Лекция 8.
Бинарные деревья

Дерево

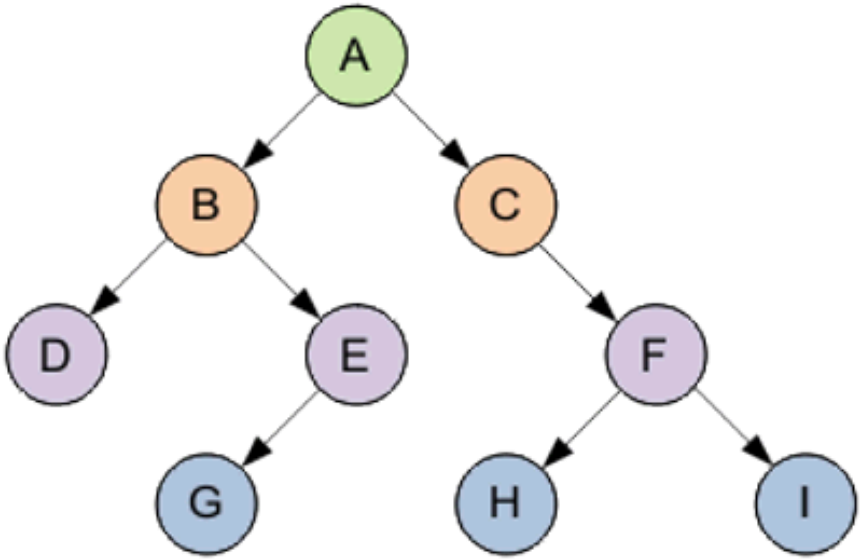
Дерево – структура данных, представляющая собой древовидную структуру в виде набора связанных узлов.
Бинарное дерево — это конечное множество элементов, которое либо пусто, либо содержит элемент (корень), связанный с двумя различными бинарными деревьями, называемыми левым и правым поддеревьями. Каждый элемент бинарного дерева называется узлом. Связи между узлами дерева называются его ветвями.

Способ представления бинарного дерева:

- A - корень дерева
- B - корень левого поддерева
- C - корень правого поддерева

Корень дерева расположен на уровне с минимальным значением.

Узел D, который находится непосредственно под узлом B, называется потомком B.



Если D находится на уровне i, то B – на уровне i-1.
Узел B называется предком D.

Максимальный уровень какого-либо элемента дерева называется его глубиной или высотой.

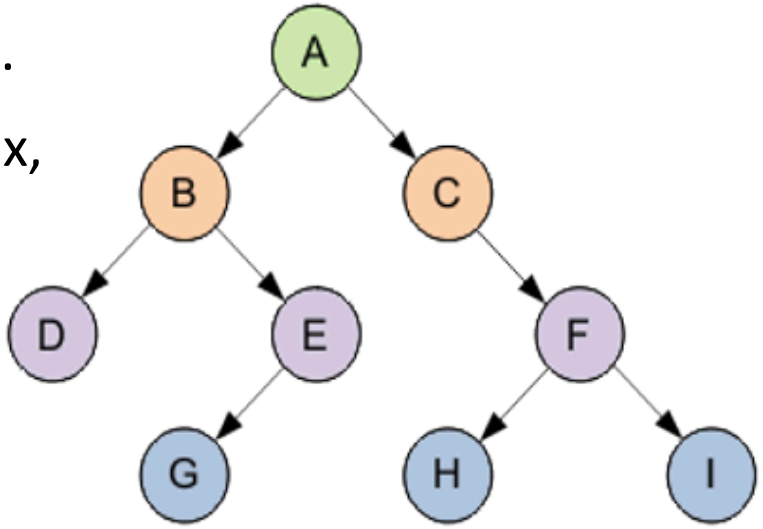
Если элемент не имеет потомков, он называется листом или терминальным узлом дерева. Остальные элементы – внутренние узлы (узлы ветвления).

Число потомков внутреннего узла называется его степенью. Максимальная степень всех узлов есть степень дерева.

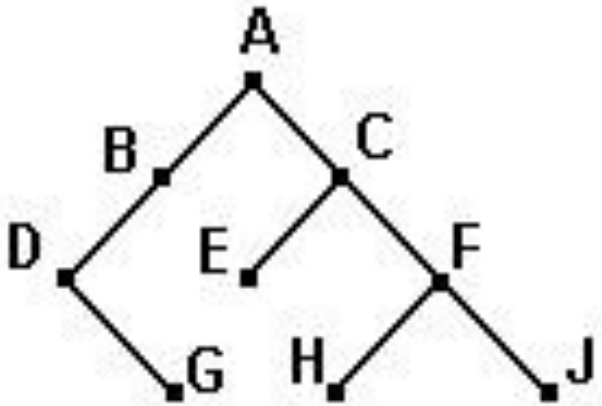
Число ветвей, которое нужно пройти от корня к узлу **x**, называется длиной пути к **x**.

