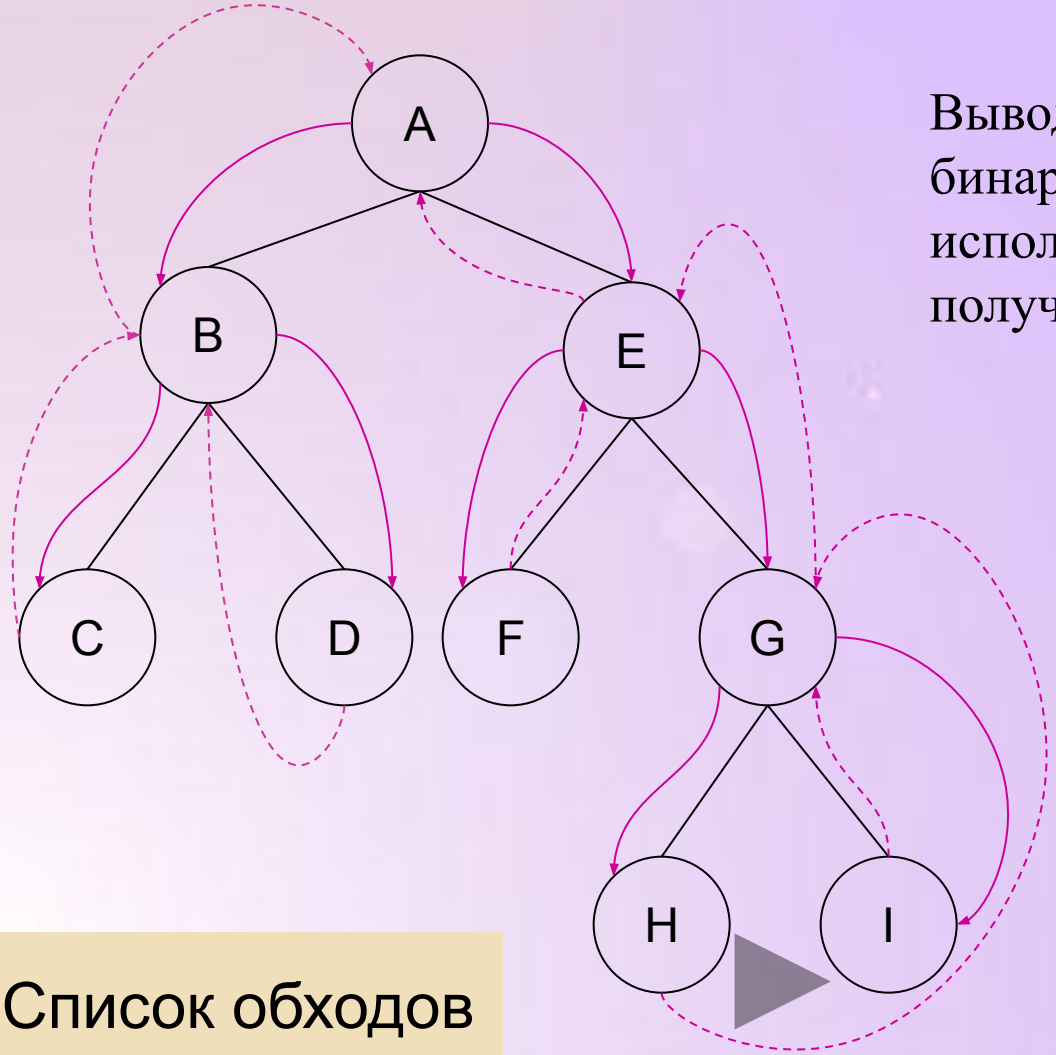
Процедура прохождения дерева прямым правым обходом на языке Паскаль:

```
procedure obxod_2(i:integer);
begin
if i<>0 then
      begin
            write(key[i]:4);
            obxod_2(right[i]);
            obxod_2(left[i]);
      end;
end;
```

Назад

Меню

# Пример




Выводя на экран значения вершин бинарного дерева (см.рис.), используя прямой правый обход, получим последовательность:

**A E G I H F B D C**

Список обходов

Назад

Меню

# Примеры использования прямых обходов:

1. решение задачи методом деления на части
2. стратегия "разделяй и властвуй" (Сортировка Фон Неймана, быстрая сортировка, одновременное нахождение максимума и минимума последовательности чисел, умножение длинных чисел).

# Обратный левый обход

**Алгоритм обратного левого обхода:**

1. Обойти левое поддерево;
2. Обойти правое поддерево;
3. Посетить корень.

