词频统计的一些做法

主要难点在大词典的读入与存储，和大文本在大词典中的搜索。
个人有几个思路，整理归纳一下。

存储

普通的数组直接pass□链表搜索起来太慢。

二叉查找树

可以把搜索的时间复杂度降到logn□
不过需要注意的是读入的字典是有序的，在创建二叉查找树时不经过一些处理会构建成最劣的链式结构。
因此需要应用到平衡二叉树的一些手段，具体可以参考自平衡二叉查找树
不过字典的情况比较特殊，字典是严格有序的，不需要运用到上面那些复杂的结构，有比较简单的处理方法
有序链表自然地将小元素划分到了左边，大元素划分到了右边。我们只要找出中间节点提为父节点即可。
找中间节点的方法也有很多，我这里用了快慢指针。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include<ctype.h>
#define MAXWORD 32
int n[3]={0},now=0;

struct lnode{
    char word[MAXWORD];
    int count;
    struct lnode* next;
};
typedef struct lnode lNode;
typedef struct lnode* lNodeptr;

lNodeptr wl[3],r[3];

struct tnode{
    char word[MAXWORD];
    int count;
    struct tnode* left;
    struct tnode* right;
};
typedef struct tnode tNode;
typedef struct tnode* tNodeptr;

int getWord(FILE *fp,char *w)
{
    int c;
```

```
    while(!isalpha(c=fgetc(fp)))
        if(c==EOF) return c;
        else continue;
    do{
        *w++=tolower(c);
    }while(isalpha(c=fgetc(fp)));
    *w='\0';
    return 1;
}

tNodeptr ListToBst(lNodeptr head, lNodeptr tail){
    if (head==tail) return NULL;
    lNodeptr fast=head;
    lNodeptr slow=head;
    while (fast!=tail&&fast->next!=tail)
    {
        slow=slow->next;
        fast=fast->next->next;
    }
    tNodeptr root=(tNodeptr)malloc(sizeof(tNode));
    strcpy(root->word,slow->word);
    root->left=ListToBst(head,slow);
    root->right=ListToBst(slow->next,tail);
    return root;
}

int isword(tNodeptr list,char *w)
{
    if (strcmp(list->word,w)==0) return 1;
    else if (strcmp(list->word,w)<0&&list->right!=NULL) return
isword(list->right,w);
    else if (strcmp(list->word,w)>0&&list->left!=NULL) return
isword(list->left,w);
    return -1;
}

void insert(tNodeptr r,char x[1000],int t)
{
    if (strcmp(r->word,x)==0){
        r->count++;
    }
    else if (strcmp(r->word,x)>0) {
        if (r->left==NULL) {
            n[t]++;
            tNodeptr p=(tNodeptr)malloc(sizeof(tNode));
            strcpy(p->word,x);
            p->count=1;
            p->left=NULL;
            p->right=NULL;
            r->left=p;
```

```
        }
        else insert(r->left,x,t);
    }
    else {
        if (r->right==NULL) {
            n[t]++;
            tNodeptr p=(tNodeptr)malloc(sizeof(tNode));
            p->count=1;
            strcpy(p->word,x);
            p->left=NULL;
            p->right=NULL;
            r->right=p;
        }
        else insert(r->right,x,t);
    }
}

void search(tNodeptr r,int n,int t)
{
    if (r!=NULL){
        search(r->left,n,t);
        if (wl[t]==NULL) {
            wl[t]=(lNodeptr)malloc(sizeof(lNode));
            strcpy(wl[t]->word,r->word);
            wl[t]->count=r->count;
            wl[t]->next=NULL;
        }
        else {
            int i=0;
            lNodeptr p=wl[t],pr;
            lNodeptr tmp=(lNodeptr)malloc(sizeof(lNode));
            strcpy(tmp->word,r->word);
            tmp->count=r->count;
            while (p->next!=NULL&&p->count>=r->count&&i<n)
            {
                pr=p;
                p=p->next;
                i++;
            }
            if (p->next==NULL) {
                if (p->count>=r->count)
                {
                    p->next=tmp;
                    tmp->next=NULL;
                }
                else if (p==wl[t]){
                    tmp->next=p;
                    wl[t]=tmp;
                }
                else {
                    pr->next=tmp;
```

