X X 大 学

综 合 论 文 训 练

题目：Tree

系　　别：计算机科学与技术系

姓　　名：Qian Xin

指导教师：导师姓名

2021 年 6 月 22 日

中文摘要

在信息科学研究中，树作为一种特定链式存储结构，在很多研究及算法领域都发挥了重大作用，对树的研究，对领域相关算法研究具有基础意义，但是关于树的研究，尚不具有综述性质的文献。在本文中，论文试图对信息科学中“树”这一特别的数据结构的相关理论及应用成果进行综述。论文将树划分为三个分类：搜索树，数据库中的树，机器学习中的树，旨在提供一份简明扼要的研究指南。论文使用了大量的官方经典文献来对文章进行解释并支撑论断。论文尝试使用这种对树相关应用的统一理论结合数学分析对树进行诠释。

关键词：信息科学；树；搜索树；二叉树；自平衡树；数据库；八叉树；机器学习；决策树

目　　录

第 1 章　引言

“树”是信息科学中经常处理的一种数据结构。本文将围绕树在信息科学中的应用展开，综述“树”这一特殊的数据结构的地理意义和实用目的。

“树”大致可分为三种用途，分别是搜索树（排序树，存储树）、数据库中的树、机器学习中的树，除这些树外还有线程树等。搜索树（存储树）是最常见的用途，也是数据库等各个方面的常用基石；数据库中的树是基于信息存储与压缩方面的特化；机器学习中的树是用于监督分类的高性能机器学习工具。

本项目中使用的代码均在 GitHub[?] 上开源。

第 2 章　搜索树

2.1　指针与地址

树中不可避免地使用指针。Donald Knuth 曾指出：“赋值语句和指针变量是‘最宝贵的财富’”[?]。指针是许多编程语言中用来存储内存地址的对象。这可以是位于计算机内存中的另一个地址值的值，或者在某些情况下，内存映射的计算机硬件的值。指针引用内存中的一个位置，而获取存储在该位置的值称为取消引用 (dereference) 指针[?]。

一个数据原语（data primitive）（或者仅仅是原语）是任何可以通过一次内存访问从计算机内存读取或写入计算机内存的数据（例如，一个字节和一个字都是原语）。

数据聚合（data aggregate）（或仅聚合）是一组在内存中逻辑上连续的基本体，它们被视为一个数据集（例如，聚合可以是 3 个逻辑上连续的字节，其值表示空间中某个点的 3 个坐标）。当一个聚合完全由相同类型的基元组成时，该聚合可以称为数组；在某种意义上，多字节字原语是一个字节数组，有些程序就是这样使用字的。

在这些定义的上下文中，字节是最小的原语；每个内存地址指定一个不同的字节。数据的初始字节的内存地址被认为是整个数据的内存地址（或基内存地址）。一个内存指针（或者仅仅是指针）是一个原语，它的值被用作内存地址；这意味着指针指向内存地址。也可以说，当指针的值是数据的内存地址时，指针指向 [内存中的] 数据。

指针是在大多数现代体系结构提供的寻址功能之上的一个非常精简的抽象。在最简单的方案中，地址或数字索引被分配给系统中的每个内存单元，其中该单元通常是一个字节或一个字（取决于体系结构是字节可寻址还是字可寻址），有效地将所有内存转换为一个非常大的数组。然后，系统还将提供一个操作来检索存储在给定地址的内存单元中的值（通常使用机器的通用寄存器）。

在通常情况下，指针的大小足以容纳比系统内存单元更多的地址。这就引入了一种可能性，即程序可能会试图访问一个地址，而该地址不对应于任何内存单元，这可能是因为没有安装足够的内存（即超出可用内存的范围），也可能是因为体系结构不支持这样的地址。在某些平台（如 Intel x86 体系结构）中，第一种情

况可能称为分段故障（segfault）。第二种情况在 AMD64 的当前实现中是可能的，其中指针是 64 位长，地址只扩展到 48 位。指针必须符合某些规则（规范地址），因此如果非规范指针被取消引用 (dereference)，处理器将引发一般性保护错误。

数据结构通常基于计算机在其内存中的任何位置获取和存储数据的能力，这些数据结构由一个指针（一个表示内存地址的位字符串）指定，它本身可以存储在内存中并由程序操作。因此，数组和记录数据结构基于通过算术运算计算数据项的地址，而链接数据结构基于在结构本身中存储数据项的地址。[?]

