Тематика семестра

Нелинейные структуры данных

Иерархическая структура – дерево

Граф

Стратегии разработки алгоритмов (задачи выбора решения)

Методы и алгоритмы кодирования и сжатия информации

Учебный план дисциплины

Лекции 16 часов (8 лекций)

Практические занятия 32 часа (16 занятий)

Отчетность: экзамен

СРС 42 часа

1. Кормен Т.Х., Лейзерсон Ч.И., Ривест Р.Л.Алгоритмы: построение и анализ. - Вильямс, 2019

2. Дональд Э. Кнут. Искусство программирования [Текст]. Т.2 / — М.: Вильямс, 2013. — 828 с.. — Библиогр.: с.

3. Дональд Э. Кнут. Искусство программирования [Текст]. Т1. / — М.:Вильямс, 2014. — 712 с.. — Библиогр. в конце глав

4. Хусаинов Б.С. Структуры и алгоритмы обработки данных. Примеры на языке Си. Учебное пособие

5. Адитья Бхаргава Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих – Питер.:СПб, 2017

6. Н. Вирт Алгоритмы и структуры данных. Классика программирования – М.:ДМК Пресс, 2016. — 272 с.

7. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы М.:Вильямс, 2016 — 400 с.

8. Р. Круз. Структуры данных и проектирование программ : Пер. с англ. — М.: БИНОМ. Лаборатория знаний, 2017. — 766 с .

9. Г. Шилдт.Полный справочник по C++ : Пер. с англ. / — М.: ООО

"И.Д. Вильямс", 2016. — 796 с.: ил. — Предм. указ.: с. 787-796

10. Лафоре Р. Объектно-ориентированное программирование/- Питер.:СПб, 2018

Лекция 1

Введение в нелинейные структуры

Иерархическая структура – k-арное дерево

Повторим!!!!

Структуры данных

Структура данных - совокупность логически связанных элементов данных между которыми существуют некоторые отношения, при этом элементами данных могут быть как простые (атомарные) значения, так и структуры данных.

Структура данных - это способ хранения и организации данных, облегчающий доступ к этим данным и их модификацию.

Ни одна структура данных не является универсальной и не может подходить для всех целей, поэтому важно знать преимущества и ограничения, присущие некоторым из них.

Позволяет хранить данные и отношения между ними.

Так как структура данных в реализации представляет граф, то математически структуру можно представить как множество элементов S={D,R}. Элемент структуры можно описать как (d, r).

От задачи к программе

Абстрактный тип данных – это математическая модель данных с совокупностью операций, определенных в данной модели.

Три уровня представления данных

Структура данных задачи

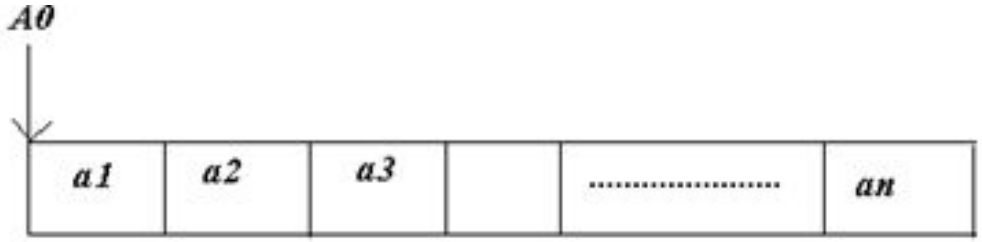
Структура данных языка программирования

Структура хранения данных

Структуры хранения

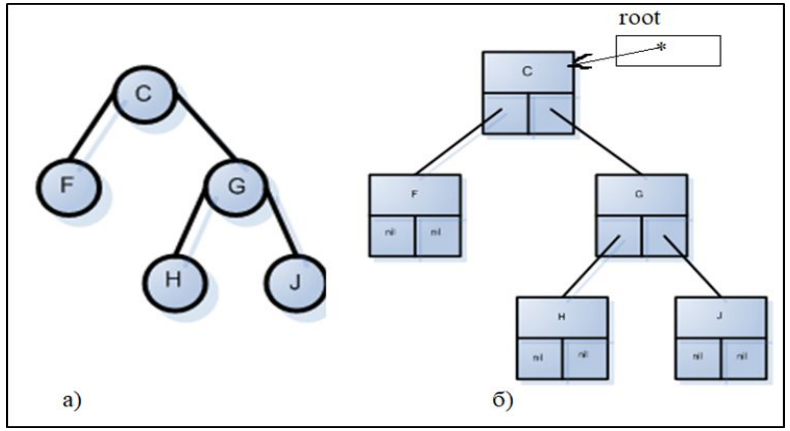
Векторное хранение

Размещение элементов структуры в последовательно организованной памяти. Элемент хранит только данные, отношения между элементами реализуются не явно.



Динамическое (связанное/списковое) хранение

Размещение элементов в динамической памяти, создании каждого элемента отдельно. Отношения реализуются явно, на основе указателей.



Сложность алгоритма

Эффективный алгоритм должен удовлетворять требованиям:

- *приемлемое время исполнения*
- *разумной объем оперативной памяти*.

Совокупность этих характеристик составляет понятие ***сложность алгоритма.***

При увеличении требующихся ресурсов (время и память) **сложность возрастает, а эффективность падает.**

Различают:

1. ***Количественная сложность*** определяется значениями:

Временная сложность **выражается** ***в количестве операций, выполняемых алгоритмом на некотором идеализированном компьютере****, для которого время выполнения каждого вида операций известно и постоянно.*

Емкостная сложность определяется объемом данных (входных, промежуточных, выходных - n), числом задействованных ячеек памяти.

2. ***Качественная сложность*** определяет эффективности:

Эффективный алгоритм

Не эффективный алгоритм

Пример 1. Определение теоретической вычислительной сложности алгоритма

Вычислить среднее арифметическое всех положительных чисел массива A из n элементов. Результат - функция T(n), определяющая зависимость количества выполняемых операций от n. Где ci – константа времени выполнения оператора на идеализированной ЭВМ

Оператор	Количество выполнений оператора	
Sum←0	1	c1
count ←0	1	c2
For i←1 to n **do**	n+1	c3
If(A[i]>0)	n	c4
sum←sum+A[i]	n	c5
count←count+1 endIf	n	c6
od		
If (count≠0)	1	c7
return sum/count	1	c8
Else		
return -1 endIf	1	c9

$T(n)=c1*1+c2*1+c3*(n+1)+c4*n+c5*n+c6*n+c7*1+c8*1+c9*1=(c3+c4+c5+c6)n+(c1+c2+c3+c7+c8+c9)=An+B$

Асимптотический анализ и асимптотические обозначения вычислительной сложности алгоритма

Асимптотический анализ – упрощенный способ оценки сложности алгоритма

Асимптотические обозначения - указывают границы роста времени выполнения алгоритма при увеличении размера задачи n.

Виды	Когда используется?	Рост	Какое правило асимптотической оценки поддерживает?
Θ (g(n))	Оценка среднего случая	Равно a=b	$f(n) = \Theta(g(n))$, c1>0, c2>0 $c_1*g(n) \leq f(n) \leq c_2*g(n)$ для всех $n > n_o$
O(g(n))	Оценка худшего случая	Меньше или равно a<=b	$f(n) = O(g(n))$, существуют $c > 0$, $n_0 > 0$ и $n > n_0$ $0 < f(n) \leq c*g(n)$
o (g(n))	Оценка худшего случая	Меньше a<b	$f(n) = O(g(n))$, для любого $c > 0$, $n_0 > 0$ $0 < f(n) < c*g(n)$ и $n > n_0$
Ω (g(n))	Оценка лучшего случая	Больше или равно a>=b	$f(n) = \Omega(g(n))$ $c > 0$, $n_0 > 0$ $0 < c*g(n) \leq f(n)$ и $n > n_0$
ω(g(n))	Оценка лучшего случая	Больше a>b	$f(n) = \Omega(g(n))$ для любой c>0 и n>0 $0 < c*g(n) < f(n)$ при $n > n_0$

Часто встречающиеся асимптотические обозначения

Порядок	Виды алгоритмов	Пример алгоритма
O(1) константа	Порядок не зависит от n. Алгоритм выполняется за константное время.	Замена элемента x в массив A в позицию с номером k. Цикл из 100 итераций.
O(n) линейный рост	Линейный алгоритм. Время пропорционально количеству входных данных.	Алгоритм линейного поиска значения в массиве.
$O(n^2)$ полиномиальный рост	Квадратичный алгоритм. Используется для обработки небольших наборов данных.	Алгоритмы простых внутренних сортировок.
$O(n^3)$ полиномиальный рост	Алгоритм с кубичным временем выполнения. Выполняется медленно.	Алгоритм Уоршалла - Флойда для графа.
$O(\log_2 n)$ логарифмический рост	Логарифмические алгоритмы. Время алгоритмов, которые делят данные на подсписки.	Бинарный поиск,
$O((n*\log_2 n)$ квазилинейный рост	Логарифмические алгоритмы. Время алгоритмов, которые делят данные на подсписки.	Быстрая сортировка. Метод Хоара.
$O(2^n)$ Экспоненциальный.	Самый сложный алгоритм, очень медленный. Используется только для малых n.	Поиск значения n – ого числа Фибоначчи

