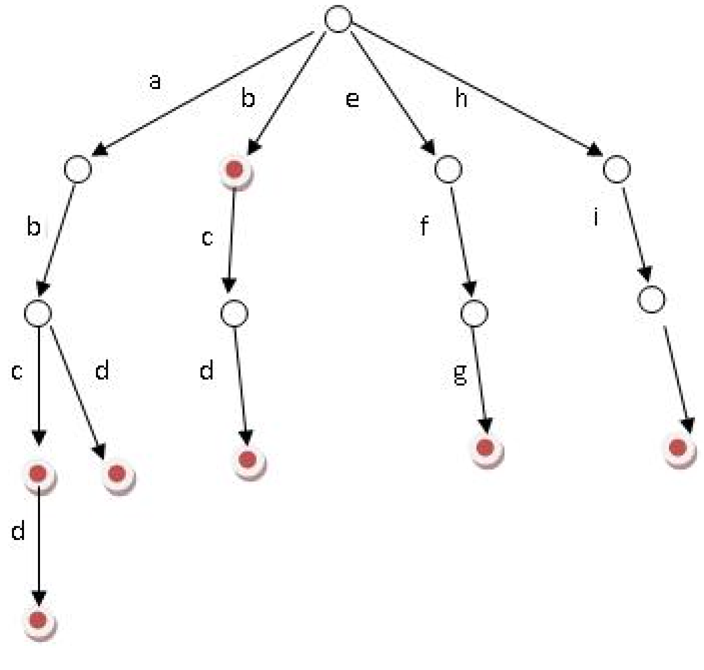
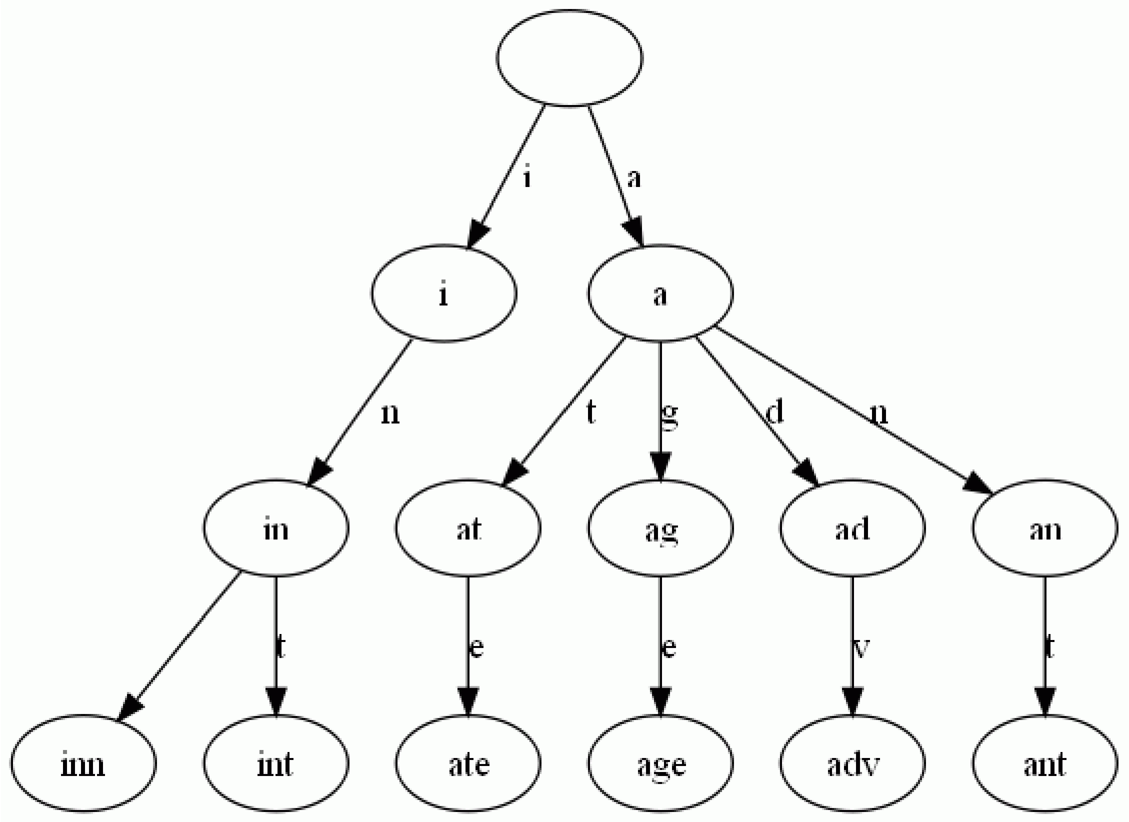
PA5实验报告

1.涉及数据结构和相关背景

- 字典树
- 定义
    - Trie树，即字典树，又称单词查找树或键树，是一种树形结构。典型应用是用于统计和排序大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是最大限度地减少无谓的字符串比较，查询效率比较高。
    - Trie的核心思想是空间换时间，利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。
- 基本性质
    - 根节点不包含字符，除根节点外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。



- 这样一来我们查询和插入可以一起完成，所用时间仅仅为单词长度
- 若文本是仅有26个字母组成的排列，则我们可以看到，trie树每一层的节点数是26^i级别的。所以为了节省空间，我们还可以用动态链表，或者用数组来模拟动态。而空间的花费，不会超过单词数×单词长度。



- 查询
    - 每条边对应一个字母。
    - 每个节点对应一项前缀。叶节点对应最长前缀，即单词本身。
    - 同时进行单词结束字符的标记，防止出现截短单词的匹配

2. 实验内容

2.1 问题描述

- 给一篇超过100000字的英文文章，你能统计出里面每一个单词出现的次数吗?
- 要求利用Trie树实现

2.2 基本要求

- 输入
- 一个纯ASCII字符组成的文本文件，统计里面每个单词出现的次数
- 单词仅由大、小写英文字母组成，处理时将所有大写字母转换为小写字母
- 当出现连接符'-'连接两个单词时，如果'-'后跟的是换行符，可以视为'-'前后是一个因换行而被拆开的单词（即删除'-'并将'-'前后连接起来成为一个单词），否则视为两个单词。
- 使用字典树作为基本数据结构

- 输出
- 输出为若干形式为<key,value>的键值对，其中键为文本中出现的单词，值为这个单词出现的次数
- 每一行输出一个键值对，输出按键的字典序升序排序

2.3 数据结构设计

- 利用字典树进行该统计词频功能，将该单词的出现次数放在该单词结束的节点处储存(不一定是叶子节点)
- `Node`节点类

```
struct Node
{
    char ch;                    //存储的字符
    bool end;                   //是否为单词结尾的标记
    string word = "";           //如果是单词结尾的话此处记录该单词
    Node* next[30];             //指针域
    int idx;                    //记录孩子节点个数
    int freq;                   //若为单词结尾记录单词出现频次
    int vis;                    //记录当前节点是否被访问过
    Node(char ch1=' ', bool ende = false, string worde = "", int index=0,
int fre=0, int visited=0) : ch(ch1), end(ende), word(worde), idx(index),
freq(fre), vis(visited) {
        for (int i = 0; i < 30; i++)
        {
            next[i] = NULL;     //初始化指针域
        }
    }
};
```

- `Trie`树类
- 方法集

```
TrieTree();                                                     //构造函数

int Insert(string str);                                         //插入函数

int Search(string str, int& f, int mode);                       //搜索函数

int findWord(string str, int& f);                               //搜索单词函数

int findPrefix(string prefix, int& f);                          //搜索前缀函数

class _Pred                                                     //频率排序谓词
{
    public:
    bool operator()(pair<string, int> p1, pair<string, int> p2)
    {
        return p1.second > p2.second;
    }
};

void counter(Node* root);                                       //记录单词频次

void deleter(Node* root);                                       //删除树与所有
子节点
```

```
void preDestructor();                                        //删除树所有附属信息

~TrieTree();                                                 //析构函数
```

- 属性集

```
Node* root_;

pair<string, int>* freqTable = new pair<string, int>[2000000];    //已知英语单词超过1000000个
```

- 方法实现
- 构造函数

```
TrieTree::TrieTree()
{
    this->root_ = new Node();
    this->num = 0;
}
```

