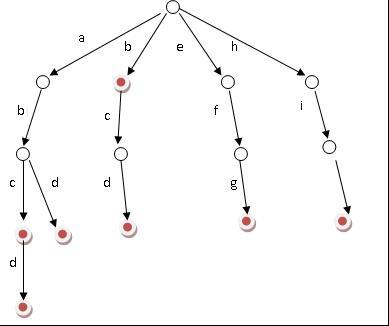
**[字典树](http://www.cnblogs.com/DiaoCow/archive/2010/04/19/1715337.html)**

今天看了字典树原理，顺便AC了几个简单的题目，做一下总结。

（字典树）

字典树的基本功能是用来查询某个单词（前缀）在所有单词中出现次数的一种数据结构，它的插入和查询复杂度都为O(len),Len为单词（前缀）长度，但是它的空间复杂度却非常高，如果字符集是26个字母，那每个节点的度就有26个，典型的以空间换时间结构。

字典树基本模板：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#define MAX 26 //字符集大小  
  
typedef struct TrieNode  
{  
 int nCount; //记录该字符出现次数  
 struct TrieNode \*next[MAX];  
}TrieNode;  
  
TrieNode Memory[1000000];  
int allocp = 0;  
  
/\*初始化\*/  
void InitTrieRoot(TrieNode \*\*pRoot)  
{  
 \*pRoot = NULL;  
}  
  
/\*创建新结点\*/  
TrieNode \*CreateTrieNode()  
{  
 int i;  
 TrieNode \*p;  
  
 p = &Memory[allocp++];  
 p->nCount = 1;  
 for(i = 0 ; i < MAX ; i++)  
 {  
 p->next[i] = NULL;  
 }  
  
 return p;  
}  
  
/\*插入\*/  
void InsertTrie(TrieNode \*\*pRoot , char \*s)  
{  
 int i , k;  
 TrieNode \*p;  
  
 if(!(p = \*pRoot))  
 {  
 p = \*pRoot = CreateTrieNode();  
 }  
 i = 0;  
 while(s[i])  
 {  
 k = s[i++] - 'a'; //确定branch  
 if(p->next[k])  
 p->next[k]->nCount++;  
 else  
 p->next[k] = CreateTrieNode();  
 p = p->next[k];  
 }  
}  
  
//查找  
int SearchTrie(TrieNode \*\*pRoot , char \*s)  
{  
 TrieNode \*p;  
 int i , k;  
  
 if(!(p = \*pRoot))  
 {  
 return 0;  
 }  
 i = 0;  
 while(s[i])  
 {  
 k = s[i++] - 'a';   
 if(p->next[k] == NULL) return 0;  
 p = p->next[k];  
 }  
 return p->nCount;  
}

1.统计难题(这里都用数组分配结点，用malloc分配太慢了）  
**这题就是统计一组字符串中某前缀出现次数（字典树第一类应用**），因此只要简单的套模板就行了（在节点中设置一个成员变量nCount，来记录该字符出现次数）

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>  
#define MAX 26   
  
typedef struct TrieNode  
{  
 int nCount;   
 struct TrieNode \*next[MAX];  
}TrieNode;  
  
TrieNode Memory[1000000];  
int allocp = 0;  
  
void InitTrieRoot(TrieNode \*\*pRoot)  
{  
 \*pRoot = NULL;  
}  
  
TrieNode \*CreateTrieNode()  
{  
 int i;  
 TrieNode \*p;  
  
 p = &Memory[allocp++];  
 p->nCount = 1;  
 for(i = 0 ; i < MAX ; i++)  
 {  
 p->next[i] = NULL;  
 }  
  
 return p;  
}  
  
void InsertTrie(TrieNode \*\*pRoot , char \*s)  
{  
 int i , k;  
 TrieNode \*p;  
  
 if(!(p = \*pRoot))  
 {  
 p = \*pRoot = CreateTrieNode();  
 }  
 i = 0;  
 while(s[i])  
 {  
 k = s[i++] - 'a'; //确定branch  
 if(p->next[k])  
 p->next[k]->nCount++;  
 else  
 p->next[k] = CreateTrieNode();  
 p = p->next[k];  
 }  
}  
  
int SearchTrie(TrieNode \*\*pRoot , char \*s)  
{  
 TrieNode \*p;  
 int i , k;  
  
 if(!(p = \*pRoot))  
 {  
 return 0;  
 }  
 i = 0;  
 while(s[i])  
 {  
 k = s[i++] - 'a';   
 if(p->next[k] == NULL) return 0;  
 p = p->next[k];  
 }  
 return p->nCount;  
}  
   
int main(void)  
{  
 char s[11];   
   
 TrieNode \*Root = NULL;   
 InitTrieRoot(&Root);   
 while(gets(s) && s[0])   
 {   
 InsertTrie(&Root , s);   
 }   
  
 while(gets(s))   
 {   
 printf("%d\n", SearchTrie(&Root , s));   
 }   
   
 return 0;  
}

2.Phone List

**这道题就是判断一组字符串中是否有一个字符串是另一个字符串的前缀（字典树第二类应用）。**

分析：我们只要在结点中添加一个nEndFlag成员变量即可。若nEndFlag == 1，说明该结点字符是某一字符串的结尾(假设为A)，若在插入B字符串的过程中经过这一结点，则说明A是B的前缀；还有一种情况，当要插入最后一个字符c时，却发现p->next[c-'0']为真，则说明该字符串是一个前缀字符串，eg:先插入abcde,再插入abc这种情况

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifhttp://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif代码

#include <stdio.h>  
#define MAX 10   
  
typedef struct TrieNode  
{  
 int nEndFlag; //标记该字符是否是某一字符串的结尾  
 struct TrieNode \*next[MAX];  
}TrieNode;  
  
TrieNode Memory[1000000];  
int allocp = 0 , nFlag = 0;  
  
void InitTrieRoot(TrieNode \*\*pRoot)  
{  
 \*pRoot = NULL;  
}  
  
TrieNode \*CreateTrieNode()  
{  
 int i;  
 TrieNode \*p;  
  
 p = &Memory[allocp++];  
 p->nEndFlag = 0;  
 for(i = 0 ; i < MAX ; i++)  
 {  
 p->next[i] = NULL;  
 }  
  
 return p;  
}  
  
void InsertTrie(TrieNode \*\*pRoot , char \*s)  
{  
 int i , k;  
 TrieNode \*p;  
  
 if(!(p = \*pRoot))  
 {  
 p = \*pRoot = CreateTrieNode();  
 }  
 i = 0;  
 while(s[i])  
 {  
 k = s[i++] - '0';   
 if(p->next[k])  
 {  
 if(p->next[k]->nEndFlag == 1 || s[i] == '\0') //先短后长 || 先长后短  
 {  
 nFlag = 1;  
 return;  
 }  
 }  
 else  
 {  
 p->next[k] = CreateTrieNode();  
 }  
 p = p->next[k];  
 }  
 p->nEndFlag = 1; //标记结尾  
}  
  
  
int main(void)  
{  
 int z , n , i;  
 char s[11];   
 TrieNode \*Root;   
  
 scanf("%d", &z);  
 while(z-- > 0)   
 {   
 nFlag = allocp = 0;  
 InitTrieRoot(&Root);   
 scanf("%d" , &n); getchar();  
 for(i = 0 ; i < n ; i++)  
 {  
 gets(s);  
 if(!nFlag) InsertTrie(&Root , s);   
 }  
 nFlag ? printf("NO\n") : printf("YES\n");  
 }   
   
 return 0;  
}