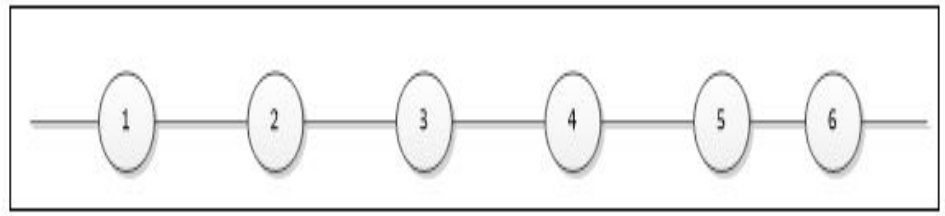
Нелинейные структуры данных

Позволяют хранить более сложные отношения между элементами структуры данных, по сравнению с линейными отношениями

Нелинейные структуры данных

1. **Иерархическая структура (Дерево)**
2. **Граф (Сеть)**
3. **Лес**

Линейная структура **представляет список элементов, упорядоченных по положению. Доступ к элементу прямой по номеру. Поддерживают мощность отношений *«один-к-одному».***

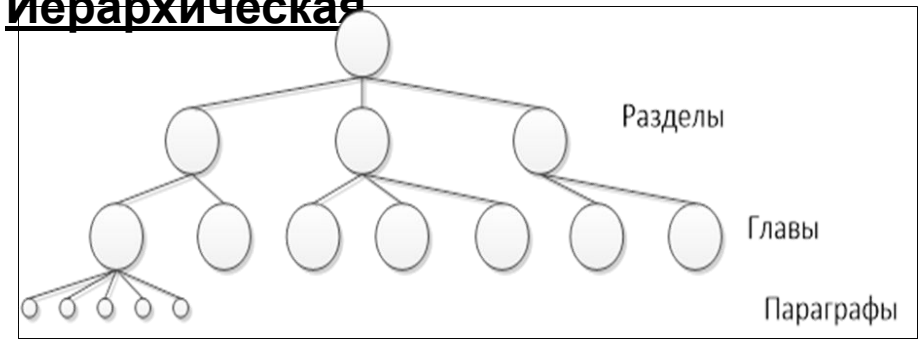


Нелинейная структура определяет совокупность элементов без позиционного упорядочения элементов.

Иерархические **(древовидные) структуры**

Поддерживают **отношения подчинения с мощностью** *один к многим:* ***у каждого потомка только один предок****.* Доступ к элементу через путь. Примером может служить дерево папок операционной системы.

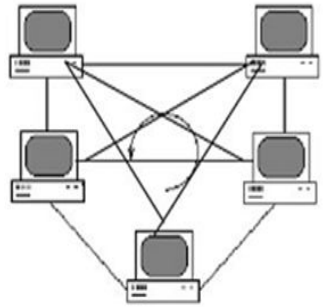
Иерархическая



Сеть (граф)

Поддерживает **отношение *соединения пар*** элементов, образуя мощность отношения *многие к многим.* Примером может служить сеть автомобильных дорог, где элементами являются населенные пункты, а дороги – это соединения (в теории графов – это ребро).

Сеть (граф)



Группа

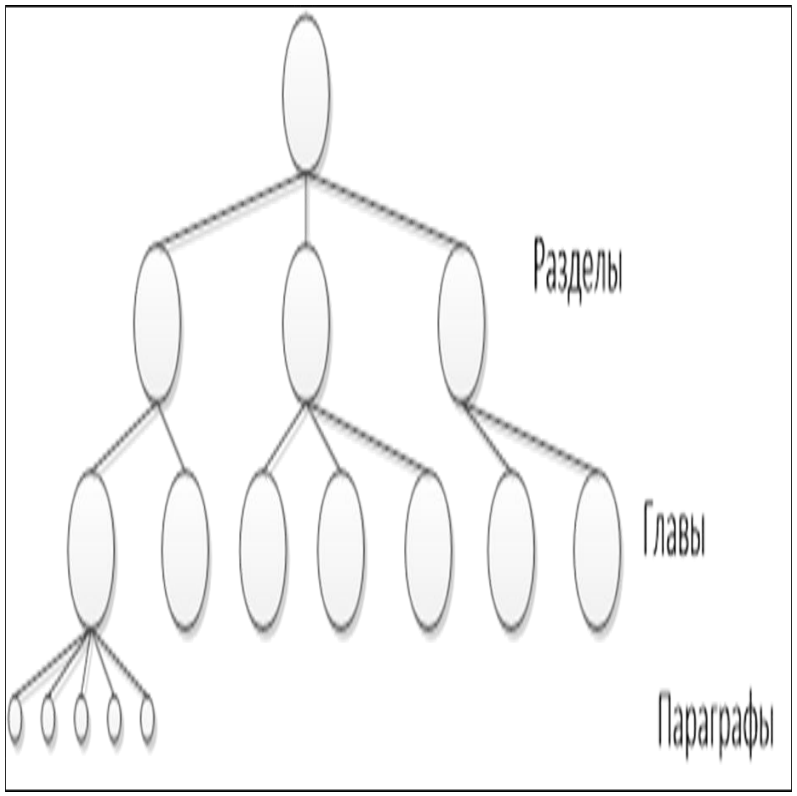
Представляет ***множество*** элементов.

Примеры структур данных с нелинейными отношениями

1) Оглавление в книге

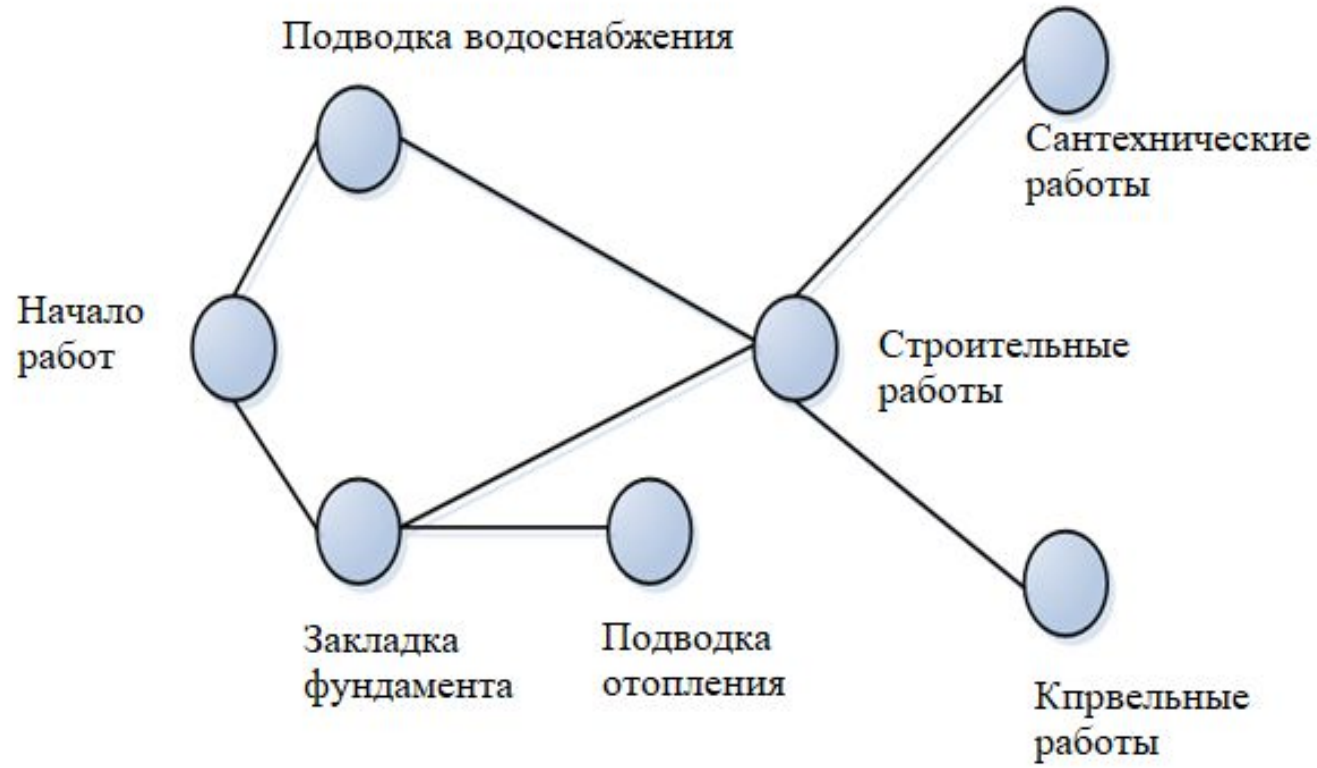
Раздел 1.
- Глава 1
  - Название темы
  - Название темы
- Глава 2
  - Название темы
  - Название темы

Раздел 2
- Глава 1
  - Название темы
  - Название темы
- Глава 2
  - Название темы
  - Название темы



2) **Организационная диаграмма, представляющая структуру административного звена организации**

3) Организационная диаграмма выполнения строительных работ при строительстве здания



Примеры структур данных с нелинейными отношениями (продолжение)

4) Карта автомобильных дорог, карта авиационных перевозок, карта метро (например, Московского).

5) Файловая структура компьютера (дерево папок).

Во всех этих структурах данных между элементами существуют сложные отношения: иерархические или сетевые.

