对于长整型数据的映射,如何解决Hash冲突和Hash表大小的设计是一个很头疼的问题。

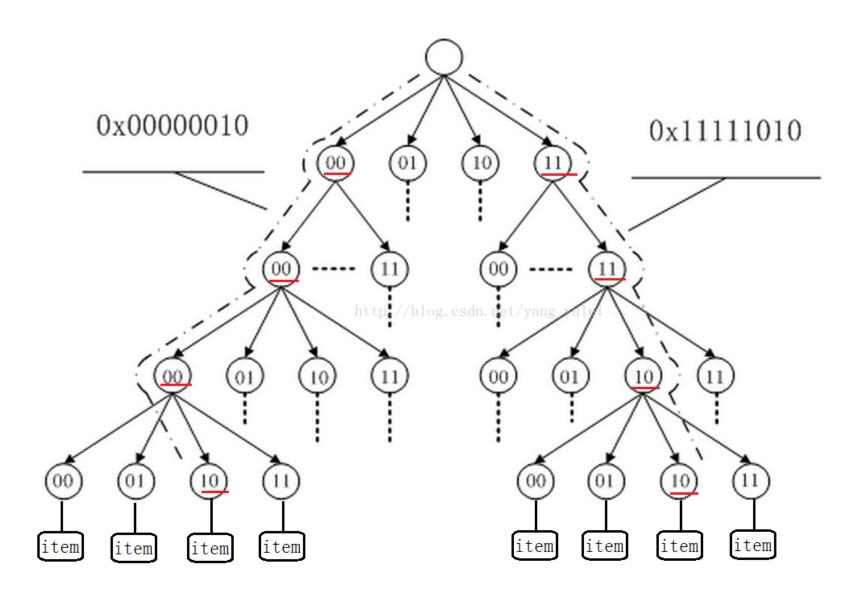
radix树就是针对这种稀疏的长整型数据查找,能快速且节省空间地完成映射。借助于Radix树,我们可以<mark>实现对于长整型数据类型的路由。**利用** radix树可以根据一个长整型(比如一个长ID)快速查找到其对应的对象指针</mark>。这比用hash映射来的简单,也更节省空间,使用hash映射hash函数难以设计,不恰当的hash函数可能增大冲突,或浪费空间。

radix tree是一种多叉搜索树,树的叶子结点是实际的数据条目。每个结点有一个固定的、2^n指针指向子结点(每个指针称为槽slot, n为划分的基的大小)

插入、删除

(使用一个比特位判断,会使树的高度过高,非叶节点过多。故在实际应用中,我们一般是使用多个比特位作为树节点的判断,但多比特位会使节点的子节点槽变多,增大节点的体积,一般选用2个或4个比特位作为树节点即可)

如图:



插入:

我们在插入一个新节点时,我们根据数据的比特位,在树中向下查找,若没有相应结点,则生成相应结点,直到数据的比特位访问完,则建立叶节 点映射相应的对象。

删除:

我们可以"惰性删除",即沿着路径查找到叶节点后,直接删除叶节点,中间的非叶节点不删除。

应用

•

Radix树在Linux中的应用:

Linux基数树 (radix tree) 是将long整数键值与指针相关联的机制,它存储有效率,并且可快速查询,用于整数值与指针的映射(如:IDR机制)、内存管理等。

IDR (ID Radix) 机制是将对象的身份鉴别号整数值ID与对象指针建立关联表,完成从ID与指针之间的相互转换。IDR机制使用radix树状结构作为由id进行索引获取指针的稀疏数组,通过使用位图可以快速分配新的ID,IDR机制避免了使用固定尺寸的数组存放指针。IDR机制的API函数在lib/idr.c中实现。

Linux radix树最广泛的用途是用于**内存管理**,结构address_space**通过radix树跟踪绑定到地址映射上的核心页**,该radix树允许内存管理代码快速 查找标识为dirty或writeback的页。其使用的是数据类型unsigned long的固定长度输入的版本。每级代表了输入空间固定位数。Linux radix树的 API函数在lib/radix-tree.c中实现。(把页指针和描述页状态的结构映射起来,使能快速查询一个页的信息。)

Linux内核利用radix树在文件内偏移快速定位文件缓存页。

Linux(2.6.7) 内核中的分叉为 64(2⁶),树高为 6(64位系统)或者 11(32位系统),用来快速定位 32 位或者 64 位偏移,radix tree 中的每一个叶子节点指向文件内相应偏移所对应的Cache项。

【radix树为稀疏树提供了有效的存储,代替固定尺寸数组提供了键值到指针的快速查找。】

后记



Radix树与Trie树的思想有点类似,甚至可以把Trie树看为一个基为26的Radix树。(也可以把Radix树看做是Tire树的变异) Trie树一般用于字符串到对象的映射,Radix树一般用于长整数到对象的映射。

trie树主要问题是树的层高,如果要索引的字的拼音很长很变态,我们也要建一个很高很变态的树么? radix树能固定层高(对于较长的字符串,可以用数学公式计算出其特征值,再用radix树存储这些特征值)