- 插入函数

```
int TrieTree::Insert(string str)
{
    Node* p = this->root_;
    //转换为小写字母
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] >= 65 && str[i] <= 90)
            str[i] += 32;
    }

    int found = 0;
    for (int i = 0; i < str.length(); i++)      //依次在树中寻找插入的单词中的每一个字符
    {
        found = 0;
        for (int j = 0; j < 26 ; j++)
        {
            //此处的遍历从指针域的小下标开始，对应字典序
            if (p->next[j] != NULL && p->next[j]->ch == str[i])
            {
                found = 1;
                p = p->next[j];
                break;                          //在下一层找到了当前字母，进入这个节点继续向下寻找
            }
        }
        if (!found)                             //没有找到就创建当前节点，添加到树中
        {
            Node* node = new Node(str[i], false, "", 0);
            if (node == NULL)
                return 0;

            p->next[str[i] - 'a'] = node;
            p->idx++;
            p = node;

        }
    }
    p->end = true;                              //标记此为单词结尾
    p->word = str;                              //记录单词
    p->freq++;                                  //更新频次
    return 1;
}
```

> 这里遍历的顺序是按照指针域的下标顺序遍历，代表着a ~ z的字典序，这样对应输出时为字典序

- 搜索函数（搜索单词与搜索前缀复用）

```
int TrieTree::Search(string str, int& f, int mode)
{
    /*mode : 搜索单词和搜索前缀的不同模式*/
    //转换为小写字母
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] >= 65 && str[i] <= 90)
            str[i] += 32;
    }

    Node* p = this->root_;
    if (p == NULL)
        return 0;
    int found = 0;
    for (int i = 0; i < str.length(); i++)
    {
        found = 0;
        //遍历指针域进行搜索
        for (int j = 0; j < 26; j++)
        {
            if ( p->next[j] != NULL && p->next[j]->ch == str[i])
            {
                found = 1;
                p = p->next[j];
                break;
            }
        }
        if (!found)                         //有一个位置的字母没有找到就说明该单词
不存在
        {
            f = -1;
            return 0;
        }
    }

    //此处加一个判断是为了防止出现在存储时存储的是长单词，但是在查找时查找的是该单词的截短单词
    //比如存储的是Tongji，查询的时候查的是Tong，需要检查此处是否是单词结尾
    if (p->end == 0 && mode)
    {
        f = -1;
        return 0;
    }

    f = p->freq;
    return 1;
}
```

> 注意检查单词结尾的问题，当查询原单词的截短单词的时候需要检查是否几位出是一个单词的结尾

- 查询单词/前缀

```
int TrieTree::findWord(string str, int& f)
{
    return this->Search(str, f, 1);

}

int TrieTree::findPrefix(string prefix, int& f)
{
    return this->Search(prefix, f, 0);
}
```

- 查询单词出现个数

```
void TrieTree::counter(Node* root)
{
    if (root->idx == 0)                     //到达叶子节点
    {
        this->freqTable[this->num++] = { root->word , root->freq };
        return;
    }

    if (root->end)                          //到达一个单词的结尾
    {
        this->freqTable[this->num++] = { root->word , root->freq };
    }                                       //将单词记录在频率表中

    for (int i = 0; i < 26; i++)
    {
        if (root->next[i] != NULL)
```

```
            this->counter(root->next[i]);
    }
}
```

- 递归删除该树的所有节点以及频率表

```
void TrieTree::deleter(Node* root)
{
    for (int i = 0; i < 26; i++)
    {
        if (root->next[i] != NULL)
            deleter(root->next[i]);
    }
    delete root;

}

void TrieTree::preDestructor()
{
    //后序遍历删除节点，删除存储列表
    this->deleter(this->root_);
    delete[] this->freqTable;
}

//析构函数
TrieTree::~TrieTree()
{
    this->preDestructor();

}
```

2.4 功能说明

- 命令行指令集

```
string commandSet[] = { "load", "dic", "stat", "sort"};
```

- 读取文件

```
int readFile(string path, string& out)
{
    ifstream readFrom;
    readFrom.open(path, ios::in);
    if (readFrom.is_open() == 0)
        return 0;
    string temp = "";
    while (readFrom.good())
    {
        getline(readFrom, temp);
        if (temp.length() && temp[temp.length() - 1] == '-')
```

```
            temp = temp.substr(0, temp.length() - 1);
        if (temp.length())
            out += temp;
    }
    readFrom.close();
    return 1;

}

int buildTree(string src, TrieTree* tree)
{
    cout << endl << endl;
    for (int i = 0; i < src.length(); i++)
    {
        string temp = "";

        //连字符不能在单词中出现
        while (i < src.length() && (src[i] >= 'A' && src[i] <= 'Z' || src[i] >= 'a' && src[i] <= 'z'))  // || src[i] == '-'  //连字符字典序遍历就失效了
        {
            temp += src[i++];
        }

        if (temp.size())
        {
            tree->Insert(temp);
        }
    }

    return 1;
}
```

> 单词中不能出现连字符，若出现则无法按照字典序进行遍历
>
> 若需要实现则需要将每个存储节点扩容二倍，在每两个字母之间添加`-`，保证字典序输出

- 命令行检测

```
int command(string& ope)
{
    cout << ">>>>";
    char ch = ' ';
    if ((ch = _getch()) == '\n')
        return -1;
    else
    {
        ope += ch;
        cout << ch;
    }

    while (1)
    {
        ch = _getch();
        if (ch == 13)
        {
            cout << endl;
            return 1;
        }

        int x = 0, y = 0;
        cct_getxy(x, y);
        if (ch == 8 && ope.length() && x >= 4)
        {
            ope = ope.substr(0, ope.length() - 1);

            cct_gotoxy(x - 1, y);
            cout << " ";
            cct_gotoxy(x - 1, y);
            continue;
        }
        else if (ch == 8 && x < 4)
        {
            cct_gotoxy(x + 1, y);
        }

        cout << ch;
        ope += ch;
    }

}
```

> 通过检测输入命令的每一位来决定
>
> - 是否在屏幕上显示
> - 是否换行
> - 是否识别命令等
>
> 此处用了图形化界面的工具集`cmd_console_tools.cpp`

- 命令转换

```
int convert(string com)
{
    string out = "";
    for (int i = 0; i < com.size(); i++)
    {
        while (com[i] >= 65 && com[i] <= 90 || com[i] >= 97 && com[i] <= 
122)
        {
            if (com[i] >= 65 && com[i] <= 90)
                com[i] += 32;
            out += com[i++];
        }

        if (out == commandSet[0])
            return i;
        else if (out == commandSet[1])
            return -2;
        else if (out == commandSet[2])
            return -3;
        else if (out == commandSet[3])
            return -4;
    }

    return -1;
}
```

