7，字典和集合-4

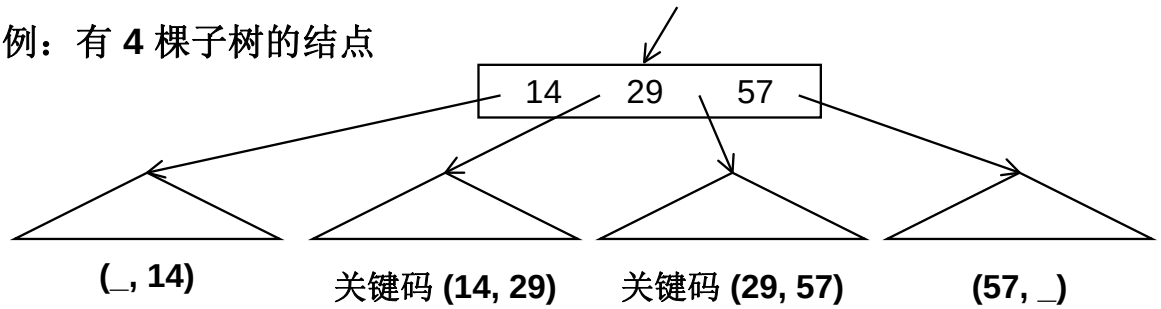
- 数据检索和字典，数据规模和检索，索引和字典
- 基于表和排序表实现字典和检索，二分检索
- 散列的概念，散列表和字典，散列函数和冲突消解，实现
- 集合数据结构，集合的实现，位向量实现
- Python 的字典和集合
- 二叉排序树的概念和实现
- 最佳二叉排序树，等概率和不等概率情况，构造算法
- 支持动态操作的排序树：AVL 树
- 支持字典实现的其他树形结构简介

其他树形结构（简介）

- 二叉排序树类结构以二叉树为基础，控制子树的高度差，从而控制平均检索长度，保证最长检索长度与所存数据量之间的对数关系
- 目标相同的另一批树形结构基于多分支排序树，其共性是保持到叶节点的所有路径长度相同，并保证每个分支结点至少有两个或更多分支，这样，n 个结点的树里的路径长度自然不会超过 O(log n)
- 多分支排序树的分支结点保存一些关键码值，作为检索时的导向

