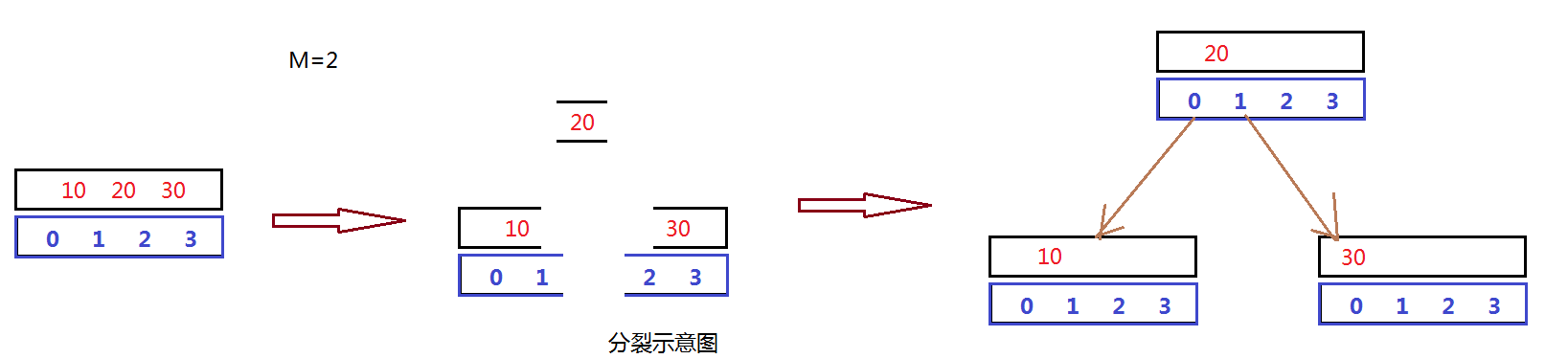
**B树简介**

B树，是为磁盘或其他直接存取辅助存储设备二设计的一种平衡查找树，可以大大减少访问磁盘I/O的次数，因此在数据库系统常使用Ｂ数或Ｂ树的变形来存储信息。

Ｂ树满足某种条件，与红黑树或其他搜索树不同，一棵M（M>2）的B树，是一棵M路的平衡搜索树，它允许有多条分支子树，它可以是一条空树，或者满足以下性质：

1. 根节点至少有两个孩子
2. 每个非根节点有[ M/2，M ]个孩子
3. 每个非根节点有[ (M/2) -1，M-1 ]个关键字，并且以升序排序
4. key[i]和key[i+1]之间的孩子节点的值介于两者之间
5. 所有的叶子节点都在同一层

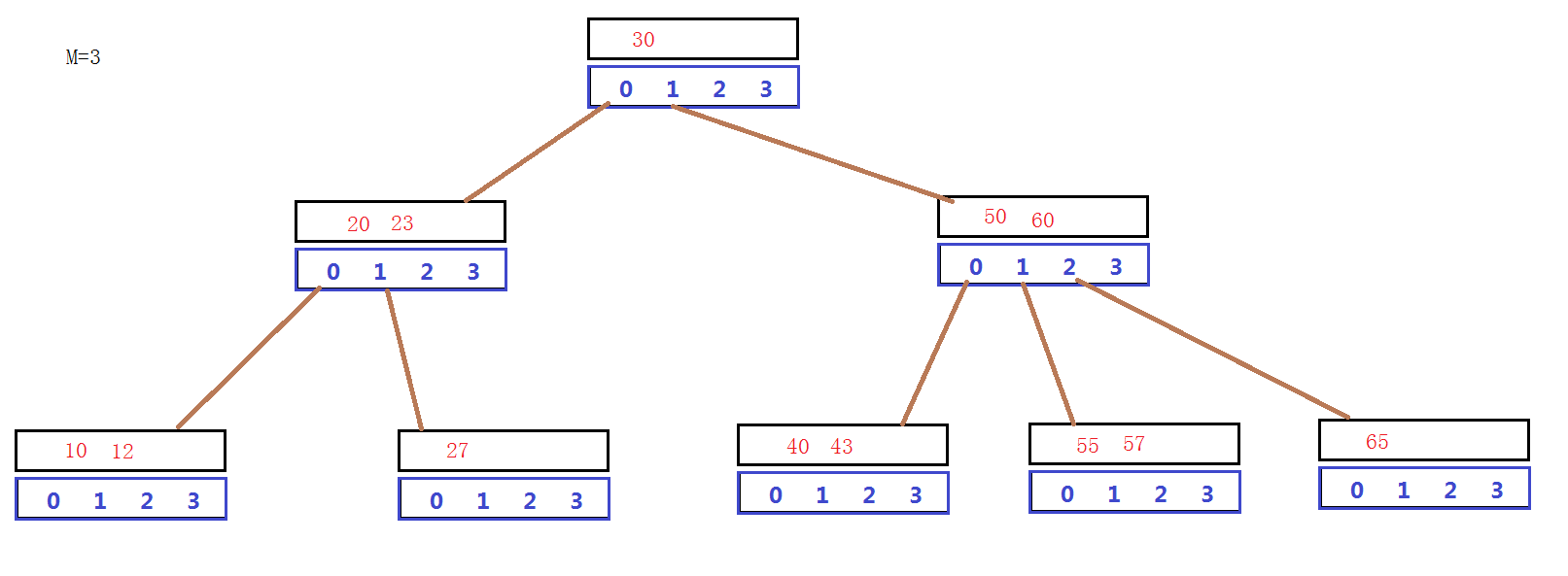
B树是一棵向上生长的树，当一个节点中的关键字个数达到上限之后，会进行分裂，同时会向上产生一个新的节点，分裂得到两个子节点和一个父节点，父节点只有原来节点中的中间key值，两个子节点将平分原来节点中剩下的key和孩子。这些原因使得B树满足上述条件2~5。

