Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

В.Г. Пак

Структуры и алгоритмы компьютерной обработки данных

Слайды видеолекций
для студентов заочного бакалавриата направления подготовки высшего образования «Прикладная информатика»

Санкт-Петербургский политехнический университет Петра Великого
2022

Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

ЛЕКЦИЯ №3

Нелинейные структуры данных. Деревья

Санкт-Петербургский политехнический университет Петра Великого
2022

Содержание

III. Нелинейные структуры данных

§2. Деревья

2.1. Основные определения

2.2. АТД деревьев

2.3. Реализации деревьев

2.4. Обходы деревьев

2.4.1. Префиксный обход

2.4.2. Постфиксный обход

2.4.3. Инфиксный обход

2.4.4. Обход в ширину

III. Нелинейные структуры данных

§2. Деревья

2.1. Основные определения

Дерево — одна из наиболее широко распространённых структур данных, эмулирующая древовидную структуру в виде набора связанных узлов. Математически дерево как структура данных описывается связанным графом, не содержащим циклы, т.е. деревом-графом.

Дерево — совокупность элементов (*узлов*, *вершин*) и «родительских» отношений, образующих иерархическую структуру узлов.

2.1. Деревья. Основные определения

Определение. *Деревом* называется абстрактная структура, построенная по следующим правилам:

1. Один узел является деревом. Этот узел – *корень* этого дерева.
2. Пусть n – узел, а $T_1, \dots, T_k$ - деревья с корнями $n_1, \dots, n_k$. Тогда новое дерево получается объявлением n *родителем* узлов $n_1, \dots, n_k$. В этом дереве n будет корнем, $T_1, \dots, T_k$ - *поддеревьями* этого корня. Узлы $n_1, \dots, n_k$ называются *сыновьями* узла n.

Нулевое дерево Λ– дерево без узлов.

2.1. Деревья. Основные определения

Определение. *Путём из узла* n_1 *в узел* n_k называется последовательность узлов $n_1, \ldots, n_k$ $(k > 1)$, в которой для всех $1 \leq i < k$ узел n_i является родителем узла n_{i+1}. Число $k - 1$ называется *длиной* пути. При $k = 1$ узел n_1 считается путём длины 0 из n_1 в него самого.

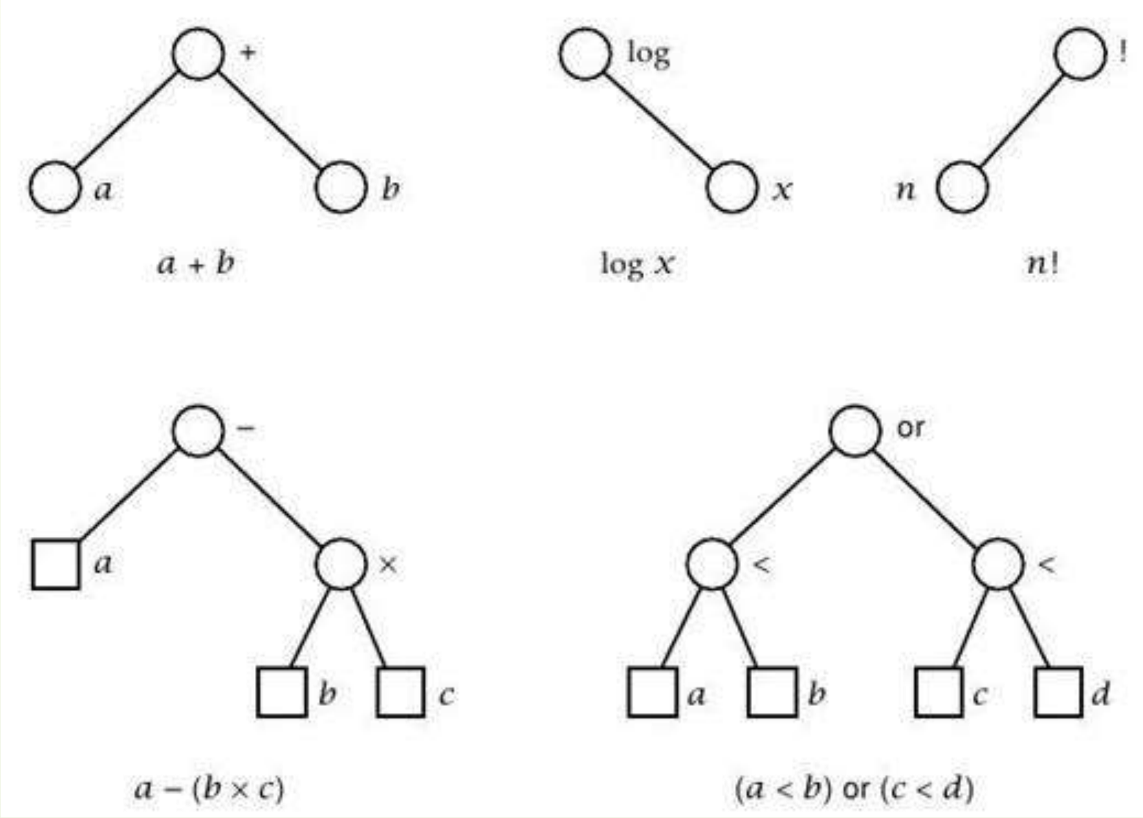
Определение. Если существует путь из a в b, то a называется *предком* b, а b – *потомком* a. *Предок (потомок) узла*, не совпадающий с самим узлом, называется его *истинным предком* (*потомком*).

