**文本索引**

1. 实验目的

编写一个构建大块文本索引的程序，然后进行快速搜索，来查找某个字符串在该文本中的出现位置。

1. 实验内容

你的程序应该使用两个文件名作为命令行参数：文本文件（我们称为语料库）和包含查询的文件。 假 设这两个文件只包含小写字母、空格和换行符，查询文件中的查询由换行符分隔。 这不是一个限制，因为你可以使用一个过滤器将任何文件转换为此格式。 你的程序应该读取语料库，将其存储为（可能巨大）字符串，并可能为其创建索引，如下所述。 然后 它应该逐个读取查询（假设在命令行中的第二个命名文件中，每行有一个查询），并打印出语料库中每个查询在文本文件中首次出现的位置。

1. 代码实现

代码如下：

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <streambuf>

#include <iomanip>

using std::cin;

using std::cout;

using std::endl;

using std::ifstream;

using std::string;

using std::vector;

using namespace std;

const int MAX\_CHAR\_ASCLL = 255;

int shift[MAX\_CHAR\_ASCLL];

*// Sunday算法的实现*

int Sunday(const string &thestring, const string &mystring)

{

    int length1 = thestring.size();

    int length2 = mystring.size();

    for (int i = 0; i < MAX\_CHAR\_ASCLL; i++)

    {

        shift[i] = length2 + 1;

    }

    for (int i = 0; i < length2; i++)

    {

        shift[mystring[i]] = length2 - i;

    }

    int i = 0;

    int j;

    while (i <= length1 - length2)

    {

        j = 0;

        while (thestring[i + j] == mystring[j])

        {

            j++;

            if (j >= length2)

                return i;

        }

        i += shift[thestring[i + length2]];

    }

    return -1;

}

int calculate\_index(int k, string txt)

{

    if (k == -1)

        return -1;

    int j = 1;

    for (int i = 0; i <= k; i++)

    {

        if (txt[i] == ' ')

            j++;

    }

    return j;

}

*// 使用更加高效的Sunday算法来进行模式匹配*

int main()

{

    std::ifstream sin("C:\\Users\\lenovo\\Desktop\\message.txt");

    string txt((std::istreambuf\_iterator<char>(sin)), std::istreambuf\_iterator<char>());

    string mystring;

    ifstream in("C:\\Users\\lenovo\\Desktop\\in.txt", ios::in);

    if (!in.is\_open())

    {

        cerr << "open error!" << endl;

        exit(0);

    }

    vector<string> vec;

    while (!in.eof())

    {

        in >> mystring;

        vec.push\_back(mystring);

    }

    for (int i = 0; i < vec.size(); i++)

    {

        int k = Sunday(txt, vec[i]);

        k = calculate\_index(k, txt);

        cout << k << "  " << vec[i] << endl;

    }

    return 0;