Корень имеет длину пути равную 0;
узел на уровне i имеет длину пути равную i.

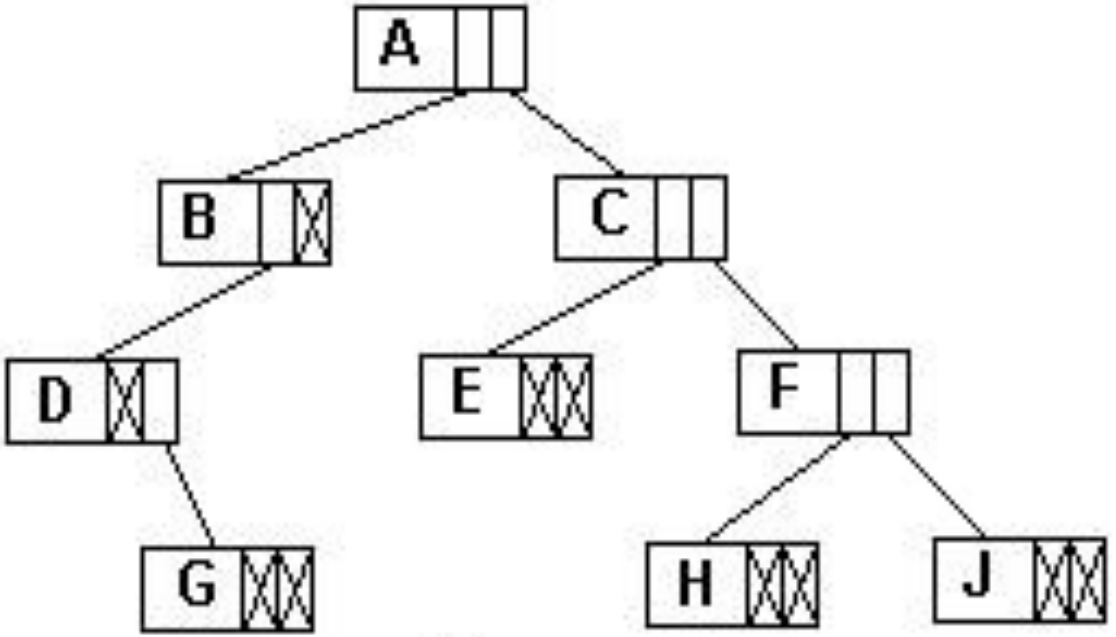
Бинарное дерево применяется в тех случаях, когда в каждой точке вычислительного процесса должно быть принято одно из двух возможных решений.



Бинарное дерево и структура в динамической памяти



а



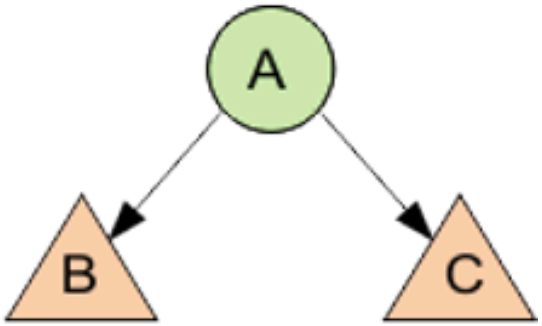
б

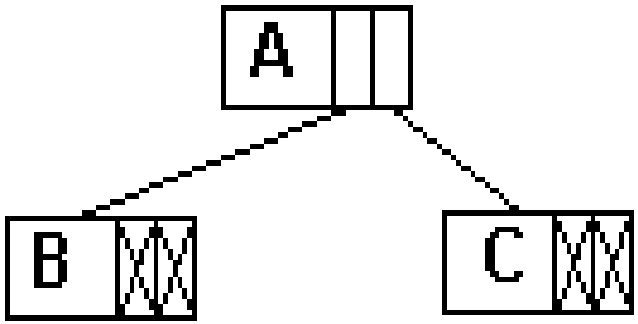
Способы обхода дерева

Пусть имеем дерево, где A — корень, B и C — левое и правое поддеревья.

Способы обхода дерева:

- обход дерева в прямом порядке: A, B, C - префиксная форма;
- обход дерева в обратном порядке (слева направо): B, A, C - инфиксная форма;
- обход дерева в концевом порядке (снизу вверх): B, C, A - постфиксная форма.