```
                    tmp->next=p;
                }
            }
            else if (p==wl[t]){
                tmp->next=p;
                wl[t]=tmp;
            }
            else {
                pr->next=tmp;
                tmp->next=p;
            }
        }
        search(r->right,n,t);
    }
}

double min1(double x,double y)
{
    if (x<y) return x;
    else return y;
}

double max1(double x,double y)
{
    if (x>y) return x;
    else return y;
}

int main()
{
    FILE *fp1,*fp2,*fstop,*dic,*res;
    fp1=fopen("article1.txt","r");
    fp2=fopen("article2.txt","r");
    fstop=fopen("stopwords.txt","r");
    dic=fopen("dictionary.txt","r");
    res=fopen("results.txt","w");
    lNodeptr Wordlist,Wordlists,end;
    Wordlist=NULL;
    Wordlists=NULL;
    char word[MAXWORD];
    int nn;
    scanf("%d",&nn);
    while (getWord(dic,word)!=EOF)
    {
        if (Wordlist==NULL) {
            Wordlist=(lNodeptr)malloc(sizeof(lNode));
            strcpy(Wordlist->word,word);
            Wordlist->next=NULL;
            end=Wordlist;
        }
```

```
        else {
            lNodeptr p=(lNodeptr)malloc(sizeof(lNode));
            strcpy(p->word,word);
            p->next=NULL;
            end->next=p;
            end=p;
        }
    }
    lNodeptr ends;
    while (getWord(fstop,word)!=EOF)
    {
        if (Wordlists==NULL) {
            Wordlists=(lNodeptr)malloc(sizeof(lNode));
            strcpy(Wordlists->word,word);
            Wordlists->next=NULL;
            ends=Wordlists;
        }
        else {
            lNodeptr p=(lNodeptr)malloc(sizeof(lNode));
            strcpy(p->word,word);
            p->next=NULL;
            ends->next=p;
            ends=p;
        }
    }
    tNodeptr wordlist,wordlists;
    wordlist=ListToBst(Wordlist, end->next);
    wordlists=ListToBst(Wordlists, end->next);
    tNodeptr wordlist1,wordlist2;
    if(fp1!=NULL)
    {
        while(getWord(fp1,word)!=EOF){
            if(isword(wordlist,word)==1&&isword(wordlists,word)==-1)
            {
                if (n[1]==0)
                {
                    wordlist1=(tNodeptr)malloc(sizeof(tNode));
                    strcpy(wordlist1->word,word);
                    wordlist1->count=1;
                    wordlist1->left=NULL;
                    wordlist1->right=NULL;
                    n[1]=1;
                }
                else insert(wordlist1,word,1);
            }
        }
    }
    if(fp2!=NULL)
    {
        while(getWord(fp2,word)!=EOF){
            if(isword(wordlist,word)==1&&isword(wordlists,word)==-1)
```

```
            {
                if (n[2]==0)
                {
                    wordlist2=(tNodeptr)malloc(sizeof(tNode));
                    strcpy(wordlist2->word,word);
                    wordlist2->count=1;
                    wordlist2->left=NULL;
                    wordlist2->right=NULL;
                    n[2]=1;
                }
                else insert(wordlist2,word,2);
            }
        }
    }
    int n1=min1(nn,n[1]);
    int n2=min1(nn,n[2]);
    search(wordlist1,n1,1);
    now=0;
    search(wordlist2,n2,2);
    int freq1=0,freq2=0,freqm1=0,freqm2=0;
    double pro1,pro2;
    int i=0;
    for (lNodeptr r=wl[1];i<n1;r=r->next,i++)
    {
        freq1=freq1+r->count;
        int j=0;
        for (lNodeptr rr=wl[2];j<n2;rr=rr->next,j++)
            if (strcmp(r->word,rr->word)==0){
                freqm1=freqm1+r->count;
                freqm2=freqm2+rr->count;
            }
    }
    i=0;
    for (lNodeptr r=wl[2];i<n2;r=r->next,i++) freq2=freq2+r->count;
    pro1=(double)freqm1/(double)freq1;
    pro2=(double)freqm2/(double)freq2;
    double sim=min1(pro1,pro2)/max1(pro1,pro2);
    printf("%.5lf",sim);
    fprintf(res,"%3.5lf\n\n",sim);
    i=0;
    for (lNodeptr r=wl[1];i<n1;r=r->next,i++) fprintf(res,"%s
%d\n",r->word,r->count);
    fprintf(res,"\n");
    i=0;
    for (lNodeptr r=wl[2];i<n2;r=r->next,i++) fprintf(res,"%s
%d\n",r->word,r->count);
    getchar();
    getchar();
    return 0;
```

