

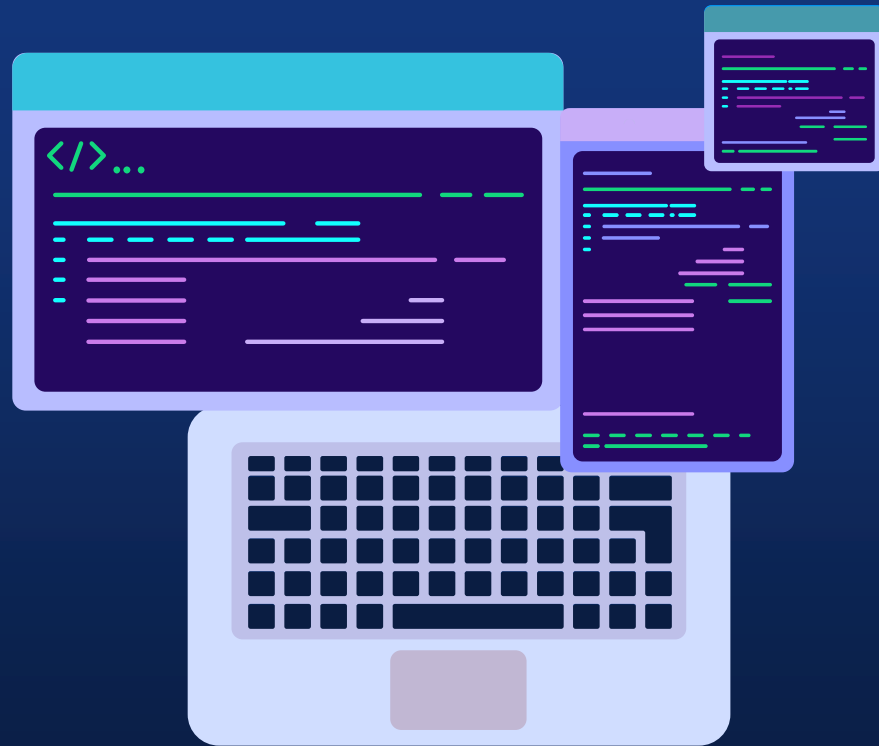


fit@hcmus

GIẢI THUẬT SEARCH ENGINE

20120412

20120463



NỘI DUNG

01

Xử lý dữ liệu

02

Giải thuật xây dựng cấu trúc dữ liệu lưu trữ

03

Truy vấn

04

Cập nhật văn bản

05

GUI

06

Thực nghiệm





Nhiệm Vụ

\\ Rút trích đơn giản
nội dung văn bản
Tiếng Việt

\\ Tìm kiếm các văn
bản liên quan đến từ
khóa

\\ Xếp hạng theo
điểm số



01

Xử lý dữ liệu





Ghi thông tin văn bản vào Index

```
bool add_file_to_index(const string &path_file)
{
    ofstream f_index("../Config\\index.txt", ios::app);
    if (!f_index)
    {
        cout << "LOG: failed to open index.txt\n";
        return false;
    }
    f_index << path_file << endl;
    f_index.close();
    return true;
}
```

● ***Index.txt chứa thông tin, đường dẫn đến văn bản***

```
..\File\VanBanTV_train\Am nhac\AN_TN_ (878).txt
..\File\VanBanTV_train\Am nhac\AN_TN_ (879).txt
..\File\VanBanTV_train\Am nhac\AN_TN_ (880).txt
..\File\VanBanTV_train\Am nhac\AN_TN_ (881).txt
..\File\VanBanTV_train\Am nhac\AN_TN_ (882).txt
..\File\VanBanTV_train\Am nhac\AN_TN_ (883).txt
....
```



Đọc văn bản

```
wifstream fin(path_file, ios::binary);
fin.imbue(locale(fin.getloc(),
    new codecvt_utf16<wchar_t, 0x10ffff, little_endian>));
for (wchar_t c; fin.get(c);)
{
    wcout << c;
}
```