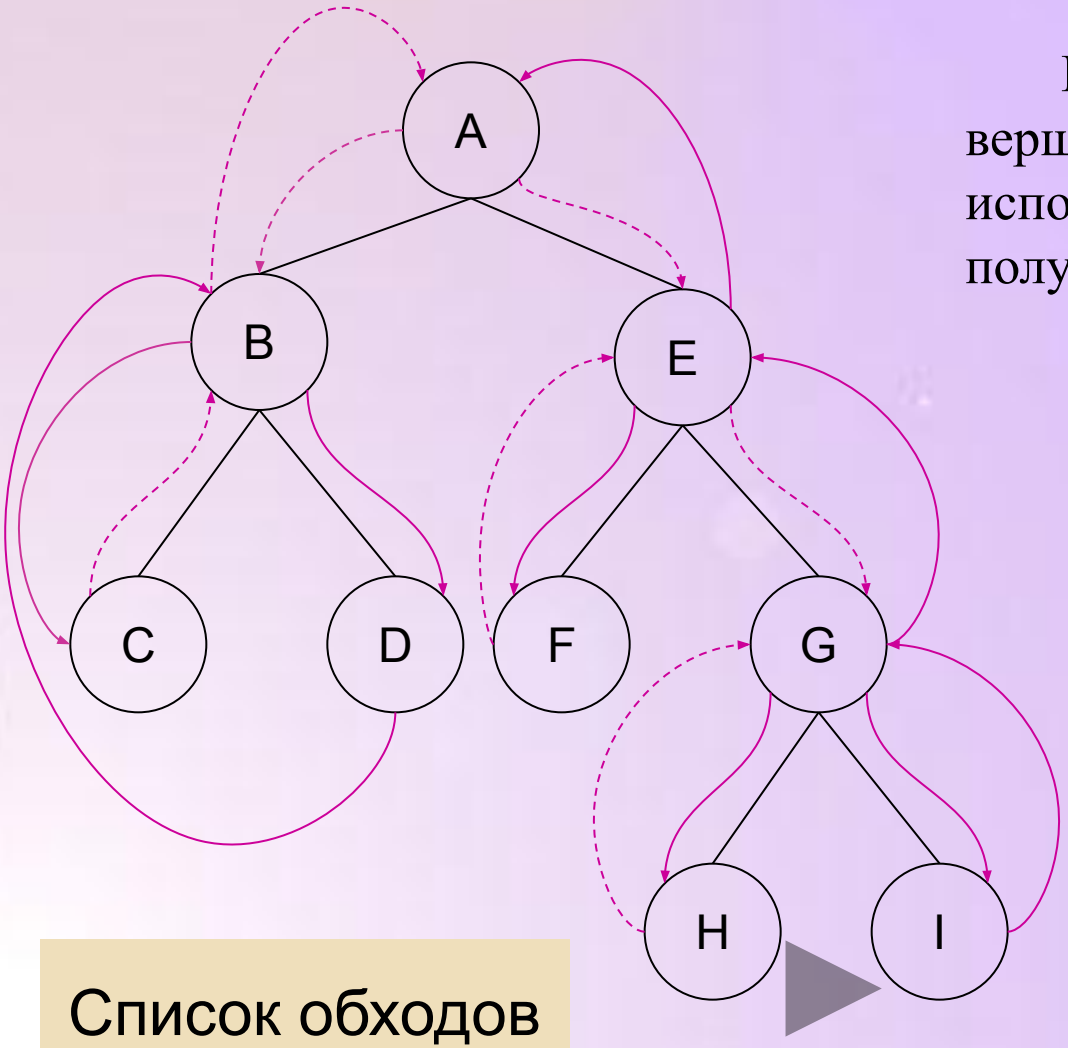
Процедура прохождения дерева обратным левым обходом на языке Паскаль:

```
procedure obxod_3(i:integer);
begin
if i<>0 then
    begin
        obxod_3(left[i]);
        obxod_3(right[i]);
        write(key[i]:4);
    end;
end;
```

Назад

Меню

# Пример




Выводя на экран значения вершин бинарного дерева (см.рис.), используя обратный левый обход, получим последовательность:

**C D B F H I G E A**

Список обходов

Назад

Меню

# Обратный правый обход

**Алгоритм обратного правого обхода:**

1. Обойти правое поддерево;
2. Обойти левое поддерево;
3. Посетить корень.