> 检测输入命令的正确性并返回相应命令序号
>
> 该函数保证了程序健壮性

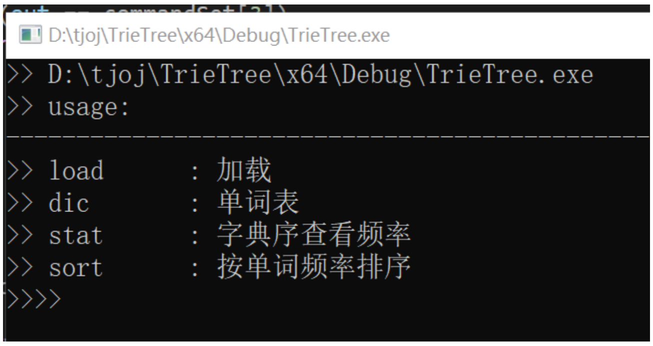
- 主函数命令调用

```
int main(int argc, char*argv[])
{
    /*
        FUNCTION :
            temp >= 0               加载树建树
            temp == -1              命令转换失败标识符
            temp == -2              展示单词表
            temp == -3              按照字典序查看单词
            temp == -4              按照频率查看单词
        */


    cout << ">> " << argv[0] << endl;
    cout << ">> " << "usage:" << endl;
    cout << setiosflags(ios::left);
    cout << "----------------------------------------------------------------------------" << endl;
    cout << ">> " << setw(10) << "load" <<  ": 加载" << endl;
    cout << ">> " << setw(10) << "dic"  << ": 单词表" << endl;
    cout << ">> " << setw(10) << "stat"  << ": 字典序查看频率" << endl;
    cout << ">> " << setw(10) << "sort"  << ": 按单词频率排序" << endl;
    cout << resetiosflags(ios::left);

    //字典树初始化
    TrieTree* tree = new TrieTree();


    //文件内容
    string res = "";

    while (1)
    {
        string ope = "";
        command(ope);
        if (ope.size() == 1 && (ope[0] == 'q' || ope[0] == 'Q'))
        {
            cout << ">>>> ";
            return 0;

        }
        int temp = convert(ope);
        /*
        FUNCTION :
            temp >= 0               加载树建树
            temp == -1              命令转换失败标识符
            temp == -2              展示单词表
            temp == -3              按照字典序查看单词
            temp == -4              按照频率查看单词
        */
        if (temp >= 0)
        {
            tree->preDestructor();
            tree = new TrieTree();

            string path = "";
            for (int j = temp + 1; j < ope.size(); j++)
            {
                if (ope[j] == ' ')
                    break;
                path += ope[j];
            }
            if (readFile(path, res))
            {
                //建树
                buildTree(res, tree);
                //排序

                tree->counter(tree->root_);

                cout << "成功加载" << endl;

            }
            else
            {
                cout << "文件打开失败" << endl;
                continue;
            }

        }
        else if (temp == -1)
        {
            continue;
        }
        else if (temp == -2)
        {

            cout << setiosflags(ios::left) << endl;
            for (int i = 0; i < tree->num; i++)
            {
                cout << setw(20) << i + 1 << tree->freqTable[i].first <<
endl;
            }
            cout << resetiosflags(ios::left);
        }
        else if (temp == -3)
        {
            cout << setiosflags(ios::left);
            cout << setw(25) << "WORD" << "|" << setw(25) << "FREQUNCY" <<
"|PAIR" << endl;
            cout << "-------------------------------------------------------
---------------------" << endl;

            for (int i = 0; i < tree->num; i++)
            {
                cout << setw(25) << tree->freqTable[i].first << "|" <<
setw(25) << tree->freqTable[i].second <<
                "|<" << tree->freqTable[i].first << "," << tree-
>freqTable[i].second << ">" << endl;
            }
            cout << resetiosflags(ios::left);

        }
        else if (temp == -4)
        {
            cout << setiosflags(ios::left);
            cout << setw(25) << "WORD" << "|" << setw(25) << "FREQUNCY" <<
"|PAIR" << endl;
            cout << "-------------------------------------------------------
---------------------" << endl;
            pair<string, int>* sorter = new pair<string, int>[2000000];
            for (int i = 0; i < tree->num; i++)
            {
                sorter[i] = tree->freqTable[i];
            }
            sort(sorter, sorter + tree->num, TrieTree::_Pred());

            for (int i = 0; i < tree->num; i++)
            {

                cout << setw(25) << sorter[i].first << "|" << setw(25) <<
sorter[i].second <<
                    "|<" << sorter[i].first << "," << sorter[i].second <<
">" << endl;
            }
            cout << resetiosflags(ios::left);
            delete[] sorter;

        }

    }
    return 0;
}
```

> 输出形式进行格式控制，相应功能进行相应分支输出

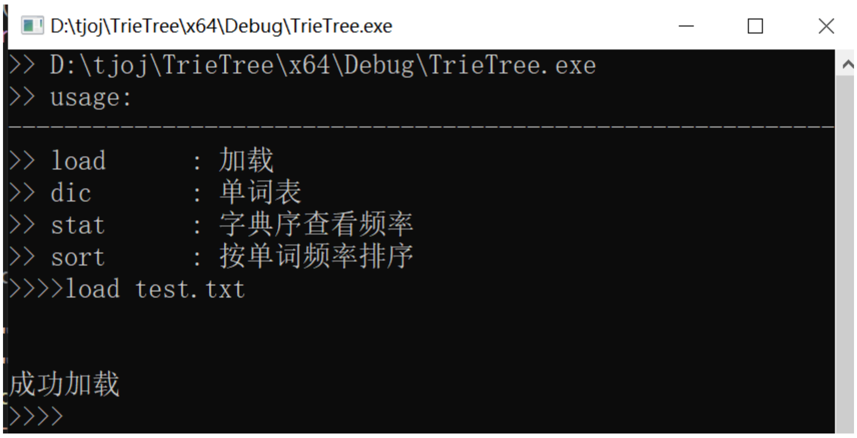
2.5 调试分析

- 此处测试两个测试集，大小分别为`2.14KB`和`371KB`， 前者为维基百科上同济大学部分介绍（`test.txt`），后者为***吹响吧上低音号英文版第一卷***（关西大赛前）（`test1.txt`）
- 初始页面



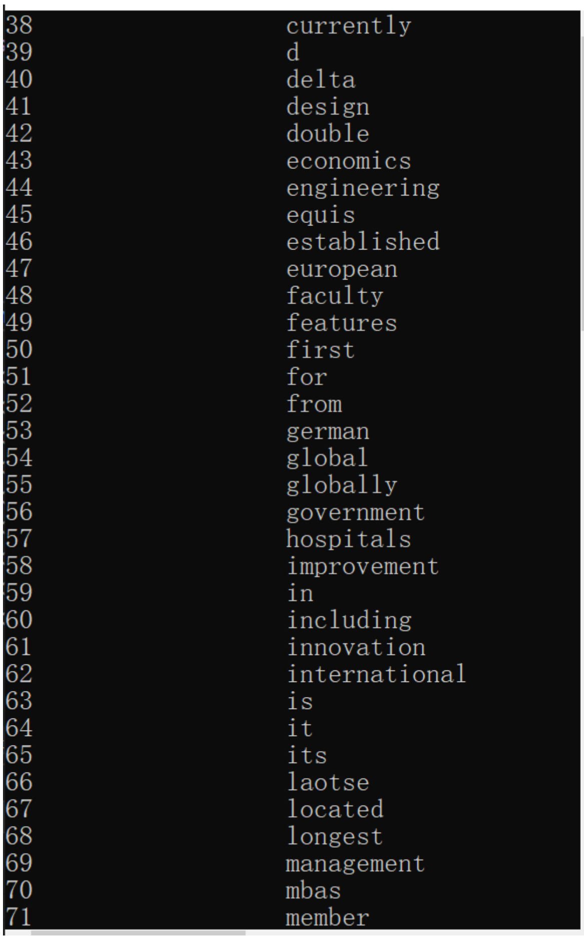
```
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> usage:
---------------------------------------------
>> load       : 加载
>> dic        : 单词表
>> stat       : 字典序查看频率
>> sort       : 按单词频率排序
>>>>
```