```
}
```

不过实测下来效果并不理想，算上比较字符串的时间即使是logn也是不可接受的。

Trie树

应对大词典的一个算法，以字母作为节点进行存储，节省存储空间的同时也能加快存储速度。
时间上只与最长单词长度相关。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include<ctype.h>
#define MAXWORD 32
#define MULT  37
int n[3]={0},now=0;

struct lnode{
    char word[MAXWORD];
    int count;
    struct lnode* next;
};
typedef struct lnode lNode;
typedef struct lnode* lNodeptr;

struct trie{
    char isword;
    char isleaf;
    struct trie *ptr[26];
};
typedef struct trie Trie;
typedef struct trie* Trieptr;

Trieptr root,roots;

lNodeptr wl[3],r[3];

struct tnode{
    char word[MAXWORD];
    int count;
    struct tnode* left;
    struct tnode* right;
};
typedef struct tnode tNode;
typedef struct tnode* tNodeptr;

int getWord(FILE *fp,char *w)
{
```

```
    int c;
    while(!isalpha(c=fgetc(fp)))
        if(c==EOF) return c;
        else continue;
    do{
        *w++=tolower(c);
    }while(isalpha(c=fgetc(fp)));
    *w='\0';
    return 1;
}

int isword(Trieptr list,char *w)
{
    Trieptr p;
    for (p=list;*w!='\0';w++){
        if (p->ptr[*w-'a']==NULL) return -1;
        p=p->ptr[*w-'a'];
    }
    if (p->isword==1) return 1;
    else return -1;
}

void insert(tNodeptr r,char x[1000],int t)
{
    if (strcmp(r->word,x)==0){
        r->count++;
    }
    else if (strcmp(r->word,x)>0) {
        if (r->left==NULL) {
            n[t]++;
            tNodeptr p=(tNodeptr)malloc(sizeof(tNode));
            strcpy(p->word,x);
            p->count=1;
            p->left=NULL;
            p->right=NULL;
            r->left=p;
        }
        else insert(r->left,x,t);
    }
    else {
        if (r->right==NULL) {
            n[t]++;
            tNodeptr p=(tNodeptr)malloc(sizeof(tNode));
            p->count=1;
            strcpy(p->word,x);
            p->left=NULL;
            p->right=NULL;
            r->right=p;
        }
        else insert(r->right,x,t);
```

```
    }
}

void search(tNodeptr r,int n,int t)
{
    if (r!=NULL){
        search(r->left,n,t);
        if (wl[t]==NULL) {
            wl[t]=(lNodeptr)malloc(sizeof(lNode));
            strcpy(wl[t]->word,r->word);
            wl[t]->count=r->count;
            wl[t]->next=NULL;
        }
        else {
            int i=0;
            lNodeptr p=wl[t],pr;
            lNodeptr tmp=(lNodeptr)malloc(sizeof(lNode));
            strcpy(tmp->word,r->word);
            tmp->count=r->count;
            while (p->next!=NULL&&p->count>=r->count&&i<n)
            {
                pr=p;
                p=p->next;
                i++;
            }
            if (p->next==NULL) {
                if (p->count>=r->count)
                {
                    p->next=tmp;
                    tmp->next=NULL;
                }
                else if (p==wl[t]){
                    tmp->next=p;
                    wl[t]=tmp;
                }
                else {
                    pr->next=tmp;
                    tmp->next=p;
                }
            }
            else if (p==wl[t]){
                tmp->next=p;
                wl[t]=tmp;
            }
            else {
                pr->next=tmp;
                tmp->next=p;
            }
        }
        search(r->right,n,t);
    }
```

```
}

