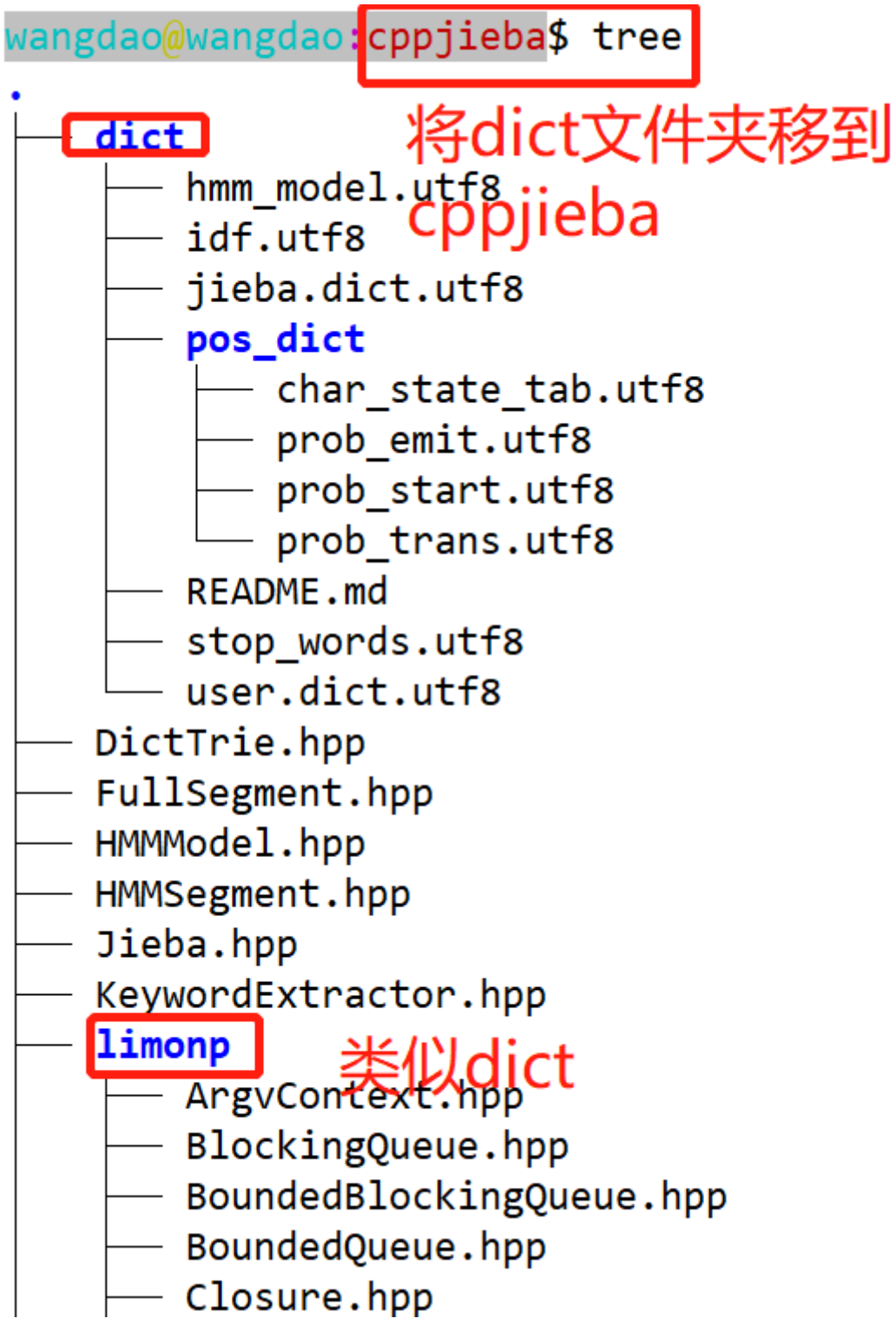
