

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA CÔNG NGHỆ PHẦN MỀM



ĐỒ ÁN MÔN HỌC
ADVANCED DATA STRUCTURE & ALGORITHMS
ỨNG DỤNG TRỰC QUAN ĐỐI SÁNH CHUỖI

Giảng viên hướng dẫn : ThS. Nguyễn Thanh Sơn

Sinh viên thực hiện 1 : Hồ Nguyễn Tài Lợi

Mã sinh viên 1 : 23520869

Sinh viên thực hiện 2 : Nguyễn Trung Kiên

Mã sinh viên 2 : 23520802

Lớp : CS523.P21

Bộ môn : CTDL>NC

TP. HỒ CHÍ MINH, THÁNG 3 NĂM 2025

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Trường Đại học Công nghệ Thông tin, Đại học Quốc gia Thành phố Hồ Chí Minh đã đưa môn học Cấu trúc dữ liệu và giải thuật nâng cao vào chương trình giảng dạy. Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn – ThS. Nguyễn Thanh Sơn đã truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Môn học này là môn học thú vị và vô cùng bổ ích. Tuy nhiên do vốn kiến thức còn nhiều hạn chế và khả năng tiếp thu kiến thức thực tế còn nhiều bỡ ngỡ, chúng em tuy đã cố gắng hết sức nhưng không thể không tránh khỏi những sai sót trong đề án này, kính mong cô xem xét và góp ý để bài đề án của chúng em được hoàn thiện và tốt hơn. Từ đó rút kinh nghiệm cho những bài đề án sau này.

Lời cuối cùng, chúng em xin chân thành cảm ơn! Hơn hết, chúng em xin gửi lời cảm ơn chân thành nhất đến từng thành viên của nhóm – những cá thể xa lạ chưa từng biết đến nhau trước đó đã cùng góp một phần công sức không nhỏ vào đề án môn học này. Nhưng sau tất cả, chúng em nhận thức được rằng với lượng kiến thức và kinh nghiệm ít ỏi của bản thân, chắc chắn bài luận sẽ khó tránh khỏi thiếu sót.

Kính mong quý thầy cô thông cảm và góp ý để chúng em ngày càng hoàn thiện hơn.

Chúng em xin chân thành cảm ơn!

Tp. HCM, ngày ... tháng ... năm ...

Nhóm sinh viên thực hiện

Nguyễn Trung Kiên – Hồ Nguyễn Tài Lợi

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp.HCM, ngày ... tháng ... năm ...

GVHD

Mục lục

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI	1
1.1 Tên đề tài.....	1
1.2 Mô tả đề tài	1
1.3 Lý do chọn đề tài	1
1.4 Các chức năng chính của đề tài	2
1.5 Công nghệ sử dụng	2
1.6 Môi trường lập trình	2
1.7 Công cụ hỗ trợ	2
CHƯƠNG 2: GIỚI THIỆU CÔNG NGHỆ	1
2.1 Lịch sử công nghệ.....	1
2.2 Chức năng và nhiệm vụ công nghệ	1
2.3 Áp dụng công nghệ trong đề tài.....	2
CHƯƠNG 3: CƠ SỞ LÝ THUYẾT.....	1
3.1 Thuật toán Brute Force (hay thuật toán Naive)	1
3.2 Thuật toán KMP - Knuth-Morris-Pratt.....	2
3.3 Thuật toán Boyer-Moore.....	5
3.4 Thuật toán Rabin Karp	7
KẾT LUẬN	1
TÀI LIỆU THAM KHẢO	1

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1 Tên đề tài

AlgoMatching Application (Ứng dụng trực quan thuật toán đối sánh chuỗi)

1.2 Mô tả đề tài

- AlgoMatch App biến những dòng code khô khan thành hình ảnh sinh động, giúp bạn hiểu rõ từng bước thực thi thuật toán một cách trực quan và dễ nhớ.
- Không chỉ quan sát, bạn có thể tương tác, điều chỉnh tốc độ chạy thuật toán, nhập dữ liệu tùy chỉnh và thấy ngay kết quả thay đổi.
- Từ sinh viên, lập trình viên đến người mới bắt đầu, ai cũng có thể tiếp cận và nắm vững các thuật toán phức tạp một cách trực quan và thú vị.
- AlgoMatch không chỉ giúp học mà còn hỗ trợ giảng dạy, nghiên cứu với giao diện thân thiện, giúp truyền đạt thuật toán một cách sinh động và hiệu quả.

