ĐẠI HỌC UEH
TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ
KHOA CÔNG NGHỆ THÔNG TIN KINH DOANH

ĐỀ TÀI
ỨNG DỤNG CẤU TRÚC DICTIONARY ĐỂ TẠO MỘT BỘ TỪ ĐIỂN ANH – VIỆT

Học phần: Cấu trúc dữ liệu & giải thuật

Danh sách nhóm:
1.Trương Thị Kiều Phương
2. Đoàn Thị Mai Linh
3. Bùi Anh Minh
4. Bùi Thị Thùy Linh

CHUYÊN NGÀNH: KHOA HỌC MÁY TÍNH
KHÓA: 50

Giảng viên: TS. Đặng Ngọc Hoàng Thành

TP.HCM ngày 4 tháng 4 năm 2025

Mục lục

Lời mở đầu

Trong thời đại số hoá, việc tra cứu thông tin, đặc biệt là từ ngữ, diễn ra hằng ngày. Một ứng dụng từ điển, dù đơn giản hay phức tạp thì đều có một điểm chung: Phải cho phép người dùng **tra cứu nhanh một từ và biết nghĩa của nó**. Điều này đặt ra yêu cầu rằng dữ liệu bên trong cần được **tổ chức sao cho việc tìm kiếm theo từ khóa là nhanh nhất có thể**. Đặc biệt, giải tình huống thực tế triển khai bảng băm, một vấn đề không thể tránh khỏi là **va chạm chỉ số (collision)** - tức là khi hai từ khác nhau nhưng sau khi qua hàm băm lại cho ra cùng một vị trí trong bảng.

Có nhiều cách để lưu trữ từ và nghĩa, ví dụ như dùng danh sách tuyến tính (mảng, danh sách liên kết), nhưng với cách đó thì mỗi lần tra từ, hệ thống phải duyệt từng phần tử để tìm, rất chậm khi dữ liệu lớn.

Chính vì vậy, cấu trúc phù hợp hơn là một **dictionary** - tức là một kiểu dữ liệu cho phép ánh xạ từ **một khóa (key)** sang **một giá trị (value)**, giúp việc truy xuất trở nên nhanh chóng, thường chỉ trong **thời gian hằng số (O(1))** nếu cấu trúc được tổ chức tốt.

Đề tài nhóm thực hiện mang tên *"Ứng dụng cấu trúc Dictionary để tạo ra một bộ từ điển Anh-Việt"*. Trong phạm vi đề tài, "Dictionary" không chỉ đơn giản là một lớp lập trình có sẵn, mà được hiểu là một kiểu cấu trúc dữ liệu dùng để lưu trữ và truy xuất thông tin theo dạng cặp khóa - giá trị (key - value).

Từ góc nhìn đó, nhóm lựa chọn tiếp cận bài toán bằng cách xây dựng lại cấu trúc dữ liệu nền tảng một cách thủ công, thay vì sử dụng các thư viện dựng sẵn. Đây không chỉ là yêu cầu từ phía đề tài, mà còn là cơ hội để nhóm rèn luyện tư duy về cách tổ chức và xử lý dữ liệu hiệu quả thông qua việc lập trình từ cấu trúc lõi.

Chương 1: Cấu trúc từ điển Dictionary

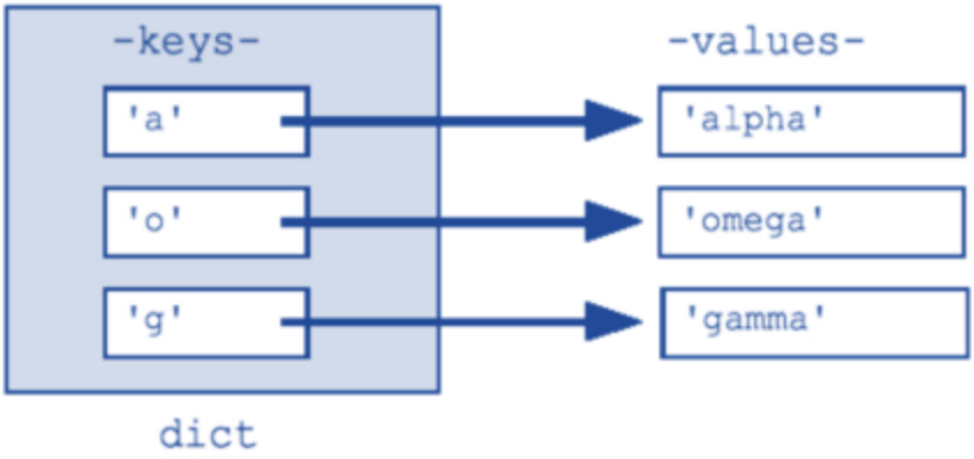
1.1. Khái niệm Dictionary

Trong đề tài này, "Dictionary" không được hiểu theo nghĩa là một lớp có sẵn trong ngôn ngữ lập trình (như Dictionary<,> trong C#), mà được tiếp cận như một cấu trúc dữ liệu dùng để lưu trữ các cặp dữ liệu theo dạng khóa (key) và giá trị (value). Ở đó, mỗi khóa là duy nhất, cho phép truy xuất giá trị tương ứng một cách nhanh chóng và hiệu quả.

1.2. Cấu trúc & Cài đặt từ điển

Định nghĩa

Cụ thể, từ tiếng Anh sẽ đóng vai trò là **khóa (key)**, và thông tin liên quan (nghĩa tiếng Việt, loại từ, phiên âm) sẽ là **giá trị (value)**. Việc thiết kế cấu trúc sao cho có thể **tìm được nghĩa nhanh nhất có thể** từ một từ khóa là mục tiêu trọng tâm của chương trình.



(Cấu trúc Dictionary)

- Từ điển là cấu trúc dữ liệu lưu trữ các cặp khóa (key) và giá trị (value). (A dictionary is a data structure that stores key-value pairs.)

- Mỗi khóa là duy nhất và ánh xạ tới một giá trị. (Each key is unique and maps to one value.)

Cài đặt từ điển

1. DictionaryBase:

- Sử dụng lớp cơ bản DictionaryBase để quản lý các cặp Key-Value trị.
- Người dùng có thể mở rộng lớp này để định nghĩa cách thêm, xóa và truy cập dữ liệu.

```
using System;
using System.Collections;

0 references
public class DictionaryManager : DictionaryBase
{
    0 references
    public void AddWord(string word, string meaning)
    {
        base.InnerHashtable.Add(word, meaning);
    }

    0 references
    public bool Contains(string word)
    {
        return
            base.InnerHashtable.Contains(word);
    }
```

2. Generic Dictionary:

- Hỗ trợ khai báo rõ kiểu dữ liệu cho cả khóa và giá trị.
- Linh hoạt và an toàn khi làm việc với dữ liệu được định kiểu rõ ràng.

```
Dictionary<string,string> myips = new Dictionary<string,string>();
            myips.Add("Mike", "192.155.12.1");
```

3. SortedList:

- Là một cấu trúc danh sách kết hợp, trong đó các cặp Key-Value được lưu trữ theo thứ tự tăng dần của khóa.

- Phù hợp khi cần dữ liệu luôn được sắp xếp.

```
SortedList myips = New SortedList(); //hoặc SortedDictionary
            myips.Add("Mike", "192.155.12.1");
            myips.Add(“David", "192.155.12.2");
```

Cấu Trúc Bảng Băm (Hash-Table)

Định Nghĩa

Bảng băm là một cấu trúc dữ liệu dùng để ánh xạ các khóa (Key) tới các giá trị (Value) thông qua hàm băm (Hash Function). Hàm băm chuyển đổi một khóa thành một chỉ số trong mảng. Bảng băm giúp tối ưu hóa việc tìm kiếm và quản lý dữ liệu.