- 测试1：
- 功能1： 加载



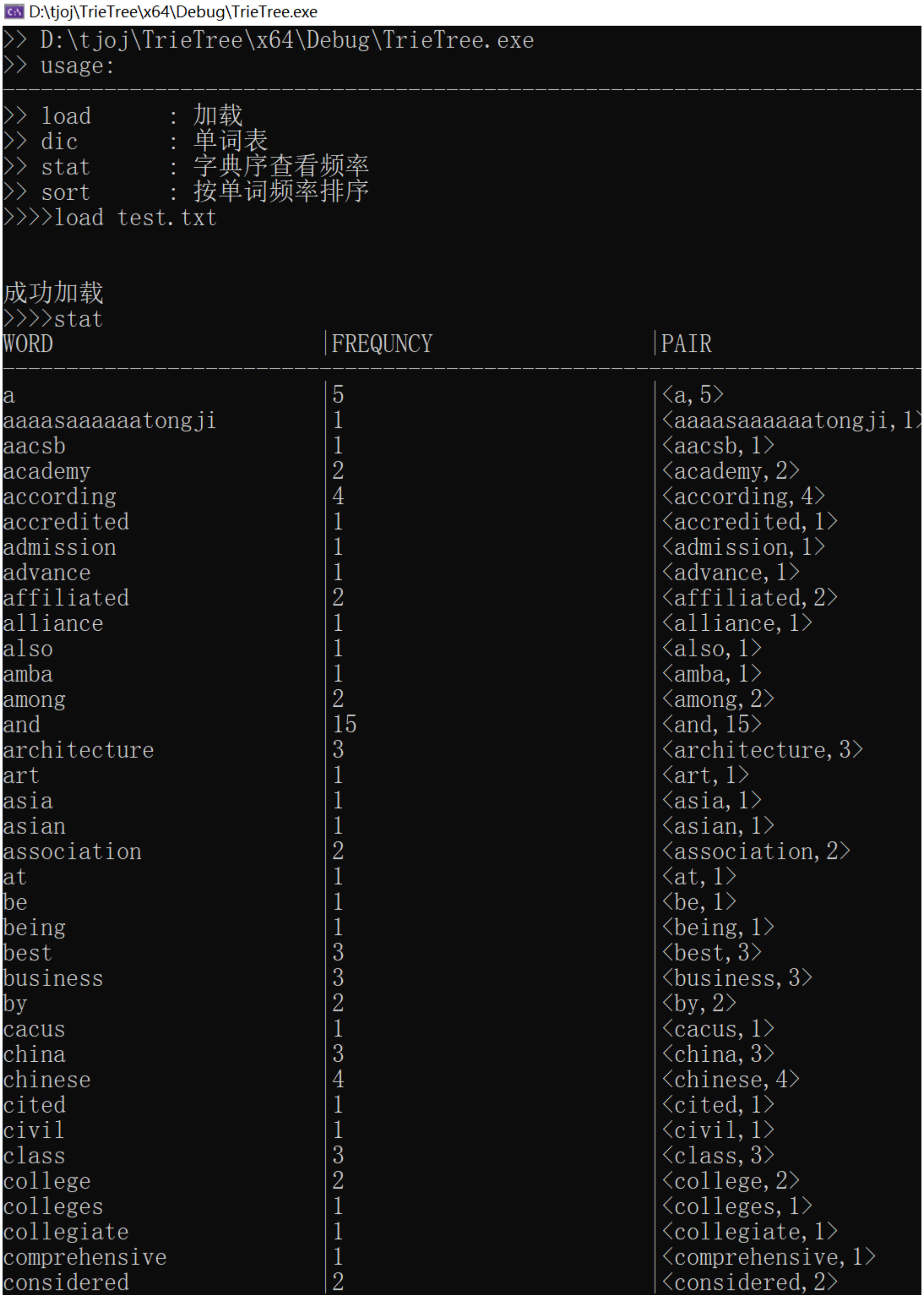
```
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> usage:
------------------------------------------------------------
>> load       : 加载
>> dic        : 单词表
>> stat       : 字典序查看频率
>> sort       : 按单词频率排序
>>>>load test.txt


成功加载
>>>>
```

- 功能2： 单词表



```
38                    currently
39                    d
40                    delta
41                    design
42                    double
43                    economics
44                    engineering
45                    equis
46                    established
47                    european
48                    faculty
49                    features
50                    first
51                    for
52                    from
53                    german
54                    global
55                    globally
56                    government
57                    hospitals
58                    improvement
59                    in
60                    including
61                    innovation
62                    international
63                    is
64                    it
65                    its
66                    laotse
67                    located
68                    longest
69                    management
70                    mbas
71                    member
```

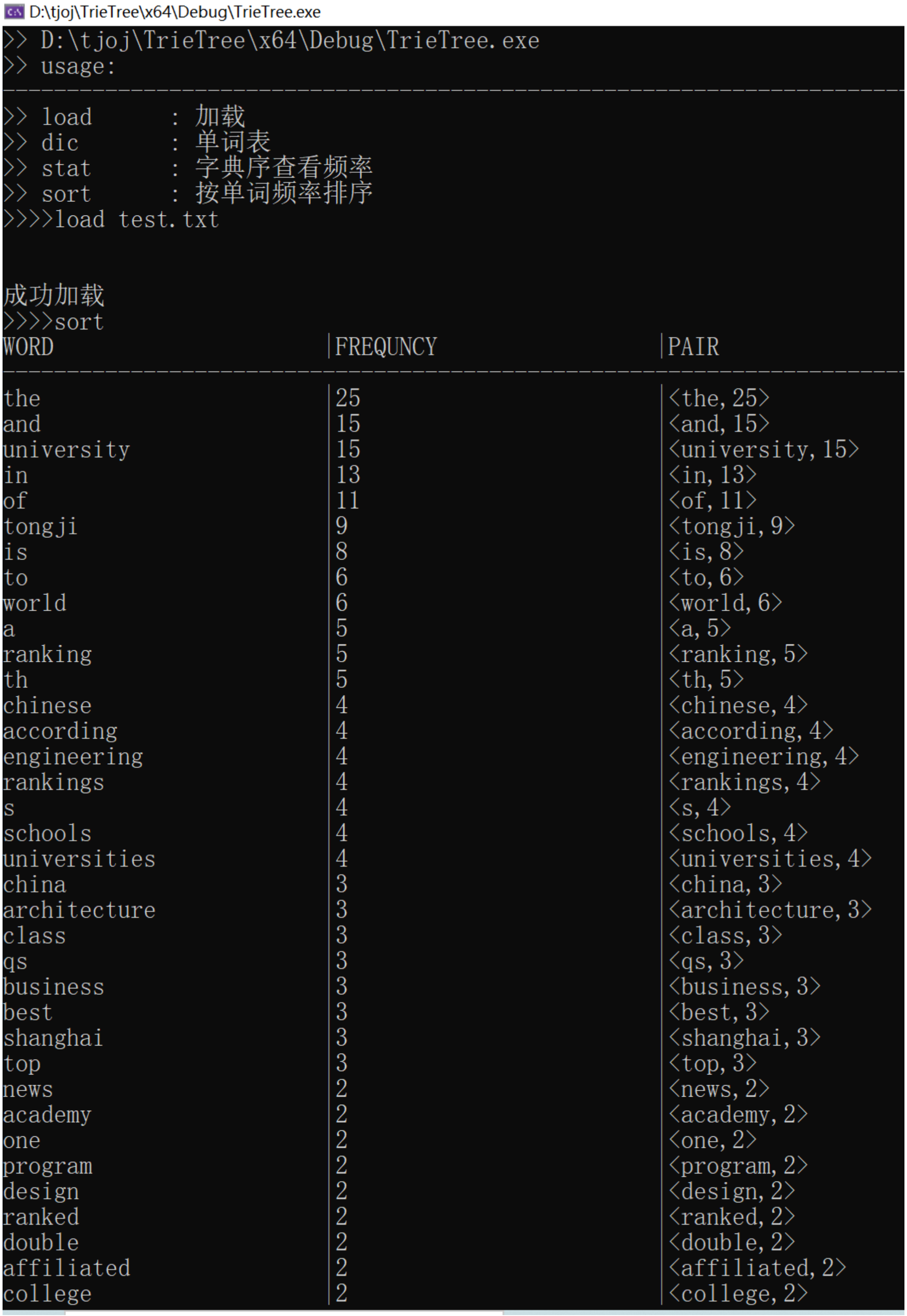
- 功能3： 按照字典序进行排序



D:\tjoj\TrieTree\x64\Debug\TrieTree.exe

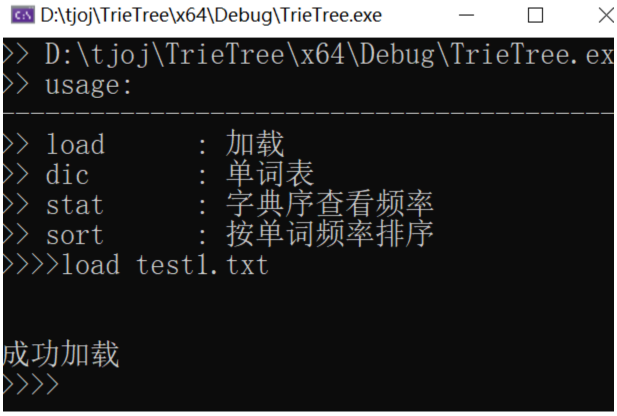
```
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> usage:
------------------------------------------------------------------------
>> load      : 加载
>> dic       : 单词表
>> stat      : 字典序查看频率
>> sort      : 按单词频率排序
>>>>load test.txt


成功加载
>>>>stat
WORD                     |FREQUNCY                 |PAIR
------------------------------------------------------------------------
a                        |5                        |<a,5>
aaaasaaaaaatongji        |1                        |<aaaasaaaaaatongji,1>
aacsb                    |1                        |<aacsb,1>
academy                  |2                        |<academy,2>
according                |4                        |<according,4>
accredited               |1                        |<accredited,1>
admission                |1                        |<admission,1>
advance                  |1                        |<advance,1>
affiliated               |2                        |<affiliated,2>
alliance                 |1                        |<alliance,1>
also                     |1                        |<also,1>
amba                     |1                        |<amba,1>
among                    |2                        |<among,2>
and                      |15                       |<and,15>
architecture             |3                        |<architecture,3>
art                      |1                        |<art,1>
asia                     |1                        |<asia,1>
asian                    |1                        |<asian,1>
association              |2                        |<association,2>
at                       |1                        |<at,1>
be                       |1                        |<be,1>
being                    |1                        |<being,1>
best                     |3                        |<best,3>
business                 |3                        |<business,3>
by                       |2                        |<by,2>
cacus                    |1                        |<cacus,1>
china                    |3                        |<china,3>
chinese                  |4                        |<chinese,4>
cited                    |1                        |<cited,1>
civil                    |1                        |<civil,1>
class                    |3                        |<class,3>
college                  |2                        |<college,2>
colleges                 |1                        |<colleges,1>
collegiate               |1                        |<collegiate,1>
comprehensive            |1                        |<comprehensive,1>
considered               |2                        |<considered,2>
```