Иерархические структуры данных (деревья)

В иерархической структуре данных, элементы подчинены отношению:

у каждого потомка есть только один предок

Модель иерархической структуры

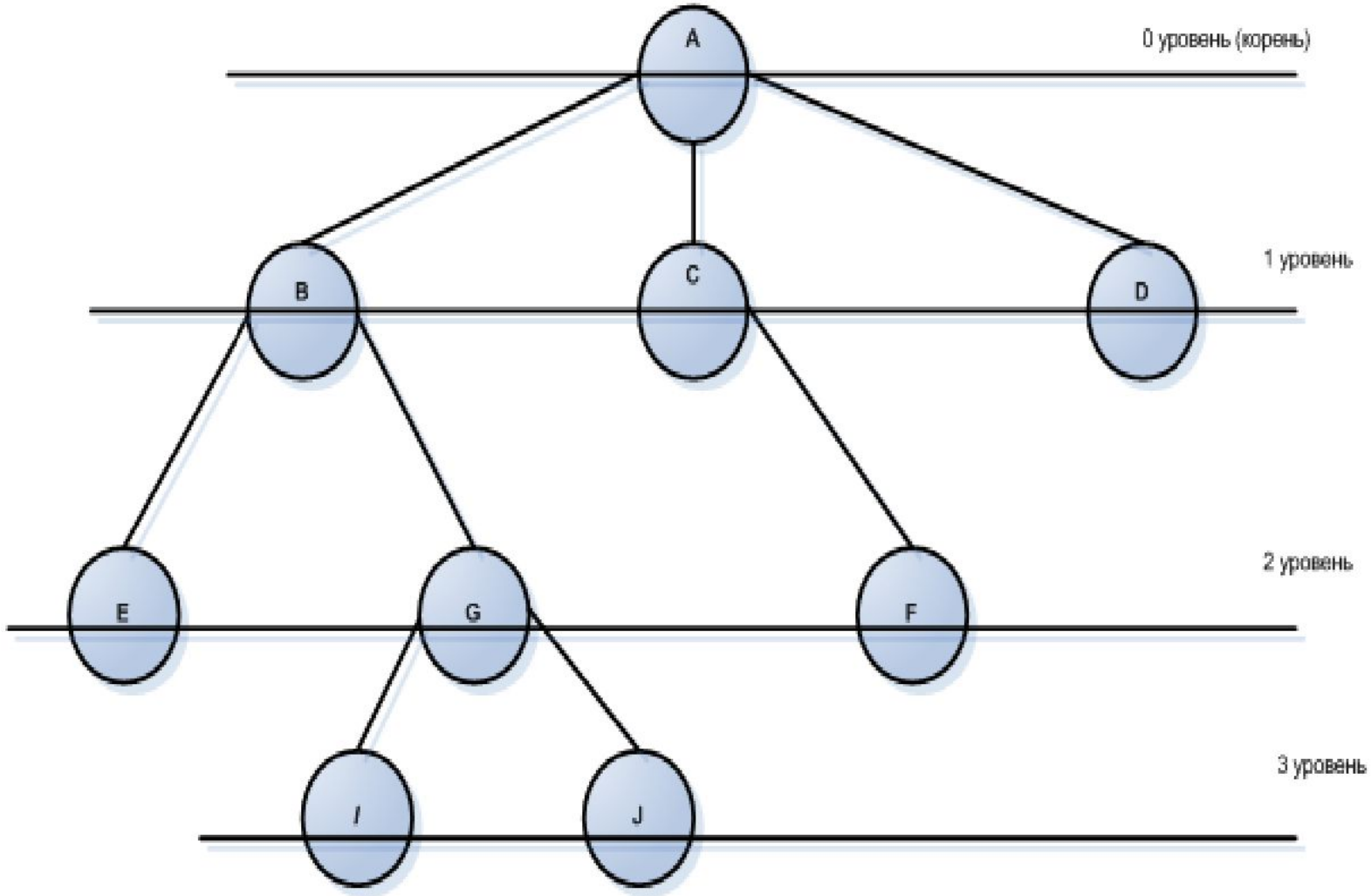


Рис.1. Дерево – иерархическая структура данных

Основные термины иерархической структуры

В виде кружков представлены элементы данных (вершины дерева/узлы), линии, их соединяющие, указывают связь узлов.

Модель на рис. 1 представляет иерархические отношения между узлами.

Узел ***A –*** корневой узел (предок или отец), он находится на нулевом уровне, не имеет предков.

Узлы ***B, C, D*** – потомки (сыновья/ дочерние узлы) узла ***A,*** находятся на уровне 1. Представляют корни своих деревьев.

Узел ***B*** является предком или отцом деревьев с корнями ***E*** и ***G***.

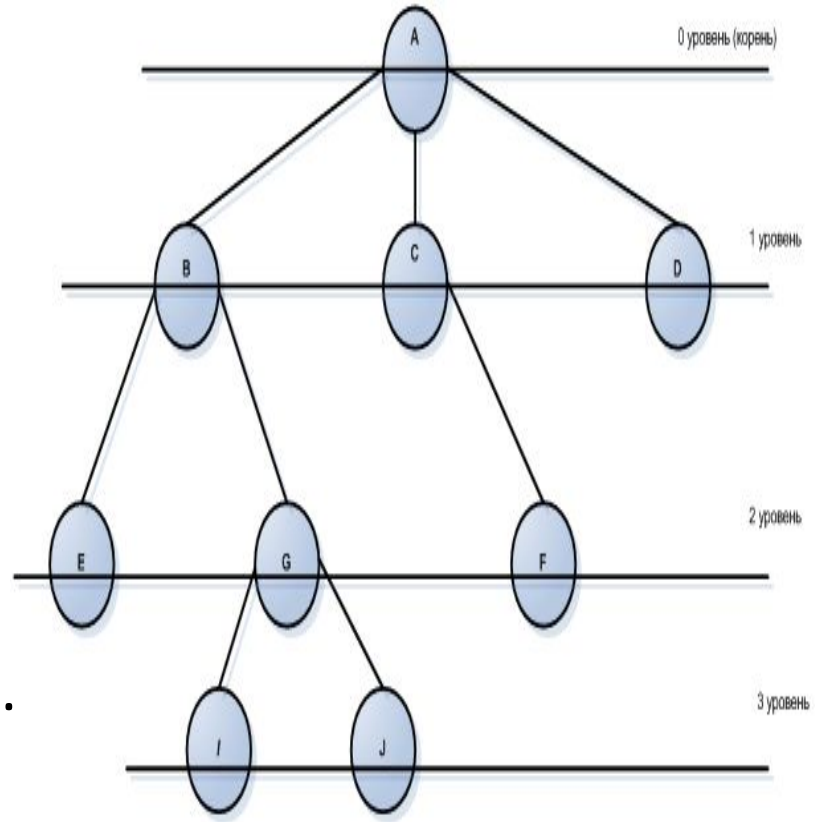
Основные термины иерархической структуры (продолжение)

1. В деревьях действует отношение: ***у каждого потомка только один предок***.
2. Каждый узел – это корень своего дерева или поддерево родителя (предка).
3. Узел, который не имеет предков, называют ***корневым узлом***.
4. ***Высота дерева*** – кол-во ребер между корнем и максимально удаленным узлом (листом) или максимальный уровень.

Высота дерева, представленного на рис.1. равна 3.

5. ***Лист дерева*** – узел, который не имеет потомков.

Листья дерева A (рис.1.): E, I, J, F, D.



6. ***Степень узла*** – число непосредственных потомков.

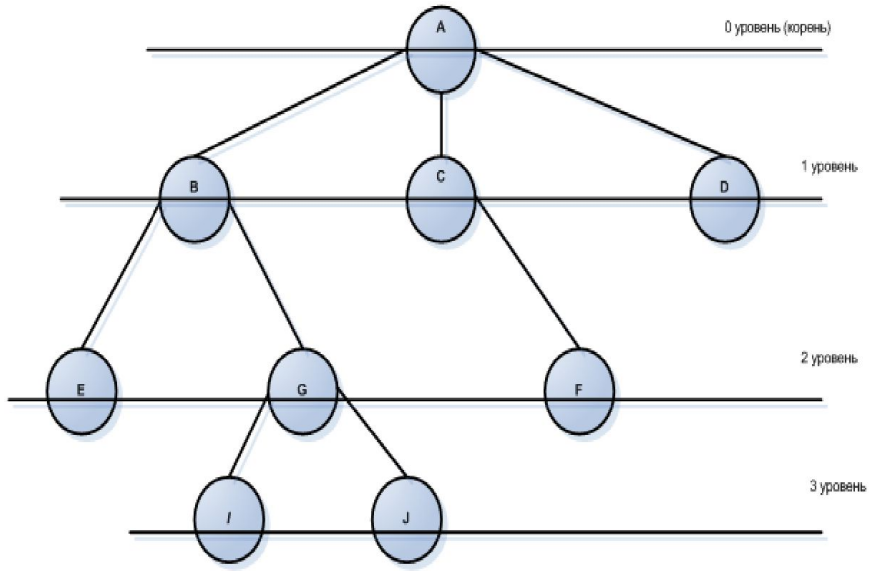
Степень узла A равна 3, а степень узла G равна 2, степень узла C – 1, J – 0.

7. ***Степень дерева*** - максимальная степень его узлов. Дерево A имеет степень 3.

Основные термины иерархической структуры (продолжение)

8. ***Путь в дереве*** – последовательность узлов от корня к указанному узлу.

Пример. Путь к узлу J в дереве на рис.1: A B G J.



9. ***Длина пути в дереве до узла*** – кол-во ребер от корня до узла или номер уровня, на котором находится узел. Длина пути до узла J в дереве рис.1 равна 3.

10. ***Длина пути в дереве*** равна сумме длин всех его ребер (или количество ребер).

Для рассматриваемого дерева длина пути равна 8.

