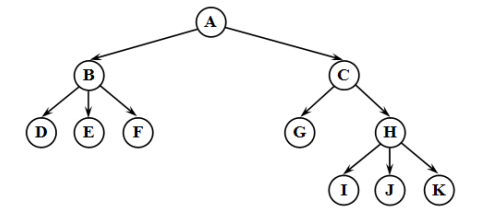
**Дерево как структура данных**

Какую выгоду можно извлечь из такой структуры данных, как дерево? В этой статье мы расскажем о данных в виде дерева, рассмотрим основные определения, которые следует знать, а также узнаем, как и зачем используется дерево в **программировании**.

Спойлер: бинарные деревья часто применяют для поиска информации в базах данных, для сортировки данных, для проведения вычислений, для кодирования и в других случаях. Но давайте обо всем по порядку.

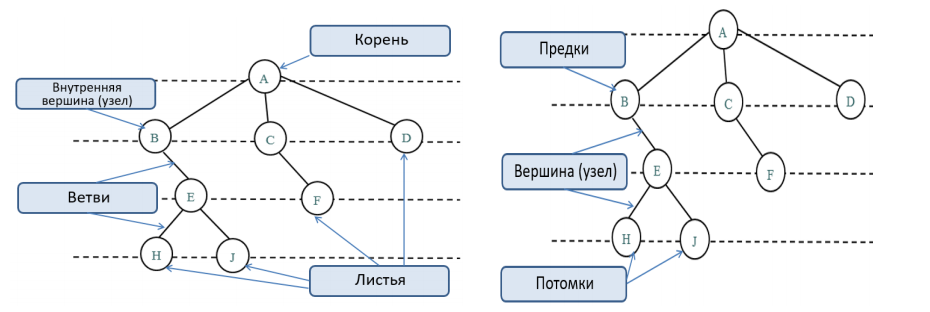
**Основные термины**

**Дерево**— это, по сути, один из частных случаев графа. Древовидная модель может быть весьма эффективна в случае представления динамических данных, особенно тогда, когда у разработчика стоит цель быстрого поиска информации, в тех же базах данных, к примеру. Еще **древом** называют структуру данных, которая представляет собой совокупность элементов, а также отношений между этими элементами, что вместе образует **иерархическую древовидную структуру**.



Каждый элемент — это вершина или **узел** дерева. Узлы, соединенные направленными дугами, называются **ветвями**. Начальный узел — это **корень** дерева (корневой узел). **Листья**— это узлы, в которые входит 1 ветвь, причем не выходит ни одной.

Общую терминологию можно посмотреть на левой и правой части картинки ниже:



Какие **свойства** есть у каждого древа:

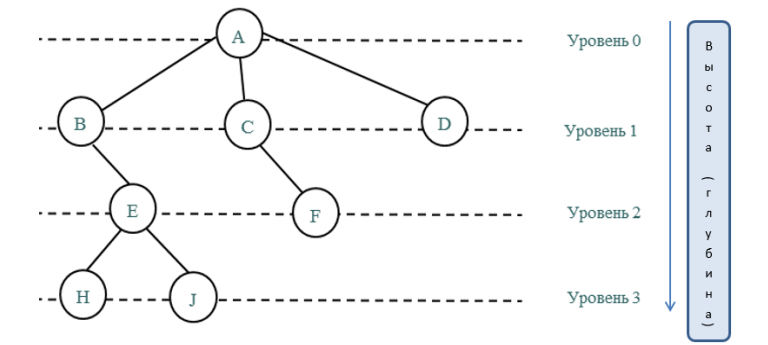
— существует узел, в который не входит ни одна ветвь;

— в каждый узел, кроме корневого узла, входит 1 ветвь.

На практике деревья нередко применяют, изображая различные иерархии. Очень популярны, к примеру, генеалогические древа — они хорошо известны. Все узлы с ветвями, исходящими из единой общей вершины, являются потомками, а сама вершина называется предком (родительским узлом). Корневой узел не имеет предков, а листья не имеют потомков.

Также у дерева есть **высота (глубина)**. Она определяется числом уровней, на которых располагаются узлы дерева. Глубина пустого древа равняется нулю, а если  речь идет о дереве из одного корня, тогда единице. В данном случае на нулевом уровне может быть лишь одна вершина – корень, на 1-м – потомки корня, на 2-м – потомки потомков корня и т. д.

Ниже изображен графический вывод древа с 4-мя уровнями (дерево имеет глубину, равную четырем):



Следующий термин, который стоит рассмотреть, — это **поддерево**. Поддеревом называют часть древообразной структуры, которую можно представить в виде отдельного дерева.

Идем дальше. Древо может быть **упорядоченным**— в данном случае ветви, которые исходят из каждого узла, упорядочены по некоторому критерию.

**Степень вершины**в древе — это число ветвей (дуг), выходящих из этой вершины. Степень равняется максимальной степени вершины, которая входит в дерево. В этом случае листьями будут узлы, имеющие нулевую степень. По величине степени деревья бывают:

— двоичные (степень не больше двух);

— сильноветвящиеся (степень больше двух).

Деревья — это **рекурсивные структуры**, ведь каждое поддерево тоже является деревом. Каждый элемент такой рекурсивной структуры является или пустой структурой, или компонентом, с которым связано конечное количество поддеревьев.