- 功能4：按照频率进行排序



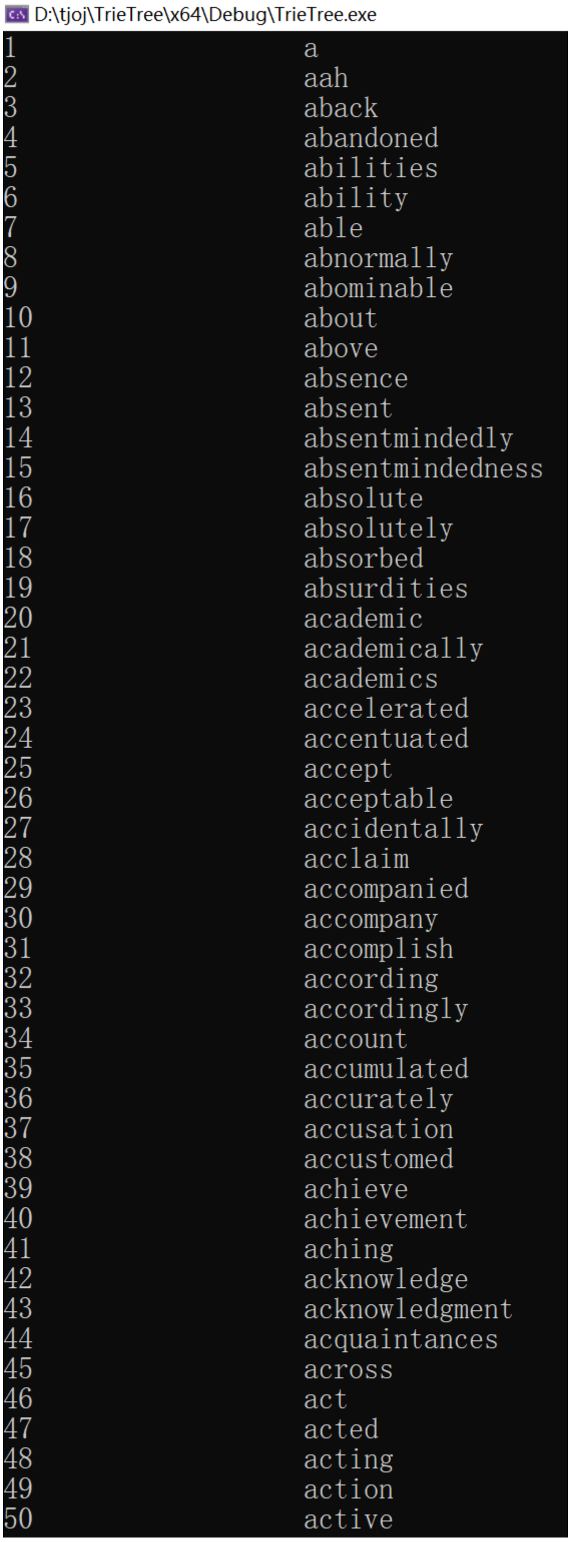
```
D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> usage:
------------------------------------------------------------------------
>> load      : 加载
>> dic       : 单词表
>> stat      : 字典序查看频率
>> sort      : 按单词频率排序
>>>>load test.txt


成功加载
>>>>sort
WORD                     |FREQUNCY                 |PAIR
------------------------------------------------------------------------
the                      |25                       |<the,25>
and                      |15                       |<and,15>
university               |15                       |<university,15>
in                       |13                       |<in,13>
of                       |11                       |<of,11>
tongji                   |9                        |<tongji,9>
is                       |8                        |<is,8>
to                       |6                        |<to,6>
world                    |6                        |<world,6>
a                        |5                        |<a,5>
ranking                  |5                        |<ranking,5>
th                       |5                        |<th,5>
chinese                  |4                        |<chinese,4>
according                |4                        |<according,4>
engineering              |4                        |<engineering,4>
rankings                 |4                        |<rankings,4>
s                        |4                        |<s,4>
schools                  |4                        |<schools,4>
universities             |4                        |<universities,4>
china                    |3                        |<china,3>
architecture             |3                        |<architecture,3>
class                    |3                        |<class,3>
qs                       |3                        |<qs,3>
business                 |3                        |<business,3>
best                     |3                        |<best,3>
shanghai                 |3                        |<shanghai,3>
top                      |3                        |<top,3>
news                     |2                        |<news,2>
academy                  |2                        |<academy,2>
one                      |2                        |<one,2>
program                  |2                        |<program,2>
design                   |2                        |<design,2>
ranked                   |2                        |<ranked,2>
double                   |2                        |<double,2>
affiliated               |2                        |<affiliated,2>
college                  |2                        |<college,2>
```

- 测试2:
- 功能1： 加载



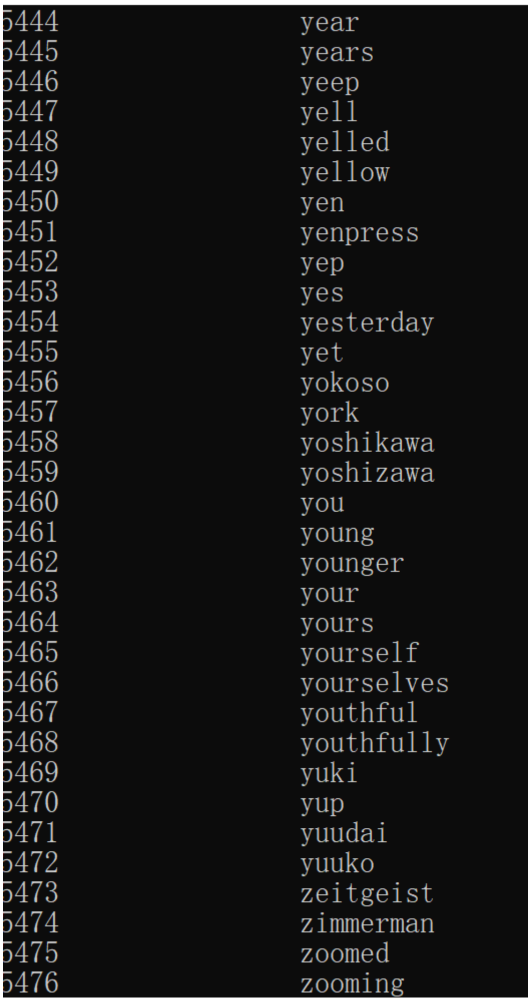
```
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.ex
>> usage:
-----------------------------------------
>> load       : 加载
>> dic        : 单词表
>> stat       : 字典序查看频率
>> sort       : 按单词频率排序
>>>>load test1.txt


成功加载
>>>>
```