2.2 树与链表

定义链表即定义一个数据聚合结构，应首先定义另一个数据聚合结构“结点”从而使用对应的指针，然后通过链表的头尾指针将结点串在一起。在链表结点中，应使用指针存储前后链表结点的基地址。

1857 年，英国数学家 Arthur Cayley 创造了“树”[?]这个名词。树的名词来源于图论中的直接应用，在图论中，树被定义为一个满足任意两个顶点被“一条确定的边”相连的无向图，或等价于一个连通的无环无向图。而“森林”定义为一个满足两个顶点被“至多一条确定的边”相连的无向图，或等价于一个无环无向图，或等价于树的不相交并集。数据结构中的树直接来源于图论。

因为树是无环无向图的原因，树常常比图数据结构更容易考虑。

在图论中，路径图 (path graph)[?]，又称线性图 (linear graph)，由在一条线上的 n 个顶点组成，其顶点 V_1 与 V_n 被属于 $i = 1, 2, ..., n-1$ 的边 $\{v_i, v_{i+1}\}$ 相连。这与链表十分相似，但链表的结点具有数据，且至多有两个指针（线性图是一种树，因此是无向图）。

而线性列表实际上也是一种数据结构上的树。但数据结构中的树的父子结点之间的边的方向是“不固定的”，可以有特定个指针，而数学中的树，一定是无向图。因此，可以说链表是一种特殊的“一叉树”。

2.3 二叉树与二叉搜索树

数据结构中的树是著名的非线性数据结构。树可以用基本数据结构（链表，数组）的形式实现，因此，我们不会认为链表是一棵树。当我们考虑多路树（multi-way tree）时，我们首先考虑一种最简单的多路树，但亦彰显了树与基本数据结构不同的树，即二叉树（binary tree）。而一切有根有序的多叉树，都可以等价地转

换并实现为二叉树。

二叉树可以使用集合论概念进行递归定义：

（非空）二叉树是元组 (L, S, R)，其中 L 和 R 是二叉树或空集，S 是包含根的单例集。

在数学中，元组是元素的有限有序列表（序列）。n 元组是 n 个元素的序列（或有序列表），其中 n 是非负整数。只有一个 0 元组，称为空元组。一个单例集，也称为单位集，是一个只有一个元素的集合。例如，集合 $\{null\}$ 是包含元素 $null$ 的单例。这样定义从而限定了二叉树的的左右子树。

通过二叉树的递归定义及其他树的定义推论可知，树与递归以及集合论有很明确的联系。而二叉树的二分实现，也使二叉搜索树的查找时间复杂度最好为 $O(log_2 n)$。在特殊情况下，二叉树退化为类链表形式，查找时间复杂度为 $O(n)$。

二叉搜索树（BST）[?]也称为有序或排序二叉树，是一种根二叉树，其内部节点存储的键大于节点左子树中的所有键，小于节点右子树中的所有键。二叉树是一种以有组织的方式存储数字等数据的数据结构。二进制搜索树允许二进制搜索快速查找、添加和删除数据项，并可用于实现动态集和查找表。BST 中节点的顺序意味着每次比较都会跳过剩余树的一半，因此整个查找所需的时间与树中存储的项数的二元对数成正比。这比在（未排序的）数组中按键查找项所需的线性时间要好得多，但比哈希表上相应的操作要慢。

