**ĐỒ ÁN CUỐI KỲ**

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

**Mã lớp học phần: DASA230179\_10**

**Học kỳ 1 – Năm học 2024-2025**

**Nhóm 7**

|  |  |
| --- | --- |
| MSSV | Họ và tên |
| 23110186 | Tôn Hoàng Cầm |
| 23110372 | Nguyễn Thị Kim Oanh |
| 23110348 | Nguyễn Phạm Bảo Trân |
| 23162019 | Trần Ngọc Đạt |

**NHẬN XÉT CỦA GIẢNG VIÊN**

**Mã Học phần:DASA230179\_10 DASA230179\_10**

* **Hình thức ........................................................................................................................................................................................................................................................................................................................................................................................................................**
* **Nội dung**

**....................................................................................................................................................................................................................................................................................................................................................................................................................**

*TP. Hồ Chí Minh, ngày … tháng… năm 2024*

*Giảng viên ký tên*

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

**CUỐI KỲ CẤU TRÚC CƠ SỞ DỮ LIỆU**

**HỌC KÌ I 2024 - 2025**

**Bảng phân công nhiệm vụ:**

|  |  |
| --- | --- |
| Sinh viên thực hiện | Nội dung phân công |
| Tôn Hoàng Cầm | Bài 5, Cài đặt cấu trúc hàm băm, hàm thêm từ, xoá từ trong bài 4 |
| Nguyễn Thị Kim Oanh | Bài 1, Cài đặt hàm lưu ở bài 4 |
| Nguyễn Phạm Bảo Trân | Bài 2, Cài đặt hàm load và hàm main ở bài 4 |
| Trần Ngọc Đạt | Bài 3, Cài đặt hàm tra cứu từ, hiển thị từ điển ở bài 4 |

**Bảng hoàn thành nhiệm vụ:**

|  |  |
| --- | --- |
| Sinh viên thực hiện | Tỉ lệ hoàn thành |
| Tôn Hoàng Cầm | 100% |
| Nguyễn Thị Kim Oanh | 100% |
| Nguyễn Phạm Bảo Trân | 100% |
| Trần Ngọc Đạt | 100% |

**PHẦN NỘI DUNG**

**I-Bài 1**

* 1. **Giải thích code**

1. **Struct Node và Stack**

A white background with blue text

Description automatically generated

Node:Trong Node chứa :

* Info: Lưu giá trị của phần tử.
* pNext: Con trỏ trỏ đến phần tử tiếp theo trong Stack

A white background with black text

Description automatically generated

Stack là cấu trúc dữ liệu ngăn xếp , lưu trữ dùng để lưu trữ Top là con trỏ trỏ đến phần tử đầu của Stack

1. **Hàm khởi tạo và kiểm tra trạng thái Stack**

A white background with blue text

Description automatically generated

InitStack là khởi tạo Stack rỗng bằng cách gán Top bằng NULL

A white background with blue and red text

Description automatically generated

IsEmpty dùng để kiểm tra Stack rỗng hay không nếu có thì trả về 1 (true) là Top == NULL, nếu không thì trả về 0 (false)

1. **Hàm tạo và thao tác với Node**

A computer code with text

Description automatically generated with medium confidence

CreateNode dùng để tạo một node mới trong danh sách liên kết:

Đầu tiên khai báo một con trỏ kiểu node dùng để trỏ đến node mới được khai báo. Tiếp đến là cấp phát bộ nhớ động (Node\* p = new Node) cho một đối tượng kiểu Node mà không làm chương trình gặp lỗi nếu bộ nhớ không đủ và kiểm tra xem có cấp phát bộ nhớ thành công hay không nếu con trỏ p là NULL, việc cấp phát thất bại và trả về NULL ngay lập tức.

Gán giá trị cho các thành phần của Node p->Info = x dùng để truy cập các thành viên trong Info của node tông qua con trỏ p.

p->pNext = NULL: Truy cập các thành viên trong pNext của node thông qua con trỏ p và dùng để trỏ đến node kế tiếp trong danh sách liên kết và trả về con trỏ node mới (return p)

A screen shot of a computer code

Description automatically generated

Push dùng để thêm một phần tử mới vào Stack:

Hàm CreateNode(x) được gọi để tạo một node mới với giá trị x. Kiểm tra xem nếu cấp phát bộ nhớ không thành công sẽ trả CreateNode(x) về 0 để báo lỗi.

p->pNext = s.Top là một con trỏ của node mới, nó dùng để trỏ tiếp đến node tiếp theo trong Stack

Cập nhật Top của Stack s.Top = p sau khi node mới được liên kết vào Stack, cập nhật con trỏ Top ở Stack (s.Top) để tỏ đến node mới (p) và node mới sẽ trở thành đỉnh mới của Stack.

Trả kết quả về 1 để báo Push đã thực hiện thành công

A computer screen shot of a code

Description automatically generated

Pop được sử dụng để lấy và xóa phần tử trên cùng (Top) của một Stack được cài đặt bằng danh sách liên kết:

Đầu tiên kiểm tra xem Stack có rỗng hay không nếu rỗng hàm Pop trả về 0 để báo rằng không thể thực hiện được thao tác

Lưu giá trị của Node Top: Node\* p = s.Top con trỏ p tỏ đến node của hiện tại nằm trên cùng của Stack lấy giá trị trên cùng của Top gán vào biến x để trả về và cập nhật con tro Top của Stack để trỏ đến node kế tiếp. sau khi cập nhật tại Top sẽ không được Stack tham chiếu tới nữ.

Xóa Node Top cũ để giải phóng bộ nhớ và trả về 1 khi Pop thực hiện thành công.

1. **Các hàm tiện ích**

A computer screen shot of a code

Description automatically generated with medium confidence

Peek được sử dụng để lấy giá trị của phần tử trên cùng (Top) của Stack mà không xóa nó :

Hàm kiểm tra xem Stack có rỗng không bằng cách gọi IsEmpty(s). Nếu Stack rỗng, hàm trả về 0 để báo lỗi

Nếu Stack không rỗng, giá trị của node Top được truy cập thông qua thành viên Info và gán vào biến x, trong khi cấu trúc của Stack vẫn giữ nguyên. Cuối cùng, hàm trả về 1 để báo thao tác đã thực hiện thành công.

A screen shot of a computer code

Description automatically generated

Size được sử dụng để tính số lượng phần tử trong một Stack.

Hàm khởi tạo biến đếm count bằng 0 và sử dụng một con trỏ tạm thời temp trỏ đến node đầu tiên (Top) của Stack.

Hàm duyệt qua các node bằng vòng lặp While, trong đó mỗi lần lặp kiểm tra xem temp có khác NULL hay không. Nếu temp trỏ đến một node hợp lệ, biến count được tăng thêm 1 và con trỏ temp được cập nhật để trỏ đến node tiếp theo thông qua con trỏ pNext.

Vòng lặp kết thúc khi temp trở thành NULL, tức là đã duyệt qua toàn bộ Stack. Cuối cùng, hàm trả về giá trị của count, chính là tổng số phần tử hiện có trong Stack.

A screenshot of a computer code

Description automatically generated

DisplayStack được sử dụng để hiển thị các phần tử trong một ngăn xếp (stack) theo thứ tự từ đỉnh xuống :

Hàm kiểm tra xem ngăn xếp có rỗng hay không bằng cách sử dụng hàm IsEmpty(s). Nếu ngăn xếp rỗng, hàm in ra thông báo "Stack rong." và kết thúc ngay lập tức. Ngược lại, nếu ngăn xếp không rỗng, hàm in thông báo "Cac phan tu trong Stack: " và bắt đầu duyệt qua các phần tử trong ngăn xếp.