****

**B树算法**

**关键字：分裂算法、插入算法、删除算法、中序遍历算法、**

根据上述B树的要求，这里给出一张B树的示意图（M=3）



上面说过，M表示的是该节点的孩子个数，但上图中，孩子给出了4个，那么对应的关键字会有3个，和一开始的理论不相符。这里需要说明一下，因为每次向B树中插入节点之后，会进行判断，该节点的关键字个数是否超过了M，如果超过，我们需要进行分裂算法（后面会提到）。

经过简单分析，这里给出B树的节点的定义及构造函数。

template <typename K,int M>

struct BTreeNode

{

K \_key[M]; //关键字数组

BTreeNode<K,M>\* \_sub[M+1]; //指向孩子节点的指针数组

BTreeNode<K,M>\* \_parent; //指向父节点的指针

size\_t \_size; //该节点中已经插入的关键字的个数

BTreeNode()

:\_parent(NULL)

,\_size(0)

{

size\_t i = 0;

for(i = 0; i < M; i++)

{

\_key[i] = K();

\_sub[i] = NULL;

}

\_sub[i] = NULL;

}

}

1. 查找

假设这里我们依旧有一棵B树，在B树中查找一个结点，和普通的平衡树基本思路一样，比该点的key大就向右查找，比该点的key值小，就向左查找。只不过对于B树而言，每个节点有M-1个关键字。因此在向下查找的同时，需要对每个节点中的每个key进行比较。

由于每个节点这里有M个关键字，下标从0~M-1，每个节点有M+1个孩子，指针数组的下标从0~M，仔细观察上树，对于某个节点node而言，比节点中的某个key小的一个值，下一次查找的孩子应该和该key的下标相同。

还需要注意的一点，就是我这里的Find函数是希望能够被其他函数使用的，不仅仅是希望得到一个bool值或找到的Node\*，在这里设计Find函数，是希望***当找到该key值的话返回key所在节点的下标，同时返回一个指向该节点的指针；没有找到返回 -1，同时返回该节点应该所在位置的父节点***。初衷很简单，是为了给待会需要实现的Insert函数调用，达到代码的复用性。如果我们只是判断该节点在不在B树内，那对返回值我们就只需要关注bool即可。要实现返回两个参数，有两种思路：第一就是通过函数传参数的方式，传递引用达到目的，第二就是使用pair类型。

|  |
| --- |
| Pair是库中定义好的一个双变量结构体，这里参考库的pair实现  template<class \_Ty1,class \_Ty2>  struct pair  { // store a pair of values  typedef \_Ty1 first\_type;  typedef \_Ty2 second\_type;  } |

代码实现：

typedef pair<Node\*, int> FindType;

FindType Find(const K& key)

{

Node\* parent = NULL;

Node\* cur = \_root;

while(cur)

{

size\_t i =0;

while(i < cur->\_size)

{

if(key > cur->\_key[i])

{

i++;

}

else if(key < cur->\_key[i])

{

break;

}

else

{

return FindType(cur, i);

}

}

parent = cur;

cur = cur->\_sub[i];

}

return FindType(parent,-1);

}

1. 插入