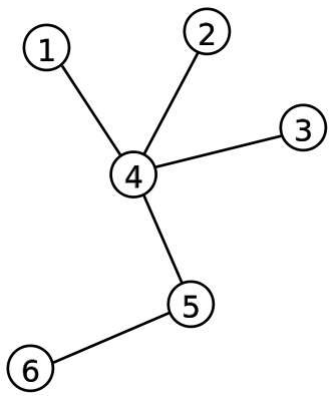
树 (图论)

在图论中，**树**（英语：Tree）是一种无向图（undirected graph），其中任意两个顶点间存在唯一一条路径。或者说，只要没有回路的连通图就是树。**森林**是指互相不交并树的集合。树图广泛应用于计算机科学的数据结构中，比如二叉查找树，堆，Trie树以及数据压缩中的霍夫曼树等等。

树



包括6个顶点，5条边的树

顶点	v
边	$v - 1$
色数	2

目录

定义

如果一个无向简单图G满足以下相互等价的条件之一，那么G是一棵**树**：

- G是没有回路的连通图。
- G没有回路，但是在G内添加任意一条边，就会形成一个回路。
- G是连通的，但是如果去掉任意一条边，就不再连通。
- G是连通的，并且3顶点的完全图K_3不是G的子图。
- G内的任意两个顶点能被唯一路径所连通。

如果无向简单图G有有限个顶点（设为n个顶点），那么G是一棵**树**还等价于：

- G是连通的，有$n-1$条边，并且G没有简单回路。

如果一个无向简单图G中没有简单回路，那么G是**森林**。

性质

一棵树中每两个点之间都有且只有一条路径（指没有重复边的路径）。一颗有N个点的树有N-1条边，也就是连接N个点所需要的最少边数。所以如果去掉树中的一条边，树就会不连通。

如果在一棵树中加入任意的一条边，就会得到有且只有一个环的图。这是因为这条边连接的两个点（或是一个点）中有且只有一条路径，这条路径和新加的边连在一起就是一个环。如果把一个连通图中的多余边全部删除，所构成的树叫做这个图的**生成树**。

如果要在树中加入一个点，就要加入一条这个点和原有的点相连的边。这条边不会给这棵树增加一

个环或者多余的路径。所以每次这样加入一个点，就可以构成一棵树。

一棵树既可以是有向的也可以是无向的。显然，树是连通图，但不会是双连通图（对于无向图）或者强连通图（对于有向图）。树可以算是稀疏图。

显然树中也没有自环和重复边。

有根树

在一棵树中可以指定一个特殊的节点：**根**。一个有根的树叫做**有根树**。

有根树中的节点可以根据到根的距离分**层**。一颗有根树的层数叫做这棵树的**高度**。节点最多的那一层的节点数叫做这棵树的**宽度**。对于有根树，每条边都有一个特殊的方向：指向根节点的方向，或者说上一层的方向（或者相反的，指向叶节点的方向，下一层的方向）。一条边的两个端点中，靠近根的那个节点叫做另一个节点的**父节点**（也叫**父亲**、**双亲**、**双亲节点**），相反的，距离根比较远的那个节点叫做另一个节点的**子节点**（也可以叫**孩子**，**儿子**，**子女**等）。父亲方向的所有节点都叫做这个节点的**祖先**，儿子方向的所有节点都叫做这个节点的**子孙**。没有子节点的子节点叫做**叶节点**（或者**叶子节点**）。由于到根的路径只有一条，根节点以外的节点的父节点永远只有一个，祖先就是这个点到根的路径上的所有节点（包括根，不包括这个节点本身）。另外，以一个节点为根的树是指包括这个节点和其所有子孙，并以这个节点为根的树。由于一般不需要这以外的子树，每一个节点也可以对应到一个以其为根的树，一个节点的子树通常也是指以这个节点的子节点为根的树。

如果一颗有根树每个节点的子树最多有n个，同时每个节点在其父节点中都有固定的可能可以留空的位置，这棵树叫做**n叉树**。其中每个节点都可以有两个固定位置的子树的有根树叫做**二叉树**，二叉树中每个节点的两个子树分别叫做**左子树**和**右子树**，由于位置固定，没有左子树的时候也是可以有右子树的。而“多叉树”通常并不指n为任意值的n叉树，只是在和n叉树作比较的时候表示普通的有根树。

对于随机的树，高度的平均复杂度是O(logn)，但是没有限制而且不随机的树高度也可以达到O(n)，也就是除了叶节点都只有一个子树，或者常数个分支的情况。所以树作为数据结构时通常需要另外进行平衡。

