

IR 大作业：自适应文本检索系统的实现

姓名：吴朱冠宇

学号：22920182204320

日期：2021 年 5 月 26 日

摘 要

此次作业完成了任务：“每一次检索后对返回的文档执行相关反馈的判断，重新生成查询”。项目全部使用 C++ 语言，并且独自从零开始编写。代码总量为 18KB、530 行。由于文本检索需要服务器与客户端，此项目使用 socket 编程进行通信。项目代码发布在 GitHub 上：<https://github.com/GoatWu/XMU-IR-Information-Retrieval-System>

1 介绍

1.1 文件架构

1. initialize.cpp

用于初始化服务器，即构造向量空间模型。这里包括：

- 获取全部文档的绝对路径，并将文档与一个数字编号一一映射；
- 读取全部文档，并将所有单词与一个数字编号一一映射；
- 构造词频矩阵 $tf_{t,d}$ ；
- 构造文档频率向量 df ；
- 构造 $tf-idf$ 权重矩阵，并且进行余弦归一化；

2. myfunc.cpp

用于提供各种函数支持，并且定义全局变量（如：词频矩阵 $tf_{t,d}$ 、文档频率向量 df 等）。各函数的功能将在下文详细介绍。

3. server.cpp

此文件是服务器代码。首先的工作是初始化服务器，这里用到了 `initialize.cpp` 中的各个函数；然后是建立 socket 服务，绑定服务器管理员指定的端口后监听此端口。当有客户端进程来 `connect` 的时候，主进程会 `fork` 一个子进程与其通信，以满足多用户同时查询；每次用户查询结束之后，服务器会给客户端提供 3 个选项：

- (a) 提供相关反馈信息，以取得更为精确的查询结果；
- (b) 不提供相关反馈信息，继续新的查询；
- (c) 退出查询。

4. client.cpp

此文件是客户端代码。客户端负责向服务器发送查询、接受信息，直到客户端用户输入 `bye()` 或者按下 `control+C` 强制退出。

1.2 编译运行

1. 编译

由于代码使用了 lambda 函数(匿名函数)等 C++11 特性,需加入编译选项 `-std=c++11`。由图1可见,编译生成了两个可执行文件 `server` 和 `client`, 分别对应服务器程序和客户端程序。

```
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject > g++ server.cpp -std=c++11 -o server
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject > g++ client.cpp -std=c++11 -o client
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject > ls -al
total 824
drwxr-xr-x  11 GoatWu  staff   352  5 27 01:08 .
drwxr-xr-x   8 GoatWu  staff   256  5 27 00:05 ..
-rw-r--r--   1 GoatWu  staff  6148  5 26 14:31 .DS_Store
drwxr-xr-x   5 GoatWu  staff   160  5 25 11:49 .vscode
-rwxr-xr-x   1 GoatWu  staff  51480  5 27 01:08 client
-rw-r--r--   1 GoatWu  staff  1853  5 26 20:29 client.cpp
drwxrwxr-x   8 GoatWu  staff   256  5 26 17:50 datafile
-rw-r--r--   1 GoatWu  staff   3748  5 26 15:32 initialize.cpp
-rw-r--r--   1 GoatWu  staff   6295  5 26 17:55 myfunc.cpp
-rwxr-xr-x   1 GoatWu  staff 333640  5 27 01:00 server
-rw-r--r--   1 GoatWu  staff   6376  5 26 23:57 server.cpp
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject >
```

图 1: 使用 `-std=c++11` 选项编译文件

2. 运行

打开两个终端。第一个终端运行服务器进程。输入命令: `./server 12345`, 表示服务器运行在本机 12345 号端口上;第二个终端对应客户端进程。输入命令: `./client localhost 12345`, 表示客户端请求和本机的 12345 号端口建立通信。服务器和客户端的运行效果如图2和图3所示:

```
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject > ./server 12345
Total file count: 145
different words: 8260
等待询问.....
Accept 127.0.0.1:49333
```

图 2: 运行服务器

```
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject > ./client localhost 12345
Server:
请输入查询:
Me :
```

图 3: 运行客户端

2 代码详解

2.1 initialize.cpp

2.1.1 全局变量定义

```
1 #include "myfunc.cpp"
2 const int MAXFILENAME = 10000;
3 char buffer[MAXFILENAME];
4 extern std::vector<std::string> filePaths;    // 编号 -> 文档
5 extern std::vector<std::string> words;       // 编号 -> 单词
6 extern std::map<std::string, int> fileNum;    // 文档 -> 编号
7 extern std::map<std::string, int> wordNum;    // 单词 -> 编号
8 extern std::set<char> separationChar;        // 分隔符
9 extern std::vector<int> df;                  // 出现词项的文档数目
10 extern std::vector< std::vector<int> > tf;    // 词频矩阵
11 extern std::vector< std::vector<double> > tf_idf; // TF-IDF 权重
12 extern std::vector< std::vector<double> > Cos_weight; // 余弦归一化之后的权重
```

这里从 myfunc.cpp 中，引入了各个全局变量：

- `const int MAXFILENAME = 10000`
文件完整路径的最大长度；
- `char buffer[MAXFILENAME]`
用于读取文件路径的缓冲区；
- `extern std::vector<std::string> filePaths`
记录全部文件路径的可变长数组。同时，它也是从编号到文件路径的映射；
- `extern std::vector<std::string> words`
记录全部单词的可变长数组。同时，它也是从编号到单词的映射；
- `extern std::map<std::string, int> fileNum`
记录从完整路径到编号的映射。和前面的可变长数组 `filePaths` 构成了一一映射；
- `extern std::map<std::string, int> wordNum`
记录从单词到编号的映射。和前面的可变长数组 `words` 构成了一一映射；
- `extern std::set<char> separationChar`
记录全部的标点符号，用于将文章划分词项；
- `extern std::vector<int> df`
文档频率向量。即记录各个单词在多少个文档中出现过；
- `extern std::vector< std::vector<int> > tf`
词频矩阵。即记录每个单词在每篇文档中出现的次数；
- `extern std::vector< std::vector<double> > tf_idf`
通过 `df` 和 `tf` 计算出来的权重矩阵；
- `extern std::vector< std::vector<double> > Cos_weight`
将每篇文档对应的向量归一化后得到的矩阵。

2.1.2 获取全部文件的绝对路径

```
1 int initFilePaths() {
2     getcwd(buffer, sizeof(buffer));
3     std::string path;
4     path.assign(buffer);
5     path += "/datafile/";
6
7     getFiles(path, filePaths);
8
9     for (size_t i = 0; i < filePaths.size(); i++) {
10         fileNum[filePaths[i]] = i;
11         // STD::COUT << FILEPATHS[I] << "\n";
12     }
13     return (int) filePaths.size();
14 }
```

首先使用 `getcwd` 函数获取了当前的完整目录，并使用 `std::string.assign` 方法将其转化为 `std::string` 类型。由于数据文件在 `/datafile/` 目录下，因此将根目录 `path` 后面添加了 `/datafile/`。后面用到了 `myfunc.cpp` 文件中的 `getFile` 函数，将读取到的全部文件的完整路径存入 `filePaths` 向量中，并利用一一映射的关系完成对 `fileNum` 的初始化。`getFile` 函数如下：

```
1 void getFiles(std::string path, std::vector<std::string>& files) {
2     std::string path0 = path;
3     DIR* pDir;
4     struct dirent* ptr;
5     struct stat s;
6     lstat(path.c_str(), &s);
7
8     if (!S_ISDIR(s.st_mode) || !(pDir = opendir(path.c_str()))) {
9         std::cout << "not a valid directory: " << path << "\n";
10        return;
11    }
12
13    std::string subFile;
14    while ((ptr = readdir(pDir)) != 0) {
15        subFile = ptr->d_name;
16        if (subFile == "." || subFile == "..") continue;
17        struct stat tmp;
18        subFile = path0 + subFile + "/";
19        lstat(subFile.c_str(), &tmp);
20        if (S_ISDIR(tmp.st_mode)) {
21            getFiles(subFile, files);
22        }
23        else {
24            subFile.pop_back();
```

