**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**ĐỒ ÁN CUỐI KỲ**

**MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**Mã lớp học phần: DASA230179\_10**

**Học kỳ 1 – Năm học 2024-2025**

**Giảng viên hướng dẫn: Trần Đắc Tốt**

**Danh sách sinh viên thực hiện:**

|  |  |
| --- | --- |
| **MSSV** | **Họ tên** |
| 23110273 | Trương Nhất Nguyên |
| 23110207 | Nguyễn Hoàng Hà |
| 23110222 | Nghiêm Quang Huy |
| 23110304 | Đặng Ngọc Tài |

**Thành phố Hồ Chí Minh, tháng 11 năm 2024**

**KẾ HOẠCH PHÂN CÔNG NHIỆM VỤ THỰC HIỆN ĐỀ TÀI**

**CUỐI KỲ MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**HỌC KỲ I NĂM HỌC 2024-2025**

**1. Mã lớp môn học: DASA230179\_10**

**2. Giảng viên hướng dẫn: Trần Đắc Tốt**

**3. Bảng phân công nhiệm vụ:**

|  |  |
| --- | --- |
| **Sinh viên thực hiện** | **Nội dung thực hiện** |
| Trương Nhất Nguyên | * Code phần mềm từ điển Anh – Việt. * Code giao diện minh họa cây Binary Search Tree. * Tổng hợp và hoàn thiện báo cáo hai phần trên. |
| Nguyễn Hoàng Hà | * Code ứng dụng bài toán Balan ngược. * Code giao diện minh họa cây Binary Search Tree. * Tổng hợp và hoàn thiện báo cáo hai phần trên. |
| Nghiêm Quang Huy | * Code phần xây dựng cấu trúc dữ liệu cây Binary Search Tree. * Code giao diện minh họa STACK và QUEUE. * Tổng hợp và hoàn thiện báo cáo hai phần trên. |
| Đặng Ngọc Tài | * Code phần xây dựng cấu trúc dữ liệu STACK và QUEUE. * Code giao diện minh họa STACK và QUEUE. * Tổng hợp và hoàn thiện báo cáo hai phần trên. |

**LỜI CẢM ƠN**

Để thực hiện được đề tài này, em xin gửi lời cảm ơn chân thành đến thầy Trần Đắc Tốt - giảng viên dạy học và hướng dẫn bộ môn Cấu trúc dữ liệu và giải thuật của lớp DASA230179\_10 vì đã chỉ dẫn và hỗ trợ chúng em hết mình trong việc học tập và trong quá trình thực hiện tiểu luận cuối kỳ.

Trong quá trình học tập, nghiên cứu và thực hiện đề tài chúng em đã nổ lực và cố gắng rất nhiều, nhưng do kinh nghiệm và hiểu biết còn hạn hẹp nên có thể phần báo cáo có thể còn nhiều thiếu sót. Chúng em rất mong nhận được sự động viên, hỗ trợ và góp ý từ thầy để nhóm em có thể hoàn thiện hơn không chỉ trong môn học này mà còn làm tốt hơn nữa trong những chặng đường sắp tới.

Một lần nữa, chúng em xin trân trọng cảm ơn sự quan tâm giúp đỡ của thầy trong suốt hành trình vừa qua.

# PHẦN 1: MỞ ĐẦU

## Lý do chọn đề tài.

## Môn học Cấu trúc dữ liệu và giải thuật đóng vai trò quan trọng trong việc xây dựng nền tảng tư duy thuật toán và khả năng lập trình. Đồ án cuối kỳ nhằm áp dụng các kiến thức đã học để giải quyết các bài toán thực tế, giúp sinh viên nắm vững và triển khai các cấu trúc dữ liệu cơ bản cũng như nâng cao kỹ năng lập trình và tư duy giải thuật.

## Mục tiêu đề tài.

## Đồ án tập trung vào việc triển khai các cấu trúc dữ liệu như STACK, QUEUE, cây nhị phân tìm kiếm, bảng băm, và các thuật toán duyệt đồ thị. Các mục tiêu cụ thể bao gồm:

* Hiểu và cài đặt các thuật toán cơ bản và nâng cao.
* Xây dựng các ứng dụng thực tế dựa trên các cấu trúc dữ liệu.
* Tính toán và xử lý biểu thức toán học bằng thuật toán Balan ngược.
* Phát triển ứng dụng từ điển Anh-Việt với dữ liệu lưu trữ dưới dạng bảng băm hoặc tập tin nhị phân.
* Minh họa trực quan các thuật toán để hỗ trợ việc học tập và giảng dạy.

1. **Phương pháp tiếp cận.**

* Phân tích yêu cầu bài toán và áp dụng các cấu trúc dữ liệu phù hợp.
* Sử dụng các công cụ và ngôn ngữ lập trình như C++, Visual Studio, hoặc Winform để thực hiện các bài toán.
* Tổ chức nhóm làm việc, phân công nhiệm vụ rõ ràng, và đảm bảo tiến độ thực hiện.

1. **Cấu trúc báo cáo.**

Báo cáo được chia thành các phần chính:

* Chương 1: Triển khai các cấu trúc dữ liệu cơ bản QUEUE và ứng dụng vào bài toán duyệt đồ thị theo chiều rộng (BFS)
* Chương 2: Xây dựng cấu trúc dữ liệu cây nhị phân tìm kiếm.
* Chương 3: Viết ứng dụng cho phép người dùng nhập vào một chuỗi của một biểu thức toán học. Sau đó tính giá trị của biểu thức toán này (Gợi ý: Tham khảo bài toán Balan ngược)
* Chương 4: Viết một phần mềm từ điển tiếng Anh đơn giản (không cần giao diện đồ họa).
* Chương 5: Minh họa bằng hình ảnh cho các ứng dụng ở bài 1 và bài 2

# PHẦN 2. NỘI DUNG

## CHƯƠNG 1: XÂY DỰNG CẤU TRÚC DỮ LIỆU QUEUE VÀ ỨNG DỤNG VÀO BÀI TOÁN DUYỆT ĐỒ THỊ

* 1. **Mục đích của chương trình**.

Chương trình được xây dựng để:

* Tìm hiểu và triển khai cấu trúc dữ liệu hàng đợi (Queue)
* Ứng dụng hàng đợi vào việc giải quyết bài toán duyệt đồ thị theo chiều rộng (Breadth-First Search - BFS).
  1. **Cơ sở lý thuyết.**
     1. **Hàng đợi (Queue):**
* Là một cấu trúc dữ liệu hoạt động theo nguyên tắt FIFO (First in, First out).
* Các thao tác chính:
* Enqueue: Thêm phần tử vào cuối hàng đợi.
* Dequeue: Lấy và loại bỏ phần tử đầu hàng đợi.
* IsEmpty: Kiểm tra xem hàng đợi có rỗng không.
  + 1. **Duyệt đồ thị theo chiều rộng (Breadth-First Search - BFS).**
* BFS là một thuật toán dùng để duyệt hoặc tìm kiếm đồ thị.
* Nguyên tắc hoạt động:
* Sử dụng hàng đợi để duy trì các đỉnh cần được duyệt.
* Duyệt qua từng đỉnh theo chiều rộng, tức là duyệt hết tất cả các đỉnh kề trước khi duyệt sâu hơn.
  1. **Phương pháp thực hiện.**
     1. **Biểu diễn đồ thị**

Đồ thị được biểu diễn bằng danh sách kề: Sử dụng danh sách để lưu trữ các đỉnh kề của từng đỉnh.

