数据结构课程设计

项目说明文档

关键字检索系统

|  |  |
| --- | --- |
| 作者姓名： | 高逸轩 |
| 学 号： | 2053385 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

# 1项目分析

## 项目需求分析

建立一个文本文件，里面附有部分文字内容。文件名由用户用键盘输入，输入一个不含空格的关键字，统计输出关键字在文本中的出现次数。

* 执行效率高

当用户所输入的迷宫数据比较大的情况下，该系统需要能够在比较短的时间内找出所要求解两点间的路径。

* 准确性

根据题目描述，本次题目要求检索某个单词在文章中出现的次数。应当实现大小写不敏感、前后缀敏感等需求。

# 1.2 项目要求

### 1.2.1 功能要求

首先建立一个文本文件，文件名由用户用键盘输入；然后输入一个不含空格的关键字，统计输出该单词在文本中的出现次数。

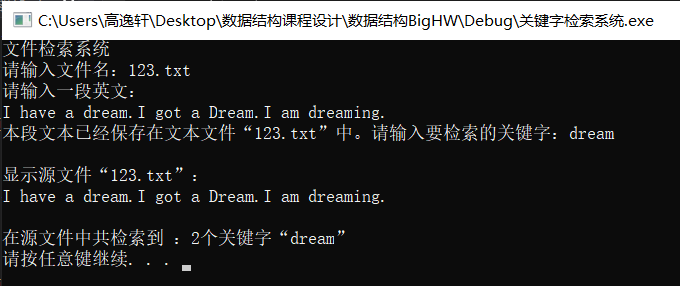
### 1.2.2 输入格式

先输入一个文件的名字，再输入一段文本作为文件内容，最后输入要查找的关键词。

### 1.2.3 输出格式

先输出文件内容，再输出关键词在文中出现的次数。

### 1.2.4 项目示例



解释：在文中搜索单词dream，由于大小写不敏感，记录为dream 、Dream共两次。dreaming 中dream只是一个前缀，不是一个相同的单词，故不计入次数。

# 2 项目设计

## 2.1 数据结构设计

在本题中，由于要**避免关键词在某个单词中仅仅作为单词的一部分出现而误计入频次**，不能采取简单的字符串模式匹配或者是其优化算法KMP，而是采取了字典树。

字典树（**Trie Tree**)是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存**大量的字符串**（但不仅限于字符串）。其特点有：

1.Trie的根节点是空的。

2.除根节点外，每个节点储存一个单词/字母

3.从根节点到每个单词节点的路径上的所有字母连接而成的字符串就是该节点对应的字符串

4.每个非叶子结点一般都会被多次使用，以节省遍历时的时间效率

它的优点是：利用字符串的**公共前缀**来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。并利用了字符串的公共前缀来节约存储空间。

**另外，在文章末尾，拓展介绍了AC自动机，同样可以解决此问题。**

## 2.2 类设计

字典树的建立包括两个抽象数据类型（ADT）——结点类（TrieNode）与字典树类（Trie），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。以下为部分代码：

### 2.2.1 节点

// 字典树节点

struct TrieNode

{

TrieNode()

int wordNum; // 单词频数，若为0说明当前位置不在词尾

TrieNode\* next[26]; // 指向26个可能的后继节点（26个字母）

};

### 2.2.2 字典树

// 字典树 Trie树

class Trie {

public:

// 构造函数，为根节点申请空间

Trie()

// 向字典树插入单词word

void insert(string word)