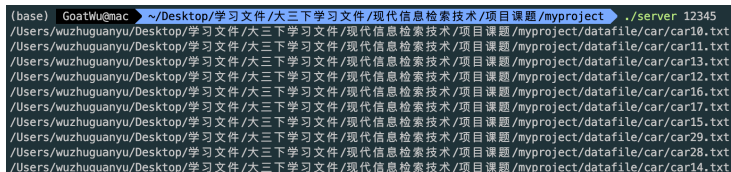
```

25     files.push_back(subFile);
26 }
27 }
28 closedir(pDir);
29 }

```

其思路是，先确定当前位置是目录；然后读取除了.和..之外的所有子项，如果是文件则将其 `push_back` 到入 `filePaths` 向量中，否则进行递归查询。

我们可以尝试输出所有的路径，效果如图4所示：



```

(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject ./server 12345
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car10.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car11.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car13.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car12.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car16.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car17.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car15.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car29.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car28.txt
/Users/wuzhuguan/Deskto/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject/datafile/car/car14.txt

```

图 4：获得全部文件的绝对路径

2.1.3 记录标点符号

```

1 void initSeperationChar() {
2     std::set<char> tmp{
3         ' ', 13, 10, '\t', ',', '.', ';',
4         ':', '?', '!', '\'', '\\";
5     };
6     separationChar = tmp;
7 }

```

这里选用 `std::set<char>` 来记录标点符号的原因是，我们可以用简短的代码和较低的复杂度 ($O(\log n)$) 判断一个字符是否属于标点符号。

2.1.4 获得全部单词及其与编号的映射

```

1 int initWordsNum() {
2     int wordCnt = 0;
3     for (size_t i = 0; i < filePaths.size(); i++) {
4         std::set<std::string> used; // 单词在当前文档中是否出现过
5         std::ifstream t(filePaths[i]);
6         std::stringstream buffer;
7         buffer << t.rdbuf();
8         std::string contents(buffer.str());
9         std::vector<std::string> w = split(contents, separationChar);
10        for (size_t j = 0; j < w.size(); j++) {
11            std::string now = toLower(w[j]);
12            if (!wordNum.count(now)) {
13                wordNum[now] = wordCnt++;
14                df.push_back(1);

```

```

15         used.insert(now);
16         words.push_back(now);
17     }
18     else if (!used.count(now)) {
19         df[wordNum[now]]++;
20         used.insert(now);
21     }
22 }
23 t.close();
24 }
25 return wordCnt;
26 }

```

这里的主要思想是：

我们按顺序读取所有的文档，每篇文档读入到一个 `std::string` 中，并用 `myfunc.cpp` 文件中的 `split` 函数将之分解为一系列单词，装入到 `std::vector<std::string>` 中。

对于每个单词，如果原来没有被读入过（即在 `std::string` 到 `int` 的映射中没有找到相应的键），则将其赋予一个新的编号（即将其 `push_back` 到入 `words` 向量中），并在 `wordNum` 中做相应的映射。

这里我们还可以一同将文档频率向量 `df` 完成初始化：我们用一个 `std::set<std::string>` 类型的集合 `used` 来表示当前文档中出现的单词集合。下面来讨论遍历到单词的各种情况：

1. 此单词在所有文档中第一次出现

由前文可知，此单词要被赋予一个新的编号（即将其 `push_back` 到入 `words` 向量中），此时出现这个单词的文档数一定是 1，因此我们也在 `df` 向量中 `push_back` 一个 1，同时把这个单词加入到当前文档出现过的单词集合中；

2. 此单词出现过，但是在当前文档中第一次出现

由于此单词出现过，我们可以通过 `wordNum` 的映射找出其对应的编号，然后令出现过此单词的文档数加 1，同时把这个单词加入到当前文档出现过的单词集合中；

3. 此单词在当前文档中也出现过

此单词不会产生任何影响。不用处理。

这里有个要注意的地方，由于单词存在着大小写，但是其意思本质上是相同的，因此我们用 `myfunc.cpp` 文件中的 `toLower` 函数将单词转化为全部小写。下面介绍 `toLower` 和 `split` 两个函数：

```

1 std::string toLower(std::string s) {
2     std::string res = s;
3     for (size_t i = 0; i < s.size(); i++) {
4         if (s[i] >= 'A' && s[i] <= 'Z') {
5             res[i] += 'a' - 'A';
6         }
7     }
8     return res;
9 }

```

```

10
11 std::vector<std::string> split(const std::string &str,
12                               const std::set<char> &pattern) {
13     // CONST CHAR* CONVERT TO CHAR*
14     char * strc = new char[strlen(str.c_str())+1];
15     strcpy(strc, str.c_str());
16     size_t len = strlen(strc);
17     std::vector<std::string> resultVec;
18     char *tmpStr = strc;
19     for (size_t i = 0; i < len; i++) {
20         if (pattern.count(strc[i])) {
21             strc[i] = 0;
22             if (tmpStr[0] != 0) {
23                 resultVec.push_back(std::string(tmpStr));
24             }
25             tmpStr = strc + i + 1;
26         }
27     }
28     if (tmpStr[0] != 0) {
29         resultVec.push_back(std::string(tmpStr));
30     }
31     delete[] strc;
32     return resultVec;
33 }

```

`toLower` 函数很简单，只要把所有的大写字母变小写即可；

`split` 函数有两个参数，`str` 是我们分解的字符串，`pattern` 是分隔符集合。我们先 `new` 一个字符串数组作为 `const std::string &str` 的深拷贝对象，使之可编辑；然后双指针扫描此串，遇到分隔符则将之变为 `\0`，并将两个指针之间的单词转化为 `std::string` 类型后推入结果 `std::vector<std::string>`。最后要注意，把 `new` 的数组 `delete` 掉，防止内存泄漏。

2.1.5 构造词频矩阵

```

1 void initTf() {
2     tf.resize(words.size());
3     for (size_t i = 0; i < tf.size(); i++) {
4         tf[i].resize(filePaths.size());
5     }
6     std::ifstream inFile;
7     for (size_t i = 0; i < filePaths.size(); i++) {
8         std::ifstream t(filePaths[i]);
9         std::stringstream buffer;
10        buffer << t.rdbuf();
11        std::string contents(buffer.str());
12        std::vector<std::string> w = split(contents, separationChar);
13        for (size_t j = 0; j < w.size(); j++) {

```

```

14         std::string now = toLower(w[j]);
15         tf[wordNum[now]][i]++;
16     }
17     t.close();
18 }
19 }

