TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG ĐIỆN – ĐIỆN TỬ

BÁO CÁO

BÀI TẬP LỚN SỐ 5

Đề tài: Xây dựng từ điển từ viết tắt cho người dùng

Giảng viên hướng dẫn: TS Đỗ Thị Ngọc Diệp

Nhóm 4

Sinh viên 1: **Phạm Gia Minh 20203741**
Sinh viên 2: **Đặng Hữu Công Hiếu 20203417**
Sinh viên 3: **Nguyễn Đình Duy 20202357**

Hà Nội, ngày 14 tháng 7 năm 2022

Bảng danh sách nhóm:

STT	Họ và tên	Mã số sinh viên	Email	Công việc	Mức độ hoàn thành
1	Phạm Gia Minh	20203741	minh.pg203741@sis.hust.edu.vn	Nội dung báo cáo	100%
				Tìm lỗi phần mềm	100%
				Tạo dữ liệu thử nghiệm	100%
2	Đặng Hữu Công Hiếu	20203417	hieu.dhc203417@sis.hust.edu.vn	Lập nhóm	100%
				Chọn chủ đề	100%
				Đưa ra ý tưởng và thiết kế phần mềm	100%
				Nhận xét và chỉnh sửa, bổ sung, hoàn thiện báo cáo	100%
				Thiết kế cấu trúc dữ liệu, giải thuật và các chức năng của phần mềm	100%
3	Nguyễn Đình Duy	20202357	duy.nd202357@sis.hust.edu.vn	Lập nhóm	100%
				Chọn chủ để	100%
				Lên lịch trình	100%
				Chỉnh sửa, bổ sung và hoàn thiện báo cáo	100%
				Triển khai, chỉnh sửa, tìm và vá lỗi phần mềm	100%

Mục lục

Bảng danh sách nhóm: 2

1. Giới thiệu bài toán: 5

Từ điển viết tắt: 5

Các chức năng: 5

2. Mô tả giải thuật sử dụng: 6

3. Cấu trúc dữ liệu : 7

Danh sách liên kết đôi: 7

Thông tin của từ điển: 7

Cấu trúc dữ liệu của từ điển: 7

Cấu trúc lưu trữ: 8

Định nghĩa cấu trúc struct : 8

Dữ liệu mẫu hiển thị: 8

4. Thiết kế giải thuật : 9

4.1 Đọc file và nhập vào dữ liệu: 9

4.2 Thuật toán thêm vào: 11

a) Thiết kế sơ bộ: 11

b) Thiết kế chi tiết: 11

4.3 Thuật toán sửa: 14

a) Thiết kế sơ bộ : 14

b) Thiết kế chi tiết: 15

4.4 Thuật toán xóa: 18

a) Thiết kế sơ bộ: 18

b) Thiết kế chi tiết: 18

4.5 Thuật toán sắp xếp: 20

4.5.1: Thuật toán InsertionSort: 20

4.5.2: Thuật toán Quick Sort: 21

4.6 . Thuật toán tìm kiếm: 22

a) Thiết kế sơ bộ: 22

b) Thiết kế chi tiết: 22

5. Phân tích tính đúng đắn của giải thuật. 25

5.1. Phân tích tính đúng đắn của GT : 25

5.2. Xác định độ phức tạp của GT đề xuất : 25

6. Mô tả triển khai trên ngôn ngữ lập trình cụ thể: 26
7. Tự đánh giá kết quả, các điểm hạn chế: 28
Lời cảm ơn 29

1. Giới thiệu bài toán:

Từ điển viết tắt:

Viết tắt là một dạng rút gọn cách viết của một từ hoặc cụm từ. Thông thường, nó bao gồm một hoặc nhiều chữ cái lấy từ chính từ ngữ được viết tắt. Ví dụ, chính chữ *viết tắt* có thể được viết tắt thành "vt".

Viết tắt thường được sử dụng khi câu từ khi viết đầy đủ bị cho là quá dài, hoặc chỗ trống cho việc viết đầy đủ (trên giấy, bảng hiệu) bị thiếu.

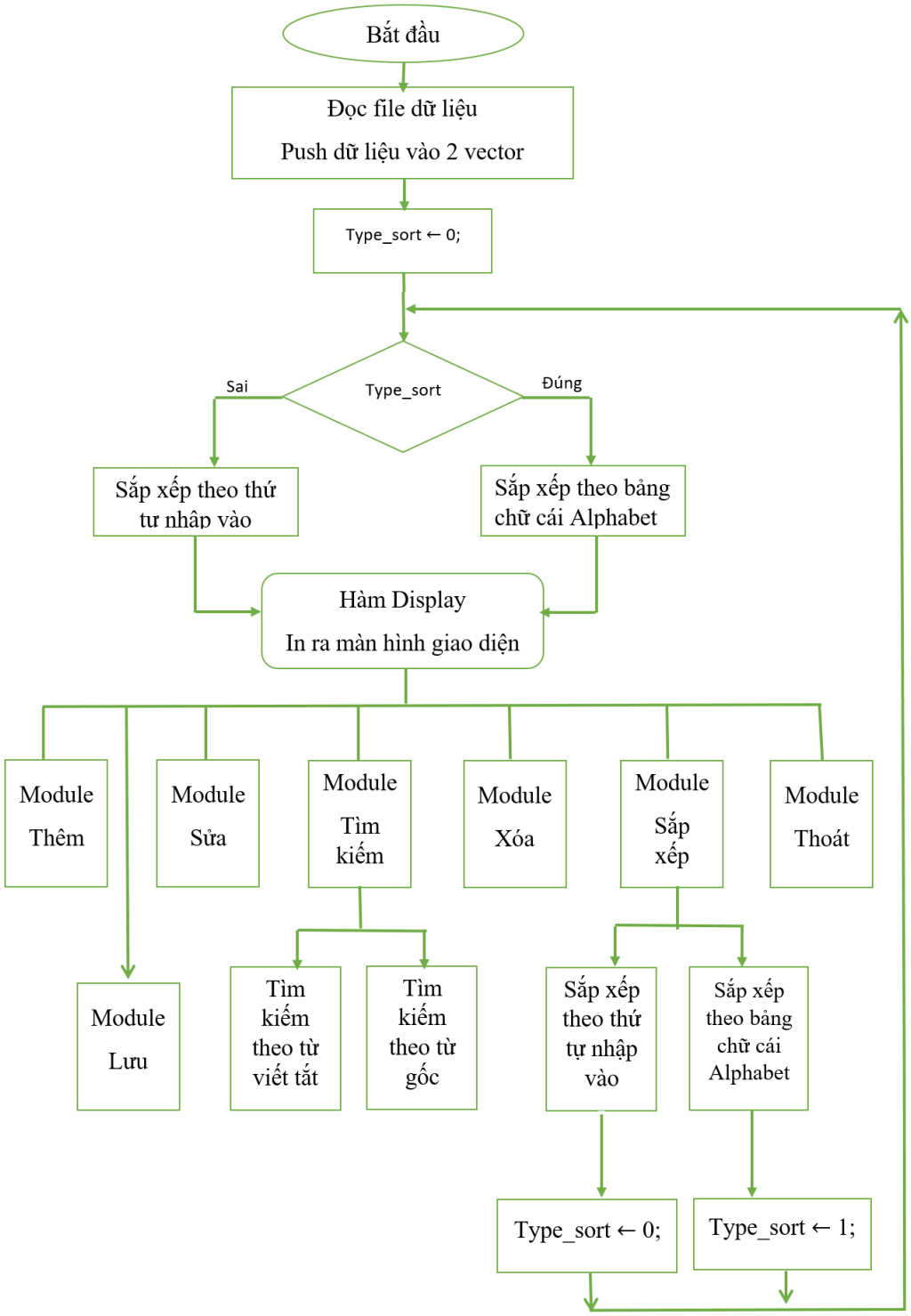
Từ điển viết tắt cho phép xây dựng một từ điển các từ ngữ viết tắt để tra cứu.