1.3 Lý do chọn đề tài

- Thuật toán là nền tảng của lập trình, nhưng nhiều người gặp khó khăn khi tiếp cận. AlgoMatch giúp trực quan hóa từng bước, biến những khái niệm trừu tượng thành hình ảnh sinh động, dễ hiểu.
- Thay vì chỉ đọc code trên sách vở, người học có thể tương tác trực tiếp, nhập dữ liệu, điều chỉnh tốc độ thực thi, từ đó hiểu sâu sắc cách thuật toán hoạt động trong thực tế.
- AlgoMatch hỗ trợ giáo viên, giảng viên mô phỏng thuật toán một cách sinh động, giúp học sinh – sinh viên tiếp thu kiến thức nhanh hơn và nhớ lâu hơn.
- Dù là sinh viên CNTT, lập trình viên hay người mới học lập trình, bất kỳ ai cũng có thể sử dụng AlgoMatch để cải thiện kỹ năng thuật toán một cách hiệu quả và thú vị.

1.4 Các chức năng chính của đề tài

- ❖ Render ra giao diện các chuỗi dữ liệu
- ❖ Hiển thị các bước của thuật toán
- ❖ Hiệu chỉnh thời gian thực hiện các bước thuật toán
- ❖ Tạo chuỗi ngẫu nhiên
- ❖ Hướng dẫn code các thuật toán (Java, C++, C#, Python3)


1.5 Công nghệ sử dụng

- ❖ .NET Framework
- ❖ WPF

1.6 Môi trường lập trình

 Visual Studio 2022

1.7 Công cụ hỗ trợ

 Github, Google

CHƯƠNG 2: GIỚI THIỆU CÔNG NGHỆ

2.1 Lịch sử công nghệ

❖ .NET Framework

Ra đời vào năm 2002 do Microsoft phát triển, nhằm cung cấp một nền tảng mạnh mẽ để xây dựng ứng dụng desktop, web, và dịch vụ. Ban đầu, nó tập trung hỗ trợ các công nghệ của Windows và nhanh chóng phát triển qua các phiên bản với nhiều cải tiến như Generics, LINQ, WPF, và WCF. Đến năm 2016, Microsoft giới thiệu .NET Core, một nền tảng mã nguồn mở, đa nền tảng, đánh dấu bước chuyển từ .NET Framework sang công nghệ hiện đại hơn.

❖ WPF - Windows Presentation Foundation

Ra mắt năm 2006, là công nghệ giao diện người dùng mạnh mẽ trong .NET Framework 3.0. Với ngôn ngữ XAML, WPF mang đến khả năng thiết kế giao diện đồ họa tiên tiến, hỗ trợ đồ họa vector, video, và hình ảnh. Được Microsoft phát triển để xây dựng ứng dụng Windows hiện đại, WPF vẫn được sử dụng rộng rãi dù các công nghệ như .NET Core đã xuất hiện.

2.2 Chức năng và nhiệm vụ công nghệ

❖ .NET Framework

▪ Chức năng:

- Cung cấp nền tảng phát triển ứng dụng trên Windows như desktop, web, và dịch vụ.
- Bao gồm thư viện lập trình phong phú (Base Class Library - BCL) và công cụ Runtime (Common Language Runtime - CLR) để quản lý mã nguồn và tối ưu hóa hiệu suất.
- Hỗ trợ đa ngôn ngữ lập trình như C#, VB.NET, F#.

▪ Nhiệm vụ:

- Đóng vai trò là môi trường chạy ứng dụng .NET, quản lý bộ nhớ, bảo mật, và xử lý lỗi.
- Tích hợp các công nghệ phát triển giao diện (WPF, Windows Forms) và dịch vụ (WCF, ASP.NET).