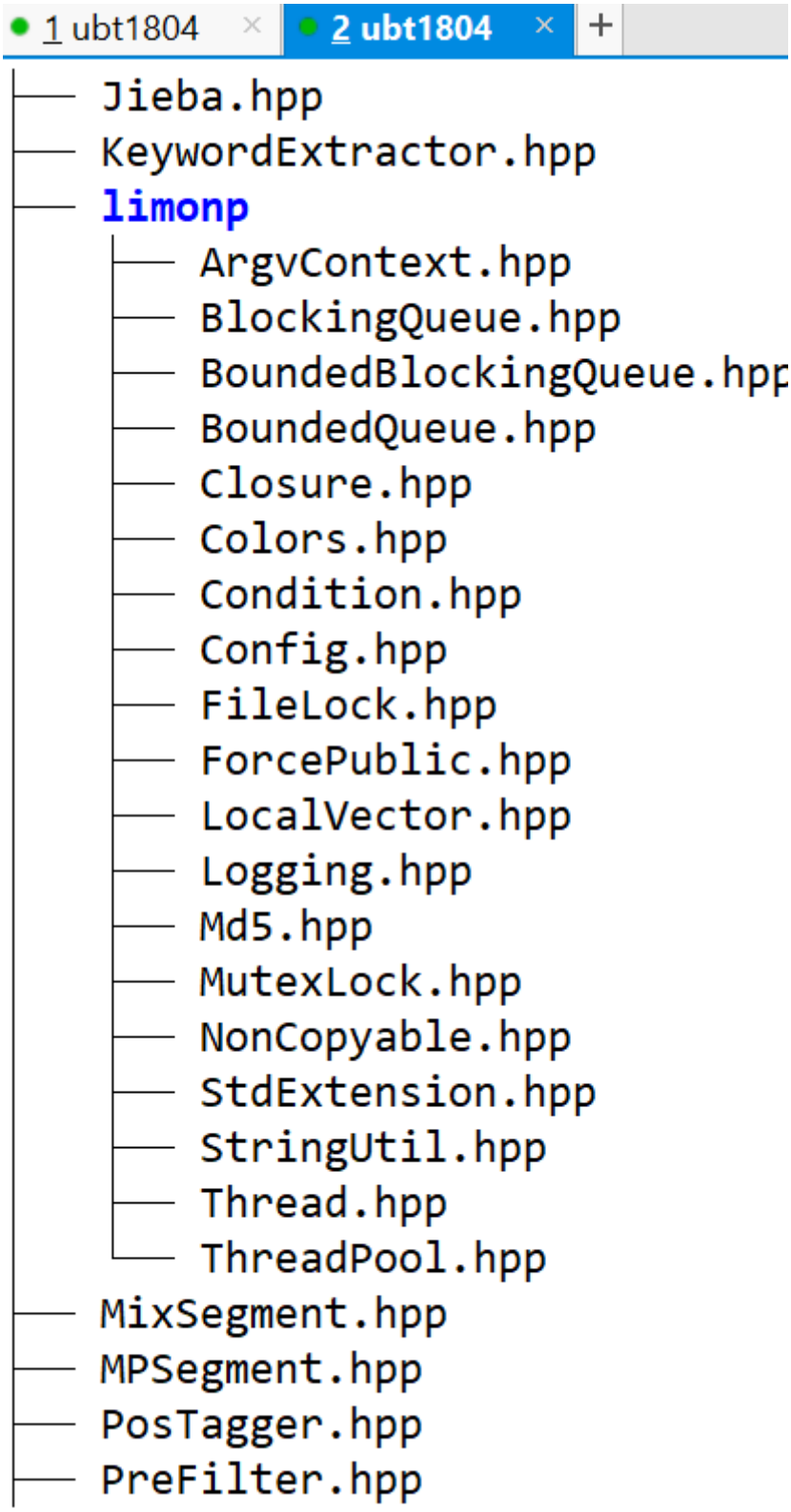
问题一、cppjieba如何安装与使用