Xây dựng một từ điển từ viết tắt cho gồm số thứ tự nhập vào (stt); từ viết tắt (tvt); từ gốc (tg).

Các chức năng:

- Cho phép thêm, sửa, xóa các từ viết tắt cùng dạng viết đầy đủ của nó.
- Cho phép tìm kiếm một từ viết tắt, hiển thị từ viết đầy đủ của nó.
- Cho phép tìm kiếm từ theo dạng viết đầy đủ, hiển thị từ viết tắt của nó.
- Sắp xếp và hiển thị theo thứ tự nhập vào hoặc theo thứ tự alphabet.

2. Mô tả giải thuật sử dụng:



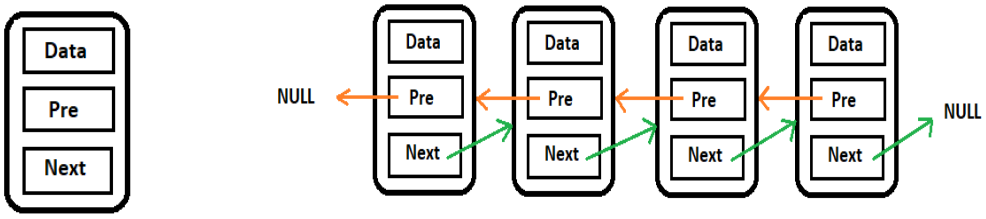
3. Cấu trúc dữ liệu :

Danh sách liên kết đôi:

a) Định nghĩa:

Là một tập hợp các **Node** động được phân bố động, được sắp xếp sao cho mỗi Node chứa:

- Một dữ liệu(data) hoặc nhiều dữ liệu (data có thể là int, string,…)
- Một con trỏ (**Next)** để trỏ tới phần tử kế tiếp của danh sách liên kết đó, nếu con trỏ trỏ tới **NULL** thì đấy là phần tử cuối cùng của danh sách.
- Một con trỏ (**Pre**) để trỏ tới phần tử trước của danh sách liên kết đó, nếu con trỏ trỏ tới **NULL** thì đó là phần tử đầu tiên của danh sách.



Thông tin của từ điển:

- stt : Số thứ tự nhập vào.
- tvt : Từ viết tắt, chiều dài khoảng từ 0 đến 15 ký tự.
- tg : Từ gốc, chiều dài khoảng từ 0 đến 50 ký tự.

Cấu trúc dữ liệu của từ điển:

- Cấu trúc tuyến tính :
  - Vector
- Cấu trúc phi tuyến:
  - Các module như thêm, sửa, xóa, tìm kiếm, sắp xếp.
- Gồm 2 struct :
  - Một struct để lưu danh sách theo thứ tự thêm vào.
  - Một struct để lưu danh sách theo thứ tự alphabet.
- stt: Số nguyên không âm (stt>=0)
- tvt: Chuỗi tối đa 20 ký tự.
- tg : Chuỗi tối đa 50 ký tự.

- Các thao tác trên cấu trúc dữ liệu:
  - Duyệt phần tử
  - Thêm
  - Sắp xếp
  - Chèn mới
  - Tìm kiếm

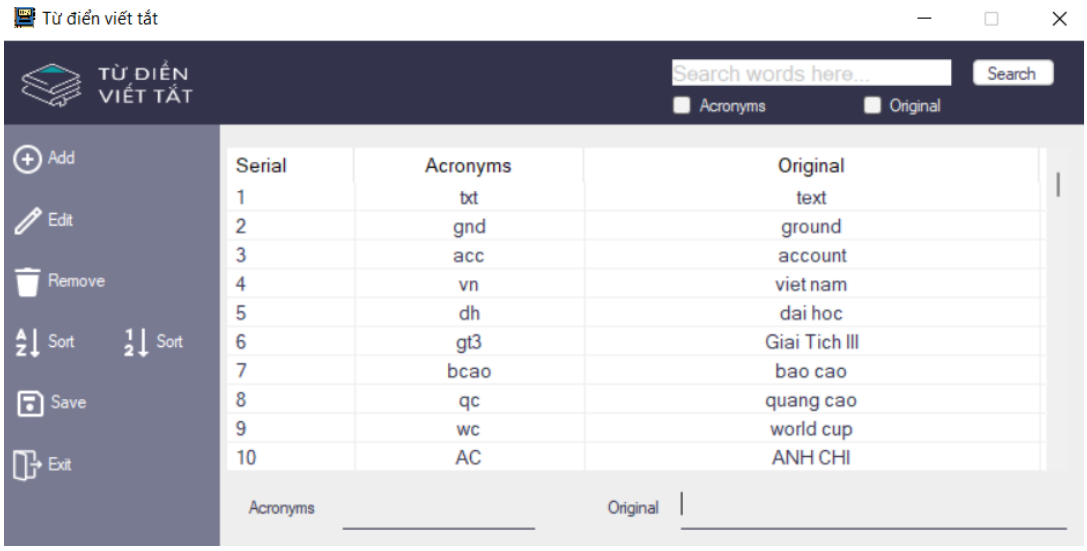
Cấu trúc lưu trữ:

- Là cấu trúc lưu trữ trong: Nằm trong bộ nhớ trong.
- Là loại cấu trúc lưu trữ động: kích thước dữ liệu có thể thay đổi khi chạy chương trình.

Định nghĩa cấu trúc struct :

```
     struct DATA_123 //Struct lưu danh sách theo thứ tự
thêm vào
{
     int stt;
     std::string tvt;
     std::string tg;
};
struct DATA_abc //Struct lưu danh sách theo thứ tự
alphabet
{
     int abc;
     std::string tvt;
     std::string tg;
     };
```

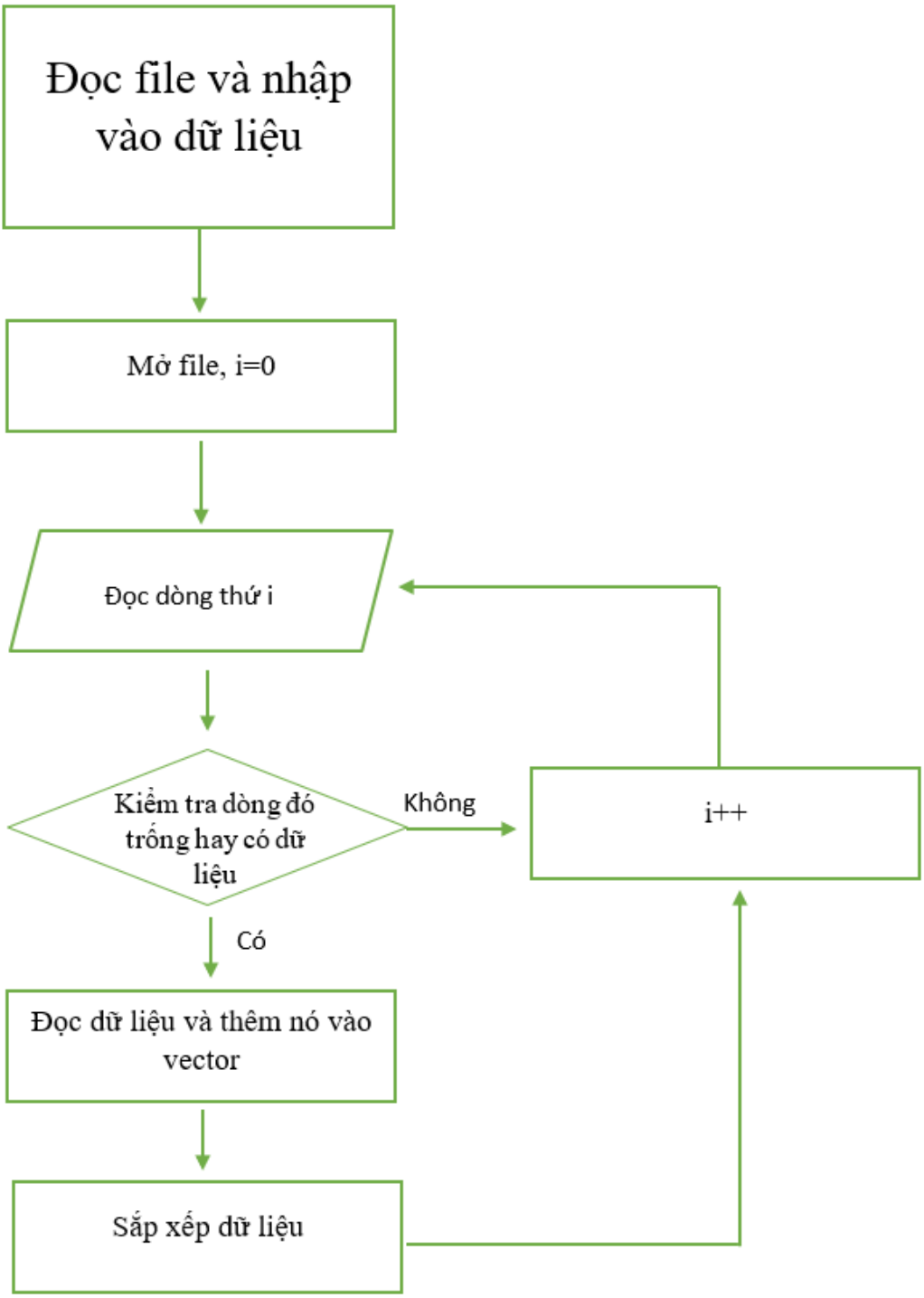
Dữ liệu mẫu hiển thị:



4. Thiết kế giải thuật :

4.1 Đọc file và nhập vào dữ liệu:

a) Thiết kế sơ bộ:



b) Thiết kế chi tiết:

Bước đầu tiên, mở file, khai báo các biến cần thiết và đọc toàn bộ dữ liệu có trong file.

```
//Hàm đọc file dữ liệu(Chỉ đọc 1 lần đầu tiên khi mở app)
//Lưu dữ liệu vào 2 vector là vector sx theo 123 và sx theo alphabet
void readFile(String^ fileName) {
    StreamReader^ DataIn = File::OpenText(fileName);
    String^ DataStr;
    //khai báo biến để sử dụng hàm sscanf bên dưới
    char* tuviettat = new char[20];
    char* tugoc = new char[50];
    int stt = 0;
```

Bước tiếp theo, đọc dữ liệu theo từng dòng và thêm vào vector.

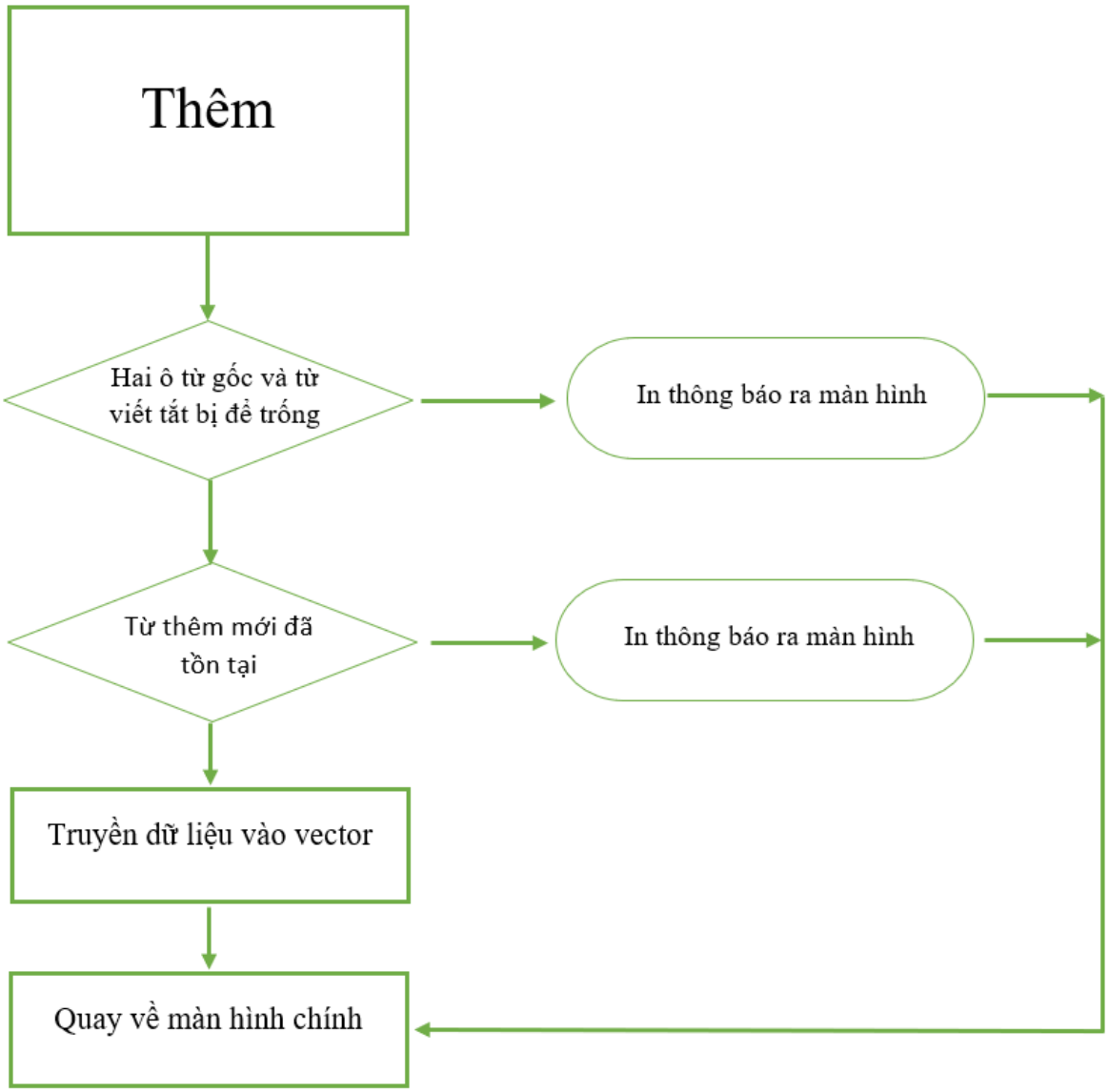
```
while ((DataStr = DataIn->ReadLine()) != nullptr){
      //chuyển DataStr(kiểu String^) sang string
      char cStr[50] = { 0 };
      if (DataStr->Length < sizeof(cStr)) sprintf(cStr, "%s", DataStr);
      //cStr là char hoặc string, còn phần trong ngoặc "" không thể là string
      sscanf(cStr, "[%d] %s %[^.]", &stt, tuviettat, tugoc);
      // biến temp là biến có kiểu DATA chứa dữ liệu của từng hàng
      temp_abc = TaoData_abc(stt, std::string(tuviettat), std::string(tugoc));
      temp_123 = TaoData_123(stt, std::string(tuviettat), std::string(tugoc));
      //push từng hàng vào vector DATABASE_
      DATABASE_abc.push_back(temp_abc);
      DATABASE_123.push_back(temp_123);
}
```

Bước cuối cùng, đóng file và thực hiện thuật toán sắp xếp.

```
DataIn->Close();// đóng file
delete[] tuviettat;
delete[] tugoc;
//Sắp xếp vector ABC theo alphabet theo giải thuật sắp xếp chèn(InsertionSort)
//InsertionSort(0, DATABASE_abc.size());
//Sắp xếp vector ABC theo alphabet theo giải thuật sắp xếp nhanh(QuickSort)
QuickSort(DATABASE_abc.size());
```

4.2 Thuật toán thêm vào:

a) Thiết kế sơ bộ:



b) Thiết kế chi tiết:

Kiểm tra xem từ nhập vào có phải là từ rỗng hay không, nếu không phải là từ rỗng thì thông báo ra màn hình.

```
private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
             if (txtTVT->Text == "" || txtTG -> Text == "") {
             MessageBox::Show("Can't perform this operation");
             return;
}
```

Nếu không thì mới xét xem từ nhập vào đã trùng từ gốc hoặc từ viết tắt hay chưa.