2.4 二叉树的遍历与线索二叉树

以下是离散数学中对有根树 $T = (V, E)$ 的所有顶点和根 r 多个顺序遍历的递归定义：

1. 如果 T 中仅有一个顶点 r，r 本身构成 T 的前序，后序及中序遍历。
2. 其他情况下，设 $T_1, ...T_n$ 为 T 从左到右的子树。
   - 前序遍历：$Pre(T) = r, Pre(T_1), ..., Pre(T_n)$
   - 后序遍历：$Post(T) = Post(T_1), ..., Post(T_n), r$
   - 中序遍历 (仅 $n = 2$ 时二叉树情况存在，此时设 T 的左子树为 T_L，右子树为 T_R)：$In(T) = In(T_L), r, In(T_R)$

在 1968 年的一本教科书中，Donald Knuth 询问是否存在一种顺序遍历的非递归算法，该算法不使用堆栈，并且不修改树。这个问题的解决方案之一来自 1979 年的 J.H.Morris。在 1969 年的后续版本中，Knuth 将线程树表示法归因于 Perlis

和 Thornton。

二叉树的线程（线索）化是通过使所有右子指针（通常为空）指向节点的顺序遍历后继对象（如果存在），而所有左子指针（通常为空）指向节点的顺序遍历前继对象来实现的。

在二叉树的线索化中，可以选择任何一种顺序遍历方法。线索二叉树有助于二叉树按特定顺序进行遍历。

2.5 自平衡二叉搜索树，Red-Black 树，AVL 树

自平衡（或高度平衡）二叉搜索树是任何基于节点的二叉搜索树，它在面对任意项目的插入和删除时自动保持其高度（根下的最大级别数）较小（Donald Knuth，1988）[?]。

高度为 h 的二叉树最多可包含 $2^{h+1}-1$ 个节点。仅由根节点组成的树的高度为 0。因此，对于任何具有 n 个节点和高度 h 的树：

$$n \leqslant 2^{h+1}-1$$

对等式两边 +1，取对数后减去 1 可得:

$$h \geqslant \lceil \log_2(n+1)-1 \rceil \geqslant \lfloor \log_2 n \rfloor$$

可知 BST 的最小高度为 $\log_2(n)$

然而，当项目按排序键顺序插入时，树将退化为具有 n 个节点的链表。

自平衡二叉树通过在键插入时对树执行变换（如树旋转）来解决此问题，以保持高度与 $log_2 n$ 成比例。尽管涉及到一定的开销，但从长远来看，通过确保后期操作的快速执行，这可能是合理的。

AVL 树[?]是以两位苏联发明家乔治·阿德尔森·维尔斯基（Georgy Adelson Velsky）和叶夫根尼·兰迪斯（Evgenii Landis）的名字命名 (Adelson-Velsky and Landis) 的，他们在 1962 年的论文《信息组织的算法》（An algorithm for The organization of information）中发表了这一树。AVL 树是历史上第一个自平衡二叉树。所有 AVL 结点使用以下平衡因子 (balance factor)：

$$BF(X) := Height(RightSubTree(X)) - Height(LeftSubtree(X))$$

:= 是 = 的变体，通常在程序语言的数学中使用，表示赋值。通常 = 用以表示相等。

红黑树是一种自平衡二叉搜索树。每个节点存储一个表示“color”（“red”或

“black”）的额外位，用于确保树在插入和删除期间保持平衡。1972 年，Rudolf Bayer 发明了一种数据结构，它是 B 树的一个特殊的 4 阶情况。这些树以相同的节点数保持从根到叶的所有路径，从而创建完全平衡的树。然而，它们不是二叉搜索树。拜耳在他的论文中称之为“对称二叉 B 树”，后来它们成为流行的 2-3-4 树或 2-4 树。这种树的 B 树性质使得 2-3-4 树是红黑树的同构，这意味着它们是等价的数据结构。换句话说，对于每 2−3−4 树，至少存在一个具有相同顺序的数据元素的红黑树。此外，2−3−4 树上的插入和删除操作会导致节点扩展、拆分和合并，相当于红黑树中的颜色翻转和旋转。然而，2-3-4 树很难在大多数编程语言中实现，因为树上的操作涉及大量的特殊情况。红-黑树更容易实现，因此倾向于使用。在 1978 年的一篇论文《平衡树的二色框架》中，Leonidas J。Guibas 和 Robert Sedgewick 从对称二叉 B-树中导出了红黑树[?]。