Con trỏ tạm temp được khởi tạo bằng con trỏ Top của ngăn xếp, trỏ đến phần tử đỉnh. Dùng vòng lặp while được sử dụng để lần lượt in giá trị của từng phần tử, sau đó con trỏ temp được cập nhật để trỏ đến phần tử tiếp theo trong danh sách liên kết. Khi tất cả các phần tử đã được duyệt, hàm in một dòng trống để xuống hàng, kết thúc việc hiển thị.

1. **Duyệt DFS**

**A screen shot of a computer code

Description automatically generated**

DFS thực hiện duyệt đồ thị theo thuật toán Depth-First Search (DFS) bằng cách sử dụng ngăn xếp để lưu các đỉnh cần xử lý :

Hàm nhận vào ba tham số: tham chiếu đến ngăn xếp stack, danh sách kề List biểu diễn đồ thị dưới dạng danh sách các đỉnh kề, và mảng visited để đánh dấu các đỉnh đã duyệt.

Dùng vòng lặp While để tiếp tục chạy miễn là ngăn xếp không rỗng tiếp theo lấy đỉnh ra khỏi ngăn xếp và lưu giá trị của vào biến c.

Kiểm tra trạng thái của đỉnh c nếu chưa được duyệt thì in ra đỉnh c và đánh dấu đỉnh c đã được duyệt bằng cách gán visited[c] = true.

Duyệt các đỉnh c của danh sách List[c] nếu đỉnh kề chưa được duyệt (!visited [neightbor]) , có sẽ đươc đẩy vào ngăn xếp bằng hàm Push. Khi ngăn xếp rỗng, nghĩa là tất cả các đỉnh có thể duyệt đã được xử lý và in ra một dòng trống để kết thúc.

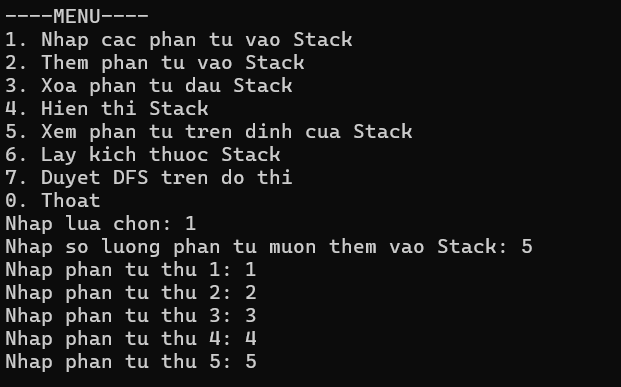
1. **Menu chính**

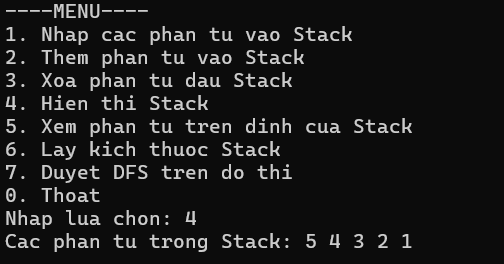
A screenshot of a computer code

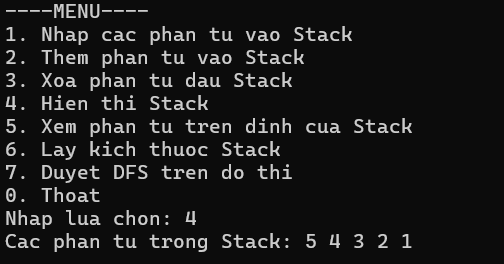
Description automatically generated

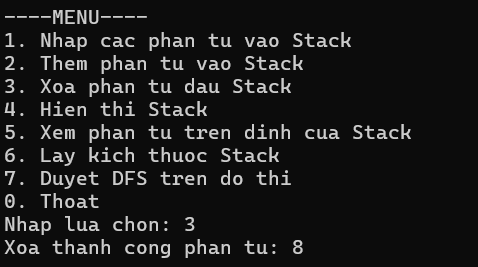
**Gồm các lựa chọn:**

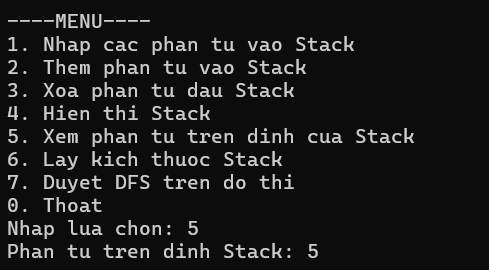
1. Nhap cac phan tu vao Stack
2. Them phan tu vao Stack
3. Xoa phan tu dau Stack
4. Hien thi Stack
5. Xem phan tu tren dinh cua Stack
6. Lay kich thuoc Stack
7. Xoa toan bo Stack
8. Duyet DFS tren do thi
9. Thoat
   1. **Kết quả**

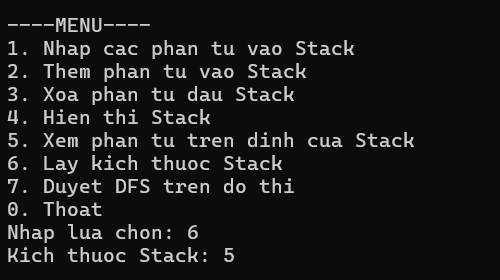


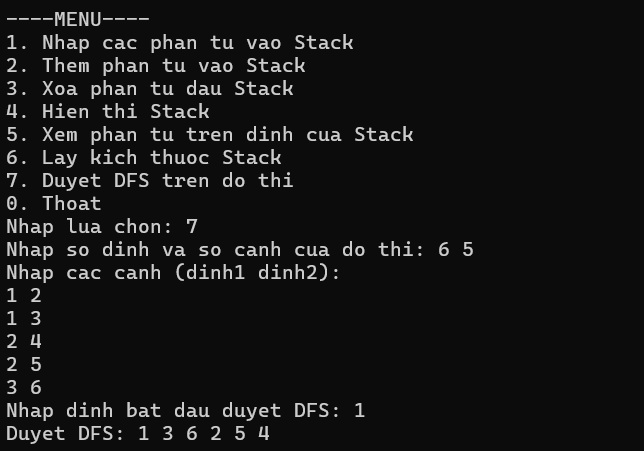












**II-Bài 2**

**2.1 Giải thích code**

**a. Cấu trúc Sach và các toán tử so sánh**

**A screen shot of a computer code

Description automatically generated**

Đầu tiên, cấu trúc Sach lưu trữ thông tin của cuốn sách, bao gồm:

maSach: Mã sách (kiểu int).

tenSach: Tên sách (kiểu string).

tacGia: Tác giả của sách (kiểu string).

giaBan: Giá bán của sách (kiểu float).

Định nghĩa cấu trúc Sach giúp bạn lưu trữ thông tin của từng cuốn sách, bao gồm mã sách, tên sách, tác giả và giá bán. Toán tử so sánh (<, >, ==) trong Sach là rất quan trọng trong BST vì chúng xác định cách các nút được so sánh và đặt vào cây.

Toán tử < giúp xác định vị trí của sách trong cây khi thêm mới.

Toán tử == đảm bảo rằng bạn không thêm các cuốn sách trùng mã.

Toán tử > giúp so sánh khi duyệt cây hoặc thêm sách vào cây con phải.

**b. Tạo nút mới**

**A computer code with black text

Description automatically generated**

Cấp phát bộ nhớ động cho một nút mới. Nếu không đủ bộ nhớ (trả về NULL), chương trình sẽ thoát.

Gán x cho trường key của nút.

Đặt con trỏ trái (pLeft) và phải (pRight) là NULL.

Trả về địa chỉ của nút vừa tạo.

