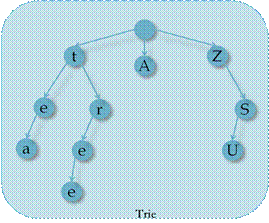
**Trie—单词查找树**

  **简介**

**Trie，又称单词查找树、前缀树，是一种哈希树的变种。应用于字符串的统计与排序，经常被搜索引擎系统用于文本词频统计。**

****

**含有单词“*tea*”“*tree*”“*A*”“*ZSU*”的一棵*Trie*。**

  **性质**

  **根节点不包含字符，除根节点外的每一个节点都只包含一个字符。**

  **从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。**

  **每个节点的所有子节点包含的字符都不相同。**

  **优点**

  **查询快。对于长度为m的键值，最坏情况下只需花费O(m)的时间；而BST最坏情况下需要O(m log n)的时间。**

  **当存储大量字符串时，Trie耗费的空间较少。因为键值并非显式存储的，而是与其他键值共享子串。**

  **Trie适用于“最长前缀匹配”。**

  **操作**

  **初始化或清空**

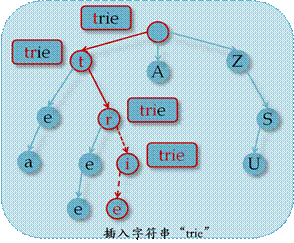
**遍历Trie，删除所有节点，只保留根节点。**

  **插入字符串**

**1.     设置当前节点为根节点，设置当前字符为插入字符串中的首个字符；**

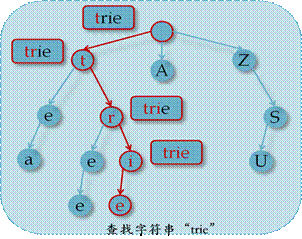
**2.     在当前节点的子节点上搜索当前字符，若存在，则将当前节点设为值为当前字符的子节点；否则新建一个值为当前字符的子节点，并将当前结点设置为新创建的节点。.**

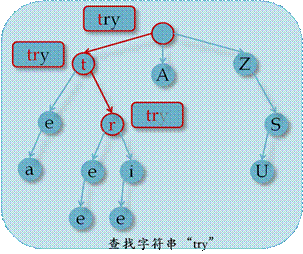
**3.     将当前字符设置为串中的下个字符，若当前字符为0，则结束；否则转2.**

****

  **查找字符串**

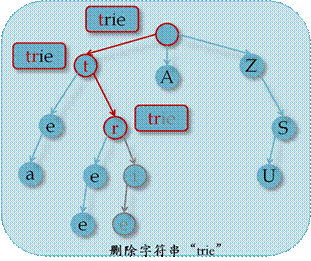
**搜索过程与插入操作类似，当字符找不到匹配时返回假；若全部字符都存在匹配，判断最终停留的节点是否为树叶，若是，则返回真，否则返回假。**

