

西北工业大学计算机学院

课 程 设 计

《数据结构》

姜佳君

班号：10011104

学号：2011302490

实验报告

题目：单词（词组）检索

班级：10011104 姓名：姜佳君 学号：2011302490 完成日期：2012年1月16日

一、题目要求：

1) 基本型问题（必须采用字符串哈希，hash 散列算法）

- (1) 将所有的英文单词生成一个字典 Dictionary。
- (2) 给定一个单词，判断这个单词是否在字典 Dictionary 中。如果在单词库中，输出这个单词总共出现的次数。否则输出 NO。
- (3) 输出 Dictionary 中出现次数最高的 10 个单词。（必须采用快速排序或堆排序算法）

2) 扩展型问题（可选择合适的数据结构）

- (4) 给定一个单词，按字典序输出字典 Dictionary 中所有以这个单词为前缀的单词。例如，如果字典 $T=\{a, aa, aaa, b, ba\}$ ，如果你输入 a, 那么输出应该为 {a, aa, aaa}。
- (5) 给定一个单词，输出在 Dictionary 中以这个单词为前缀的单词的出现频率最高的 10 个单词，对于具有相同出现次数的情况，按照最近（即最后）插入的单词优先级比较高的原则输出。

3) 高级型问题

- (6) 现在我们有一些 Document，每个 Document 由一些单词组成，现在的问题就是给你一个 word，检索出哪些 Document 包含这个 word，输出这些 Document 的 DocumentID（就如同搜索引擎一样，即输入一些关键字，然后检索出和这些关键字相关的文档）。
- (7) 在第（6）问中，我们只考虑了一个 word 在哪些 Document 中的情况，我们进一步考虑 2 个相邻 word 的情况，检索出同时包含这两个相邻 word 的 DocumentID。

4) 挑战型问题

- (8) 现在我们再对（7）的问题进行扩展，把（7）中的只检索相邻 2 个 word 推广到可以检索多个 word（即连续的 k 个 word，其中 $k \geq 2$ ），检索出同时包含 k 个连续 word 的 DocumentID。

二、概要设计：

结构设计：

1) 基本型问题所用结构

```
typedef struct Node                                     //hash 表后的链表结构
{
    char    ch[50];                                     //保存单词
    int     count;                                       //保存单词出现的频数
}Node,*PNode;

typedef struct vocabulary                               //hash 表结构
{
    Node    Vocab[MAX];                                 //哈希表
    int     total;                                       //总单词个数，包含重复的
    int     kinds_num;                                   //总单词种类数
}VocabularyTable,*PVocabularyTable;
```

2) 扩展型问题所用结构

```
typedef struct Node                                     //字典树结构
{
    char    word;                                        //字母
    int     count;                                       //单词词频
    int     firsttime;                                   //插入该新单词的次序
    struct Node *next[26];                              //26 个孩子指针
}Node,*PNode,*PVocabularyTree;

typedef struct List                                     //保存前缀对应单词的 top10 链表
{
    PNode   node;                                       //节点指针
    string   Vocab;                                     //单词
    int     length;                                     //链表长度，不大于 10，否则从尾删除节点
    struct List *next;                                  //后继指针
}Linklist,*PLinklist;
```

3) 高级型问题和挑战型问题所用结构

```
typedef struct Pos
{
    long position;                                       //存放单词在文档中相对于第一个字符的位置
    int DocumentID;                                     //文档的 ID 号
    int flag;                                           //标志
    struct Pos *next;                                   //链表后继指针
}Pos,*PPos;

typedef struct Node
{
    PPos Location;                                       //单词的位置链表
    char letter;                                        //字母
    struct Node *next[26];                              //26 个孩子指针
}
```

```
}Tree,*PTree,*PNode;
```

函数设计：

1) 基本型问题所涉及函数

- (1) int Hash(char *Vocab) //哈希函数
- (2) void Init_Vocabulary(PVocabularyTable PVobTab) //初始化 hash 表内容
- (3) bool Push_VocabularyTable(char* Vocab,PVocabularyTable PVobTab) //将单词存入 hash 表
- (4) PVocabularyTable Hash_Creat_VocabularyTable() //创建一个哈希表
- (5) void SearchWordInVocabulary(PVocabularyTable PVobTab) //在哈希表中查找单词
- (6) int Q_Sort(PVocabularyTable PVobTab,int low,int high) //快速排序
- (7) void Qick_Sort(PVocabularyTable PVobTab,int low,int high,int sign) //快速排序，递归函数
- (8) void MostFrequentWord(PVocabularyTable PVobTab) //查找词频最大的前十个单词