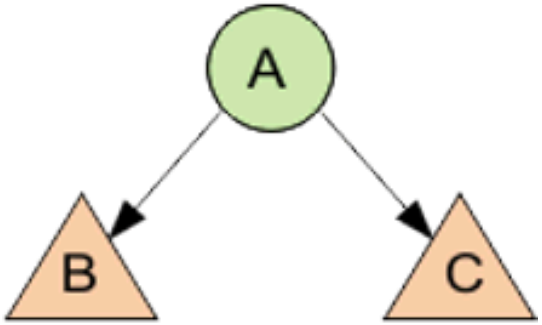


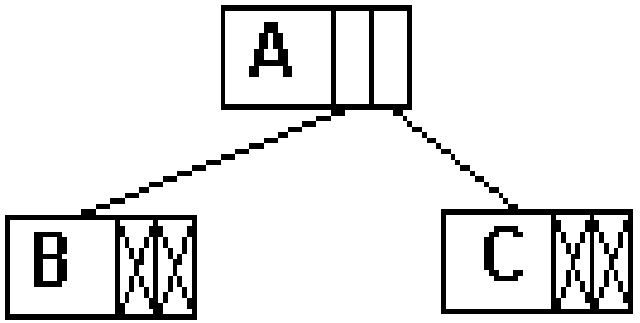


Прямой порядок

1. Попасть в корень.
2. Пройти левое поддерево.
3. Пройти правое поддерево

A B C

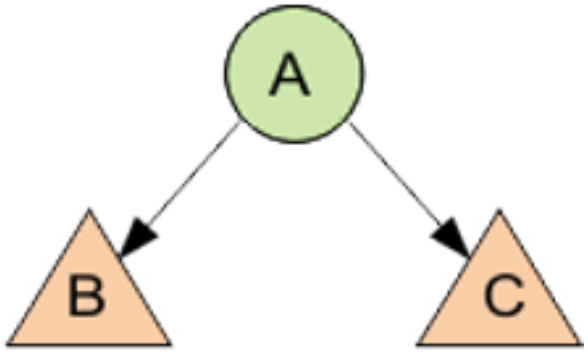


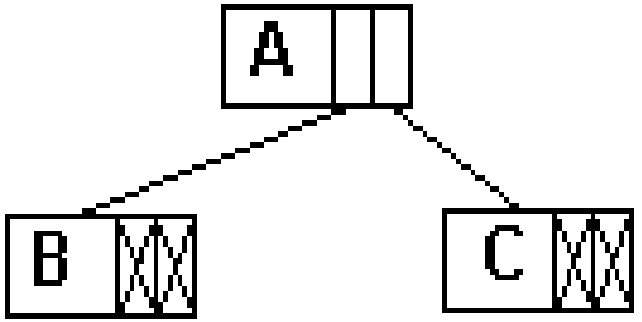


Обратный порядок

1. Пройти левое поддерево.
2. Попасть в корень.
3. Пройти правое поддерево.

B A C

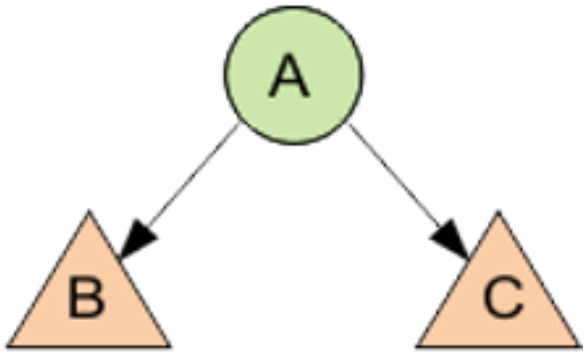


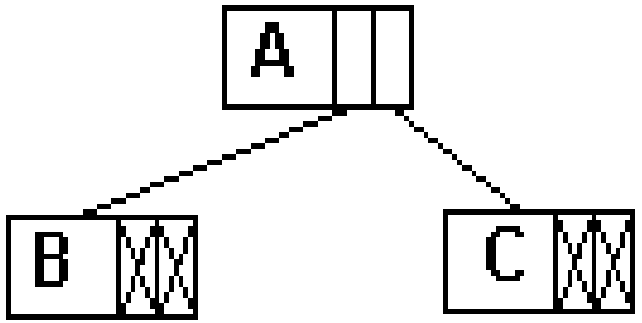


Концевой порядок

1. Пройти левое поддерево.
2. Пройти правое поддерево.
3. Попасть в корень.