问题二、cppjieba的使用

只需要一个Cut函数就ok

头文件

```
class SplitTool{
public:

    virtual ~SplitTool(){}  //虚析构函数
    virtual vector<string> cut(const string &sentence) = 0; //分词函数，纯虚函数提供接口
};

class SplitToolCppJieba
: public SplitTool
{
public:
    SplitToolCppJieba(Configuration & conf); //构造函数
    virtual ~SplitToolCppJieba(){}  //虚析构函数
    virtual vector<string> cut(const string &sentence);
private:
    Configuration &_conf;
};

}
#endif
```

实现文件

使用配置文件，将其存起来

```
vector<string> SplitToolCppJieba::cut(const string &sentence){
    auto map = _conf.GetConfigMap();

    const char *const DICT_PATH = map[_conf.mapKey.DICT_PATH].c_str();
    const char *const HMM_PATH = map[_conf.mapKey.HMM_PATH].c_str();
    const char* const USER_DICT_PATH = map[_conf.mapKey.USER_DICT_PATH].c_str();
    const char* const IDF_PATH = map[_conf.mapKey.IDF_PATH].c_str();
    const char* const STOP_WORD_PATH = map[_conf.mapKey.STOP_WORD_PATH].c_str();

      cppjieba::Jieba jieba(DICT_PATH,
        HMM_PATH,
        USER_DICT_PATH,
        IDF_PATH,
        STOP_WORD_PATH);
  vector<string> words;

  jieba.Cut(sentence, words);
  //cout << limonp::Join(words.begin(), words.end(), " ") << endl;

  return words;
}
```

问题三、中英文索引的建立需要使用不同的函数吗?

可以直接写到一起

```
//map<string, int> _dict;
// hello   1
//武汉  2

void DictProducer::buildIndex()
{
     int i = 0;  //记录下标
     for(auto elem : _dict)
     {
         string word = elem.first;
         size_t charNums = word.size()/getByteNum_UTF8(word[0]);
         for(size_t idx = 0, n = 0; n != charNums; ++idx, ++n)//得到字符数
```

```
            {
                //按照字符个数切割
                size_t charLen = getByteNum_UTF8(word[idx]);
                string subWord = word.substr(idx, charLen);//按照编码格式，进行拆分
                _index[subWord].insert(i);
                idx += (charLen -1);
            }
            ++i;
        }
    }

size_t DictProducer::getByteNum_UTF8(const char byte)
{
    int byteNum = 0;
    for (size_t i = 0; i < 6; ++i)
    {
        if (byte & (1 << (7 - i)))
            ++byteNum;
        else
            break;
    }

    return byteNum == 0 ? 1 : byteNum;
}
```

问题四、关键字推荐中，如何比较?

直接加载到内存中来，存放在vector<pair<string, int>>，以及将索引存放在内存中来

```
    string line;
    string word;
    int frequency;
    //插入英文词典
    while(getline(enDictIfs,line)){
        istringstream iss(line);
        iss >> word >> frequency;
        _dict.push_back(make_pair(word, frequency));
    }
    size_t cnDictOffset = _dict.size();
    //插入中文词典
    while(getline(cnDictIfs,line)){
        istringstream iss(line);
        iss >> word >> frequency;
        _dict.push_back(make_pair(word, frequency));
    }
```

```
64      //插入英文索引
65      while(getline(enIdxIfs,line)){
66          istringstream iss(line);
67          iss >> word;
68          while(iss >> index){
69              _indexTable[word].insert(index);
70          }
71      }
72      //插入中文索引
73      while(getline(cnIdxIfs,line)){
74          istringstream iss(line);
75          iss >> word;
76          while(iss >> index){
77              _indexTable[word].insert(cnDictOffset + index);
78          }
79      }
80
```