// 搜索word出现的词频

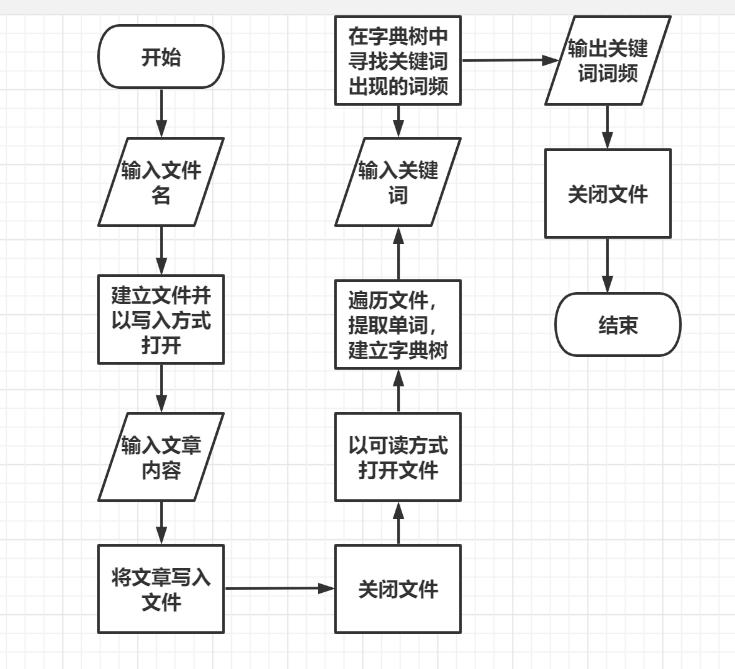
int search(string word)

private:

TrieNode\* root; // 根节点

};

## 2.3 项目流程图



# 3 核心代码介绍

## 3.1 文件处理

 本题中需要频繁对文件进行处理，这里用fopen()和fclose()来打开和关闭文件，用fputs()与fgets()函数分别从文件进行字符串的写入和读取操作。

## 3.2 字典树的插入

首先，我们需要将指针current指向根节点root，并将插入的单词word分为字符数组（在此之前需要先将其转换为全部为小写字母，以便于字典树的储存）。

由于当前节点下面可能有26个节点，我们取单词word的第一个字符并减去字符’a’来获取一个下标值，这个下标值指的是单词word的第一个字符应该插入到当前节点current下面的26个节点中的某个节点的位置。如果该位置的节点为空，那么就新建一个节点，代表单词word的第一个字符节点。若不为空，那么就说明该字符已经插入过了，指针指向下一个节点，并重复以上的步骤直到遍历至字符串word的最后一个字符。

当单词的所有字符都遍历后，在最后一个节点处标记其词频+1（仅在单词的末尾字符更新词频即可）。

## 3.3 字典树的查询

search方法基本思路与insert方法是相似的。不同的是，当在查找某个单词时，若该单词还没查找完，单词字符序列没有与之对应的字典树中的字符节点序列，也就在查找过程中发现了空节点，那么说明该单词不存在于字典树中。若单词查找完了，但是该单词的最后一个字符所对应的字典树中节点的词频wordNum的值为0，即该节点不是终止节点，那么也说明该单词不存在与字典树中。

## 3.4 将单词转换为小写

由于字典树设置为仅能存在为小写字母，所以在插入和查询时候，需要先将全部的字符转换为小写字母再进行操作。在这个过程中，我使用了部分系统函数：用isalpha()函数来判断当前字符是否为字母，用tolower()函数将某个字母转换为小写，用transform()函数将某个字符串直接转换为小写字母。