2) 扩展型问题所涉及函数

- (1) void Initial_Node(PNode &node, char word) //初始化字典树的节点
- (2) void Creat_VocabularyTree(PVcabularyTree VT) //建立字典树
- (3) void Print_PrefixWord(string Preword,PNode CurrNode,FILE *Pre_Result)//输出前缀查找单词
- (4) void TotPrefixWord(PVcabularyTree Dic) //前缀单词查找
- (5) void Push_Linklist(PNode CurrNode,PLinklist head,string Preword) //将单词插入链表中
- (6) void Print_MostPrefixFrequence(PLinklist head,string Preword,PNode CurrNode) //递归查找
//前缀单词，并保存到链表中
- (7) void PrefixFrequence(PVcabularyTree Dic) //查找前缀对应的 top10 单词

3) 高级型问题和挑战型问题所涉及函数

- (1) void CreatDocumentTree(PTree DocTree) //建立 DocumentTree
- (2) void SearchInDocumentTree(PTree DocumentTree) //在 DocumentTree 中查找一个的单词情况
- (3) void TwoAndMultiWordInDoc(PTree DocTree,FILE *Search, FILE *Result,int Num=2,bool flag=true)
//在文档中查找两个即两个以上的单词

三、详细设计：

1) 基本型问题实现过程：

- (1) 第一问：哈希表的建立，首先我对哈希表的冲突采用链地址法实现了单词的存储和查找，但是在用快排时结构不能适应，但通过运行计算出单词的种类数大约 30000 多种，所以我采用静态的存储结构申请了 50000 大小的空间，每个节点包含单词信息和单词出现的次数。在哈希过程中采用线性探测再散列的方法。
- (2) 第二问：在第一问的基础上，第二问的查找相对简单，读入要查找的单词后经过哈希得到地址，如果遇到冲突，按照建表时的处理方法线性探测查找，如果找到，就输出单词出现的词频，如果走到字符串为空位置，说明该单词不存在，输出 NO。

- (3) 第三问：要求用快排或堆排，个人认为堆排在初始的调整阶段比较浪费时间，所以使用快排。然后将排好的哈希表中有序字符串的前 10 输出即可。

2) 扩展型问题实现过程：

- (1) 第四问：该问和第五问所用到的是键树结构，每一个节点只保存一个字符，有 26 个后继节点指针，在建立键树的过程中，如果单词重复出现，则在节点的计数器会自加一，计数器在节点的初始化时全部都置为 0；所以在查找单词的过程中计数器是否为零可以作为一个该节点所对应的单词是否在单词文件中存在。第四问的思想是查找前缀单词，那么我就根据读入的单词一个字母一个字母的解析，即在键树的根开始，按照字母走相应的后继节点，直到前缀解析完毕或是走到了一个空指针。如果前缀还没有解析完而走到了一个空指针位置，说明该前缀无对应单词，否则将该前缀解析到的节点作为查找的根节点，然后对其子树按字典序进行遍历，遇到单词节点（即单词计数器不为 0）则输入到文件。在递归过程中，我用一个 `string` 类型变量保存走到当前节点时所对应的单词，因为该节点只保存有一个字母，而整个单词是从树根到当前节点的一个路径，以此方法可将前缀对应所有单词查到。
- (2) 第五问：该问和第四问的实现过程一样，只是要求输出前缀对应单词的频数前十个，所以该程序是在第四问的基础上加以改进，即增加了一个链表，链表中保存的是字符串和对应节点指针和单词出现的频数，在插入链表中时按照频数由大到小有序插入，当频数相同时，按照建立树时单词后插入优先原则，在插入的过程中，如果发现链表中的节点大于 10，就将尾节点直接删除，当递归返回后链表中存下的就是符合要求的 `top10`（如果有 10 个），然后输入文件即可。

3) 高级型问题和挑战型问题实现过程：

- (1) 第六问：根据需要，首先我先选择了一个合适的结构，树形结构，在建立字典树的过程中，不仅保存了每一个单词所在的文档编号，同时保存了每一个单词在文档中的位置信息，用一个链表结构保存。在建立字典树的过程中选择了一个比较合适的读取方式，即每次读入一个 `Document` 的全部单词，这样就有利于单词的文档编号存储。第六问是要查找一个独立的单词在哪些文档中出现。由于在建立字典树的时候为了缩短时间复杂度，单词的文档编号是由大到小保存的，即每次插入单词的文档编号时，用的是头插法。所以在查找一个单词时如果找到了该单词，则需要将其逆序输出，在此用了一个栈来实现。先将文档的编号 `push` 到栈中（同一个文档出现多次，只 `push` 一次），再输出栈中信息即可。
- (2) 第七问和第八问：第七问和第八问有共性，都属于查找多个单词的情况，所以我只用一个函数来实现，为此，我设一个参数传递 `int num` 和一个标志 `bool flag`，其缺省值分别为 2 和 `true`，