**c.Thêm nút vào cây (đệ quy và không đệ quy)**

**Thêm bằng đệ quy:**

A computer code with text

Description automatically generated with medium confidence

Nếu root là NULL, tức là cây trống hoặc đã tìm thấy vị trí thích hợp, ta sẽ tạo một nút mới.

Nếu mã sách của cuốn sách cần thêm nhỏ hơn mã sách của nút hiện tại, ta sẽ gọi đệ quy vào cây con trái.

Nếu mã sách của cuốn sách cần thêm lớn hơn mã sách của nút hiện tại, ta sẽ gọi đệ quy vào cây con phải.

Lợi ích: Phương pháp này sử dụng đệ quy để tìm vị trí phù hợp cho cuốn sách, làm cho mã ngắn gọn và dễ hiểu hơn. Tuy nhiên, khi cây trở nên quá sâu, có thể gây ra tràn ngăn xếp (stack overflow).

**Thêm không đệ quy:**

A screenshot of a computer code

Description automatically generated

Nếu cây hiện tại là rỗng (tức là root == NULL), hàm sẽ đặt root trỏ tới newNode. Điều này có nghĩa là nút mới sẽ trở thành gốc của cây.

Nếu cây không rỗng, hàm sẽ bắt đầu tìm vị trí thích hợp để chèn nút mới.

Biến p giữ tham chiếu đến nút cha của nút cần thêm, còn biến q giữ tham chiếu đến nút hiện tại trong quá trình duyệt cây.

**Nếu mã sách của cuốn sách cần thêm nhỏ hơn mã sách của nút hiện tại**, ta sẽ tiếp tục tìm kiếm ở cây con trái.

**Nếu mã sách của cuốn sách cần thêm lớn hơn mã sách của nút hiện tại,** ta sẽ tiếp tục tìm kiếm ở cây con phải

Khi vòng lặp kết thúc và tìm được vị trí thích hợp, ta sẽ thêm nút mới vào cây. Nếu mã sách nhỏ hơn mã sách của nút cha, ta thêm nút mới vào cây con trái. Nếu mã sách lớn hơn mã sách của nút cha, ta thêm nút mới vào cây con phải.

Lợi ích: Cách này giúp tránh sử dụng ngăn xếp đệ quy và có thể tiết kiệm tài nguyên bộ nhớ cho các cây lớn. Cách tiếp cận này có thể dễ dàng kiểm soát và dễ dàng điều khiển hơn đối với các trường hợp lớn hoặc khi người dùng muốn tránh sự phát sinh của tràn ngăn xếp (stack overflow) trong các cây quá sâu.

**d. Xóa nút khỏi cây (đệ quy và không đệ quy)**

**Xóa đệ quy:**

A screenshot of a computer program

Description automatically generated

Tìm nút cần xóa: Nếu mã sách cần xóa nhỏ hơn mã sách của nút gốc, tìm tiếp ở cây con trái. Nếu mã sách cần xóa lớn hơn mã sách của nút gốc, tìm tiếp ở cây con phải. Hàm sẽ đệ quy tìm đến vị trí của nút cần xóa.

Khi tìm được nút cần xóa, sẽ có ba trường hợp xảy ra tùy vào cấu trúc của cây con của nút đó.

Nếu nút cần xóa không có cây con trái, thì chỉ cần thay thế nút đó bằng cây con phải. Sau khi xóa nút gốc, cây sẽ nối trực tiếp vào cây con phải (nếu có).

Nếu nút cần xóa không có cây con phải, thì chỉ cần thay thế nút đó bằng cây con trái. Sau khi xóa nút gốc, cây sẽ nối trực tiếp vào cây con trái (nếu có).

Nếu nút cần xóa có cả cây con trái và cây con phải, thì cần tìm một phần tử thay thế. Phần tử thay thế sẽ là nút có giá trị nhỏ nhất trong cây con phải của nút cần xóa (nút ở vị trí cực trái của cây con phải). Sau khi tìm được nút thay thế, giá trị của nút cần xóa sẽ được thay thế bằng giá trị của nút thay thế. Tiếp theo, gọi đệ quy để xóa nút nút thay thế trong cây con phải, vì nó đã được sao chép vào vị trí nút cần xóa.

Lợi ích: Xóa đệ quy giúp mã dễ đọc và bảo trì, đặc biệt là khi xử lý các trường hợp phức tạp như xóa nút có hai con. Các nút không có con hoặc chỉ có một con được xử lý đơn giản bằng cách trả về con của nút đó. Đệ quy giúp duy trì quy tắc cây nhị phân mà không cần thao tác thủ công, như trong việc tìm người kế thừa theo thứ tự.

**Xóa không đệ quy:**

A screenshot of a computer program

Description automatically generated

q là con trỏ dùng để duyệt qua cây, ban đầu trỏ tới gốc (root).

p giữ tham chiếu đến nút cha của nút hiện tại (q).

Dùng vòng lặp while để kiểm tra từng nút. Nếu giá trị cần xóa nhỏ hơn giá trị của nút hiện tại, ta tiếp tục tìm kiếm qua cây con trái. Ngược lại, nếu giá trị cần xóa lớn hơn, ta tìm kiếm qua cây con phải. Việc di chuyển qua các nhánh cây và thay thế các nút cần xóa dễ dàng hơn mà không cần phải kiểm tra nhiều điều kiện phức tạp.

Nếu sau khi tìm kiếm không tìm thấy nút cần xóa, hàm sẽ trả về mà không làm gì.

Nếu tìm thấy nút và nút đó không có con hoặc chỉ có một con, hàm sẽ thay thế nút đó bằng đứa con của nó. Nếu nút không có con trái, nó sẽ được thay thế bởi con phải, hoặc ngược lại. Nút cha của nút bị xóa (được theo dõi bởi p) sẽ được cập nhật để trỏ tới đứa con của nút bị xóa. Nút bị xóa sau đó sẽ được xóa khỏi bộ nhớ.

Nếu nút cần xóa có hai con, một quy trình phức tạp hơn được yêu cầu. Trong trường hợp này, hàm tìm người kế thừa theo thứ tự (in-order successor) của nút, là nút nhỏ nhất trong cây con bên phải (nút trái nhất của con phải). Hàm sao chép giá trị của người kế thừa theo thứ tự vào nút cần xóa. Sau khi sao chép giá trị, người kế thừa theo thứ tự sẽ được xóa khỏi vị trí ban đầu của nó, quy trình này dễ dàng hơn vì người kế thừa sẽ có tối đa một con (vì nó là nút trái nhất của cây con bên phải).

Lợi ích: Không sử dụng ngăn xếp như đệ quy, giúp tránh tràn bộ nhớ. Giảm nguy cơ tràn ngăn xếp khi cây quá sâu. Quá trình duyệt và xóa dễ dàng theo dõi qua vòng lặp thay vì các cuộc gọi đệ quy phức tạp.

**e. Tìm kiếm sách theo mã sách**

**A computer code with black text

Description automatically generated**

Nếu cây rỗng (root == NULL) hoặc đã duyệt hết cây mà không tìm thấy mã sách. Trả về NULL để báo rằng mã sách không tồn tại.

Nếu Tìm thấy nút chứa mã sách cần tìm (root->key.maSach == maSach). Trả về con trỏ root trỏ tới nút tìm thấy.

Nếu mã sách cần tìm nhỏ hơn mã sách của nút hiện tại. Theo quy tắc của BST, mã sách cần tìm chỉ có thể nằm trong cây con trái. Gọi đệ quy với cây con trái (root->pLeft).

Nếu mã sách cần tìm lớn hơn mã sách của nút hiện tại. Theo quy tắc của BST, mã sách cần tìm chỉ có thể nằm trong cây con phải. Gọi đệ quy với cây con phải (root->pRight).