存储

对于普通的树，可以像图一样为每一个点存储一个边表（通常按顺序存和每一个点的关系的叫做邻接矩阵，存具体的边的叫做邻接表），或者直接存储所有边的边表等。由于树是稀疏图，所以一般不用邻接矩阵存储。对于有根树，如果用为每一个点储存一个边表的方法，由于每一棵树都只有一个父节点，所以通常指向父节点的边不存在这个表中。同时如果子节点是没有顺序的，也是因为一个节点的子节点不会同时是其他节点的子节点，也可以把子节点直接当成存边的链表的节点，这时候每个节点只需要储存两个指针，所以这种存储方法有时候也会被叫做多叉树转二叉树。

对于子节点是有顺序的有根树，每条边都可以以固定的位置分别储存。对于完全二叉树甚至能直接用一个数组访问所有节点，不另外储存边的信息。有的树可以被设计成固定的从根节点开始访问，这时候可以不储存父节点。同样的，有的树也可以省略子节点，例如并查集。

树的遍历

对于一般的树，可以用和普通的图一样的方法遍历，比如深度优先搜索和宽度优先搜索。如果和树的每个节点相邻的点有固定的顺序，深度优先搜索可以不储存当前点以外的任何信息，而且不用判重。而在有根树中更方便，所以有根树中很少使用宽度优先搜索。

对于有根树的从根开始的深度优先搜索遍历，有三种特定的顺序：

前序遍历：先访问根节点，然后再访问所有的子树；

后序遍历：先访问子树，然后再访问根节点；
中序遍历：二叉树专用，先访问左子树，然后是根节点，最后是右子树。

注意对于每一种遍历，事实上都得先访问根节点，这里的遍历顺序是指处理节点中的数据的顺序。已知中序遍历和任一其他遍历的情况下，可以还原一个二叉树。一个直观的方法是按前序或者反转的后序插入一个按中序排序的搜索树。已知前序和中序也可以还原一棵树，但是不能知道二叉树中一个节点唯一的子树是在左边还是右边。

事实上也可以把左右的顺序反过来。这些由根开始的遍历方法也适用于特定的一个子树。

森林

森林比树少一个条件，指没有回路的图。也就是说，森林可能是不连通的，边数可能比树还少，就是说小于顶点数。

森林也可以看成是好多棵互不相连的非空的树，只有一棵树也可以算是森林。不过森林不一定是一棵树。

森林也可以是有根的，这时候森林中的每一棵树都有一个根。

一棵树去掉若干条边也会成为森林，这时候可以看成一棵树分成了好多棵树。森林中的两棵树之间加一条边，也可以把这两棵树合在一起，最终合成一棵树。

很多地方用森林，都是用来表示很多棵树，包括作为逻辑结构、数据结构的时候等。有一种重要的数据结构**并查集**就是一个有根的森林，可以很快的判断两个元素是不是属于同一个互相独立的集合，以及合并两个集合等。

逻辑结构

树也通常会用来表示逻辑结构，例如搜索树。表示逻辑结构的树一般是有根树。这种结构类似于有拓扑序的图，每个节点是其之前的节点的后继、分支、子节点等。树的结构中，每个节点之前的节点是唯一的（就是说有唯一的前驱、上层容器、父节点等），另外每一个节点及其后面的部分也都是一棵树。

作为数据结构

树也是一类重要的数据结构，同时也有逻辑结构的性质，通常也是有根树。主要有搜索树和堆两种，前者的内容是按中序遍历的顺序排序的，后者每个节点的关键字都比它的子节点大（或者小）。复杂度一般在树的高度，也就是O(nlogn)以内。

搜索树可以快速的查找有序的内容或者新内容在已有内容中的位置，也可以进行一些和按这个顺序的范围有关的统计。

堆 (数据结构)是一种优先队列，比搜索树功能少，通常只能很方便的求堆中关键字最小（最大）的数据，不能查找。（当然有的时候求次小和第三小也是很方便的）

很多这类数据结构会给每个点或者边加上一些别的参数。有些数据结构还会破坏本来的树的结构，但是基本还是用的树的模式，一般还是叫做“树”。

树的类型

- 自由树
- 有根树

    - 有向树
- 二叉树
    - 满二叉树
    - 完全二叉树
- Positional tree
- 空树

- 本页面最后修订于2017年7月30日 (星期日) 14:28。