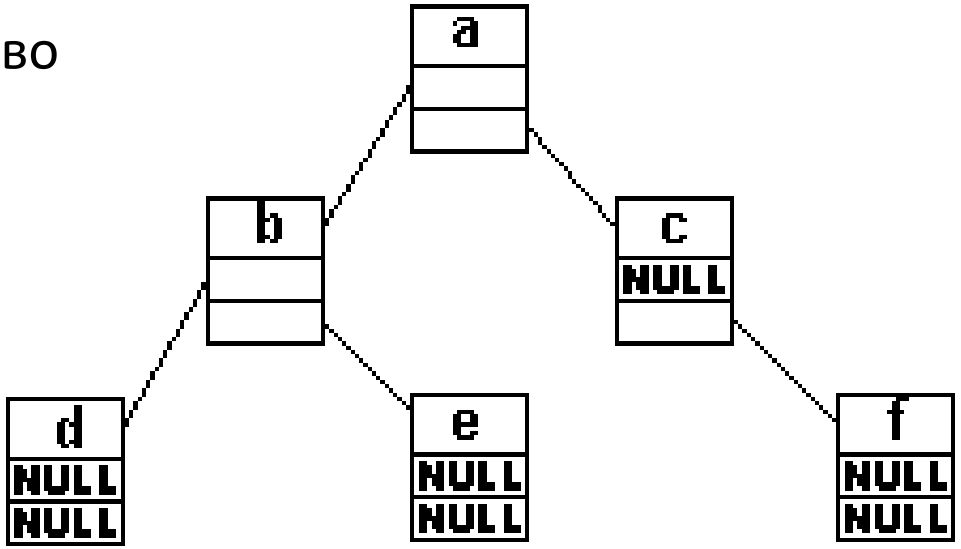
B C A





Обход дерева в прямом порядке

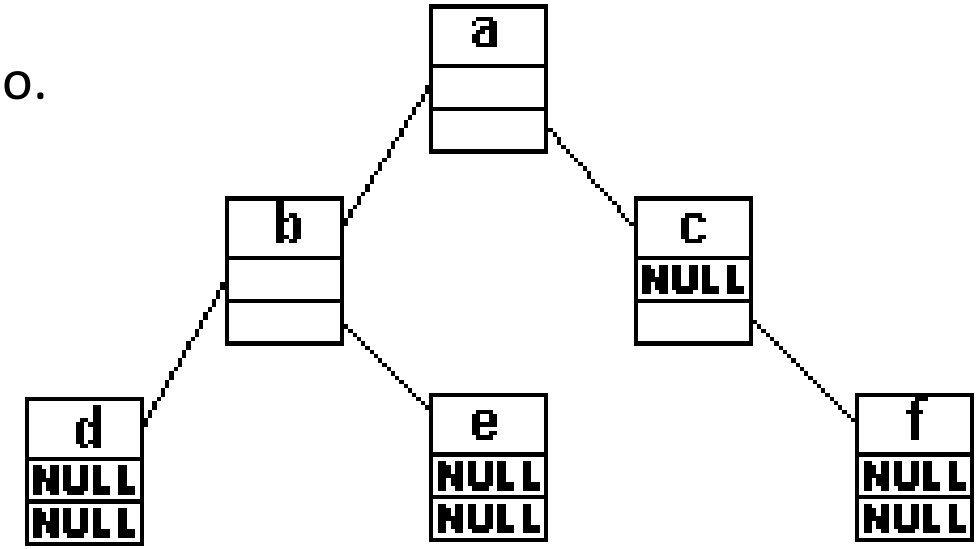
1. Попасть в корень.
2. Пройти левое поддерево.
3. Пройти правое поддерево



a b d e c f

Обход дерева в обратном порядке

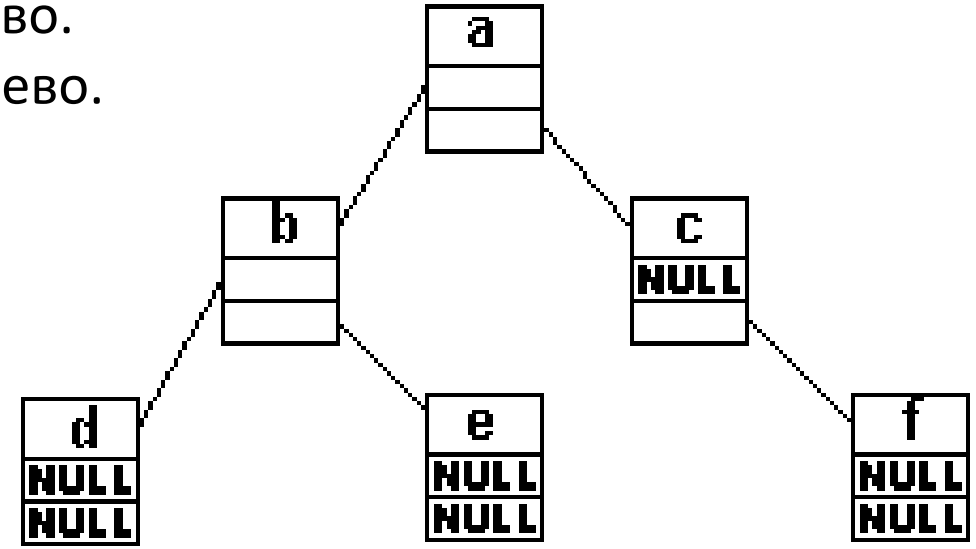
1. Пройти левое поддерево.
2. Попасть в корень.
3. Пройти правое поддерево.



