阅读：4,171　　作者：解学武

键树查找法（双链树和字典树）及C语言实现

键树，又称为数字查找树（根结点的子树个数 >= 2），同以往所学习的树不同的是，键树的结点中存储的不是某个关键字，而是只含有组成关键字的单个符号。

> 如果关键字本身是字符串，则键树中的一个结点只包含有一个字符；如果关键字本身是数字，则键树中的一个结点只包含一个数位。每个关键字都是从键树的根结点到叶子结点中经过的所有结点中存储的组合。

例如，当使用键树表示查找表 {CAI，CAO，CHEN，LI，LAN，ZHAO} 时，为了查找方便，首先对该查找表中关键字按照首字符进行分类（相同的在一起）：

```
{{CAI,CAO,CHEN},{LI,LAN},{ZHAO}}
```

然后继续分割，按照第二个字符、第三个字符、...，最终得到的查找表为：

```
{{CAI,CAO},{CHEN},{LI,LAN},{ZHAO}}
```

然后使用键树结构表示该查找表，如图 1 所示：

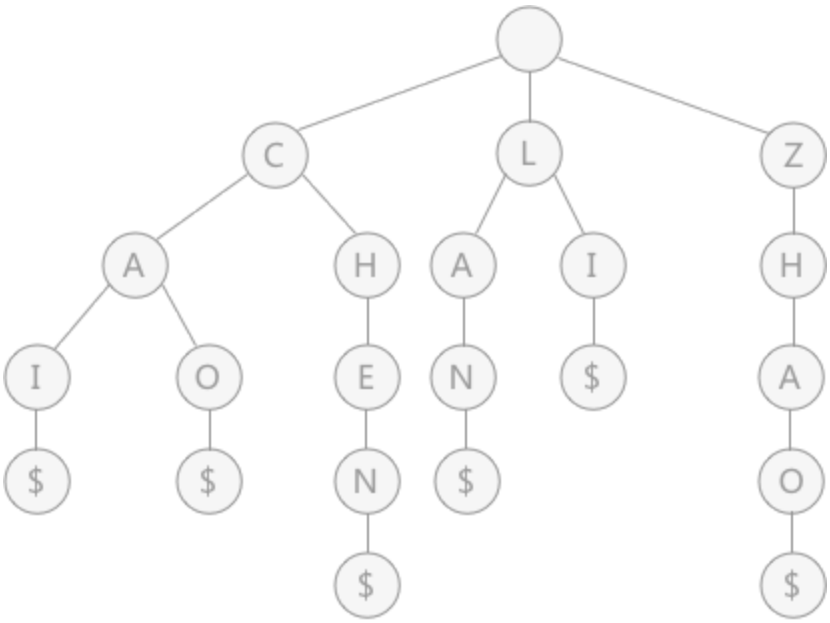

图 1 键树表示多个关键字

> 注意：键树中叶子结点的特殊符号 $ 为结束符，表示字符串的结束。使用键树表示查找表时，为了方便后期的查找和插入操作，约定键树是有序树（兄弟结点之间自左至右有序），同时约定结束符 ‘$’ 小于任何字符。

键树的存储结构

键树的存储结构有两种：一种是通过使用树的孩子兄弟表示法来表示键树，即双链树；另一种是以树的多重链表表示键树，即 Trie 树，又称字典树。

双链树

当使用孩子兄弟表示法表示键树时，树的结点构成分为3部分：

- symbol域：存储关键字的一个字符；
- first域：存储指向孩子结点的指针；
- next域：存储指向兄弟结点的指针；

> 注意：对于叶子结点来说，由于其没有孩子结点，在构建叶子结点时，将 first 指针换成 infoptr 指针，用于指向该关键字。当叶子结点（结束符 ‘$’ 所在的结点）中使用 infoptr 域指向该自身的关键字时，此时的键树被称为双链树。