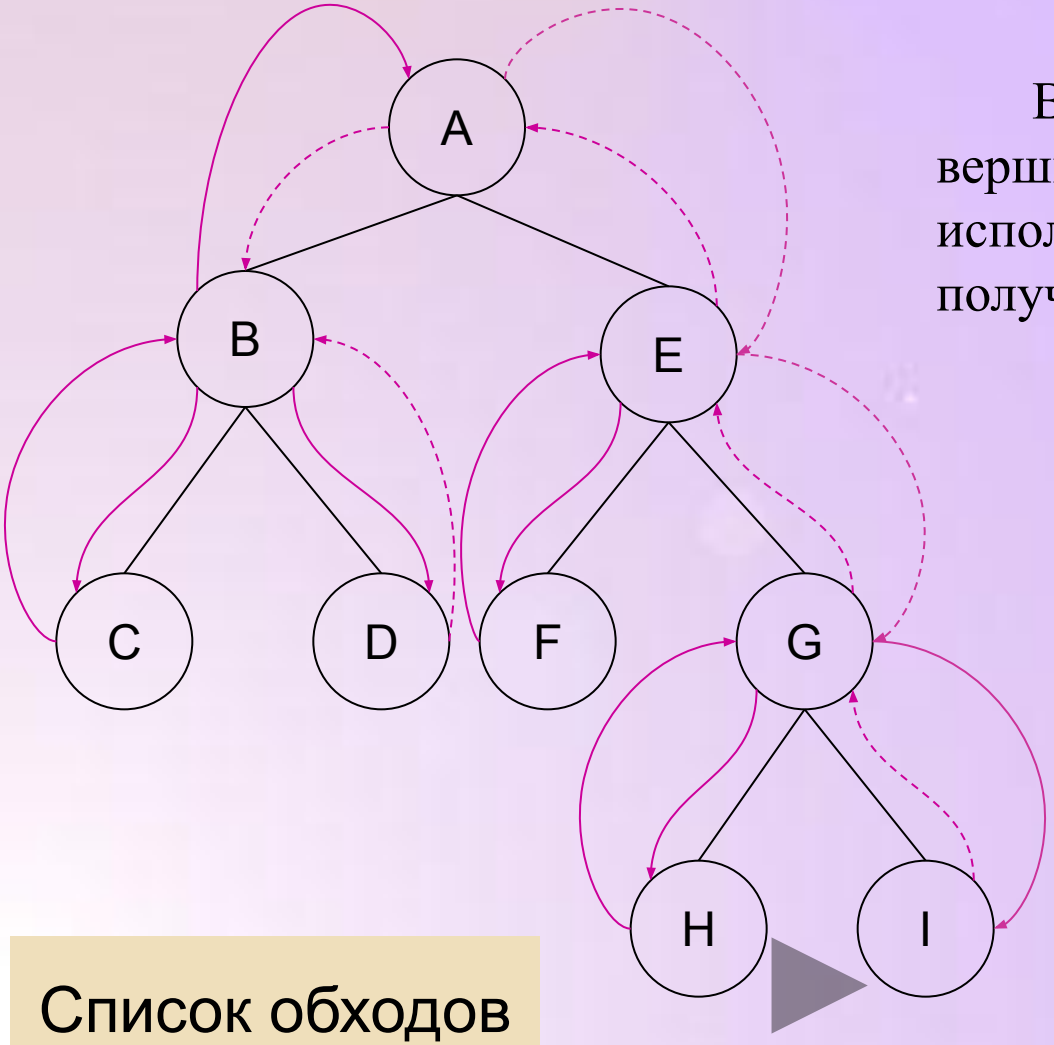
Процедура прохождения дерева обратным правым обходом на языке Паскаль:

```
procedure obxod_4(i:integer);
begin
if i<>0 then
    begin
        obxod_4(right[i]);
        obxod_4(left[i]);
        write(key[i]:4);
    end;
end;
```

Назад

Меню

# Пример




Выводя на экран значения вершин бинарного дерева (см. рис.), используя обратный правый обход, получим последовательность:

**I H G F E D C B A**

Список обходов

Назад

Меню

# Внутренний левый обход

**Алгоритм внутреннего левого обхода:**

1. Обойти левое поддерево;
2. Посетить корень;
3. Обойти правое поддерево.