**f. Duyệt cây theo thứ tự In-order**

A computer screen shot of text

Description automatically generated

Hàm kiểm tra xem nút hiện tại (root) có tồn tại không. Nếu root == NULL (nút rỗng), thì không thực hiện gì và trả về.

Đệ quy gọi lại hàm InOrder cho cây con bên trái (root->pLeft). Điều này đảm bảo các nút ở có giá trị nhỏ hơn được duyệt trước.

Sau khi hoàn thành cây con bên trái sẽ in ra thông tin của sách.

Đệ quy gọi lại hàm InOrder cho cây con bên phải (root->pRight). Điều này đảm bảo các nút ở có giá trị lớn hơn được duyệt sau.

Duyệt cây theo thứ tự In-order (trái - gốc - phải) giúp in ra các cuốn sách theo thứ tự tăng dần của mã sách giúp dễ dàng xem danh sách các cuốn sách được sắp xếp.

**g.Tính chiều cao của cây**

A computer code with text

Description automatically generated

Gọi đệ quy để tính chiều cao của cây con bên trái. Hàm sẽ tiếp tục tính chiều cao cho các cây con trái của các nút cho đến khi gặp điều kiện dừng.

Tương tự như cây con trái, gọi đệ quy Height(root->pRight) để tính chiều cao của cây con bên phải.

Hàm trả về chiều cao của cây hiện tại bằng cách chọn chiều cao lớn hơn giữa cây con trái và cây con phải, rồi cộng thêm 1 để tính thêm nút gốc của cây hiện tại.

Lợi ích: Hàm Height tính chiều cao của cây cho biết độ sâu của cây hiện tại. Đây là một chỉ số quan trọng để đánh giá sự cân bằng của cây. Nếu cây quá cao (nghĩa là bất kỳ thao tác tìm kiếm nào đều mất quá nhiều thời gian), ta có thể cần phải cân bằng lại cây (như trong các cây AVL hoặc Red-Black Trees).

**2.2 Kết quả**

**Thêm bằng đệ quy:**

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

**Thêm không đệ quy:**

**A black background with white text

Description automatically generated**

**A black background with white text

Description automatically generated**

**Xóa đệ quy:**

**A black background with white text

Description automatically generated**

**Xóa không đệ quy:**

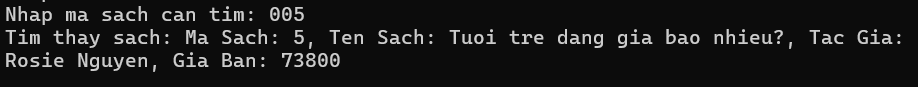
**A black background with white text

Description automatically generated**

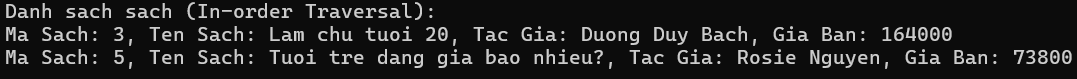
**Tìm kiếm sách bằng mã sách:**

**A black background with white text

Description automatically generated**

****

**Hiển thị cây:**

****

**Chiều cao của cây:**

****

**Mục đích:** Ứng dụng cấu trúc dữ liệu cây nhị phân tìm kiếm (BST) để lưu trữ và quản lý dữ liệu hiệu quả, cụ thể là kiểu dữ liệu Sach. Bài toán mô phỏng một hệ thống lưu trữ thông tin sách (như mã sách, tên sách, tác giả). Thông qua BST, việc thêm, xóa, và tìm kiếm thông tin sách được thực hiện nhanh chóng với độ phức tạp trung bình là O(log⁡n).

**III-Bài 3**

**3.1 Công việc:**

**Đọc yêu cầu của đề:**

* Đọc và hiểu yêu cầu của bài toán: 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.
* Đọc và tham khảo 1 vài tài liệu liên quan bài toán Balan ngược.

**Thiết kế kiến trúc:** Lên kế hoạch cho cấu trúc mã nguồn, xác định các lớp và phương thức cần thiết. Ở đây là lớp calculator và các phương thức như evaluate, infixToPostfix, countPostfix.

**Viết mã:**

* Cài đặt lớp calculator: Tạo lớp chính và các phương thức bên trong.
* Triển khai precedence: Xác định mức độ ưu tiên của các toán tử.
* Triển Khai count: Thực hiện các phép toán số học.
* Triển Khai infixToPostfix: Chuyển đổi biểu thức từ dạng infix sang postfix.
* Triển Khai countPostfix: Đánh giá biểu thức postfix.

**Xử lý lỗi**

Thêm các cơ chế xử lý lỗi để đảm bảo chương trình có thể xử lý các tình huống không hợp lệ (chia cho 0, dấu ngoặc không khớp, toán tử không hợp lệ).

Kiểm tra và chạy chương trình

* Viết các trường hợp kiểm tra để đảm bảo chương trình hoạt động như mong đợi.
* Chạy chương trình với các biểu thức khác nhau và xác minh kết quả.

Tối ưu mã nguồn

* Xem xét mã nguồn để cải thiện hiệu suất và tính dễ đọc.
* Có thể thêm ghi chú và tài liệu hóa mã nguồn để người khác dễ hiểu.

**3.2 Chi tiết ý tưởng:**

Chương trình này triển khai một máy tính đơn giản có thể đánh giá các biểu thức toán học được cho theo dạng infix (tiền tố) và chuyển đổi chúng sang dạng postfix (hậu tố) để đánh giá. Nó xử lý các phép toán cơ bản: Cộng, trừ, nhân và chia, bao gồm cả kiểm tra lỗi cho các biểu thức không hợp lệ và chia cho số không.

VD: 3 + 4 \* (10-3)/7 -> 3 4 10 3-\*7/+

**Các thành phần chính:**

**Định nghĩa lớp**: Lớp calculator bao hàm tất cả chức năng liên quan đến việc đánh giá các biểu thức.

**Phương thức công khai:**

double evaluate(const string& expression): Đây là phương thức chính nhận một biểu thức dưới dạng chuỗi, chuyển đổi nó sang dạng, đánh giá nó và trả về giá trị double kết quả.

A computer screen shot of text

Description automatically generated

**Phương thức riêng :**



int precedence(char op): Trả về độ ưu tiên của các toán tử (+, - , \* , /). Số lớn hơn biểu thị độ ưu tiên cao hơn.



count: Thực hiện các phép toán số học dựa trên toán tử. Nếu có phép chia cho 0, nó sẽ ném ra lỗi.

A screen shot of a computer program

Description automatically generated

infixToPostfix: Chuyển đổi biểu thức từ dạng infix sang postfix,, sử dụng một ngăn xếp để quản lý các toán tử và đảm bảo xử lý đúng mức độ ưu tiên và dấu ngoặc.

A black background with blue and white text

Description automatically generated

countPostfix: Tính toán giá trị của biểu thức postfix bằng cách sử dụng ngăn xếp. (stack), đẩy các số vào ngăn xếp và lấy chúng ra để thực hiện các phép toán.

**3.3 Phân tích source code:**

**a. Phương thức evaluate**

A computer screen shot of text

Description automatically generated

Gọi infixToPostfix để chuyển đổi biểu thức từ dạng infix sang dạng postfix.

Truyền biểu thức postfix thu được cho countPostfix để tính toán kết quả cuối cùng.

**b. Phương thức precedence**

A black background with white text

Description automatically generated

Định nghĩa các mức độ ưu tiên:

+ và - có độ ưu tiên 1.

\* và / có độ ưu tiên 2.

**c. Phương thức count**

A computer screen shot of a program code

Description automatically generated

Thực hiện các phép toán số học dựa trên toán tử được truyền vào.

Ném ngoại lệ cho các phép toán không hợp lệ (ví dụ: chia cho số 0).