  如果一个分支结点有 k 棵子树，就用 k-1 个关键码区分各子树保存的值。检索时通过与这些关键码比较就能决定进入哪棵子树

- 例：有 4 棵子树的结点



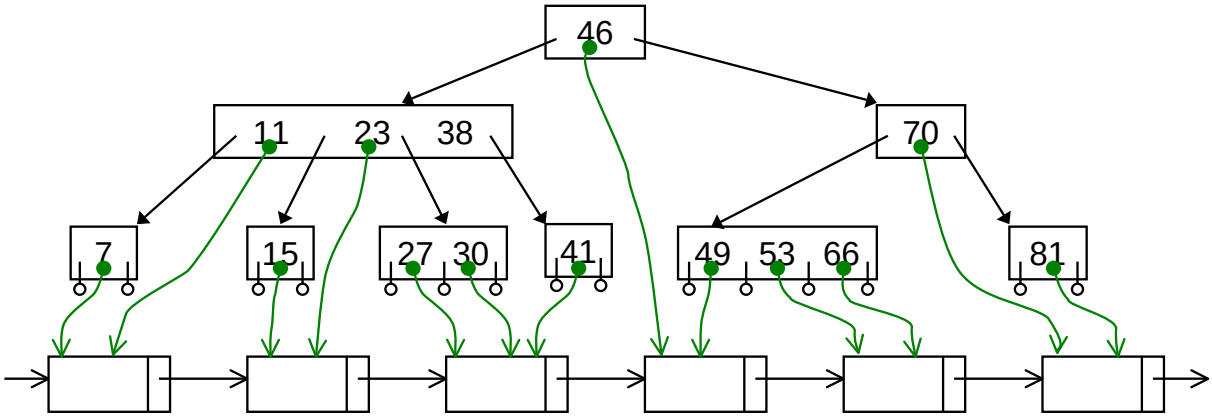
其他树形结构（简介）

- 实现多分支排序树的一个问题是每个结点的大小可能变动
  - 为便于管理通常采用统一大小的结点，这决定了最大分支数 m
  - 为保证空间利用率（和良好树结构），规定结点的最小分支数≥2
  - 允许具体结点的分支数（子树个数）在这两个值的范围内变化
- 维持树高度（也确定了最长路径的长度）的技术：
  - 加入新数据时，若目标结点数据项数将要超出允许范围，则或在兄弟结点间调整数据项（同时修改父结点关键码），或分割结点
  - 删除数据时，若目标结点数据项数将要低于允许范围，则或在其兄弟结点之间调整，或者合并结点
- 人们通常基于上面基本考虑开发一套技术，保证多分支树的结构良好。由于树中最长路径的长度相同，数据按关键码顺序存储，检索效率高

  下面看一个例子：数据库技术中常用于实现索引结构的 B 树

B 树及其使用

- B 树是一种多分排序树，通过维护结点分支数保证良好结构
- B 树常被用于实现大型数据库（也可以看作字典）的索引
  - 数据库记录（项）存在一批外存区块里（外层存储单元比较大）
  - B 树里记录的是关键码到数据（记录）存储位置的索引
- 下面是一棵 4 阶 B 树，树中的分支结点至多有 4 个分支



B 树

- B 树的定义。一棵 m 阶 B 树或空，或有下面特征：

  分支结点至多 m-1 个排序存放的关键码。根结点至少一个关键码，其他结点至少 ⌊(m-1)/2⌋ 个关键码。叶结点位于同一层表示检索失败

  如果分支结点有 j 个关键码，它就包含 j+1 棵子树，结点信息为序列 $(p_0, k_0, p_1, k_1, \ldots, k_{j-1}, p_j)$，其中 k 为关键码，p 为子结点引用，$k_i$ 大于子树 p_i 里的所有关键码，小于 p_{i+1} 里的所有关键码

- 检索：从根出发，在分支结点检索关键码，或找到，或确定了一棵可能存在被检索关键码的子树并转入该子树继续检索，直至成功或确定失败
- 插入新数据：基于关键码找到应该插入位置（在叶结点）。如果这里的数据项少于 m-1 项就直接按序插入；否则分裂该结点，把较大的一半关键码放入另一结点，居中的关键码插入其父结点相应位置

  注意：结点分裂时将关键码插入父结点，也是关键码插入。如果这时父结点有 m-1 个关键码，它也要分裂并将一个关键码插入其上层。这种分裂可能传播到根结点。根结点的分裂导致这棵树升高一层