当查找两个连续单词时该参数可以不传参，使用缺省值，在函数的具体实现过程中就会默认为两个连续单词。而对于第八问的不定数连续单词查询，首先从文件中读入单词数，然后再依次读入连续的单词进行查找。查找基于一个“基”的思想，即首先查找第一个单词，然后将第一个单词的位置信息链表作为“基”链表，然后查找下一个单词，将其位置信息链表与“基”链表的位置信息链表进行比较，两个单词是连续的则保留，如果第二个单词位置在第一个单词的两个位置之间，且两个单词位置不连续，则在“基”链表中将比较靠前的位置标记为假，即在输出时该位置不输出，以后的单词在此基础上再比较筛选，一个“查找”结束，将“基”中剩余的位置输出到文件即可。

四、调试分析与心得体会：

本次的课程设计使我学到了更多的编程知识，对于我的编程能力是一次很好的锻炼。在编程的过程中走了很多的弯路，比如在一开始时建立哈希表时用的是链地址方法处理冲突，结果到第三问的排序时出现了问题，对于相互独立的链表要如何用快排和堆排？是一个很难解决的问题，我曾想过将单词都提取出台存到一维数组中，但是这样做有一个很大的弊端就是比较消耗内存，而且相当于做了两次存储，所以又改变了存储结构，改用线性探测再散列的方法存储。在此过程中话费了很多的时间，但学到了很多，比如哈希表用链表实现和线性探测再散列方法处理冲突都实现过了，对哈希表的运用有了更深的理解。在后几个问中，还算比较顺利，而且自我认为方法还是比较有想法，比如前缀查找函数进行传字符串递归查找，多连续单词查找与两个单词查找的完美结合，收获了很多，也很有成就感。

五、用户操作说明：

只要运行程序即可，无需任何输入，但应存在一些文件，我的程序中会附带，只要将其与程序代码放到同一路径下即可。

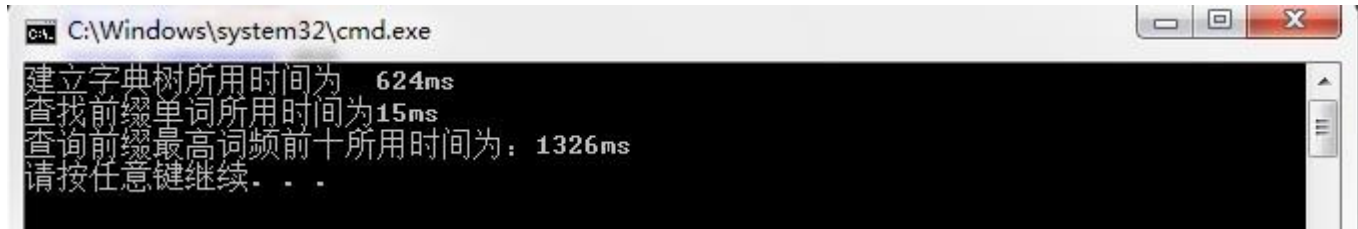
六、运行结果：

第 1-3 问截图：



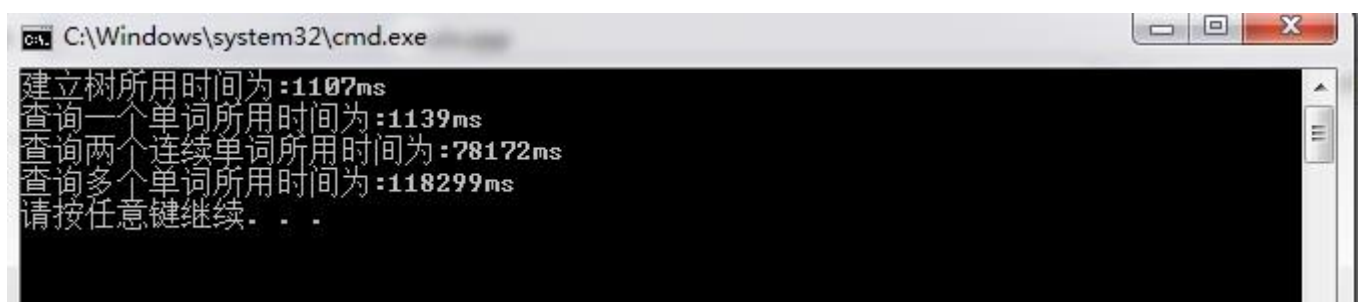
```
C:\Windows\system32\cmd.exe
建立哈希表所用时间: 655ms
查找单词所用时间: 296ms
单词排序所用时间: 0ms
请按任意键继续. . .
```

第 4-5 问截图：



```
C:\Windows\system32\cmd.exe
建立字典树所用时间为 624ms
查找前缀单词所用时间为15ms
查询前缀最高词频前十所用时间为：1326ms
请按任意键继续. . .
```

第 6-8 问截图：



```
C:\Windows\system32\cmd.exe
建立树所用时间为:1107ms
查询一个单词所用时间为:1139ms
查询两个连续单词所用时间为:78172ms
查询多个单词所用时间为:118299ms
请按任意键继续. . .
```

七、附录：

由于电脑原因时间运行时间只作为参考，源程序见附件。