11. ***Помеченные деревья*** - узлы, которых имеют метки – имена или номера.

Основные термины иерархической структуры (продолжение)

13. Упорядоченные и неупорядоченные деревья

Дерево считается упорядоченным, если существует порядок перечисления узлов. Порядок узлов определяется обычно так: сыновья упорядочены слева направо: «левый» сын и его «правые» братья.

Так для дерева A рис.1. узел B – это левый сын, C и D его правые братья узла B, но все они сыновья A.

Если порядок сыновей игнорируется, то дерево называется *неупорядоченным.*

На рис. 3 представлены два разных упорядоченных дерева, так как их сыновья перечислены по разному.

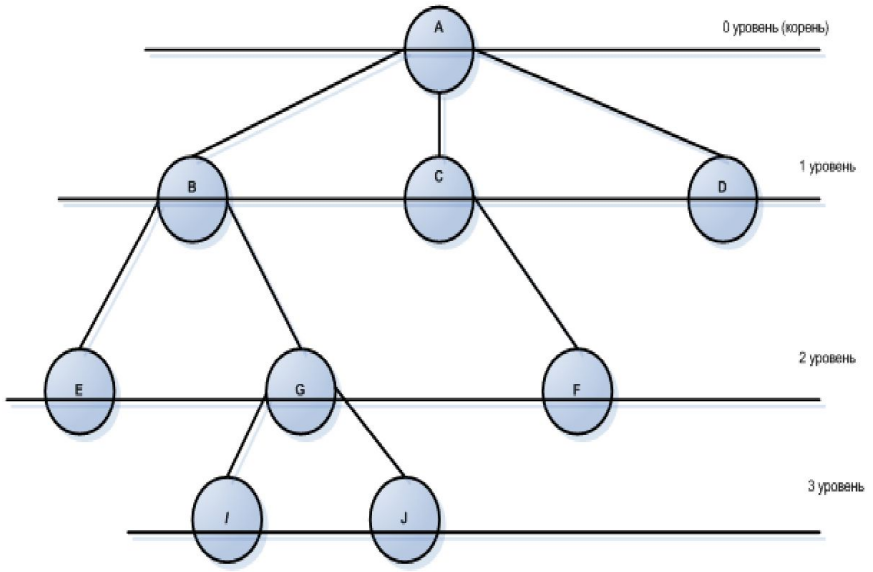
Рисунок 3. Разные упорядоченные деревья

Определение иерархической структуры (дерева)

Дерево - это совокупность узлов, один из которых корень, и отношений, образующих иерархическую структуру.

Дерево может быть:

- пустым;
- или состоять из одного узла, который является корнем своего дерева;
- или состоять из корня, связанного с несколькими узлами – корнями деревьев (поддеревьев данного дерева).

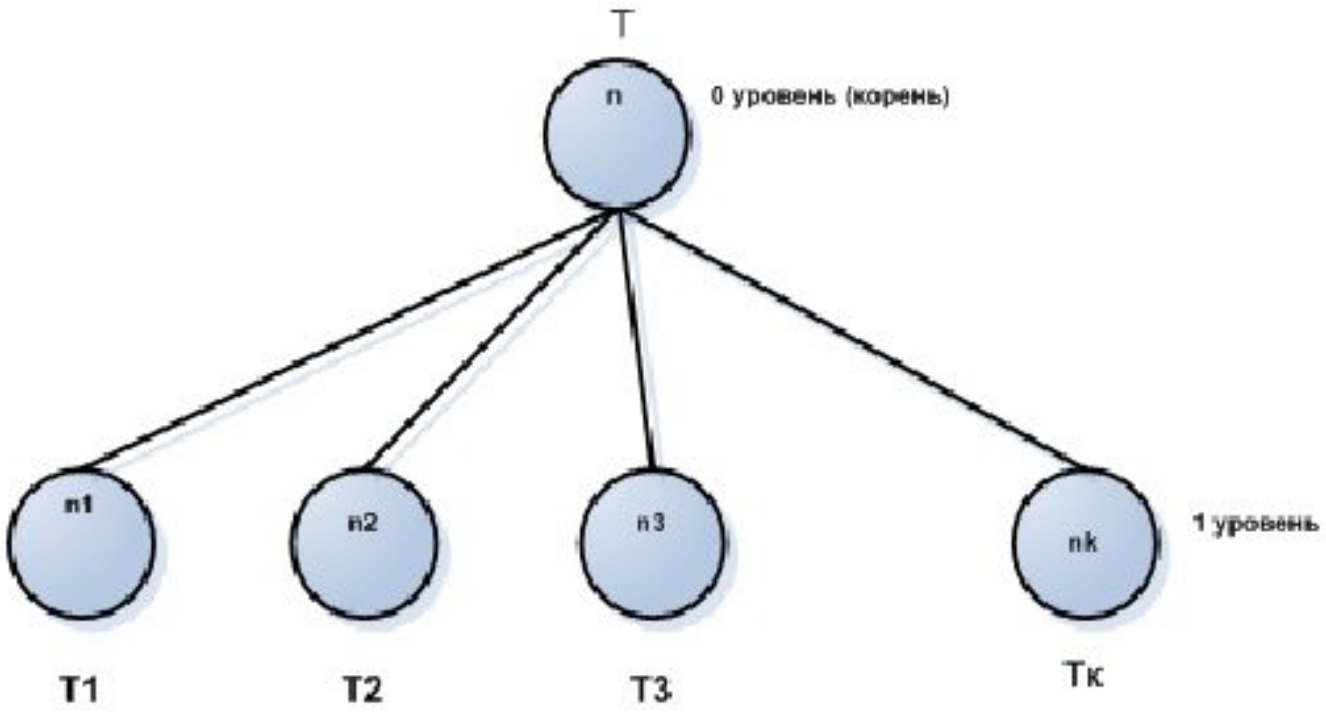


Определение k – ароного дерева

Арность дерева определяется степенью дерева.

Пусть $T_1\ T_2\ T_3 \ldots T_k$ деревья с корнями $n_1, n_2, n_3 \ldots n_к$.

Тогда можно построить новое дерево T, корнем которого будет узел n, а узлы $n_1, n_2, n_3 \ldots n_к$ – корни его поддеревьев.



Рекурсивное определение дерева

Из приведенных определений видно, что ***структура определена с помощью рекурсии,*** поэтому можно ввести следующее определение дерева с базовым типом T.

Дерево типа T это:

- Пустая структура
- Или Состоит из одного узла ***n*** типа T
- Или Состоит из узла типа T, с которым связано конечное число (k) древовидных структур с корнями (n1,n2,…nk) базового типа T, т.е. поддеревьями .

Виды деревьев

- Сильно ветвящиеся деревья (степень дерева >2).
- Двоичное (бинарное) дерево (степень дерева <=2).
  - Двоичное идеально сбалансированное дерево.
  - Двоичное дерево поиска.
  - Двоичное идеально сбалансированное дерево поиска (AVL – дерево).
  - Красно – черное дерево.
  - Косое дерево.
- B - дерево – сбалансированное k-арное дерево

Способы реализации деревьев

Для представления упорядоченных деревьев в памяти компьютера можно использовать:

- линейные структуры данных (массив, список)
- более сложные пользовательские структуры узла дерева, с указателями на сыновей, для создания иерархических структур.

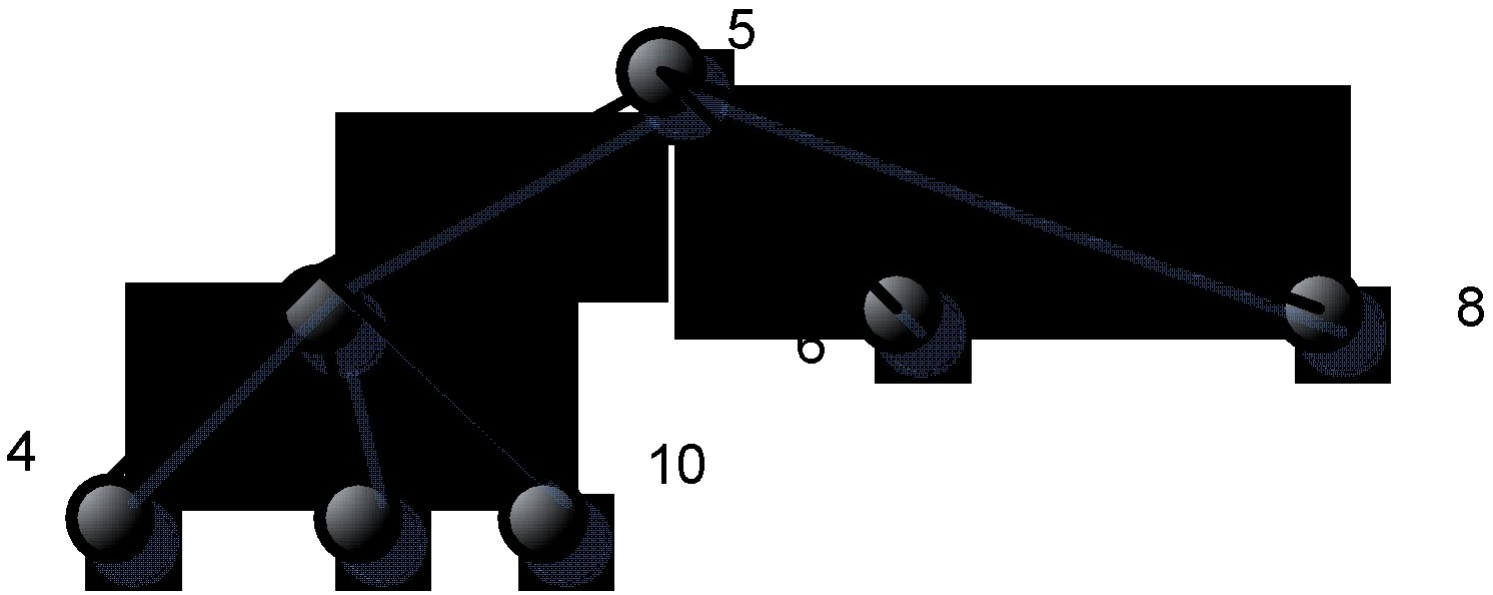
Структура данных должна обеспечить хранение: данных каждого узла и иерархических связей между узлами потомков и родителей.