Tìm kiếm tuần tự theo vector lưu theo số thứ tự.Duyệt tất cả các phần tử của vector, đại diện là vector DATABASE_123

```
for (int i = 0; i < DATABASE_123.size(); i++) {
//Chuyển std::string sang String^
String^ str2 = gcnew String(DATABASE_123[i].tvt.c_str());
String^ str3 = gcnew String(DATABASE_123[i].tg.c_str());
```

Nếu tìm thấy từ trùng lặp thì cho dừng vòng chạy và in ra các thông báo:

```
      if (str2 == (txtTVT->Text) && str3 == (txtTG->Text)) {
            MessageBox::Show("This word already exists");
            return;
      }
      //Đã tồn tại từ gốc này
      else if (str3 == (txtTG->Text)) {
            MessageBox::Show("Original already exists");
            return;
      }
      //Đã tồn tại từ viết tắt này
      else if (str2 == (txtTVT->Text)) {
            MessageBox::Show("Acronyms already exists");
            return;
       }
```

```
"This word already exists" //Đã tồn tại từ này

"Original already exists"  //Đã tồn tại từ gốc này

"Acronyms already exists"  //Đã tồn tại từ viết tắt này
```

Khi kiểm tra hết các vector mà không có từ trùng lặp thì thuật toán sẽ bắt đầu thêm từ mới vào.
Lấy giữ liệu trong 2 ô textbox trên giao diện.

Cấp phát động 2 con trỏ với độ dài tương ứng để lưu dữ liệu của từ mới:

```
char* cStr1 = new char[txtTVT->Text->Length + 1];
sprintf(cStr1, "%s", txtTVT->Text);
char* cStr2 = new char[txtTG->Text->Length + 1];
sprintf(cStr2, "%s", txtTG->Text);
```

Tạo 2 biến là temp_123 với kiểu DATA_123 và temp_abc với kiểu DATA_abc để push vào 2 vector tương ứng

```
temp_abc = TaoData_abc(stt, std::string(tuviettat), std::string(tugoc));
temp_123 = TaoData_123(stt, std::string(tuviettat), std::string(tugoc));
```

Thêm vào vector xếp theo số thứ tự trước sau đó thêm vào sắp xếp alphabet:

Nếu là kiểu DATABASE_123 thì push ngay vào cuối.

```
temp_123 = TaoData_123(DATABASE_123.size() + 1, cStr1, cStr2);
DATABASE_123.push_back(temp_123);
```

Nếu là kiểu DATABASE_abc thì push như sau:

Duyệt từ đầu đến cuối của vector, đến khi từ viết tắt của từ mới bé hơn từ viết tắt thứ i trong danh sách thì thêm ngay vào vị trí i đó.

```
//Thêm từ mới vào trước vị trí i, dùng insert

//int p là vị trí thêm vào của từ mới trong ds sắp xếp theo alphabet
int p = -1;

for (int i = 0; i < DATABASE_abc.size(); i++) {
	if (cStr1 < DATABASE_abc[i].tvt) {
		DATABASE_abc.insert(DATABASE_abc.begin() + i, temp_abc);
		p = i;//VT thêm vào
		break;
	}
}
```

Biến p = -1 khi không có từ viết tắt trong danh sách lớn hơn từ viết tắt thêm vào.

Nếu cả vector không có từ viết tắt lớn hơn từ viết tắt mới thì thêm vào cuối vector:

```
if (p == -1) {
	DATABASE_abc.push_back(temp_abc);
	p = DATABASE_abc.size()-1;//VT thêm vào
}
```

Xóa dữ liệu con trỏ:

```
delete[] cStr1;
delete[] cStr2;
```

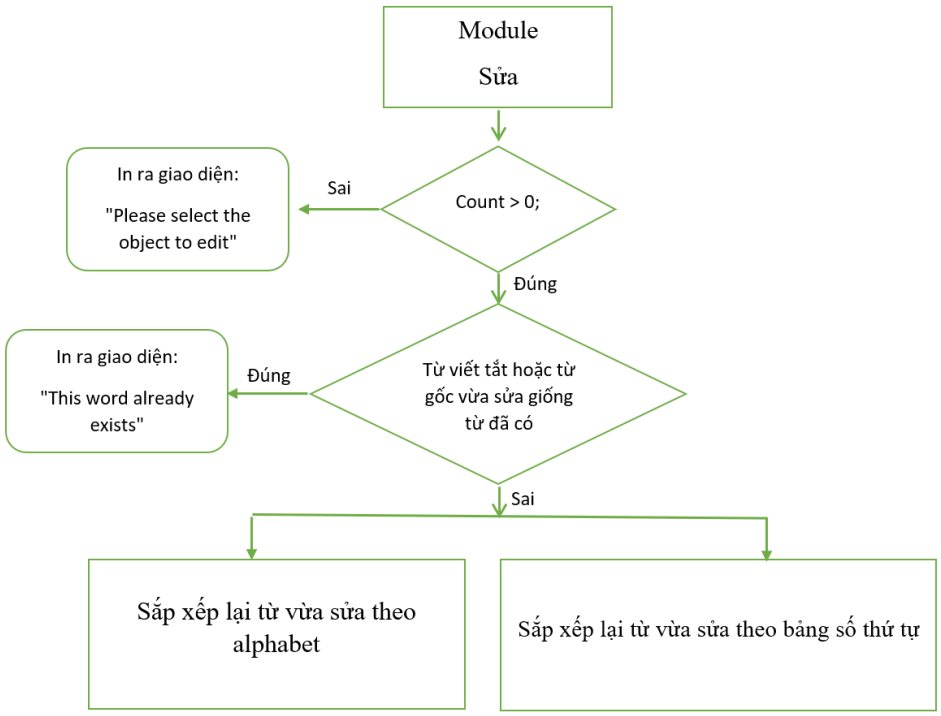
Sau khi xóa giữ liệu con trỏ, in ra màn hình đã thêm thành công.

Nếu type_sort == True thì ra màn hình theo sắp xếp alphabet, nếu không thì sẽ in ra theo số thứ tự đã được thêm vào:

```
if (type_sort == 0) {
      p = DATABASE_123.size();

      MessageBox::Show("Added success");
      display_data();
      listView1->Items[p - 1]->Selected = 1;
      listView1->Items[p - 1]->Focused = 1;
      listView1->Items[p - 1]->EnsureVisible();
      PanelTitleBar->Focus();
}
else if (type_sort == 1) {

      MessageBox::Show("Added success");
      display_data();
      listView1->Items[p]->Selected = 1;
      listView1->Items[p]->Focused = 1;
      listView1->Items[p]->EnsureVisible();
      PanelTitleBar->Focus();
}
```

4.3 Thuật toán sửa:

a) Thiết kế sơ bộ :



b) Thiết kế chi tiết:

Kiểm tra xem đã ấn chọn từ muốn sửa hay chưa:

```
if (listView1->SelectedItems->Count > 0){

{//…}

}

else {
   MessageBox::Show("Please select the object to edit");
}
```

Biến lưu thứ tự của từ sửa: `char cStr[10] = { 0 };`

Chuyển vị trí đối tượng thành kiểu int để xác định vị trí :

```
sprintf(cStr, "%s", listView1->SelectedItems[0]->Text);
int _stt = std::stoi(std::string(cStr));
```

Tạo con trỏ để lưu giá trị sửa được nhập vào:

```
char* cStr1 = new char[txtTVT->Text->Length + 1];
sprintf(cStr1, "%s", txtTVT->Text);
char* cStr2 = new char[txtTG->Text->Length + 1];
sprintf(cStr2, "%s", txtTG->Text);
```