Ưu điểm: Tìm kiếm nhanh, quản lý dữ liệu hiệu quả → thích hợp với các ứng dụng cần truy cập dữ liệu nhanh chóng.

Cài Đặt Bảng Băm

1. Lớp BucketHash:

- Tạo cấu trúc bảng băm cơ bản với kích thước mảng cố định.
- Dữ liệu được phân tán vào các ô nhớ dựa trên hàm băm.

```
public class BucketHash{
   private const int SIZE = 10;
   ArrayList[] data;
   public BucketHash(){
      data = new ArrayList[SIZE];
      for(int i=0; i<=SIZE-1; i++)
         data[i] = new ArrayList(4);
   }
```

2. Lớp Hashtable:

- Là một lớp có sẵn trong C#, hỗ trợ quản lý các cặp khóa-giá trị với hiệu suất cao.
- Có khả năng xử lý xung đột khóa thông qua cơ chế liên kết.

```
Hashtable infos = new Hashtable(5);
infos.Add("salary", 100000);
infos.Add("name", "David Job");
infos.Add("age", 45);
```

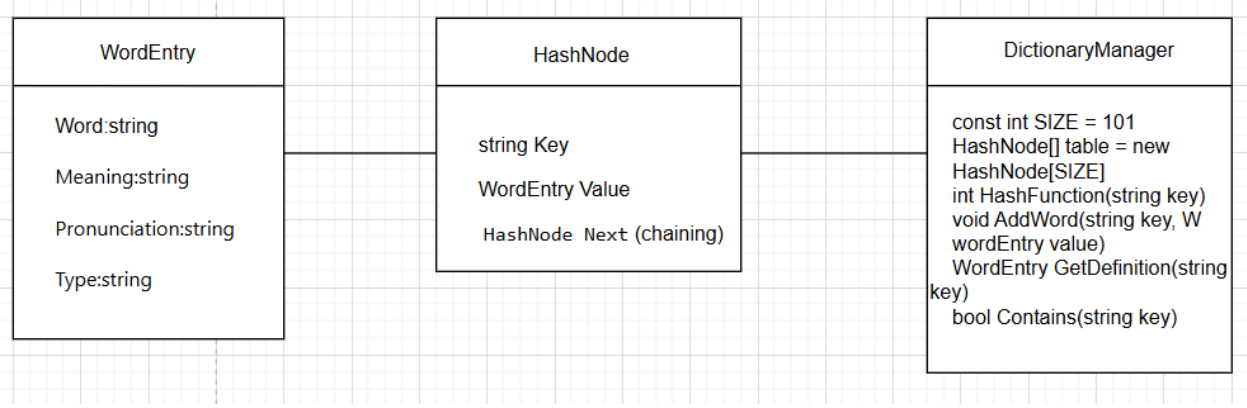
Chương 2. Phân tích và thiết kế lớp

2.1. Phân tích yêu cầu và ứng dụng cấu trúc Dictionary trong thiết kế từ điển Anh - Việt

Để sử dụng cấu trúc Dictionary, đầu tiên ta cần xác định tập key và tập value sẽ chứa giá trị gì. Trong tình huống này, vì đang xây dựng bộ từ điển nên ta cho tập key là các từ tiếng Anh cần tra và tập value là thông tin của các từ đó bao gồm:

- Nghĩa tiếng Việt
- Loại từ (danh từ, động từ, tính từ, trạng từ, giới từ,...)
- Phiên âm (IPA)

2.2. Sơ đồ lớp



2.3. Cài đặt lớp

Lớp WordEntry – Lưu trữ thông tin của từ

Để lưu trữ thông tin của từ, ta sẽ xây dựng lớp WordEntry để chứa các thuộc tính cần thiết của một từ điển, bao gồm:

```
public class WordEntry
{
    0 references
    public string Word { get; set; }
    1 reference
    public string Meaning { get; set; }
    1 reference
    public string Pronunciation { get; set; }
    1 reference
    public string Type { get; set; }
}
```

(code trong lớp WordEntry)

Trong đó:

- Word: từ tiếng Anh.
- Meaning: nghĩa tiếng Việt.
- Pronunciation: phiên âm.
- Type: loại từ (danh từ, động từ, tính từ).

Việc đóng gói dữ liệu như trên giúp việc quản lý, tra cứu và hiển thị từ điển trở nên thuận tiện, rõ ràng hơn.

Lớp HashNode – Đại diện cho mỗi phần tử trong bảng băm

Để xây dựng cấu trúc bảng băm nhằm lưu trữ và tra cứu từ điển nhanh chóng, ta định nghĩa lớp HashNode. Mỗi đối tượng HashNode đại diện cho một nút (node) trong một "bucket" của bảng băm. Cấu trúc lớp như sau:

```
public class HashNode
{
    public string Key;
    public WordEntry Value;
    public HashNode Next;
```

(code trong lớp dictionaryManager)

Trong đó:

- Key: là chuỗi từ tiếng Anh (sử dụng để băm).
- Value: là đối tượng WordEntry chứa toàn bộ thông tin của từ.
- Next: tham chiếu đến node kế tiếp trong trường hợp xảy ra va chạm (collision).

Khi xảy ra va chạm (nhiều từ có cùng chỉ số băm), các node sẽ được liên kết theo dạng danh sách liên kết đơn (linked list), đảm bảo không mất dữ liệu.

```
public HashNode(string key, WordEntry value)
{
    Key = key;
    Value = value;
    Next = null;
}
```

(code trong lớp dictionaryManager)

Hàm tạo (constructor) của lớp HashNode. Mục đích của nó là **khởi tạo một đối tượng HashNode mới** khi bạn muốn thêm một phần tử (node) vào bảng băm.

public HashNode(string key, WordEntry value) là **hàm tạo giúp tạo một node mới** trong bảng băm, chứa key, value và thiết lập node tiếp theo ban đầu là null. Đây là bước đầu tiên để xây dựng một bảng băm có xử lý va chạm bằng **chaining (danh sách liên kết)**.

Đọc dữ liệu và khởi tạo bảng từ điển

File dữ liệu chứa danh sách từ vựng, mỗi dòng gồm 4 trường phân cách bằng dấu phẩy và khoảng trắng:

Từ tiếng Anh, Nghĩa tiếng Việt, Loại từ, Phiên âm

```
pure, thuần khiết, tính từ, /pjʊə(r)/
```

(từ điển trong note)

Trong Form1, đọc từng dòng của file, phân tách bằng dấu ", ", sau đó tạo đối tượng WordEntry tương ứng và chèn vào bảng băm:

```
foreach (var line in File.ReadAllLines(dictionaryPath))
{
    var parts = line.Split(new string[] { ", " }, StringSplitOptions.None);
    if (parts.Length == 4)
    {
        string english = parts[0].Trim();
        string vietnamese = parts[1].Trim();
        string partOfSpeech = parts[2].Trim();
        string pronunciation = parts[3].Trim();
```

(code trong lớp form1)

Việc chia nhỏ dữ liệu từ file và gán vào các thuộc tính của WordEntry giúp chương trình dễ dàng quản lý và hiển thị thông tin mỗi khi người dùng tra từ.

2.4. Các phương thức của lớp

2.4.1. Cấu trúc bảng băm và cơ chế ánh xạ từ khóa

Để xây dựng một từ điển dạng **key-value**, trong đó mỗi từ tiếng Anh (key) ánh xạ đến nghĩa tiếng Việt (value), ta cần một cấu trúc dữ liệu giúp tra cứu nhanh chóng và hiệu quả. **Bảng băm (hash table)** là một lựa chọn lý tưởng cho yêu cầu này.

```
private const int SIZE = 101;
private HashNode[] table;
```

Trong đoạn mã trên:

- SIZE = 101 là kích thước của bảng băm – tức số lượng ô (slots) trong mảng. Việc chọn số nguyên tố như 101 giúp giảm khả năng xảy ra va chạm (collision) khi nhiều khóa cùng ánh xạ đến một vị trí.
- table là mảng chứa các phần tử kiểu HashNode. Mỗi phần tử trong table[i] có thể là null hoặc một HashNode lưu trữ một cặp từ và nghĩa. Nếu xảy ra va chạm, các phần tử có thể được nối với nhau bằng danh sách liên kết (chaining).