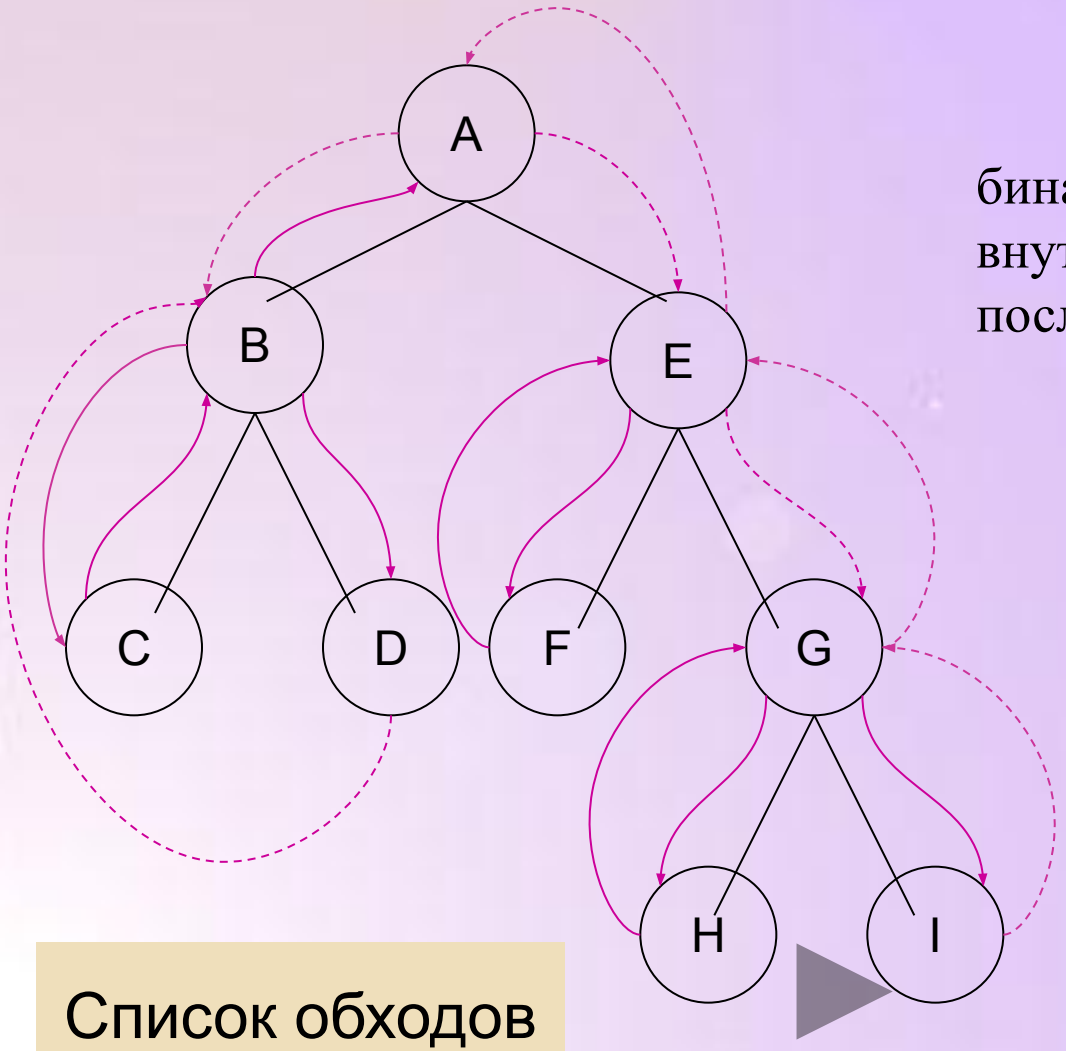
Процедура прохождения дерева внутренним левым обходом на языке Паскаль:

```
procedure obxod_5(i:integer);
begin
if i<>0 then
      begin
            obxod_5(left[i]);
            write(key[i]:4);
            obxod_5(right[i]);
      end;
end;
```

Назад

Меню

# Пример




Выводя на экран значения вершин бинарного дерева (см.рис.), используя внутренний левый обход, получим последовательность:

**C B D A F E H G I**

Список обходов

Назад

Меню

# Внутренний правый обход

**Алгоритм внутреннего правого обхода:**

1. Обойти правое поддерево;
2. Посетить корень;
3. Обойти левое поддерево.

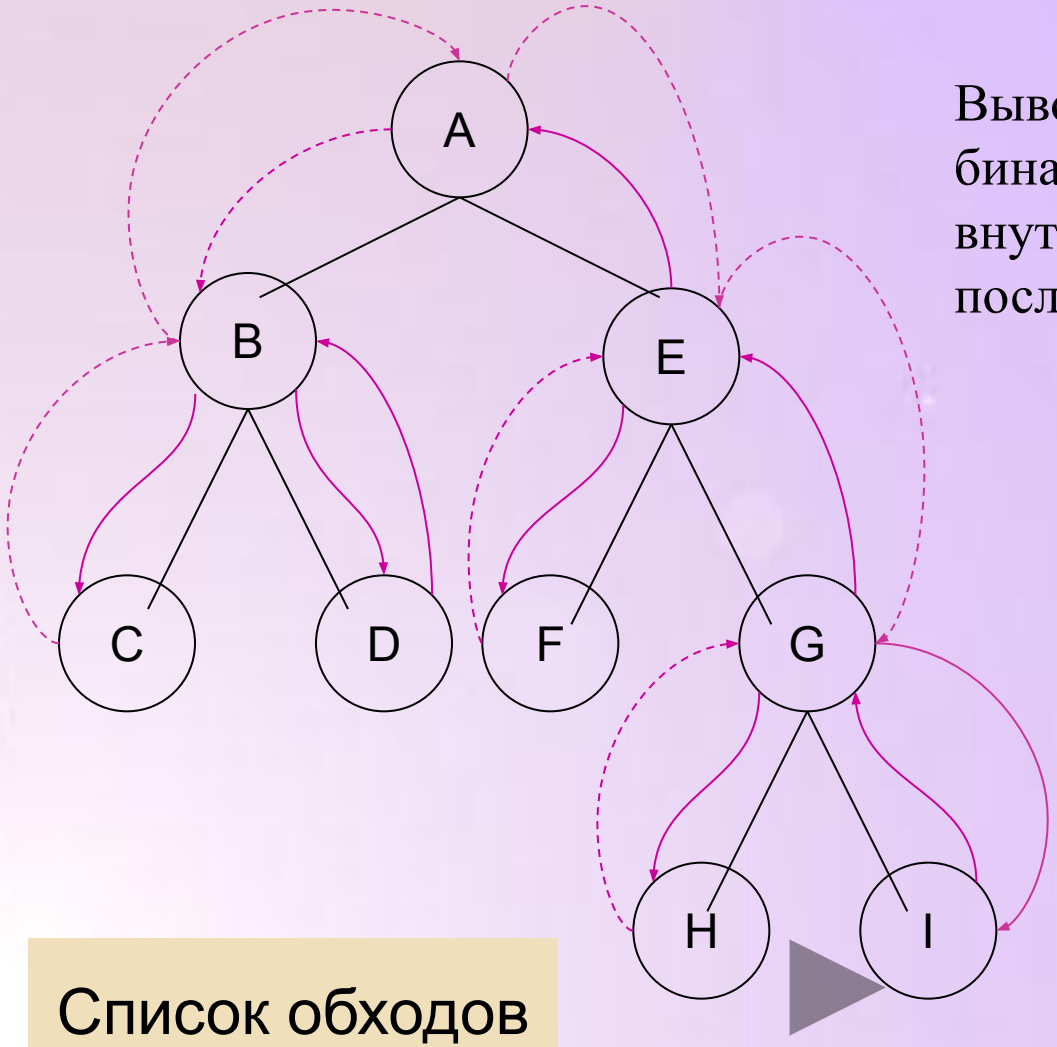
Процедура прохождения дерева внутренним правым обходом на языке Паскаль:

```
procedure obxod_6(i:integer);
begin
if i<>0 then
      begin
            obxod_6(right[i]);
            write(key[i]:4);
            obxod_6(left[i]);
      end;
end;
```