**d. Phương thức infixToPostfix**

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**Đầu vào:** Một chuỗi đại diện cho biểu thức toán học.

**Đầu ra:** Một vector các chuỗi đại diện cho biểu thức ở dạng postfix.

**Logic:**

* Lặp qua từng ký tự trong biểu thức đầu vào.
* Nếu là chữ số, nó xây dựng số đầy đủ (để xử lý số nhiều chữ số) và đẩy nó vào đầu ra.
* Nếu là toán tử, nó quản lý ngăn xếp toán tử dựa trên độ ưu tiên bằng cách sử dụng vòng lặp while để đảm bảo thứ tự thực hiện đúng.
* Xử lý dấu ngoặc đơn bằng cách đẩy chúng vào ngăn xếp và lấy các toán tử ra cho đến khi tìm thấy dấu ngoặc mở phù hợp.

**e. Phương thức countPostfix**

A computer screen shot of a program code

Description automatically generated

**Đầu vào:** Một vector các chuỗi ở dạng postfix.

**Đầu ra:** Kết quả đánh giá dưới dạng double.

**Logic:** Sử dụng một ngăn xếp để đánh giá biểu thức postfix.

Đối với từng token:

* Nếu là số, đẩy nó vào ngăn xếp.
* Nếu là toán tử, lấy hai số ở trên cùng ngăn xếp, thực hiện phép toán, và đẩy kết quả trở lại.
* Cuối cùng, nó kiểm tra xem có đúng một giá trị còn lại trên ngăn xếp (kết quả).

Ném ngoại lệ nếu:

* Chia cho số 0.
* Dấu ngoặc không khớp.
* Các biểu thức không hợp lệ (VD: không đủ toán hạng cho một toán tử).

**f. Hàm chính**

A computer screen with text

Description automatically generated

Yêu cầu người dùng nhập một biểu thức toán học

Tạo một thể hiện của lớp calculator và cố gắng đánh giá biểu thức.

Bắt và hiển thị bất kỳ lỗi runtime nào xảy ra trong quá trình đánh giá.

**3.4 Tác dụng và lý do sử dụng**

**a. Tác dụng**

**Tính toán biểu thức:** Cung cấp chức năng để tính toán các biểu thức toán học từ chuỗi.

**Kiểm tra lỗi:** Xử lý các trường hợp lỗi như chia cho 0, biểu thức không hợp lệ và thiếu dấu ngoặc.

**b. Lý do sử dụng**

Sử dụng ngăn xếp giúp đơn giản hóa việc xử lý toán tử và thứ tự thực hiện phép toán.

Chuyển đổi từ dạng infix sang dạng postfix giúp đơn giản hóa việc tính toán mà không cần lo lắng về thứ tự ưu tiên của các toán tử.

**3.5 Kết quả Demo**

A screenshot of a computer screen

Description automatically generated

Chuyển đổi sang postfix : Khi nhập vào 3 + 4 \* (10 - 3) / 7, nó sẽ chuyển đổi thành: 3 4 10 3 - \* 7 / +

Tính toán giá trị postfix: Chương trình sẽ duyệt qua các phần tử trong postfix:

Tính 10 - 3 → 7.

Tính 4 \* 7 → 28.

Tính 28 / 7 → 4.

Tính 3 + 4 → 7.

Cuối cùng: Code sẽ báo: “Gia tri bieu thuc: 7”

A screenshot of a computer

Description automatically generated

Chuyển đổi sang postfix: Khi nhập biểu thức 14 / 0, nó sẽ chuyển đổi thành 14 0 /

Tính toán giá trị postfix: Khi gặp toán tử “/”, chương trình sẽ thực hiện phép chia 14 / 0.

Xử lý lỗi: Do phép chia cho 0 không hợp lệ, code sẽ báo lỗi: “Khong the chia cho 0”.

A screenshot of a computer screen

Description automatically generated

Chuyển đổi sang postfix: Khi nhập vào 38 + (64 / 7- 5 + 1, chương trình sẽ gặp lỗi do thiếu dấu ngoặc đóng.

Kết quả: Code sẽ báo lỗi: “Bieu thuc thieu dau ngoac”.

A screenshot of a computer screen

Description automatically generated

Chuyển đổi sang postfix: Khi nhập vào 1 + 5 – 14 / 3 - w, nó sẽ chuyển đổi thành 1 5 + 14 3 / - w -

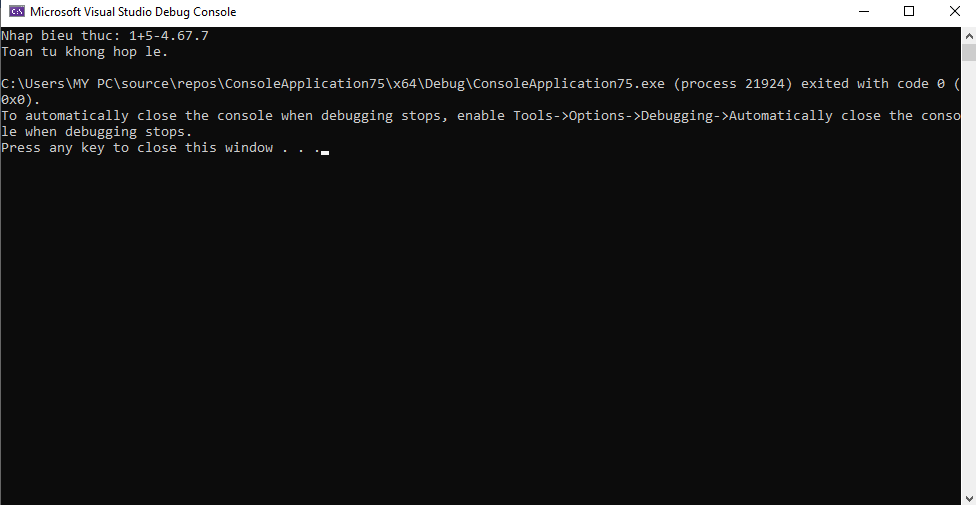
Tính toán giá trị postfix :

* Giá trị w: Nếu w là biến chưa được định nghĩa hoặc không có giá trị, chương trình sẽ ném ra lỗi.
* Nếu w có giá trị (ví dụ: w = 7), thì sẽ thực hiện các phép toán:
  + Tính 1 + 5 → 6.
  + Tính 14 / 3 → 4.67 (xấp xỉ).
  + Tính 6 - 4.67 → 1.33 (xấp xỉ).
  + Tính 1.33 – 7.

Kết quả :

* Nếu w là biến không xác định, code sẽ báo lỗi: “Ki tu khong hop le”.
* Nếu w là biến xác định, nó sẽ trả về một số cụ thể, mà ở đây là xấp xỉ -5.67.

\*Chú ý: Ở trên ta tính xấp xỉ để dễ hình dung, nhưng khi nhập vào vẫn phải giữ nguyên 14/3, nếu không khi nhập 4.67 code sẽ báo lỗi: “Toan tu khong hop le”.



**IV-Bài 4**

**4.1. Ý Tưởng**

Tạo một từ điển Anh-Việt dựa trên cấu trúc dữ liệu bảng băm với các chức năng:

* Thêm từ mới.
* Tra cứu nghĩa của từ.
* Xóa từ khỏi từ điển.
* Hiển thị toàn bộ từ điển.
* Lưu và tải dữ liệu từ tệp nhị phân.

Mô hình hoạt động :

* **Hàm băm:** Chuyển từ tiếng Anh thành chỉ số trong bảng băm.
* **Danh sách liên kết:** Lưu trữ nhiều từ tại cùng chỉ số nếu xảy ra xung đột.
* **Chức năng :** Các thao tác như thêm, xóa, tra cứu dựa trên chỉ số từ hàm băm.