Tuy nhiên, để bảng băm hoạt động hiệu quả, cần có một **hàm băm (hash function)** – cơ chế quyết định mỗi từ khóa sẽ nằm ở ô nào trong bảng.

```
private int HashFunction(string key)
{
    int hash = 0;
    foreach (char c in key)
    {
        hash = (hash * 31 + c) % SIZE;
    }
    return hash;
```

(code trong lớp dictionaryManager)

Đây là hàm chuyển một chuỗi key thành chỉ số nguyên trong phạm vi từ 0 đến SIZE - 1. Cụ thể:

- Ban đầu, hash được khởi tạo bằng 0.
- Hàm duyệt từng ký tự c trong chuỗi key. Mỗi ký tự được chuyển thành giá trị mã ASCII, sau đó áp dụng công thức:

⎚hash=(hash×31+c)%SIZE

- Số 31 là một số nguyên tố nhỏ, được chọn vì có khả năng phân tán tốt, giảm va chạm khi nhiều khóa gần giống nhau.
- Toán tử % SIZE đảm bảo chỉ số luôn nằm trong giới hạn hợp lệ của bảng băm (từ 0 đến 100).

Ví dụ với từ “cat”:

- 'c' = 99, 'a' = 97, 't' = 116
- Bước 1: hash = (0 * 31 + 99) % 101 = 99
- Bước 2: hash = (99 * 31 + 97) % 101 = 3265 % 101 = 32
- Bước 3: hash = (32 * 31 + 116) % 101 = 1118 % 101 = 7

Kết quả cuối cùng là 7, nghĩa là từ "cat" sẽ được lưu tại table[7].

Việc nhân 31 theo từng bước giúp các từ có thứ tự ký tự khác nhau (như "tap" và "pat") sinh ra chỉ số khác nhau, nhờ đó tăng tính phân tán và giảm nguy cơ va chạm.

2.4.2. Hàm AddWord – Thêm từ vào bảng băm (XỬ LÝ COLLISION)

Trước hết, nhóm giải thích tình huống collision.

Lý do rất đơn giản: tập hợp chuỗi có thể là vô hạn, nhưng bảng băm thì chỉ có một số lượng ô cố định (ví dụ: 101 ô). Dù hàm băm được thiết kế tốt đến đâu, vẫn luôn tồn tại xác suất nhiều khóa cùng rơi vào một chỉ số, nhất là khi dữ liệu tăng lên.

Ví dụ, "cat" và "tac" là hai từ hoàn toàn không liên quan, nhưng nếu chỉ cộng tổng mã ASCII rồi lấy dư với kích thước bảng, thì rất có thể cả hai cùng ra table[9]. Nếu không có cơ chế xử lý, từ sau có thể **ghi đè lên** từ trước, dẫn đến mất dữ liệu hoặc trả kết quả sai khi tra cứu.

Có nhiều cách phổ biến để xử lý collision, trong đó hai hướng chính là:

- **Open addressing**: tìm ô trống gần đó để lưu.
- **Chaining (liên kết)**: mỗi ô lưu một danh sách các phần tử có cùng chỉ số.

Trong bài toán này, nhóm lựa chọn phương pháp **chaining** vì nó phù hợp hơn với dữ liệu văn bản, dễ mở rộng, dễ kiểm soát và không cần dịch mảng hay tái cấu trúc khi thêm từ. Theo đó, mỗi ô trong mảng table[] sẽ lưu một chuỗi các phần tử dưới dạng danh sách liên kết đơn - mỗi node chứa một từ và thông tin nghĩa tương ứng, kèm theo con trỏ Next đến phần tử kế tiếp.

Chẳng hạn, nếu "cat", "tac" và "act" đều được ánh xạ vào table[9], thì cấu trúc sẽ trông như sau: ***table[9] = node("act") → node("tac") → node("cat") → null***

Cách này mang lại nhiều điểm mạnh thực tế:

- **Không cần dời mảng hay sắp xếp lại cấu trúc khi thêm từ mới** - chỉ cần móc nối thêm phần tử.
- **Thao tác thêm hoặc tìm kiếm đơn giản** - vì luôn làm việc trong một danh sách nhỏ tại chỉ số đã biết.
- **Dễ cài đặt thủ công và trực quan khi kiểm tra lỗi** - phù hợp với phạm vi đề tài sinh viên và cho phép kiểm thử dễ dàng.

Tuy nhiên, nếu không kiểm tra trùng khóa khi thêm từ, có thể dẫn đến dữ liệu dư thừa hoặc không đồng nhất. Cơ chế kiểm tra và móc nối sẽ được trình bày ở phần sau.n

Hàm `AddWord()` thực hiện việc thêm một từ mới vào bảng băm. Nếu từ đã tồn tại, hàm sẽ cập nhật thông tin thay vì thêm mới.

Quá trình hoạt động gồm hai bước:

- Tính vị trí lưu từ bằng hàm băm.

- Kiểm tra tại vị trí đó để thêm mới hoặc cập nhật.

```
public void AddWord(string key, WordEntry value)
{
    int index = HashFunction(key);
    HashNode current = table[index];

    while (current != null)
    {
        if (current.Key == key)
        {
            current.Value = value;
            return;
        }
        current = current.Next;
    }

    HashNode newNode = new HashNode(key, value);
    newNode.Next = table[index];
    table[index] = newNode;
}
```

(code trong class DictionaryManager)

Hàm AddWord(string key, WordEntry value) có chức năng **thêm một cặp (key, value)** vào bảng băm table. Nếu key đã tồn tại, nó sẽ **cập nhật giá trị** mới.

```
int index = HashFunction(key);
```

- Gọi hàm băm để tính chỉ số (vị trí trong mảng table) từ key.
- index là vị trí mà cặp key-value nên được lưu.

```
HashNode current = table[index];
```

Lấy node hiện tại tại vị trí đó (có thể là null hoặc một chuỗi các node nếu đã có collision).

```
while (current != null)
{
    if (current.Key == key)
    {
        current.Value = value;
        return;
    }
    current = current.Next;
}
```

- Duyệt qua danh sách liên kết tại table[index].
- Nếu tìm thấy key đã tồn tại:
- **Cập nhật Value** (thay thế nghĩa, phát âm,... mới).
- **Thoát khỏi hàm** (vì không cần thêm mới nữa).
- Nếu không khớp, tiếp tục sang node kế tiếp (current.Next).

```
HashNode newNode = new HashNode(key, value);
newNode.Next = table[index];
table[index] = newNode;
```

Ngược lại, nếu duyệt hết danh sách mà không thấy trùng khóa, tức là từ này hoàn toàn mới, chương trình sẽ tạo một HashNode mới. Cách nối rất đơn giản: gắn newNode.Next trỏ về node cũ (nếu có), rồi đẩy newNode lên đầu danh sách tại table[index].

Cách thêm vào đầu danh sách như vậy giúp tiết kiệm thời gian (không phải duyệt tới cuối chuỗi) và đơn giản hóa thao tác. Do bảng băm đã đảm bảo tính phân tán, độ dài danh sách tại mỗi ô thường không dài, nên giải pháp này ổn định và hiệu quả.