Назад

Меню

# Пример




Выводя на экран значения вершин бинарного дерева (см. рис.), используя внутренний правый обход, получим последовательность:

**I G H E F A D B C**

Список обходов

Назад

Меню

# Обходы дерева по уровням

Для того чтобы осуществить любой из четырех обходов дерева по уровням необходимо знать, сколько уровней в дереве (высоту дерева), которое нужно обойти. Если дерево неполное и содержит большое количество вершин, то считать количество уровней вручную очень сложно. Рационально будет использовать специальную процедуру, которая определяет высоту любого бинарного дерева.

Процедура вычисления высоты бинарного дерева на языке Паскаль:

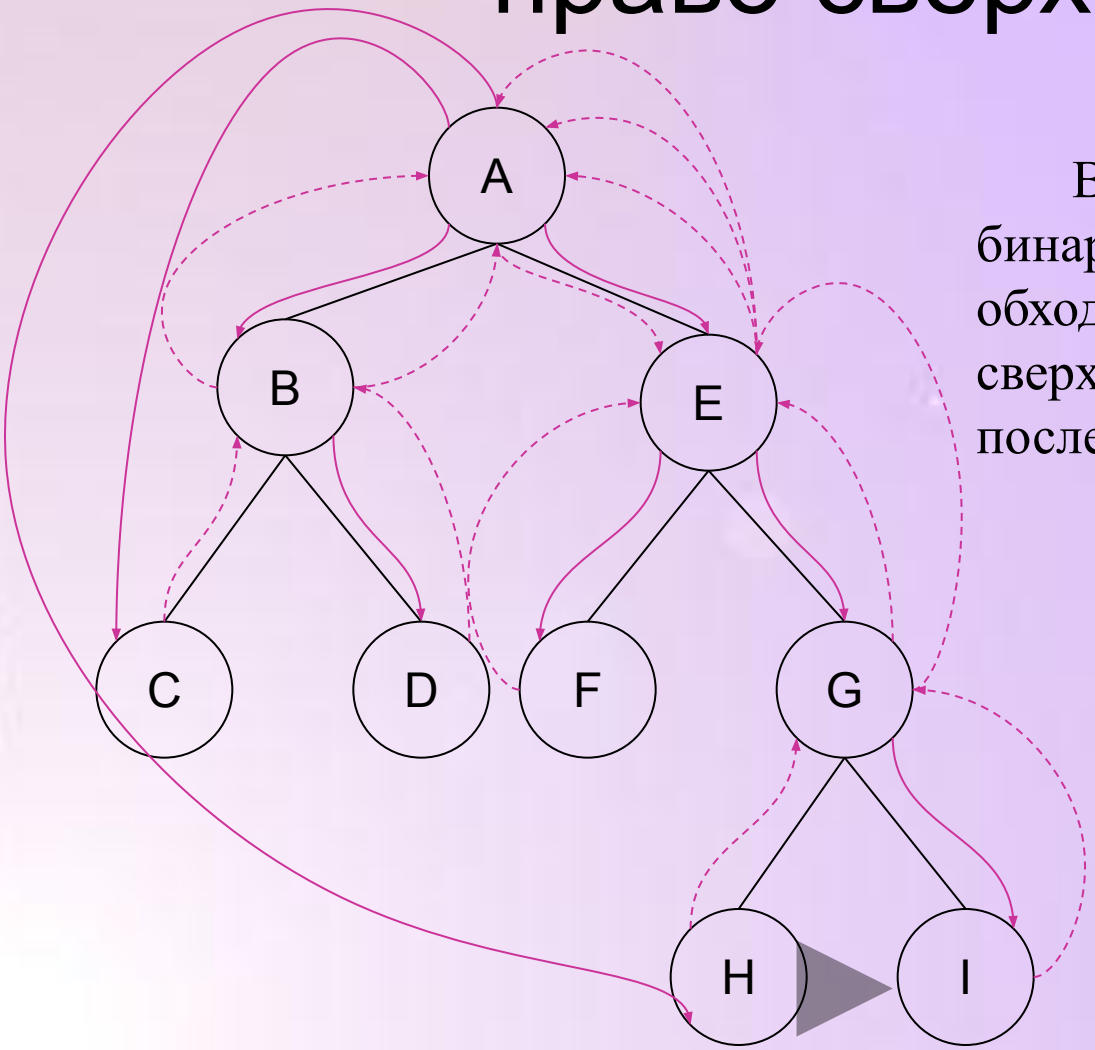
```
procedure H_tree(i,u:integer);
begin
if i<>0 then  begin  if h<u then h:=u; H_tree(left[i],u+1); H_tree(right[i],u+1); end;
end;
```