Kiểm tra tuần tự xem từ mới vừa sửa vào có trùng từ viết tắt hoặc từ gốc đã có hay không :

```
for (int i = 0; i < DATABASE_123.size(); i++) {
      if (cStr1 == DATABASE_123[i].tvt || cStr2 == DATABASE_123[i].tg)
      {
      MessageBox::Show("This word already exists");
      return;
      }
}
```

Nếu chọn và sửa ở vector DATABASE_123 thì sau khi chuyển sang vector DATABASE_abc không biết từ đó là từ nào, vì 2 vector không có tính liên kết với nhau, nên do đó phải chia thành 2 trường hợp, đó là:

- Chọn và sửa khi đang ở chế độ sắp xếp theo số thứ tự(type_sort = 0)
- Chọn và sửa khi đang ở chế độ sắp xếp theo alphabet(type_sort = 1)

Với type_sort = 0: `if (type_sort == 0)`

Truyền giá trị từ giao diện vào vector DATABASE_123:

```
DATABASE_123[_stt - 1].tvt = cStr1;
DATABASE_123[_stt - 1].tg = cStr2;
```

Lần lượt so sánh số thứ tự của từ muốn sửa với số thứ tự của vector xếp theo alphabet, nếu giống nhau thì ta sẽ xóa từ gốc và từ viết tắt giống của vector xếp theo số thứ tự :

```
// xóa từ ở vị trí tương tự bên vector DATABASE_123
int p = -1;
for (int i = 0; i < DATABASE_abc.size(); i++) {
    //so sánh stt(tức biến abc)  = _stt thì xóa nó
    if (DATABASE_abc[i].abc == _stt) {
        temp_abc = TaoData_abc(DATABASE_abc[i].abc, cStr1, cStr2);
        DATABASE_abc.erase(DATABASE_abc.begin() + i);
        break;
    }
}
```

Duyệt từ đầu đến cuối của vector xếp theo alphabet, đến khi từ viết tắt của từ mới bé hơn từ viết tắt thứ i trong danh sách thì thêm ngay vào vị trí i đó.

```
//Sắp xếp từ mới được thêm, vào DATABASE_abc
for (int i = 0; i < DATABASE_abc.size(); i++) {
    if (cStr1 < DATABASE_abc[i].tvt) {
        DATABASE_abc.insert(DATABASE_abc.begin() + i, temp_abc);
        p = i;
        break;
    }
}
```

Biến p = -1 khi không có từ viết tắt trong danh sách lớn hơn từ viết tắt thêm vào.

Nếu cả vector không có từ viết tắt lớn hơn từ viết tắt mới thì thêm vào cuối vector:

```
//Nếu từ mới không bé hơn từ nào thì thêm vào cuối vector
if (p == -1) {
    DATABASE_abc.push_back(temp_abc);
    p = DATABASE_abc.size() - 1;
}
```

Xóa dữ liệu con trỏ:

```
delete[] cStr1;

delete[] cStr2;
```

In ra giao diện màn hình:

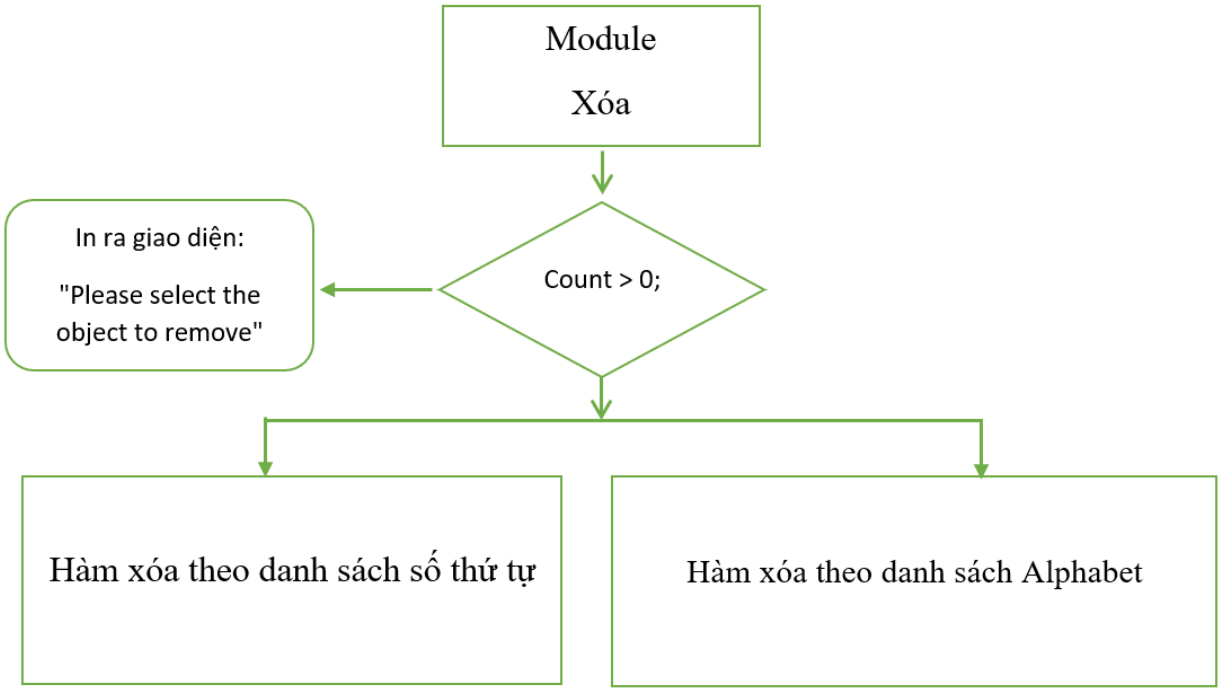
```
MessageBox::Show("Edited success");

display_data();

listView1->Items[_stt - 1]->Selected = 1;
listView1->Items[_stt - 1]->Focused = 1;
listView1->Items[_stt - 1]->EnsureVisible();
PanelTitleBar->Focus();
```

Với type_sort = 1:

```
else if (type_sort == 1) {
    //Ngược trường hợp trên 1 chút
    DATABASE_123[(DATABASE_abc[_stt - 1].abc) - 1].tvt = cStr1;
    DATABASE_123[(DATABASE_abc[_stt - 1].abc) - 1].tg = cStr2;

    temp_abc = TaoData_abc(DATABASE_abc[_stt - 1].abc, cStr1, cStr2);
    DATABASE_abc.erase(DATABASE_abc.begin() + _stt - 1);
    int p = -1;
    for (int i = 0; i < DATABASE_abc.size(); i++) {
        if (cStr1 < DATABASE_abc[i].tvt) {
            DATABASE_abc.insert(DATABASE_abc.begin() + i, temp_abc);
            p = i;
            break;
        }
    }
    if (p == -1) {
        DATABASE_abc.push_back(temp_abc);
        p = DATABASE_abc.size() - 1;
    }
    //Xoa du lieu con tro
    delete[] cStr1;
    delete[] cStr2;

    MessageBox::Show("Edited success");
    display_data();
    listView1->Items[p]->Selected = 1;
    listView1->Items[p]->Focused = 1;
    listView1->Items[p]->EnsureVisible();
    PanelTitleBar->Focus();
}
```

4.4 Thuật toán xóa:

a) Thiết kế sơ bộ:



b) Thiết kế chi tiết:

Biến lưu thứ tự của từ sửa: `char cStr[4] = { 0 };`

Chuyển vị trí đối tượng thành kiểu int để xác định vị trí :

```
sprintf(cStr, "%s", listView1->SelectedItems[0]->Text);
int _stt = std::stoi(std::string(cStr));
```

Tương tự như thuật toán sửa, nếu xóa ở vector DATABASE_123 thì sau khi chuyển sang vector DATABASE_abc không biết từ đó là từ nào, vì 2 vector không có tính liên kết với nhau, nên do đó phải chia thành 2 trường hợp, đó là:

- Chọn và sửa khi đang ở chế độ xóa theo số thứ tự (type_sort = 0)
- Chọn và sửa khi đang ở chế độ xóa theo alphabet (type_sort = 1)

