6 树

6.1 基本概念

和数学中的定义(无圈图)不太一样, 数算里的树是除了根之外所有点入度均为1的有向树, **被摆放的很整齐**.

> 同属于形式语言的数学语言和计算机语言, 前者只刻画性质, 而后者只保证可计算性. 我觉得树就是最好的例子. 根对于数学毫无意义, 但它在计算机语言中则是递归的起点
> 本质是因为数学语言的主体是推理者, 计算机语言的主体是电路

- **根** : 唯一没有入度的节点
- **子树**, **前驱**
- **有向有序树** : 有序是指子树间有次序
  - 度为2的有序树 $\neq$ 二叉树 : 前者的单边节点的子树不分左右,都是第一子树
- **度数** : 指出度, 即子树个数
- **森林** : 0或多棵不相交的树的集合

树形结构的表示法

- 树形表示法
  画树状图
- 形式语言表示法
  定义节点集合和关系集合
- 文氏图表示法
  用Venn 图的包含表示树的有向边
- 凹入表表示法
  类似图书目录
- 嵌套括号表示法
  迭代 Node()()

森林与二叉树的一一映射

- 树或森林与二叉树存在一一对应的映射.
- **森林**$\rightarrow$**二叉树**
  - 数学语言:

    - **加线**：在树中所有相邻的兄弟之间加一连线(从左指向右).
    - **抹线**：对树中每个结点, 除最左孩子外, 抹去该结点与孩子间的连线.
    - **整理**
  - 伪代码:

```
ForestToBinaryTree(F): \\F = [T_1,T_2,...T_n]
  空二叉树根节点 T
  F_1 = [T_11,T_12,...T_1m] \\ T_1除去根节点后的森林
  F_2 = [T_2,...T_n] \\ F除去T_1后的森林
  T->leftchild = ForestToBinaryTree(F_1)
  T->rightchild = ForestToBinaryTree(F_2)
  返回 T
```

- **二叉树**→**森林**
  - 以上变换的逆, 容易书写不做赘述

森林的遍历

- 先根dfs = 前序遍历二叉树
- 后根dfs = 中序遍历二叉树
- bfs : 同深度层被定义为二叉树储存结构的右斜线, 不能用二叉树的广度遍历模版

6.2 树的存储

子结点表示法

实质上就是图的邻接表

- 优势:
  - 查孩子个数, 结点值, 归并与删除
- 劣势:
  - 找兄弟结点

动态表示法

每个节点包含值和**所有**子节点指针
– 每个结点分配可变的存储空间 (若子结点数目发生变化, 需要重新分配存储空间)

"左孩子/右兄弟"表示法

每个节点包含: ***第一个**子节点, ***下一个**兄弟节点, 节点值, (父亲节点)

> 笔记符号说明: *表示指针, ()表示备选

- 优点:
  - 节省空间
- **静态**用数组实现, **动态**用类实现

> 本质是二叉树与森林间的双射

> 一些方法
> **寻找父结点**
>
> - 本质上是树/森林的遍历
> - 直观的做法用队列+while循环遍历森林即可, 递归方法则只适合树的遍历.
>
> **镜像变换**
>
> - 我认为应该递归, 对每个节点的处理是先有序地找到所有子节点, 再修改自身左子树指针和子节点的右兄弟指针, 最后对所有子节点递归.
>
> **删除给定树根的子树**
>
> - 非常繁琐

父指针表示法

每个结点仅保存指向其父结点的指针域

- 优点:
  - 寻找父结点, 树根
- 缺点:
  - 寻兄弟结点麻烦, 需要查询整个树结构.
  - 属于**无序树**

并查集

由不相交子集构成的集合.

- Find: 查询节点所在集合
- Union：归并两个集合
- 用于求解等价类问题

> 等价关系 : 自反, 对称, 传递

实现：用一棵树代表一个集合，树使用父指针表示法
Union：将结点较少树的根结点指向结点较多树的根结点，这可以把树的整体深度限制在O(logn).