❖ **Windows Presentation Foundation (WPF)**

▪ **Chức năng:**

- Là công nghệ giao diện người dùng (UI) cho ứng dụng Windows, hỗ trợ thiết kế giao diện hiện đại, đồ họa 2D/3D và xử lý đa phương tiện.
- Sử dụng XAML (Extensible Application Markup Language) để thiết kế giao diện trực quan, tách biệt với mã logic (code-behind).
- Hỗ trợ tính năng binding dữ liệu, animation, và styles để tạo giao diện tương tác phong phú.

▪ **Nhiệm vụ:**

- Xây dựng giao diện người dùng cho các ứng dụng desktop Windows, hỗ trợ tương tác với người dùng một cách hiệu quả.
- Cung cấp các công cụ thiết kế UI hiện đại, tích hợp chặt chẽ với các công nghệ .NET như Entity Framework.

2.3 Áp dụng công nghệ trong đề tài

❖ **Áp dụng .NET Framework**

▪ **Lý do áp dụng:**

- Ứng dụng cần một nền tảng ổn định và đáng tin cậy để xử lý toàn bộ logic và hoạt động mượt mà trên hệ điều hành Windows.
- Tích hợp tốt với các công nghệ khác trong .NET như WPF và Entity Framework.

▪ **Cách triển khai:**

- .NET Framework được sử dụng làm nền tảng chính cho ứng dụng.
- Hỗ trợ các thành phần cốt lõi như quản lý sự kiện, xử lý nghiệp vụ, và kết nối với cơ sở dữ liệu.
- Các thư viện tích hợp trong .NET Framework giúp tối ưu hóa hiệu suất và bảo mật của ứng dụng.

❖ **Áp dụng WPF**

▪ **Lý do áp dụng:**

- WPF cho phép thiết kế giao diện người dùng hiện đại, thân thiện, và tương tác cao.
- Hỗ trợ các tính năng mạnh mẽ như binding dữ liệu, template, và biểu đồ trực quan.

▪ **Cách triển khai:**

- WPF được sử dụng để xây dựng giao diện chính của ứng dụng, bao gồm bảng điều khiển, các bảng nhập liệu, và biểu đồ phân tích.
- Binding dữ liệu từ Entity Framework giúp hiển thị thông tin thu nhập, chi tiêu một cách tự động và linh hoạt
- Sử dụng các thư viện bổ sung như LiveCharts để tạo biểu đồ tài chính, hỗ trợ người dùng theo dõi chi tiêu hiệu quả.

CHƯƠNG 3: CƠ SỞ LÝ THUYẾT

Thuật toán string-matching (hay còn gọi là thuật toán đối sánh chuỗi) là một lĩnh vực quan trọng trong khoa học máy tính, xử lý văn bản và xử lý ngôn ngữ tự nhiên.

3.1 Thuật toán Brute Force (hay thuật toán Naive)

- **Mô tả**

Thuật toán **Naive String Matching** (so khớp chuỗi ngây thơ) là một thuật toán đơn giản để tìm kiếm một chuỗi con (**pattern**) trong một chuỗi lớn hơn (**text**). Đây là cách tiếp cận **cơ bản nhất** để giải quyết bài toán tìm kiếm chuỗi con.

- **Nguyên lý hoạt động**

Thuật toán duyệt qua từng vị trí trong **text**, kiểm tra xem **pattern** có xuất hiện ở vị trí đó không bằng cách so sánh từng ký tự một. So sánh pattern với mọi vị trí khả thi trong text (vét cạn)

- **Quy trình thực hiện**

B1: Duyệt qua **text** từ vị trí 0 đến $n-m$ (trong đó n là độ dài của **text**, m là độ dài của **pattern**).

B2: Tại mỗi vị trí i , so sánh lần lượt từng ký tự của **pattern** với các ký tự tương ứng trong **text**.

B3: Nếu tất cả các ký tự khớp, lưu lại vị trí i .

B4: Tiếp tục dịch chuyển sang vị trí tiếp theo cho đến khi duyệt hết **text**.