}

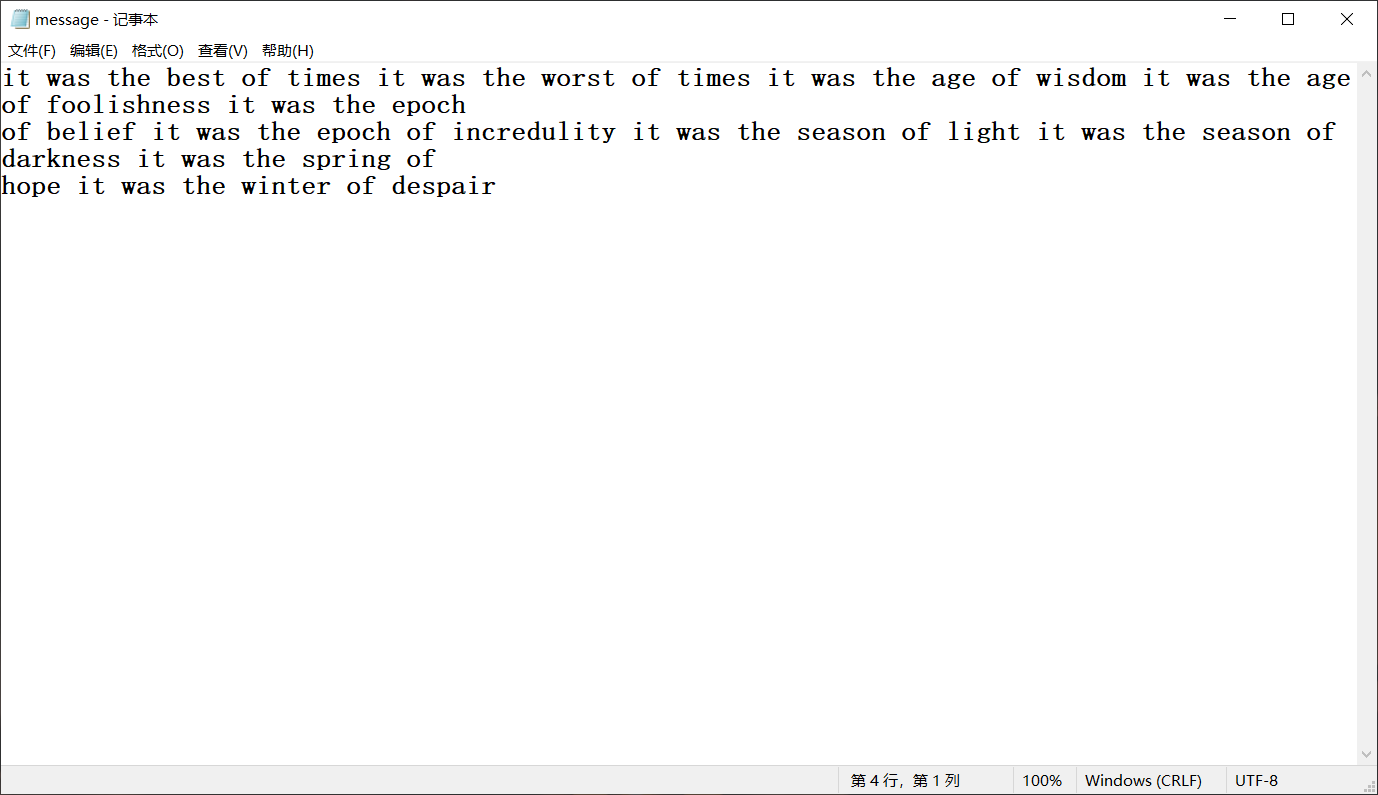
代码实现思路：

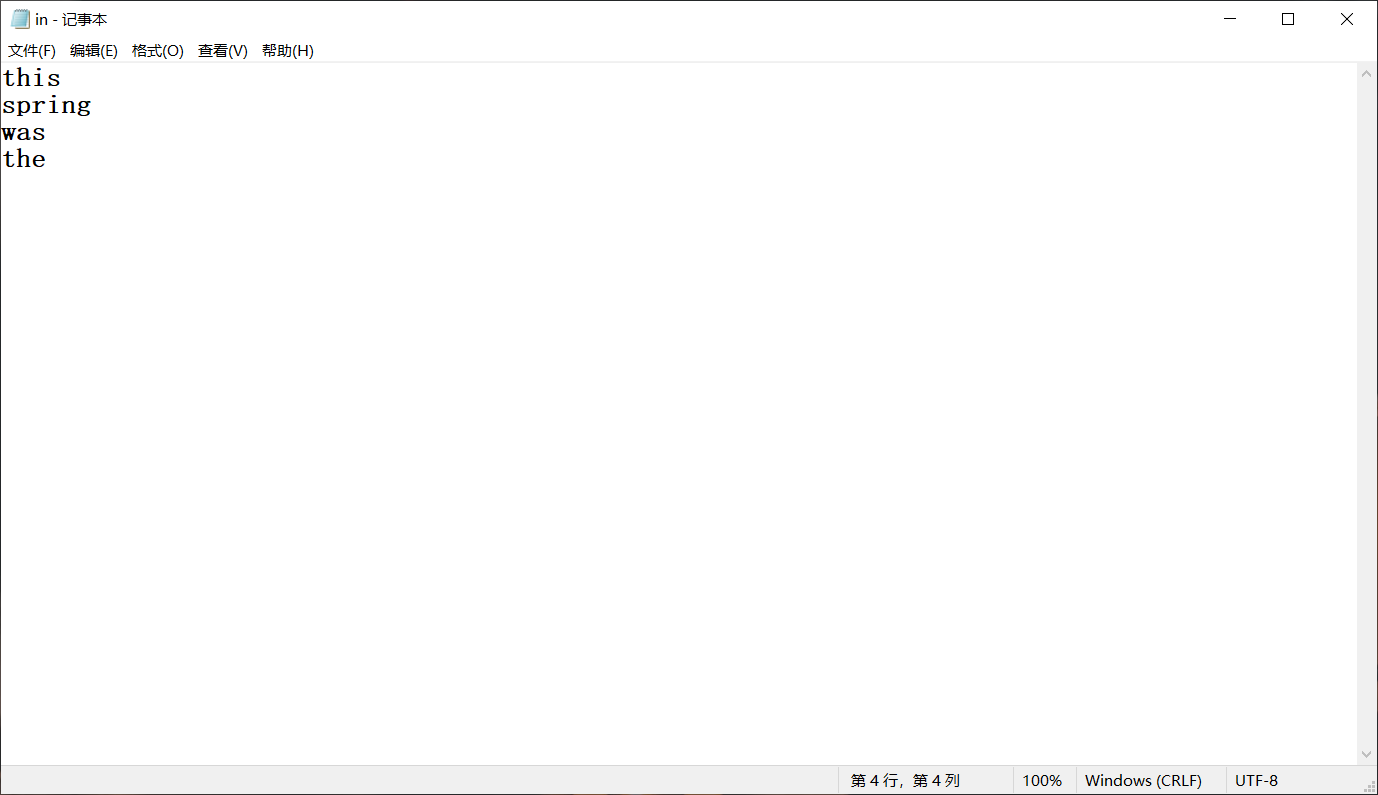
我在实现本题的算法时，并没有采用PDF中所提供的思路，而是直接采用了一种比较优秀的检索算法——Sunday算法，假设文本长度为n，所需要检索的字符串的长度为m，该算法的平均性能为O(n)，最好情况为O(n/m)，最坏情况为O(nm)，该算法的性能优于KMP算法和BM算法，因此在本次题目中我使用了该算法来检索。如果查找到了一个单词，则返回其首字母所在的指针位置，否则返回值为-1。