Trong quá trình thêm từ, nếu nhiều từ khóa khác nhau cùng cho ra một chỉ số băm (collision), hàm AddWord() sẽ xử lý bằng cách **nối từ mới vào đầu danh sách liên kết tại ô đó**. Cụ thể, từ mới sẽ trở thành node đầu tiên (newNode.Next = table[index]), còn các node cũ lùi xuống sau.

Cách này tránh ghi đè dữ liệu, giữ thao tác thêm nhanh và đơn giản. Đây là ưu điểm của kỹ thuật chaining – phù hợp với hệ thống từ điển quy mô nhỏ đến trung bình.

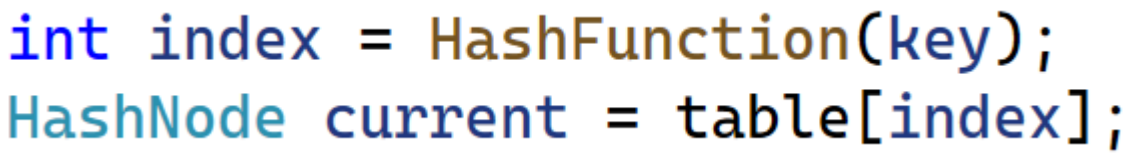
2.4.3. Tra nghĩa từ - Hàm GetDefinition()

Chức năng tra nghĩa là một trong những hành vi cơ bản và quan trọng nhất của ứng dụng từ điển. Khi người dùng nhập một từ cần tra, chương trình sẽ gọi GetDefinition(key) để tìm thông tin tương ứng trong bảng băm.

Đây là đoạn mã thể hiện rõ quá trình trên:

```
public WordEntry GetDefinition(string key)
{
    int index = HashFunction(key);
    HashNode current = table[index];
    while (current != null)
    {
        if (current.Key == key)
            return current.Value;
        current = current.Next;
    }
    return null;
}
```

(code trong lớp dictionarymanager)



Đầu tiên, chương trình dùng HashFunction để tính chỉ số băm của từ khóa. Chỉ số này xác định ô trong mảng table[] mà từ có thể đang được lưu.

```
while (current != null)
{
    if (current.Key == key)
        return current.Value;
    current = current.Next;
}
```

Sau đó, chương trình duyệt danh sách liên kết tại ô vừa tìm được. Với mỗi HashNode, nó so sánh Key hiện tại với key đầu vào. Nếu trùng, trả về Value, tức là đối tượng WordEntry chứa toàn bộ thông tin như nghĩa tiếng Việt, loại từ và phiên âm. Nếu không khớp, chương trình tiếp tục duyệt sang node kế tiếp.

```
return null;
```

Nếu duyệt hết danh sách mà không tìm thấy, chương trình trả về null, tức là từ không có trong từ điển.

Điểm mạnh của cách triển khai này là hiệu quả cao: nhờ dùng bảng băm kết hợp với liên kết đơn (chaining), chương trình không cần tìm toàn bộ từ điển, chỉ cần kiểm tra đúng chuỗi node tại một vị trí duy nhất. Điều này giúp việc tra cứu luôn nhanh chóng, ổn định, và giữ được hiệu suất ngay cả khi số lượng từ vựng ngày càng tăng.

2.4.4. Kiểm tra tồn tại - Hàm Contains()

Trong một số tình huống, người dùng không cần xem chi tiết nghĩa, mà chỉ đơn giản muốn biết: *Từ này đã có trong từ điển hay chưa?* Đây là một thao tác phổ biến trong các ứng dụng thực tế như kiểm tra trùng từ khi thêm mới, hoặc lọc dữ liệu nhập vào. Để xử lý yêu cầu này, nhóm cài đặt hàm Contains(), một phương thức trả về kiểu bool kiểm tra sự tồn tại của từ khóa trong bảng băm.

Thay vì viết lại toàn bộ logic dò tìm, hàm này tái sử dụng luôn GetDefinition(). Bản chất hai thao tác là tương tự: nếu từ đã tồn tại trong bảng băm, GetDefinition(key) sẽ trả về một đối tượng WordEntry hợp lệ; ngược lại nếu không tìm thấy, hàm sẽ trả về null.

```
public bool Contains(string key)
{
    return GetDefinition(key) != null;
}
```

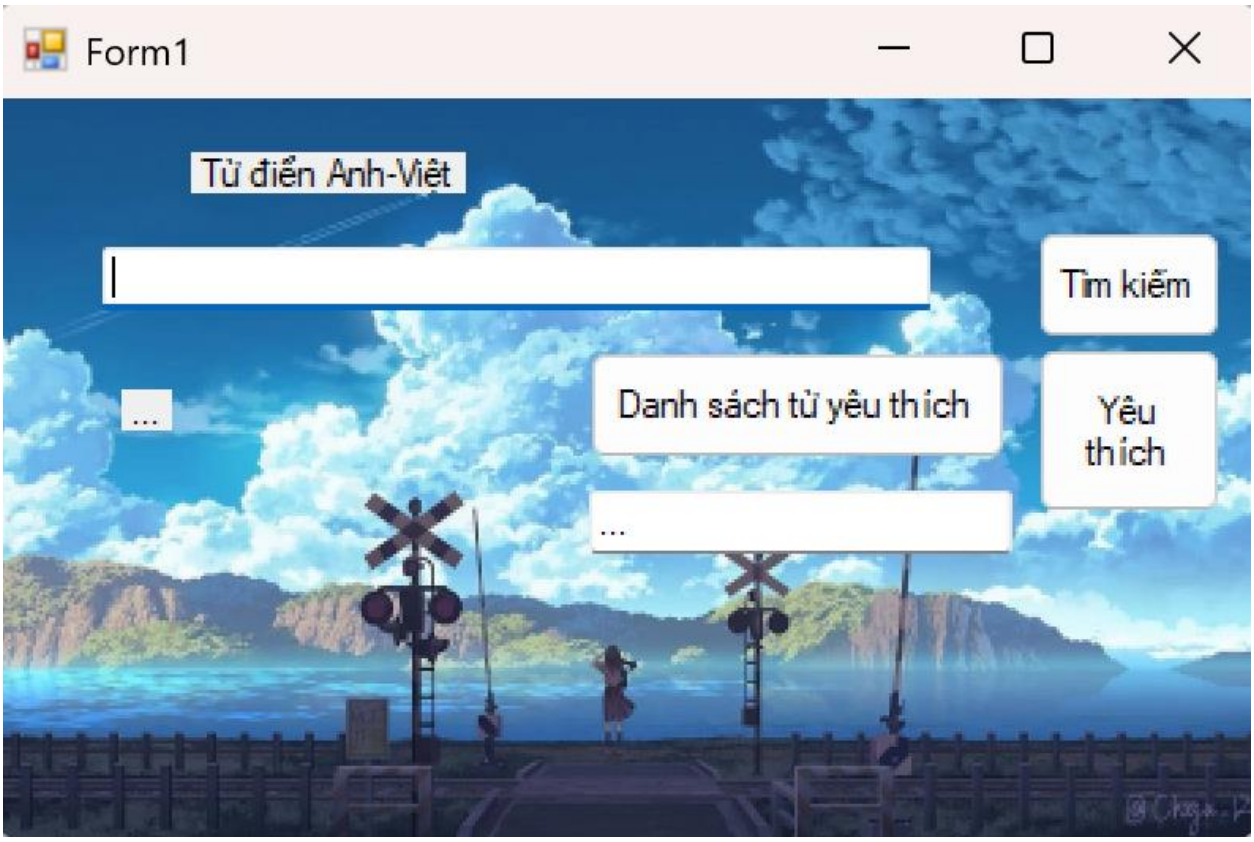
Cách viết này có hai ưu điểm rõ rệt. **Thứ nhất**, nó giúp tránh lặp lại logic tìm kiếm đã được kiểm chứng trong GetDefinition(), đảm bảo mọi thay đổi về sau (nếu có) chỉ cần sửa một chỗ duy nhất. **Thứ hai**, nó giúp giữ cho mã nguồn ngắn gọn, dễ đọc, mà vẫn đầy đủ chức năng - một nguyên tắc lập trình quan trọng khi xây dựng hệ thống lớn.