По окончанию работы данной процедуры в переменной h будет храниться высота дерева, которое нужно обойти.

Назад

Меню

# Обход по уровням, слева на право сверху вниз




Выводя на экран значения вершин бинарного дерева (см. рис.), используя обход по уровням, слева на право сверху вниз, получим последовательность:

**A B E C D F G H I**

Назад

Меню

Процедура обхода бинарного дерева по уровням, слева на право сверху вниз на языке Паскаль:

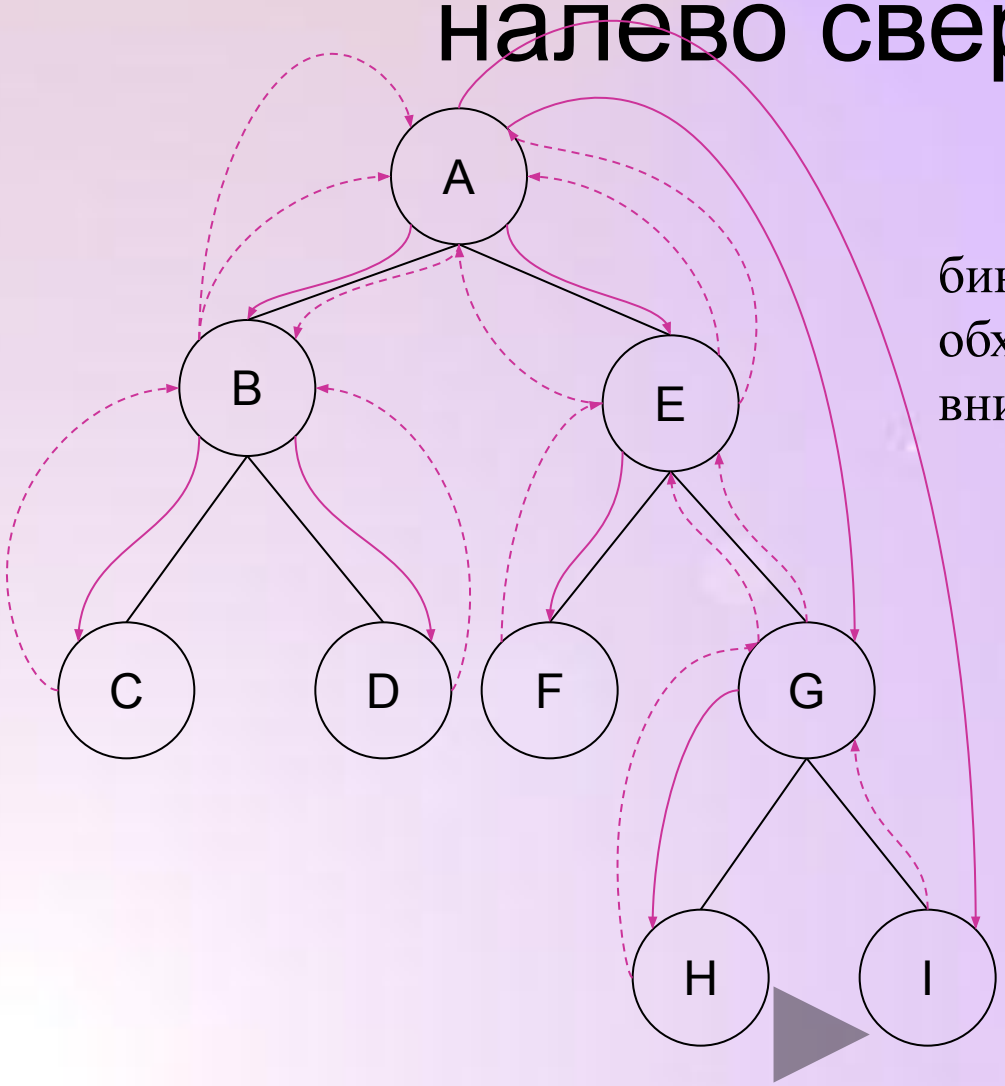
```
procedure obxod_7(i,u:integer);
begin
if i<>0 then
    begin
        if j = u then write(key[i]:4)
        Else begin  obxod_1(left[i],u+1); obxod_1(right[i],u+1); end;
    end;
end;
```