**4.2 Nội dung**

**a. Lớp HashNode**

Một từ trong từ điển có đặc điểm : key: từ tiếng Anh và value: nghĩa tiếng Việt.

**b. Lớp HashTable**

Lưu trữ từ điển sử dụng với table là vector với các phần tử là 1 danh sách có kiểu HashNode để dễ dàng thực hiện các thao tác chèn, xoá,…

Hàm int hashFunction : Chuyển chuỗi (từ tiếng Anh) thành chỉ số trong bảng :

* Sử dụng mã ASCII của từng ký tự để tính giá trị băm.
* Kết hợp phương pháp băm chuỗi: hash = (hash \* 31 + character) % Table\_Size ( Giá trị 31 là số nguyên tố nhỏ, giảm xung đột )

**c. Hàm insert :**

**Chức năng :** Thêm một từ mới (hoặc cập nhật nghĩa nếu từ đã tồn tại)

**Hoạt động :**

* Tính chỉ số băm: Dựa trên từ tiếng Anh (key)
* Duyệt qua danh sách tại chỉ số băm: Nếu key tồn tại, cập nhật value. Nếu không, thêm mới bằng emplace\_back ( dùng thay push\_back vì sẽ thêm trực tiếp vào không cần tạo biến sao chép )

**c.Hàm remove :**

**Chức năng :** Xóa một từ khỏi từ điển.

**Hoạt động :**

* Tính chỉ số băm: Tìm danh sách chứa từ cần xóa.
* Duyệt danh sách: So sánh key với từng nút.
* Nếu tìm thấy, sử dụng erase để xóa nút khỏi danh sách.

**d. Phương thức search**

Phương thức search được thiết kế để tra cứu nghĩa của một từ trong từ điển. Nó sử dụng một hàm băm để xác định vị trí của từ trong bảng băm, sau đó tìm kiếm trong danh sách liên kết tại vị trí đó.

Thuộc tính:

* key: Tham số đầu vào, là từ tiếng Anh mà người dùng muốn tra cứu.
* table: Là bảng băm lưu trữ các từ và nghĩa, được định nghĩa như một vector chứa danh sách (list<HashNode>).
* hashFunction(key): Tính toán chỉ số băm cho từ khóa.
* Duyệt qua danh sách tại table[index]: Tìm kiếm từng node trong danh sách để so sánh với key.
* Trả về giá trị: Nếu tìm thấy, trả lại nghĩa; nếu không, trả về thông báo không tìm thấy.

A screen shot of a computer code

Description automatically generated

**e. Phương thức printDictionary**

Phương thức printDictionary được thiết kế để hiển thị toàn bộ từ điển, in ra tất cả các từ và nghĩa của chúng. Nó giúp người dùng có cái nhìn tổng quát về nội dung của từ điển.

Thuộc tính :

* table: Là bảng băm chứa các từ và nghĩa, được định nghĩa như một vector chứa danh sách liên kết (list<HashNode>).
* Duyệt qua tất cả các chỉ số trong table: Kiểm tra nếu danh sách tại chỉ số đó không rỗng.
* In ra từng từ và nghĩa: Đối với mỗi node trong danh sách không rỗng, in ra key và value.

A computer screen with white text

Description automatically generated

**e. Hàm chính (int main)**

Tra cứu từ (case 2):

Mục đích: Cho phép tra cứu nghĩa của một từ trong từ điển. Khi người dùng nhập từ, chương trình sẽ tìm kiếm nghĩa tương ứng và hiển thị kết quả.

Nhập từ:

* string key:: Tạo biến key để lưu từ mà người dùng nhập.
* cin >> key:: Người dùng nhập từ cần tra cứu từ bàn phím.

Gọi phương thức search: dictionary.search(key): Gọi phương thức search trong đối tượng dictionary để tra cứu nghĩa của từ.

Hiển thị kết quả: cout << "Nghia : " << dictionary.search(key) << endl;: In ra nghĩa của từ. Nếu từ không tồn tại, thông báo tương ứng sẽ được hiển thị.

A black background with white text

Description automatically generated

Hiển thị từ điển (case 4):

Mục đích: Cho phép người dùng xem toàn bộ nội dung của từ điển, bao gồm tất cả các từ và nghĩa của chúng.

Thông báo cho người dùng: cout << "Tu dien : \n";: In ra tiêu đề cho phần hiển thị từ điển.

Gọi phương thức printDictionary: dictionary.printDictionary();: Gọi phương thức printDictionary trong đối tượng dictionary để in ra toàn bộ các từ và nghĩa từ bảng băm.

A screen shot of a computer screen

Description automatically generated

**f. Hàm save**

A computer code with text

Description automatically generated with medium confidence

Mở tệp để ghi ở chế độ nhị phân . Dùng ofstream vì cần ghi dữ liệu ra tệp.

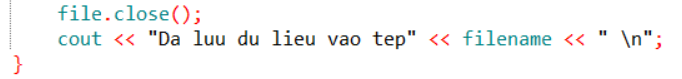
Kiểm tra nếu không thể mở tệp (ví dụ, không tồn tại hoặc lỗi quyền truy cập), xuất thông báo lỗi qua và thoát khỏi hàm.

A computer code with black and red text

Description automatically generated

Lặp qua từng chuỗi chain trong bảng table:

Lặp qua từng phần tử node trong chain: node có hai trường dữ liệu: key là chuỗi đại diện cho khóa, value là chuỗi đại diện cho giá trị. Lưu độ dài của khóa và giá trị: keyLen và valueLen chứa độ dài chuỗi key và value tương ứng. Tiếp theo là ghi dữ liệu vào tệp là ghi độ dài của key (sizeof(int) byte), ghi nội dung của key (dùng node.key.c\_str() để lấy con trỏ C-string), ghi độ dài của value và nội dung của value tương tự.



Sau khi ghi xong thì sẽ đóng tệp

**g. Hàm load**

Hàm load có nhiệm vụ **tải dữ liệu từ tệp nhị phân** (file binary) và đưa vào bảng băm của từ điển.

Dùng ifstream để mở tệp filename để đọc dữ liệu. Chế độ ios::binary đảm bảo rằng dữ liệu được đọc dưới dạng nhị phân, không phải văn bản. Nếu tệp không mở được (có thể do không tồn tại hoặc lỗi), thông báo lỗi được in ra (Khong the mo tep !).

Dùng vòng lặp while chạy liên tục miễn là tệp còn dữ liệu để đọc. Qua mỗi lần lặp sẽ đọc một cặp key-value từ tệp.

Hai biến keyLen và valueLen dùng để lưu độ dài của key và value sẽ được đọc từ tệp.

Dùng file.read() để đọc sizeof(keyLen) từ tệp và gán vào biến keyLen. keyLen cho biết số ký tự của key.

Chuỗi key ban đầu được khởi tạo với keyLen ký tự khoảng trắng (' ').

Sử dụng file.read() để đọc keyLen ký tự từ tệp và gán chúng vào key.

Dùng file.read() để đọc sizeof(valueLen) từ tệp và gán vào biến valueLen. valueLen cho biết số ký tự của value.

Chuỗi value ban đầu được khởi tạo với valueLen ký tự khoảng trắng (' ').

Sử dụng file.read() để đọc valueLen ký tự từ tệp và gán chúng vào value.

Sau khi đọc được key và value, chúng được thêm vào bảng băm bằng cách gọi hàm insert.

Khi việc đọc tệp hoàn tất, tệp được đóng bằng file.close() để giải phóng tài nguyên.

In ra thông báo cho biết dữ liệu đã được tải thành công từ tệp.