Chương 3. Thiết kế giao diện

Chương này trình bày giao diện người dùng (*UI*) của ứng dụng từ điển, được xây dựng bằng *Windows Forms* trong môi trường *Visual Studio code*.

Giao diện được thiết kế theo hướng tối giản, thân thiện với người dùng, đảm bảo hiển thị đầy đủ thông tin từ vựng (nghĩa, loại từ, phiên âm) và cung cấp các nút chức năng thao tác như tìm kiếm, thêm vào danh sách yêu thích, và xem lại từ đã lưu.

3.1. Giao diện menu chính



3.2. Chi tiết chức năng

Loại Control	Tên biến	Chức năng
Label	lblTitle	Hiển thị tiêu đề "Từ điển Anh-Việt"
TextBox	txtSearch	Nhập từ cần tìm kiếm
Button	btnSearch	Nút tìm kiếm nghĩa của từ
Label	lblResult	Hiển thị kết quả tra cứu
Button	btnAddToFavorites	Thêm từ vào danh sách yêu thích
Button	btnShowFavorites	Hiển thị danh sách từ yêu thích
TextBox	txtWord	Ô hiển thị danh sách từ yêu thích
Button	btnRemoveFromFavorites	Xóa từ khỏi danh sách yêu thích

Chương 4: Thảo luận & đánh giá

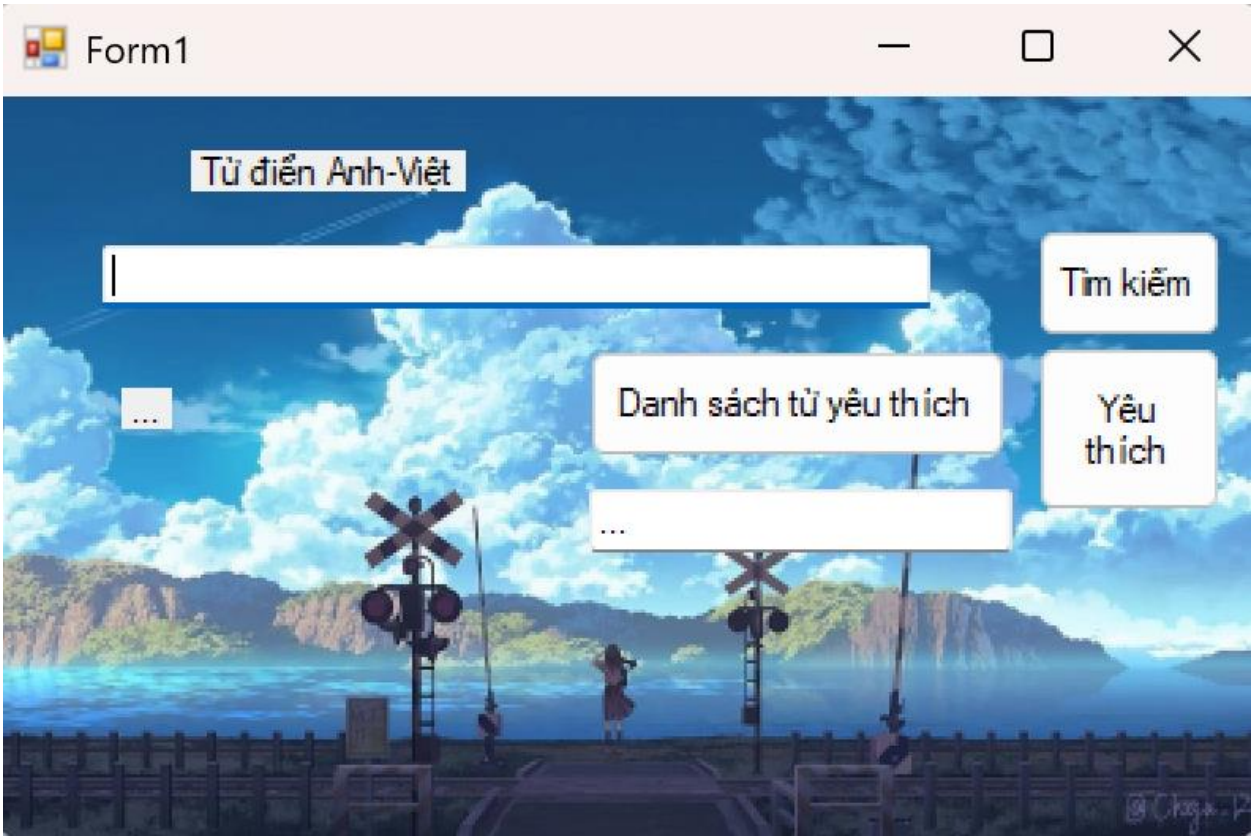
4.1. Các kết quả nhận được

Sau quá trình thực hiện, nhóm đã hoàn thành một ứng dụng từ điển Anh–Việt với các chức năng cơ bản:

- Tìm kiếm từ vựng, hiển thị nghĩa, loại từ và phiên âm.
- Thêm từ vào danh sách yêu thích và hiển thị lại khi cần.
- Giao diện có hình nền trực quan, dễ sử dụng.
- Dữ liệu được nạp từ file .txt, dễ cập nhật và mở rộng.

Ứng dụng hoạt động ổn định, xử lý các thao tác nhanh, đúng mục tiêu ban đầu của đề tài.