- **Cài đặt thuật toán**

```
using System;

class PatternSearch
{
    static void Search(string pat, string txt)
    {
        int M = pat.Length;
        int N = txt.Length;
```

```

// A loop to slide pat[] one by one
for (int i = 0; i <= N - M; i++)
{
    int j;

    // For current index i, check for pattern match
    for (j = 0; j < M; j++)
    {
        if (txt[i + j] != pat[j])
        {
            break;
        }
    }

    // If pattern matches at index i
    if (j == M)
    {
        Console.WriteLine($"Pattern found at index {i}");
    }
}
}

```

3.2 Thuật toán KMP - Knuth-Morris-Pratt

- **Mô tả**

Thuật toán **Knuth-Morris-Pratt (KMP)** là một thuật toán tìm kiếm chuỗi con (pattern) trong một chuỗi lớn hơn (text) hiệu quả hơn so với phương pháp Naive. Nó tránh việc kiểm tra lại các ký tự đã so sánh trước đó bằng cách sử dụng bảng tiền xử lý **LPS (Longest Prefix Suffix)**.

- **Nguyên lý hoạt động**

- Tận dụng thông tin về sự khớp một phần đã biết để tránh so sánh lại
- Xây dựng một bảng tiền tố (failure function/LPS array) để biết cần dịch chuyển bao nhiêu khi gặp không khớp

- **Quy trình thực hiện**

Bước 1: Xây dựng bảng lps (longest prefix suffix)

1. Khởi tạo mảng lps có kích thước bằng độ dài pattern

2. Gán $lps[0] = 0$
3. Sử dụng hai con trỏ $i = 1$ và $j = 0$
4. Nếu $pattern[i] = pattern[j]$, gán $lps[i] = j+1$, tăng i và j
5. Nếu $pattern[i] \neq pattern[j]$:
 - Nếu $j > 0$, gán $j = lps[j-1]$
 - Nếu $j = 0$, gán $lps[i] = 0$ và tăng i
6. Lặp lại bước 4-5 cho đến khi i bằng độ dài pattern

Bước 2: Tìm kiếm chuỗi trong text

1. Khởi tạo hai con trỏ $i = 0$ (cho text) và $j = 0$ (cho pattern)
2. Nếu $text[i] = pattern[j]$, tăng cả i và j
3. Nếu $j = \text{độ dài pattern}$, ghi nhận kết quả tìm thấy tại vị trí $(i-j)$
4. Nếu $i < \text{độ dài text}$ và $text[i] \neq pattern[j]$:
 - Nếu $j > 0$, gán $j = lps[j-1]$
 - Nếu $j = 0$, tăng i
5. Lặp lại bước 2-4 cho đến khi i bằng độ dài text

• Cài đặt thuật toán

```
// C# program to search the pattern in given text using
// KMP Algorithm

using System;
using System.Collections.Generic;

class GfG {
    static void ConstructLps(string pat, int[] lps) {
        // len stores the length of longest prefix which
        // is also a suffix for the previous index
        int len = 0;

        // lps[0] is always 0
        lps[0] = 0;

        int i = 1;
        while (i < pat.Length) {
```

```

        // If characters match, increment the size of lps
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        }

        // If there is a mismatch
        else {
            if (len != 0) {
                len = lps[len - 1];
            }
            else {

                // If no matching prefix found, set lps[i] to 0
                lps[i] = 0;
                i++;
            }
        }
    }
}

```

```

static List<int> search(string pat, string txt) {
    int n = txt.Length;
    int m = pat.Length;

    int[] lps = new int[m];
    List<int> res = new List<int>();

    ConstructLps(pat, lps);

    int i = 0;
    int j = 0;

    while (i < n) {

        if (txt[i] == pat[j]) {
            i++;
            j++;

            if (j == m) {
                //find match
                res.Add(i - j);
                j = lps[j - 1];
            }
        }

        // If there is a mismatch
        else {

            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
    }
}

```

```
}  
    return res;  
}  
  
}
```

3.3 Thuật toán Boyer-Moore

- **Mô tả**

Thuật toán **Boyer-Moore** là một trong những thuật toán tìm kiếm chuỗi con nhanh nhất, đặc biệt hiệu quả khi tìm kiếm trong văn bản dài. Nó hoạt động bằng cách duyệt chuỗi từ **phải sang trái** (thay vì từ trái sang phải như KMP) và sử dụng hai quy tắc dịch chuyển để bỏ qua nhiều ký tự nhất có thể.