如图 1 中的键树用孩子兄弟表示法表示为双链树时，如图 2 所示：

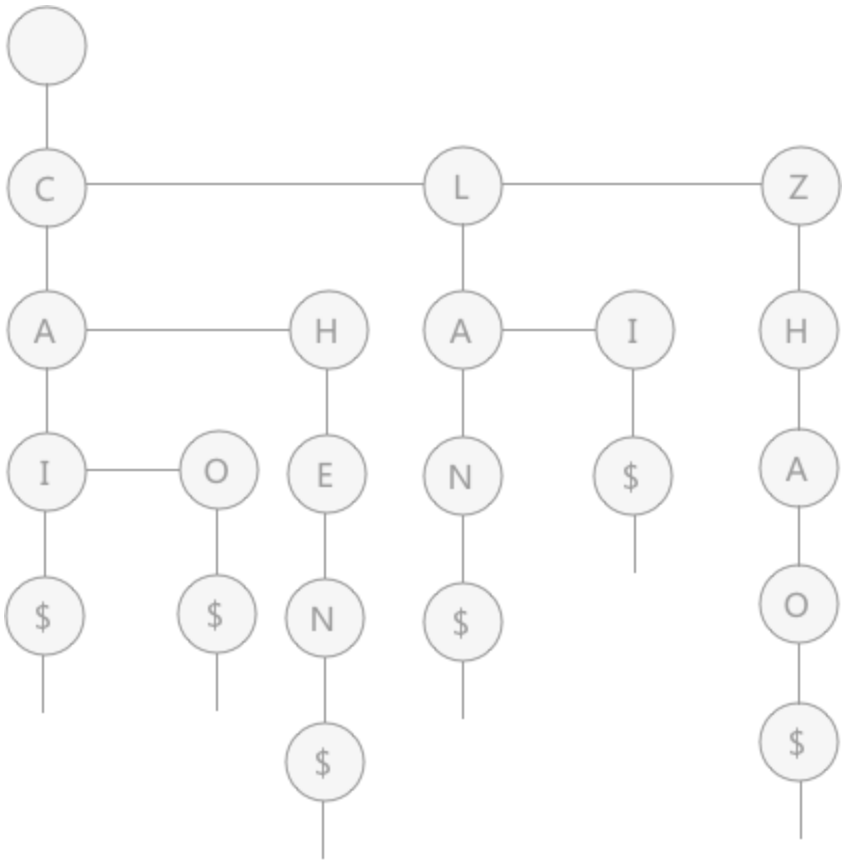

图 2 双链树

> 提示：每个关键字的叶子结点 $ 的 infoptr 指针指向的是各自的关键字，通过该指针就可以找到各自的关键字的首地址。

双链树查找功能的具体实现

在使用孩子兄弟表示法表示的键树中做查找操作，从树的根结点出发，依次同被查找的关键字进行比对，如果比对成功，进行下一字符的比对；反之，如果比对失败，则跳转至该结点的兄弟结点中去继续比对，直至比对成功或者为找到该关键字。

具体实现的代码：

```
#include <stdio.h>
typedef enum{LEFT,BRANCH}NodeKind;//定义结点的类型，是叶子结点还是其他类型的结点
typedef  struct {
    char a[20];//存储关键字的数组
    int num;//关键字长度
}KeysType;
//定时结点结构
typedef struct DLTNode{
    char symbol;//结点中存储的数据
    struct DLTNode *next;//指向兄弟结点的指针
    NodeKind *kind;//结点类型
    union{//其中两种指针类型每个结点二选一
        struct DLTNode* first;//孩子结点
        struct DLTNode* infoptr;//叶子结点特有的指针
    };
}*DLTree;
//查找函数，如果查找成功，返回该关键字的首地址，反则返回NULL。T 为用孩子兄弟表示法表示的键树，K为被查找的
DLTree SearchChar(DLTree T, KeysType k){
    int i = 0;
    DLTree p = T->first;//首先令指针 P 指向根结点下的含有数据的孩子结点
    //如果 p 指针存在，且关键字中比对的位数小于总位数时，就继续比对
    while (p && i < k.num){
        //如果比对成功，开始下一位的比对
        if (k.a[i] == p->symbol){
            i++;
            p = p->first;
        }
        //如果该位比对失败，则找该结点的兄弟结点继续比对
        else{
            p = p->next;
        }
    }
    //比对完成后，如果比对成功，最终 p 指针会指向该关键字的叶子结点 $，通过其自有的 infoptr 指针找到该
    if ( i == k.num){
        return p->infoptr;
    }
    else{
        return NULL;
    }
}
```

Trie树（字典树）

若以树的多重链表表示键树，则树中如同双链树一样，会含有两种结点：
1. 叶子结点：叶子结点中含有关键字域和指向该关键字的指针域；
2. 除叶子结点之外的结点（分支结点）：含有 d 个指针域和一个整数域（记录该结点中指针域的个数）；