为了方便检验，我将所有的输入写入到了一个文件之中，以此来提高输入的效率。

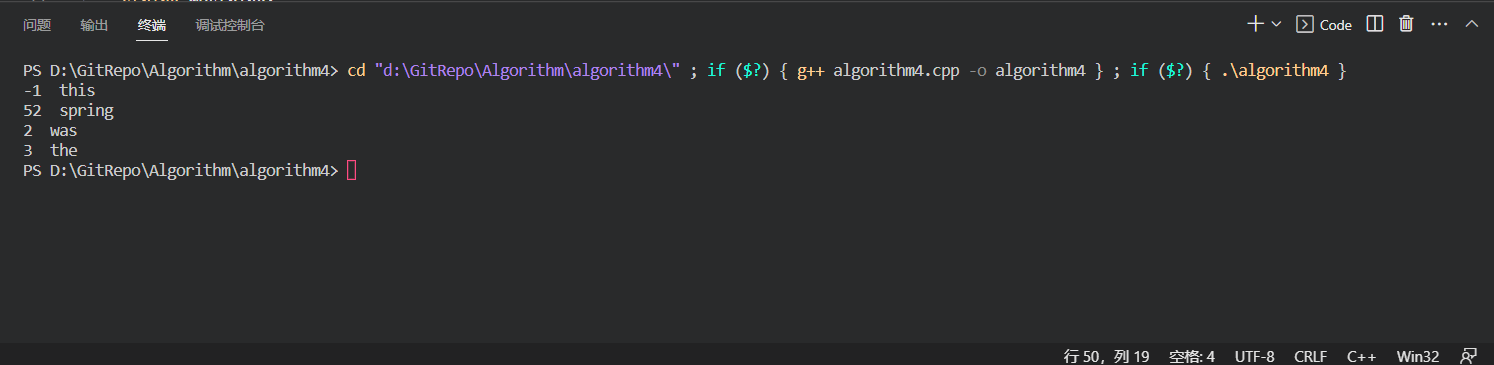
1. 实验结果

文件的输入如图所示：



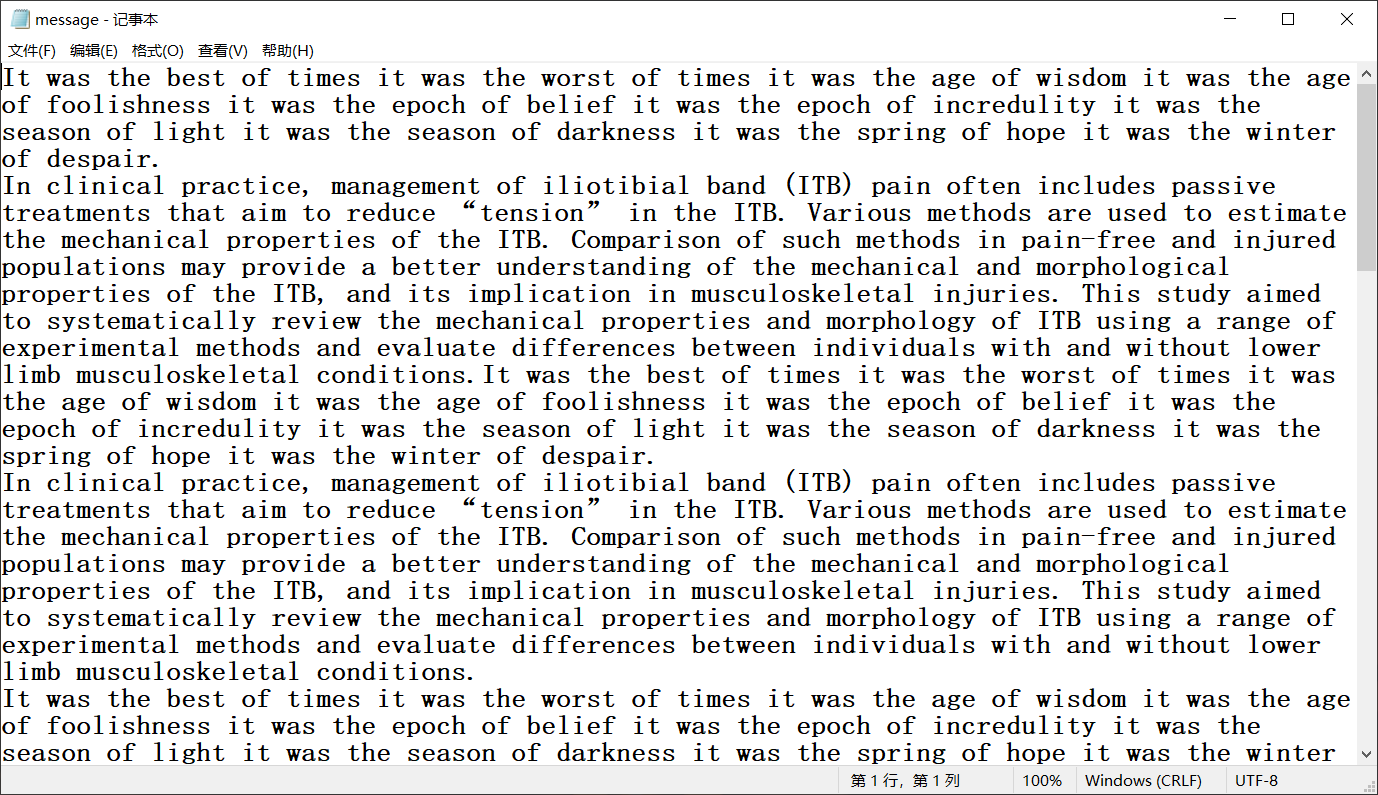


程序的输出结果如下所示：

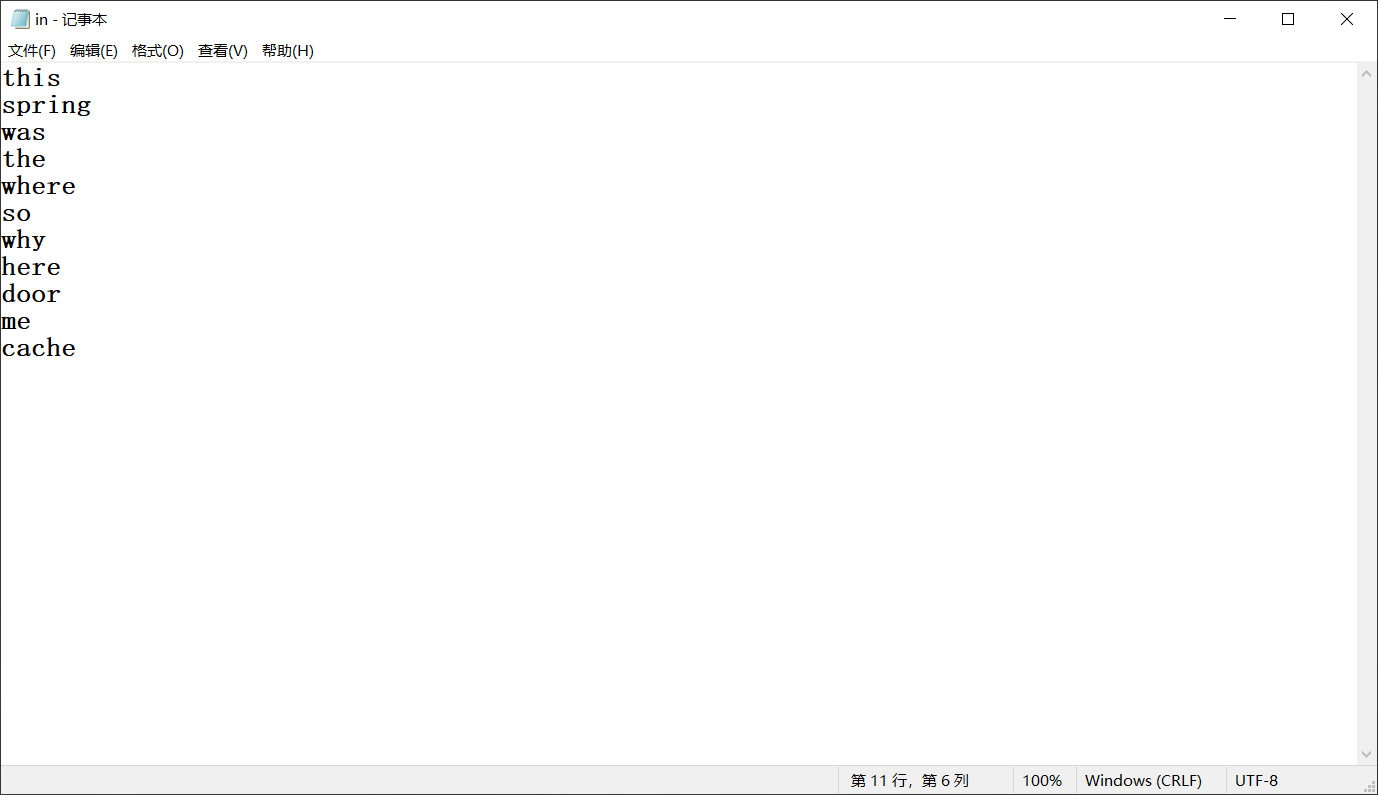


1. 结果分析

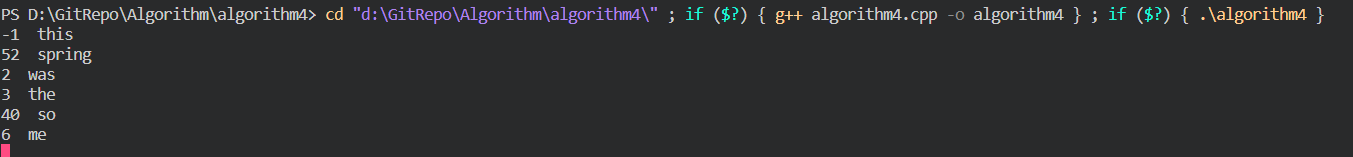
根据程序的结果不难看出程序输出是正确的，当然，为了验证结果更为广泛的正确性，我又修改了message文件，将其规模扩大，选用了很大的语料库。



改变搜索内容如下：



搜索结果如下：



可见程序是可以应用于大型数据的。