Вызов процедуры в основной программе:

```
for j:=1 to h do
    obxod_7(1,1);
```

Список обходов Назад Меню

# Обход по уровням, справа налево сверху вниз




Выводя на экран значения вершин бинарного дерева (см. рис.), используя обход по уровням, справа налево сверху вниз, получим последовательность:

**A E B G F D C I H**

Назад

Меню

Процедура обхода бинарного дерева по уровням, справа налево сверху вниз на языке Паскаль:

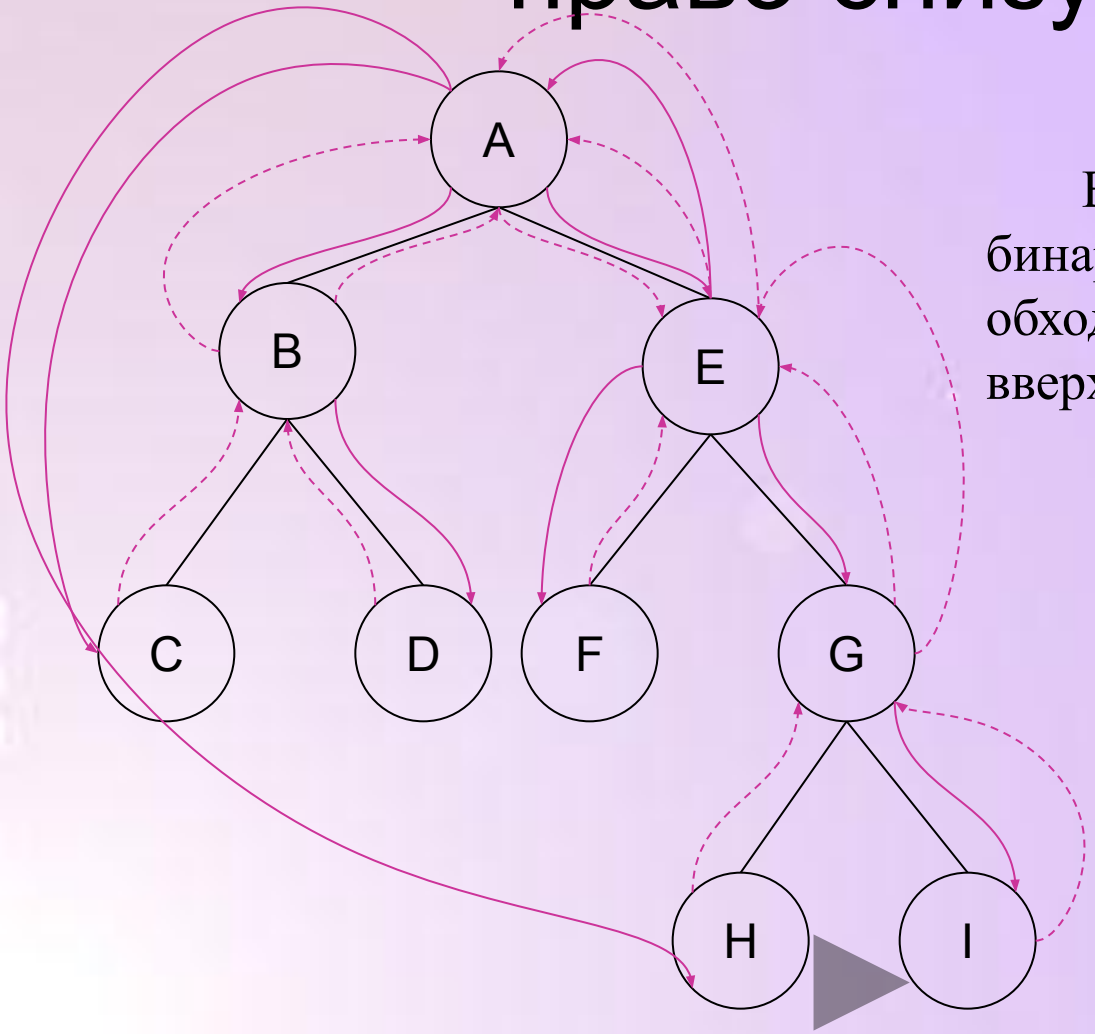
```
procedure obxod_8(i,u:integer);
begin
    if i<>0 then
    begin
        if j = u then write(key[i]:4)
        else begin  obxod_1(right[i],u+1);  obxod_1(left[i],u+1); end;
    end;
end;
```