double min1(double x,double y)
{
    if (x<y) return x;
    else return y;
}

double max1(double x,double y)
{
    if (x>y) return x;
    else return y;
}

Trieptr talloc()
{
    int i;
    Trieptr p;
    p=(Trieptr)malloc(sizeof(Trie));
    p->isword=0;
    p->isleaf=1;
    for (i=0;i<26;i++)
        p->ptr[i]=NULL;
    return p;
}

void wordTree(Trieptr root,char *w)
{
    Trieptr p;
    for (p=root;*w!='\0';w++){
        if (p->ptr[*w-'a']==NULL){
            p->ptr[*w-'a']=talloc();
            p->isleaf=0;
        }
        p=p->ptr[*w-'a'];
    }
    p->isword=1;
}

int main()
{
    FILE *fp1,*fp2,*fstop,*dic,*res;
    fp1=fopen("article1.txt","r");
    fp2=fopen("article2.txt","r");
    fstop=fopen("stopwords.txt","r");
    dic=fopen("dictionary.txt","r");
    res=fopen("results.txt","w");
    char word[MAXWORD];
    int nn;
    scanf("%d",&nn);
```

```
    root=talloc();
    roots=talloc();
    while (getWord(dic,word)!=EOF)
    {
        wordTree(root,word);
    }
    while (getWord(fstop,word)!=EOF)
    {
        wordTree(roots,word);
    }
    tNodeptr wordlist1,wordlist2;
    if(fp1!=NULL)
    {
        while(getWord(fp1,word)!=EOF){
            if(isword(root,word)==1&&isword(roots,word)==-1)
            {
                if (n[1]==0)
                {
                    wordlist1=(tNodeptr)malloc(sizeof(tNode));
                    strcpy(wordlist1->word,word);
                    wordlist1->count=1;
                    wordlist1->left=NULL;
                    wordlist1->right=NULL;
                    n[1]=1;
                }
                else insert(wordlist1,word,1);
            }
        }
    }
    if(fp2!=NULL)
    {
        while(getWord(fp2,word)!=EOF){
            if(isword(root,word)==1&&isword(roots,word)==-1)
            {
                if (n[2]==0)
                {
                    wordlist2=(tNodeptr)malloc(sizeof(tNode));
                    strcpy(wordlist2->word,word);
                    wordlist2->count=1;
                    wordlist2->left=NULL;
                    wordlist2->right=NULL;
                    n[2]=1;
                }
                else insert(wordlist2,word,2);
            }
        }
    }
    int n1=min1(nn,n[1]);
    int n2=min1(nn,n[2]);
    search(wordlist1,n1,1);
    now=0;
```

```
    search(wordlist2,n2,2);
    int freq1=0,freq2=0,freqm1=0,freqm2=0;
    double pro1,pro2;
    int i=0;
    for (lNodeptr r=wl[1];i<n1;r=r->next,i++)
    {
        freq1=freq1+r->count;
        int j=0;
        for (lNodeptr rr=wl[2];j<n2;rr=rr->next,j++)
            if (strcmp(r->word,rr->word)==0){
                freqm1=freqm1+r->count;
                freqm2=freqm2+rr->count;
            }
    }
    i=0;
    for (lNodeptr r=wl[2];i<n2;r=r->next,i++) freq2=freq2+r->count;
    pro1=(double)freqm1/(double)freq1;
    pro2=(double)freqm2/(double)freq2;
    double sim=min1(pro1,pro2)/max1(pro1,pro2);
    printf("%.5lf",sim);
    fprintf(res,"%3.5lf\n\n",sim);
    i=0;
    for (lNodeptr r=wl[1];i<n1;r=r->next,i++) fprintf(res,"%s
%d\n",r->word,r->count);
    fprintf(res,"\n");
    i=0;
    for (lNodeptr r=wl[2];i<n2;r=r->next,i++) fprintf(res,"%s
%d\n",r->word,r->count);
    getchar();
    getchar();
    return 0;
}
```