问题五、如何建立中文词典

```
61  void DictProducer::buildCnDict(){
62      getFiles();
63      for(auto &file: _files){
64          ifstream ifs(file,ios::ate); //打开文件，并将ptr指向文件末尾
65          if(!ifs.good()){
66              cerr << "openCnDict file fail" << endl;
67              return;
68          }
69          size_t length = ifs.tellg();
70          ifs.seekg(std::ios_base::beg);
71          char *buff = new char[length + 1];
72          ifs.read(buff, length + 1);
73          string txt(buff);
74
75          vector<string> tmp =  splitTool->cut(txt);
76          for(auto &i : tmp){
77              //不是停用词 且 UTF-8字节数为3
78              if(!_stopWord.count(i) && getByteNum_UTF8(i[0]) == 3){
79                  pushDict(i);
80              }
81          }
82          ifs.close();
```

整篇文章一次性加载进来，然后在分词

判断是不是中文

```
185 void DictProducer::pushDict(const string &word){
186     ++_dict[word];
187 }
```

问题六、代码如何调试

是不是出现代码就直接懵圈了？好无助，好烦，好心伤，要要放弃，要死了？我是不是不行了，NO，你是神，你是可以更加腾飞的。

```
wangdao@wangdao:20220513$ cat main.cc
#include <iostream>

using std::cout;
using std::endl;

int main(int argc, char **argv)
{
    char *pstr = "hello";
    pstr[0] = 'H';
    return 0;
}

wangdao@wangdao:20220513$ g++ main.cc -o main -g
main.cc: In function ‘int main(int, char**)’:
main.cc:8:18: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
     char *pstr = "hello";
                  ^~~~~~~
wangdao@wangdao:20220513$ ./main
Segmentation fault (core dumped)
```

段错误

```
wangdao@wangdao:20220513$ g++ main.cc -o main -g
main.cc: In function ‘int main(int, char**)’:
main.cc:8:18: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
     char *pstr = "hello";
                  ^~~~~~~
```

```
Segmentation fault (core dumped)
wangdao@wangdao:20220513$ gdb ./main
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./main...done.
(gdb) r
Starting program: /home/wangdao/teach/42th/20220513/main

Program received signal SIGSEGV, Segmentation fault.
0x0000555555554754 in main (argc=1, argv=0x7fffffffe398) at main.cc:9
9           pstr[0] = 'H';
(gdb) bt
#0  0x0000555555554754 in main (argc=1, argv=0x7fffffffe398) at main.cc:9
(gdb)
```

开始调试

l list显示代码
s单步调试
c 继续
n 下一步
p 打印变量

开始运行程序

查看堆栈信息，从下向上看

问题七、对第三方库的使用与理解?

在整个C++学习过程中，我们用过第三方库如下：log4cpp、tinyxml2、hiredis、cppjieba、nolman/json、simhash、redis-plus-plus，所以对于第三方库，大家一定要学会如何安装与使用。当然源码的阅读也是必须的，这样才能更好的使用第三方库。

问题八、项目中算法自己有研究吗？（面试需要考虑的）

最小编辑距离算法，使用到了动态规划的思想，那么什么是动态规划，这类算法的核心思想大家有没有思考，能不能掌握其核心。其实单独这个短算法，大家就可以拿出去说，以点到面，与面试官交谈。