Với type_sort = 0 : `if (type_sort == 0)`

Giảm mỗi số thứ tự ở sau phần tử ở vectorlưu theo số thứ tự được xóa một đơn vị:

```
for (int i = _stt; i < DATABASE_123.size(); i++) {
	(DATABASE_123[i].stt)--;
}
```

Xóa phần tử được chọn: `DATABASE_123.erase(DATABASE_123.begin() + _stt - 1);`

Lần lượt so sánh số thứ tự của từ muốn sửa với số thứ tự của vector xếp theo alphabet, nếu giống nhau thì ta sẽ xóa từ gốc và từ viết tắt giống của vector xếp theo số thứ tự :

```
for (int i = 0; i < DATABASE_abc.size(); i++) {
	if (DATABASE_abc[i].abc == _stt) {
		DATABASE_abc.erase(DATABASE_abc.begin() + i);
		break;
	}
}
```

Giảm mỗi số thứ tự ở sau phần tử ở vector lưu theo alphabet được xóa một đơn vị:

```
for (int i = 0; i < DATABASE_abc.size(); i++) {
	if (DATABASE_abc[i].abc > _stt) {
		(DATABASE_abc[i].abc)--;
	}
}
```

Tương tự với type_sort = 1, ta có:

```
else if (type_sort == 1) {

    for (int i = (DATABASE_abc[_stt - 1].abc); i < DATABASE_123.size(); i++) {
        (DATABASE_123[i].stt)--;
    }
    DATABASE_123.erase(DATABASE_123.begin() + (DATABASE_abc[_stt - 1].abc) - 1);

    for (int i = 0; i < DATABASE_abc.size(); i++) {
        if (DATABASE_abc[i].abc > (DATABASE_abc[_stt - 1].abc)) {
            (DATABASE_abc[i].abc)--;
        }
    }
    DATABASE_abc.erase(DATABASE_abc.begin() + _stt - 1);
}
```

4.5 Thuật toán sắp xếp:

Ở đây, do thuật toán sắp xếp theo thứ tự nhập vào đã được sắp xếp ngay khi nhập nên chỉ cần xây dựng thuật toán sắp xếp theo alphabet.

4.5.1: Thuật toán InsertionSort:

a. Thiết kế sơ bộ:

Bước 1: truyền vào hàm giá trị m = 0 và n = kích thước vector, gán giá trị min = a[0].

Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[n-1]

- Nếu a[i] < min thì đổi vị trí của a[min] và a[i]
- Nếu i < N thì tiếp tục chạy, i= i+1;

Lặp lại bước 2

b. Thiết kế chi tiết:

Hàm sắp xếp chèn:

```
// Hàm sắp xếp chèn
void InsertionSort(int m, int n) {
    for (int i = m; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (DATABASE_abc[i].tvt > DATABASE_abc[j].tvt) {
                swap(i, j);
            }
        }
    }
}
```

Hàm đổi vị trí :

```
void swap(int i, int j)
{
	string tmp = DATABASE_abc[i].tvt;
	DATABASE_abc[i].tvt = DATABASE_abc[j].tvt;
	DATABASE_abc[j].tvt = tmp;
	tmp = DATABASE_abc[i].tg;

	DATABASE_abc[i].tg = DATABASE_abc[j].tg;
	DATABASE_abc[j].tg = tmp;
	int _tmp = DATABASE_abc[i].abc;

	DATABASE_abc[i].abc = DATABASE_abc[j].abc;
	DATABASE_abc[j].abc = _tmp;
}
```

4.5.2: Thuật toán Quick Sort:

a) Thiết kế sơ bộ:

- Truyền vào vector giá trị kích thước của vecto
- Chọn phần tử chính giữa làm pivot để chia đôi.
- Cho giá trị chạy từ điểm trái (nhỏ nhất của Partition) có giá trị là i; đến điểm phải (lớn nhất của Partition) có giá trị là j
- Nếu điểm trái >= điểm phải thì dừng lại và return
- Nếu i < j thì:
  - + Nếu phần tử bên trái nhỏ hơn pivot thì bỏ qua, i ++;
  - + Nếu phần tử bên phải nhỏ hơn pivot thì bỏ qua; j--
  - + Nếu i < j thì đổi chỗ i và j, sau đó i++ và j--
- Đệ quy: truyền vào hàm Partition(trái, j);
- Đệ quy: truyền vào hàm Partition(phải, j);

b) Thiết kế chi tiết:

Hàm Partition:

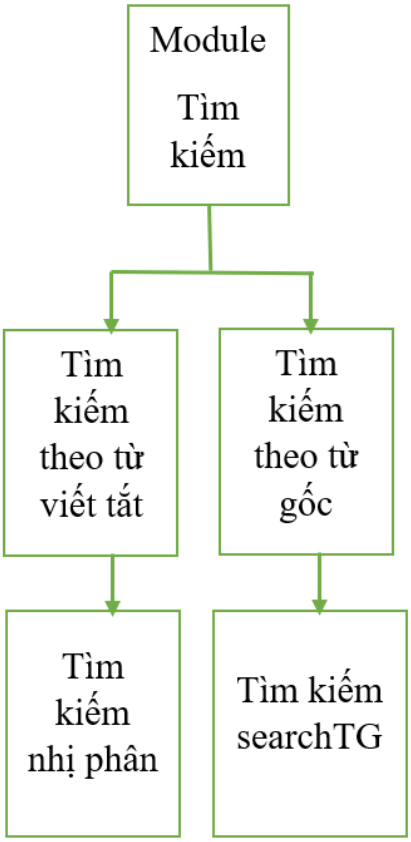
```
void Partition(int left, int right)
{

   if (left >= right)return;

   string pivot = DATABASE_abc[(left + right) / 2].tvt;
   int i = left, j = right;
   while (i < j)
         {
         while (DATABASE_abc[i].tvt < pivot) i++;
         while (DATABASE_abc[j].tvt > pivot) j--;
         if (i <= j)
          {
                if (i < j) swap(i, j);
                i++;
                j--;
         }
   }
```

Hàm hoán đổi:

```
void swap(int i, int j)
{
       string tmp = DATABASE_abc[i].tvt;
       DATABASE_abc[i].tvt = DATABASE_abc[j].tvt;
       DATABASE_abc[j].tvt = tmp;
       tmp = DATABASE_abc[i].tg;
       DATABASE_abc[i].tg = DATABASE_abc[j].tg;
       DATABASE_abc[j].tg = tmp;
       int _tmp = DATABASE_abc[i].abc;
       DATABASE_abc[i].abc = DATABASE_abc[j].abc;
       DATABASE_abc[j].abc = _tmp;
}
```

4.6. Thuật toán tìm kiếm:

a) Thiết kế sơ bộ:



b) Thiết kế chi tiết:

Tạo con trỏ lưu từ cần tìm kiếm:

```
char* TCT = new char[textBox1->Text->Length + 1];
sprintf(TCT, "%s", textBox1->Text);
int tt;
```

❖ **Module thuật toán tìm kiếm theo từ viết tắt:**

Giải thuật tìm kiếm nhị phân:

```
int BinarySearchTVT(vector<DATA_abc> database, string word) {
    return BSearchTVT(database, word, 0, database.size() - 1);
}
```

Hàm BSearch :

Xét đoạn mảng a[m…n] cần tìm kiếm phần tử tvt. Ta so sánh tvt với phần tử ở vị trí giữa của mảng (k= (m + n)/2). Nếu:

- Nếu phần tử a[k] = tvt. Kết luận và thoát chương trình.
- Nếu a[k] < tvt. Chỉ thực hiện tìm kiếm trên đoạn arr[k+1…n].