d b e a c f

Обход дерева в концевом порядке

1. Пройти левое поддерево.
2. Пройти правое поддерево.
3. Попасть в корень.



d e b f c a

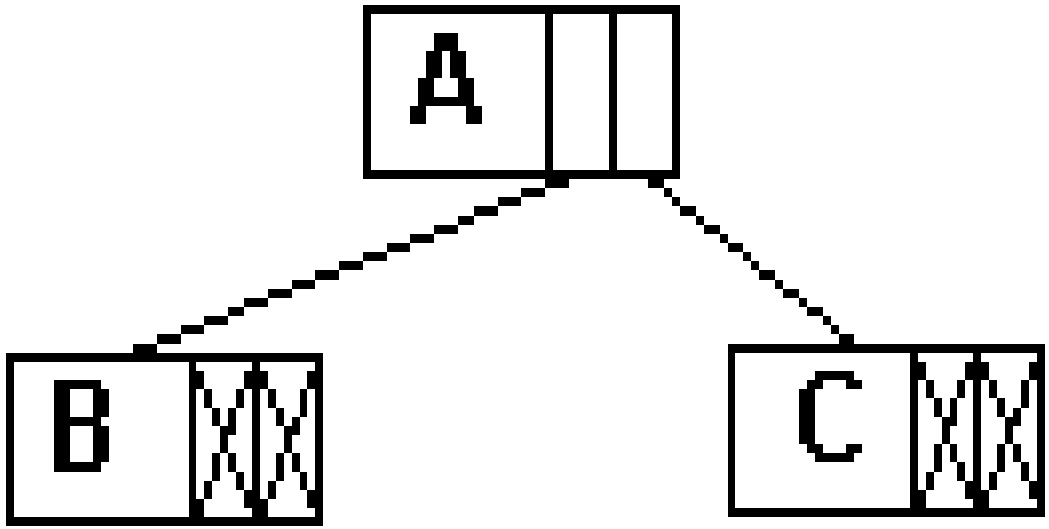
Описание типа

```
struct node
        {
                char info;
                struct node *llink;
                struct node *rlink;
        };
```

Функция main()

```
1.  int main()
2.  {
3.      struct node *root;
4.      vtree(&root);
5.      btree(root);
6.      btree2();
7.      return 0;
8.  }
```

Как вводить дерево



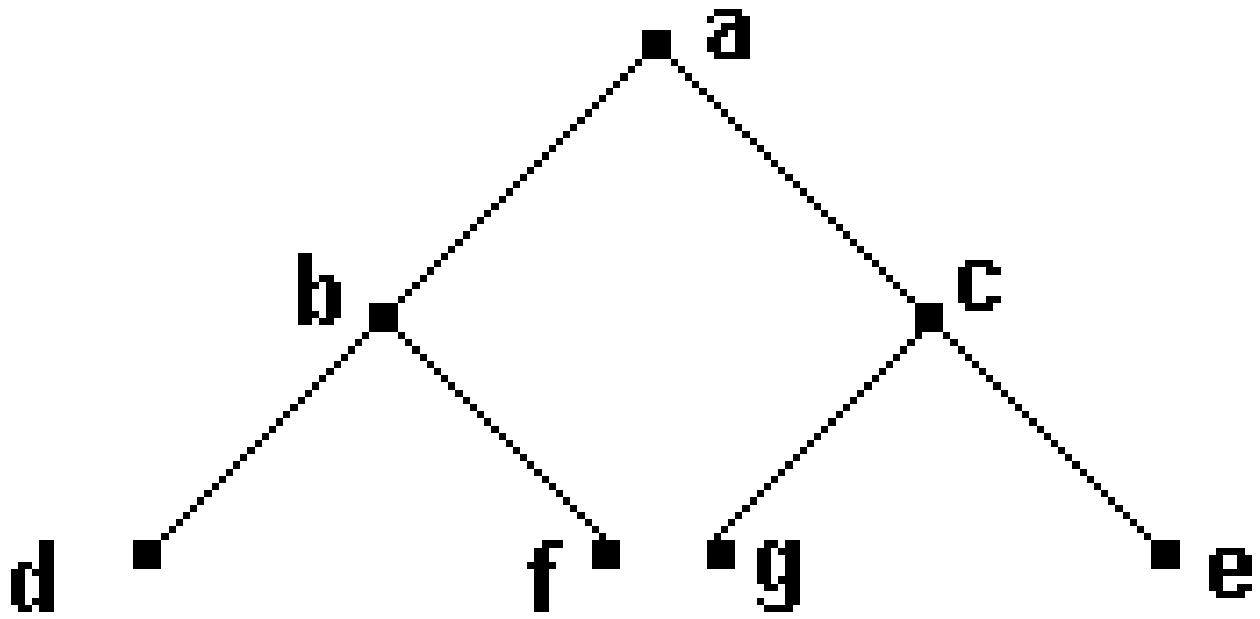
A B . . C . .