**A screenshot of a computer program

Description automatically generated**

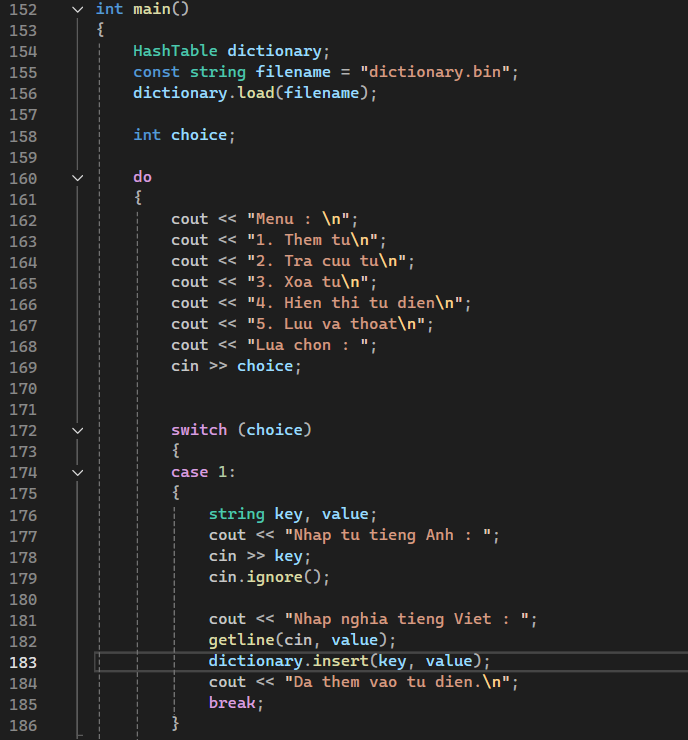
**h. Hàm main**

Khởi tạo đối tượng HashTable là dictionary

Gọi hàm load : Mở file dictionary.bin rồi đọc dữ liệu từ file để thêm vào dictionary nếu có.

* Tạo vòng lặp menu :
* Lặp cho đến khi người dùng chọn thoát (lựa chọn 5).
* Hiển thị menu với các tùy chọn.
* Nhận đầu vào từ người dùng (choice) để thực hiện hành động tương ứng

Dùng switch-case để xử lý các lựa chọn, tương ứng với mỗi lựa chọn, thực hiện các chức năng khác nhau bằng các hàm tương ứng đã tạo



A screen shot of a computer program

Description automatically generated

**V-Bài 5**

**Minh hoạ bài 1 :**

**5.1.Ý tưởng tổng quát**

Mục tiêu:

Xây dựng một chương trình minh họa thuật toán duyệt đồ thị theo chiều sâu (DFS) bằng cách vẽ đồ thị và thể hiện trực quan quá trình duyệt trên giao diện đồ họa được tạo bằng thư viện tkinter.

Chức năng chính:

* Người dùng nhập số đỉnh, danh sách cạnh của đồ thị, và đỉnh bắt đầu duyệt.
* Chương trình vẽ đồ thị (đỉnh và cạnh) trên màn hình.
* Sử dụng thuật toán DFS để duyệt đồ thị, minh họa quá trình duyệt bằng cách tô màu các đỉnh.
* Hiển thị kết quả duyệt trên giao diện người dùng.
* Có thể dừng lại và xem đỉnh stack hiện tại

Các bước thực hiện

1. Thiết kế giao diện: Sử dụng tkinter để xây dựng giao diện nhập liệu (số đỉnh, danh sách cạnh, đỉnh bắt đầu) và hiển thị kết quả.
2. Thuật toán DFS: Triển khai thuật toán DFS dựa trên danh sách kề với ngăn xếp để duyệt đồ thị.
3. Vẽ đồ thị: Các đỉnh được vẽ đều trên vòng tròn, các cạnh nối đỉnh được hiển thị, và đỉnh được tô màu khi duyệt kết hợp với các yếu tố như pause để bấm dừng, và xem đỉnh trong stack

**5.2. Giải thích code**

**a. Hàm dfs\_minhhoa**

**Mục đích:** Thực hiện thuật toán DFS và cập nhật giao diện.

Các bước thực hiện:

**Nhập dữ liệu đầu vào:** Dùng Entry và Text widget để lấy số đỉnh, danh sách cạnh, và đỉnh bắt đầu. Kiểm tra tính hợp lệ của dữ liệu. Kiểm tra xem đỉnh bắt đầu có nằm trong phạm vi hợp lệ hay không.

**A computer screen shot of a program code

Description automatically generated**

**Xây dựng danh sách kề:**

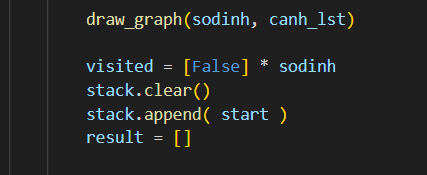
* Đầu tiên, nhập danh sách các cạnh vào, với dạng u v ( u và v là 2 đỉnh nối với nhau được cạnh ) và dùng split( “\n”) để tách mỗi dòng là một phần tử cạnh của danh sách.
* [ [] for \_ in range(sodinh) ]: Tạo một danh sách gồm sodinh phần tử, mỗi phần tử là một danh sách con rỗng ([])
* \_: Một biến giả, không cần đặt tên cụ thể, chỉ dùng để lặp đủ số lần (sodinh)
* Danh sách kề được xây dựng từ danh sách cạnh. Mỗi dòng nhập có dạng u v ( u và v là 2 đỉnh nối với nhau được cạnh ) được tách thành hai số, kiểm tra u và v có thoả không và thêm vào danh sách kề với đỉnh u thì đỉnh kề sẽ là v, với đỉnh v thì đỉnh kề sẽ là u.

A computer screen with colorful text

Description automatically generated with medium confidence

**Khởi tạo và chuẩn bị các cấu trúc dữ liệu cần thiết để duyệt đồ thị :**

* draw\_graph(sodinh, canh\_lst) : Vẽ đồ thị dựa trên số đỉnh (sodinh) và danh sách các cạnh (canh\_lst) mà người dùng cung cấp.
* visited = [False] \* sodinh : Tạo danh sách đánh dấu các đỉnh đã duyệt, ban đầu tất cả giá trị là False.
* Xóa mọi phần tử trong ngăn xếp (stack) trước khi bắt đầu duyệt DFS.
* Thêm đỉnh bắt đầu (start) vào ngăn xếp để bắt đầu duyệt DFS
* Tạo một danh sách result rỗng để lưu trữ thứ tự các đỉnh đã duyệt.



**Minh họa quá trình duyệt:**

* Thuật toán DFS sử dụng ngăn xếp để duyệt qua các đỉnh. Mỗi đỉnh được duyệt sẽ được tô màu đỏ để minh họa
* Kiểm tra xem ngăn xếp có trống hay không. Nếu stack trống, tức là đã duyệt xong tất cả các đỉnh có thể truy cập.
* Lấy phần tử trên cùng của ngăn xếp để xử lý (đỉnh đang được duyệt do ngăn xếp hoạt động theo nguyên tắc **LIFO** (Last In, First Out), tức là đỉnh được thêm vào sau cùng sẽ được xử lý trước )
* if not visited[current] : Kiểm tra xem đỉnh current đã được thăm chưa.Nếu chưa thì đánh dấu rằng đỉnh current đã được thăm ( visited[ current ] = True ). Rồi bỏ vào mảng result.
* Sau đó, dùng hàm highlight để tô màu đỏ đỉnh current để minh họa rằng đây là đỉnh đang được xử lý. Dùng canvas.update( ) để cập nhật giao diện đồ họa ngay lập tức để hiển thị đỉnh đang được tô màu.
* time.sleep(1.5) để tạm dừng chương trình để dễ dàng nhìn quá trình duyệt.
* Dùng hàm highlight tô màu xanh dương ngay cho nút vừa được xử lý xong.
* for neighbor in sorted(canhke\_lst[current], reverse=True) : Lặp qua tất cả các đỉnh kề của current để tìm các đỉnh chưa được duyệt
* Với canhke\_lst[ current ] là danh sách các đỉnh kề của current và sorted( canhke\_lst[current] , reverse=True ) sắp xếp danh sách các đỉnh kề theo thứ tự giảm dần để đảm bảo rằng khi duyệt các đỉnh, đỉnh có số nhỏ hơn sẽ được duyệt sau cùng (do sử dụng ngăn xếp LIFO). Lặp qua từng neighbor trong danh sách đã sắp xếp. Nếu neighbor chưa được thăm, thêm vào stack để xử lý sau.