- Mở theo định dạng nhị nhân, đọc từng cặp byte
- Văn bản có định dạng UTF-16
- BOM : Little Endian

Loại Stopwords ra khỏi văn bản

```
wstring remove_by_list_of_words(wstring &raw_str, wstring listwords[], int size_listwords)
{
    //brute
    wstring res = L""; //string after removing stopwords
    for (int i = 0; i < raw_str.size(); i++)
    {
        //get letter of a word
        wstring word = L"";
        while(raw_str[i] != ' ' && i < raw_str.size())
            word += raw_str[i++];

        //have a word, check this word in stopwords
        bool found = false;
        for (int j = 0; j < size_listwords; j++)
        {
            if (cmp(word, listwords[j]))
            {
                found = true;
                break;
            }
        }

        if(found)
        {
            //remove this word from raw_str
            ...
        }
    }
    return res;
}
```

Tách từng từ của đoạn văn bản

Kiểm tra từ có xuất hiện trong Stopwords hay không?

Xóa từ ra khỏi đoạn văn bản

Làm sạch dữ liệu



PROBLEM

Sử dụng toàn bộ kí tự



SOLUTION

1. Xây dựng hàm `filter()` loại bỏ các kí tự không cần thiết
2. Xây dựng hàm `utf16_to_utf8()` Chuẩn hóa các kí tự còn lại trở nên không dấu => Độ phủ tìm kiếm tăng

Các hàm `filter()`, `utf16_to_utf8()`

```
bool filter(char char_value)
{
    if (char_value == '?' || char_value == '.' ||
        char_value == '{' || char_value == '}' ||
        char_value == '|')
        return false;
    return true;
}
```

`filter()`

```
char utf16_to_utf8(int char_value)
{
    if (0 <= char_value && char_value <= 126)
    {
        if ('A' <= char_value && char_value <= 'Z')
            return (char)(char_value + 'a' - 'A');
        else
            return (char)char_value;
    }
    switch (char_value)
    {
        ....
    }
}
```

`utf16_to_utf8()`



Tokenization và rút Keywords

<div>Độ quan trọng/Số từ rút trích</div> <div>Truy vấn</div>	0%-100% 14365 từ	0%-1% 12781 từ	0%-7% 13890 từ	1.5%-7% 6871 từ
Query_1	91.3%	89.1%	90%	95.4%
Query_2	81.4%	81.2%	81.4%	72.5%
Query_3	93.9%	91%	92.3%	89.4%
...
Query_100	88.8%	86.9%	87.8%	79.7%



Chọn độ quan trọng 1.5%-7%: Tỷ lệ tìm kiếm cao tiết kiệm bộ nhớ



Trung bình tỷ lệ xuất hiện của các từ trong Query_i
(Bảng dưới đây có được sau khi chương trình hoàn thành và nhóm đã thử nghiệm trên nhiều độ quan trọng khác nhau)



Tổ chức metadata




1. Thêm các token sau khi rút trích vào */Dictionary*.
2. Mỗi token tương ứng với một file (được đặt "*_term.dat*").
3. Mỗi file chứa thông tin: *term*, *hash của term*, *tên văn bản chứa term* và *chỉ số TF tương ứng*.

Dictionary	_term.dat
_term1.dat	<term> <term_hash>
_term2.dat	<Doc1>
_term3.dat	<term's TF in Doc 1>
_term4.dat	<Doc2>
.....	



Tại sao đặt tên file là "_term.dat"?