Реализации

- На массиве родителей
- На списке сыновей
  - Список левых сыновей и правых братьев
  - Реализация на курсоре
  - Реализация дерева на таблице «левых» сыновей и «правых» братьев

Способы реализации деревьев: Массив родителей

Пусть есть дерево, представленное на рисунке, создадим его массив родителей



1	2	3	4	5	6	7	8	9	10	Метки узлов
-1	-1	-1	7	0	5	5	5	7	7	Ссылки на родителей (метки родителей)

Способы программной реализации массива родителей

Способ 1.

Определение количества максимально возможных узлов Maxlen.

Определение массива для хранения ссылок на родителей.

Определение массива для хранения данных каждого узла.

Данные, хранящиеся в узле, могут иметь любую структуру, в рассматриваемом примере введем для этого тип TDataNode.

```
const   Maxlen=100;
typedef unsigned int[MaxLen] Tree;    //реализация отношений
typedef  TDataNode[MaxLen] infoNodeTree;
Tree TreeParents;                          //массив родителей
infoNodeTree  InfoTree;              // хранения  данных каждого узла
```

Способы программной реализации массива родителей

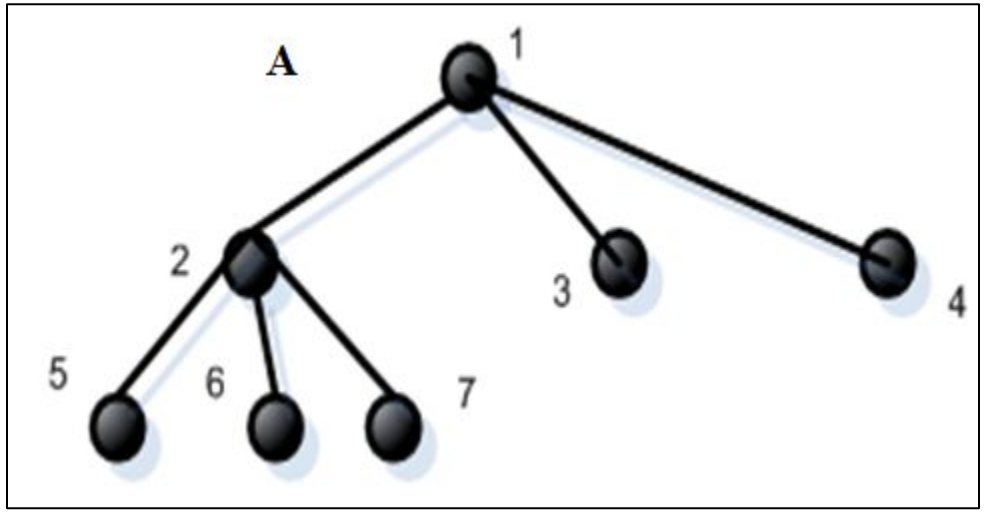
Способ 2. Определение всех параметров дерева в одной структуре

```
struct Tree
{
   unsigned int  Tree [MaxLen];          //массив родителей
   TDataNode infoNodeTree[MaxLen];//массив значений
   n:byte;   //количество узлов
};
   Tree T;                                //реализация дерева
```

Применение представления дерева на массиве родителей

Задача. Найти левого сына заданного узла и его правого брата.

Для данной реализации задача выполнима, если ввести нумерацию сыновей: последовательно слева направо, например, так, как показано на рисунке. Тогда можно сформировать массив родителей. Разделим задачу на две задачи: 1) поиск левого сына; 2) поиск правого брата. *Описание решений на следующих слайдах.* :



1	2	3	4	5	6	7
0	1	1	1	2	2	2

Задача 1.Найти левого сына заданного узла

Для поиска ***левого сына*** узла 2 в дереве A , при использовании массива родителей, достаточно найти первый элемент массива со значением равным 2.

Найти левого сына заданного узла i и вернуть его метку.

Постановка задачи

Дано. Дерево T, реализованное по способу 2(слайд 31). Метка узла i.

Результат. Метка левого сына узла i.

//предусловие: T не пустое дерево.i < T.n.

//постусловие: возвращает номер элемента массива T.TreeParents, значением

//которого является метка i или -1, если такого значения нет в массиве.

```
int leftSon(Ttree T, int i){ //первый узел со значением i
  int j=1;
    while (j<=T.n && T.TreeParents[j]!=i) {
    j++;
  }
  if (j<=T.n) return j;
```

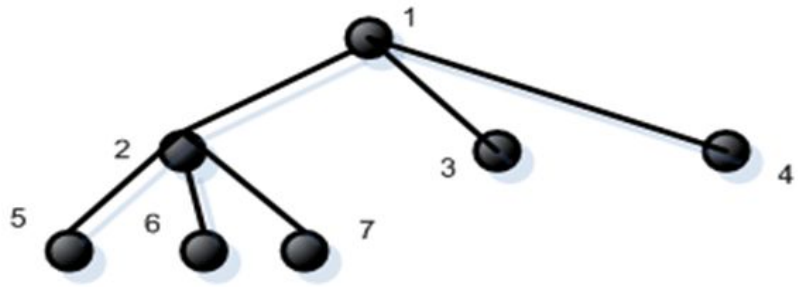
Задача 2.Найти правого брата заданного узла

При решении задачи по поиску ***правого брата*** узла достаточно найти ссылку на родителя данного узла (метку), и тогда, следующий элемент массива, содержащий метку родителя, и будет правым братом. Например, найти правого брата узла 5.

Постановка задачи

Дано. Дерево T, реализованное по способу 2 (слайд 31). Метка узла i, правого брата которого требуется найти.

Результат. Метка правого брата узла I или -1, если такое



1	2	3	4	5	6	7
0	1	1	1	2	2	2

```
//Поиск родителя узла i
int Parent(Tree T, int i){
   return T.TreeParents[i];
}
//Поиск правого брата узла i
int rightBrather(Tree T, int i){
 int j,p;
   p=Parent(T,i); //Метка родителя узла i
    //Поиск правого брата с узла следующего за исходным
    if (T.TreeParents[i][i+1]==p) return (i+1);
   else return -1;
}
```

Способы реализации деревьев: Списки сыновей

Представление дерева основано на массиве линейных списков. Каждый список хранит метки узлов сыновей узла с меткой равной индексу элемента массива. На рис.5 представлено дерево и его реализация на списках сыновей.

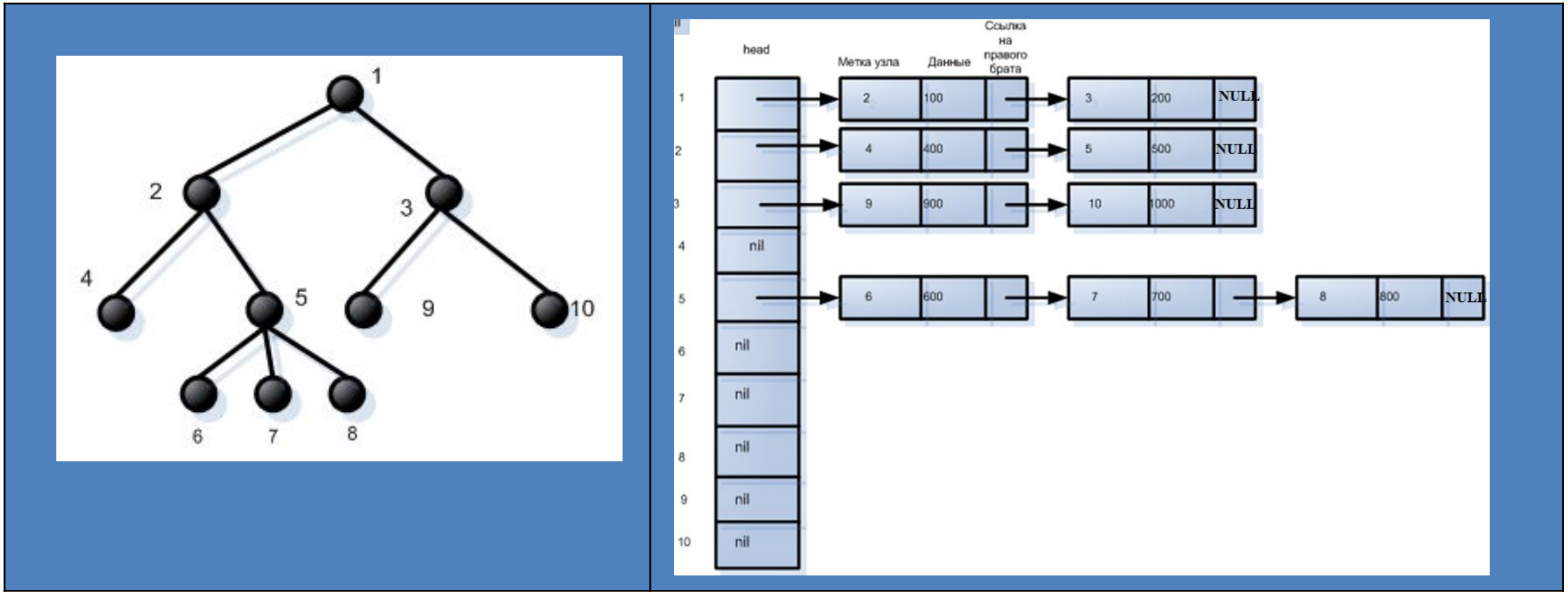


Рис.5. Представление k-арного дерева на списках сыновей

Реализация дерева на списках сыновей

Структура элемента списка

```
struct Tnode{
    Tinfo info;          //данные элемента
    Tnode *next;
};
```