- Nếu a[k] > tvt. Chỉ thực hiện tìm kiếm trên đoạn arr[m…k-1].

```
int BSearchTVT(vector<DATA_abc> database, string word, int m, int n) {
	if (m > n) return -1;
	int k = (m + n) / 2;
	if (word == database[m].tvt) return m;
	else {
	if (word < database[m].tvt) return BSearchTVT(database, word, m, k - 1);
		else return BSearchTVT(database, word, m + 1, n);
	}
}
```

Sau đó, nếu không tìm được vị trí, giá trị của tt vẫn bằng -1, khi đó, in ra giao diện:

```
if (tt == -1) {
	MessageBox::Show("Can't find this word");
}
```

Nếu tìm được giá trị, in ra màn hình theo:

Với type_sort = 1:

In đậm từ tìm kiếm theo alphabet:

```
if (type_sort == 1) {
	listView1->Items[tt]->Selected = 1;
	listView1->Items[tt]->Focused = 1;
	listView1->Items[tt]->EnsureVisible();
}
```

Với type_sort = 0:

In đậm từ tìm kiếm theo số thứ tự

```
else if (type_sort == 0) {
	for (int i = 0; i < DATABASE_abc.size(); i++) {
		if (DATABASE_abc[tt].abc == DATABASE_123[i].stt) {
			listView1->Items[i]->Selected = 1;
			listView1->Items[i]->Focused = 1;
			listView1->Items[i]->EnsureVisible();
			break;
		}
	}
}
```

❖ **Module thuật toán tìm kiếm theo từ gốc:**

Sử dụng giải thuật tìm kiếm tuần tự:

```
int SearchTG(std::vector<DATA_123> database, std::string word) {
	for (int i = 0; i < database.size(); i++) {
		if (database[i].tg == word) {
			return i;
		}
	}
return -1;
}
```

Sau đó, nếu không tìm được vị trí, giá trị của tt vẫn bằng -1, khi đó, in ra giao diện:

```
if (tt == -1) {
	MessageBox::Show("Can't find this word");
}
```

Nếu tìm được giá trị, in ra màn hình theo:

- Với type_sort = 0: (in đậm từ tìm kiếm theo số thứ tự):

```
if (type_sort == 0) {
    listView1->Items[tt]->Selected = 1;
    listView1->Items[tt]->Focused = 1;
    listView1->Items[tt]->EnsureVisible();
}
```

- Với type_sort = 1: (in đậm từ tìm kiếm theo alphabet):

```
else if (type_sort == 1) {
    for (int i = 0; i < DATABASE_abc.size(); i++) {
        if (tt + 1 == DATABASE_abc[i].abc) {
            listView1->Items[i]->Selected = 1;
            listView1->Items[i]->Focused = 1;
            listView1->Items[i]->EnsureVisible();
            break;
        }
    }

}
```

5. Phân tích tính đúng đắn của giải thuật.

5.1. Phân tích tính đúng đắn của GT :

Qua nhiều lần chạy thử nghiệm, chúng em thấy chương trình hoạt động khá ổn định, thực hiện được các chức năng như năng như yêu cầu của bài toán đưa ra, giao diện thân thiện và dễ nhìn.

Việc chạy chương trình không có lỗi gì xảy ra.

File dữ liệu thử lên đến hơn 100 cụm từ viết tắt, gồm các từ khóa đặc biệt, từ nhiều kí tự, các chữ số...

File dữ liệu thử :

```
78] SWB Single With Breakfast.
79] TWN Twin bed room.
80] STD Standard.
81] ROH Run of the house.
82] SUP Superior.
83] PhD Doctor of Philosophy.
84] JD Juris Doctor.
85] PA Personal Assistant.
86] MD Managing Director.
87] VP Vice President.
88] SVP Senior Vice President.
89] EVP Executive Vice President.
90] CMO Chief Marketing Officer.
91] CFO Chief Financial Officer.
92] CEO Chief Executive Officer.
93] fb facebook.
94] ACC Account.
95] ASAP As soon as possible.
96] OMG oh my god.
97] HUST Bach Khoa Ha Noi.
98] NEU kinh te quoc dan.
99] FTU Dai Hoc Ngoai Thuong.
100] HMU Dai Hoc Y.
101] QTKD Quan tri kinh doanh.
102] KTQT Kinh te Quoc Te.
103] IT cong nghe thong tin.
104] ET Dien tu vien thong.
105] BKCK Bach Khoa Co Khi.
106] TM Tach Mon.
107] hobo Hoc Bong.
108] UET Dai Hoc Quoc Gia.
109] UEB Dai hoc kinh te.
110] PTIT Buu chinh vien thong.
111] SKDA San Khau Dien Anh.
112] 113 canh sat.
113] bik biet.
114] 114 cuu hoa.
115] 115 cuu thuong.
```

5.2. Xác định độ phức tạp của GT đề xuất :

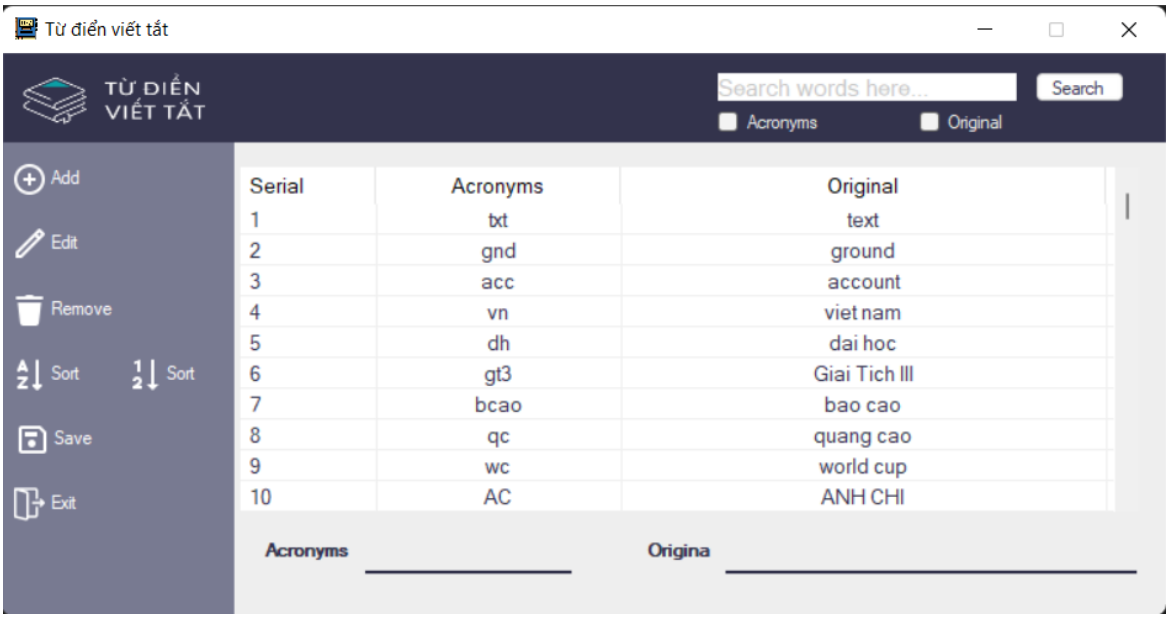
- Độ phức tạp của thuật toán đọc file O(n)
- Độ phức tạp của module thêm, sửa, xóa là O(n)
- Độ phức tạp của hàm in ra màn hình giao diện có độ phức tạp là O(n)
- Thuật toán sắp xếp (Selection Sort) là một thuật toán khá đơn giản có độ phức tạp là $O(n^2)$.
- Thuật toán sắp xếp nhanh (Quick Sort) là một thuật toán có độ phức tạp trong trường hợp xấu là $O(n^2)$ và trong trường hợp trung bình là O(nlogn)

- Độ phức tạp của thuật toán tìm kiếm nhị phân là: O = $log_2(n)$
- Độ phức tạp của thuật toán tìm kiếm tuần tự là: O = $log_2(n)$

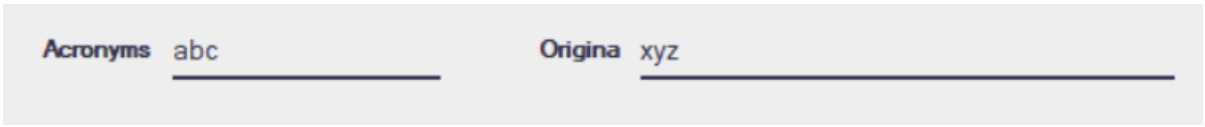
6. Mô tả triển khai trên ngôn ngữ lập trình cụ thể:

Khi chạy, tạo ra 1 giao diện có nhiều module để lựa chọn :

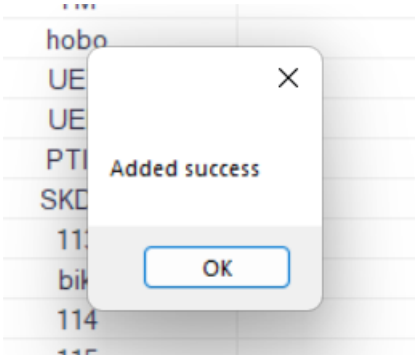
- **Module add:**



Khi nhập từ viết tắt và từ gốc vào đây :

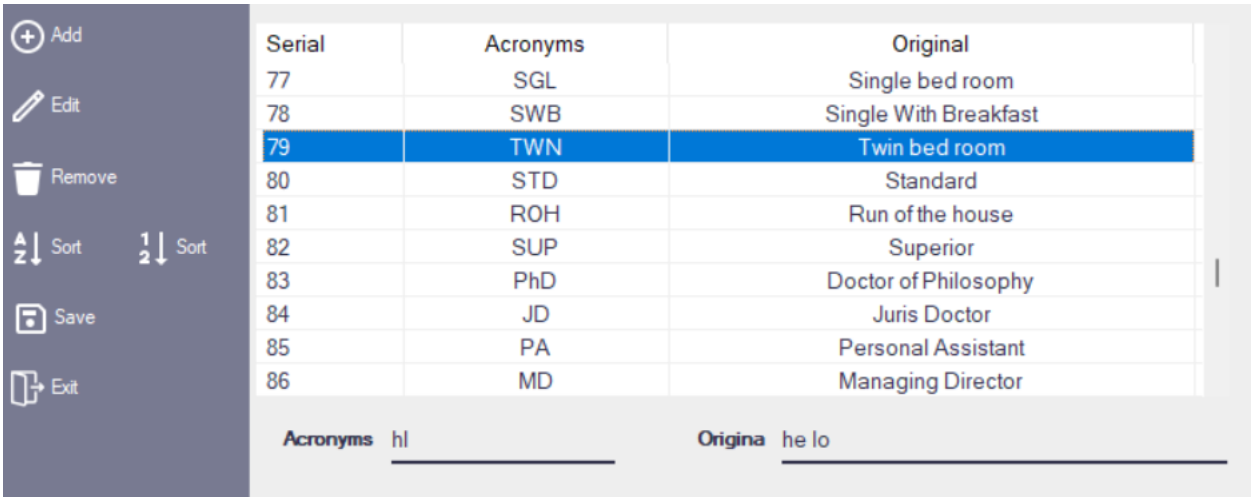


Và ấn vào “Add” sẽ cho ra kết quả như sau:

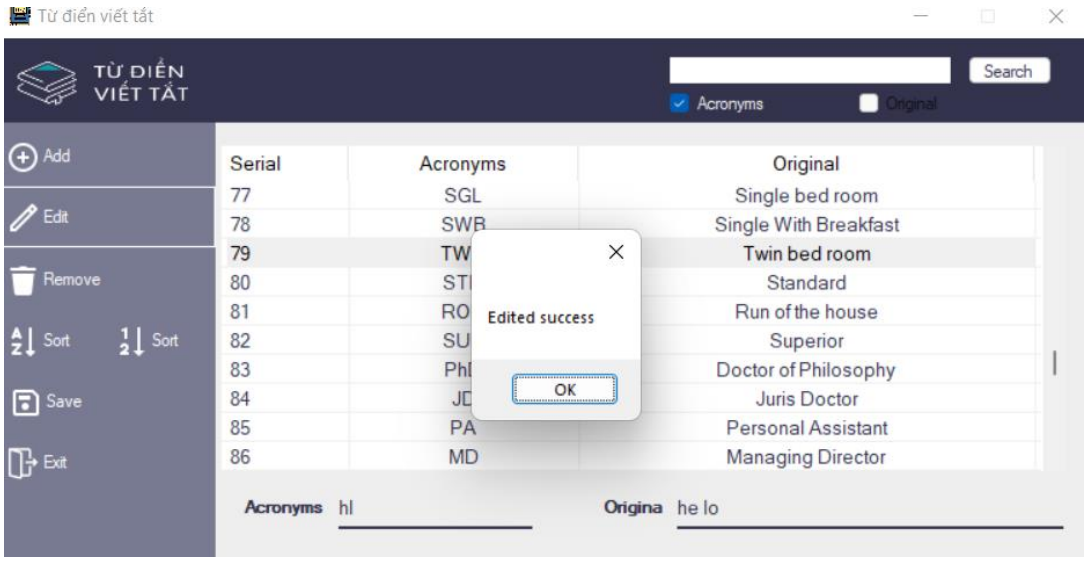


- **Module Edit:**

Khi ấn vào từ cần edit, sau đó nhập lại từ cần edit như hình dưới đây :



Ấn vào "Edit" thành công sẽ ra kết quả như sau:



- **Module sắp xếp:**

Khi chọn vào sắp xếp theo alphabet, ấn vào biểu tượng : 

Kết quả khi sắp xếp theo alphabet:

Serial	Acronyms	Original
1	113	canh sat
2	114	cuu hoa
3	115	cuu thuong
4	AC	ANH CHI
5	ACC	Account
6	ACE	anh chi em
7	ASAP	As soon as possible
8	BCH	Ban Chap Hanh
9	BKCK	Bach Khoa Co Khi
10	CEO	Chief Executive Officer

Khi chọn vào sắp xếp theo số thứ tự, ấn vào biểu tượng : 

Kết quả khi sắp xếp theo số thứ tự:

Serial	Acronyms	Original
1	txt	text
2	gnd	ground
3	acc	account
4	vn	viet nam
5	dh	dai hoc
6	gt3	Giai Tich III
7	bcao	bao cao
8	qc	quang cao
9	wc	world cup
10	AC	ANH CHI

7. Tự đánh giá kết quả, các điểm hạn chế:

Về kết quả, sau nhiều lần chạy thử nghiệm trên chính máy tính cá nhân và một số máy tính của bạn bè, chúng em đã hạn chế đến mức tối đa các lỗi có thể xuất hiện trên app của mình, đồng thời chạy chính xác trên tập dữ liệu mẫu.

Về mặt hạn chế, nhóm chúng em tồn tại duy nhất một vài điểm nhỏ về mặt triển khai bài toán. Thứ nhất, do tính năng của một số lớp có sẵn, giải pháp tối ưu nhóm chúng em tìm được bắt buộc phải tạo ra 2 vector trong giải thuật sắp xếp, dẫn đến tốn bộ nhớ hơn khi chạy. Thứ hai, do giới hạn về mặt công cụ, bắt buộc các chức năng của các buttom phải nằm trong file header mà chương trình đã tạo, dẫn đến file header của chúng em có tổng số dòng code hơn 1000 dòng.

Lời cảm ơn

Trên đây là báo cáo của nhóm 4 về đề tài Từ điển viết tắt. Do lần đầu được làm việc với nhau, cũng do nhiều yếu tố khác dẫn đến bản báo cáo của nhóm chúng em có thể vẫn còn nhiễu thiếu sót và lỗi. Mong cô và các bạn có thể đóng góp thêm ý kiến để nhóm chúng em có thể rút kinh nghiệm và hoàn thiện bản thân mình hơn trong tương lai.

Cuối cùng, chúng em xin phép gửi lời cảm ơn đến cô Đỗ Thị Ngọc Diệp, người đã giảng dạy và hướng dẫn chúng em trong suốt thời gian qua. Một lần nữa cảm ơn cô rất nhiều.