理应是最快的，但也许是我优化的不够好吧，实际效果比不上hash

Hash

比较容易想到的一个方法，做起来也比较简单。
用一定的算法将单词转为数字，存在hash表里，达到理论上O(1)的速度，不过实际上会有重复，需要一个链表来存储重复单词。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include<ctype.h>
#define MAXWORD 32
#define NHASH  1000000
#define MULT  37
```

```
int n[3]={0},now=0;

struct lnode{
    char word[MAXWORD];
    int count;
    struct lnode* next;
};
typedef struct lnode lNode;
typedef struct lnode* lNodeptr;

lNodeptr hash[1000010],hashstop[1000010];

lNodeptr wl[3],r[3];

struct tnode{
    char word[MAXWORD];
    int count;
    struct tnode* left;
    struct tnode* right;
};
typedef struct tnode tNode;
typedef struct tnode* tNodeptr;

int getWord(FILE *fp,char *w)
{
    int c;
    while(!isalpha(c=fgetc(fp)))
        if(c==EOF) return c;
        else continue;
    do{
        *w++=tolower(c);
    }while(isalpha(c=fgetc(fp)));
    *w='\0';
    return 1;
}

int isword(lNodeptr *list,char *w)
{
    char *p;
    unsigned int h=0;
    for (p=w;*p!='\0';p++)
        h= MULT*h+*p;
    h=h%NHASH;
    if (list[h]==NULL) return -1;
    else {
        lNodeptr p=list[h];
        while (p!=NULL&&strcmp(p->word,w)<0)
        {
            p=p->next;
        }
        if (p==NULL) return -1;
```

```
        else if (strcmp(p->word,w)>0) return -1;
        else if (strcmp(p->word,w)==0) return 1;
    }
    return -1;
}

void insert(tNodeptr r,char x[1000],int t)
{
    if (strcmp(r->word,x)==0){
        r->count++;
    }
    else if (strcmp(r->word,x)>0) {
        if (r->left==NULL) {
            n[t]++;
            tNodeptr p=(tNodeptr)malloc(sizeof(tNode));
            strcpy(p->word,x);
            p->count=1;
            p->left=NULL;
            p->right=NULL;
            r->left=p;
        }
        else insert(r->left,x,t);
    }
    else {
        if (r->right==NULL) {
            n[t]++;
            tNodeptr p=(tNodeptr)malloc(sizeof(tNode));
            p->count=1;
            strcpy(p->word,x);
            p->left=NULL;
            p->right=NULL;
            r->right=p;
        }
        else insert(r->right,x,t);
    }
}

void search(tNodeptr r,int n,int t)
{
    if (r!=NULL){
        search(r->left,n,t);
        if (wl[t]==NULL) {
            wl[t]=(lNodeptr)malloc(sizeof(lNode));
            strcpy(wl[t]->word,r->word);
            wl[t]->count=r->count;
            wl[t]->next=NULL;
        }
        else {
            int i=0;
            lNodeptr p=wl[t],pr;
```

```
            lNodeptr tmp=(lNodeptr)malloc(sizeof(lNode));
            strcpy(tmp->word,r->word);
            tmp->count=r->count;
            while (p->next!=NULL&&p->count>=r->count&&i<n)
            {
                pr=p;
                p=p->next;
                i++;
            }
            if (p->next==NULL) {
                if (p->count>=r->count)
                {
                    p->next=tmp;
                    tmp->next=NULL;
                }
                else if (p==wl[t]){
                    tmp->next=p;
                    wl[t]=tmp;
                }
                else {
                    pr->next=tmp;
                    tmp->next=p;
                }
            }
            else if (p==wl[t]){
                tmp->next=p;
                wl[t]=tmp;
            }
            else {
                pr->next=tmp;
                tmp->next=p;
            }
            //printf("%d\n",now);
        }
        search(r->right,n,t);
    }
}

double min1(double x,double y)
{
    if (x<y) return x;
    else return y;
}

double max1(double x,double y)
{
    if (x>y) return x;
    else return y;
}

int main()
```

```
{
    FILE *fp1,*fp2,*fstop,*dic,*res;
    fp1=fopen("article1.txt","r");
    fp2=fopen("article2.txt","r");
    fstop=fopen("stopwords.txt","r");
    dic=fopen("dictionary.txt","r");
    res=fopen("results.txt","w");
    char word[MAXWORD];
    int nn;
    scanf("%d",&nn);
    while (getWord(dic,word)!=EOF)
    {
        char *p;
        unsigned int h=0;
        for (p=word;*p!='\0';p++)
            h= MULT*h+*p;
        h=h%NHASH;
        if (hash[h]==NULL)
        {
            hash[h]=(lNodeptr)malloc(sizeof(lNode));
            strcpy(hash[h]->word,word);
            hash[h]->next=NULL;
        }
        else {
            lNodeptr p=hash[h];
            lNodeptr q=(lNodeptr)malloc(sizeof(lNode));;
            strcpy(q->word,word);
            if (strcmp(p->word,word)>0)
            {
                q->next=hash[h];
                hash[h]=q;
            }
            else {
                lNodeptr r=p;
                while (p!=NULL&&strcmp(p->word,word)<0)
                {
                    r=p;
                    p=p->next;
                }
                if (p==NULL) {
                    r->next=q;
                    q->next=NULL;
                }
                else {
                    q->next=r->next;
                    r->next=q;
                }
            }
        }
    }
```