Определение. Узел, не имеющий истинных потомков, называется *листом*.

Определение. *Высотой узла* называется длина самого длинного пути от этого узла до какого-либо листа. *Высотой дерева* называется высота корня.

Определение. *Глубиной узла* называется длина пути от корня до этого узла.

2.1. Деревья. Основные определения



Примеры деревьев выражений

2.2. АТД деревьев

Одно из наиболее важных применений деревьев – использование при разработке реализаций различных АТД.

Примеры: деревья двоичного поиска при реализации АТД множеств, частично упорядоченные деревья.

Некоторые операторы АТД деревьев:

1. PARENT(n, T). Функция возвращает родителя узла n в дереве T, если n является корнем, то возвращается Λ (выход за пределы дерева).
2. LEFTCHILD(n, T). Функция возвращает самого левого сына узла n в дереве T, если n является листом, то возвращается Λ. Сыновья каждого узла пронумерованы слева направо, таким образом, самый левый сын – первый в списке сыновей.

2.2. АТД деревьев

3. RIGHTBROTHER(n, T). Функция возвращает правого брата узла n в дереве T, если таковой существует, если нет, то возвращается Λ. При нумерации братьев слева направо функция находит следующего брата.
4. LABEL(n, T). Функция возвращает метку узла n в дереве T.
5. CREATE(v, $T_1, \dots, T_i$). Функция создаёт новое дерево с корнем, получающим метку v, с сыновьями, которые становятся корнями поддеревьев $T_1, \dots, T_i$. Если $i = 0$, т.е. список поддеревьев пуст, то создаётся дерево с одним узлом с меткой v, являющимся и корнем, и листом.
6. ROOT(T). Возвращает корень дерева T, если $T = \Lambda$, то возвращается Λ.
7. MAKENULL(T). Делает дерево T пустым.

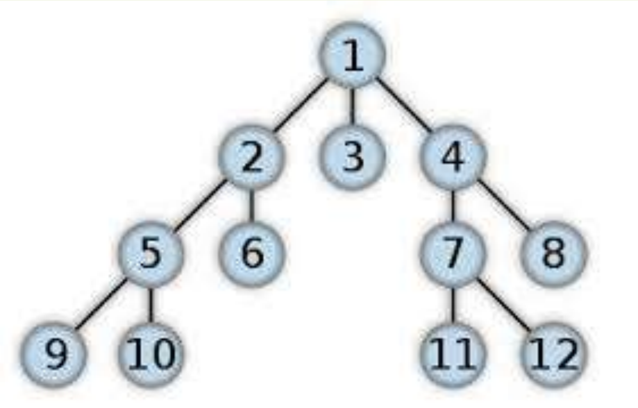
2.3. Реализации деревьев

Реализация деревьев с помощью массивов.

Самая простая реализация, поддерживающая оператор PARENT.

В линейном массиве A каждый элемент A[i] есть указатель или курсор на родителя узла i. Корень имеет нулевой указатель или указатель на самого себя.