Структура дерева

```
struct Tree{
    Tnode *L[MaxLen];     //Массив указателей на списки сыновей
    Int Root;             // корень – метка узла
    int n;
};
Tree T;                   //Дерево типа Tree
```

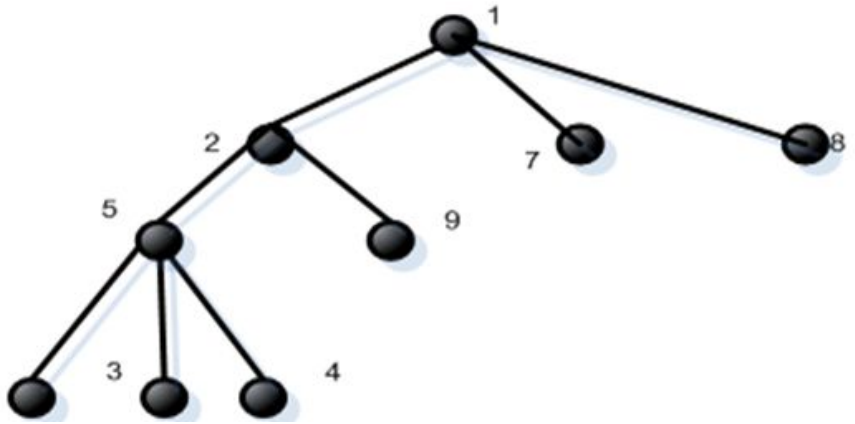
Способы реализации деревьев: список левых сыновей и правых братьев

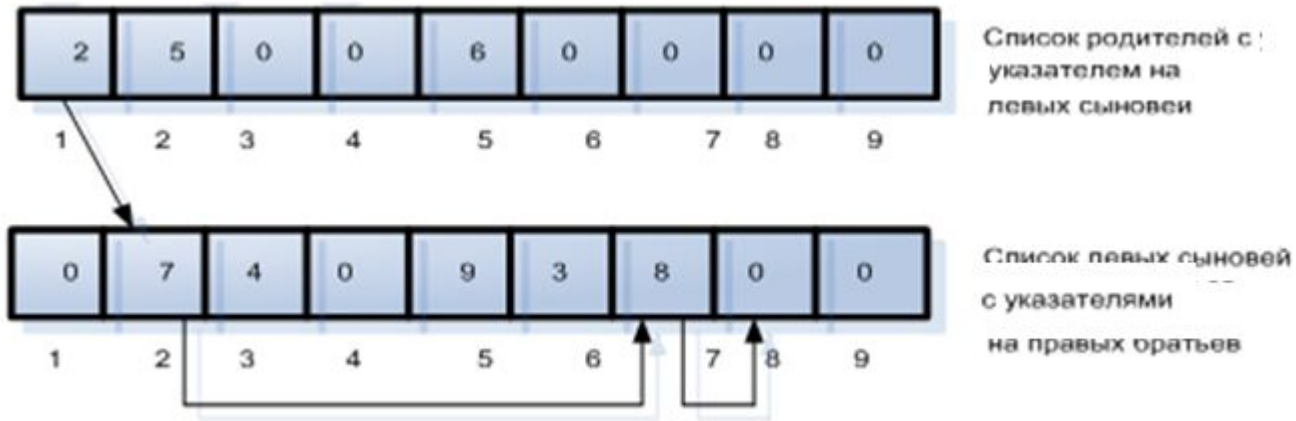
Такой подход по реализации списков на массивах, называется *Реализация на курсоре.*

Списки реализуются в двух массивах:

Список родителей со ссылкой на левых сыновей: элемент одного массива (это 1) хранит метку левого сына узла с меткой равной индексу элемента,

Список левых сыновей со ссылкой на правых братьев - массив хранит цепочку

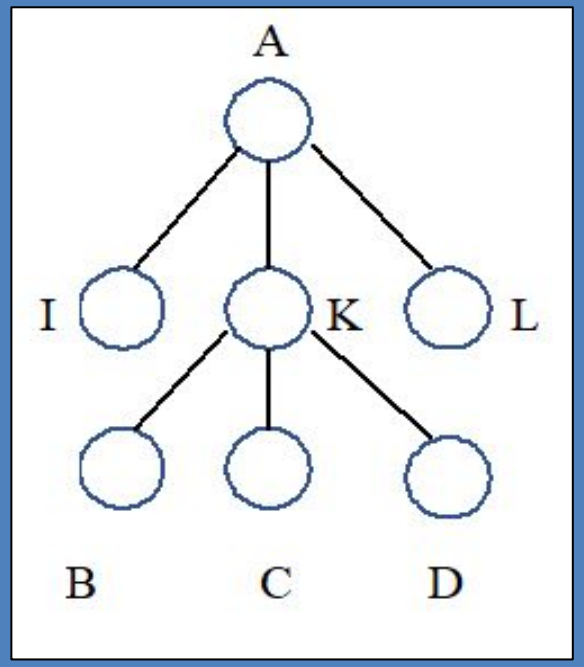




Цепочка меток: левый сын узла 1 имеет метку 2, переходим во второй массив и по индексу 2 находим следующего сына узла 1 это 7, по метке 7 находим следующего сына узла 1 это узел с меткой 8, значение этого узла равно 0, что означает конец списка сыновей узла 1.

Способы реализации деревьев:

Список левых сыновей и правых братьев в таблице



Размещение узлов в таблице

Индекс элемента таблицы	Индекс левого сына	Метка узла	Индекс правого брата узла
1	0	B	0
2	0	L	0
3	0	D	0
4	0	K	0
5	0	I	0
6	0	A	0
7	0	C	0

Сохранение связей узлов

Индекс элемента таблицы	Индекс левого сына	Метка узла	Индекс правого брата узла
1	0	B	7
2	0	L	0
3	0	D	0
4	1	K	2
5	0	I	4
6	5	A	0
7	0	C	3

Реализация дерева в таблице

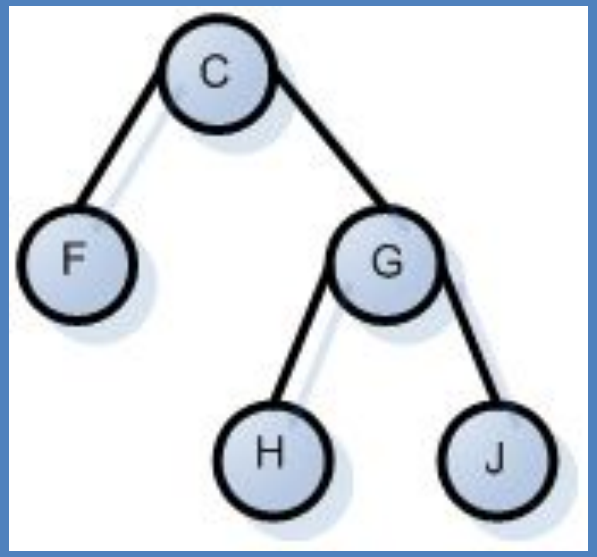
Структура элемента таблицы

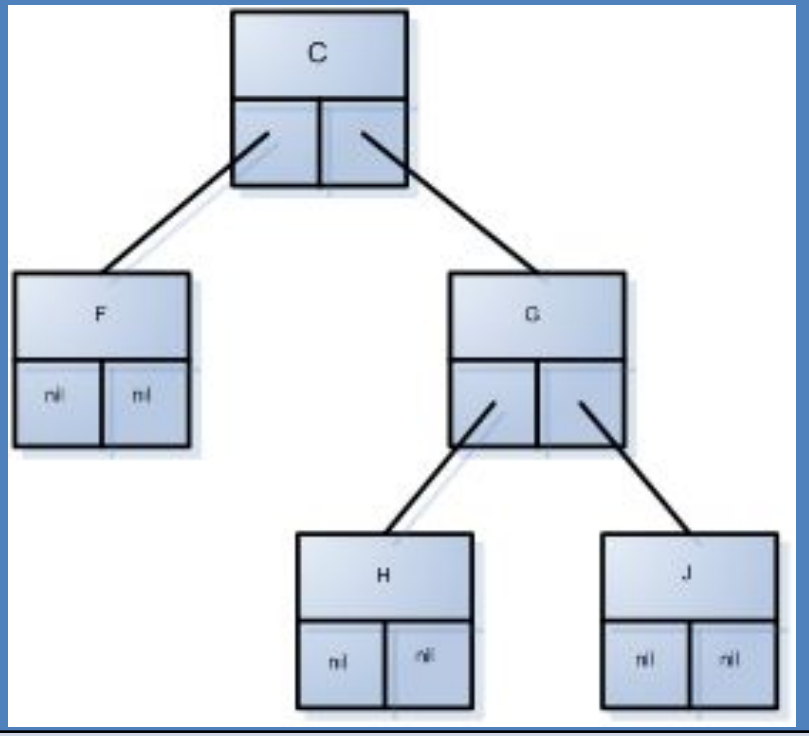
```
struct TNode{
  int Left;     //левое поддерево
  char Nod;  //метка
  int Right;   //правое поддерево
End;
```

Реализация структуры хранения дерева а таблице

```
struct Ttree{
  TNode tabl[MaxLen];      //таблица сыновей
  Tifo info[MaxLen];          //массив значений узлов
  Int Root;              // индекс корня дерева
};
Ttree T;              // дерево
```

Способы реализации деревьев: связанное размещение узлов дерева в динамической памяти





На рисунке представлено бинарное дерево и модель его реализации на основе связанного размещения узлов в динамической памяти. Узел бинарного дерева может иметь не более двух поддеревьев: левое и правое.