红黑树是满足下列性质的树：

1. 每个节点不是红色就是黑色。
2. 所有的 NIL 叶子都被认为是黑色的。
3. 如果一个节点是红色的，那么它的两个子节点都是黑色的。
4. 从一个给定节点到它的任何后代 NIL 叶子的每一条路径都经过相同数量的黑色节点。

第 3 章　数据库中的树

3.1　四叉树，八叉树

四叉树，八叉树是两种 m 路树的应用，这两种树的 m 均固定，分别为 4 和 8。

八叉树是一种树型数据结构，其中每个内部节点正好有八个子节点。八叉树最常用于通过递归地将三维空间细分为八个八叉树来划分空间。而四叉树其中每个内部节点正好有四个子节点。四叉树是八叉树的二维模拟，通常用于通过递归地将二维空间细分为四个象限或区域来划分二维空间。

1. 将空间分解成适应性单元。
2. 每个单元有一个最大容量。当达到最大容量时，单元分解。
3. 树目录遵循四叉树的空间分解。

四叉树，八叉树主要用于空间数据的压缩和相关图像处理。

3.2　B 树，B+ 树，B* 树

B 树[?]是一种自平衡树数据结构，它维护已排序的数据，并允许在对数时间内进行搜索、顺序访问、插入和删除。与其他自平衡 m 路搜索树不同，B 树非常适合读写相对较大数据块的存储系统，如磁盘。

B 树是由 Rudolf Bayer 和 Edward M. McCreight 在波音研究实验室工作时，为了高效地管理大量随机访问文件的索引页而发明。B 树的索引非常庞大，从而具有大量索引页。B 的意思可能代表“波音，平衡，宽广，浓密，拜耳”($Boeing, balanced, broad, bushy, and Bayer$) 等多个意思。术语 B 树可以指特定的设计，也可以指一般的设计类别。在狭义上，B 树在其内部节点中存储键，但不需要在叶子处的记录中存储这些键。一般类包括 B+ 树和 B* 树等变体。

依据 Donald Knuth 的定义,m 阶 B 树是满足以下性质的树：

1. 每个结点最多有 m 个子树。
2. 每个非叶子结点（除了根）有至少 $\lceil m/2 \rceil$ 个子节点。
3. 如果根结点不是叶子结点，根结点有最少两个子树。
4. 一个有 k 个子树的非叶子结点包含 $k-1$ 个键。
5. 所有的叶子都出现在同一水平面上，不携带任何信息。

每个内部节点（非叶子结点与根节点）的键分隔其子树。例如，如果一个内

部节点有 3 个子树，那么它必须有 2 个键：a_1 和 a_2。最左侧子树中的所有值小于 a_1，中间子树中的所有值介于 a_1 和 a_2 之间，最右侧子树中的所有值大于 a_2。

《大型有序索引的组织与维护》中首先对 B 树进行了描述。没有一篇论文正式介绍过 B+ 树的概念。相反，维护叶节点中所有数据的概念作为一个有趣的变体被反复提出。Douglas Comer 对 B+ 树的早期调查也包括 B+ 树。Comer 指出，B+ 树用于 IBM 的 VSAM 数据访问软件，他引用了 IBM 在 1973 年发表的一篇文章。

一个 B+ 树的数据完全存储在叶子上，而不是有一部分存储在内部节点上的键。B+ 树将键的副本存储在内部节点中；键和记录存放在叶子结点中；此外，叶节点可以包括指向下一个叶节点的指针，以加速顺序访问，这通常使用链表实现。