* + 1. **Thiết kế cấu trúc Queue.**
* Cấu trúc Node : Đại diện cho 1 phần tử trong hàng đợi
* Cấu trúc Queue :
* Các thuộc tính: front, back đại diện cho phần tử đầu và cuối hàng đợi.
* Các phương thức:
* Enqueue(): Thêm phần tử vào cuối hàng đợi.
* Dequeue(): Lấy giá trị phần tử đầu hàng đợi và xóa ra khỏi hàng đợi.
* isEmpty(): Kiểm tra hàng đợi.
  + 1. **Thiết kế thuật toán BFS:**
* Input :
* Một đồ thị.
* Đỉnh bắt đầu duyệt.
* Output:
  + Thứ tự các đỉnh được duyệt.
  + Quy trình:

Đánh dấu đỉnh bắt đầu là đã thăm và đưa vào hàng đợi.

Duyệt từng đỉnh trong hàng đợi:

* Lấy ra khỏi hàng đợi.
* Đưa các đỉnh kề chưa thăm vào hàng đợi.
  1. **Kết quả:**
* Input: tôi sẽ đưa vào chương trình đồ thị với 9 cạnh như sau:

A screenshot of a computer

Description automatically generated

* + - * + Với đỉnh bắt đầu là 1 thì chương trình sẽ cho ra kết quả là:

A black screen with white text

Description automatically generated

* + - * + Sau mỗi lần duyệt BFS xong đồ thị chương trình sẽ đưa ra câu hỏi có muốn tiếp tục với đỉnh khác hay không. Tôi sẽ thử với đỉnh bắt đầu là 3:

A screenshot of a computer program

Description automatically generated

* + - * + Chương trình sẽ cho bạn lựa chọn tiếp tục duyệt BFS với đồ thị khác hay không.
  1. **Kết luận:**
* Trong báo cáo này, chúng tôi đã xây dựng và triển khai kiểu danh sách Queue (hàng đợi) để ứng dụng vào bài toán duyệt đồ thị theo chiều rộng (BFS). Duyệt đồ thị theo chiều rộng là một thuật toán quan trọng trong lý thuyết đồ thị, với các ứng dụng rộng rãi trong việc tìm kiếm, lập lịch, và phân tích kết nối trong mạng.

## CHƯƠNG 2: XÂY DỰNG CẤU TRÚC DỮ LIỆU CÂY NHỊ PHÂN TÌM KIẾM

1. **Mục đích của chương trình:**

Chương trình được xây dựng để:

* Quản lí các Account tránh việc trùng lặp ID dựa vào tính chất của BST
* Tìm kiếm và thêm Account và các thao tác xóa,…với độ phức tạp trung bình là , tệ hơn nữa là trường hợp cây bị lệch về 1 phía trái hoặc phải với n là số lượng tài khoản hiện tại.

1. **Cơ sở lý thuyết:**
   * 1. **Cây BST:**

* Là một cấu trúc dữ liệu dạng cây, với mỗi node có tối đa 2 con, và số cạnh tối đa là (số node – 1), và node con bên phải luôn lớn hơn node con bên trái, và node cha luôn nhỏ hơn node con phải và lớn hơn node con trái, dựa vào tính chất này chúng ta lưu các BankAccount trên cây thì có thể tìm kím hay thao tác với các Account nhanh hơn so với cấu trúc dữ liệu mảng hay là list.
* Các thao tác chính:

|  |  |  |
| --- | --- | --- |
| Thao tác | Trường hợp tốt nhất | Trường hợp xấu nhất |
| addNode: thêm node |  |  |
| removeNode: xóa node |  |  |
| findNode: tìm node |  |  |
| HeightTree: chiều cao cây |  |  |
| printInOrder: in cây |  |  |
| isEmptyTree: cây rỗng |  |  |
| insertNoRecursive |  |  |
| deleteNoRecursive |  |  |

* + 1. **Cấu trúc BankAccount:**
* Là một cấu trúc dữ liệu được viết dưới dạng class chứa các thuộc tính như accountID là id của tài khoản, accountHolderName là tên của chủ tài khoản bankName là tên ngân hàng, balance là số dư. Nó được xem là một node trên cây BST.
* Các thao tác chính:
* getAccountID (lấy id của Account)
* getBalance (lấy số dư của Account)
* getBankName (lấy tên chủ của Account)
* setBalance (đặt lại số dư của Account)
* display (in ra thông tin của tài khoản)

1. **Phương pháp thực hiện.**
   * 1. **Đọc ghi dữ liệu BankAccount**.

* Đọc dữ liệu từ file account.txt dữ liệu sẽ được đọc và ghi vào cây BST dựa vào accountID để làm tiêu chí xét thứ tự trên cây.
  + 1. **Thiết kế cấu trúc của cây BST**

Một node của cây có dạng là (BankAccount, pLeft, pRight) trong đó BankAccount lưu 1 cấu trúc dữ liệu BankAccount chứa các thông tin tài khoản của 1 Account, pLeft lại lưu 1 node con bên trái, pRight lại lưu 1 node con bên phải.

* + 1. **Thực thi chương trình**
* Khi thực thi chương trình thì ngay lúc đó chương trình sẽ hiện ra một bảng chọn các chức năng như sau:

A screenshot of a computer menu

Description automatically generated

* Khi người dùng bấm 0 thì sẽ thực hiện thêm Account vào cây, chương trình sẽ hiện ra bảng chọn kêu người dùng nhập các thông tin vào như sau:

A screen shot of a computer

Description automatically generated

* Khi người dùng bấm 1 hoặc 2 thì sẽ thực hiện thêm Account vào cây:

A screenshot of a computer

Description automatically generated

* Khi người dùng bấm 3 thì sẽ thực hiện xóa Account vào cây:

Nếu người dùng xóa accountID không tồn tại trên cây thì chương trình sẽ hiện ra thông báo rằng:

A screenshot of a computer

Description automatically generated

Nếu người dùng xóa accountID tồn tại trên cây thì chương trình sẽ hiện ra thông báo rằng:

A screenshot of a computer

Description automatically generated

* Khi người dùng bấm 4 thì sẽ thực hiện xóa Account khỏi cây:

Nếu người dùng nhập vào accountID không tồn tại trên cây thì chương trình sẽ thông báo tìm không thấy như sau:

A black and white text

Description automatically generated

Nếu người dùng nhập vào accountID tồn tại trên cây thì chương trình sẽ thông báo tìm thấy như sau:

A black and white text

Description automatically generated

* Khi người dùng bấm 5 thì sẽ thực hiện tìm kiếm tài khoản và chương trình sẽ hiển thị thông báo như sau:

Nếu tài khoản không tồn tại trên cây nó sẽ thông báo như sau:

A black background with white text

Description automatically generated

Nếu tài khoản tồn tại trên cây nó sẽ thông báo như sau:

A screenshot of a computer screen

Description automatically generated

* Khi người dùng bấm 6 thì chương trình sẽ lấy ra chiều cao của cây:

A black background with white text

Description automatically generated

## CHƯƠNG 3: VIẾT ỨNG DỤNG CHO PHÉP NGƯỜI DÙNG NHẬP VÀO MỘT CHUỖI CỦA MỘT BIỂU THỨC TOÁN HỌC SAU ĐÓ TÍNH GIÁ TRỊ BIỂU THỨC NÀY

* 1. **Mục đích của chương trình.**

Chương trình chuyển đổi từ dạng infix (dạng biểu diễn trung tố) sang posfix (dạng biểu diễn hậu tố). Đây là một phần quan trong các hệ thống xử lý biểu thức toán học, đặc biệt là trong việc biên dịch và thực thi mã nguồn.

* 1. **Khái niệm cơ bản.**

Biểu thức Infix: Là biểu thức mà toán tử nằm giữa các toán hạng. Infix yêu cầu quy tắc độ ưu tiên hoặc dấu ngoặc để xác định thứ tự thực hiện các toán tử. Ví dụ: ( A + B ) \* C

Biểu thức Posfix: Là biểu thức mà toán tử nằm sau các toán hạng. Còn được gọi là **Reverse Polish Notation (RPN)**. Trong Posfix, thứ tự thực hiện không phụ thuộc vào dấu ngoặc mà chỉ phụ thuộc vào vị trí toán tử . Khi gặp một toán tử, áp dụng toán tử đó cho hai toán hạng gần nhất bên trái của nó. Ví dụ: A B + C \*

* Ưu điểm:
* Loại bỏ sự cần thiết của dấu ngoặc trong biểu thức.
* Thuận tiện cho máy tính xử lý nhanh chóng bằng cách sử dụng ngăn xếp (Stack).
  1. **Phương pháp thực hiện.**

1. Bước 1: Đọc biểu thức Infix từ trái sang phải
2. Bước 2:

* Nếu gặp toán hạng (số hoặc biến): thêm trực tiếp vào biểu thức Posfix.
* Nếu gặp toán tử, thực hiện:
* Đẩy toán tử vào ngăn xếp nếu ngắn xếp trống hoặc toán tử đang xét có độ ưu tiên cao hơn toán tử trên đỉnh ngăn xếp.
* Nếu không thỏa điều kiện trên, lấy các toán tử ra khỏi ngăn xếp (theo thứ tự) và thêm vào biểu thức Posfix cho đến khi điều kiện thỏa mãn, sau đó đấy toán tử đang xét vào ngăn xếp.
* Nếu gặp dấu ngoặc mở ‘ ( ’ , đẩy vào ngăn xếp.
* Nếu gặp dấu ngoặc đóng ‘ ) ’ , lấy các toán tử ra khỏi ngăn xếp và thêm vào biểu thức Posfix cho đến khi gặp dấu ngoặc mở.

1. Bước 3: Sau khi duyệt hết biểu thức, lấy toàn bộ các toán tử còn lại trong ngăn xếp và thêm vào biểu thức Posfix.

Ví dụ: **( A + B ) x C – D**

A screenshot of a computer

Description automatically generated

* 1. **Cấu trúc chương trình.**

Ngôn ngữ sử dụng: java (giao diện ứng dụng) và c++ (source code)

Hình ảnh mô tả ứng dụng:

A screenshot of a computer

Description automatically generated

Hình ảnh mô tả ứng dụng.

* Chức năng: Lấy biểu thức Infix từ dữ liệu người dùng nhập vào, chuyển đổi thành biểu thức Posfix và xuất ra kết quả.
* Nhập biểu thức vào khung **Enter Infix**
* Sau khi chuyển đổi, biểu thức Posfix sẽ được hiển thị ở ô **Posfix**
* Kết quả sẽ được giển thị ở ô **Result**
* Nút **Enter** sẽ thi hành các lệnh chuyển đổi và tính toán.
* Nút **Clear** sẽ xóa hết dữ liệu ở cả ba ô và trở về trạng thái ban đầu.

A screenshot of a computer

Description automatically generated

Hình ảnh mô tả kết quả

Hình ảnh mô tả kết quả khi biểu thức rỗng hoặc người dùng nhập sai biểu thức Infix.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. **Kết luận:**

Chương trình chuyển đổi biểu thức Infix sang Posfix không chỉ là một công cụ hữu ích trong việc xử lý và biểu diễn các biểu thức toán học mà còn là nền tảng quan trọng cho các ứng dụng lập trình như biên dịch và máy tính. Việc áp dụng thuật toán sử ngăn xếp giúp đảm bảo tính chính xác và hiệu quả trong quá trình chuyển đổi, đồng thời giảm thiểu sự phức tạp của biểu thức. Chương trình giúp người dùng hiểu rõ hơn về cấu trúc biểu thức toán học và tính toán một cách nhanh chóng, vận dụng tốt hơn các khái niệm toán học.

## CHƯƠNG 4: XÂY DỰNG MỘT PHẦN MỀM TỪ ĐIỂN ANH – VIỆT

### 4.1. Giới thiệu chung về phần mềm

**Tên phần mềm:** DICTIONARY ENGLISH - VIETNAMESE WITH AUTOCOMPLETE AND AUTOCORRECT

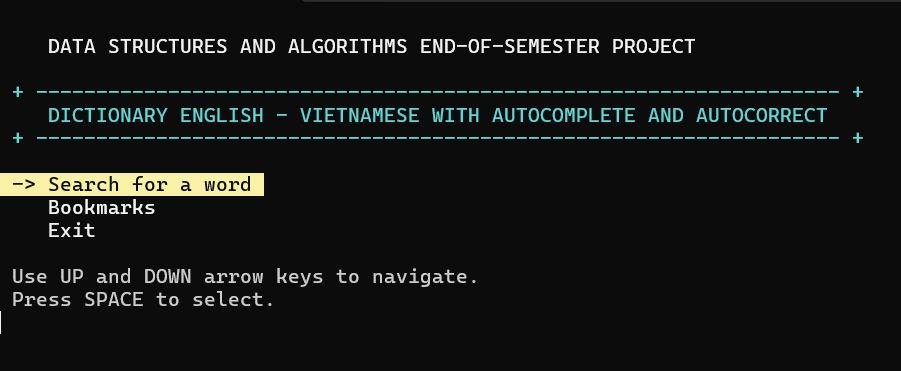
**Mục đích phát triển:**

* Hỗ trợ người dùng học từ vựng hiểu quả hơn .
* Giảm thời gian tra cứu tiếng anh.

