**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**ĐHQG TP HỒ CHÍ MINH**

Logo

Description automatically generated

**BÁO CÁO ĐỒ ÁN CUỐI KÌ**

**SEARCH ENGINE**

**Sinh viên thực hiện:** Nguyễn Anh Khoa – 20120118

Dũ Quốc Huy – 20120101

**Giáo viên hướng dẫn: Võ Hoài Việt**

**SEARCH ENGINE**

[I) Giai đoạn TrainData 2](#_Toc77234362)

[1) Đọc từng tệp tin từ thư mục newtrain 2](#_Toc77234363)

[2) Đọc dữ liệu từ tệp tin txt 4](#_Toc77234364)

[3) Chuẩn hóa các dữ liệu đọc được: 4](#_Toc77234365)

[3.1 Chuyển đổi chữ hoa thành chữ thường: 4](#_Toc77234366)

[3.2 Xóa các kí tự của câu: 4](#_Toc77234367)

[3.3 Xóa những từ là StopsWord: 5](#_Toc77234368)

[3.4 Xóa các dấu của kí tự (sắc, huyền, hỏi , ngã, nặng): 6](#_Toc77234369)

[3.5 Xóa các từ được xem là không quan trọng: 7](#_Toc77234370)

[3.6 Xóa khoảng cách thừa: 9](#_Toc77234371)

[4) Xuất dữ liệu đã chuẩn hóa 10](#_Toc77234372)

[II) Giai đoạn xử lý yêu cầu người dùng 10](#_Toc77234373)

[1) Xử lý tiền tìm kiếm 11](#_Toc77234374)

[2) Bộ công cụ tìm kiếm 11](#_Toc77234375)

[2.1) Nhập dữ liệu tìm kiếm 11](#_Toc77234376)

[2.2) Chiến thuật tìm kiếm 12](#_Toc77234377)

[2.3) Sắp xếp giữ liệu và truy xuất: 13](#_Toc77234378)

[3) Hàm thêm file 15](#_Toc77234379)

[4) Hàm xóa file 16](#_Toc77234380)

[III) Tài liệu tham khảo 17](#_Toc77234381)

# Giai đoạn TrainData

*Đây là giai đoạn đọc dữ liệu từ đường dẫn cố định, chuyển đổi về định dạng UTF8 để xử lí và xuất ra dữ liệu cần thiết cho giai đoạn tìm kiếm*

## Đọc từng tệp tin từ thư mục newtrain

Sử dụng thư viện **dirent.h** (được thêm vào chương trình) để đọc từng tệp tin. Với mỗi tệp đọc được lại tiếp tục đọc các tệp dữ liệu để tạo đường dẫn cố định đến tệp dữ liệu đó, truyền đường dẫn vào hàm DocFile để tiến hành đọc dữ liệu.

DIR\* pDIR;

struct dirent\* entry;

string temp = "D:/new train";

if (pDIR = opendir((const char\*)temp.c\_str())) {

while (entry = readdir(pDIR)) {

if (strcmp(entry->d\_name, ".") != 0 && strcmp(entry->d\_name, "..") != 0)

{

cout << entry->d\_name << "\n";

string temp2 = temp + "/";

for (int i = 0; i < sizeof(entry->d\_name); i++)

{

if (entry->d\_name[i] != '\0')

temp2 = temp2 + entry->d\_name[i];

else break;

}

DIR\* pDIR2;

struct dirent\* entry2;

pDIR2 = opendir((const char\*)temp2.c\_str());

while (entry2 = readdir(pDIR2))

if (strcmp(entry2->d\_name, ".") != 0 && strcmp(entry2->d\_name, "..") != 0)

{

string link = temp2 + "/";

for (int i = 0; i < sizeof(entry2->d\_name); i++)

{

if (entry2->d\_name[i] != '\0')

link = link + entry2->d\_name[i];

else break;

}

cout << link << endl;

string st = "";

DocFile(link, st);

fo << link << endl << st << endl;

}

closedir(pDIR2);

}

}

## Đọc dữ liệu từ tệp tin txt

Vì dữ liệu đầu vào có định dạng UTF-16LE, đọc bằng file nhị phân và chuyển dữ liệu sang định dạng UTF-8 để dễ dàng cho quá trình xử lí tiếp theo.