第二个simhash算法，安装与使用比较简单，在我们的网页去重时候使用到，那么simhash的原理、思想是什么，这个也是可以与面试官谈的资本。

问题N、项目过程的体会？（面试或者项目演示时候需要考虑）

每个人都应该可以从这个项目中有所体会：

1、自己的Linux基础、C++语法基础学的怎么样，能不能在写完代码后，以思维导图的形式全面整理一次，一边整理一边看笔记，来个通盘的学习？

2、自己的码代码的水平如何，能不能将自己的思维转换为代码？如果不行，自己的代码水平后续该如何提升？

3、调试bug的能力有没有在项目中得到提升，除了之前的printf或cout外，gdb这个调试方法自己掌握的怎么样？

4、对于项目的整体掌控能力，自己在项目前后，有没有什么改观？可以通过类图、阅读代码，从main函数入手，走读代码等等。

5、项目中的相互合作，沟通能力，以及展示自己贡献那部分的表达能力，有没有得到提升？

44期追加问题

一、cppjieba与simhash在一起使用的时候会出错，这个怎么处理？

将cppjieba删除，如果之前安装了，可以卸载，直接使用simhash中的cppjieba。

附加小问题：

1、关键字推荐模块的中文部分与网页搜索部分，语料库的问题？

关键字推荐：

英文部分：直接使用English.txt、The_Holy_Bible.txt

中文部分：art下面的所有文件

网页部分：所有的xml文件

2、关键字推荐与网页搜索两个部分之间的关系？

本项目中，两个模块是独立的，没有任何关系

3、停用词集合？

stop_words_zh.txt、stop_words_eng.txt

4、七个文件生成之后，是不是每次开启程序都需要生成？

这七个文件，可以单独的写两个独立程序，分别生成关键字推荐的四个文件与网页搜索的三个文件。

二、cppjieba在使用的时候，每次都初始化不同cppjieba对象吗?

cppjieba的初始化，非常耗时，所以最好只初始化一次，或者减少cppjieba对象的初始化次数

三、如何读取目录

opendir、chdir、readdir、closedir

```
168 void DictProducer::getFiles()
169 {
170     DIR *pDir = opendir(_dir.c_str());
171     if(!pDir)
172     {
173         perror("opendir");
174         cout << _dir << endl;
175         return;
176     }
177     struct dirent *ptr;
178     while((ptr = readdir(pDir))!=0)
179     {
180         //排除.和..文件
181         if(strcmp(ptr->d_name,".")!=0 && strcmp(ptr->d_name,"..")!=0)
182         {
183             _files.push_back(_dir + "/" + ptr->d_name);
184         }
185     }
186     closedir(pDir);
187 }
```

```
1  struct dirent
2  {
3      ino_t          d_ino;       /* Inode number */
4      off_t          d_off;       /* Not an offset; see below */
5      unsigned short d_reclen;    /* Length of this record */
6      unsigned char  d_type;      /* Type of file; not supported by all
   filesystem types */
7      char           d_name[256]; /* Null-terminated filename */
8  };
```

四、最小编辑距离算法

UTF-8是一种变长字节编码方式。对于某一个字符的UTF-8编码，如果只有一个字节则其最高二进制位为0；如果是多字节，其第一个字节从最高位开始，连续的二进制位值为1的个数决定了其编码的位数，其余各字节均以10开头。UTF-8最多可用到6个字节。
如表：

```
1 1字节 0xxxxxxx
2 2字节 110xxxxx 10xxxxxx
3 3字节 1110xxxx 10xxxxxx 10xxxxxx
4 4字节 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
5 5字节 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
6 6字节 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
```

因此UTF-8中可以用来表示字符编码的实际位数最多有31位，即上表中x所表示的位。除去那些控制位（每字节开头的10等），这些x表示的位与UNICODE编码是一一对应的，位高低顺序也相同。 实际将UNICODE转换为UTF-8编码时应先去除高位0，然后根据所剩编码的位数决定所需最小的UTF-8编码位数。 因此那些基本ASCII字符集中的字符（UNICODE兼容ASCII）只需要一个字节的UTF-8编码（7个二进制位）便可以表示。

因此UTF-8中可以用来表示字符编码的实际位数最多有31位，即上表中x所表示的位。除去那些控制位（每字节开头的10等），这些x表示的位与UNICODE编码是一一对应的，位高低顺序也相同。 实际将UNICODE转换为UTF-8编码时应先去除高位0，然后根据所剩编码的位数决定所需最小的UTF-8编码位数。 因此那些基本ASCII字符集中的字符（UNICODE兼容ASCII）只需要一个字节的UTF-8编码（7个二进制位）便可以表示。

对于上面的问题，代码中给出的两个字节是
十六进制：C0 B1
二进制：11000000 10110001
对比两个字节编码的表示方式：
110xxxxx 10xxxxxx
提取出对应的UNICODE编码：
00000 110001
可以看出此编码并非“标准”的UTF-8编码，因为其第一个字节的“有效编码”全为0，去除高位0后的编码仅有6位。由前面所述，此字符仅用一个字节的UTF-8编码表示就够了。