```

重新读取一遍所有文件，每篇文档的每个单词都对应词频矩阵的一个位置。每次将其加 1 即可。

2.1.6 构造 tf-idf 权重矩阵并余弦归一化

```

1 void initWeight() {
2     int n = filePaths.size();
3     int m = words.size();
4     tf_idf.resize(m);
5     Cos_weight.resize(m);
6     for (size_t i = 0; i < m; i++) {
7         tf_idf[i].resize(n);
8         Cos_weight[i].resize(n);
9     }
10    for (size_t i = 0; i < m; i++) {
11        for (size_t j = 0; j < n; j++) {
12            if (tf[i][j] == 0) tf_idf[i][j] = 0;
13            else {
14                double w_df = 1 + log10((double)tf[i][j]);
15                double idf_t = log10(1.0 * n / df[i]);
16                tf_idf[i][j] = w_df * idf_t;
17            }
18        }
19    }
20    for (size_t j = 0; j < n; j++) {
21        double sum = 0, sqrtsum;
22        for (size_t i = 0; i < m; i++) {
23            sum += tf_idf[i][j] * tf_idf[i][j];
24        }
25        sqrtsum = sqrt(sum);
26        for (size_t i = 0; i < m; i++) {
27            Cos_weight[i][j] = tf_idf[i][j] / sqrtsum;
28        }
29    }
30 }

```

tf-idf 权重矩阵构造公式：

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) \cdot \log_{10} \frac{N}{df_t} \quad (1)$$

利用公式1可以很快写出代码。这里要特别注意 $tf_{t,d} = 0$ 的情况，应该直接将权重赋值 0，以免出现计算 $\log_{10} 0 = -\text{INF}$ 的情况。

向量归一化公式：

$$\vec{q}' = \frac{\vec{q}}{\sqrt{\sum_{i=1}^{|q|} q_i^2}} \quad (2)$$

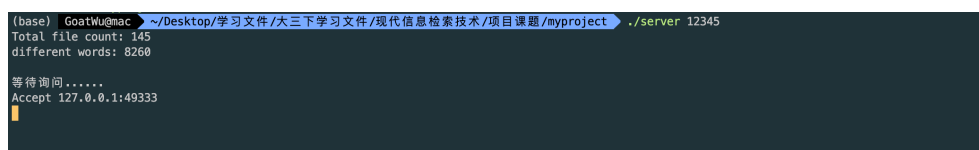
利用公式2也可以很快写出归一化代码。

2.2 server.cpp

2.2.1 调用 initialize.cpp 中的函数

```
1 initSeperationChar();
2 int fileCnt = initFilePaths();
3 int wordCnt = initWordsNum();
4 initTf();
5 initWeight();
6 std::cout << "Total file count: " << fileCnt << "\n";
7 std::cout << "different words: " << wordCnt << "\n\n";
```

初始服务端运行结果如图5所示。可以看见，总共有 145 个文件、8260 个不同的单词。



```
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject $ ./server 12345
Total file count: 145
different words: 8260
等待询问.....
Accept 127.0.0.1:49333
```

图 5: 初始服务端运行结果

2.2.2 建立 socket 服务

```
1 if (argc > 2 || argc < 2 || atoi(argv[1]) == 0) {
2     printf("usage: ./server <port>\n");
3     exit(1);
4 }
5 int sockfd;
6 if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
7     perror("create socket error");
8 }
9 struct sockaddr_in server_addr;
10 struct sockaddr_in client_addr;
11 server_addr.sin_family = AF_INET;
12 server_addr.sin_port = htons(atoi(argv[1]));
13 server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //INADDR_ANY表示本机所有IP地址
14 memset(&server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充
15 //绑定SOCKET与地址
```

```

16 bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)); //监听SOCKET
17 listen(sockfd, 0);
18 int cfd;
19 int num = 0;
20 pid_t pid;
21 char client_ip[128];
22 printf("等待询问.....\n");
23 while (233) {
24     socklen_t len = sizeof(client_addr);
25     if ((cfd = accept(sockfd, (struct sockaddr *)&client_addr, &len)) == -1) {
26         perror("accept error");
27         exit(0);
28     }
29     // 清除僵尸进程
30     signal(SIGCHLD, SIG_IGN);
31     // 获取客户端IP地址、端口
32     const char *ip = inet_ntop(AF_INET, &client_addr.sin_addr.s_addr, client_ip, sizeof(
        client_ip));
33     const int port = ntohs(client_addr.sin_port);
34     printf("Accept %s:%d\n", ip, port);
35     fflush(stdout);
36     pid = fork();

```

首先需要判断程序的参数个数，需要指定一个服务器的端口号。接下来确定服务器的协议、IP 地址、端口、零填充，并将其与 socket 绑定。然后就可以监听端口、等待客户端的连接了。

由于服务器是永不停止的运行的，因此要在最外层套一个死循环。对于每个接受了客户端请求，我们分出一个子进程去处理，这样可以让服务器同时处理多个查询。

2.2.3 查询处理

```

1 if (pid == 0) {
2     close(sockfd);
3     memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
4     ssize_t tag;
5     send(cfd, "请输入查询: ", strlen("请输入查询: "), 0);
6     while ((tag = recv(cfd, recv_msg, sizeof(recv_msg), 0)) != 0) {
7         std::string msg(recv_msg);
8         if (tag == -1) perror("receive error");
9         if (msg == "bye()") {
10             break;
11         }
12         std::string sendstr("搜索结果:\n");
13         std::string queryMsg = msg;
14         std::vector<int> answer = query(msg);
15         sendstr = getResultStr(answer);
16         sendstr += "输入「1」进行相关反馈查询, \n";

```

```

17     sendstr += "输入「2」进行新的词项查询, \n";
18     sendstr += "输入「3」结束查询: \n";
19     send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
20     memset(recv_msg, 0, sizeof(recv_msg));
21     recv(cfd, recv_msg, sizeof(recv_msg), 0);
22     msg = recv_msg;
23     while (msg[0] != '1' && msg[0] != '2' && msg[0] != '3') {
24         // STD::COUT << "WTF? " << MSG << FFLUSH;
25         sendstr = "输入「1」进行相关反馈查询, \n";
26         sendstr += "输入「2」进行新的词项查询, \n";
27         sendstr += "输入「3」结束查询: \n";
28         send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
29         memset(recv_msg, 0, sizeof(recv_msg));
30         recv(cfd, recv_msg, sizeof(recv_msg), 0);
31         msg = recv_msg;
32     }
33     if (msg[0] == '1') {
34         sendstr = "请输入相关文档以供相关反馈查询, 用逗号(,)隔开. \n";
35         sendstr += "注: 如果格式不符合, 则此次查询取消相关反馈: \n";
36         send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
37         memset(recv_msg, 0, sizeof(recv_msg));
38         recv(cfd, recv_msg, sizeof(recv_msg), 0);
39         msg = recv_msg;
40         std::set<char> u{'', ''};
41         std::vector<std::string> tmp = split(msg, u);
42         std::vector<int> isRel(10);
43         std::vector<int> relevant;
44         std::vector<int> irrelevant;
45         int okFlag = 1;
46         for (size_t i = 0; i < tmp.size(); i++) {
47             if (!isnumber(tmp[i])) {
48                 okFlag = 0;
49                 break;
50             }
51             int relevantID = atoi(tmp[i].c_str()) - 1;
52             if (relevantID < 0 || relevantID > 9) {
53                 okFlag = 0;
54                 break;
55             }
56             isRel[relevantID] = 1;
57         }
58         for (size_t i = 0; i < isRel.size(); i++) {
59             if (isRel[i]) relevant.push_back(answer[i]);
60             else irrelevant.push_back(answer[i]);
61         }
62         if (!okFlag) {
63             sendstr = "非法的反馈序列! 本次查询结束! \n";