路径压缩算法

查找 X 的路径，沿路径将所有节点的父指针都改为 X 的树根

- 产生极浅树
- 路径压缩使 Find 操作开销接近常数
  - 对 n 个结点进行 n 次 Find 操作的开销为 $O(n\alpha(n))$，约为 $\Theta(n\log^* n)$
    - α 是单变量Ackermann函数的逆
    - $\log^* n$ 为对 n 不断取对数直至 ≤ 1 的次数，$\log^* 65536 = 4$
    -

6.3 树的顺序存储

带右链的先根次序表示法

- 任何结点的子树的所有结点都直接跟在该结点之后.
- 每棵子树的所有结点都聚集在一起，中间不会插入别的结点.
- 任何一个分支结点后面跟的都是它的第一个子结点（**如果存在的话**）.
- 结点除包含本身数据外，还附加两个表示结构的信息字段ltag（0有子结点，1无子结点），info, rlink（右指针，指向下一个兄弟）. ltag可以重塑llink，但是占用存储空间更少.

6.3.2 带双标记位的先根次序表示法

- 用rtag代替rlink.
- rtag为1，结点无兄弟；rtag为0，有右兄弟.
- 当结点x的rtag为0时，它的rlink应指向结点序列中排在以结点x为根的子树中最后结点的后面的那个结点y.
- **有兄弟结点和无孩子结点一一对应，满足栈特性. **
  结点x的兄弟结点y的确定方法：

  ➢ **由排列次序和ltag，rtag的值推知rlink的值**

  - **先根次序中子树结点嵌套出现，在顺序搜索中要嵌套处理x的所有子树，因此确定y要用到栈结构**
  - **有兄弟的结点(rtag=0)，都唯一对应一个无孩子的结点(ltag=1)，成对出现，满足栈特性，即**
    - **扫描到一个rtag为0的结点就将它进栈**
    - **扫描到一个ltag为1的结点时，就从栈顶弹出一个结点，并设置其rlink指向下一个要读出的节点，即其兄弟节点**

有兄弟就入栈, 无孩子就出栈.

6.3.3 带度数的后根次序表示法

- info是结点的数据, degree是结点的度数.
- 将带度数的后根次序转化成森林时
  - 从左至右进行扫描, 度为0的结点是叶子结点（也可看做一棵子树）.
  - 当遇到度数非0（设为k）的结点时, 则排在该结点之前且离它最近的k个子树的根就是该结点的k个子结点.
- 利用栈实现:
  - 遇到零度顶点就入栈.
  - 遇到非零k度顶点就从栈中弹出k个节点作为其子结点, 然后将该非零顶点入栈.
  - 持续扫描, 直至序列扫描完毕.
- 思考:
  - 带度数的先根次序? 从右到左即可.
  - 带度数的层次次序?

6.3.4 带双标记的层次次序表示

- 先看rtag:
  - rtag=0: 下一个结点即为其兄弟结点.
  - rtag=1: 无兄弟结点.
- 再看ltag:
  - ltag=1: 无孩子结点.
  - ltag=0: 有孩子结点. <—**重点考虑**
- **有孩子节点与无兄弟节点一一对应, 满足队列特性. **
- **有孩子则入队列, 无兄弟则出队列**.

> **求最大矩形面积**: 给定 n 个非负整数, 代表柱状图上每个柱的高度（每个柱的宽度均为 1）, 求这个柱状图中最大的矩形面积.

- O(n^3)
  - 枚举所有左右边界可能性, 并且在边界内找最低的柱子, 枚举边界复杂度O(n^2), 找最低柱子O(n), 总复杂度O(n^3)
- O(n^2)
  - 枚举每个柱子作为右边界, 往回枚举所有之前的柱子作为左边界, 枚举过程中记录最低柱子, 则最矮柱子*左右边界距离为当前矩阵面积.

    - 取所有这样的面积最大值, 右边界有O(n)种可能, 往回找比它矮的柱子也是O(n), 总复杂度为O(n^2)
- O(n)
    - 关键：维护一个栈, 用于存储直方图柱子中的索引, 并保证栈中索引对应的柱子高度递增(**单调栈**)
    - 枚举直方图中的每一个柱子
    - 当栈为空, 或者当前柱子的高度大于栈顶柱子的高度时, 将当前柱子的索引压入栈.
    - 当前柱子的高度小于或等于栈顶索引对应柱子的高度时, 持续从栈中弹出柱子, 直到栈为空或者栈顶柱子的高度小于当前柱子的高度. 弹出操作结束后, 压入当前柱子的索引.
    - 对于每一个弹出的柱子, 计算以该柱子为高度向两边扩展能得到的最大宽度：
        - 若栈非空, 最大宽度为当前柱子到弹出柱子的距离(不含当前柱子)
        - 若栈为空, 最大宽度为到当前柱子为止的总柱子数减1, 根据高度与最大宽度来求当前矩形面积, 并更新最大面积.