字典树[Trie]

|  |
| --- |
| Trie树也叫字典树，是一种用于快速检索的多叉树结构。如英文字母的字典树是一个26叉树。数字的字典树是一个10叉树。Trie树把要查找的关键词看作一个字符序列，并根据构成关键词字符的先后顺序构造用于检索的树结构；一棵m度的Trie树或者为空，或者由m棵m度的Trie树构成。特别的：和二叉查找树不同，在Trie树中，每个结点上并非存储一个元素。在Trie树中查找一个关键字的时间和树中包含的结点数无关，而取决于组成关键字的字符数。  http://hiphotos.baidu.com/matrush/pic/item/e6a06ed55a7853e2a044df52.jpg  特点： ①利用串的公共前缀->节约内存。 ②根结点(root)不包含任何字母。 ③其余结点仅包含一个字母(非元素)。 ④每个结点的子结点包含字母不同。  查找过程： ①在Trie树上进行检索总是始于根结点。 ②取得要查找关键词的第一个字母，并根据该字母选择对应的子树并转到该子树继续进行检索。 ③在相应的子树上，取得要查找关键词的第二个字母，并进一步选择对应的子树进行检索。 ④在某个结点处，关键词的所有字母已被取出，则读取附在该结点上的信息，即完成查找。  字典树比较一般的实现是用指针，所以又可分为动态开辟内存的字典树和静态开辟内存的字典树。两种写法各有长处，一般静态的速度较快，而动态的代码较简。以下是两种写法的模板：  动态：  #include <iostream> using namespace std; const int MAXM = 30,KIND = 26; int m; struct node {     char\* s;     int prefix;     bool isword;     node\* next[KIND];     node()     {         s = NULL;         prefix = 0;         isword = false;         memset(next,0,sizeof(next));     } }\*root;//根 void insert(node \*root,char \*s)//插入 {     node \*p = root;     for (int i = 0;s[i];i++)     {         int x = s[i] - 'a';         p->s = s+i;         if (p->next[x] == NULL)             p->next[x] = new node;         p = p->next[x];         p->prefix++;     }     p->isword = true; } bool del(node \*root,char \*s)//删除 {     node \*p = root;     for (int i = 0;s[i];i++)     {         int x = s[i] - 'a';         if (p->next[x] == NULL)             return false;         p = p->next[x];     }     if (p->isword)         p->isword = false;     else         return false;     return true; } bool search(node \*root,char\* s)//查找 {     node\* p = root;     for (int i = 0;s[i];i++)     {         int x = s[i]-'a';         if (p->next[x] == NULL)             return false;         p = p->next[x];     }     return p->isword; } int count(node \*root,char \*s)//统计后缀 {     node \*p = root;     for (int i = 0;s[i];i++)     {         int x = s[i] - 'a';         if (p->next[x] == NULL)             return 0;         p = p->next[x];     }     return p->prefix; } int main() {     m = 0;     root = new node;     char s[MAXM];     while (gets(s))     {         if (strcmp(s,"") == 0)             break;         insert(root,s);     }     while (gets(s))         printf("%d\n",count(root,s)); }  静态：  #include <iostream> using namespace std; const int MAXN = 100010,MAXM = 30,KIND = 26; int m; struct node {     char\* s;     int prefix;     bool isword;     node\* next[KIND];     void init()     {         s = NULL;         prefix = 0;         isword = false;         memset(next,0,sizeof(next));     } }a[MAXN\*MAXM],\*root;//根 void insert(node \*root,char \*s) {     node \*p = root;     for (int i = 0;s[i];i++)     {         int x = s[i] - 'a';         p->s = s+i;         if (p->next[x] == NULL)         {             a[m].init();             p->next[x] = &a[m++];         }         p = p->next[x];         p->prefix++;     }     p->isword = true; } bool del(node \*root,char \*s) {     node \*p = root;     for (int i = 0;s[i];i++)     {         int x = s[i] - 'a';         if (p->next[x] == NULL)             return false;         p = p->next[x];     }     if (p->isword)         p->isword = false;     else         return false;     return true; } bool search(node \*root,char\* s) {     node\* p = root;     for (int i = 0;s[i];i++)     {         int x = s[i]-'a';         if (p->next[x] == NULL)             return false;         p = p->next[x];     }     return p->isword; } int count(node \*root,char \*s) {     node \*p = root;     for (int i = 0;s[i];i++)     {         int x = s[i] - 'a';         if (p->next[x] == NULL)             return 0;         p = p->next[x];     }     return p->prefix; } int main() {     m = 0;     a[m].init();     root = &a[m++];     char s[MAXM];     while (gets(s))     {         if (strcmp(s,"") == 0)             break;         insert(root,s);     }     while (gets(s))         printf("%d\n",count(root,s)); } |