```

```

64         sendstr += "请输入查询: ";
65         send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
66     }
67     else {
68         std::vector<int> answer = feedback(queryMsg, relevant, irrelevant);
69         sendstr = getResultStr(answer);
70         sendstr = "相关反馈查询结果: \n" + sendstr;
71         sendstr += "请输入查询: ";
72         send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
73     }
74 }
75 else if (msg[0] == '2') {
76     send(cfd, "请输入查询: ", strlen("请输入查询: "), 0);
77 }
78 else {
79     break;
80 }
81 memset(recv_msg, 0, sizeof(recv_msg));
82 }
83 }

```

上一节讲到，我们用一个子进程来处理查询。我们知道，在子进程中，有 `pid=0`。因此全部的查询处理工作处于 `if (pid == 0)` 的条件判断中。

首先向客户端发送消息：“请输入查询:”；而后接受客户端发送过来的查询信息。这里需要判断，如果客户端发来的消息是 `bye()`，则查询结束。接下来我们需要开始处理查询。这里首先用到了 `myfunc.cpp` 中的函数 `query`，其作用是对于一个询问字符串，查询与之最为相关的 10 个文档的编号，具体实现见第 2.3.2 小节；然后调用了 `getResultStr` 函数，其作用是对于刚刚获得的文档编号向量，得出对应的文档名称，并以一个字符串的形式返回，具体实现见第 2.3.3 小节；经过一些细节处理后，服务器将结果字符串返回给客户端，并且给客户端 3 个选项：

1. 进行相关反馈查询；
2. 进行新的词项查询；
3. 结束查询；

此时如果客户端输入不合法，则重复发送选项消息给客户端，直到客户端正确输入。如果客户端输入“2”，则结束此次查询、开始新的查询，也即服务器发送给客户端“请输入查询:”；如果客户端输入“3”，则退出循环，当前子进程结束后退出。下面着重说明当客户端输入“1”的情况，也即进行相关反馈查询。

首先服务器发送提示信息给客户端：“请输入相关文档以供相关反馈查询，用逗号(,)隔开。注：如果格式不符合，则此次查询取消相关反馈:”

得到客户端的响应后，我们使用前文 2.1.4 一节中介绍的 `split` 函数将字符串按照逗号分割成小的字符串数组。对于每个小的字符串，我们使用 2.3.5 一节中介绍的 `isnumber` 函数来判断是否为整数。如果一切都合法，我们可以得到两个向量：`relevant` 和 `irrelevant`，分别表示前面返回的 10 个文档编号中，哪几个是相关的文档、哪几个是不相关的。然后我们就可以用 2.3.4 一

节中的 `feedback` 函数得到一个新的文档编号集合。最后，我们再调用一次 `getResultStr` 函数，将最终的查询结果返回给客户端，结束此次查询。

2.3 myfunc.cpp

2.3.1 获得查询字符串的权重向量的函数 `getWeight`

```
1 std::vector<double> getWeight(std::string q) {
2     std::vector<std::string> w = split(q, separationChar);
3     std::vector<double> res;
4     std::vector<int> cnt(tf.size());
5     for (size_t i = 0; i < w.size(); i++) {
6         std::string now = toLower(w[i]);
7         if (!wordNum.count(now)) continue;
8         int id = wordNum[now];
9         cnt[id]++;
10    }
11    for (size_t i = 0; i < cnt.size(); i++) {
12        if (!cnt[i]) res.push_back(0);
13        else {
14            double w_df = 1 + log10((double)cnt[i]);
15            double idf_t = log10(1.0 * tf[0].size() / df[i]);
16            res.push_back(w_df * idf_t);
17        }
18    }
19    // 归一化
20    normalize(res);
21    return res;
22 }
```

在《向量空间模型》一章中，我们学到，对于一个查询，应该将其当作一个文档处理。具体来说，将其处理成一个和其他文档相同维度的向量后，找出与其余弦归一化夹角最小的几个文档。在这一小节中，我们的工作就是得到这个查询字符串的余弦归一化向量。

具体来说，先用前文2.1.4一节中介绍的 `split` 函数将字符串按照逗号分割成小的字符串数组，并用2.1.4中的 `toLower` 函数转换大写字母。对于在全部文档中没有出现过的词汇，我们将其忽略掉；然后统计其余词汇，得到一个词频向量。利用公式1，我们就可以得到查询字符串的 `tf-idf` 权重向量，然后利用2.3.5一节中的 `normalize` 函数进行归一化即可。

2.3.2 查询处理函数 `query`

```
1 typedef std::pair<int, double> pidb;
2 std::vector<int> query(std::vector<double> qryWeight, size_t retCnt=10) {
3     std::vector<pidb> dist(tf[0].size());
4     std::vector<double> docWeight(qryWeight.size());
5     for (size_t i = 0; i < dist.size(); i++) {
```

```

6      // 计算归一化距离
7      double vecDis = 0;
8      for (size_t j = 0; j < docWeight.size(); j++) {
9          vecDis += Cos_weight[j][i] * qryWeight[j];
10     }
11     dist[i] = std::make_pair(i, vecDis);
12 }
13 sort(dist.begin(), dist.end(), [&](pidb a, pidb b) {
14     return a.second > b.second;
15 });
16 std::vector<int> res;
17 for (size_t i = 0; i < std::min(retCnt, dist.size()); i++) {
18     res.push_back(dist[i].first);
19 }
20 return res;
21 }
22
23 std::vector<int> query(std::string q, size_t retCnt=10) {
24     std::vector<double> qryWeight = getWeight(q);
25     return query(qryWeight, retCnt);
26 }

```

这里有两个功能相近的同名函数，分别对应是否获得了查询字符串的权重向量。对于未获得权重向量的函数，我们只需要调用2.3.1一节中的函数 `getWeight`，情况就变得和获得权重向量的查询相同。下面着重介绍获得了权重向量后如何查询：

我们遍历全部的文档，计算查询字符串的权重向量和文档权重向量之间的余弦归一化夹角。其公式为：

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1} |V| q_i d_i}{\sqrt{\sum_{i=1} |V| q_i^2} \sqrt{\sum_{i=1} |V| d_i^2}} \quad (3)$$

由于我们对权重向量已经做了余弦归一化处理，因此由公式3，我们只需将两个向量对应维度相乘后求和即可。

为了得到与查询向量距离最近的 10 个文档编号，我们使用 `std::pair<int, double> dist` 来记录各文档和查询向量的相似度。其中，`dist.first` 为文档编号，`dist.second` 为相似度。我们按照相似度从大到小排序，也即第二维从大到小排序，选取前 10 个的文档编号作为返回结果，`push_back` 到结果向量中。

2.3.3 获得结果字符串函数 `getResultStr`

```

1 std::string getResultStr(std::vector<int> &answer) {
2     std::string sendstr;
3     for (size_t i = 0; i < answer.size(); i++) {
4         std::set<char> u{'/'};
5         std::vector<std::string> tmp = split(filePaths[answer[i]], u);

```

```

6     sendstr += std::to_string(i + 1) + ". ";
7     sendstr += "/" + tmp[tmp.size() - 3];
8     sendstr += "/" + tmp[tmp.size() - 2];
9     sendstr += "/" + tmp[tmp.size() - 1] + "\n";
10 }
11 return sendstr + "\n";
12 }