- **Nguyên lý hoạt động**

So sánh pattern từ phải sang trái

Sử dụng hai quy tắc dịch chuyển:

1. Bad Character Rule
2. Good Suffix Rule

Có thể bỏ qua nhiều ký tự, đạt hiệu suất tốt nhất trong các thuật toán

- **Quy trình thực hiện**

Bước 1: Xây dựng bảng bad character

1. Khởi tạo bảng bad character với giá trị mặc định là -1
2. Duyệt qua pattern và lưu vị trí xuất hiện cuối cùng của mỗi ký tự

Bước 2: Xây dựng bảng good suffix (tùy chọn)

1. Tính toán bảng shift cho good suffix
2. Tính toán bảng prefix

Bước 3: Tìm kiếm chuỗi trong text

1. Đặt $s = 0$ (vị trí bắt đầu so sánh trong text)
 2. Bắt đầu so sánh từ cuối pattern ($j = \text{độ dài pattern} - 1$)
 3. Nếu $\text{text}[s+j] = \text{pattern}[j]$, giảm j và tiếp tục so sánh
 4. Nếu tất cả ký tự khớp ($j < 0$), ghi nhận kết quả tìm thấy tại vị trí s
 5. Nếu có ký tự không khớp:
 - Tính khoảng cách dịch chuyển dựa trên bad character
 - Nếu có good suffix, tính khoảng cách dịch dựa trên good suffix
 - Chọn khoảng cách dịch lớn nhất giữa hai giá trị
 6. Dịch chuyển s theo khoảng cách tính được
 7. Lặp lại bước 2-6 cho đến khi s vượt quá giới hạn text
- **Cài đặt thuật toán**

```
using System;
public class AWQ {

    static int NO_OF_CHARS = 256;

    static int max(int a, int b) { return (a > b) ? a : b; }

    static void badCharHeuristic(char[] str, int size,
                                int[] badchar)
    {
        int i;

        for (i = 0; i < NO_OF_CHARS; i++)
            badchar[i] = -1;

        for (i = 0; i < size; i++)
            badchar[(int)str[i]] = i;
    }

    static void search(char[] txt, char[] pat)
    {
        int m = pat.Length;
        int n = txt.Length;

        int[] badchar = new int[NO_OF_CHARS];
```



```

badCharHeuristic(pat, m, badchar);

int s = 0;
while (s <= (n - m)) {
    int j = m - 1;

    while (j >= 0 && pat[j] == txt[s + j])
        j--;
    if (j < 0) {
        //find match

        s += (s + m < n) ? m - badchar[txt[s + m]] : 1;
    }

    else
        s += max(1, j - badchar[txt[s + j]]);
    }
}

```

3.4 Thuật toán Rabin Karp

- **Mô tả**

Thuật toán **Rabin-Karp** là một phương pháp tìm kiếm chuỗi con trong chuỗi lớn bằng cách sử dụng **hàm băm** để so sánh nhanh thay vì so khớp từng ký tự một.

- **Nguyên lý hoạt động**
- Sử dụng hàm băm để so sánh chuỗi
- Tính toán giá trị băm của pattern và các chuỗi con có cùng độ dài trong text
- Chỉ so sánh trực tiếp khi giá trị băm trùng khớp
- **Quy trình thực hiện**

Bước 1: Chọn tham số

1. Chọn cơ số d (thường là số ký tự trong bảng mã)
2. Chọn số nguyên tố q để thực hiện phép mod

Bước 2: Tính giá trị hash

1. Tính giá trị hash cho pattern
2. Tính giá trị hash cho cửa sổ đầu tiên của text (có độ dài bằng pattern)

Bước 3: Tìm kiếm chuỗi trong text

1. Duyệt qua text từ đầu đến cuối:
2. So sánh hash của cửa sổ hiện tại với hash của pattern
3. Nếu hai giá trị hash bằng nhau, kiểm tra từng ký tự để xác nhận
4. Nếu tìm thấy, ghi nhận kết quả
5. Cập nhật giá trị hash cho cửa sổ tiếp theo bằng công thức rolling hash
6. Lặp lại bước 2-5 cho đến khi duyệt hết text