UTF-16LE

UTF-8

ifstream fi((const char\*)link.c\_str(), ios::binary);

fi.seekg(0, ios::end);

size\_t size = (size\_t)fi.tellg();

fi.seekg(2, ios::beg);

size -= 2;

u16string u16((size / 2) + 1, '\0');

fi.read((char\*)&u16[0], size);

st=wstring\_convert<codecvt\_utf8\_utf16<char16\_t>, char16\_t>{}.to\_bytes(u16);

fi.close();

## Chuẩn hóa các dữ liệu đọc được:

*Thực hiện chuẩn hóa xâu dữ liệu theo từng bước sau:*

### 3.1 Chuyển đổi chữ hoa thành chữ thường:

Duyệt từ đầu đến cuối xâu chuyển những kí tự có dạng chữ hoa

(‘A’ -> ‘Z’ ) và chuyển thành chữ thường (‘a’ -> ‘z’ )

for (int i = 0; i < st.length(); i++)

if (st[i] <= 'Z' && st[i] >= 'A')

st[i] = st[i] + 32;

### 3.2 Xóa các kí tự của câu:

Duyệt từ đầu đến cuối xâu, nếu kí tự đó là các dấu câu (phẩy, chấm, hỏi chấm, …) hoặc các kí tự ẩn (\n , \r ) thì xóa kí tự đó.

int i = 0;

while (i < st.length())

{

if ( st[i]== \*các kí tự của câu )

st.erase(i, 1);

else i++;

}

### 3.3 Xóa những từ là StopsWord:

Bước 1: Tạo List stopword từ file stopword.txt với mỗi dòng đọc được là một Node dung để lưu từ stopword của dòng đó.

ifstream in("stopword.txt");

List L;

string temp;

while (!in.eof())

{

getline(in, temp);

Push(L, temp);

}

in.close();

return L;

Bước 2: Duyệt hết List stopword vừa tạo, sử dụng hàm find để tìm vị trí của từ stopword đó trong xâu dữ liệu và xóa từ đó đi (cho đến khi không tìm được nữa).

for (Node\* p = L.head; p; p = p->next)

{

int pos = -1;

int x = st.find(p->word, pos + 1);

while (x >= 0)

{

if (Tu(st, x, p->word.length()))

st.erase(x, p->word.length());

x = st.find(p->word, x + 1);

}

}

### 3.4 Xóa các dấu của kí tự (sắc, huyền, hỏi , ngã, nặng):

Bước 1: Tạo List các kí tự có dấu từ file daukitu.txt với mỗi dòng đọc được là một Node dung để lưu kí tự của dòng đó.

string temp;

List L;

ifstream in("daukitu.txt");

while (!in.eof())

{

getline(in, temp);

Push(L, temp);

}

in.close();

return L;

Bước 2: Duyệt hết List vừa tạo, sử dụng hàm find để tìm vị trí xuất hiện của kí tự đó trong xâu dữ liệu và thay thế nó bằng kí tự tương ứng (kí từ chữ thường, không dấu, thay cho đến khi không thể tìm được). Vị trí của biến dem phụ thuộc vào sự bố trí các dòng trong file daukitu.txt.

int dem = 0;

for (Node\* p = L.head; p->next != NULL; p = p->next)

{

int pos = -1;

int x = st.find(p->word, pos + 1);

while (x >= 0)

{

int len = p->word.size();

if (dem < 34)

st = st.replace(x, len, "a");

else if (dem == 34 || dem == 35)

st = st.replace(x, len, "d");

else ……. ;

x = st.find(p->word, x + 1);

}

dem++;

}

### 3.5 Xóa các từ được xem là không quan trọng:

Bước 1: Cắt xâu dữ liệu thành các từ riêng biệt, mỗi từ sẽ được lưu bằng một Node. Để đủ điều kiện từ một từ thì trước từ kí tự bắt đầu của từ khác dấu cách, trước từ đó dấu khoảng cách ( trường hợp từ đầu câu thì vị trí tìm được phải là 0). Nếu từ đó đã có trong List thì tăng biến count, ngược lại thì tạo Node mới để lưu từ.

int i = 0;

st = st + " ";

while (i < st.length())

{

int len = 1;

if (st[i] != ' ' && (i == 0 || st[i - 1] == ' '))

{

string temp = "";

int j = i;

while (st[j] != ' ' && st[j] != '\0')

{

temp = temp + st[j];

j++;

}

len = len + temp.length();

bool kt = false;

for (Node\* p = L.head; p; p = p->next)

{

if (p->word == temp)

{

kt = true;

p->count++;

break;

}

}

if (kt == false)

Push(L, temp);

}

i += len;

}

Bước 2: Tạo ra List từ được xem là không quan trọng. Duyệt List các từ ở bước một, nếu số lần xuất hiện (count) của từ đó so với tổng số từ của xâu nhỏ hơn 0.01 thì từ đó được xem là không quan trọng. Lưu từ không quan trong vào List kqt.