```
    while (getWord(fstop,word)!=EOF)
    {
        char *p;
        unsigned int h=0;
        for (p=word;*p!='\0';p++)
            h= MULT*h+*p;
        h=h%NHASH;
        if (hashstop[h]==NULL)
        {
            hashstop[h]=(lNodeptr)malloc(sizeof(lNode));
            strcpy(hashstop[h]->word,word);
            hashstop[h]->next=NULL;
        }
        else {
            lNodeptr p=hashstop[h];
            lNodeptr q=(lNodeptr)malloc(sizeof(lNode));;
            strcpy(q->word,word);
            if (strcmp(p->word,word)>0)
            {
                q->next=hashstop[h];
                hashstop[h]=q;
            }
            else {
                lNodeptr r=p;
                while (p!=NULL&&strcmp(p->word,word)<0)
                {
                    r=p;
                    p=p->next;
                }
                if (p==NULL) {
                    r->next=q;
                    q->next=NULL;
                }
                else {
                    q->next=r->next;
                    r->next=q;
                }
            }
        }
    }
    tNodeptr wordlist1,wordlist2;
    if(fp1!=NULL)
    {
        while(getWord(fp1,word)!=EOF){
            if(isword(hash,word)==1&&isword(hashstop,word)==-1)
            {
                if (n[1]==0)
                {
                    wordlist1=(tNodeptr)malloc(sizeof(tNode));
                    strcpy(wordlist1->word,word);
                    wordlist1->count=1;
```

```
                    wordlist1->left=NULL;
                    wordlist1->right=NULL;
                    n[1]=1;
                }
                else insert(wordlist1,word,1);
            }
        }
    }
    if(fp2!=NULL)
    {
        while(getWord(fp2,word)!=EOF){
            if(isword(hash,word)==1&&isword(hashstop,word)==-1)
            {
                if (n[2]==0)
                {
                    wordlist2=(tNodeptr)malloc(sizeof(tNode));
                    strcpy(wordlist2->word,word);
                    wordlist2->count=1;
                    wordlist2->left=NULL;
                    wordlist2->right=NULL;
                    n[2]=1;
                }
                else insert(wordlist2,word,2);
            }
        }
    }
    int n1=min1(nn,n[1]);
    int n2=min1(nn,n[2]);
    search(wordlist1,n1,1);
    now=0;
    search(wordlist2,n2,2);
    int freq1=0,freq2=0,freqm1=0,freqm2=0;
    double pro1,pro2;
    int i=0;
    for (lNodeptr r=wl[1];i<n1;r=r->next,i++)
    {
        freq1=freq1+r->count;
        int j=0;
        for (lNodeptr rr=wl[2];j<n2;rr=rr->next,j++)
            if (strcmp(r->word,rr->word)==0){
                freqm1=freqm1+r->count;
                freqm2=freqm2+rr->count;
            }
    }
    i=0;
    for (lNodeptr r=wl[2];i<n2;r=r->next,i++) freq2=freq2+r->count;
    pro1=(double)freqm1/(double)freq1;
    pro2=(double)freqm2/(double)freq2;
    double sim=min1(pro1,pro2)/max1(pro1,pro2);
    printf("%.5lf",sim);
```

```
    fprintf(res,"%3.5lf\n\n",sim);
    i=0;
    for (lNodeptr r=wl[1];i<n1;r=r->next,i++) fprintf(res,"%s
%d\n",r->word,r->count);
    fprintf(res,"\n");
    i=0;
    for (lNodeptr r=wl[2];i<n2;r=r->next,i++) fprintf(res,"%s
%d\n",r->word,r->count);
    getchar();
    getchar();
    return 0;
}
```

查找

查找的方法依据存储方法而定。特别的是这次有两个字典（dictionary与stopwords）需要去除dictionary中的stopwords。
我个人的处理办法是将两个单词表都记录下来，对于文本中的一个单词在两个表中都执行搜索操作，若在dictionary中而不在stopwords中则判断为合法单词。
这个应该不是时间上最快的处理方法，但在编写上是相当简单的。
对于文本的存储方法我采用的是简单的二叉查找树，因为文本量比起字典量来说小很多，所以不同方法理应影响不大（大概），时间原因更好地方法没有来得及编写。