Реализация структуры узла динамически размещаемого узла дерева и самого дерева

```
struct Tnode{
  Tdata data;                 //поле для данных
  Tnode  *leftTree;           //ссылка на левого сына
   Tnode  *rightTree;         //ссылка на правого брата
}
 Tnode *root;          //указатель на корень дерева
```

При таком подходе к реализации дерева, каждый узел создается как отдельный динамический объект. Связи осуществляются на основе указателей, родительский узел содержит поля, которые являются указателями на дочерние узлы.

При включении нового узла в дерево, создается динамический объект (узел), определяется родительский узел для нового узла и в поле указателя родителя записывается адрес размещения нового дочернего узла.

Такой подход представления дерева в оперативной памяти наиболее удобен при выполнении вставки новых узлов в дерево и удалении узлов, так как надо только переключить указатели.

Способы реализации деревьев: связанное размещение узлов со ссылкой *на сыновей и родителя*

В некоторых алгоритмах удобно в узле хранить еще и указатель на родителя узла. это упрощает выполнение отдельных операций на дереве. Тогда в узле три поля являются указателями.

Пример. Программная реализация структуры узла дерева, содержащего ссылку на узел родителя и самого дерева

```
struct Tnode{
 Tdata data;              //поле для данных
 Tnode  *parentNode;      //ссылка на родительский узел
  Tnode  *leftTree;        //ссылка на левое поддерево
   Tnode  *rightTree; //ссылка на правое поддерево
}
 Tnode *root;      //указатель на корень дерева
```

Алгоритмы обхода К– арного дерева

К-арное дерево типа Т это:

- Пустая структура или
- Состоит из 1-го узла – корня ***n*** типа Т или
- Состоит из узла - корня, с которым связано конечное число древовидных структур типа Т с корнями (n1,n2,…nk), называемых поддеревьями.

Обход дерева *– это алгоритм, обеспечивающий посещение каждого узла дерева с целью выполнить операцию с данными узла.*

Методы обхода дерева

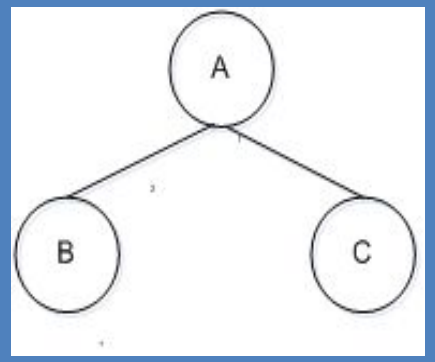
Требование. Обход дерева возможен, если дерево упорядочено.

- ***обход методом в глубину*** - это посещение узлов по поддеревьям (по ветвям);
- ***обход в методом ширину -*** это обход по уровням (обход сыновей).

Схемы обхода дерева методом в глубину

Существует три алгоритма обхода в глубину: ***прямой, обратный, симметричный***.

Рассмотрим схемы алгоритмов обхода для дерева, представленного на рисунке. Дерево с корнем A имеет два поддерева – левое B и правое C.



Прямой обход просматривает узлы в следующем порядке:

посетить корень, ***обход*** левого поддерева, ***обход*** правого поддерева, т.е. **A**BC.

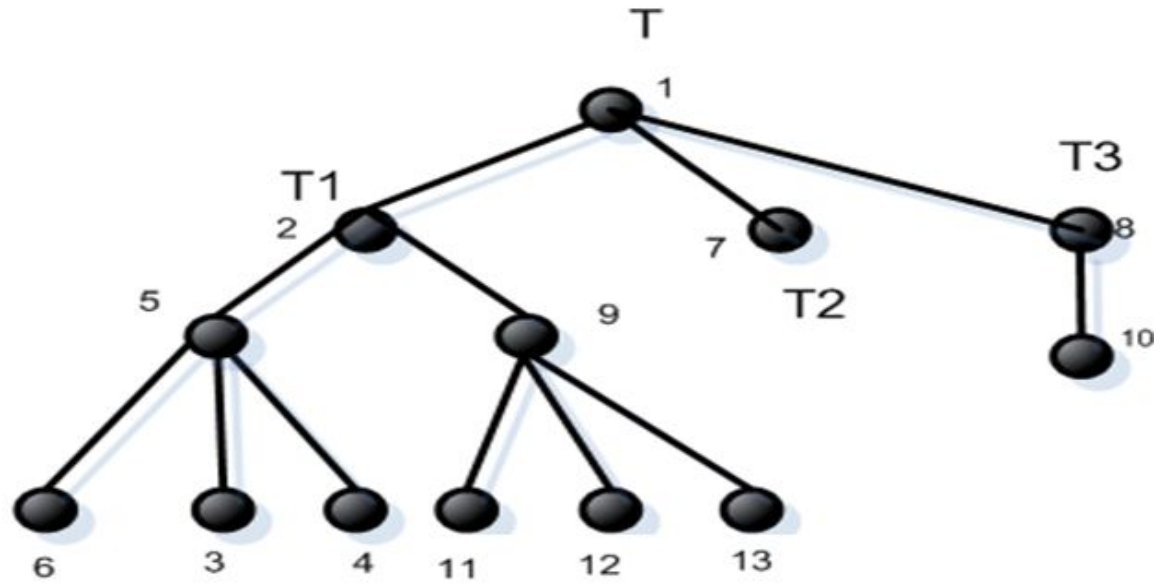
Симметричный обход предусматривает посещение узлов в следующем порядке: ***обход*** левого поддерева, **посетить корень**, ***обход*** правого поддерева, т.е. B**A**C.

Обратный обход предусматривает посещение узлов в следующем порядке: ***обход*** левого поддерево, ***обход*** правого поддерева, **посетить корень**, т.е. BC**A**.

Алгоритм обхода К - арного дерева в прямом порядке

1. **Посетить** корень дерева (узел n).
2. Обойти левое поддерева Т1 дерева Т в прямом порядке (т.е. этим же алгоритмом).
3. После обхода Т1 последовательно выполнить обход в ***прямом порядке***

поддеревьев Т2, Т3,……Тк.

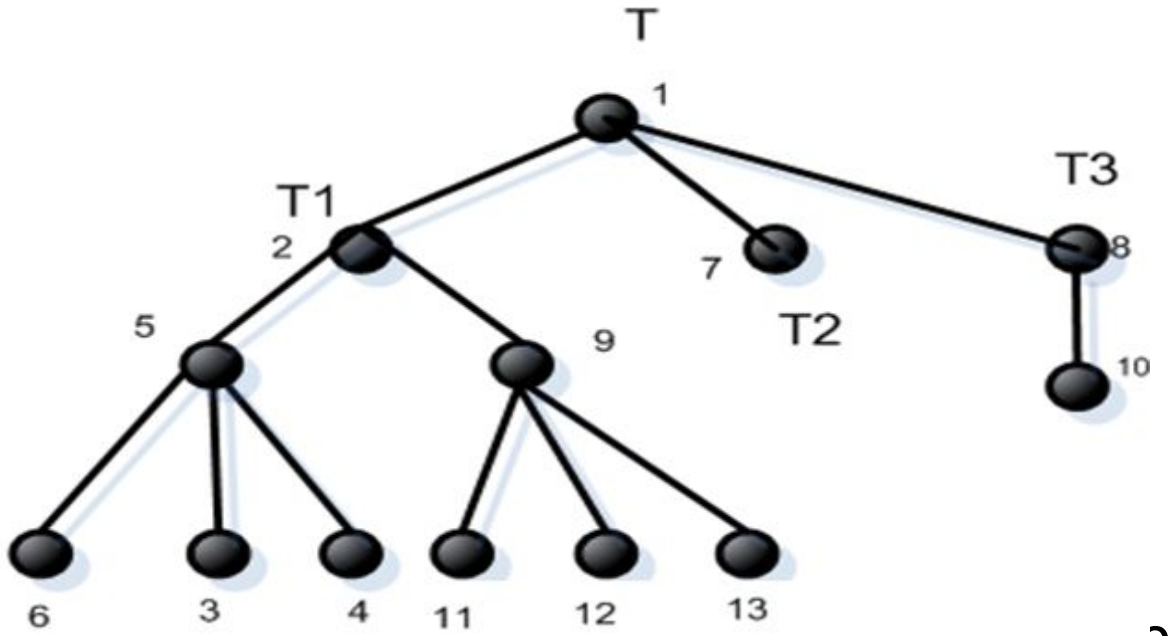


Список меток в порядке обхода: 1, 2, 5, 6, 3, 4, 9, 11, 12, 13, 7, 8, 10

Алгоритм обхода непустого К - арного дерева в симметричном порядке

1. Сначала обходим левое поддерево Т1 дерева Т алгоритмом симметричного обхода.
2. **Посетить** корень дерева Т.
3. После посещения корня осуществляется последовательный обход всех остальных поддеревьев дерева Т: Т2, Т3,……Тк в симметричном порядке.