**Các chức năng chính:**

* Chức năng tra cứu nghĩa của từ .
* Chức năng gợi ý từ dựa trên tiền tố (AutoComplete).
* Chức năng sửa lỗi chính tả hoặc từ gần đúng (AutoCorrect).
* Chức năng lưu trữ và quản lý danh sách từ đánh dấu(Bookmark).

**Ngôn ngữ sử dụng:** Ngôn ngữ C++.



Hình ảnh giao diện phần mềm

### 4.2. Cơ sở lý thuyết

#### 4.2.1. Hash Table

Hash Table là một cấu trúc dữ liệu cho phép truy cập phần tử theo thời gian O(1) trong trường hợp lý tưởng (không có xung đột).Khi nhập một từ , Hash Table sẽ tìm kiếm nhanh chóng trong các khóa để xác định xem từ đó có trong từ điển hay chưa. Nó sử dụng một hàm băm để ánh xạ từ khóa tới một chỉ mục trong bảng.

#### 4.2.2. Trie

Trie là một cây phân nhánh giúp thực hiện các tác vụ như tìm kiếm tiền tố của từ (prefix search) một cách hiệu quả. Đây là cấu trúc dữ liệu hữu ích cho việc hỗ trợ tính năng AutoComplete. Mỗi nút trong Trie đại diện cho một ký tự của từ, và đường dẫn từ gốc đến nút đó có thể tạo thành các từ.

#### 4.2.3. Linked List

Linked List là một cấu trúc dữ liệu tuyến tính trong đó mỗi phần tử (gọi là node) chứa hai phần là dữ liệu (thông tin mà node lưu trữ) và một con trỏ trỏ đến node tiếp theo trong danh sách. Đối với node cuối cùng của danh sách, con trỏ này thường trỏ đến NULL hoặc một giá trị đặc biệt khác để đánh dấu kết thúc.

### 4.3. Phương pháp thực hiện

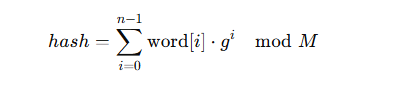
#### 4.3.1. Phương pháp thực hiện Hash Table

**Cấu trúc lớp HashNode:** Đại diện cho một mục từ trong từ điển. Lưu từ khóa và định nghĩa của từ và con trỏ trỏ đến phần tử tiếp theo trong danh sách liên kết.

**Cấu trúc lớp HashTable**: Quản lý từ điển qua bảng băm. Thành phần gồm mảng con trỏ HashNode(chứa danh sách liên kết) nhằm để giải quyết xung đột theo phương pháp Separate Chaining.

**Hàm băm:** Sử dụng băm đa thức (Polynomial Rolling Hash)

Polynomial Rolling Hash là một trong những kỹ thuật băm phổ biến được sử dụng trong từ điển và xử lý chuỗi. Trong đó: g là một số nguyên cơ sở(thường là số nguyên nhỏ 31,33 hoặc 57) và M là kích thước bảng băm.



Hình 29. Hàm băm Polynomial Rolling Hash

Hàm băm đa thức có thể được tính toán trong O(n), với n là độ dài của chuỗi. Hàm băm đa thức sử dụng cơ sở g khác nhau cho từng vị trí của ký tự trong chuỗi. Điều này giúp nó dễ dàng phân biệt giữa các chuỗi ký tự gần giống nhau (ví dụ: “abc” và ”cab”).

Chọn g là số nguyên tố làm tăng tính chất toán học của hàm băm, giảm khả năng va chạm. Trong thực tế, số lượng từ điển trong tiếng Anh lớn nhưng vẫn ít hơn rất nhiều so với số giá trị có thể tạo ra bởi hàm băm đa thức. Điều này giúp phân phối từ khóa tốt hơn và giảm va chạm.

#### 4.3.2. Phương pháp thực hiện Trie

**Cấu trúc lớp TrieNode:** Đại diện cho một nút trong cây Trie. Các thành phần bao gồm biến string def dùng để lưu nghĩa của từ(nếu nút là từ kết thúc), biến int value nhằm đếm số lượng từ trong Trie(mục đích cũng là đánh dấu nút là từ kết thúc), mảng con trỏ, mỗi phần tử trỏ đến một nút con tương ứng với ký tự từ ‘a’ đến ‘z’ và node\_char dùng để lưu ký tự tại nút.

**Cấu trúc lớp Trie:** Thực hiện các chức năng cơ bản của một cây tiền tố(Trie) nhằm hỗ trợ việc gợi ý từ và chỉnh sửa từ sai chính tả. Cấu trúc bao gồm một điểm gốc của cây, là một con trỏ TrieNode.

**Cấu trúc của việc gợi ý từ và chỉnh sửa từ sai chính tả:** Tìm các từ bắt đầu bằng một tiền tố cho trước sau đó duyệt cây và in ra các từ có thể là từ muốn tìm bắt đầu từ một nút.Việc tìm kiếm một tiền tố có độ dài P trong Trie sẽ có độ phức tạp là O(P). Vì cần duyệt qua tất cả ký tự trong tiền tố để tìm nút cuối cùng của tiền tố trong Trie. Sau đó muốn thu thập được K từ có chung tiền tố với từ này. Ta phải duyệt qua các nhánh trong Trie bắt đầu từ nút tiền tố bằng DFS hoặc BFS. Độ phức tạp của việc thu thập tất cả các từ có tiền tố sẽ phụ thuộc vào số lượng từ K cần tìm và độ dài của các từ. Nếu mỗi từ có độ dài trung bình là L, thì độ phức tạp của việc thu thập từ sẽ là O(K \*L). Vậy tổng độ phức tạp của việc gợi ý từ sẽ là O(P +K \* L ).

#### 4.3.3. Phương pháp thực hiện Linked List

**Cấu trúc lớp Node:** Đại diện cho một nút trong Linked List. Lưu dữ liệu data và con trỏ trỏ đến nút tiếp theo.

**Cấu trúc lớp Linked List:** Dùng để thực hiện chức năng Bookmark trong phần mềm. Mỗi khi người dùng thêm một mục đánh dấu mới, mục đó được chèn vào cuối danh sách và ghi vào file.