- 功能2： 单词表



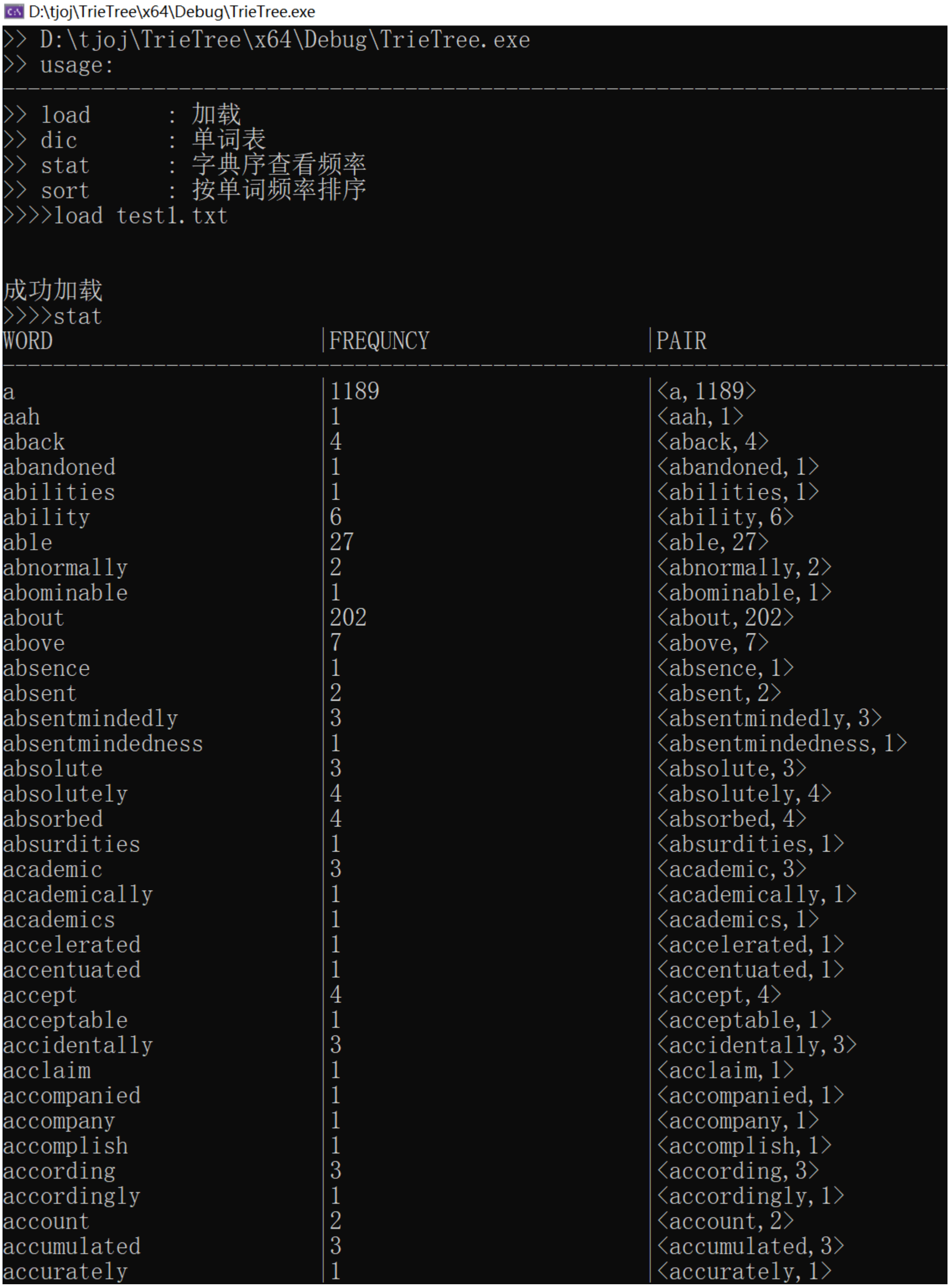
```
1                    a
2                    aah
3                    aback
4                    abandoned
5                    abilities
6                    ability
7                    able
8                    abnormally
9                    abominable
10                   about
11                   above
12                   absence
13                   absent
14                   absentmindedly
15                   absentmindedness
16                   absolute
17                   absolutely
18                   absorbed
19                   absurdities
20                   academic
21                   academically
22                   academics
23                   accelerated
24                   accentuated
25                   accept
26                   acceptable
27                   accidentally
28                   acclaim
29                   accompanied
30                   accompany
31                   accomplish
32                   according
33                   accordingly
34                   account
35                   accumulated
36                   accurately
37                   accusation
38                   accustomed
39                   achieve
40                   achievement
41                   aching
42                   acknowledge
43                   acknowledgment
44                   acquaintances
45                   across
46                   act
47                   acted
48                   acting
49                   action
50                   active
```

- 其中单词总数达到了5476个



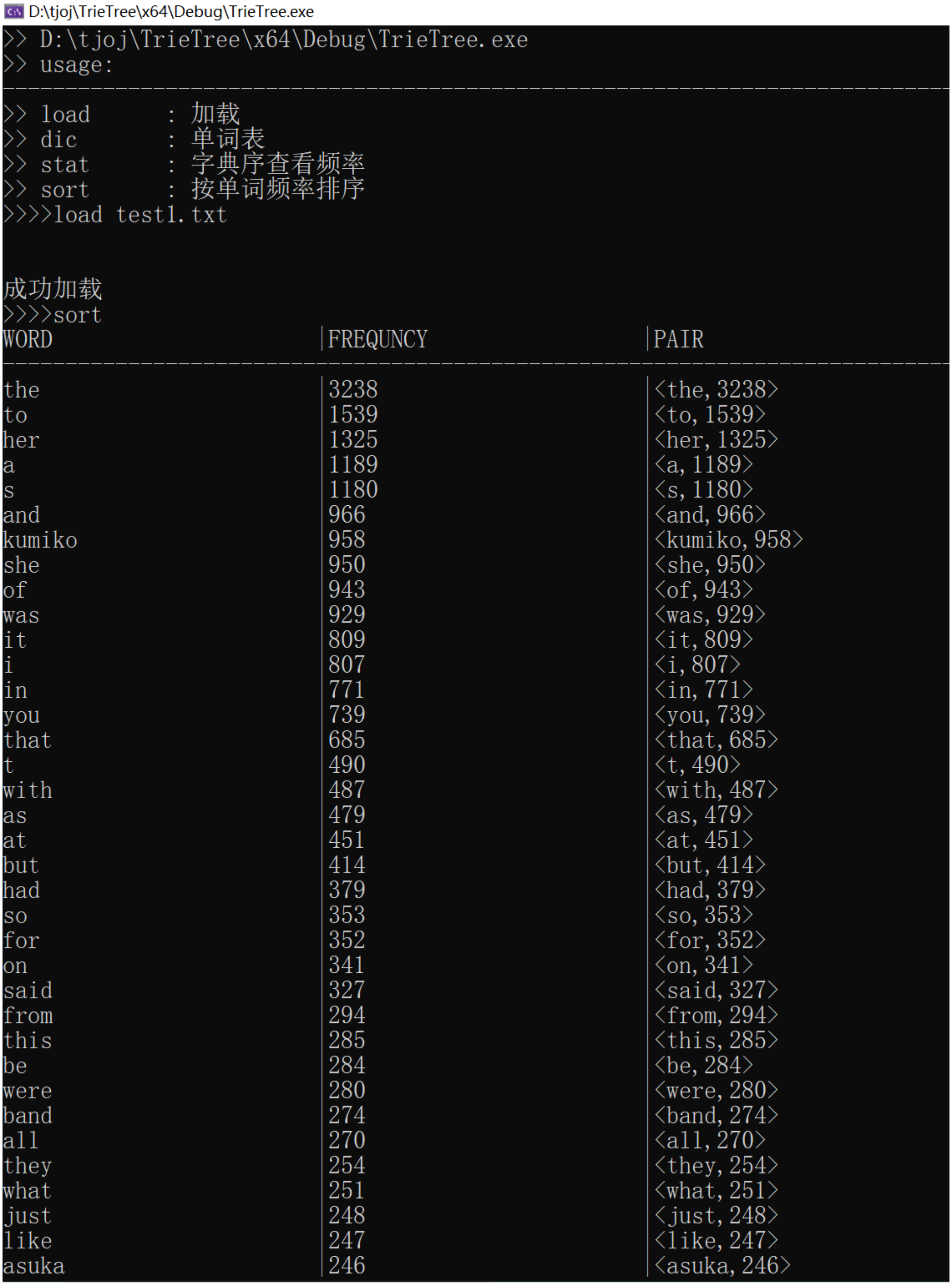
```
5444                    year
5445                    years
5446                    yeep
5447                    yell
5448                    yelled
5449                    yellow
5450                    yen
5451                    yenpress
5452                    yep
5453                    yes
5454                    yesterday
5455                    yet
5456                    yokoso
5457                    york
5458                    yoshikawa
5459                    yoshizawa
5460                    you
5461                    young
5462                    younger
5463                    your
5464                    yours
5465                    yourself
5466                    yourselves
5467                    youthful
5468                    youthfully
5469                    yuki
5470                    yup
5471                    yuudai
5472                    yuuko
5473                    zeitgeist
5474                    zimmerman
5475                    zoomed
5476                    zooming
```