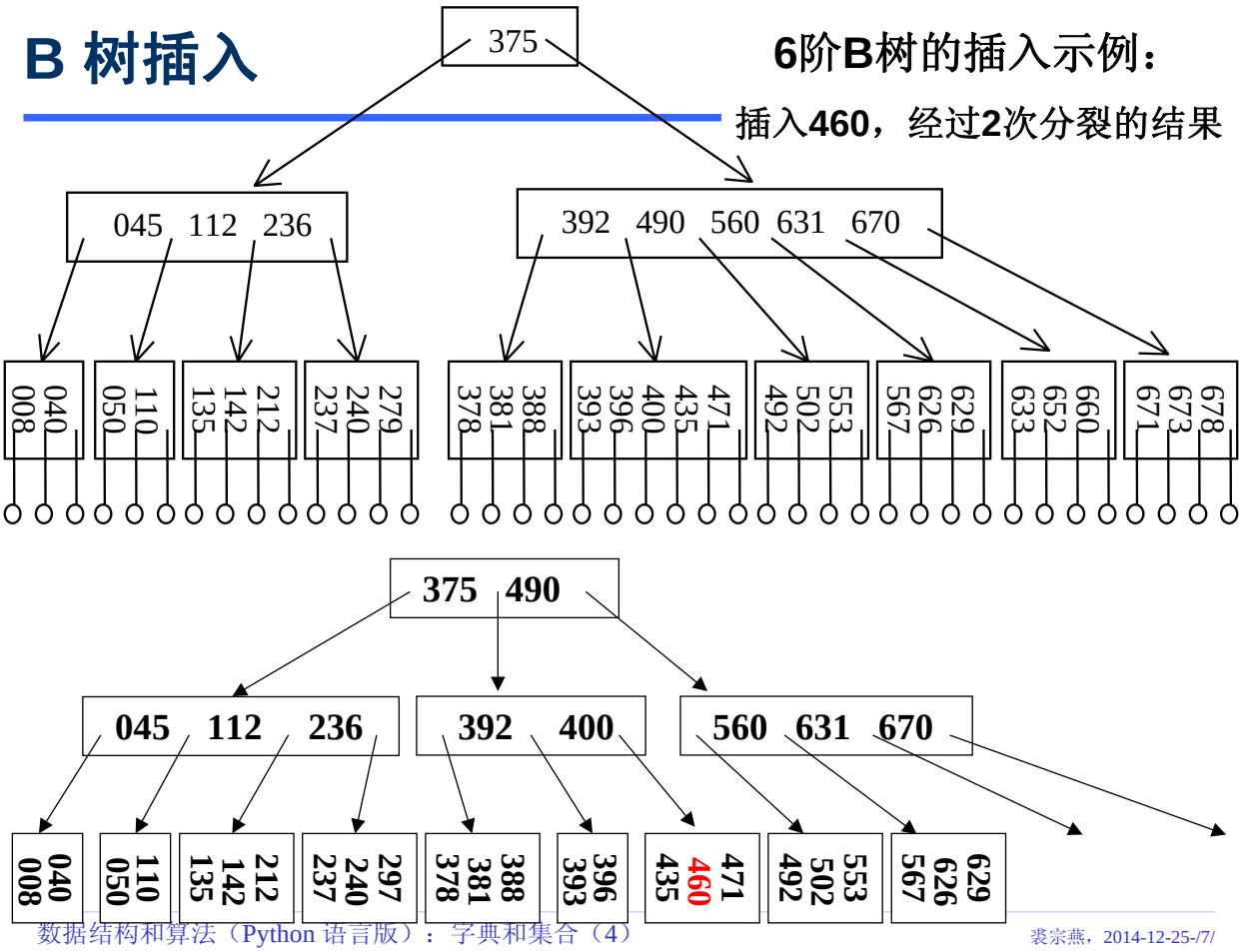
B 树简介

- 删除数据。基于关键码找到要删除数据项，具体删除分几种情况
- 要删除关键码在最下层结点。如关键码个数多于 (m-1)/2 就直接删除
  - 关键码个数不足时先在其兄弟结点间调整（并调整父结点分隔码）
  - 关键码过少无法调整时就与一个兄弟结点合并，这时位于其父结点的分隔关键码也要拿到合并后的结点里
  - 合并动作也导致从其父结点删除一个关键码，这也可能导致父结点与其兄弟结点之间的关键码/数据调整，或者结点合并
- 要删除的关键码在分支结点
  - 将其左子树的最右结点（一定在叶结点，设其为 r）关键码（和数据）拷贝到被删除结点的位置
  - 随后的工作就像是从 r 原来的位置删除结点一样，同样有上面说的各种情况：直接删除，兄弟间调整，或结点合并
- 结点合并到达根时，如果它仅有两个子结点且需合并，就使树降低一层

