根据平衡二叉树的启发，想到平衡多路查找树结构，也就是这篇文章所要阐述的第一个主题B-tree，即B树结构(后面，我们将看到，B树的各种操作能使B树保持较低的高度，从而达到有效避免磁盘过于频繁的查找存取操作，从而有效提高查找效率)。

B-树是一种**平衡的多路查找树**，它在文件系统中很有用（原因之前已经介绍了）。B-树的结构有如下的特点：   
一棵度为m的B-树称为m阶B-树。一个结点有k个孩子时，必有k-1个关键字才能将子树中所有关键字划分   
为k个子集。B-树中所有结点的孩子结点最大值称为B-树的阶，通常用m表示。从查找效率考虑，一般要求  
m≥3。一棵m阶的B-树或者是一棵空树，或者是满足下列要求的m叉树：

* 树中的每个结点至多有m颗子树。
* 若根结点不是叶子结点，则至少有两颗子树。
* 除根结点外，所有非终端结点至少有[ m/2 ] ( 向上取整 )颗子树。
* 所有的非终端结点中包括如下信息的数据

（n,A0,K1,A1,K2,A2,….,Kn,An）   
其中：Ki（i=1,2,…,n）为关键码，且Ki < K(i+1)，

Ai 为指向子树根结点的指针(i=0,1,…,n)，且指针A(i-1) 所指子树中所有结点的关键码均小于Ki (i=1,2,…,n)，An 所指子树中所有结点的关键码均大于Kn.   
这里写图片描述   
n 为关键码的个数。

* 所有的叶子结点都出现在同一层次上，并且不带信息（可以看作是外部结点或查找失败的结点，实际上这些结点不存在，指向这些结点的指针为空）。

由于B- 树通常存储在磁盘上， 则前一查找操作是在磁盘上进行的， 而后一查找操作是在内存中进行的， 即在磁盘上找到指针p 所指结点后， 先将结点中的信息读入内存， 然后再利用顺序查找或折半查找查询等于K 的关键字。显然， 在磁盘上进行一次查找比在内存中进行一次查找的时间消耗多得多.   
 因此， 在磁盘上进行查找的次数、即待查找关键字所在结点在B- 树上的层次树， 是决定B树查找效率的首要因素，对于有n个关键字的m阶B-树，从根结点到关键字所在结点的路径上路过的结点数不超过： 这里写图片描述

//插入删除详见<http://blog.csdn.net/qq_35644234/article/details/66969238>

 B-树中有两种节点类型：索引节点和叶子节点。叶子节点是用来存储数据的，而索引节点则用来告诉用户存储在叶子节点中的数据顺序，并帮助用户找到相应的数据。

       B-树的搜索，从根节点开始，对节点内的关键字有序进行二分查找，如果命中则结束，否则进入查询关键字所属范围的儿子节点，重复。直到所对应的儿子指针为空，或已经是叶子节点。

       B-树是一种多路搜索树：

       (1). 定义任意非叶子节点最多有M个儿子，且M>2;

       (2). 根节点的儿子数为[2,M];

       (3). 除根节点以外的非叶子节点的儿子数为[M/2,M];

       (4). 每个节点存放至少M/2-1(取上整)和至多M-1个关键字;

       (5). 非叶子节点的关键字个数=指向儿子节点的指针的个数-1;

       (6). 非叶子节点的关键字：k[i]<k[i+1];

       (7). 非叶子节点的指针：p[1]，p[2]，·····，p[M]；其中p[1]指向的关键字小于k[1]的子树，p[M]指向的关键字大于K[m-1]的子树;

       (8). 所有的叶子节点位于同一层;

**B+树**

数据结构是B-树实现的增强版本。尽管B+树支持B-树索引的所有特性，它们之间最显著的不同点在于B+树中底层数据是根据被提及的索引列进行排序的。B+树还通过叶子节点之间的附加引用来优化扫描性能。

       B+搜索和B-搜索不同，区别是B+树只有达到叶子节点才命中（B-树可以在非叶子节点命中），其性能等价于关键字全集做一次二分搜索。

      B+树的特性：

     （1）所有关键字都出现在叶子节点的链表中，叶子节点相当于存储数据的数据层。

     （2）不可能在非叶子节点上命中。

     （3）非叶子节点相当于是叶子节点的索引，叶子节点相当于数据层。