```

初始化时在 `filePaths` 中记录的是完整路径。这里，我们想要得到相对路径。其方法是，将完整路径按照 “/” 分隔开，取最后 3 项，再合并起来即可。

2.3.4 相关反馈处理函数 `feedback`

```

1  const double alpha = 1.0;
2  const double beta = 0.75;
3  const double gama = 0.15;
4  std::vector<int> feedback(std::string q,
5                           std::vector<int> relevant,
6                           std::vector<int> irrelevant) {
7      std::vector<double> qryWeight = getWeight(q);
8      std::vector<std::string> w = split(q, separationChar);
9      std::vector<double> res;
10     std::vector<int> q0(tf.size());
11     std::vector<double> qm(tf.size());
12     std::vector<double> muDr(tf.size());
13     std::vector<double> muDnr(tf.size());
14     for (size_t i = 0; i < w.size(); i++) {
15         std::string now = toLower(w[i]);
16         if (!wordNum.count(now)) continue;
17         int id = wordNum[now];
18         q0[id]++;
19     }
20     for (size_t i = 0; i < tf.size(); i++) {
21         for (size_t j = 0; j < relevant.size(); j++) {
22             int docID = relevant[j];
23             muDr[i] += tf[i][docID];
24         }
25         muDr[i] /= 1.0 * relevant.size();
26         for (size_t j = 0; j < irrelevant.size(); j++) {
27             int docID = irrelevant[j];
28             muDnr[i] += tf[i][docID];
29         }
30         muDnr[i] /= 1.0 * irrelevant.size();
31     }
32     for (size_t i = 0; i < tf.size(); i++) {
33         qm[i] = alpha * q0[i] + beta * muDr[i] - gama * muDnr[i];
34         qm[i] = std::max(qm[i], 0.0);

```

```

35     }
36     normalize(qm);
37     return query(qm);
38 }

```

这里我们使用了 Rocchio 1971 算法，即使用公式：

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j \quad (4)$$

其中， \vec{q}_0 表示原始查询向量， \vec{q}_m 表示修改后的查询向量， D_r 和 D_{nr} 表示已知的相关和不相关文档集合。在这里，我们定义超参数： $\alpha = 1.0, \beta = 0.75, \gamma = 0.15$ 。

需要注意的是，最终得到的新查询向量 \vec{q}_m 中，有些维度可能是负数，这时需要把相应的维度置 0。此外，`feedback` 函数的作用是处理相关反馈，因此得到新的查询向量后，我们要对其进行归一化、再利用 2.3.2 一节中的 `query` 函数查询得到相关文档。

2.3.5 其他函数

1. 判断字符串是否为数的函数 `isnumber`：

```

1 bool isnumber(std::string s) {
2     int len = (int)s.length();
3     for (int i = 1; i < len; i++) {
4         if (!isdigit(s[i])) {
5             return false;
6         }
7     }
8     return true;
9 }

```

由于此项目只需判断字符串是否为整数，因此可以忽略负数和小数的判断。直接逐字符判断是否在‘0’到‘9’之间即可。

2. 向量归一化函数 `normalize`：

```

1 void normalize(std::vector<double> &res) {
2     double sum = 0, sqrtsum;
3     for (size_t i = 0; i < res.size(); i++) {
4         sum += res[i] * res[i];
5     }
6     sqrtsum = sqrt(sum);
7     for (size_t i = 0; i < res.size(); i++) {
8         res[i] /= sqrtsum;
9     }
10 }

```

此函数用于将一个向量归一化。其原理如公式 2 所示。

2.4 client.cpp

2.4.1 建立 socket 服务

```
1 if (argc != 3) {
2     printf("argument error.\nusage: ./client <ip address> <port>\n");
3     exit(1);
4 }
5 int sockfd;
6 if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
7     perror("create socket error!");
8 }
9 struct sockaddr_in server_addr;
10 server_addr.sin_family = AF_INET;
11 server_addr.sin_port = htons(atoi(argv[2]));
12 char *server_ip = (char*)malloc(strlen(argv[1]) * sizeof(char));
13 strcpy(server_ip, argv[1]);
14 if (strcmp(server_ip, "localhost") == 0) {
15     strcpy(server_ip, "127.0.0.1");
16 }
17 if ((server_addr.sin_addr.s_addr = inet_addr(server_ip)) == INADDR_NONE) {
18     perror("invalid ip address");
19 }
20 memset(server_addr.sin_zero, 0, sizeof(server_addr.sin_zero));
21 //连接服务器
22 if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
23     perror("connect error");
24     exit(0);
25 }
```

首先判断参数是否正确。客户端需要正确的输入服务器的 IP 地址和端口，才能建立连接。接下来和服务器的代码类似。不同的地方是，服务器需要自己绑定端口，而客户端只需要等待系统分配随机端口即可。另外，如果客户端输入的 IP 地址为“localhost”，也是允许的，因为客户端会将其转化为“127.0.0.1”。

2.4.2 轮流收发消息

```
1 //发送消息
2 recv(sockfd, recv_msg, sizeof(recv_msg), 0);
3 printf("\nServer: %s\n\n", recv_msg);
4 printf("Me : \n");
5 while (std::cin.getline(send_msg, MAXLEN)) {
6     send(sockfd, send_msg, strlen(send_msg), 0);
7     if (!strcmp(send_msg, "bye()") || send_msg[0] == '3') {
8         break;
9     }
10    //接收并显示消息
```

```
11  memset(recv_msg, 0, MAXLEN * sizeof(char)); //接收数组置零
12  if (recv(sockfd, recv_msg, sizeof(recv_msg), 0) == -1) {
13      perror("receive error!");
14  }
15  printf("\nServer: %s\n", recv_msg);
16  printf("Me : \n");
17 }
18 //关闭SOCKET
19 close(sockfd);
```

客户端的操作十分简单。只需要循环的“接受服务器消息——由用户手动输入消息后发送给服务器”即可。这里使用 `std::cin.getline(send_msg, MAXLEN)` 来读取用户的一行输入发送给服务器。

3 实验

1. 客户端输入查询 game

如图6所示，客户端输入查询 **game**（比赛）后，服务器返回了一系列相关文档，其中 7 项是关于 NBA、2 项是关于 football、1 项是关于 nokia 的，直观上来看都和 **game** 有所关联。尤其是 NBA，因为 NBA 球赛的英文是 "basketball game"。

```
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject master ./client localhost 12345
Server:
请输入查询:
Me :
game
Server:
1. /datafile/NBA/NBA18.txt
2. /datafile/nokia/nokia3.txt
3. /datafile/NBA/NBA20.txt
4. /datafile/football/football9.txt
5. /datafile/NBA/NBA4.txt
6. /datafile/football/football14.txt
7. /datafile/NBA/NBA2.txt
8. /datafile/NBA/NBA5.txt
9. /datafile/NBA/NBA13.txt
10. /datafile/NBA/NBA7.txt
输入「1」进行相关反馈查询。
输入「2」进行新的项查询。
输入「3」结束查询:
```

图 6: 客户端输入查询 **game**

2. 客户端进行相关反馈查询

假设用户想要查找关于 NBA 的信息，于是输入“1”进行相关反馈查询。紧接着客户端输入相关的标号“1,3,5,7,8,9,10”。如图7，可以看见，服务器新返回的相关反馈查询结果全部 10 条记录都是关于 NBA 的：

```
Me :
1
Server:
请输入相关文档以供相关反馈查询，用逗号(,)隔开。
注：如果格式不符合，则此次查询取消相关反馈：
Me :
1,3,5,7,8,9,10
Server:
相关反馈查询结果：
1. /datafile/NBA/NBA7.txt
2. /datafile/NBA/NBA13.txt
3. /datafile/NBA/NBA18.txt
4. /datafile/NBA/NBA2.txt
5. /datafile/NBA/NBA19.txt
6. /datafile/NBA/NBA4.txt
7. /datafile/NBA/NBA12.txt
8. /datafile/NBA/NBA3.txt
9. /datafile/NBA/NBA5.txt
10. /datafile/NBA/NBA20.txt
请输入查询:
Me :