Giải thích vì sao lại chọn 0.01: xâu dữ liệu khi đọc vào có độ dài trung bình khoảng 500-600 từ, sau những bước chuẩn hóa trên, dữ liệu được rút ngắn đi nhiều chỉ còn khoảng 250-300 từ. Ta chọn giữ lại những từ có xác xuất từ 0.01 trở lên để đảm bảo giữ lại những từ có sự xuất hiện vừa đủ (dưới 0.01 thì từ đó không còn phổ biến so với nội dung của dữ liệu) cho qua trình tìm kiếm.

List temp;

int s = TongTu(L);

for (Node\* p = L.head; p; p = p->next)

{

if ((p->count \* 1.0 / s) < 0.01)

Push(temp, p->word);

}

return temp;

Bước 3: Duyệt List từ không quan trọng vừa tạo ở bước 2, dung hàm find để tìm và xóa các từ đó khỏi xâu kí tự.

for (Node\* p = L.head; p; p = p->next)

{

int pos = -1;

int x = st.find(p->word, pos + 1);

while (x >= 0)

{

if (Tu(st, x, p->word.length()))

st.erase(x, p->word.length());

x = st.find(p->word, x + 1);

}

}

### 3.6 Xóa khoảng cách thừa:

Xóa các kí tự khoảng cách ở đầu xâu. Nếu có hơn 1 khoảng cách liền nhau ở giữa xâu thì sẽ xóa cho đến khi chỉ còn 1 dấu.

int i = 0;

while (st[0] == ' ')

{

st.erase(0, 1);

}

while (i < st.length())

{

if (st[i] == ' ' && st[i + 1] == ' ')

st.erase(i, 1);

else i++;

}

## Xuất dữ liệu đã chuẩn hóa

Dữ liệu được ghi ra file metadata.txt theo cú pháp: một dòng là đường dẫn tuyệt đối đến dữ liệu đó, và dòng tiếp theo chính là xâu dữ liệu đã chuẩn hóa được lưu ở định dạng UTF-8.

fo << link << endl << st << endl;

Lưu ý: xóa các List đã tạo trong quá trình để làm trống bộ nhớ.

while (L.head != NULL)

{

Node\* temp = L.head;

L.head = L.head->next;

delete temp;

}

# Giai đoạn xử lý yêu cầu người dùng

*Sau khi train data, ta sẽ thu được có 1 file cấu trúc là dạng nén của nhiều file, mỗi file sẽ chiếm 2 dòng, dòng thứ nhất là link lưu file đó trong máy tính, và dòng thứ hai là gồm những từ có tần suất xuất hiện cao hơn trọng số đã xác định >=0.01*

File kết quả được lưu trong 1 list, mà mỗi node của list gồm 4 thành phần:

* Link: Lưu đường dẫn tới file
* Data: Lưu nội dung của metadata
* Count: đây là biến sẽ định tính giá trị của mỗi file, dựa vào đây chúng ta sẽ sắp xếp
* Next: trỏ tới node Result tiếp theo

struct Result

{

string link = "";

string data = "";

int count = 5000;

Result\* next = NULL;

};

## Xử lý tiền tìm kiếm

**Mục tiêu: Tạo một list để chứa thông tin xử lý**

Ở đây chúng ta sử dụng hàm ListMetadata, chúng ta chỉ cần tạo 2 biến string, 1 biến sẽ nhận link và 1 biến sẽ nhận dòng dữ liệu, cứ như vậy cho đến hết file. Sau đó ta sẽ dùng hàm push để ghi cả hai vào trong 1 list có sẵn

Hàm push cũng khá tương đồng với các hàm push bên train data:

void Push(ListResult& LR, string link, string data)

{

Result\* newResult = new Result;

newResult->link = link;

newResult->data = data;

newResult->next = LR.head;

LR.head = newResult;

}

## Bộ công cụ tìm kiếm

### 2.1) Nhập dữ liệu tìm kiếm

**Mục tiêu: Chia nhỏ dữ liệu tìm kiếm để xử lý từng chữ**

Người dùng sẽ thực hiện nhập liệu không dấu từ bàn phím, đó là 1 tiếng, 1 từ hoặc có thể là 1 câu. Sau khi nhận được dữ liệu, câu đó sẽ được phân chia thành nhiều từ, rồi đặt các từ đó vào trong 1 danh sách liên kết.

