为什么是B+Tree

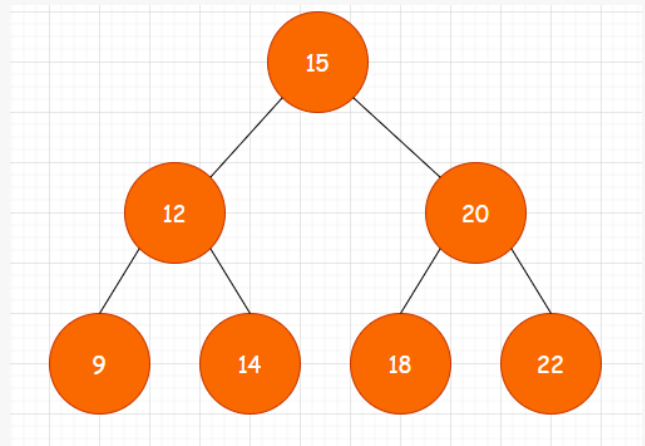
需求分析

- 查询
  - 精确查询
  - 范围查询

  一个有序数组似乎就可以满足这两种查询，通过二分搜索速度也不慢
- 插入：为了保证元素的有序，有序数组显然就没办法支持快速的插入需求了

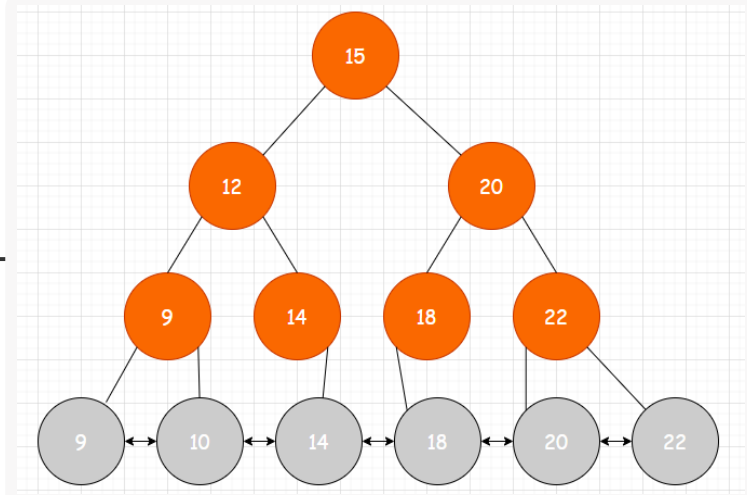
二分搜索树与多叉树

二分搜索树BST



范围查询很慢，需要从根节点不断的向下遍历

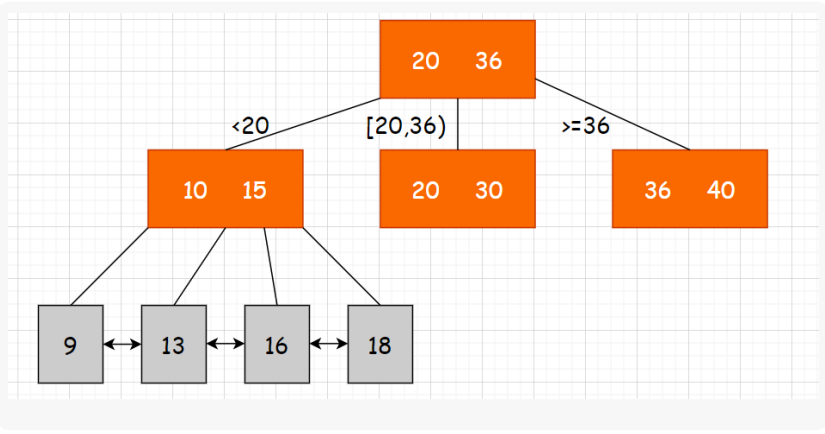
二分搜索树BST升级版（类似skiplist）



将数据都放在叶子节点，非叶子节点称之为索引，解决了范围查询需要不断遍历的问题。通过prev和next指针向前或者向后遍历

但是，数据量大的话，这棵树就会变得很高。2000万数据的话，这颗树就会达到25层左右的高度。换句话说，也就是需要25次磁盘的IO才能取出一条数据。

B+Tree



多叉搜索树，降低了树的的高度，也就降低了查询IO次数，这样子4层高度就可以容纳上亿的数据

B-Tree 与 B+Tree

B-Tree同样也是多叉平衡树，但是与B+树不同的地方在于它的非叶子节点也会保存数据

- 范围查询
  - B+Tree定位到第一条记录，然后通过叶子节点上数据链表的指针，不断向下直到第一条不满足条件的节点即可
  - B-Tree因为非叶子节点也保存了数据，因此无法通过链表单向遍历。需要不断进行回溯查询，每一次左右节点的获取都是一次随机IO