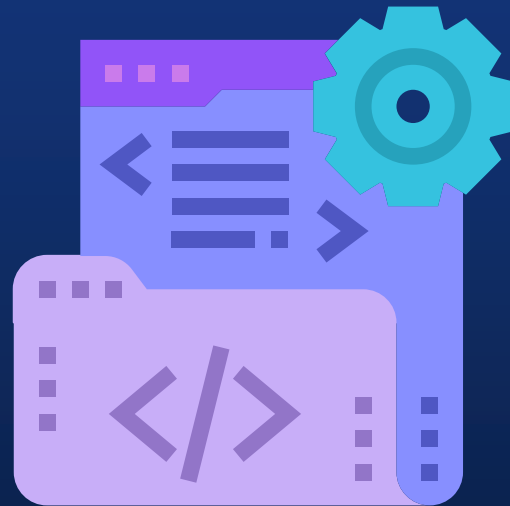
-> Để tránh xung đột cách đặt tên file của Windows nếu vô tình





02

Giải thuật xây
dựng cấu trúc dữ
liệu lưu trữ



Xây dựng cấu trúc dữ liệu dạng *INVERTED INDEX*

*Cấu trúc sau khi load tất
cả dữ liệu từ /Dictionary
lên RAM*

Term	DocID
phần	VanBan1,..
mềm	VanBan1, VanBan2. ...
tìm	VanBan3,..
....	

Minh họa đơn giản



HASHTABLE

(INVERTED INDEX)

Triển khai

Dữ liệu cho mảng `HashTable[]`

- Xâu
- Mã hóa xâu
- Danh sách các văn bản chứa xâu

```
struct Info_doc //in4 for document
{
    string doc_name;
    double score; //TF of doc
};

struct Info_ht //in4 for hash_table
{
    string term;
    int hash_term;
    List<Info_doc> list_document;
};
```



Lý do tổ chức dữ liệu như trên?



HASHTABLE - LISTDOCUMENT



Chưa xác định
được số lượng văn
bản chứa term?



Bộ nhớ động, thao tác thêm phần
tử linh hoạt.

⇒ *LINKED LIST*

```
List<Info_doc> list_document;
```

HASHTABLE - HASH

Nếu có một
`term_query` cho
sẵn, làm cách nào để
truy xuất nhanh và
hiệu quả?



Duyệt toàn bộ `HashTable[]` ->
Kiểm tra từng term
`(HashTable[i].term == query_term)`
ĐPT: $O(N)$ -> Có thể cải tiến

HASHTABLE - HASH (IMPROVE PERFORMANCE)



Việc so sánh lần lượt các kí tự trong chuỗi , làm giảm hiệu suất chương trình. Thay vào đó việc so sánh giữa các chuỗi kí tự có thể thực hiện trong $O(1)$ thay vì $O(N)$ -> Mã hóa các chuỗi thành 1 con số. `int hash_term;`

ĐPT: $O(1)$

HASHTABLE - HASH (IMPROVE PERFORMANCE)



Giờ đây `HashTable[]` có thể được xem như là một mảng số nguyên => Dễ thuận tiện cho việc tìm kiếm tuyến tính =>

Sắp xếp mảng theo trường `int hash_term;`

Nhóm đề xuất giải thuật sắp xếp `QuickSort`

ĐPT: $O(N \log N)$

Mảng đã được sắp xếp, giờ đây có thể tìm kiếm tuyến tính

Nhóm đề xuất Tìm kiếm nhị phân (`Binary Search`)

ĐPT: $O(\log N)$

HASHTABLE – HASH – QUICK SORT - PIVOT

1. Quick sort là một thuật toán chia để trị(*Divide and Conquer algorithm*)
 2. Nó chọn một phần tử trong mảng làm điểm đánh dấu(*pivot*)
 3. Thuật toán sẽ thực hiện chia mảng thành các mảng con dựa vào pivot đã chọn
- => Làm sao để chọn pivot?

```
template <class T>
void quickSort(T arr[], int low, int high, bool (*cmp)(T, T))
{
    if (low < high)
    {
        int pi = partition(arr, low, high, cmp);
        // pi is the id that divides array to two sub-array: left and right
        quickSort(arr, low, pi - 1, cmp);
        quickSort(arr, pi + 1, high, cmp);
    }
}
```

HASHTABLE – HASH – QUICK SORT - PARTITION

Mấu chốt chính của thuật toán *quick sort* là việc phân đoạn dãy số (*partition()*)

partition() :

Cho một mảng và một phần tử x là pivot. Đặt x vào đúng vị trí của mảng đã sắp xếp.