Ta tận dụng các hàm đã có sẵn để xử lý và chia thành các từ nhỏ hơn:   
B1: Hàm ChuThuong để chuyển toàn bộ kí tự in hoa trong dữ liệu nhập thành kí tự thường

B2: Hàm demtu sẽ đưa từng tiếng đã được chia nhỏ vào trong list word

B3: Bắt đầu tìm kiếm

### 2.2) Chiến thuật tìm kiếm

Để tìm kiếm cụm từ hay 1 từ xuất hiện trong đoạn văn, ta lần lượt đếm số lần xuất hiện của **cụm từ** đó trong đoạn metadata đã xử lý. Lưu ý rằng cách tính này chỉ có tính tương đối nhưng nó mang lại hiệu quả chính xác rất cao. Thuật toán đưa ra là: Đếm số lần xuất hiện của từng tiếng, trong đoạn metadata, số lần xuất hiện của cụm từ đó = giá trị nhỏ nhất của số lần xuất hiện từng tiếng trong cụm từ tìm kiếm. Ví dụ cụ thể như sau:

* **Trời xanh mây trắng** gồm có 4 tiếng:

**+** File thứ nhất gồm có 7 từ Trời, 5 từ xanh, 3 từ mây và 3 từ trắng => min{7,5,3,3}=1

**+** File thứ hai gồm có 8 từ trời, 3 từ xanh, 2 từ mây, 2 từ trắng => min{8,3,2,2}=1

Như vậy, độ ưu tiên của file thứ nhất sẽ cao hơn file thứ hai. Sau đây là code thực tế hóa chiến thuật tìm kiếm:

ListResult ListMetadata2(ListResult& LR, List Word)

{

for (Result\* p = LR.head; p; p = p->next)

{

p->count = 5000;

for (Node\* p2 = Word.head; p2; p2 = p2->next)

{

int pos = -1;

int x = p->data.find(p2->word, pos + 1);

int dem = 0;

while (x >= 0 && Tu(p->data, x, p2->word.length()))

{

dem++;

x = p->data.find(p2->word, x + 1);

}

if (dem < p->count)

p->count = dem;

}

}

return LR;

}

Khởi tạo giá trị count cho mỗi file kết quả là 5000(hay có thể gọi là giá trị infinity). Hàm sẽ bắt đầu đếm số lần xuất hiện của mỗi từ trong dữ liệu tìm kiếm đã qua xử lý để tìm số lần xuất hiện. Nếu như trong những từ tìm kiếm có 1 từ có số lần xuất hiện bằng 0 thì dĩ nhiên count cũng sẽ nhận giá trị là 0:

### 2.3) Sắp xếp giữ liệu và truy xuất:

Thực hiện việc sắp xếp các file bằng cách thực hiện merge sort (phù hợp nhất với danh sách liên kết)

Megre sort gồm có 3 hàm:   
- Hàm trộn – Merge: Đầu vào là 2 list đã được sắp xếp từ lớn đến bé, sau đó trộn cả 2 thành 1 list cũng có thứ tự từ lớn đến bé. Do đặc tính của list ta phải kết hợp với hàm push, và phép so sánh, cuối cùng ta nối phần dài hơn của 1 trong 2 chuỗi vào chuỗi kết quả.

ListResult merge(ListResult a, ListResult b) {

if (a.head == NULL) { return b; }

if (b.head == NULL) { return a; }

ListResult c = init();

Result\* p = a.head;

Result\* q = b.head;

Result\* tmp = new Result;

while (p != NULL && q != NULL) {

if (p->count > q->count) { tmp = addtail(c, p->link,p->data,p->count); p = p->next; }

else { tmp = addtail(c, q->link, q->data, q->count); q = q->next; }

}

if (p != NULL) { tmp->next = p; }

if (q != NULL) { tmp->next = q; }

return c;

}

- Hàm spilitlist: chia list làm thành hai phần

void spilitlist(ListResult &a, ListResult& b, ListResult c) {

Result\* fast = c.head;

a.head = c.head;

Result\* slow = c.head;

while (fast->next != NULL && fast->next->next != NULL) {

fast = fast->next->next;

slow = slow->next;