Рекурсивная функция построения бинарного дерев

- Параметр функции – указатель на указатель на корень дерева
- Вершины дерева – символьные данные

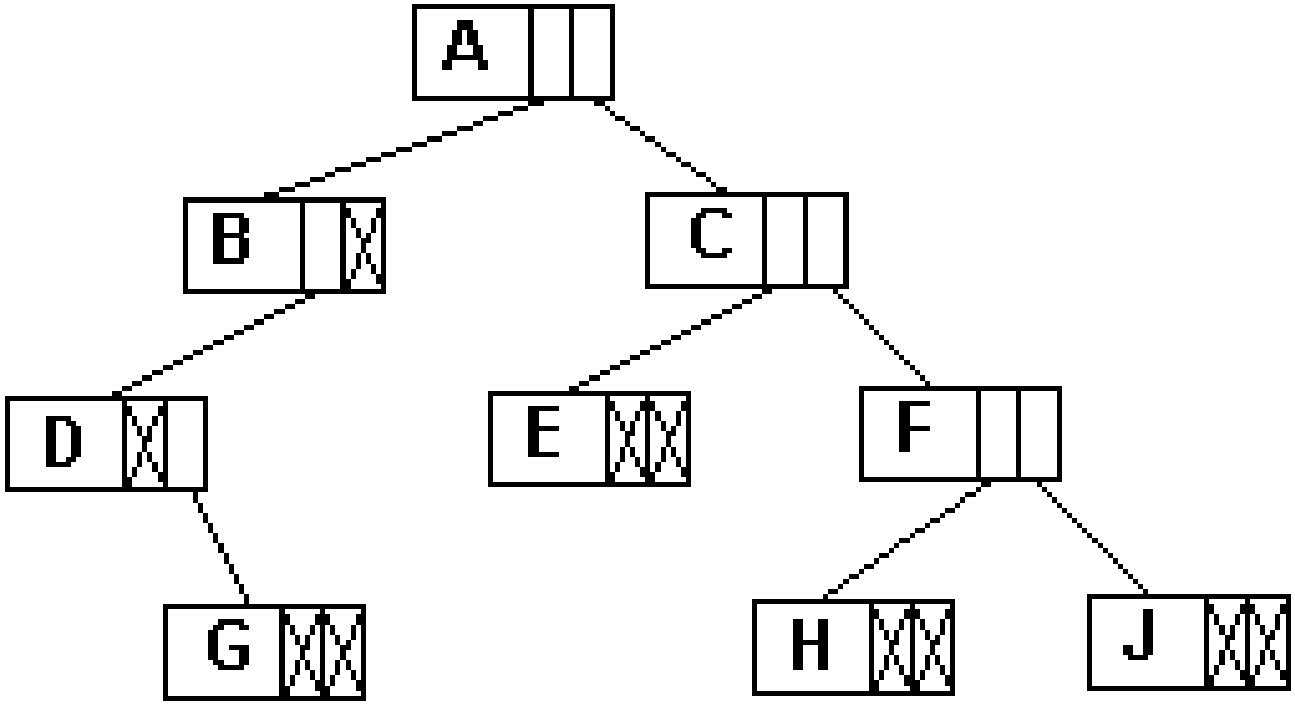
```
1.  void vtree(struct node **t)
2.  {
3.      char c;
4.      scanf("%c", &c);
5.      if (c != '.')
6.      {
7.          *t=(struct node *)malloc(sizeof(struct node ));
8.          (*t)->info=c;
9.          vtree(&((*t)->llink));
10.         vtree(&((*t)->rlink));
11.     }
12.     else
13.         *t=NULL;
14. }
```

Как вводить дерево



a b d .. f . . c g . . e . .

Как вводить дерево



A B D . G . . . C E . . F H . . J . .

Функция рекурсивного обхода дерева

```
1. void btree(struct node *t)
2.  {
3.     if (t != NULL)
4.     {
5.        btree(t->llink); // обойти левое поддерево
6.        printf("%c ", t->info); // попасть в корень
7.        btree(t->rlink); // обойти правое поддерево
8.      }
9.  }
```

Алгоритм стекового обхода дерева

ROOT – указатель на бинарное дерево;

A – стек, в который заносятся адреса еще не пройденных вершин;

TOP – вершина стека;

P – рабочая переменная-указатель.