Когда мы говорим о рекурсивных структурах, то действия с ними удобнее описывать посредством рекурсивных алгоритмов.

**Обход древа**

Чтобы выполнить конкретную операцию над всеми вершинами, надо все эти узлы просмотреть.  Данную задачу называют **обходом дерева**. То есть обход представляет собой упорядоченную последовательность узлов, в которой каждый узел встречается лишь один раз.

В процессе обхода все узлы должны посещаться в некотором, заранее определенном порядке. Есть ряд способов обхода, вот три основные:

— прямой (префиксный, preorder);

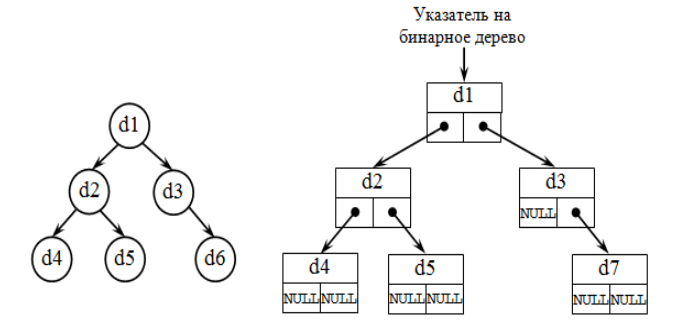
— симметричный (инфиксный, inorder);

— обратный (постфиксный, postorder).

Существует много древовидных структур данных: двоичные (бинарные), красно-черные, В-деревья, матричные, смешанные и пр. Поговорим о бинарных.

**Бинарные (двоичные) деревья**

Бинарные имеют степень не более двух. То есть двоичным древом можно назвать динамическую структуру данных, где каждый узел имеет не большое 2-х потомков. В результате двоичное дерево состоит из элементов, где каждый из элементов содержит информационное поле, а также не больше 2-х ссылок на различные поддеревья. На каждый элемент древа есть только одна ссылка.



У бинарного древа каждый текущий узел — это структура, которая состоит из 4-х видов полей. Какие это поля:

— информационное (ключ вершины);

— служебное (включена вспомогательная информация, однако таких полей может быть несколько, а может и не быть вовсе);

— указатель на правое поддерево;

— указатель на левое поддерево.

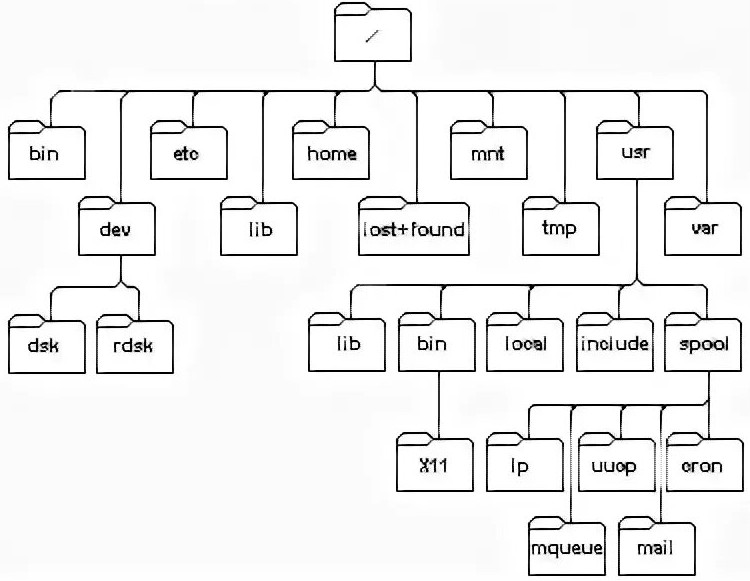
Самый удобный вид бинарного древа — бинарное дерево поиска.

**Что значит древо в контексте программирования?**

Мы можем долго рассуждать о математическом определении древа, но это вряд ли поможет понять, какие именно выгоды можно извлечь из древовидной структуры данных. Тут важно отметить, что **древо является способом организации данных в форме иерархической структуры**.

В каких случаях древовидные структуры могут быть полезны при программировании:

1. Когда данная иерархия **существует в предметной области** разрабатываемой программы. К примеру, программа должна обрабатывать генеалогическое древо либо работать со структурой каталогов. В таких ситуациях иногда есть смысл **сохранять** между объектами программы **существующие** иерархические отношения. В качестве примера можно вывести древо каталогов операционной системы UNIX:



* Когда между объектами, которые обрабатывает программа, отношения иерархии не заданы явно, но их **можно задать**, что сделает обработку данных удобнее. Как тут не вспомнить разработку парсеров либо трансляторов, где весьма полезным может быть *древо синтаксического разбора?*
* А сейчас очевидная вещь: поиск объектов более **эффективен**, когда объекты упорядочены, будь то те же базы данных. К примеру, поиск значения в неструктурированном наборе из тысячи элементов потребует до тысячи операций, тогда как в упорядоченном наборе может хватить всего дюжины. Вывод прост: раз **иерархия — эффективный способ упорядочивания объектов**, почему же не использовать древовидную иерархию для **ускорения поиска** узлов со значениями? Так и происходит: если вспомнить бинарные деревья, то на практике их можно применять в следующих целях:

— поиск данных в базах данных (специально построенных деревьях);

— сортировка и вывод данных;

— вычисления арифметических выражений;

— кодирование по методу Хаффмана и пр.

*Источники:*