Дерево



Курсоры на родителей

A	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	1	2	2	4	4	5	5	7	7

2.3. Реализации деревьев

Что можно сделать легко и за фиксированное время:

- найти родителя любого узла (оператор PARENT);
- пройти по любому пути от родителя к родителю, найти глубину любого узла;
- присвоить метки узлам и определить метки узлов (оператор LABEL).

Что сделать трудно:

- реализовать операторы, требующие информацию о сыновьях (операторы LEFTCHILD, RIGHTBROTHER);
- найти высоту узла.

2.3. Реализации деревьев

Если использовать нумерацию сыновей слева направо, то оператор RIGHTBROTHER можно реализовать так:

```
Procedure RIGHTBROTHER(n, T)
Begin
    parent:=T[n]
    for i:=n+1 to maxnodes – максимальный номер узлов
       do
         if T[i]=parent then
                return(i)
         return(0)  правый брат не найден
End RIGHTBROTHER
```

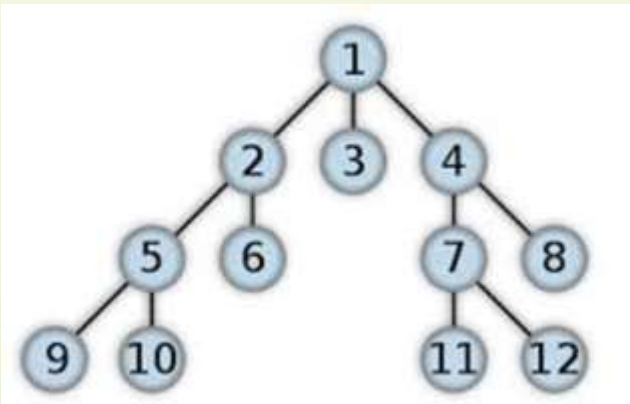
2.3. Реализации деревьев

Реализация деревьев с помощью списков сыновей.

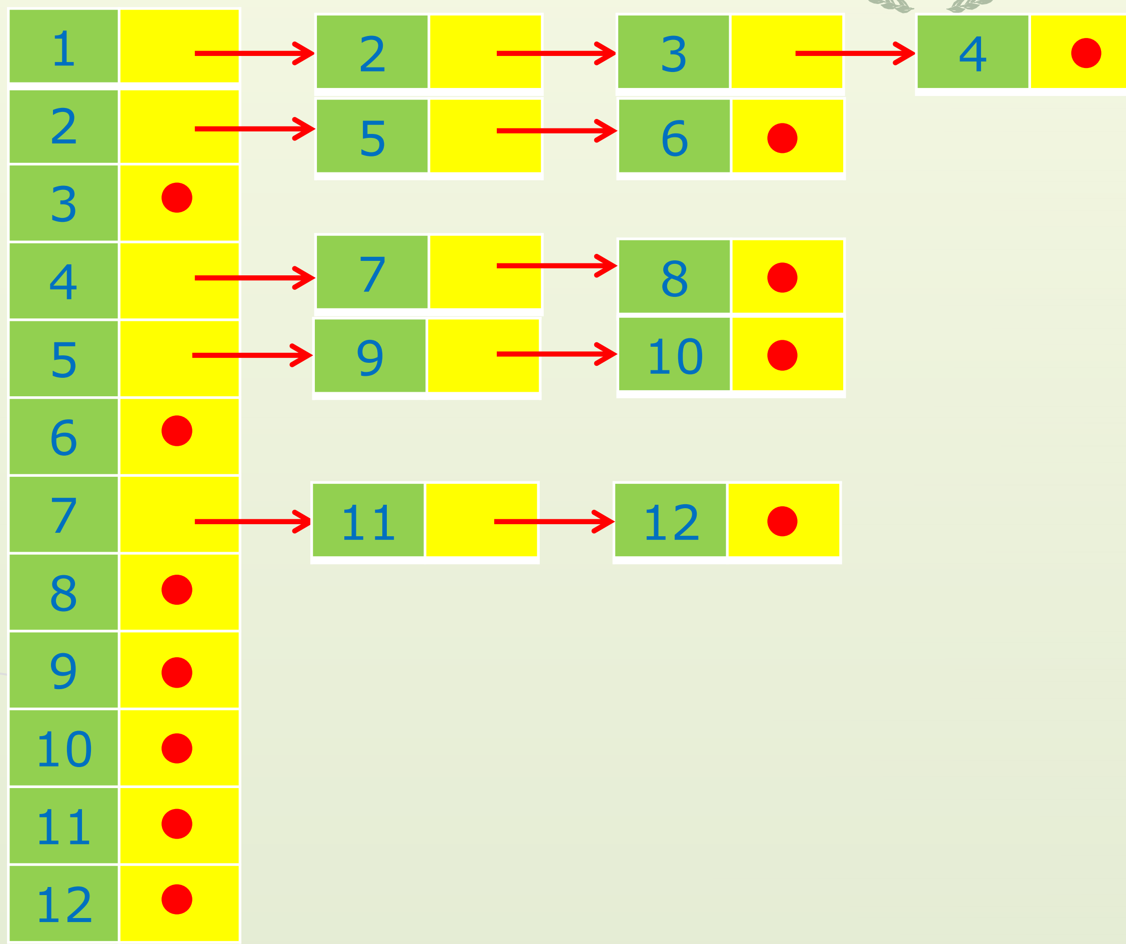
В массиве ячеек заголовков, индексированном номерами (именами) узлов, каждый заголовок указывает на связанный список узлов-сыновей.