Di chuyển tất cả các phần tử của mảng mà nhỏ hơn x sang bên trái vị trí của x , và di chuyển tất cả các phần tử của mảng mà lớn hơn x sang bên phải vị trí của x .

```
template <class T>
int partition(T arr[], int low, int high, bool (*cmp)(T, T))
{
    T pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {
        if (cmp(arr[j], pivot))
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return (i + 1);
}
```

HASHTABLE – HASH – BINARY SEARCH

Đoạn code minh họa, khi có *query_term* cần tìm và trả về danh sách các văn bản mà chứa *query_term*, trong *HashTable* []

```
List<Info_doc> *Inverted_index::find_doc_by_term(const string &query_term)
{
    //find term in hash_table
    int l = 0, r = size - 1;
    List<Info_doc> *res = nullptr;
    //get hash from term
    int64_t hash_term = hash_fr_term(query_term);
    while (l <= r) //until interval has length which is one
    {
        int mid = (l + r) / 2;
        if (hash_table[mid].hash_term <= hash_term)
        {
            if (hash_table[mid].hash_term == hash_term) //find this term
            {
                res = &hash_table[mid].list_document;
            }
            //hash_table[mid].hash_term <= hash_term, so we do not need to care the interval: [l...mid]
            //increase left
            l = mid + 1;
        }
        else
        {
            //hash_table[mid].hash_term > hash_term, so we do not need to care the interval: [mid...r]
            //decrease right
            r = mid - 1;
        }
    }
    //return address of linked list that has term equals query term
    return res;
}
```



03


Truy vấn





Làm thế nào so sánh độ liên quan giữa truy
vấn với văn bản bất kỳ ?





TF-IDF

*(Term frequency-
Inverse document
frequency)*

Dữ liệu cần

- TF

$TF = (\text{số lần xuất hiện của 1 từ}) / (\text{tổng số từ trong văn bản})$

- IDF

$IDF = 1 + \ln(\text{Tổng số văn bản trong data set} / \text{Số văn bản chứa từ})$

- Similarity

TF-IDF – DOCUMENT



TF: Term Frequency của từng query[i] đã có sẵn trong metadata, việc còn lại là tổng hợp

Dictionary	_term.dat
_term1.dat	<term> <term_hash>
_term2.dat	<Doc1>
_term3.dat	<term's TF in Doc 1>
_term4.dat	<Doc2>
.....	

*IDF: Qua hàm `gen_TFIDFvector`
Tính IDF và nhân vào từng từ
=> **Vector Document hoàn chỉnh***

```
for(int i = 0; i < query_size; i++)  
    vector_doc[i] = get_TF(query[i]);  
  
//the rest is multipling IDF in vector  
gen_TFIDFvector(vector_doc, query_size, totaldocs, numbdoc_haveterm);
```

TF-IDF – QUERY



*TF: Xem các từ có tần suất ngang nhau,
bằng $1.0/(\text{số từ trong query})$*

*IDF: Chỉ số IDF sử dụng của Vector
Document trên
Qua hàm `gen_TFIDFvector()`
=> **Vector Query hoàn chỉnh***

```
for(int i = 0; i < query_size; i++)  
    vector_query[i] = 1.0 / query_size;  
  
//the rest is multipling IDF in vector  
gen_TFIDFvector(vector_query, query_size, totaldocs, numbertdoc_haveterm);
```

TF-IDF – SIMILARITY



Giả sử query có n từ = ["từ 1", "từ 2", ...].