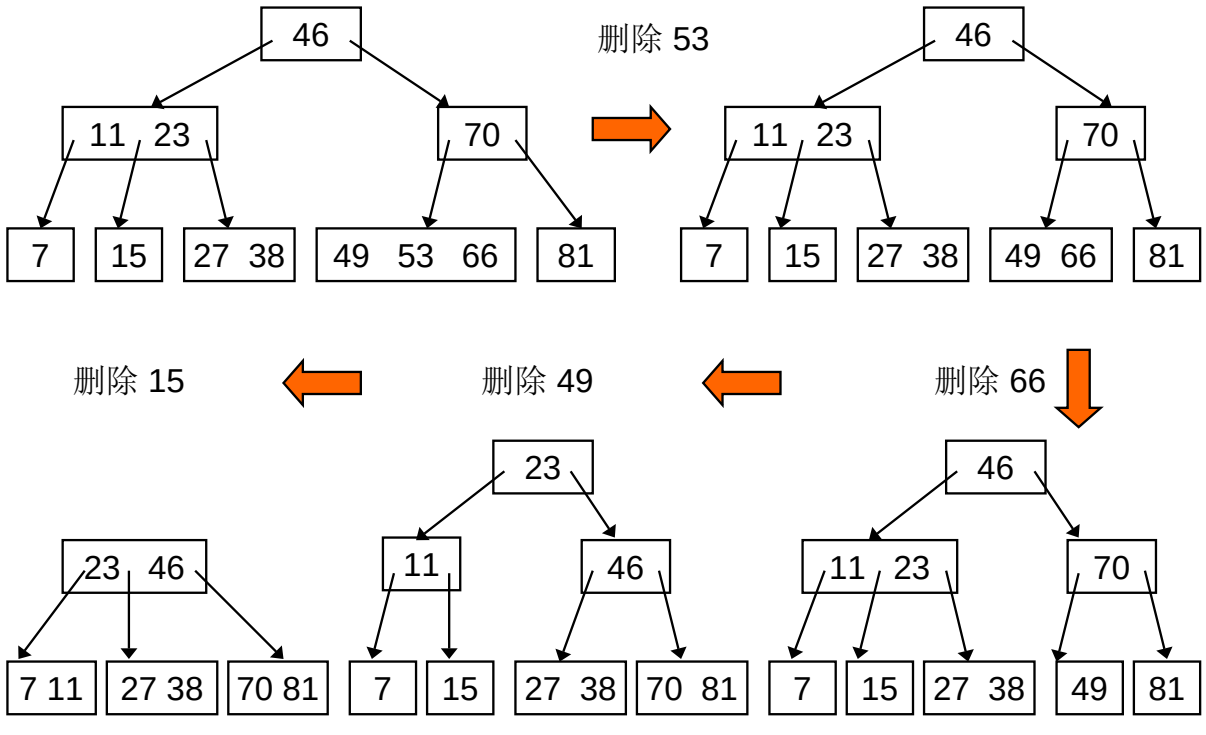
B 树插入

6阶B树的插入示例：

插入460，经过2次分裂的结果



B 树删除（考虑下面4 阶 B 树）



B+ 树

- B+ 树是另一种与 B 树类似的结构
- 定义：一棵 m 阶的 B+ 树或为空，或是满足下列条件的树：
  - 每个分支结点至多有 m 棵子树，除根外的分支结点至少有⌈m/2⌉棵子树，如果根结点不是叶结点则至少有两棵子树
  - 结点里排序存储关键码，叶结点保存数据项的关键码及其存储位置分支结点里的一个关键码关联着一棵子树，这个关键码等于其子树的根结点里的最大关键码
  - 叶结点可以看作基本索引块，分支结点可以看作索引的索引
- B+ 树也通过结点分裂/合并处理插入和删除，维护树结构良好。B+ 树相对简单，操作比 B 树简单，使用更广泛。具体情况可参考其他资料
- B 树/B+ 树都通过允许结点分支在一定范围里变化的方式维护树结构，支持动态操作。实际中的 B 树和 B+ 树结点都很大，分支可能达到数百个，因此常常只有很少几层就能索引到大量的数据项

字典：总结

- 字典是计算机系统理最常用的一类结构，实现数据的存储和检索
- 字典有许多实现方式，主要的包括线性表、散列和树形结构

  不同实现方式具有不同的性质，适合不同的字典规模和使用方式

  具有不同的操作复杂性，这方面应该理解
- 开发最多的技术属于树形结构。二叉排序树不能保证树结构良好，也不能保证结构的动态维持。最佳二叉排序树是静态构造的结构
- 人们开发了一些支持动态操作又能自动维护结构的树，主要有
  - 能在插入删除中维护结构的二叉树，包括 AVL 树（平衡二叉排序树），红黑树等。适合作为内存字典的基础结构。主要技术是通过局部的微调保持全局的结构良好
  - B 树/B+树，属于多分树，一个结点有许多分支（通常大大多于两个），一个结点可以作为一个外存存储单位。这类树有自己定义的一套维护树结构的方法，通常采用结点分裂/合并技术