Обход дерева в прямом порядке.

Содержимое стека A:

[a]

[a,b]

[a,b,c]

[a,b,f]

[a,b,f,g]

[a,b,f,g,m]

[a,b,f,n]

[a,b,h]

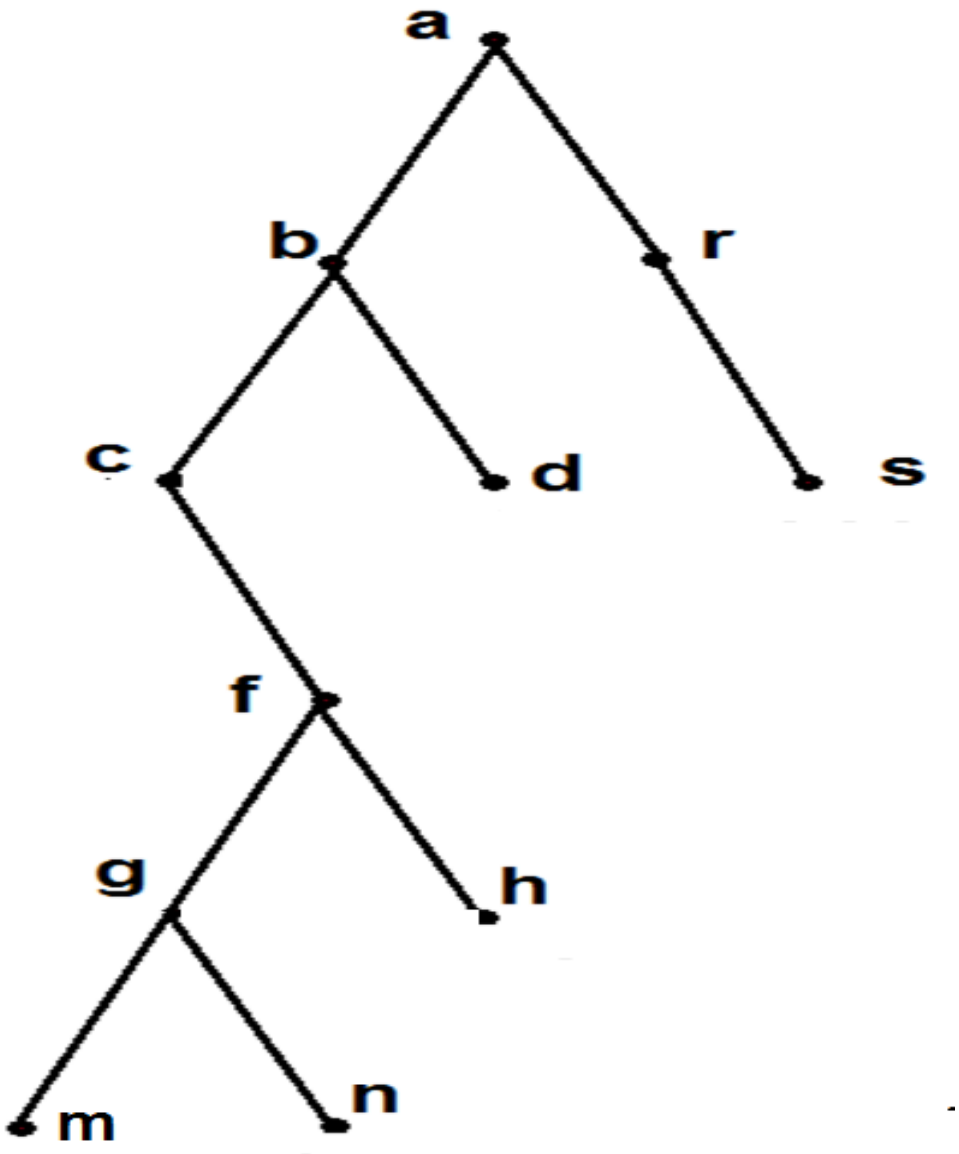
[a,d]

[r]

[s]

Вершина стека:

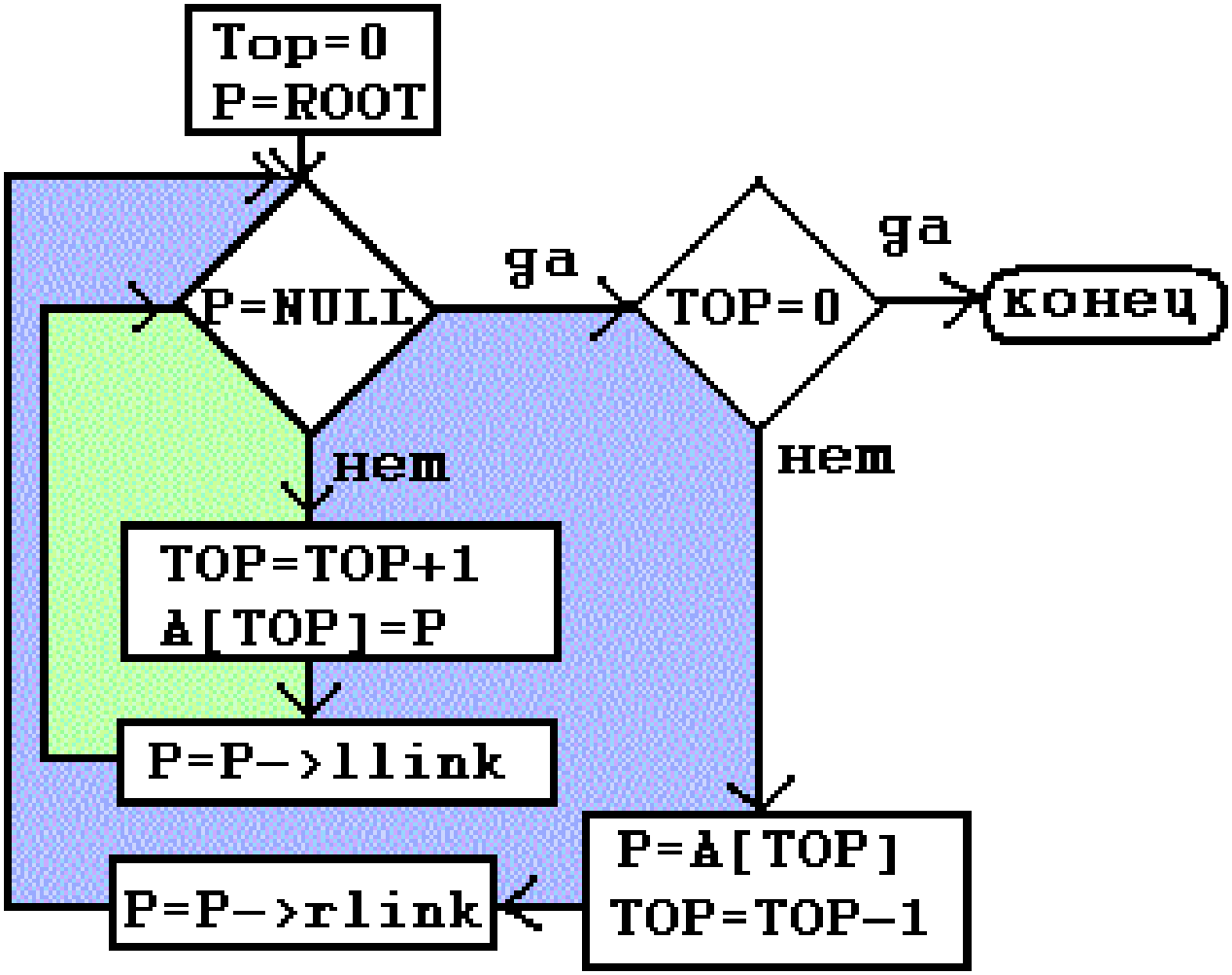
a b c f g m n h d r s



Алгоритм стекового обхода дерева

1. Начальная установка:
   TOP = 0; P = ROOT;
2. Если P = NULL, то перейти на 4 (конец ветви).
3. Вершину заносим в стек:
   TOP = TOP+1; A[TOP] = P;
   шаг по левой ветви: P = P->llink;
   перейти на 2 (идем по левой ветви).
4. Если TOP = 0, то КОНЕЦ.
5. Достаем вершину из стека:
   P = A[TOP]; TOP = TOP-1;
   Шаг по правой связи: P = P->rlink;
   перейти на 2.

Блок-схема алгоритма обхода бинарного дерева



Бинарное (двоичное) дерево поиска – это бинарное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

- оба поддерева – левое и правое, являются двоичными деревьями поиска;
- у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, чем значение ключа данных самого узла X;
- у всех узлов правого поддерева произвольного узла X значения ключей данных не меньше, чем значение ключа данных узла X.

Данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных.

Для сортировки с помощью дерева исходная сортируемая последовательность представляется в виде структуры данных «дерево».

Например, исходная последовательность имеет вид:

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

Корнем дерева будет начальный элемент последовательности. Далее все элементы, меньшие корневого, располагаются в левом поддереве, все элементы, большие корневого, располагаются в правом поддереве. Это правило должно соблюдаться на каждом уровне. После того, как все элементы размещены в структуре «дерево», необходимо вывести их, используя инфиксную форму обхода.