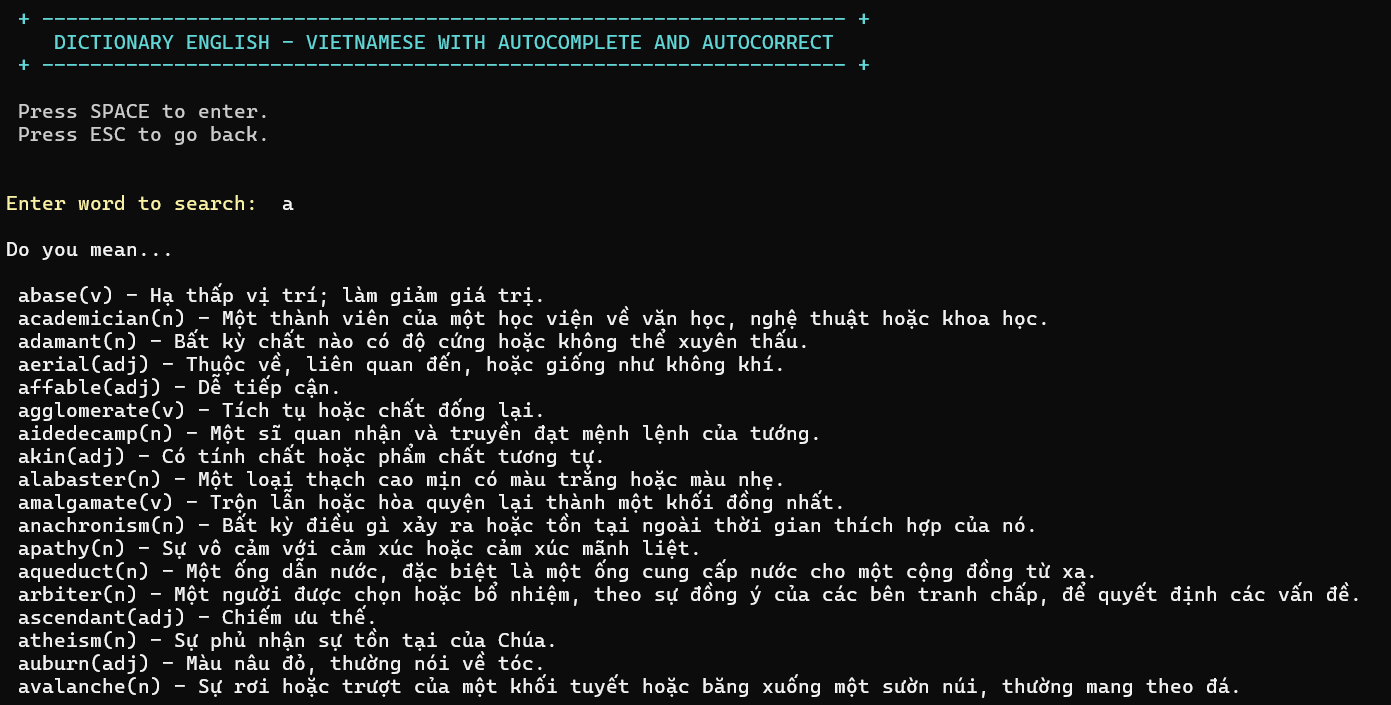
#### 4.3.4. Phương pháp thực hiện UI (User Interface)

**Class UI** được thiết kế quản lý giao diện người dùng cho ứng dụng từ điển với các chức năng tìm kiếm, quản lý Bookmark, và điều hướng qua các màn hình. Class này sử dụng ba màn hình chính. Mỗi màn hình này hiển thị các tùy chọn và điều khiển khác nhau. Màn hình chính hiển thị các tùy chọn như tìm kiếm từ, quản lý bookmark và thoát ứng dụng. Màn hình tìm kiếm cho phép người dùng nhập từ và tìm kiếm định nghĩa từ đó. Màn hình bookmark hiển thị các từ đã được lưu vào danh sách bookmarks của người dùng.

### 4.4. Kết quả và đánh giá

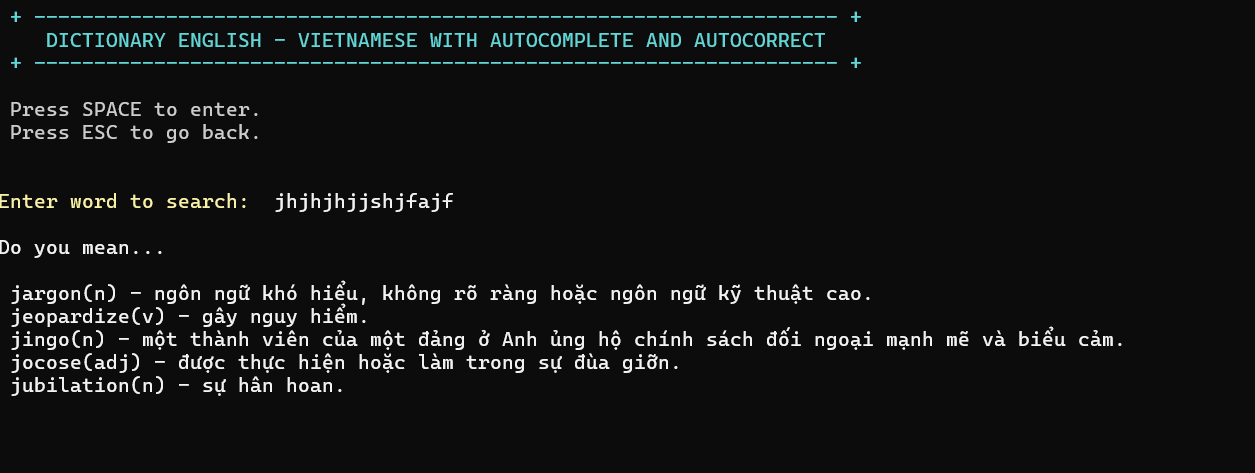
#### 4.4.1. Kết quả

Khi tra cứu từ thì sẽ hiển thị ra gợi ý cho người dùng (tính năng AutoComplete)



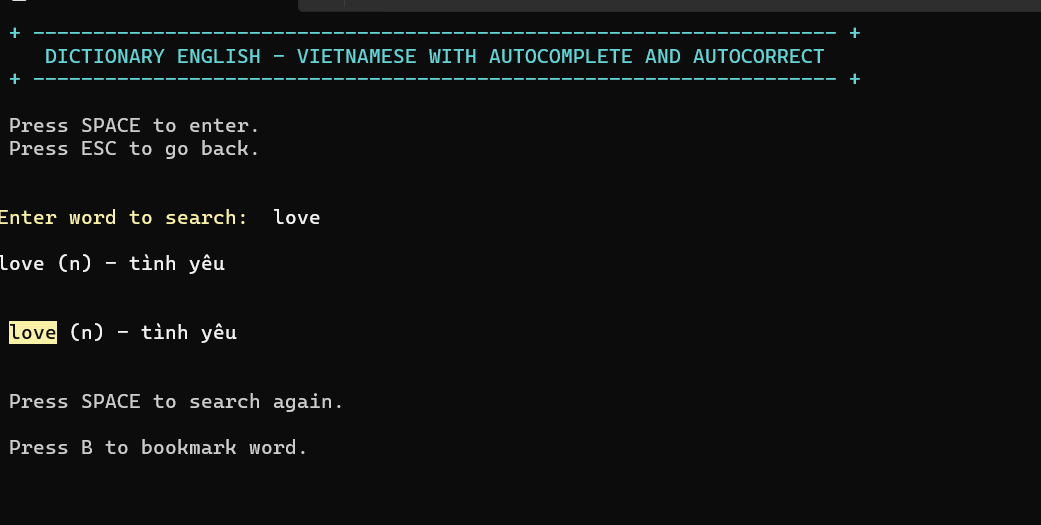
Hình ảnh tính năng AutoComplete

Khi tra cứu từ nếu viết sai thì sẽ hiển thị ra gợi ý cho người dùng có thể sửa (tính năng AutoCorrect)