2.3. Реализации деревьев

Дерево



Курсоры на списки сыновей



Header

2.3. Реализации деревьев

Что можно сделать легко и за фиксированное время:

- найти сыновей любого узла;
- пройти по любому пути от родителя к потомкам, найти высоту любого узла;
- присвоить метки узлам и определить метки узлов (оператор LABEL).

Что сделать трудно:

- реализовать операторы, требующие информацию о предках (оператор PARENT);
- найти глубину узла.

2.3. Реализации деревьев

Функция нахождения самого левого сына:

```
Function LEFTCHILD(n, T)
Begin
    L:=header[n]
    if Empty(L) then
            return(0) n является листом
           else
            return(First(L)) возвращает первый элемент списка L
                             (сыновья пронумерованы слева направо)
End LEFTCHILD
```

2.3. Реализации деревьев

Функция нахождения родителя:

```
Function PARENT(n, T)
Begin
For p:=1 to maxnodes do – максимальный номер узлов
     begin
        L:=header[p]
        if not Empty(L) then
             begin
                  nodeI:=First(L)
                    if nodeI.N=n then nodeI.N – поле ячейки nodeI,
                                        содержащее номер (метку) узла
                                        return(p)
                  while nodeI.P<>0 do nodeI.P – поле ячейки nodeI,
                              содержащее указатель на следующего сына
                           begin
```

2.3. Реализации деревьев

```
            nodeI:=Next(L, nodeI) получаем следующий элемент списка
             if nodeI.N=n then
                    return(p)
         end
       end
    end
  return(0) родитель не найден
End PARENT
```

2.4. Обходы деревьев

Дано дерево T.

Требуется систематически обойти (упорядочить) по определённому правилу все узлы T.

Виды обходов деревьев:

1) в глубину;
2) в ширину.

Виды обходов в глубину:

1) прямой (префиксный, в прямом порядке, «сверху вниз»);
2) обратный (постфиксный, в обратном порядке, «снизу вверх»);
3) синтаксический (инфиксный, симметричный, «слева направо»).

2.4.1. Префиксный обход

Рекурсивное описание алгоритма:

- Если $T = \Lambda$, то в список обхода заносится пустая запись.
- Если T состоит из одного узла, то в список обхода заносится этот узел.
- Если T – дерево с корнем n и поддеревьями $T_1, \ldots, T_k$, то сначала посещается корень n, затем в прямом порядке узлы поддерева T_1, далее узлы T_2 и т.д. Последними посещаются узлы T_k.

Итак, нужно начать с корня дерева (текущий узел в начале алгоритма), пометить (занести в список обхода) текущий узел. Затем совершить прямой обход самого левого поддерева, прямой обход следующего правого поддерева и т.д.