****

****

  **删除字符串**

**首先查找该字符串，边查询边将经过的节点压栈，若找不到，则返回假；否则依次判断栈顶节点是否为树叶，若是则删除该节点，否则返回真。**

****

 **实现  
对于字符表大小为S的字符串集，需建立一个S叉树来代表这些字符串的集合。**

  **代码**

http://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gifhttp://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.giftrie.h  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif/\*\*//\*\* 版权所有 (C) 2009 喻扬 中山大学  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif/\* 本程序只作学习用途，未经许可，不得用于任何商业目的。  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif\*/  
http://www.cppblog.com/Images/OutliningIndicators/None.gif#include <string.h>  
  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif/\*\*//\* trie的节点类型 \*/  
http://www.cppblog.com/Images/OutliningIndicators/None.giftemplate <int Size> //Size为字符表的大小  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gifstruct trie\_node http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 数据成员 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    bool terminable; //当前节点是否可以作为字符串的结尾  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    int node; //子节点的个数  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    trie\_node \*child[Size]; //指向子节点指针  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 构造函数 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    trie\_node() : terminable(false), node(0) http://www.cppblog.com/Images/dot.gif{ memset(child, 0, sizeof(child)); }  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif};  
http://www.cppblog.com/Images/OutliningIndicators/None.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gif/\*\*//\* trie \*/  
http://www.cppblog.com/Images/OutliningIndicators/None.giftemplate <int Size, typename Index> //Size为字符表的大小，Index为字符表的哈希函数  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedBlock.gifclass trie http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gifpublic:  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 定义类型别名 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    typedef trie\_node<Size> node\_type;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    typedef trie\_node<Size>\* link\_type;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 构造函数 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    trie(Index i = Index()) : index(i) http://www.cppblog.com/Images/dot.gif{}  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 析构函数 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    ~trie() http://www.cppblog.com/Images/dot.gif{ clear(); }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 清空 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    void clear() http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        clear\_node(root);  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        for (int i = 0; i < Size; ++i) root.child[i] = 0;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 插入字符串 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    template <typename Iterator>  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    void insert(Iterator begin, Iterator end) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        link\_type cur = &root; //当前节点设置为根节点  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif        for (; begin != end; ++begin) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif            if (!cur->child[index[\*begin]]) http://www.cppblog.com/Images/dot.gif{ //若当前字符找不到匹配，则新建节点  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                cur->child[index[\*begin]] = new node\_type;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                ++cur->node; //当前节点的子节点数加一  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif            }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            cur = cur->child[index[\*begin]]; //将当前节点设置为当前字符对应的子节点  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        cur->terminable = true; //设置存放最后一个字符的节点的可终止标志为真  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 插入字符串，针对C风格字符串的重载版本 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    void insert(const char \*str) http://www.cppblog.com/Images/dot.gif{ insert(str, str + strlen(str)); }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 查找字符串，算法和插入类似 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    template <typename Iterator>  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    bool find(Iterator begin, Iterator end) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        link\_type cur = &root;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif        for (; begin != end; ++begin) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            if (!cur->child[index[\*begin]]) return false;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            cur = cur->child[index[\*begin]];  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        return cur->terminable;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 查找字符串，针对C风格字符串的重载版本 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    bool find(const char \*str) http://www.cppblog.com/Images/dot.gif{ return find(str, str + strlen(str)); }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 删除字符串 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    template <typename Iterator>  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    bool erase(Iterator begin, Iterator end) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        bool result; //用于存放搜索结果  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        erase\_node(begin, end, root, result);  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        return result;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 删除字符串，针对C风格字符串的重载版本 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    bool erase(char \*str) http://www.cppblog.com/Images/dot.gif{    return erase(str, str + strlen(str)); }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 按字典序遍历单词树 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    template <typename Functor>  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    void traverse(Functor &execute = Functor()) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        visit\_node(root, execute);  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gifprivate:  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 访问某结点及其子结点 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    template <typename Functor>  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    void visit\_node(node\_type cur, Functor &execute) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        execute(cur);  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif        for (int i = 0; i < Size; ++i) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            if (cur.child[i] == 0) continue;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            visit\_node(\*cur.child[i], execute);  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 清除某个节点的所有子节点 \*/  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    void clear\_node(node\_type cur) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif        for (int i = 0; i < Size; ++i) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            if (cur.child[i] == 0) continue;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            clear\_node(\*cur.child[i]);  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            delete cur.child[i];  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            cur.child[i] = 0;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            if (--cur.node == 0) break;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 边搜索边删除冗余节点  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif       返回值用于向其父节点声明是否该删除该节点 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    template <typename Iterator>  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    bool erase\_node(Iterator begin, Iterator end, node\_type &cur, bool &result) http://www.cppblog.com/Images/dot.gif{  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif        if (begin == end) http://www.cppblog.com/Images/dot.gif{ //当到达字符串结尾：递归的终止条件  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            result = cur.terminable; //如果当前节点可以作为终止字符，那么结果为真  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            cur.terminable = false; //设置该节点为不可作为终止字符，即删除该字符串  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            return cur.node == 0; //若该节点为树叶，那么通知其父节点删除它  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        if (cur.child[index[\*begin]] == 0) return result = false; //当无法匹配当前字符时，将结果设为假并返回假，  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif                                                                  //即通知其父节点不要删除它  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif        else if (erase\_node(++begin--, end, \*(cur.child[index[\*begin]]), result)) http://www.cppblog.com/Images/dot.gif{ //判断是否应该删除该子节点  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            delete cur.child[index[\*begin]]; //删除该子节点  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            cur.child[index[\*begin]] = 0; //子节点数减一  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif            if (--cur.node == 0 && cur.terminable == false) return true; //若当前节点为树叶，那么通知其父节点删除它  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif        return false; //其他情况都返回假  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 根节点 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    node\_type root;  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedSubBlockStart.gifhttp://www.cppblog.com/Images/OutliningIndicators/ContractedSubBlock.gif    /\*\*//\* 将字符转换为索引的转换表或函数对象 \*/  
http://www.cppblog.com/Images/OutliningIndicators/InBlock.gif    Index index;  
http://www.cppblog.com/Images/OutliningIndicators/ExpandedBlockEnd.gif};