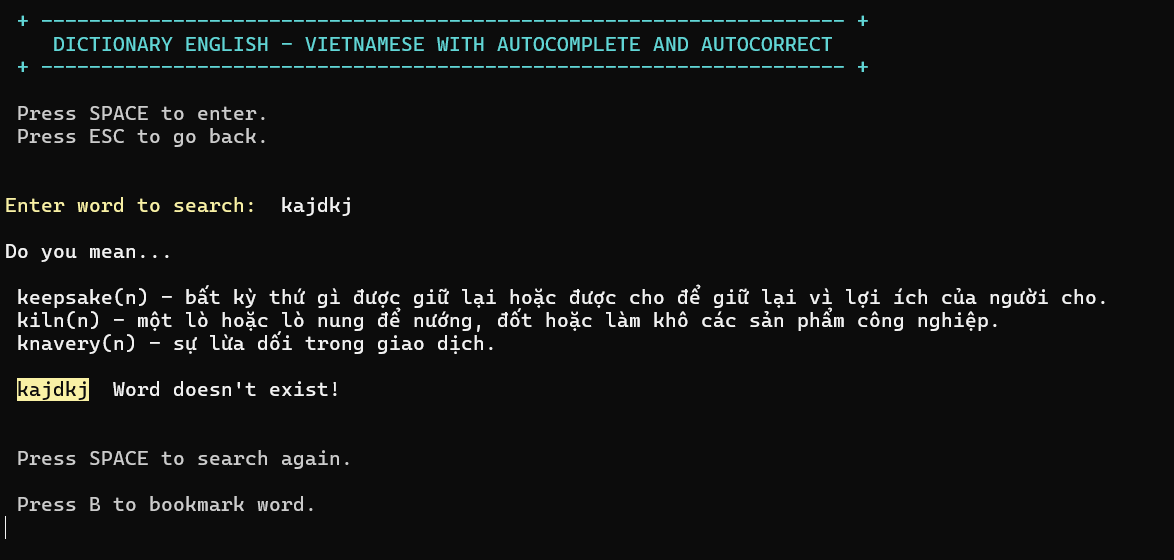


Hình ảnh tính năng AutoCorrect

Khi tra cứu từ có trong từ điển thì nó sẽ hiển thị ra nghĩa của từ đó và nếu không thì hiển thị thông báo không có từ.

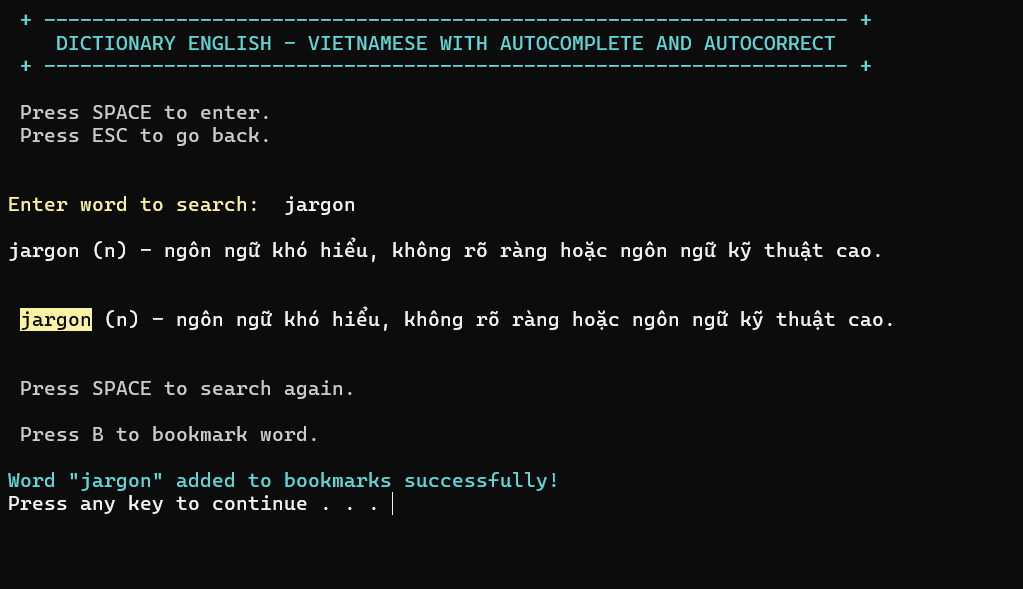


Hình ảnh tra từ



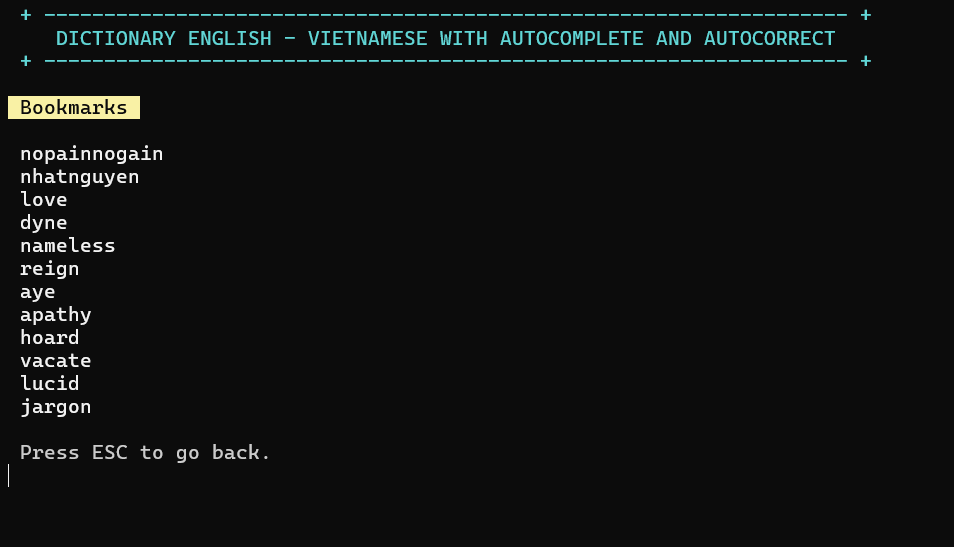
Hình ảnh tra từ

Khi muốn thêm từ vào danh sách bookmark nếu từ chưa xuất hiện trong bookmark thì sẽ thông báo được thêm vào, ngược lại sẽ thông báo từ đã xuất hiện.



Hình ảnh thêm từ vào bookmark

Chức năng hiển thị danh sách bookmark của ứng dụng.