Вызов процедуры в основной программе:

```
for j:=1 to h do
    obxod_8(1,1);
```

Список обходов Назад Меню

# Обход по уровням, слева на право снизу вверх




Выводя на экран значения вершин бинарного дерева (см. рис.), используя обход по уровням, слева на право снизу вверх, получим последовательность:

**H I C D F G B E A**

Назад

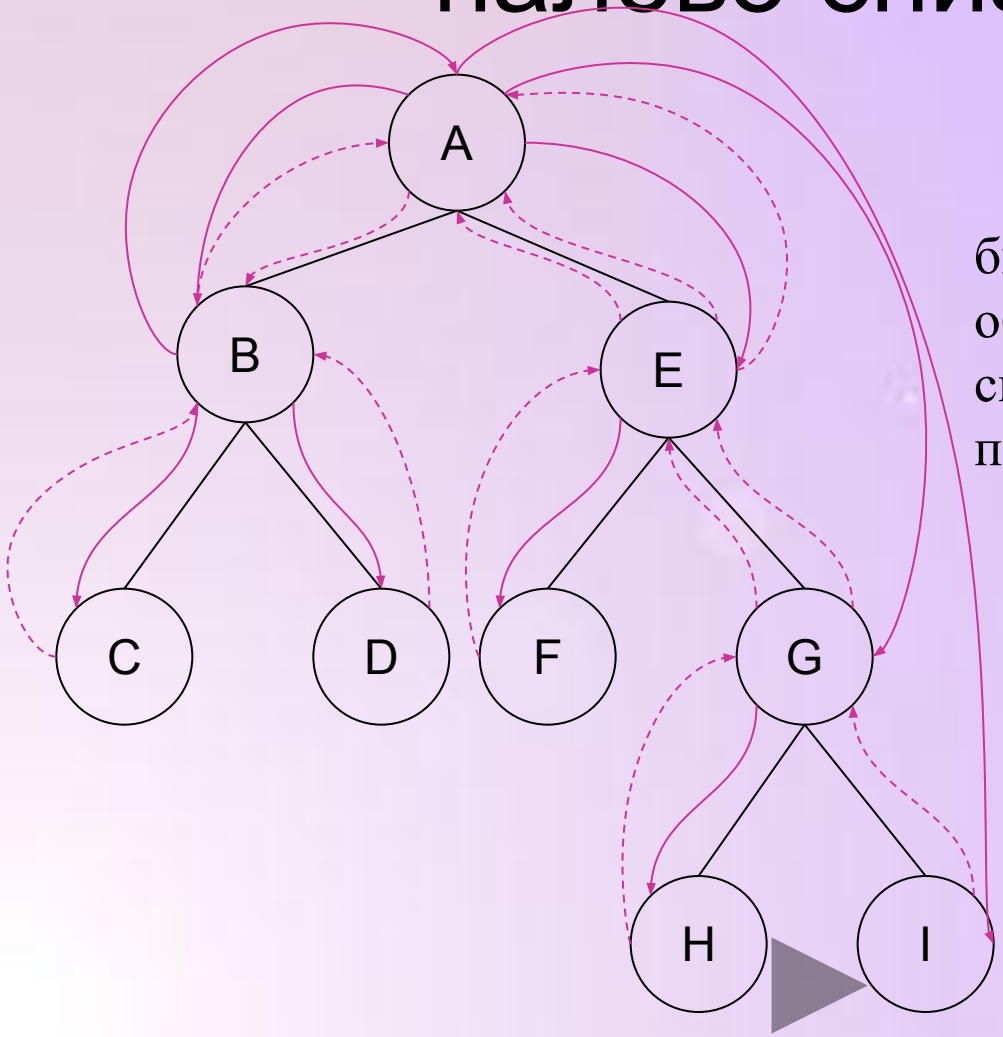
Меню

Процедура обхода бинарного дерева по уровням, слева на право снизу вверх абсолютно идентична процедура обхода бинарного дерева по уровням, слева на право сверху вниз, изменится только вызов данной процедуры в основной программе:

```
for j:= h downto 1 do  obxod_7(1,1);
```

Список обходов Назад Меню

# Обход по уровням, справа налево снизу вверх




Выводя на экран значения вершин бинарного дерева (см. рис.), используя обход по уровням, справа налево снизу вверх, получим последовательность:

**I H G F D C E B A**

Назад

Меню

Процедура обхода бинарного дерева по уровням, справа налево снизу вверх абсолютно идентична процедура обхода бинарного дерева по уровням, справа налево сверху вниз, изменится только вызов данной процедуры в основной программе:

```
for j:= h downto 1 do  obxod_8(1,1);
```

Список обходов Назад Меню