B* 树平衡更多相邻的内部节点，以保持内部节点更密集。此变体确保非根节点至少占满 2/3 而不是 1/2。因为在 B 树中插入节点的操作最昂贵的部分是分割节点，创建 B*-树是为了尽可能推迟拆分操作。为了保持这一点，与其在节点满时立即拆分节点，不如将其键与旁边的节点共享。这种溢出操作的成本比拆分要低，因为它只需要在现有节点之间移动键，而不需要为新节点分配内存。对于插入，首先检查节点中是否有一些可用空间，如果有，则只需将新键插入节点。但是，如果节点已满（它有 $m-1$ 个键，其中 m 是树作为从一个节点指向子树的指针的最大数目)，需要检查是否存在正确的兄弟，以及是否有一些可用空间。如果右同胞有 $j < m-1$ 个键，然后在两个兄弟节点之间均匀地重新分配键。为此，来自当前节点的 $m-1$ 键、插入的新键、来自父节点的一个键和来自兄弟节点的 j 个键被视为 $m+j+1$ 键的有序数组。数组被分成两半，这样 $\lfloor (m+j+1)/2 \rfloor$ 最低的键保留在当前节点中，下一个（中间）键插入父节点，其余的键插入到右兄弟节点。（新插入的键可能会在三个位置中的任何一个结束。）右兄弟节点已满，而左兄弟节点未满的情况类似。当两个兄弟节点都已满，然后将两个节点（当前节点和兄弟节点）拆分为三个节点，并在树上向上移动一个键到父节点。如果父节点已满，则进行递归的溢出和拆分操作。

B 树是一种 m 路树的应用，满足下列最佳情况 h_{min} 和最坏情况 h_{max} 高度：

$$h_{min} = \lceil \log_m(n+1) \rceil - 1$$

$$h_{max} = \lceil \log_{\frac{m}{2}}(\frac{n+1}{2}) \rceil$$

第 4 章　机器学习中的树

4.1　决策树与 ID3

数据以以下形式记录（假设有 n 个特征）：

$$(X, Y) = (x_1, x_2, x_3, ..., x_n, Y)$$

其中 X 是一个向量，因变量 Y 是一个标量，同时是我们试图理解、分类或概括的目标变量。向量 X 由用于该任务的属性 $x_1, x_2, x_3 ... x_n$ 等组成。

假设所有输入特征都有有限的离散域，并且有一个称为“分类”的单一目标特征 (target feature)。分类域 Y 的每个元素称为类。决策树或分类树是一种树，其中每个内部（非叶）节点都用输入特征（属性）标记。来自标记有输入特征（属性）的节点的弧被标记有目标特征（分类域）Y 的每个可能值，或者弧导致不同输入特征上的从属决策节点。树的每个叶子都用一个类或类上的概率分布来标记，表示数据集已被树分类为一个特定的类，或一个特定的概率分布（如果决策树构造良好，则该概率分布会向某些类的子集倾斜）[?]。用类标记是分类树，用类上的概率分布来标记是回归树。

而分裂基于一组基于分类特征的分裂规则。这个过程以递归的方式在每个派生子集上重复，称为递归分裂。当一个节点上的子集具有目标变量的所有相同值时，或者当拆分不再为预测增加值时，递归就完成了。这种自上而下的决策树归纳过程称为 TDIDT，是贪婪算法的一个例子。不同分裂的规则使用不同算法，即使用不同的度量标准来衡量“最佳”。这些通常测量子集内目标变量的同质性。将这些度量应用于每个候选子集，并将所得值组合（平均、求和等）以提供分裂质量的度量。

其中一种算法为 ID3（迭代二分法），是 Ross Quinlan 发明的一种算法，用于从数据集生成决策树。ID3 采用以下指标（metrics）评估分裂点：

熵 $H(S)$ 是集合 S 中不确定性量的度量（即熵表征集合 S）：

$$H(S) = \sum_{y \in Y} -p(y) \log_2 p(y)$$

其中，

1. S 正在为其计算熵的当前数据集, 这在 ID3 算法的每一步都会发生变化，在对属性进行拆分的情况下会变为上一个集合的子集，在递归终止前下会变

为父集合的“同级”递归分区。

2. Y 为 S 中的分类域，即类别的集合。
3. $p(y)$ 为类 y 中元素数与集合 S 元素数的比例 (proportion)。

当 $H(S) = 0$ 时，集合 S 被完全归类，即元素属于同一类。在 ID3 中，为每个剩余属性计算熵。熵最小的属性用于在这个迭代中拆分集合 S。

信息增益 (Information Gain)$IG(A)$ 是在属性 A 上拆分集合 S 之前到之后熵差异的度量。即，在属性 A 上拆分集合 S 之后，S 中的不确定性减少的度量。