Similarity giữa query và document được tính như sau:

```
Query = vector_query
Document = vector_doc
//Norm
||Query|| = square root(Query[TF*IDF("từ 1")]^2 + Query[TF*IDF("từ 2")]^2 + ...)
||Document|| = square root(Document 1[TF*IDF("từ 1")]^2 + Document 1[TF*IDF("từ 2")]^2 + ...)

//Dot product
Dot Product(Query, Document) = Query[TF*IDF("từ 1")] * Document [TF*IDF("từ 1")] +
| | | | | | | | | | Query[TF*IDF("từ 2")] * Document [TF*IDF("từ 2")] + ...

//Similarity
Cosine Similarity(Query, Document) = Dot Product(Query, Document) / ||Query|| * ||Document||
```

04

Cập nhật văn bản



THÊM VĂN BẢN



Tách keyword từ văn bản, tính TF và ghi thêm vào metadata (thực hiện như các bước trên)

THÊM/XÓA NHIỀU VĂN BẢN



Dùng thư viện "*dirent.h*" để liệt kê tất cả các file/folder trong một folder. Người dùng có thể chọn thêm vào 1 file văn bản, 1 folder chứa các file văn bản hoặc 1 folder chứa các folder chứa các file văn bản.

XÓA VĂN BẢN - Xóa tất cả thông tin ra khỏi Dictionary, Index



Build lại metadata ->
Mất thời gian

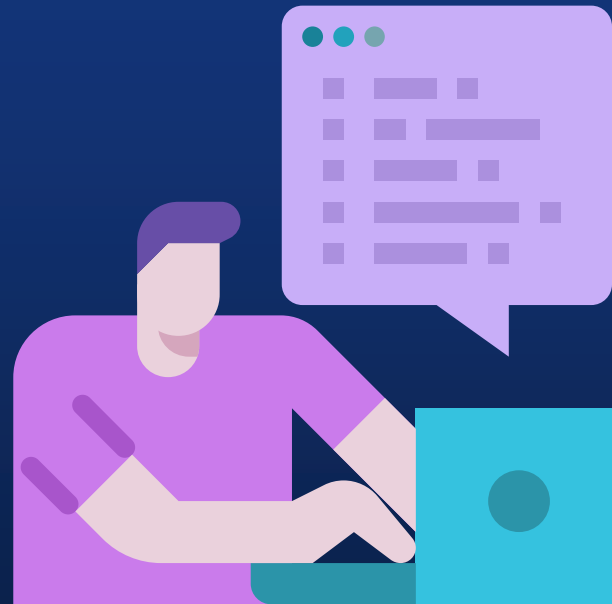



Tìm tất cả những file trong metadata có liên quan đến văn bản (tách keyword từ văn bản, mỗi keyword ứng với 1 file liên quan trong metadata). Lọc qua các file liên quan, tìm và xóa thông tin về văn bản.




05

GUI





Làm sao để tạo diện người dùng đơn giản, hiệu quả ?





PYTHON TKINTER



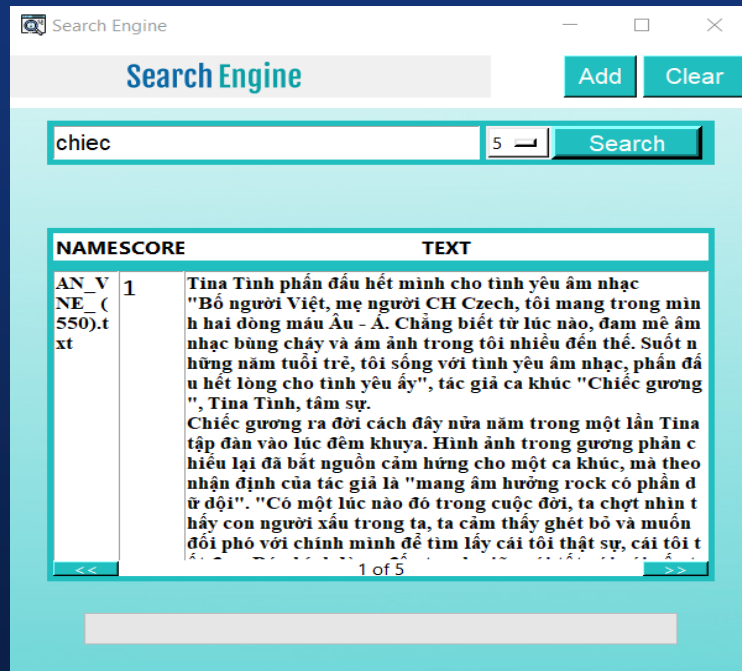
KẾT QUẢ

```
from tkinter import *
from tkinter import ttk
from tkinter import filedialog
import time
import os
import sys
import subprocess

root = Tk()

root.title("Search Engine")
root.iconbitmap('image//icon.ico')

root.geometry("600x600")
# background image
bg_image = PhotoImage(file='image//background.png')
bg_label = Label(root, image=bg_image)
bg_label.place(relwidth=1, relheight=1)
# progress bar
...
```