# 4 项目测试

为方便老师测试，提供了文件5\_test.txt，内含一组数据，测试了本程序的全部功能。以下为一组测试数据：

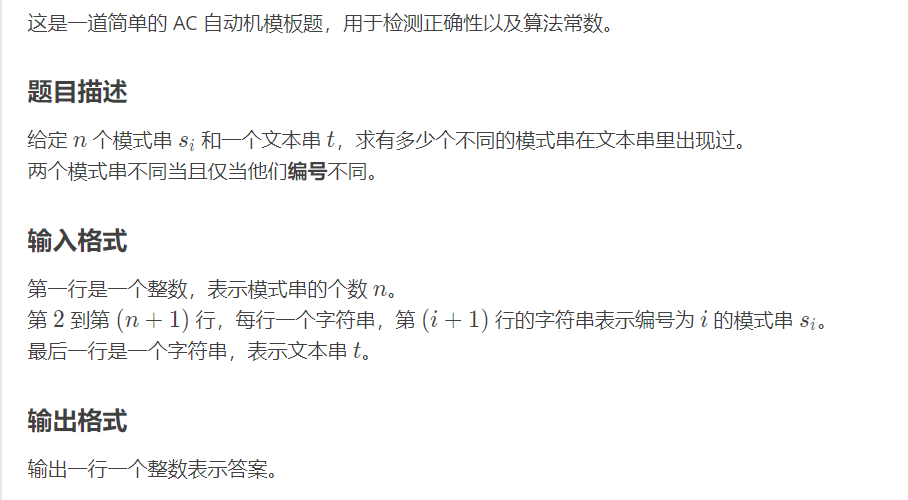
# 5 心得体会与扩展

本题为经典的字典树的例题，求某个字符串在文本中的出现次数。

如果说KMP是“一对一”（一个字符串在另一个字符串中是否匹配），字典树是“一对多”（一个字符串在字典中的匹配情况），那么**AC自动机则是实现了“多对多”**，可以在高效的时间复杂度内求解许多单词在字典中的情况。

KMP是用于一对一的字符串匹配，而Trie虽然能用于多模式匹配，但是每次匹配失败都需要进行回溯，如果模式串很长的话会很浪费时间。所以AC自动机应运而生,如同Manacher一样，AC自动机利用某些操作（综合了KMP和Trie的思想）阻止了模式串匹配阶段的回溯,将时间复杂度优化到了，n为文本串长度。

下附代码：（由于是高中信息学竞赛时编写，命名规范、面向对象等方面完成的不佳，可读性不高，请老师海涵）



#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<cmath>

#include<queue>

#include<algorithm>

using namespace std;

struct Tree//字典树

{

int fail;//失配指针

int vis[26];//子节点的位置

int end;//标记有几个单词以这个节点结尾

}AC[1000000];//Trie树

int cnt=0;//Trie的指针

inline void Build(string s)

{

int l=s.length();

int now=0;//字典树的当前指针

for(int i=0;i<l;++i)//构造Trie树

{

if(AC[now].vis[s[i]-'a']==0)//Trie树没有这个子节点

AC[now].vis[s[i]-'a']=++cnt;//构造出来

now=AC[now].vis[s[i]-'a'];//向下构造

}

AC[now].end+=1;//标记单词结尾

}

void Get\_fail()//构造fail指针

{

queue<int> Q;//队列

for(int i=0;i<26;++i)//第二层的fail指针提前处理一下

{

if(AC[0].vis[i]!=0)

{

AC[AC[0].vis[i]].fail=0;//指向根节点

Q.push(AC[0].vis[i]);//压入队列

}

}

while(!Q.empty())//BFS求fail指针

{

int u=Q.front();

Q.pop();

for(int i=0;i<26;++i)//枚举所有子节点

{

if(AC[u].vis[i]!=0)//存在这个子节点

{

AC[AC[u].vis[i]].fail=AC[AC[u].fail].vis[i];

//子节点的fail指针指向当前节点的

//fail指针所指向的节点的相同子节点

Q.push(AC[u].vis[i]);//压入队列

}

else//不存在这个子节点

AC[u].vis[i]=AC[AC[u].fail].vis[i];

//当前节点的这个子节点指向当

//前节点fail指针的这个子节点

}

}

}

int AC\_Query(string s)//AC自动机匹配

{

int l=s.length();

int now=0,ans=0;

for(int i=0;i<l;++i)

{

now=AC[now].vis[s[i]-'a'];//向下一层

for(int t=now;t&&AC[t].end!=-1;t=AC[t].fail)//循环求解

{

ans+=AC[t].end;

AC[t].end=-1;

}

}

return ans;

}

int main()

{

int n;

string s;

cin>>n;

for(int i=1;i<=n;++i)

{

cin>>s;

Build(s);

}

AC[0].fail=0;//结束标志

Get\_fail();//求出失配指针

cin>>s;//文本串

cout<<AC\_Query(s)<<endl;

return 0;

}

由于篇幅原因，报告内还有很多内容与解释没有展示，请老师和助教老师再移步源程序，在其中的注释写了每一步过程的详解。