4.1.1. Giao diện khi khởi động chương trình



Giao diện khi khởi động chương trình

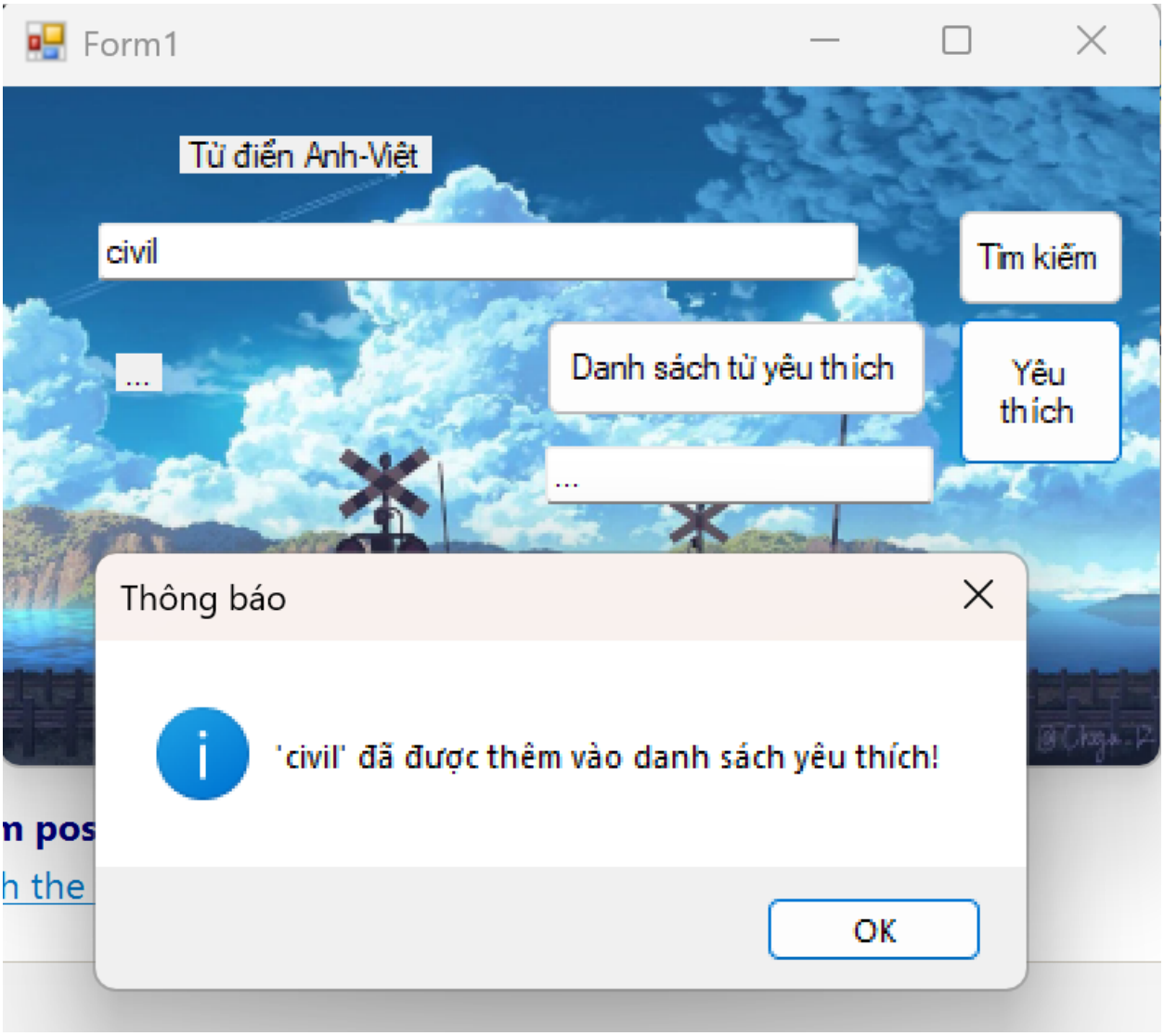
4.1.2. Tra nghĩa của từ vựng

Nhập từ vựng vào textbox “txtSearch” rồi ấn button “Tìm kiếm”.

Tra nghĩa của từ vựng

4.1.3. Thêm dữ liệu về từ vựng vào danh sách từ điển yêu thích

Nhập từ vựng vào textbox “txtSearch” rồi ấn button “yêu thích”.



Thêm từ vựng vào danh sách yêu thích

4.1.4. Xem danh sách từ điển yêu thích

Nhấn vào button "danh sách từ yêu thích" để xem danh sách yêu thích.

4.2. Khó khăn gặp phải

Trong quá trình làm việc, nhóm gặp một số khó khăn đáng kể như:

- Khi phóng to, thu nhỏ màn hình Form thì các control bên trong không thay đổi size (kích thước các control không phù hợp khi Form thay đổi).
- Khi nhấn vào các button thì chưa nổi bật (nghĩa là khi ấn một button nào đó thì nên có một dấu hiệu để cho người sử dụng biết rằng mình đang sử dụng chức năng của button đó).
- Chưa có những ví dụ về cách sử dụng của từ vựng.
- Chưa có khả năng phát âm từ vựng.

4.3. Hướng phát triển

Trong tương lai, nhóm định hướng mở rộng ứng dụng theo các hướng sau:

- Nghiên cứu sâu hơn về cách xử lý **thuộc tính của control** (Label, Button, TextBox,...) và các **sự kiện (Event)** liên quan, nhằm **tự động điều chỉnh kích thước giao diện khi Form thay đổi kích thước**, khắc phục hạn chế hiện tại về tính linh hoạt UI.
- Tìm hiểu cách thay đổi **giao diện động cho nút bấm**, cụ thể là làm cho Button đổi màu khi được nhấn để tăng tính trực quan và cải thiện trải nghiệm người dùng.

- Nhóm cần thêm thời gian để **thu thập và chọn lọc các nhóm từ đồng nghĩa – trái nghĩa**, giúp nâng cao tính ứng dụng của từ điển và hỗ trợ người học mở rộng vốn từ.
- Có thể mở rộng hệ thống bằng cách:
  + Tạo thêm một **từ điển phụ** chứa những từ nằm trong danh sách từ chính → dùng như **nguồn tham khảo nội bộ**.
  + Hoặc chèn thêm **ví dụ minh họa ngắn** ngay sau từng mục từ trong file anhviet.txt để người dùng dễ hình dung cách dùng từ.
- Cần thêm thời gian để **thu thập phát âm chuẩn** cho từng từ vựng và tích hợp các đoạn audio tương ứng vào ứng dụng, giúp người học luyện nghe và phát âm chính xác hơn.

KẾT LUẬN

Thông qua việc thực hiện đồ án này, nhóm đã có cơ hội tiếp cận và áp dụng kiến thức về cấu trúc dữ liệu và giải thuật một cách cụ thể, rõ ràng hơn so với những gì từng học lý thuyết trước đó.

Nhóm đã hiểu rõ hơn về cách lựa chọn cấu trúc dữ liệu phù hợp với từng chức năng, và thấy được tầm quan trọng của việc tổ chức code sao cho dễ hiểu, dễ mở rộng. Dù vẫn còn nhiều thứ chưa làm được như mong muốn, nhưng các thành viên đã cố hết sức để có thể hoàn thành bài tập một cách tốt nhất trong khả năng của mình.

Qua đồ án này, nhóm cũng nhận ra rằng việc viết ra một chương trình không chỉ là vấn đề về cú pháp, mà còn là quá trình tư duy, tổ chức và thử nghiệm. Việc được thử sức qua đồ án này rất quan trọng và ý nghĩa trên con đường học tập, giúp chúng em có thể biến những lý thuyết suông thành những thứ có thể áp dụng vào thực tiễn. Bài làm của nhóm sẽ không chỉ là một kết quả nộp môn, mà còn là một bước khởi đầu nho nhỏ giúp chúng em tự tin hơn khi tiếp cận những đề tài kỹ thuật trong tương lai.