```

图 7: 客户端进行相关反馈查询

3. 客户端输入查询 i like to play soccer

如图8所示，客户端输入查询 **i like to play soccer** 后，服务器返回的结果中，前 5 项有 4 项是关于足球的。可见此查询系统也可以查询句子，而不仅仅是单词。客户端输入“2”之后，可以继续查询。

4. 客户端输入 bye() 退出查询

如图9所示，客户端输入 **bye()** 退出查询。而后客户端可以再次连接到服务器，也即服务器不受影响。

```
请输入查询：
Me  :
i like to play soccer

Server:
1. /datafile/football/football19.txt
2. /datafile/NBA/NBA8.txt
3. /datafile/football/football11.txt
4. /datafile/football/football15.txt
5. /datafile/football/football9.txt
6. /datafile/NBA/NBA18.txt
7. /datafile/NBA/NBA1.TXT
8. /datafile/nokia/nokia3.txt
9. /datafile/cambridge/cambridge16.txt
10. /datafile/nokia/nokia25.txt

输入「1」进行相关反馈查询，
输入「2」进行新的词项查询，
输入「3」结束查询：

Me  :
2

Server:
请输入查询：

Me  :
```

图 8: 客户端输入查询 i like to play soccer

```
Server:
请输入查询：

Me  :
bye()
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject $ ./client localhost 12345

Server:
请输入查询：

Me  :
```

图 9: 客户端输入 bye() 退出查询

A 记一件趣事

在原来的 C++ 项目中，我向来都是想都不想便加上一句 `using namespace std;`。这次当然也不例外。但是在写到 socket 编程时，如图10，我发现客户端死活也连接不上服务器：

```
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject master > ./client localhost 12345
connect error: Connection refused
(base) GoatWu@mac ~/Desktop/学习文件/大三下学习文件/现代信息检索技术/项目课题/myproject master >
```

图 10: 客户端无法连接到服务器

经过一段时间排查，我终于发现，在 C++ 标准库中，`functional.h` 也包含了 `bind()` 函数！其解决方法是，删除掉 `using namespace std;`，并在使用 C++ 标准库的地方都加上 `std::`。这样虽然麻烦，但是能避免很多奇奇怪怪的报错。

B 完整源代码

B.1 initialize.cpp

```
1 #include "myfunc.cpp"
2
3 const int MAXFILENAME = 10000;
4 char buffer[MAXFILENAME];
5 extern std::vector<std::string> filePaths; // 编号 -> 文档
6 extern std::vector<std::string> words; // 编号 -> 单词
7 extern std::map<std::string, int> fileNum; // 文档 -> 编号
8 extern std::map<std::string, int> wordNum; // 单词 -> 编号
9 extern std::set<char> separationChar; // 分隔符
10 extern std::vector<int> df; // 出现词项的文档数目
11 extern std::vector< std::vector<int> > tf; // 词频矩阵
12 extern std::vector< std::vector<double> > tf_idf; // TF-IDF 权重
13 extern std::vector< std::vector<double> > Cos_weight; // 余弦归一化之后的权重
14
15 // 获取全部文件的绝对路径
16 int initFilePaths() {
17     getcwd(buffer, sizeof(buffer));
18     std::string path;
19     path.assign(buffer);
20     path += "/datafile/";
21
22     getFiles(path, filePaths);
23
24     for (size_t i = 0; i < filePaths.size(); i++) {
25         fileNum[filePaths[i]] = i;
26         // STD::COUT << FILEPATHS[I] << "\n";
27     }
28     return (int) filePaths.size();
29 }
```

```

29 }
30
31 void initSeperationChar() {
32     std::set<char> tmp{
33         ' ', 13, 10, '\t', ',', '.', ';',
34         ':', '?', '!', '\'', '\\"', '\\';
35     };
36     separationChar = tmp;
37 }
38
39 int initWordsNum() {
40     int wordCnt = 0;
41     for (size_t i = 0; i < filePaths.size(); i++) {
42         std::set<std::string> used; // 单次在当前文档中是否出现过
43         std::ifstream t(filePaths[i]);
44         std::stringstream buffer;
45         buffer << t.rdbuf();
46         std::string contents(buffer.str());
47         std::vector<std::string> w = split(contents, separationChar);
48         for (size_t j = 0; j < w.size(); j++) {
49             std::string now = toLower(w[j]);
50             if (!wordNum.count(now)) {
51                 wordNum[now] = wordCnt++;
52                 df.push_back(1);
53                 used.insert(now);
54                 words.push_back(now);
55             }
56             else if (!used.count(now)) {
57                 df[wordNum[now]]++;
58                 used.insert(now);
59             }
60         }
61         t.close();
62     }
63     return wordCnt;
64 }
65
66 void initTf() {
67     tf.resize(words.size());
68     for (size_t i = 0; i < tf.size(); i++) {
69         tf[i].resize(filePaths.size());
70     }
71     std::ifstream inFile;
72     for (size_t i = 0; i < filePaths.size(); i++) {
73         std::ifstream t(filePaths[i]);
74         std::stringstream buffer;
75         buffer << t.rdbuf();

```

```

76     std::string contents(buffer.str());
77     std::vector<std::string> w = split(contents, separationChar);
78     for (size_t j = 0; j < w.size(); j++) {
79         std::string now = toLower(w[j]);
80         tf[wordNum[now]][i]++;
81     }
82     t.close();
83 }
84 }
85
86 void initWeight() {
87     int n = filePaths.size();
88     int m = words.size();
89     tf_idf.resize(m);
90     Cos_weight.resize(m);
91     for (size_t i = 0; i < m; i++) {
92         tf_idf[i].resize(n);
93         Cos_weight[i].resize(n);
94     }
95     for (size_t i = 0; i < m; i++) {
96         for (size_t j = 0; j < n; j++) {
97             if (tf[i][j] == 0) tf_idf[i][j] = 0;
98             else {
99                 double w_df = 1 + log10((double)tf[i][j]);
100                 double idf_t = log10(1.0 * n / df[i]);
101                 tf_idf[i][j] = w_df * idf_t;
102             }
103         }
104     }
105     for (size_t j = 0; j < n; j++) {
106         double sum = 0, sqrtsum;
107         for (size_t i = 0; i < m; i++) {
108             sum += tf_idf[i][j] * tf_idf[i][j];
109         }
110         sqrtsum = sqrt(sum);
111         for (size_t i = 0; i < m; i++) {
112             Cos_weight[i][j] = tf_idf[i][j] / sqrtsum;
113         }
114     }
115 }

```

B.2 server.cpp

```

1 #include <bits/stdc++.h>
2 #include "initialize.cpp"
3 #include <unistd.h>