2.4.1. Префиксный обход

Псевдокод:

```
Procedure PreOrder(n) n – текущий узел – корень, из которого
                                начинается обход
Begin
  занести в список обхода узел n
  for каждый сын x узла n в порядке слева направо do
      PreOrder(x)
End PreOrder
```

Вызов:

PreOrder(Root) Root – корень дерева T

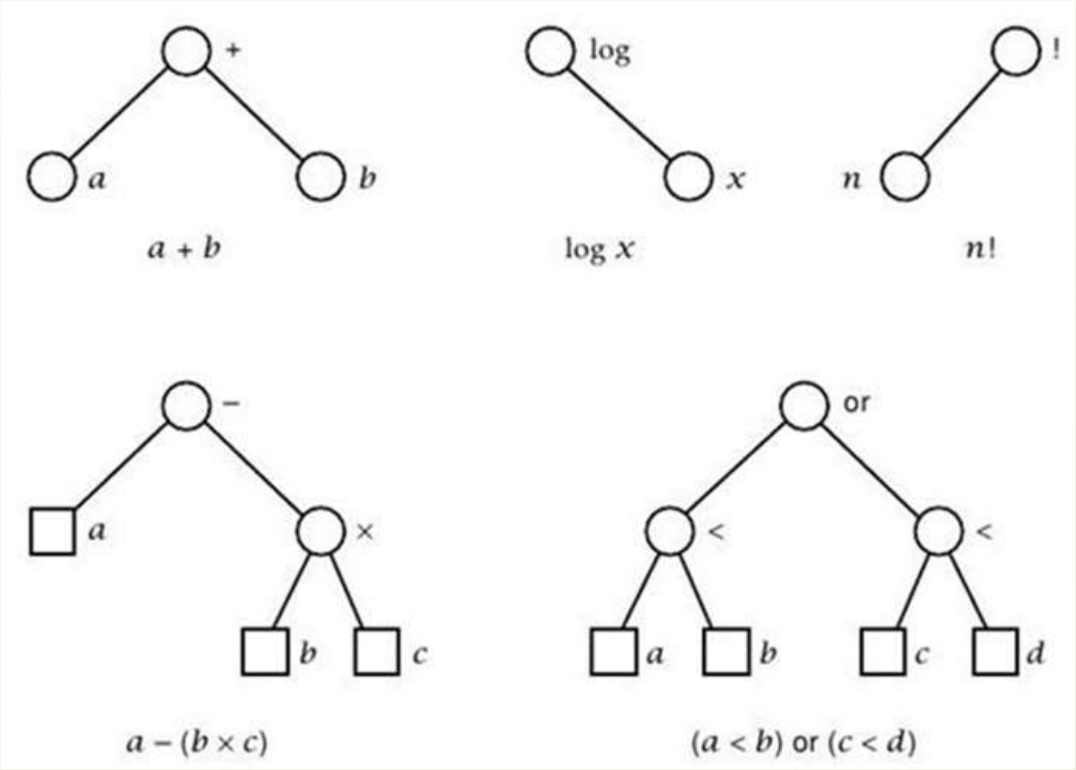
2.4.1. Префиксный обход

В случае дерева выражений получаем известную *префиксную форму* выражений.

Рекуррентное определение префиксной формы выражения:

1. Префиксной формой операнда x является сам этот операнд.
2. Префиксной формой выражения χE, где χ – унарный оператор, является χP, где P - префиксная форма выражения E.
3. Префиксной формой выражения $E_1 \circ E_2$, где $\circ$ - бинарный оператор, является $\circ P_1 P_2$, где P_1, P_2 - префиксные формы выражений E_1, E_2.

2.4.1. Префиксный обход



1. $+ab$;
2. $\log x$;
3. $!\,n$;
4. $-a \times bc$;
5. $or < ab < cd$.

2.4.2. Постфиксный обход

Рекурсивное описание алгоритма:

- Если $T = \Lambda$, то в список обхода заносится пустая запись.
- Если T состоит из одного узла, то в список обхода заносится этот узел.
- Если T – дерево с корнем n и поддеревьями $T_1, \ldots, T_k$, то сначала посещаются в обратном порядке узлы поддерева T_1, далее узлы T_2 и т.д. до поддерева T_k. Последним посещается корень n.

Итак, нужно начать с корня дерева (текущий узел в начале алгоритма), совершить обратный обход самого левого поддерева, обратный обход следующего правого поддерева и т.д. Затем перейти в корень и пометить его (занести в список обхода).