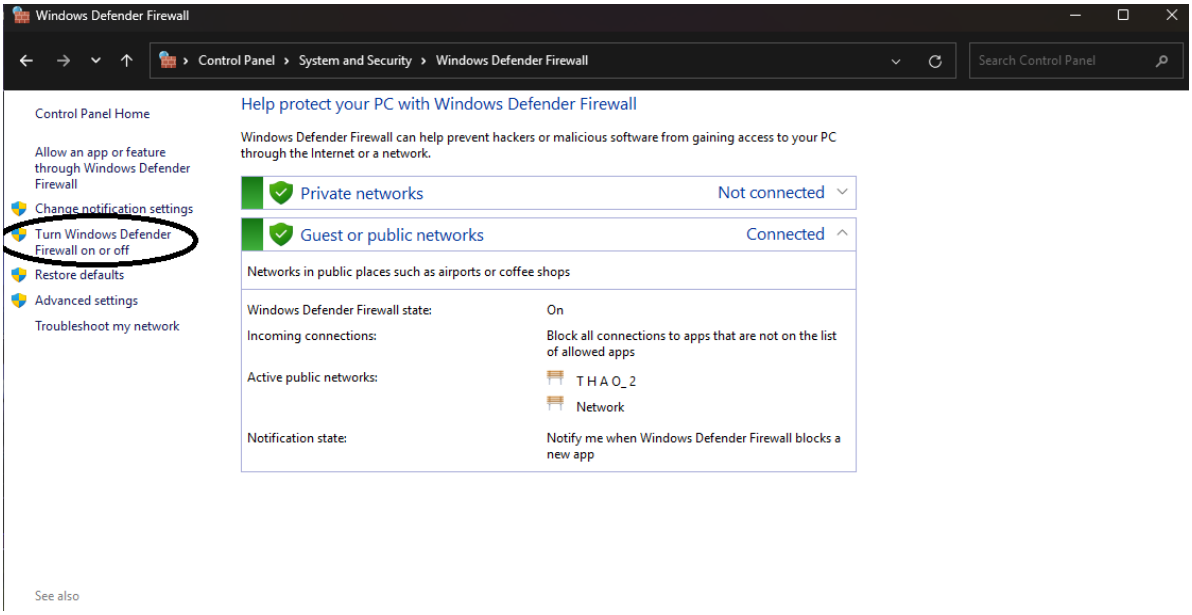
PHỤ LỤC

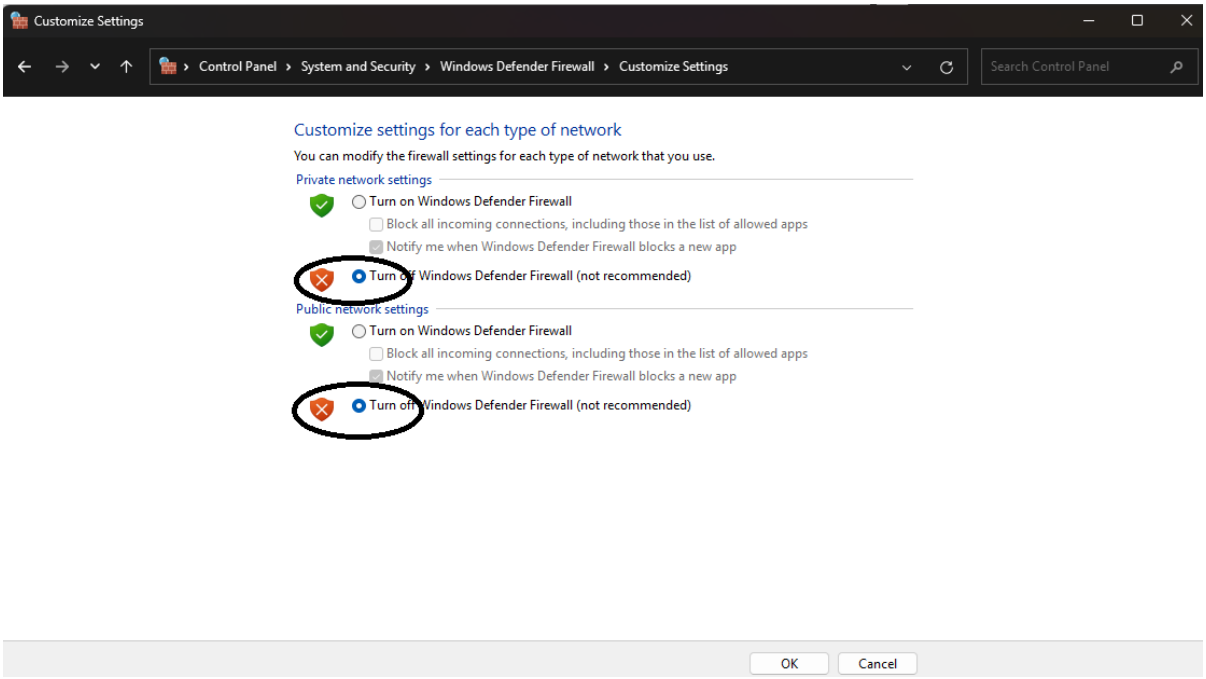
Toàn bộ mã nguồn trên GitHub: PhuongT-T/App-T-i-n-Anh_Vi-t at main

I. Hướng dẫn cài đặt:

Bước 1: Tắt FireWall (tường lửa) Vì sẽ có vài file trên GitHub tải về sẽ bị chặn nên cần phải tắt tường lửa.

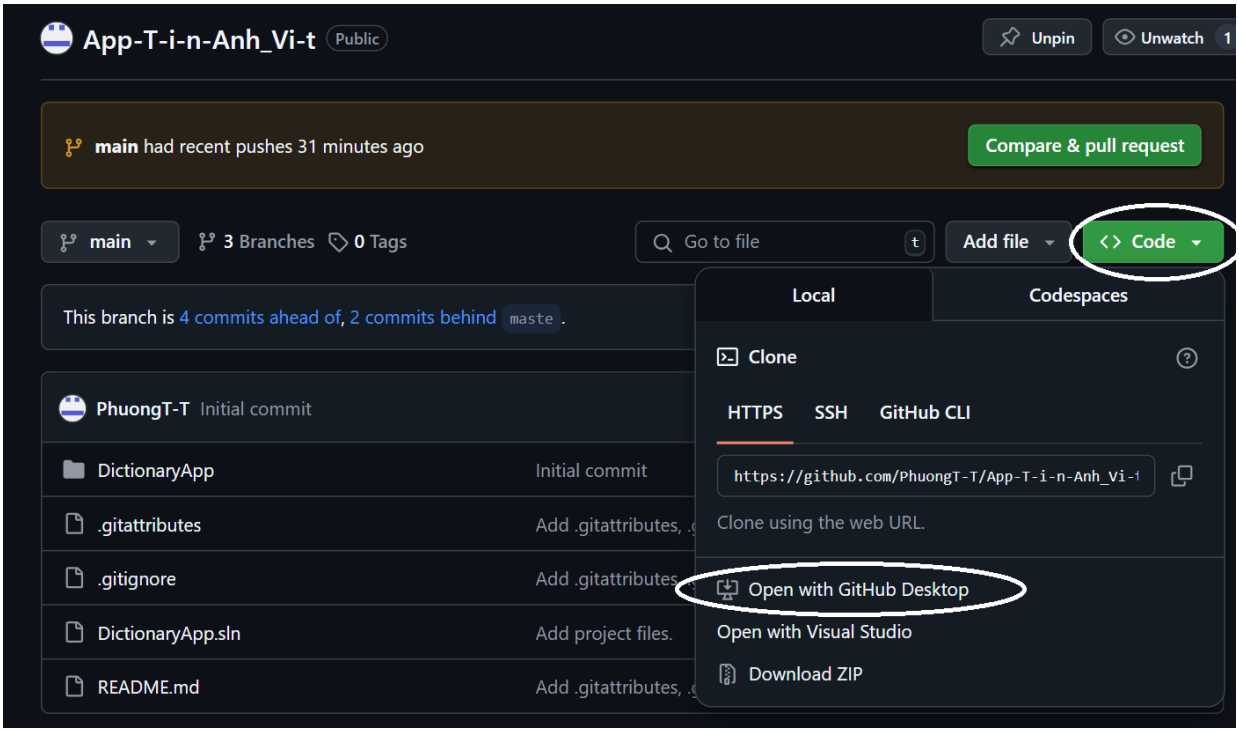
=> Mở Windows tìm Windows Defender Firewall, Nhấp vào Turn Windows Defender Firewall on or off (Bật hoặc tắt tường lửa Windows). – Chọn Turn off Windows Defender Firewall cho cả hai mục Private network settings và Public network settings.