```

```

4 #include <signal.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8 #include <arpa/inet.h>
9 #include <sys/stat.h>
10 #include <fcntl.h>
11 const int MAXLEN = 1e6;
12 char recv_msg[MAXLEN];
13
14 int main(int argc, char *argv[]) {
15     if (argc > 2 || argc < 2 || atoi(argv[1]) == 0) {
16         printf("usage: ./server <port>\n");
17         exit(1);
18     }
19
20     initSeperationChar();
21     int fileCnt = initFilePaths();
22     int wordCnt = initWordsNum();
23     initTf();
24     initWeight();
25     std::cout << "Total file count: " << fileCnt << "\n";
26     std::cout << "different words: " << wordCnt << "\n\n";
27
28     /***** SOCKET *****/
29     int sockfd;
30     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
31         perror("create socket error");
32     }
33     struct sockaddr_in server_addr;
34     struct sockaddr_in client_addr;
35     server_addr.sin_family = AF_INET;
36     server_addr.sin_port = htons(atoi(argv[1]));
37     server_addr.sin_addr.s_addr = htonl(INADDR_ANY); //INADDR_ANY表示本机所有IP地址
38     memset(&server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充
39     //绑定SOCKET与地址
40     bind(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)); //监听SOCKET
41     listen(sockfd, 0);
42     int cfd;
43     int num = 0;
44     pid_t pid;
45     char client_ip[128];
46     printf("等待询问.....\n");
47     while (233) {
48         socklen_t len = sizeof(client_addr);
49         if ((cfd = accept(sockfd, (struct sockaddr *)&client_addr, &len)) == -1) {
50             perror("accept error");

```



```

51         exit(0);
52     }
53     // 清除僵尸进程
54     signal(SIGCHLD, SIG_IGN);
55     // 获取客户端IP地址、端口
56     const char *ip = inet_ntop(AF_INET, &client_addr.sin_addr.s_addr, client_ip, sizeof(
        client_ip));
57     const int port = ntohs(client_addr.sin_port);
58     printf("Accept %s:%d\n", ip, port);
59     fflush(stdout);
60     pid = fork();
61     if (pid == 0) {
62         close(sockfd);
63         memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零
64         ssize_t tag;
65         send(cfd, "请输入查询: ", strlen("请输入查询: "), 0);
66         while ((tag = recv(cfd, recv_msg, sizeof(recv_msg), 0)) != 0) {
67             std::string msg(recv_msg);
68             if (tag == -1) perror("receive error");
69             if (msg == "bye()") {
70                 break;
71             }
72             std::string sendstr("搜索结果:\n");
73             std::string queryMsg = msg;
74             std::vector<int> answer = query(msg);
75             sendstr = getResultStr(answer);
76             sendstr += "输入「1」进行相关反馈查询, \n";
77             sendstr += "输入「2」进行新的词项查询, \n";
78             sendstr += "输入「3」结束查询: \n";
79             send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
80             memset(recv_msg, 0, sizeof(recv_msg));
81             recv(cfd, recv_msg, sizeof(recv_msg), 0);
82             msg = recv_msg;
83             while (msg[0] != '1' && msg[0] != '2' && msg[0] != '3') {
84                 // STD::COUT << "WTF? " << MSG << FFLUSH;
85                 sendstr = "输入「1」进行相关反馈查询, \n";
86                 sendstr += "输入「2」进行新的词项查询, \n";
87                 sendstr += "输入「3」结束查询: \n";
88                 send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
89                 memset(recv_msg, 0, sizeof(recv_msg));
90                 recv(cfd, recv_msg, sizeof(recv_msg), 0);
91                 msg = recv_msg;
92             }
93             if (msg[0] == '1') {
94                 sendstr = "请输入相关文档以供相关反馈查询, 用逗号(,)隔开. \n";
95                 sendstr += "注: 如果格式不符合, 则此次查询取消相关反馈: \n";
96                 send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);

```

```

97         memset(recv_msg, 0, sizeof(recv_msg));
98         recv(cfd, recv_msg, sizeof(recv_msg), 0);
99         msg = recv_msg;
100         std::set<char> u{' ',' '};
101         std::vector<std::string> tmp = split(msg, u);
102         std::vector<int> isRel(10);
103         std::vector<int> relevant;
104         std::vector<int> irrelevant;
105         int okFlag = 1;
106         for (size_t i = 0; i < tmp.size(); i++) {
107             if (!isnumber(tmp[i])) {
108                 okFlag = 0;
109                 break;
110             }
111             int relevantID = atoi(tmp[i].c_str()) - 1;
112             if (relevantID < 0 || relevantID > 9) {
113                 okFlag = 0;
114                 break;
115             }
116             isRel[relevantID] = 1;
117         }
118         for (size_t i = 0; i < isRel.size(); i++) {
119             if (isRel[i]) relevant.push_back(answer[i]);
120             else irrelevant.push_back(answer[i]);
121         }
122         if (!okFlag) {
123             sendstr = "非法的反馈序列! 本次查询结束! \n";
124             sendstr += "请输入查询: ";
125             send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
126         }
127         else {
128             std::vector<int> answer = feedback(queryMsg, relevant, irrelevant);
129             sendstr = getResultStr(answer);
130             sendstr = "相关反馈查询结果: \n" + sendstr;
131             sendstr += "请输入查询: ";
132             send(cfd, sendstr.c_str(), strlen(sendstr.c_str()), 0);
133         }
134     }
135     else if (msg[0] == '2') {
136         send(cfd, "请输入查询: ", strlen("请输入查询: "), 0);
137     }
138     else {
139         break;
140     }
141     memset(recv_msg, 0, sizeof(recv_msg));
142 }
143 }

```

```

144         else if (pid > 0) {
145             close(cfd);
146         }
147         else {
148             perror("folk error");
149         }
150     }
151
152     return 0;
153 }

```

B.3 myfunc.cpp

```

1  #include <bits/stdc++.h>
2  #include <unistd.h>
3  #include <dirent.h>
4  #include <sys/types.h>
5  #include <sys/stat.h>
6  typedef std::pair<int, double> pidb;
7  const double alpha = 1.0;
8  const double beta = 0.75;
9  const double gama = 0.15;
10 std::vector<std::string> filePaths;           // 编号 -> 文档
11 std::vector<std::string> words;              // 编号 -> 单词
12 std::map<std::string, int> fileNum;          // 文档 -> 编号
13 std::map<std::string, int> wordNum;          // 单词 -> 编号
14 std::set<char> separationChar;               // 分隔符
15 std::vector<int> df;                         // 出现词项的文档数目
16 std::vector< std::vector<int> > tf;          // 词频矩阵
17 std::vector< std::vector<double> > tf_idf;   // TF-IDF 权重
18 std::vector< std::vector<double> > Cos_weight; // 余弦归一化之后的权重
19
20 bool isnumber(std::string s) {
21     int len = (int)s.length();
22     for (int i = 1; i < len; i++) {
23         if (!isdigit(s[i])) {
24             return false;
25         }
26     }
27     return true;
28 }
29
30 void getFiles(std::string path, std::vector<std::string>& files) {
31     std::string path0 = path;
32     DIR* pDir;
33     struct dirent* ptr;

```

```

34     struct stat s;
35     lstat(path.c_str(), &s);
36
37     if (!S_ISDIR(s.st_mode) || !(pDir = opendir(path.c_str()))) {
38         // STD::COUT << "NOT A VALID DIRECTORY: " << PATH << "\n";
39         return;
40     }
41
42     std::string subFile;
43     while ((ptr = readdir(pDir)) != 0) {
44         subFile = ptr->d_name;
45         if (subFile == "." || subFile == "..") continue;
46         struct stat tmp;
47         subFile = path0 + subFile + "/";
48         lstat(subFile.c_str(), &tmp);
49         if (S_ISDIR(tmp.st_mode)) {
50             getFiles(subFile, files);
51         }
52         else {
53             subFile.pop_back();
54             files.push_back(subFile);
55         }
56     }
57     closedir(pDir);
58 }
59
60
61 std::string toLower(std::string s) {
62     std::string res = s;
63     for (size_t i = 0; i < s.size(); i++) {
64         if (s[i] >= 'A' && s[i] <= 'Z') {
65             res[i] += 'a' - 'A';
66         }
67     }
68     return res;
69 }
70
71
72 std::vector<std::string> split(const std::string &str, const std::set<char> &pattern) {
73     // CONST CHAR* CONVERT TO CHAR*
74     char * strc = new char[strlen(str.c_str())+1];
75     strcpy(strc, str.c_str());
76     size_t len = strlen(strc);
77     std::vector<std::string> resultVec;
78
79     char *tmpStr = strc;
80     for (size_t i = 0; i < len; i++) {