2.4.2. Постфиксный обход

Псевдокод:

```
Procedure PostOrder(n) n – текущий узел – корень, из которого
                             начинается обход
Begin
for каждый сын x узла n в порядке слева направо do
      PostOrder(x)
занести в список обхода узел n
End PostOrder
```

Вызов:

PostOrder(Root) Root – корень дерева T

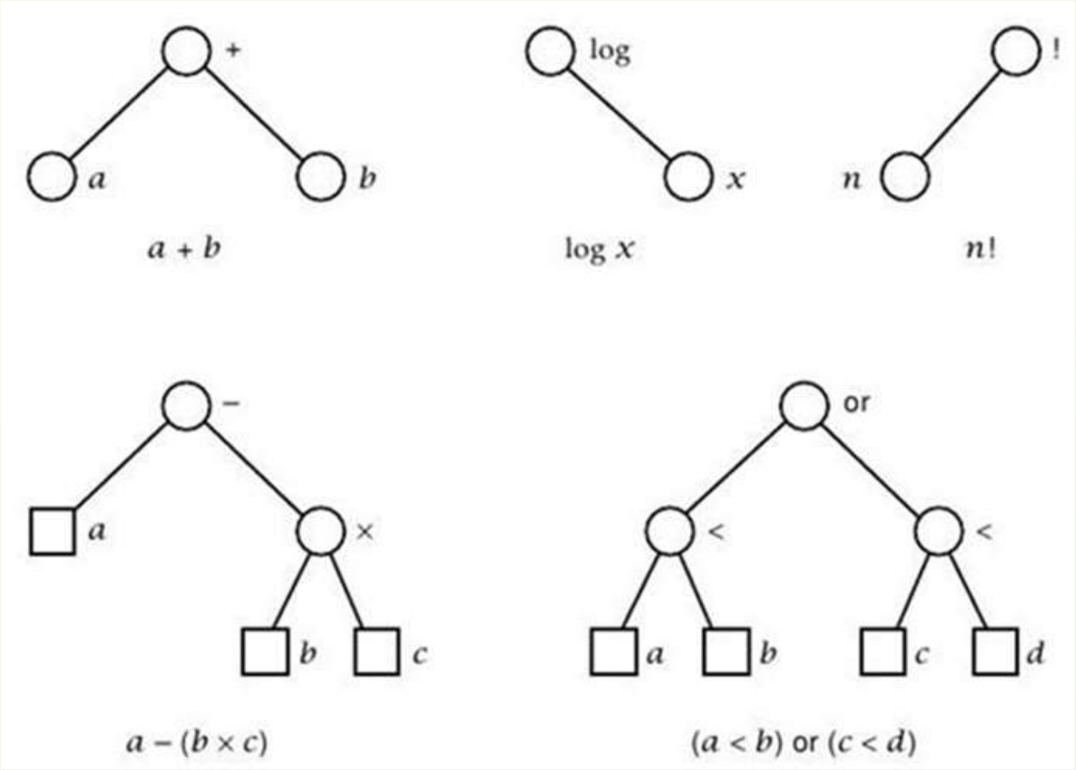
2.4.2. Постфиксный обход

В случае дерева выражений получаем известную *постфиксную* (*польскую*) *форму* записи выражений.

Рекуррентное определение постфиксной формы выражения:

1. Постфиксной формой операнда x является сам этот операнд.
2. Постфиксной формой выражения χE, где χ – унарный оператор, является $P\chi$, где P - постфиксная форма выражения E.
3. Постфиксной формой выражения $E_1 \circ E_2$, где $\circ$ - бинарный оператор, является $P_1 P_2 \circ$, где P_1, P_2 - постфиксные формы выражений E_1, E_2.

2.4.2. Постфиксный обход



1. $ab+$;
2. $x \log$;
3. $n!$;
4. $abc\times -$;
5. $ab<cd<or$.

2.4.3. Инфиксный обход

Рекурсивное описание алгоритма:

- Если $T = \Lambda$, то в список обхода заносится пустая запись.
- Если T состоит из одного узла, то в список обхода заносится этот узел.
- Если T – дерево с корнем n и поддеревьями $T_1, \dots, T_k$, то сначала посещаются в симметричном порядке узлы поддерева T_1, далее корень n, затем опять в симметричном порядке узлы T_2 и т.д. до поддерева T_k.