06

Thực nghiệm



THỜI GIAN XỬ LÝ DỮ LIỆU BAN ĐẦU



	Đọc toàn bộ dữ liệu	Làm sạch + tokenization	Rút trích (6836 từ)
Tập dữ liệu (VanBanTV_Train)	3'05	0'41	4'39



TRUY VẤN



	Tải dữ liệu lên RAM (ban đầu)	Thời gian trung bình tìm kiếm của 100 truy vấn ngẫu nhiên (liên tục)
Instant Search	0s	2'03s
Full Search	5's	0'54s

CẬP NHẬT DỮ LIỆU – THÊM DỮ LIỆU



	Tập tin đơn (Ví dụ: <i>File/VanBanTV_Train/Am Nhac/AN_TN.txt</i>)	Tập hợp nhiều tập tin đơn (Ví dụ: <i>File/VanBanTV _Train/Am Nhac</i>)	Tập hợp nhiều tập hợp nhiều tập tin đơn (Ví dụ: <i>File/VanBanTV_Train</i>)
Thời gian trung bình thêm dữ liệu (Thử nghiệm trên tập dữ liệu VanBanTV_Train)	0.01s	18s	7'85s

CẬP NHẬT DỮ LIỆU – XÓA DỮ LIỆU



	Tập tin đơn (Ví dụ: <i>File/VanBanTV_Train/AmNhac/AN_TN.txt</i>)	Tập hợp nhiều tập tin đơn (Ví dụ: <i>File/VanBanTV_Train/AmNhac</i>)	Tập hợp nhiều tập hợp nhiều tập tin đơn (Ví dụ: <i>File/VanBanTV_Train</i>)
Thời gian trung bình xóa dữ liệu (Thử nghiệm trên tập dữ liệu VanBanTV_Train)	3s	2'9s	15'8s

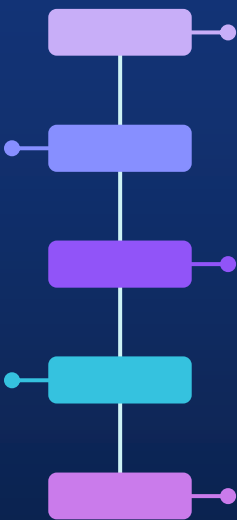
TÓM TẮT

RÚT TRÍCH METADATA

Tokenization thành các token(term), tạo file “_term.dat” để lưu trữ thông tin

TRUY VẤN

Tạo vector tương ứng, so sánh Similarity với các văn bản->Lọc ra N văn bản có Similarity cao nhất



ĐỌC DỮ LIỆU

Đọc -> Ghi thông tin văn bản vào Index -> Tách Keywords -> Lọc, chuẩn hóa

LOAD METADATA

Xây dựng HashTable(Inverted Index)

CẬP NHẬT (NẾU CẦN)

Thêm, xóa

THANKS



QUANG BÌNH – LÊ DUY