```

```

81     if (pattern.count(strc[i])) {
82         strc[i] = 0;
83         if (tmpStr[0] != 0) {
84             resultVec.push_back(std::string(tmpStr));
85         }
86         tmpStr = strc + i + 1;
87     }
88 }
89 if (tmpStr[0] != 0) {
90     resultVec.push_back(std::string(tmpStr));
91 }
92 delete[] strc;
93 return resultVec;
94 }
95
96 void normalize(std::vector<double> &res) {
97     double sum = 0, sqrtsum;
98     for (size_t i = 0; i < res.size(); i++) {
99         sum += res[i] * res[i];
100     }
101     sqrtsum = sqrt(sum);
102     for (size_t i = 0; i < res.size(); i++) {
103         res[i] /= sqrtsum;
104     }
105 }
106
107 std::vector<double> getWeight(std::string q) {
108     std::vector<std::string> w = split(q, separationChar);
109     std::vector<double> res;
110     std::vector<int> cnt(tf.size());
111     for (size_t i = 0; i < w.size(); i++) {
112         std::string now = toLower(w[i]);
113         if (!wordNum.count(now)) continue;
114         int id = wordNum[now];
115         cnt[id]++;
116     }
117     for (size_t i = 0; i < cnt.size(); i++) {
118         if (!cnt[i]) res.push_back(0);
119         else {
120             double w_df = 1 + log10((double)cnt[i]);
121             double idf_t = log10(1.0 * tf[0].size() / df[i]);
122             res.push_back(w_df * idf_t);
123         }
124     }
125     // 归一化
126     normalize(res);
127     return res;

```

```

128 }
129
130 std::vector<int> query(std::vector<double> qryWeight, size_t retCnt=10) {
131     std::vector<pidb> dist(tf[0].size());
132     std::vector<double> docWeight(qryWeight.size());
133     for (size_t i = 0; i < dist.size(); i++) {
134         // 计算归一化距离
135         double vecDis = 0;
136         for (size_t j = 0; j < docWeight.size(); j++) {
137             vecDis += Cos_weight[j][i] * qryWeight[j];
138         }
139         dist[i] = std::make_pair(i, vecDis);
140     }
141     sort(dist.begin(), dist.end(), [&](pidb a, pidb b) {
142         return a.second > b.second;
143     });
144     std::vector<int> res;
145     for (size_t i = 0; i < std::min(retCnt, dist.size()); i++) {
146         res.push_back(dist[i].first);
147     }
148     return res;
149 }
150 // 返回前 RETCNT 个结果
151 std::vector<int> query(std::string q, size_t retCnt=10) {
152     std::vector<double> qryWeight = getWeight(q);
153     return query(qryWeight, retCnt);
154 }
155
156 std::string getResultStr(std::vector<int> &answer) {
157     std::string sendstr;
158     for (size_t i = 0; i < answer.size(); i++) {
159         std::set<char> u{'/'};
160         std::vector<std::string> tmp = split(filePaths[answer[i]], u);
161         sendstr += std::to_string(i + 1) + ". ";
162         sendstr += "/" + tmp[tmp.size() - 3];
163         sendstr += "/" + tmp[tmp.size() - 2];
164         sendstr += "/" + tmp[tmp.size() - 1] + "\n";
165     }
166     return sendstr + "\n";
167 }
168
169 std::vector<int> feedback(std::string q,
170                         std::vector<int> relevant,
171                         std::vector<int> irrelevant) {
172     std::vector<double> qryWeight = getWeight(q);
173     std::vector<std::string> w = split(q, separationChar);
174     std::vector<double> res;

```

```

175     std::vector<int> q0(tf.size());
176     std::vector<double> qm(tf.size());
177     std::vector<double> muDr(tf.size());
178     std::vector<double> muDnr(tf.size());
179     for (size_t i = 0; i < w.size(); i++) {
180         std::string now = toLower(w[i]);
181         if (!wordNum.count(now)) continue;
182         int id = wordNum[now];
183         q0[id]++;
184     }
185     for (size_t i = 0; i < tf.size(); i++) {
186         for (size_t j = 0; j < relevant.size(); j++) {
187             int docID = relevant[j];
188             muDr[i] += tf[i][docID];
189         }
190         muDr[i] /= 1.0 * relevant.size();
191         for (size_t j = 0; j < irrelevant.size(); j++) {
192             int docID = irrelevant[j];
193             muDnr[i] += tf[i][docID];
194         }
195         muDnr[i] /= 1.0 * irrelevant.size();
196     }
197     for (size_t i = 0; i < tf.size(); i++) {
198         qm[i] = alpha * q0[i] + beta * muDr[i] - gama * muDnr[i];
199         qm[i] = std::max(qm[i], 0.0);
200     }
201     normalize(qm);
202     return query(qm);
203 }

```

B.4 client.cpp

```

1  #include <bits/stdc++.h>
2  #include <sys/types.h>
3  #include <sys/socket.h>
4  #include <netinet/in.h>
5  #include <arpa/inet.h>
6  #include <unistd.h>
7  const int MAXLEN = 1e6;
8  char send_msg[MAXLEN];
9  char recv_msg[MAXLEN];
10
11 int main(int argc, const char * argv[]) {
12     if (argc != 3) {
13         printf("argument error.\nusage: ./client <ip address> <port>\n");
14         exit(1);

```

```

15     }
16     int sockfd;
17     if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
18         perror("create socket error!");
19     }
20     struct sockaddr_in server_addr;
21     server_addr.sin_family = AF_INET;
22     server_addr.sin_port = htons(atoi(argv[2]));
23     char *server_ip = (char*)malloc(strlen(argv[1]) * sizeof(char));
24     strcpy(server_ip, argv[1]);
25     if (strcmp(server_ip, "localhost") == 0) {
26         strcpy(server_ip, "127.0.0.1");
27     }
28     if ((server_addr.sin_addr.s_addr = inet_addr(server_ip)) == INADDR_NONE) {
29         perror("invalid ip address");
30     }
31     memset(server_addr.sin_zero, 0, sizeof(server_addr.sin_zero));
32     //连接服务器
33     if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1) {
34         perror("connect error");
35         exit(0);
36     }
37     //发送消息
38     recv(sockfd, recv_msg, sizeof(recv_msg), 0);
39     printf("\nServer: %s\n\n", recv_msg);
40     printf("Me : \n");
41     while (std::cin.getline(send_msg, MAXLEN)) {
42         send(sockfd, send_msg, strlen(send_msg), 0);
43         if (!strcmp(send_msg, "bye()") || send_msg[0] == '3') {
44             break;
45         }
46         //接收并显示消息
47         memset(recv_msg, 0, MAXLEN * sizeof(char)); //接收数组置零
48         if (recv(sockfd, recv_msg, sizeof(recv_msg), 0) == -1) {
49             perror("receive error!");
50         }
51         printf("\nServer: %s\n\n", recv_msg);
52         printf("Me : \n");
53     }
54     //关闭socket
55     close(sockfd);
56     return 0;
57 }

```