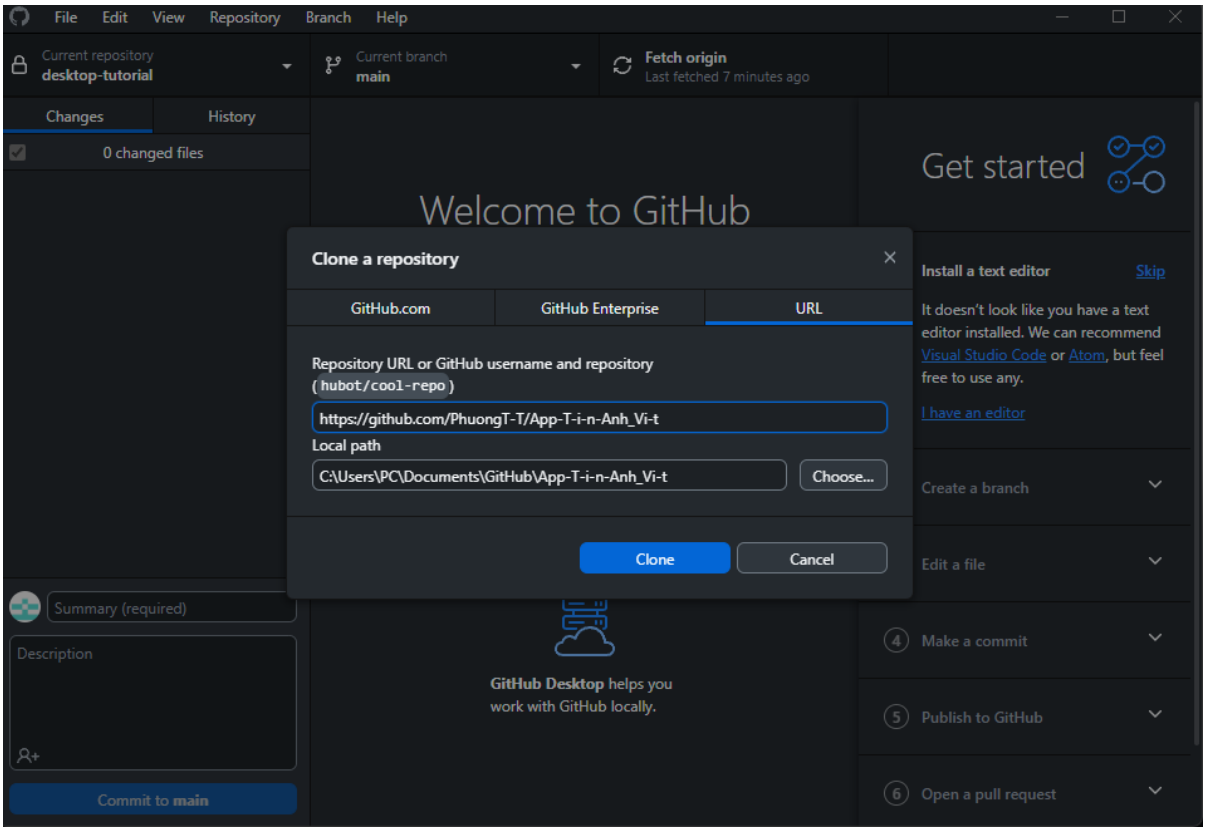


Bước 2: Truy cập đường dẫn: [PhuongT-T/App-T-i-n-Anh_Vi-t at main]

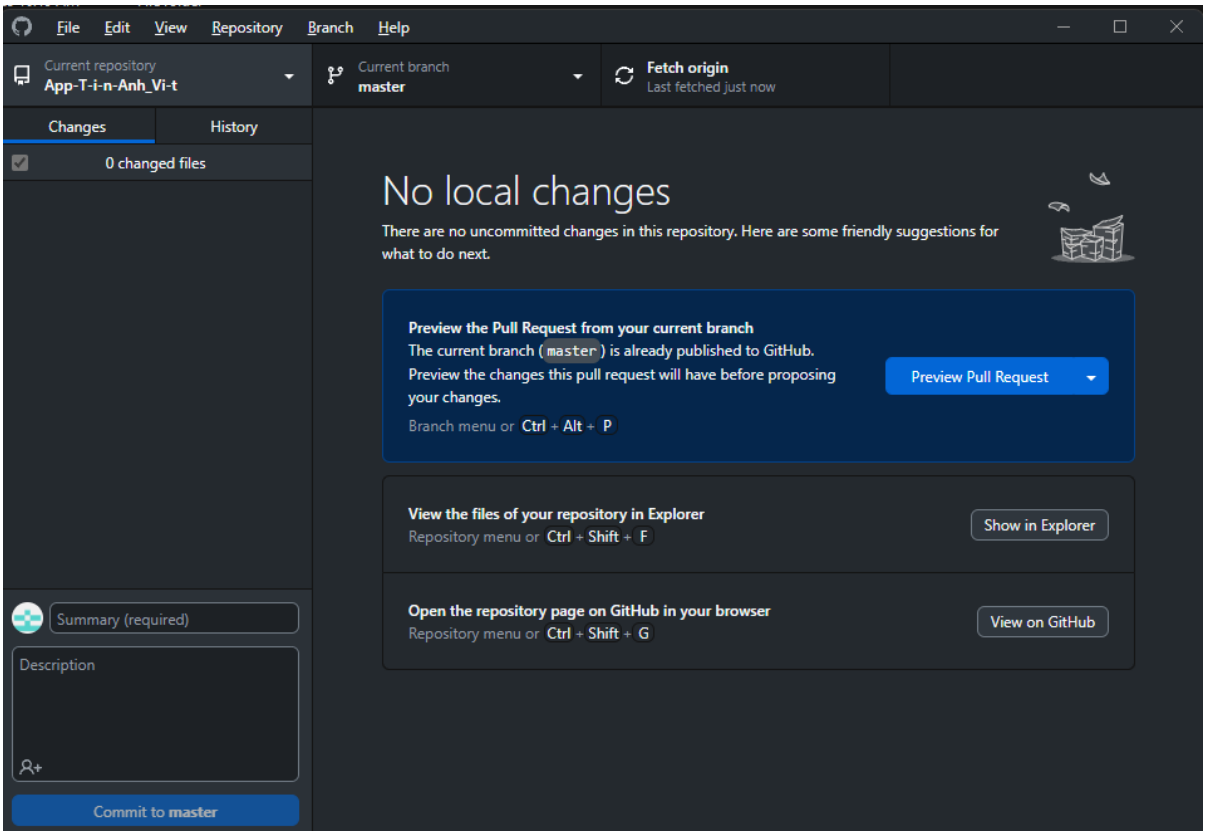
Bước 3: Click vào “Code” sau đó click vào “Open with GitHub Desktop”
(Mọi người nên tải GitHub Desktop và có tài khoản GitHub. Link tải GitHub Desktop: https://desktop.github.com/).



Bước 4: Sau khi click vào trang sẽ đưa vào GitHub Desktop, sau đó click vào “Clone”.



Bước 5: Cuối cùng click vào ”Show in Explorer” sẽ hiện file của nhóm ra.



Lưu ý: Vì vì code trên có địa chỉ file từ điển nằm trên máy tính bạn viết code nên người dùng muốn sử dụng app phải tạo 1 file từ điển trên Notepad và địa chỉ file đó thay thế vào đoạn code sau ở class form1:

```
private readonly string dictionaryPath = @"C:\Users\Phuong\Documents\anhviet.txt";
```

II. Hướng dẫn sử dụng ứng dụng

1. Khởi chạy ứng dụng

Mở file thực thi hoặc chạy từ Visual Studio thông qua Program.cs.

2. Tra từ

Nhập từ tiếng Anh vào ô tìm kiếm.

Bấm “Tìm kiếm”.

Nghĩa tiếng Việt của từ sẽ hiển thị ở phần kết quả.

3. **Xử lý không tìm thấy**

Nếu từ không có trong cơ sở dữ liệu, ứng dụng sẽ thông báo “Không tìm thấy từ”.

4. **Thoát ứng dụng**

Đóng cửa sổ chính (Form1) để kết thúc chương trình.

Bảng phân công công việc

Thành viên	**Nhiệm vụ**
Trương Thị Kiều Phương (Leader)	Định hướng bài làm Code chính
Đoàn Thị Mai Linh	Viết báo cáo Code phụ
Bùi Thị Thùy Linh	Viết báo cáo Slide
Bùi Anh Minh	Viết báo cáo Slide

TÀI LIỆU THAM KHẢO

1. Bài giảng của thầy Đặng Ngọc Hoàng Thành

2. Howkteam.com

3. Github.com