用utf8编码格式的时候，一个英文字符占用一个字节，使用中文编码的时候，一个字符占用3个字节。

```
1
2  #include <string>
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::string;
8
9  //1. 求取一个字符占据的字节数
10 size_t nBytesCode(const char ch);
11
12 //2. 求取一个字符串的字符长度
13 std::size_t length(const std::string &str);
14
15 //3. 中英文通用的最小编辑距离算法
16 int editDistance(const std::string & lhs, const std::string &rhs);
17
18 void test0()
19 {
20     //std::string本身是一个字节流的字符串
21     // 字符流的字符串
22     string s1 = "abcd";//4个字符的字符串
23     //获取的是字符的长度
24     string s2 = "中国";//2个字符的字符串
25     for(auto & ch : s2) {
26         cout << ch;
27     }
28     cout << endl;
29     cout << "s2[1]: " << s2[1] << endl;
30     cout << endl;
31     cout << "s1.size() : " << s1.size() << endl;
32     cout << "s2.size() : " << s2.size() << endl;
```

```cpp

    string s3 = s2.substr(0, 3);
    cout << "s3.size(): " << s3.size() << endl;
    cout << "s3: " << s3 << endl;

    string s4 = s2.substr(1, 3);
    cout << "s4.size(): " << s4.size() << endl;
    cout << "s4: " << s4 << endl;

    //字符的长度
    string s5 = "abc中国";
    cout << "s5.size(): " << s5.size() << endl;
    cout << "s5.length: " << length(s5) << endl;

    // 中国   => h中国 => a中国 => ab中国 => abc中国
    // s2经过编辑之后，变成s5
    cout << "s2与s5的最小编辑距离: " << editDistance(s2, s5) << endl;

    //dp[0][0]
    //dp[i][j] =
    //A: 空串
    //B: abc
}

int main(void)
{
    test0();
    return 0;
}

size_t nBytesCode(const char ch)
{
    if(ch & (1 << 7))
    {
        int nBytes = 1;
        for(int idx = 0; idx != 6; ++idx)
        {
            if(ch & (1 << (6 - idx)))
            {
                ++nBytes;
            }
            else
                break;
        }
        return nBytes;
    }
    return 1;
}

std::size_t length(const std::string &str)
{
    std::size_t ilen = 0;
    for(std::size_t idx = 0; idx != str.size(); ++idx)
    {
        int nBytes = nBytesCode(str[idx]);
        idx += (nBytes - 1);
        ++ilen;
    }
```

```
    return ilen;
}

int triple_min(const int &a, const int &b, const int &c)
{
    return a < b ? (a < c ? a : c) : (b < c ? b : c);
}

int editDistance(const std::string & lhs, const std::string &rhs)
{
    //计算最小编辑距离-包括处理中英文
    size_t lhs_len = length(lhs);
    size_t rhs_len = length(rhs);
    int editDist[lhs_len + 1][rhs_len + 1];
    for(size_t idx = 0; idx <= lhs_len; ++idx)
    {
        editDist[idx][0] = idx;
    }

    for(size_t idx = 0; idx <= rhs_len; ++idx)
    {
        editDist[0][idx] = idx;
    }

    std::string sublhs, subrhs;
    for(std::size_t dist_i = 1, lhs_idx = 0; dist_i <= lhs_len; ++dist_i,
++lhs_idx)
    {
        size_t nBytes = nBytesCode(lhs[lhs_idx]);
        sublhs = lhs.substr(lhs_idx, nBytes);
        lhs_idx += (nBytes - 1);

        for(std::size_t dist_j = 1, rhs_idx = 0;
            dist_j <= rhs_len; ++dist_j, ++rhs_idx)
        {
            nBytes = nBytesCode(rhs[rhs_idx]);
            subrhs = rhs.substr(rhs_idx, nBytes);
            rhs_idx += (nBytes - 1);
            if(sublhs == subrhs)
            {
                editDist[dist_i][dist_j] = editDist[dist_i - 1][dist_j -
1];
            }
            else
            {
                editDist[dist_i][dist_j] =
                    triple_min(editDist[dist_i][dist_j - 1] + 1,
                    editDist[dist_i - 1][dist_j] + 1,
                    editDist[dist_i - 1][dist_j - 1] + 1);
            }
        }
    }
    return editDist[lhs_len][rhs_len];
}
```

代码里面有没有考虑中英文混合查询的问题，这样代码是不是会崩溃?