Hình ảnh danh sách bookmark

#### 4.4.2. Đánh giá

**Đánh giá giải thuật:**

* Cấu trúc HashTable: Được sử dụng để lưu trữ các từ và định nghĩa của chúng. Độ phức tạp của thuật toán thao tác tìm kiếm trong bảng băm là O(1) trung bình, tuy nhiên trong trường hợp xấu(đụng độ nhiều) độ phức tạp có thể là O(N). Để khắc phục thì ở đây ứng dụng đã sử hàm băm Polynomial Rolling Hash giúp phân bố các giá trị key đều tránh độ đụng độ. Nhằm tăng tốc độ tìm kiếm từ của từ điển
* Cấu trúc Trie: Dùng cho chức năng tự động hoàn thiện từ (AutoComplete) và chức năng sửa lỗi chính tả (AutoCorrect) để gợi ý khi tra từ. Độ phức tạp tìm kiếm trong cây Trie là O(k) với k là độ dài của từ. Và độ phức tạp của việc gợi ý từ là O(P +K\*L) với P là là độ dài của tiền tố cần gợi ý , K là số từ cần gợi ý và L là độ dài trung bình của mỗi từ.
* Cấu trúc Linked List: Dùng để lưu trữ các từ vào bookmark. Độ phức tạp tìm kiếm ,in danh sách trong danh sách là O(n) với n là số lượng từ trong bookmark.

**Đánh giá cải thiện:**

* Tối ưu danh sách bookmark: Ở đây sử dụng một danh sách liên kết chỉ phục vụ trong việc trong việc dữ liệu nhỏ. Nếu danh sách quá lớn thì để tối ưu hóa các thao tác tìm kiếm và xóa bookmark ta có thể sử dụng Binary Search Tree để cải thiện tìm kiếm và xóa từ O(N) xuống còn O(logN).
* Ở phần thực hiện chức năng AutoCorrect thực tế nó chưa gợi ý từ sửa gần đúng. Có thể áp dụng thêm các thuật toán kiểm tra độ gần đúng (như khoảng cách Levenshtein,Damerau-Levenshtein) để tìm các từ “gần đúng”. Duyệt toàn bộ Trie để tìm các từ có độ khác biệt nhỏ với từ đầu vào.
* Nếu từ điển trở nên quá lớn thì có thể cân nhắc sử dụng các cấu trúc dữ liệu như cây AVL hoặc Red-Black Tree để thay thế cho bảng băm. Điều này giúp duy trì độ phức tạp tìm kiếm O(logN) thay vì O(N) trong trường hợp xấu của bản băm.

## CHƯƠNG 5: MINH HỌA HÌNH ẢNH CÁC CẤU TRÚC DỮ LIỆU

### Minh họa Queue và duyệt đồ thị BFS

**Mục đích phát triển:**

* Minh họa trực quan các thao tác trên queue và duyệt đồ thị BFS

**Các chức năng chính:**

* Chức năng nhập đồ thị theo số cạnh
* Chức năng nhập đỉnh bắt đầu
* Chức năng duyệt đồ thị BFS

**Ngôn ngữ sử dụng:** Ngôn ngữ C#.

**Dưới đây là giao diện bắt đầu của chương trình với 2 tab là BFS minh họa kết quả duyệt BFS và tab QUEUE minh họa cấu trúc dữ liệu QUEUE và các chức năng của nó :**

A screenshot of a computer

Description automatically generated

**Minh họa kết quả duyệt BFS với một đồ thị ví dụ:**

A screenshot of a computer

Description automatically generated

**Mọi dữ liệu nhập vào phải là số nguyên và phải đủ thông tin nếu không sẽ báo lỗi:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Đây là giao diện bắt đầu của tab QUEUE:

A screenshot of a computer

Description automatically generated

**Sau khi thêm lần lượt các giá trị 4, 5, 6, 2, 3 :**

A screenshot of a computer

Description automatically generated

**Khi ấn nút lấy phần tử thì phần tử đầu hàng đợi sẽ được lấy ra ngoài:**

A screenshot of a computer

Description automatically generated

**Ngoài ra còn có các chức năng kiểm tra rỗng và Lấy hết phần tử ra khỏi hàng đợi:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Có các hàm kiểm tra nếu dữ liệu nhập vào không phải số nguyên thì sẽ báo lỗi:**

A screenshot of a computer

Description automatically generated

### Minh họa Binary Search Tree

**Mục đích phát triển:**

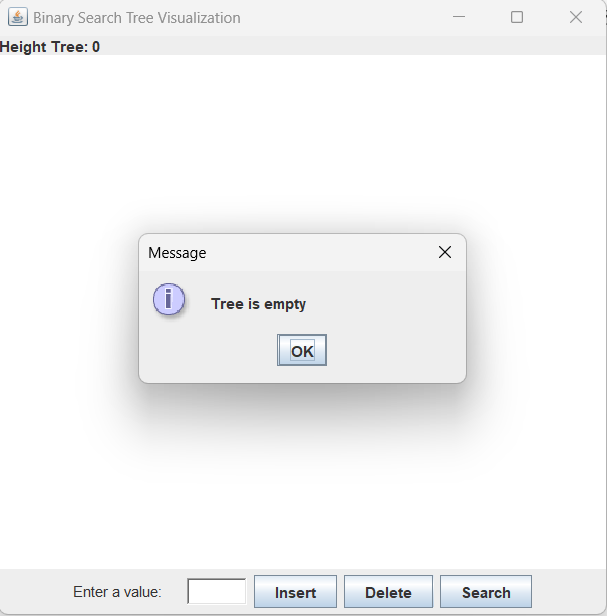
* Minh họa trực quan các thao tác trên cây nhị phân tìm kiếm (BST)

**Các chức năng chính:**

* Chức năng thêm nút vào cây.
* Chức năng xóa nút khỏi cây.
* Chức năng tìm nút trong cây.
* Chức năng hiển thị chiều cao của cây.

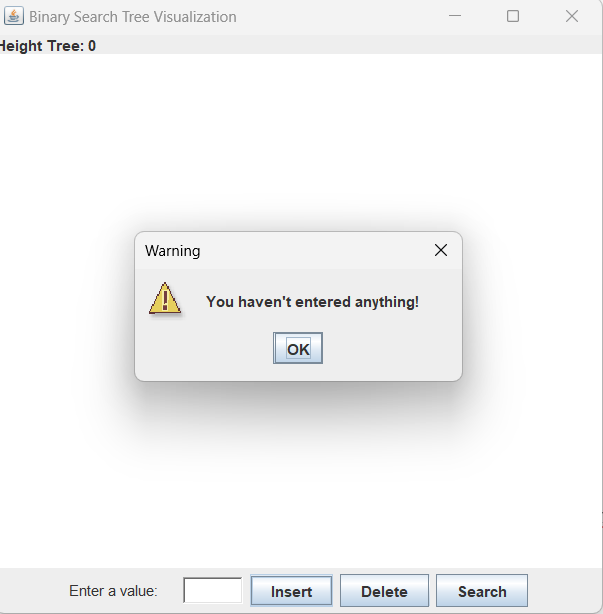
**Ngôn ngữ sử dụng:** Ngôn ngữ Java.