- 功能3： 按照字典序进行排序



```
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> usage:
--------------------------------------------------------------------------
>> load      : 加载
>> dic       : 单词表
>> stat      : 字典序查看频率
>> sort      : 按单词频率排序
>>>>load test1.txt


成功加载
>>>>stat
WORD                     |FREQUNCY                 |PAIR
--------------------------------------------------------------------------
                         |                         |
a                        |1189                     |<a,1189>
aah                      |1                        |<aah,1>
aback                    |4                        |<aback,4>
abandoned                |1                        |<abandoned,1>
abilities                |1                        |<abilities,1>
ability                  |6                        |<ability,6>
able                     |27                       |<able,27>
abnormally               |2                        |<abnormally,2>
abominable               |1                        |<abominable,1>
about                    |202                      |<about,202>
above                    |7                        |<above,7>
absence                  |1                        |<absence,1>
absent                   |2                        |<absent,2>
absentmindedly           |3                        |<absentmindedly,3>
absentmindedness         |1                        |<absentmindedness,1>
absolute                 |3                        |<absolute,3>
absolutely               |4                        |<absolutely,4>
absorbed                 |4                        |<absorbed,4>
absurdities              |1                        |<absurdities,1>
academic                 |3                        |<academic,3>
academically             |1                        |<academically,1>
academics                |1                        |<academics,1>
accelerated              |1                        |<accelerated,1>
accentuated              |1                        |<accentuated,1>
accept                   |4                        |<accept,4>
acceptable               |1                        |<acceptable,1>
accidentally             |3                        |<accidentally,3>
acclaim                  |1                        |<acclaim,1>
accompanied              |1                        |<accompanied,1>
accompany                |1                        |<accompany,1>
accomplish               |1                        |<accomplish,1>
according                |3                        |<according,3>
accordingly              |1                        |<accordingly,1>
account                  |2                        |<account,2>
accumulated              |3                        |<accumulated,3>
accurately               |1                        |<accurately,1>
```

- 功能4： 按照频率进行排序



```
D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> D:\tjoj\TrieTree\x64\Debug\TrieTree.exe
>> usage:
--------------------------------------------------------------------------------
>> load      : 加载
>> dic       : 单词表
>> stat      : 字典序查看频率
>> sort      : 按单词频率排序
>>>>load test1.txt


成功加载
>>>>sort
WORD                      |FREQUNCY                  |PAIR
--------------------------------------------------------------------------------
the                       |3238                      |<the,3238>
to                        |1539                      |<to,1539>
her                       |1325                      |<her,1325>
a                         |1189                      |<a,1189>
s                         |1180                      |<s,1180>
and                       |966                       |<and,966>
kumiko                    |958                       |<kumiko,958>
she                       |950                       |<she,950>
of                        |943                       |<of,943>
was                       |929                       |<was,929>
it                        |809                       |<it,809>
i                         |807                       |<i,807>
in                        |771                       |<in,771>
you                       |739                       |<you,739>
that                      |685                       |<that,685>
t                         |490                       |<t,490>
with                      |487                       |<with,487>
as                        |479                       |<as,479>
at                        |451                       |<at,451>
but                       |414                       |<but,414>
had                       |379                       |<had,379>
so                        |353                       |<so,353>
for                       |352                       |<for,352>
on                        |341                       |<on,341>
said                      |327                       |<said,327>
from                      |294                       |<from,294>
this                      |285                       |<this,285>
be                        |284                       |<be,284>
were                      |280                       |<were,280>
band                      |274                       |<band,274>
all                       |270                       |<all,270>
they                      |254                       |<they,254>
what                      |251                       |<what,251>
just                      |248                       |<just,248>
like                      |247                       |<like,247>
asuka                     |246                       |<asuka,246>
```

- 可以看到某些代词、介词、系动词出现次数明显较高，符合预期
- ~~可以看到主角 `Oumae Kumiko` 的名字出现了958次，比某些代词出现次数还多，这可能是由于 `Rena` 每次出现就喊 `Kumiko!!` 原因~~
- ~~同时可以看到吹奏部实际掌权人 `Tanaka Asuka` 的名字也出现了246次，说明其在小说中的核心地位~~
- 以上说明排序结果符合预期

3. 实验总结

- 本实验进行了字典树的实现，了解了 `Trie` 树的建立和一些基本操作
- 程序实现了健壮性强，用户界面友好，实现了命令行操作以及具有错误提示以及相关表格标题等

- 程序在验证中验证了不同规模的数据集，保证了程序运行的可行性
- 本实验主要实现了两种单词查看方式，即依照频率和字典顺序，并由此附加了展示单词表的功能
- 改进：
    - 命令行在删除时，有时会将命令行的提示符覆盖掉
    - 删除功能（析构）尚不完善，需要进一步优化

4. 源代码

4.1 头文件

`cmd`**绘图工具集**`cmd_console_tools.h`(略)

`TrieTree.h`

```
#pragma once
#include <iostream>
#include <algorithm>
#include <stdio.h>
using namespace std;


//Mode of Search()


class TrieTree
{
public:
    struct Node
    {
        char ch;
        bool end;
        string word = "";
        Node* next[30];      //均转换为26个字母，一共就只有最多26个孩子节点
        int idx;
        int freq;
        int vis;
        Node(char ch1=' ', bool ende = false, string worde = "", int index=0,
int fre=0, int visited=0) : ch(ch1), end(ende), word(worde), idx(index),
freq(fre), vis(visited) {
            for (int i = 0; i < 30; i++)
            {
                next[i] = NULL;
            }
        }


    };
    TrieTree();

    int Insert(string str);

    int Search(string str, int& f, int mode);

    int findWord(string str, int& f);

    int findPrefix(string prefix, int& f);

    class _Pred
    {
    public:
        bool operator()(pair<string, int> p1, pair<string, int> p2)
        {
            return p1.second > p2.second;
        }
    };

    void counter(Node* root);

    void deleter(Node* root);

    void preDestructor();

    ~TrieTree();

    Node* root_;

    pair<string, int>* freqTable = new pair<string, int>[2000000];
//已知英语单词超过1000000个

    int num;


};
```