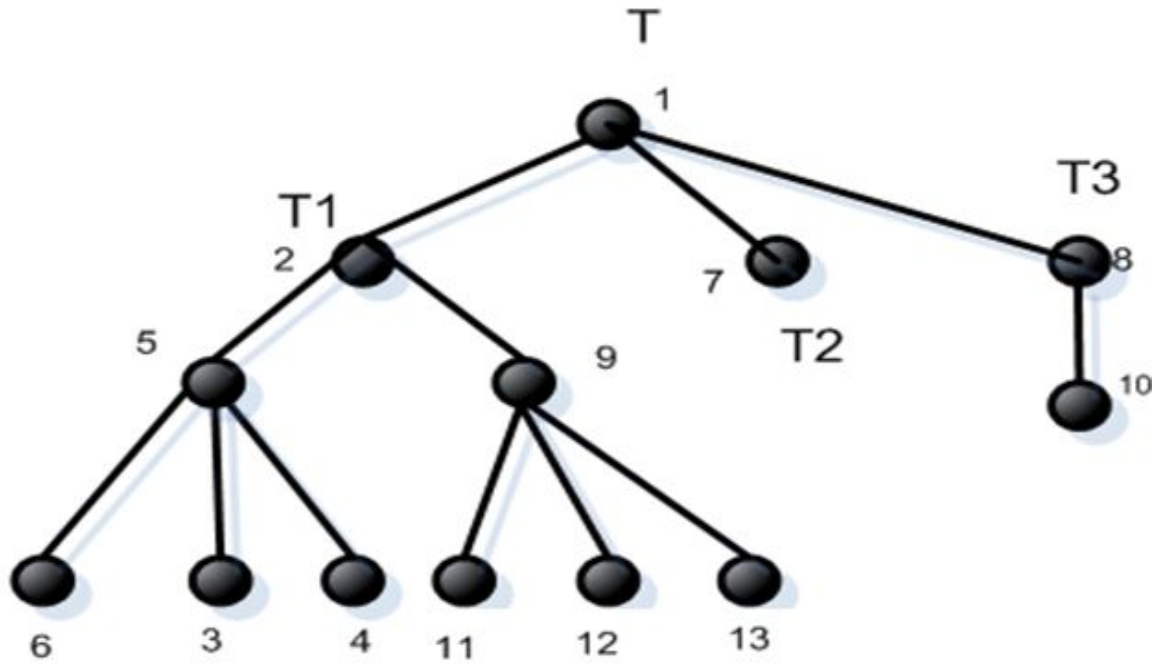
.



Список меток в порядке симметричного обхода: 6, 5, 3, 4, 2, 11, 9, 12, 13, 1, 7, 10, 8

Алгоритм обхода непустого К - арного дерева в обратном порядке

1. Сначала обходим левое поддерево Т1 дерева Т алгоритмом обратного обхода.
2. Затем осуществляется последовательный обход всех остальных поддеревьев дерева Т: Т2, Т3,…Тк в обратном порядке
3. **Посетить** корень дерева Т.



Список меток обхода в обратном порядке: 6, 3, 4, 5, 11, 12, 13, 9, 2, 7, 10, 8, 1

Выводы по алгоритмам

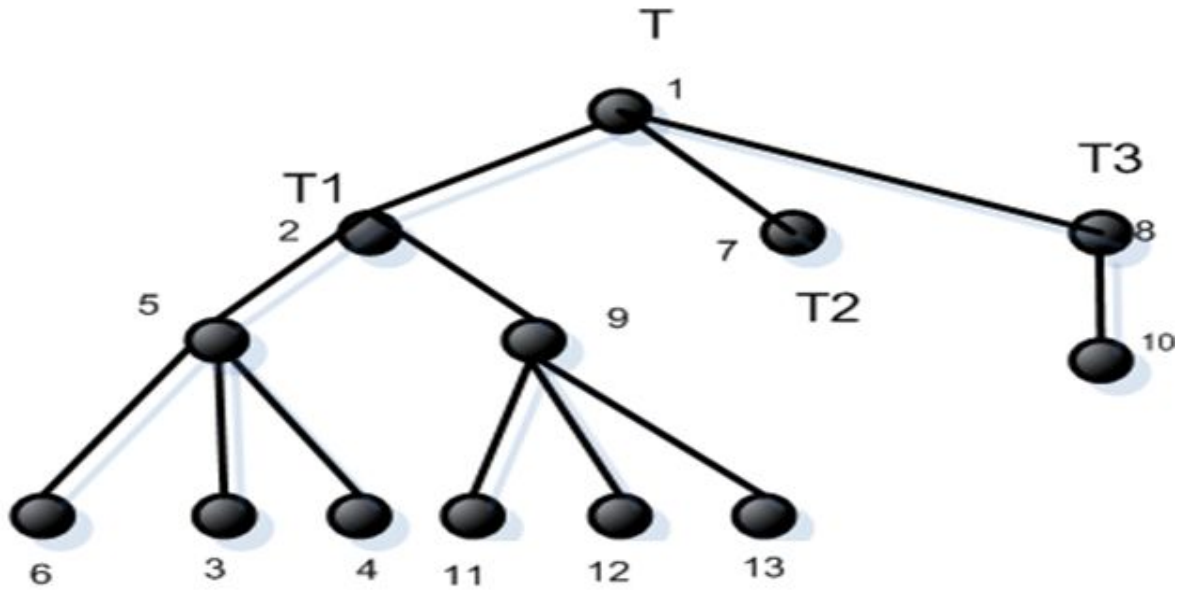
Так как дерево является рекурсивно определяемой структурой, то алгоритмы обхода методом в глубину проще реализовать рекурсивно, хотя не рекурсивная реализация тоже приемлема.

.

Алгоритм обхода методом в ширину

Обход выполняется сверху вниз по уровням.

Для сохранения сыновей узла используется дополнительная структура – очередь.



Список меток при обходе дерева в ширину: 1, 2, 7, 8, 5, 9, 10, 6, 3, 4, 11, 12, 13

!!!!!!!!Демонстрация с отдельного документа

Абстрактный тип данных – дерево

И определим в нем данные (дерево) и операции над деревом

АТД Tree

Данные

// TNode – тип узла

//TLabel - Тип метки элемента в дереве

Узел с (n), являющийся корнем дерева.

Список из k узлов (n1, n2, ……nk).

Операции

//Создание дерева из К узлов.

CreateT(T);

//Поиск левого сына узла n дерева Т, возвращает метку узла

LeftMostChild(n)

//Поиск правого брата узла n дерева Т, возвращает метку узла

RightBrather(n);

//Поиск родителя узла n дерева Т. Возвращает метку узла

Parent(n);

//вернуть Корень дерева Т

Root(T);

//проверка не пусто ли дерево

isEmpty(T)

//обход в глубину в прямом порядке дерева Т

Алгоритм обхода k – арного дерева рекурсивно

Если дерево является пустым, то при обходе в список посещенных вершин заносится пустая запись.

Если дерево состоит из одного узла, то в список заносится сам узел (или его метка).

Если T – дерево с корнем n и k поддеревьями T1, T2, ….Tk с корнями n1 n2 n3 …..nk, то

Или Алгоритм прямого обхода

Или Алгоритм симметричного обхода

Или Алгоритм обратного обхода

Алгоритм обхода в прямом порядке на псевдокоде

1) Посетить корень – n (занести в список посещенных)
2) Затем обход левого поддерево T1 таким же алгоритмом – прямого обхода
3) Далее последовательно поддеревья T2 T3 …Tk

```
Preorder(T n) {
   Visit(Label(n));            //посетить: алгоритм обработки данных
   T c←LeftMostChild(n);
   while(c )
    do
      Preorder(c);
      c ← RightBrather(n)
   od
}
```

Алгоритм обхода в симметричном порядке на псевдокоде

1) Обход поддерева T1 в симметричном порядке
2) Посетить корень n
3) Далее последовательно в симметричном порядке обойти к поддеревьев T1 T2 T3 …Tk

```
Inorder(T n){
    T c<-LeftMostChild(n);
    while(c)
      do
        Inorder(c);
        Visit(Label(c));        //посетить
        c<-RightBrather(c)
    od
}
```

Алгоритм обхода в обратном порядке на псевдокоде

1) Посещаем все узлы поддерева T1 в обратном порядке
2) Далее последовательно в симметричном порядке обходим поддеревья T2 T3 …Tk
3) Затем посещаем корень n

```
Postorder(T n)
 T c<-LeftMostChild(n);
     while(c)
 do
          Postorder(c);
         c=RightBrather(c) ;
         Visit(Label(c));        //посетить
 od
```

Алгоритм обхода дерева методом в ширину

Пусть алгоритм должен сформировать список меток узлов в порядке их посещения.

1. Корневой узел помещается в очередь.
2. Узел извлекается из очереди и включается в список посещенных узлов (или просто выводится его метка, указывая на то что узел посещен).
3. В очередь помещаются его сыновья.
4. Алгоритм продолжается с пункта 2 пока очередь не станет пустой.