Nếu khi chưa thêm nút nào cho cây hoặc khi xóa hết các nút trong cây sẽ hiển thị thông báo cây rỗng



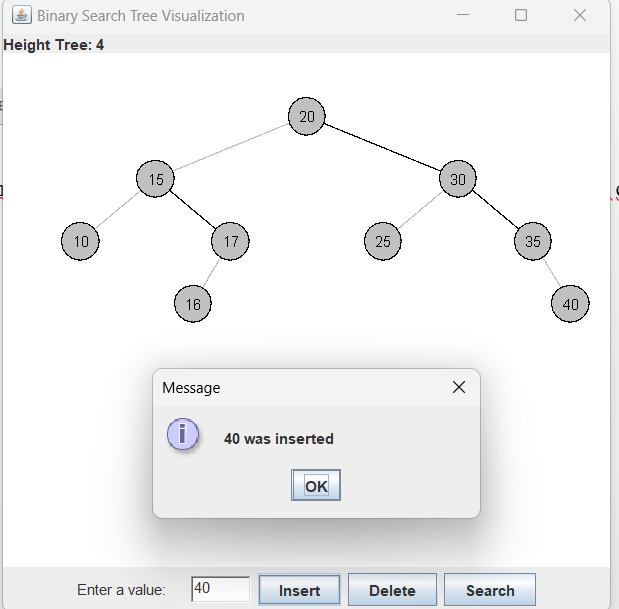
Hình ảnh minh họa giao diện

Nếu không nhập giá trị gì để chèn, xóa hay tìm kiếm thì sẽ hiển thị thông báo cảnh báo.



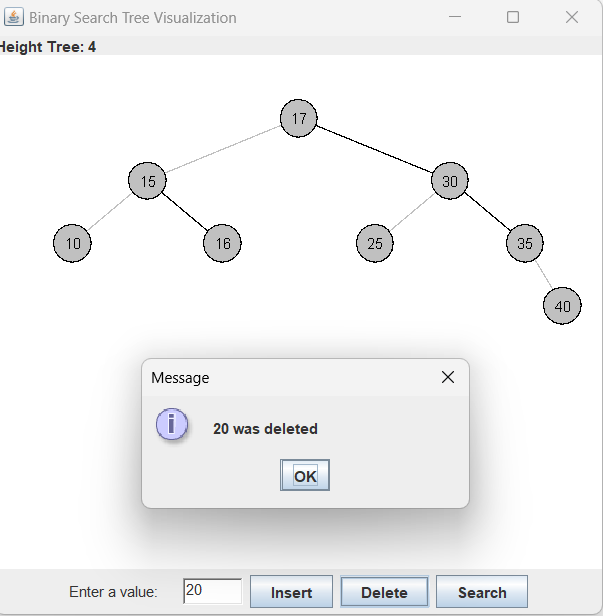
Hình ảnh minh họa giao diện

Khi thêm nút vào cây thì sẽ hiển thị thông báo được thêm vào và nút mới được thêm vào sẽ đảm bảo được tính chất của cây là nút bên phải luôn lớn hơn nút cha của nó và nút bên trái nhỏ hơn nút cha của nó. Và cập nhật chiều cao của cây mỗi lần xóa và cập nhật.



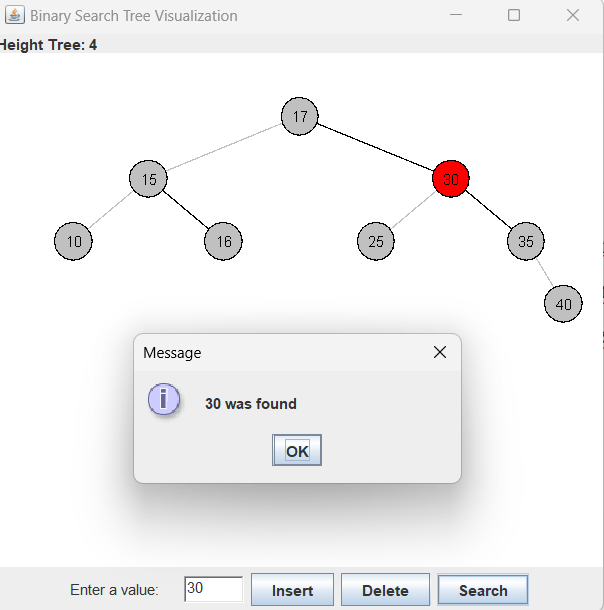
Hình ảnh minh họa giao diện

Khi xóa một nút khỏi cây thì nó bị xóa khỏi cây nhưng vẫn đảm bảo đúng theo tính chất của cây nhị phân tìm kiếm. Và sẽ hiển thị thông báo khi xóa thành công nếu nút muốn xóa không có trong cây cũng sẽ thông báo. Và nếu xóa khi cây rỗng thì sẽ hiển thị thông báo cây rỗng.



Hình ảnh minh họa giao diện

Khi tìm kiếm một nút trong cây nếu nút đó có trong cây thì sẽ nút đó sẽ được tô màu đỏ và hiển thị thông báo tìm thấy. Ngược lại nếu không tìm thấy thì sẽ hiển thị thông báo nút không có trong cây.



Hình ảnh minh họa giao diện

# PHẦN KẾT LUẬN

Qua đồ án này, chúng tôi đã áp dụng các kiến thức về cấu trúc dữ liệu và giải thuật để triển khai thành công các bài toán thực tế, bao gồm quản lý tài khoản ngân hàng, duyệt đồ thị, xử lý biểu thức toán học, và xây dựng phần mềm từ điển Anh-Việt với các tính năng nâng cao như AutoComplete và AutoCorrect.

Những nội dung đã thực hiện cho thấy tầm quan trọng của việc sử dụng đúng cấu trúc dữ liệu, chẳng hạn như cây nhị phân tìm kiếm (BST), bảng băm (HashTable), và cây Trie trong việc tối ưu hóa thao tác xử lý dữ liệu. Đặc biệt, việc minh họa giao diện và kết quả thực thi không chỉ giúp nhóm hiểu sâu hơn về cách hoạt động của các giải thuật mà còn tạo ra các ứng dụng trực quan, hữu ích cho việc học tập và sử dụng thực tế.

Mặc dù đã nỗ lực hoàn thiện, nhưng nhóm nhận thấy vẫn còn những hạn chế như khả năng xử lý dữ liệu lớn hoặc tối ưu hóa các thuật toán tìm kiếm và gợi ý từ gần đúng. Đây sẽ là tiền đề để nhóm tiếp tục cải tiến và học hỏi thêm trong tương lai.

Cuối cùng, chúng tôi chân thành cảm ơn sự hướng dẫn tận tình từ thầy Trần Đắc Tốt và hy vọng nhận được những ý kiến đóng góp từ thầy và các bạn để hoàn thiện hơn các sản phẩm của mình.