Итак, нужно начать с корня дерева (текущий узел в начале алгоритма), совершить симметричный обход самого левого поддерева, пометить (занести в список обхода) корень, совершить симметричный обход следующего правого поддерева и т.д.

2.4.3. Инфиксный обход

Псевдокод:

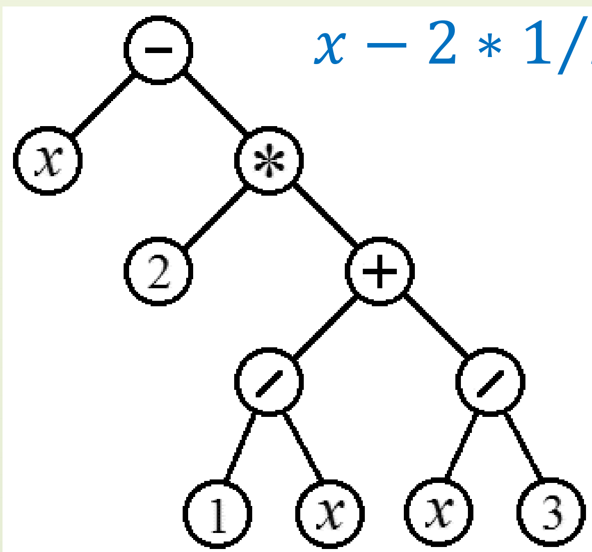
```
Procedure InOrder(n) n – текущий узел – корень, из которого
                                   начинается обход
Begin
If n – лист then
       занести в список обхода узел n
              else
                begin
                   x:=самый левый сын узла n
                   InOrder(x)
                   занести в список обхода узел n
                   for каждый сын x узла n кроме самого левого в
                          порядке слева направо do
                                                 InOrder(x)
                end
End InOrder
```

2.4.3. Инфиксный обход

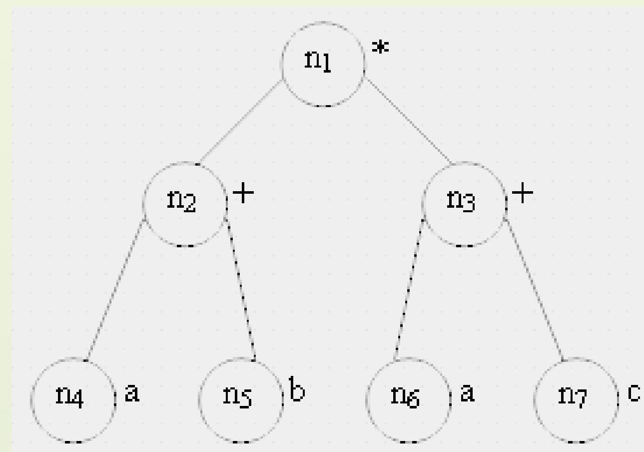
Вызов:

InOrder(Root) Root – корень дерева T

В случае дерева выражений получаем *инфиксную форму* записи выражений, совпадающую с привычной «алгебраической», но не содержащей скобок.



$$x - 2 * 1/x + x/3$$



$$a + b * a + c$$

2.4.4. Обход в ширину

Все узлы дерева разбиваются на уровни. К одному уровню относим узлы, имеющие одинаковую глубину. Таким образом, на нулевом уровне находится только один узел – корень, на 1-м – сыновья корня и т.д.

Описание алгоритма:

1) пометить узел нулевого уровня (корень дерева);
2) пометить все узлы 1-го уровня;
3) пометить все узлы 2-го уровня;

…

Обход в ширину производится с помощью очереди. Первоначально в очередь помещается корень, затем, пока очередь не пуста, выполняются следующие действия:

1. Из очереди извлекается очередной узел.
2. Этот узел помечается (заносится в список обхода).
3. В очередь добавляются сыновья этого узла (в порядке слева направо).

2.4.4. Обход в ширину

Псевдокод:

```
Procedure WideOrder(T)
Begin
Q:={root} Q – очередь, root – корень T
While Q<>∅ do
    begin
        n:=первый элемент Q
        Q:=Q\{n} n удаляется из очереди
        занести в список обхода узел n
        Q:=Q∪{сыновья n} добавляются в конец очереди
    end
End WideOrder
```