}

b.head = slow->next;

slow->next = NULL;

}

- Hàm Merge\_sort: sắp xếp bằng cách chia nhỏ đến mức mỗi list con đã được sắp xếp sẵn, sau đó bắt đầu trộn 2 chuỗi đã được sắp xếp, cứ như vậy cho đến khi chiều dài list con bằng chiều dài list ban đầu

void merge\_sort(ListResult& a) {

if (a.head == NULL || a.head->next == NULL) { return; }

else {

ListResult a1 = init();

ListResult a2 = init();

spilitlist(a1, a2, a);

merge\_sort(a1);

merge\_sort(a2);

a = merge(a1, a2);

}

}

Sau khi có được dữ liệu các file đã được sắp xếp, các file được xuất ra màn hình 1 lần 10 link file, và chương trình sẽ hỏi người dùng có muốn hiển thị thêm hay không. Sau đó người dùng có thể mở chi tiết từng file

## Hàm thêm file

**Mục tiêu: Là thêm file vào trong bộ dữ liệu đã có sẵn, sau đó tìm kiếm trên bộ dữ liệu mới**

Người dùng được yêu cầu nhâp chính xác đường dẫn, file sẽ được đọc và chuẩn hóa như ở phần train data, sau đó được thêm vào trong listResult, và thực hiện tìm kiếm bình thường. Khi người dùng nhập sai chương trình sẽ báo lỗi

void AddFile(ListResult& LR)

{

\_setmode(\_fileno(stdout), \_O\_WTEXT);

string link;

wcout << L"Nhập đường dẫn của file muốn thêm(hãy đảm bảo bạn nhập đúng, ví dụ D:/caigido.txt): ";

cin.ignore();

getline(cin, link);

if (!checkfile(link)) { //err

wcout << L"Đường dẫn đến file bạn cần thêm bị lỗi, hãy xem lại!!!\n";

}

else

{

string st = "";

DocFile(link, st);

Push(LR, link, st);

wcout << L"File đã được thêm thành công vào bộ dữ liệu\n";

}

}

## Hàm xóa file

**Mục tiêu: Xóa node đại diện file khỏi listResult, khi tìm kiếm thì file sẽ không hiển thị nữa**

Chương trình yêu cầu người dùng nhập đường dẫn vào, sau đó chương trình sẽ tìm đến file đó trong bộ dữ liệu và xóa nó đi

void DelFile(ListResult& LR)

{

\_setmode(\_fileno(stdout), \_O\_WTEXT);

string link;

wcout << L"Nhập đường dẫn đến file muốn xóa(hãy đảm bảo bạn nhập đúng, ví dụ D:/huydeptrai.txt): ";

cin.ignore();

getline(cin, link);

bool kt = false;

if (LR.head->link == link) {

kt = true;

Result\* temp = LR.head;

LR.head = LR.head->next;

delete temp;

wcout << L"File đã được xóa thành công khỏi bộ dữ liệu\n";

}

else {

for (Result\* p = LR.head; p->next; p = p->next)

{

if (p->next->link == link)

{

kt = true;

Result\* temp = p->next;

p->next = temp->next;

delete temp;

wcout << L"File đã được xóa thành công khỏi bộ dữ liệu\n";

break;

}

}

}

if (kt == false)

wcout << L"Đường dẫn đến file bạn cần xóa không tồn tại trong dữ liệu!!!\n";

}

# Tài liệu tham khảo

Cấu trúc file: <https://chungnguyen.xyz/posts/co-gi-khac-nhau-giua-utf-8-va-utf-8-without-bom?fbclid=IwAR1b0BuY8CsrTLuj_NHulHzHMFMsA7-3JF5pkb0j8RACou3haIqUpyTiqtc>

Đọc file UTF-16: <https://stackoverflow.com/questions/50696864/reading-utf-16-file-in-c?fbclid=IwAR3SNAkA05ynE45T3oTEu5sSNKjQpzB6K2aer4i3Ww3Zm_V3oybUUFBxtaU>

Đọc dữ liệu từ cây thư mục: https://linustechtips.com/topic/690245-c-making-a-list-of-paths-from-a-directory-tree/?fbclid=IwAR3Fackliricm53K4AjyKPX9wzVdSLiYOR9\_WfEY8f\_F-wD5ADeqqxjVvmk

TF-IDF: <https://nguyenvanhieu.vn/tf-idf-la-gi/>