• Cài đặt thuật toán

```
using System;
public class GFG {
    public readonly static int d = 256;

    static void search(String pat, String txt, int q)
    {
        int M = pat.Length;
        int N = txt.Length;
        int i, j;
        int p = 0;
        int t = 0;
        int h = 1;

        // The value of h would be "pow(d, M-1)%q"
        for (i = 0; i < M - 1; i++)
            h = (h * d) % q;
        for (i = 0; i < M; i++) {
            p = (d * p + pat[i]) % q;
            t = (d * t + txt[i]) % q;
        }

        for (i = 0; i <= N - M; i++) {
            if (p == t) {
                for (j = 0; j < M; j++) {
                    if (txt[i + j] != pat[j])
                        break;
                }

                if (j == M)
                    //find match
            }
        }
    }
}
```

```
if (i < N - M) {  
    t = (d * (t - txt[i] * h) + txt[i + M]) % q;  
  
    if (t < 0)  
        t = (t + q);  
}  
}  
}
```



KẾT LUẬN

1. Các kết quả đạt được của đồ án

- Sản phẩm ứng dụng đầu tay riêng của nhóm.
- Tiếp cận và hiểu về mô hình của một ứng dụng đơn giản
- Khả năng hợp tác làm việc nhóm cùng nhau
- Khám phá và tìm hiểu nhiều các thuật toán về việc tìm kiếm chuỗi
- Ứng dụng hỗ trợ việc hình dung trực quan các thuật toán đối sánh chuỗi

2. Ưu điểm của đồ án

- Giao diện ứng dụng đơn giản dễ dàng tiếp cận, tạo cơ hội cho người dùng dễ dàng sử dụng và trải nghiệm.
- Đáp ứng đa dạng các nhu cầu cần thiết của người dùng, nghiên cứu
- Giúp người dùng dễ dàng tiếp cận và hiểu rõ hơn về cách hoạt động tìm kiếm chuỗi trong thực tế

3. Hạn chế của đồ án

- Mặc dù giao diện của ứng dụng được thiết kế đơn giản, nhưng vẫn còn một số điểm cần cải thiện để tăng tính thẩm mỹ và trải nghiệm người dùng.
- Hiệu suất của hệ thống chưa được tối ưu hóa khi có lượng dữ liệu lớn.
- Còn thiếu một số tính năng cần thiết để tăng cảm nhận người dùng

4. Hướng phát triển của đồ án

- Ứng dụng vào thực tế nâng cấp việc sử dụng trực tuyến
- Phát triển thêm một số thuật toán tối ưu khác
- Chuyển đổi ngôn ngữ đa dạng giữa các quốc gia cùng với đó là chuyển đổi đơn vị tiền tệ
- Thiết kế thân thiện với tất cả hệ điều hành, sử dụng được trên cả di động
- Khả năng chạy nền để thông báo người dùng không cần khởi động ứng dụng

TÀI LIỆU THAM KHẢO

- [1] “Boyer Moore Algorithm for Pattern Searching” [Online]. Available: <https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/> [Accessed 10 3 2025].
- [2] “Rabin-Karp Algorithm for Pattern Searching”, [Online]. Available: <https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/> [Accessed 9 3 2025].
- [3] “KMP Algorithm for Pattern Searching”, [Online]. Available: <https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/> [Accessed 3 3 2025].
- [4] cp-algorithms, “Prefix function. Knuth–Morris–Pratt algorithm”, 20 8 2023 [Online]. Available: <https://cp-algorithms.com/string/prefix-function.html> [Accessed 3 10 2025].
- [5] “Aho-Corasick algorithm”, 25 3 2024 [Online]. Available: https://cp-algorithms.com/string/aho_corasick.html [Accessed 11 3 2025].
- [6] Nguyễn Minh Nhật, “Thuật toán Aho Corasick”, 8 11 2024 [Online]. Available: <https://wiki.vnoi.info/vi/algo/string/aho-corasick> [Accessed 11 3 2025].