常用算法思想：枚举（穷举，暴力破解）、递推（顺推、逆推）、递归（汉诺塔）、分治（将一个规模为N的问题分解为K个规模较小的子问题）、贪心（从问题的某一个初始解出发，逐步逼近给定的目标，以便尽快求出更好的解）、试探法（回溯）、动态迭代（是对问题状态的定义和状态转移方程的定义（一个状态---另外一个状态的））和模拟算法思想（对真实事物或者过程的虚拟）。

递推：与枚举算法思想相比，递推算法能够通过已知的某个条件，利用特定的关系得出中间推论，然后逐步递推，直到得到结果为止。由此可见，递推算法要比枚举算法聪明，它不会尝试每种可能的方案

五、几个第三方库的路径

结巴分词库：https://github.com/yanyiwu/cppjieba
json库：https://github.com/nlohmann/json
hiredis库：https://github.com/redis/hiredis
redis库redis-plus-plus：https://github.com/sewenew/redis-plus-plus
simhash库：https://github.com/yanyiwu/simhash

六、缓存系统的设计（加分项）

1、缓存设计多少个?

如果只有一个查询缓存，那么每个查询是由一个工作线程完成的，多个工作线程要对这一个缓存进行读写操作，就会设计到加锁、解锁的问题，那么对客户端的响应就会造成影响，假如1s中内有100个请求进行处理，那么每个请求会在10ms内完成。所以不能设置为1个

2、每一个工作线程对应一个缓存?

每个工作线程会对应一个cache，然后将1号线程的cache作为全局cache，为了防止断点重启之后cache中的数据会丢失，可以将缓存中的数据备份到文件中去（类似redis的持久化操作）缓存的同步更新，将2、3、4号线程中的数据写到1号线程中，然后在将1号线程中的数据同步到2、3、4号线程中来，这是思路。LRU缓存的淘汰算法，list中存放的是数据（key,value），查找速度是O(N),然后使用unordered_map，其中存储的是元素在list中的迭代器

3、如何将各个缓存进行更新

进行一段时间的运行，每个cache肯定会产生数据不一致的问题，所以可以使用一个数据结构存储从上一次更新之后，到下次更新之前产生的记录，我们称之为待更新的记录，就使用list<pair<string,string>>进行存储。list<pair<key,json>> pendingUpdateRecord;cache大小的设置也是一个问题，比如cache大小设置的比较小，比如是1000，当cache2中有400个热数据，cache3中也有400个热数据，cache4中也有400个热数据，那么这些数据同步到cache1中时候，有部分数据是存储不进去的，所以cache的大小设置也是需要设计的，但是当比较稳定的时候，每个cache中的热数据应该是不多的，那么这些数据是都会被存在cache1中，然后都同步到cache2、3、4中的。

4、如何无缝更新？（先不考虑，越想越复杂）

因为在更新的同时，查询的请求是不能中断的。类似之前在Redis中的持久化功能一样，让主进程fork出子进程，子进程负责读操作，父进程负责写操作，然后再同步。

5、持久化

类似于redis作为缓存，需要将内存（缓存）中的数据保存到磁盘中，做持久化。然后断电重启之后，缓存中的数据可以直接使用磁盘中的数据。

七、数据传输格式

json库：https://github.com/nlohmann/json

八、数据显示的格式

关键字推荐

hello：hello hell

武汉：武汉、武功、武昌、武湖、江汉

网页搜索

王道在线科技公司

标题 + 链接 + 摘要（频率最高的20单词，文章的前100字符）

摘要：可以使用静态摘要；也可以使用动态摘要。