> d 表示每个结点中存储的关键字的所有可能情况，如果存储的关键字为数字，则 d= 11（0—9，以及 $），同理，如果存储的关键字为字母，则 d=27（26个字母加上结束符 $）。

图 1 中的键树，采用字典树表示如图 3所示：

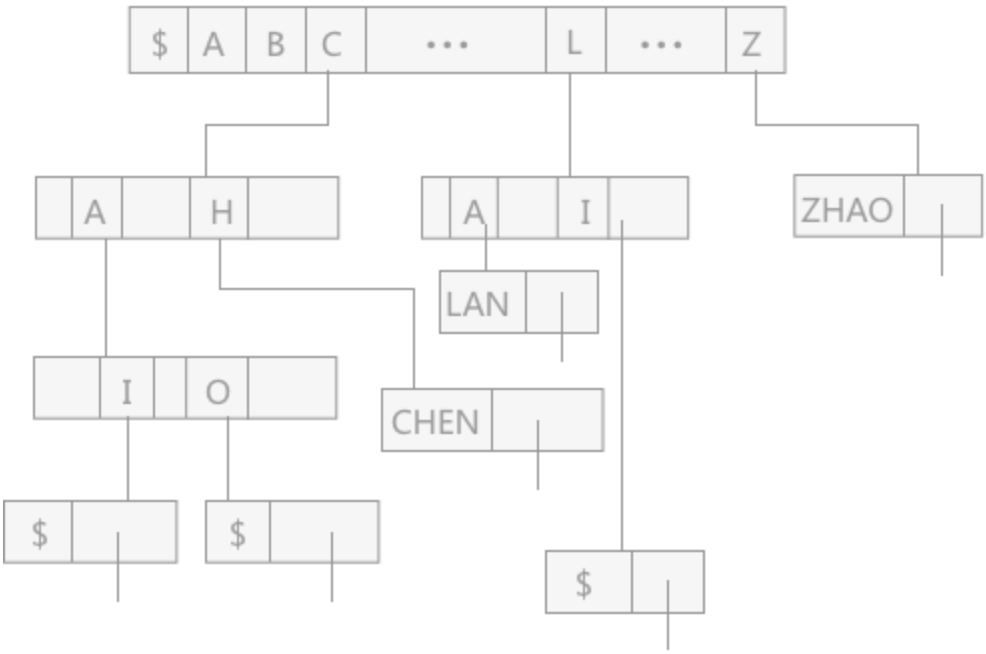

图 3 Trie树

> 注意：在 Trie 树中，如果从某个结点一直到叶子结点都只有一个孩子，这些结点可以用一个叶子结点来代替，例如 ZHAO 就可以直接作为叶子结点。

字典树查找功能的具体实现

使用 Trie 树进行查找时，从根结点出发，沿和对应关键字中的值相对应的指针逐层向下走，一直到叶子结点，如果全部对应相等，则查找成功；反之，则查找失败。
具体实现代码为：

```
01.  typedef enum{LEFT,BRANCH}NodeKind;//定义结点类型
02.  typedef struct {//定义存储关键字的数组
03.      char a[20];
04.      int num;
05.  }KeysType;
06.  //定义结点结构
07.  typedef struct TrieNode{
08.      NodeKind kind;//结点类型
09.      union{
10.          struct { KeysType k; struct TrieNode *infoptr; }lf;//叶子结点
11.          struct{ struct TrieNode *ptr[27]; int num; }bh;//分支结点
```

```
    };
}*TrieTree;
//求字符 a 在字母表中的位置
int ord(char  a){
    int b = a - 'A'+1;
    return b;
}
//查找函数
TrieTree SearchTrie(TrieTree T, KeysType K){
    int i=0;
    TrieTree p = T;
    while (i < K.num){
        if (p && p->kind==BRANCH && p->bh.ptr[ord(K.a[i])]){
            i++;
            p = p->bh.ptr[ord(K.a[i])];
        }
        else{
            break;
        }
    }
    if (p){
        return p->lf.infoptr;
    }
    return p;
}
```

使用 Trie 树进行查找的过程实际上是走了一条从根结点到叶子结点的路径，所以使用 Trie 进行的查找效率取决于该树的深度。

总结

双链树和字典树是键树的两种表示方法，各有各的特点，具体使用哪种方式表示键树，需要根据实际情况而定。例如，若键树中结点的孩子结点较多，则使用字典树较双链树更为合适。