$$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) = H(S) - H(S|x)$$

其中，

1. $H(S)$ 为集合 S 的熵。
2. T 为通过属性 x 拆分集合 S 而创建的子集，因此 $S = \bigcup_{t \in T}^{t}$ 。
3. $p(t)$ 为 t 中的元素数与集合 S 中元素数的比例。
4. $H(t)$ 为子集 t 的熵。

在 ID3 中，可以为每个剩余属性计算信息增益（而不是熵）。具有最大信息增益的属性用于在该迭代中拆分集合 S。

ID3 生成的树是一颗分类树，除了 ID3 还有 CART（分类与回归树）,C4.5(ID3 的继承者) 等算法。

4.2 树的集成学习

集成学习通过构建多个决策树增加模型性能：

1. Boost: 通过训练每个新实例来强调以前错误建模的训练实例，从而逐步增强树的集成。一个典型的例子是 AdaBoost。这些可用于回归类型和分类类型问题。通常用例有 GBDT(梯度增强决策树)，XGBoost，LightGBM 等。
2. Bagging:Bagging（Bootstrap aggregated，Bootstrap 采样聚合）决策树是一种早期的集成方法，它通过重复地对训练数据进行重采样和替换，并对树进行投票以获得一致性预测，从而构建多个决策树。随机森林是一种特定类型的 Bootstrap 采样聚合。
3. 旋转森林-其中每个决策树都是通过首先对输入特征的随机子集应用主成分分析（PCA）来训练的。

4.3 随机森林

森林是一组 $n \geqslant 0$ 个不相交的树。

树转二叉树：

1. 在所有兄弟结点之间加一连线。
2. 对每个结点，除了保留与其第一个子结点的连线外，去掉该结点与其它孩子的连线。
3. 水平调整。以树的根节点为轴，将整棵树顺时针旋转一定角度，使结构层次化 (注意，第一个子节点是节点的左子节点，兄弟节点转换的子节点是节点的右子节点）。

由于这样兄弟结点是右子树的一部分，孩子结点是左子树的一部分。从而成为二叉树。

森林转二叉树：

1. 将每棵树转换为二叉树。
2. 第一个二叉树不移动。从第二棵二叉树开始，后一棵二叉树的根节点作为前一棵二叉树的根节点的右子节点，并用线连接。

随机森林或随机决策森林是一种集成学习方法，用于分类、回归和其他任务，通过在训练时构造多个决策树进行操作。对于分类任务，随机森林的输出是大多数树选择的类。对于回归任务，返回所有单个树的平均预测值。随机决策森林纠正了决策树过度拟合其训练集的习惯。随机森林通常优于决策树，但其精度低于梯度增强决策树。但是，数据特征会影响它们的性能。[?]

第一个随机决策林算法是由 Tin Kam Ho 于 1995 年创建的使用随机子空间方法，在 Ho 的公式中，这是实现 Eugene Kleinberg 提出的“随机判别”分类方法的一种方法。随机森林经常被用作“黑箱”模型，因为它们可以在广泛的数据范围内生成合理的预测，而只需要很少的配置。

第 5 章　结论

本文使用了大量的官方经典文献来对文章进行解释并支撑论断，尝试使用这种对树相关应用的统一理论结合数学分析对树进行诠释。本文提出了一些具有客观意义的公式，并对树的基本性质，数据库中的理论和应用，机器学习中的理论和应用进行了较为全面和准确的总结和提炼。

本文对前人论文中的概念部分进行了一定程度上的公理化总结，但本文依然不是一篇可以全面概括树在信息科学中应用的文章。关于树的其他方面比如线程树，以及数据库中的树，机器学习中的树，树的基本性质以及完全的公理化仍需要更全面详细的资料和佐证。