4.2 cpp文件

绘图工具集`cmd_console_tools.cpp`（略）

`TrieTree.cpp`

```
#include "TrieTree.h"

TrieTree::TrieTree()
{
    this->root_ = new Node();
    this->num = 0;
}
```

```
int TrieTree::Insert(string str)
{
    Node* p = this->root_;

    //转换为小写字母
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] >= 65 && str[i] <= 90)
            str[i] += 32;
    }

    int found = 0;
    for (int i = 0; i < str.length(); i++)
    {
        found = 0;

//*********************************************************************MODIFIED*
*************************************************************

        for (int j = 0; j < 26 ; j++)
        {
            if (p->next[j] != NULL && p->next[j]->ch == str[i])
            {
                //cout << p->next[j]->ch << endl;

                found = 1;
                p = p->next[j];
                break;
            }
        }

//*********************************************************************MODIFIED*
*************************************************************


        if (!found)
        {
            Node* node = new Node(str[i], false, "", 0);
            if (node == NULL)
                return 0;

//*********************************************************************MODIFIED*
*************************************************************

            p->next[str[i] - 'a'] = node;
            p->idx++;
            //p->next[p->idx++] = node;
            //cout << p->next[p->idx - 1]->ch << endl;
            //cout << p->word;
            //p = p->next[str[i] - 'a'];
            p = node;

//*********************************************************************MODIFIED*
*************************************************************
```

```
        }


    }

    p->end = true;
    p->word = str;
    //cout << p->word << endl;
    p->freq++;
    return 1;
}

int TrieTree::Search(string str, int& f, int mode)
{
    //转换为小写字母
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] >= 65 && str[i] <= 90)
            str[i] += 32;
    }

    Node* p = this->root_;
    if (p == NULL)
        return 0;
    int found = 0;
    for (int i = 0; i < str.length(); i++)
    {
        found = 0;

//*********************************************************************MODIFIED*
*************************************************************

        for (int j = 0; j < 26; j++)
        {
            if ( p->next[j] != NULL && p->next[j]->ch == str[i])
            {
                found = 1;
                p = p->next[j];
                break;
            }
        }


//*********************************************************************MODIFIED*
*************************************************************

        if (!found)
        {
            //cout << i << endl;
            f = -1;
            return 0;
        }
    }
    //goo <-> google

    //cout << "-----------" << p->freq << endl;
    if (p->end == 0 && mode)
    {
        f = -1;
        return 0;
    }

    f = p->freq;
    //cout << "-----------" << outNode->freq << endl;

    return 1;


}

int TrieTree::findWord(string str, int& f)
{
    return this->Search(str, f, 1);

}

int TrieTree::findPrefix(string prefix, int& f)
{
    return this->Search(prefix, f, 0);
}

//二元谓词



void TrieTree::counter(Node* root)
{
    if (root->idx == 0)
    {
        this->freqTable[this->num++] = { root->word , root->freq };
        return;
    }

    if (root->end)
    {

        this->freqTable[this->num++] = { root->word , root->freq };
    }


//*********************************************************************MODIFIED*************************************************************

    for (int i = 0; i < 26; i++)
    {
        //cout << i << endl;
        if (root->next[i] != NULL)
            this->counter(root->next[i]);
    }

//*********************************************************************MODIFIED*
*************************************************************


//*********************************************************************MODIFIED*
*************************************************************

    //sort(this->freqTable, this->freqTable + num, _Pred());

//*********************************************************************MODIFIED*
*************************************************************


}

void TrieTree::deleter(Node* root)
{
    for (int i = 0; i < 26; i++)
    {
        if (root->next[i] != NULL)
            deleter(root->next[i]);
    }
    delete root;

}

void TrieTree::preDestructor()
{
    //后序遍历删除节点，删除存储列表
    this->deleter(this->root_);
    delete[] this->freqTable;

}

TrieTree::~TrieTree()
{
    this->preDestructor();

}
```

`mainWorker.cpp`

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
#include <conio.h>

#include "TrieTree.h"
#include "cmd_console_tools.h"
```

```
using namespace std;

string commandSet[] = { "load", "dic", "stat", "sort"};

int readFile(string path, string& out)
{
    ifstream readFrom;
    readFrom.open(path, ios::in);
    if (readFrom.is_open() == 0)
        return 0;
    string temp = "";
    while (readFrom.good())
    {
        getline(readFrom, temp);
        //cout << temp.size() << endl;
        if (temp.length() && temp[temp.length() - 1] == '-')
            temp = temp.substr(0, temp.length() - 1);       //直接将每一行末尾的连字符去掉，对行中间的连字符保留
        if (temp.length())
            out += temp;
    }
    readFrom.close();
    return 1;

}

int buildTree(string src, TrieTree* tree)
{
    cout << endl << endl;
    for (int i = 0; i < src.length(); i++)
    {
        string temp = "";

        while (i < src.length() && (src[i] >= 'A' && src[i] <= 'Z' || src[i] >= 'a' && src[i] <= 'z'))  // || src[i] == '-'  //连字符字典序遍历就失效了
        {

            temp += src[i++];
        }

        if (temp.size())
        {
            tree->Insert(temp);
        }
    }

    return 1;
}

int command(string& ope)
{
    cout << ">>>";
    char ch = ' ';
    if ((ch = _getch()) == '\n')
        return -1;
    else
    {
        ope += ch;
        cout << ch;
    }

    while (1)
    {

        ch = _getch();
        //cout << ch + 0 << endl;
        if (ch == 13)
        {
            cout << endl;
            return 1;
        }

        int x = 0, y = 0;
        cct_getxy(x, y);
        if (ch == 8 && ope.length() && x >= 4)
        {
            ope = ope.substr(0, ope.length() - 1);

            cct_gotoxy(x - 1, y);
            cout << " ";
            cct_gotoxy(x - 1, y);
            continue;
        }
        else if (ch == 8 && x < 4)
        {
            cct_gotoxy(x + 1, y);
        }

        cout << ch;
        ope += ch;
    }

}

int convert(string com)
{
    string out = "";
    for (int i = 0; i < com.size(); i++)
    {
        while (com[i] >= 65 && com[i] <= 90 || com[i] >= 97 && com[i] <= 122)
        {
            if (com[i] >= 65 && com[i] <= 90)
                com[i] += 32;
            out += com[i++];
        }

        if (out == commandSet[0])
            return i;
        else if (out == commandSet[1])
            return -2;
        else if (out == commandSet[2])
            return -3;
        else if (out == commandSet[3])
            return -4;
    }

    return -1;

}


int main(int argc, char*argv[])
{
    cout << ">> " << argv[0] << endl;
    cout << ">> " << "usage:" << endl;
    cout << setiosflags(ios::left);
    cout << "----------------------------------------------------------------------------" << endl;
    cout << ">> " << setw(10) << "load" <<  ": 加载" << endl;
    cout << ">> " << setw(10) << "dic"  << ": 单词表" << endl;
    cout << ">> " << setw(10) << "stat"  << ": 字典序查看频率" << endl;
    cout << ">> " << setw(10) << "sort"  << ": 按单词频率排序" << endl;
    cout << resetiosflags(ios::left);






    //字典树初始化
    TrieTree* tree = new TrieTree();


    //文件内容
    string res = "";

    while (1)
    {
        string ope = "";
        command(ope);
        if (ope.size() == 1 && (ope[0] == 'q' || ope[0] == 'Q'))
        {
            cout << ">>>> ";
            return 0;

        }

        int temp = convert(ope);
        if (temp >= 0)
        {
            tree->preDestructor();
            tree = new TrieTree();

            string path = "";
            for (int j = temp + 1; j < ope.size(); j++)
            {
                if (ope[j] == ' ')
                    break;
                path += ope[j];
            }



            if (readFile(path, res))
            {
                //建树
                buildTree(res, tree);
                //排序

                tree->counter(tree->root_);

                cout << "成功加载" << endl;

            }
            else
            {
                cout << "文件打开失败" << endl;
                continue;
            }

        }
        else if (temp == -1)
        {
            continue;
        }
        else if (temp == -2)
        {

            cout << setiosflags(ios::left) << endl;
            for (int i = 0; i < tree->num; i++)
            {
                cout << setw(20) << i + 1 << tree->freqTable[i].first << endl;
            }
            cout << resetiosflags(ios::left);
        }
        else if (temp == -3)
        {
            cout << setiosflags(ios::left);
            cout << setw(25) << "WORD" << "|" << setw(25) << "FREQUNCY" << 
"|PAIR" << endl;
            cout << "-----------------------------------------------------------
-----------------" << endl;

            for (int i = 0; i < tree->num; i++)
            {
                cout << setw(25) << tree->freqTable[i].first << "|" << setw(25) 
<< tree->freqTable[i].second << 
                "|<" << tree->freqTable[i].first << "," << tree-
>freqTable[i].second << ">" << endl;
            }
            cout << resetiosflags(ios::left);

        }
        else if (temp == -4)
        {
            cout << setiosflags(ios::left);
            cout << setw(25) << "WORD" << "|" << setw(25) << "FREQUNCY" <<
"|PAIR" << endl;
            cout << "-----------------------------------------------------------
-----------------" << endl;
            pair<string, int>* sorter = new pair<string, int>[2000000];
            for (int i = 0; i < tree->num; i++)
            {
                sorter[i] = tree->freqTable[i];
            }
            sort(sorter, sorter + tree->num, TrieTree::_Pred());

            for (int i = 0; i < tree->num; i++)
            {

                cout << setw(25) << sorter[i].first << "|" << setw(25) <<
sorter[i].second <<
                    "|<" << sorter[i].first << "," << sorter[i].second << ">" <<
endl;
            }
            cout << resetiosflags(ios::left);
            delete[] sorter;

        }

    }
    return 0;
}
```