A screen shot of a computer code

Description automatically generated

**Hiển thị kết quả:**

Hiển thị danh sách các đỉnh đã được duyệt theo thứ tự DFS trên giao diện.

map(str, result): Chuyển từng phần tử trong mảng result (số nguyên) thành chuỗi để hiển thị.

' '.join(...) : Nối các phần tử trong danh sách result bằng khoảng trắng, tạo ra chuỗi kết quả

result\_label.config (text = …..) : Cập nhật nội dung của nhãn (Label) result\_label trên giao diện với chuỗi kết quả.

for node in result: highlight(node, "blue") : Tô màu **xanh dương (blue)** cho tất cả các đỉnh trong danh sách result để biểu thị rằng chúng đã được duyệt hoàn tất.

Nếu có lỗi thì thông báo lỗi bằng cách hiển thị trên giao diện bằng messagebox

A black screen with white text

Description automatically generated

**b. Hàm tieptuc\_or\_pause**

**Mục đích :** ùng dể tạm ngưng việc duyệt

Khai báo biến toàn cục pause, giúp hàm có thể thay đổi giá trị của biến pause trong phạm vi toàn cục.

Đảo ngược giá trị hiện tại của pause

Thay đổi nhãn của nút pause\_button dựa trên trạng thái hiện tại

A computer screen with white text

Description automatically generated

**c. Hàm draw\_graph**

**Mục đích:** Vẽ đồ thị trên canvas.

Các bước thực hiện:

**Thiết lập giao diện chung :**

* Xóa toàn bộ các thành phần đã vẽ trên Canvas.
* Đảm bảo rằng mỗi lần vẽ lại đồ thị sẽ không bị chồng lặp bởi các thành phần cũ.
* Xóa sạch nội dung trong dictionary node\_positions để cập nhật vị trí mới cho nút. Xác định kích thước không gian vẽ, bán kính các đỉnh và toạ độ tâm.

A screen shot of a computer code

Description automatically generated

**Tính toán vị trí các đỉnh:**

* Sử dụng công thức lượng giác để đặt các đỉnh trên một vòng tròn. Các đỉnh được chia đều trên vòng tròn, với bán kính xác định trước.
* Tọa độ trung tâm của vòng tròn là (center\_x, center\_y). Bán kính là radius. Cần chia đều số đỉnh trên vòng tròn. Góc giữa 2 đỉnh = 2pi / số đỉnh, đỉnh thứ i sẽ nằm ở góc = góc giữa 2 đỉnh nhân với i.
* Từ đó tính toạ độ : x = toạ độ trung tâm + bán kính x cos( góc ) ; y = toạ độ trung tâm + bán kính x sin( góc )

Lưu vị trí của đỉnh vào node\_positions và dùng hàm draw\_node để vẽ

A black background with white text

Description automatically generated

**Vẽ các cạnh :** Kết nối các đỉnh theo danh sách cạnh

* Đầu tiên, duyệt qua từng cạnh trong danhsach\_canh. Tách chuỗi u v thành hai số nguyên u và v
* Lấy toạ độ của đỉnh u và v từ node\_positions rồi vẽ đường thẳng (cạnh) nối giữa đỉnh u và v bằng canvas.create\_line

A screen shot of a computer code

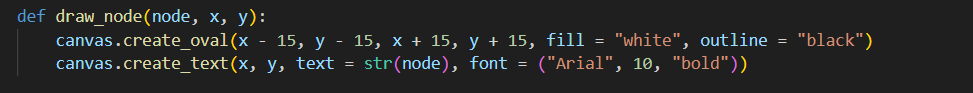
Description automatically generated

**d.Hàm draw\_node**

**Mục đích :** vẽ các hình tròn biểu diễn đỉnh

canvas.create\_oval : Vẽ hình tròn để biểu diễn đỉnh với (x - 15, y - 15, x + 15, y + 15). Xác định vùng bao quanh hình tròn với bán kính 15.

canvas.create\_text : Ghi số thứ tự của đỉnh (từ 0 đến sodinh - 1) ở giữa hình tròn.



**e. Hàm highlight(node, color)**

**Mục đích:** Tô màu đỉnh minh hoạ trạng thái duyệt

Đầu tiên, lấy toạ độ của đỉnh từ node\_positions cần tô màu

Vẽ lại hình tròn đại diện cho đỉnh bằng màu được chỉ định rồi hiển thị lại số thứ tự của đỉnh ở giữa hình tròn.

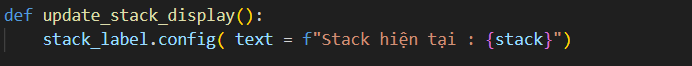
A black background with white text

Description automatically generated

**f.Hàm update\_stack\_display**

Cập nhật giao diện người dùng để hiển thị trạng thái hiện tại của stack

Nội dung hiển thị trong một Label giao diện sẽ thay đổi để phản ánh trạng thái của stack

****

**g.Hàm peek\_stack**

Hiển thị phần tử trên đỉnh của ngăn xếp (stack) cho người dùng.

Nếu ngăn xếp rỗng, thông báo "Stack hiện tại không có phần tử nào".

**A black background with white text

Description automatically generated**

**h. Thực hiện tạo giao diện Tkinter**

Đặt tiêu đề cho cửa sổ chính.

Tạo một khung (frame) để chứa các widget đầu vào. pack(pady=10): Thêm khoảng cách dọc (10px) phía trên và dưới frame1 để tạo sự tách biệt.

Trong frame1 :

* Hiển thị nhãn "Số đỉnh:". Cung cấp ô nhập liệu cho người dùng nhập số đỉnh. grid(row=0, column=1): Đặt ô nhập liệu tại hàng 0, cột 1.
* Hiển thị nhãn hướng dẫn "Danh sách các cạnh". Cung cấp vùng nhập liệu lớn để người dùng nhập danh sách cạnh, mỗi dòng có định dạng u v.
* Hiển thị nhãn "Đỉnh bắt đầu:". Cung cấp ô nhập liệu để người dùng nhập đỉnh bắt đầu duyệt DFS.
* Tạo nút nhấn với nội dung "Bắt đầu duyệt". Khi nhấn nút, hàm dfs\_minhhoa được gọi để thực hiện duyệt đồ thị.
* Tạo một nút hiển thị dòng chữ "Lấy đỉnh". Khi người dùng nhấn nút, hàm peek\_stack được gọi để hiển thị phần tử hiện tại trên đỉnh ngăn xếp.
* Tạo một nút hiển thị dòng chữ "Dừng". Khi người dùng nhấn nút, hàm tieptuc\_or\_pause được gọi để dừng hoặc tiếp tục quá trình duyệt DFS

A computer screen with text

Description automatically generated

A computer code on a black background

Description automatically generated

Trong frame2 :

* Tạo khung chứa cho canvas . Tạo vùng vẽ với kích thước 500 x 500, màu trắng.
* Tạo nhãn để hiển thị kết quả duyệt DFS (thứ tự các đỉnh đã duyệt).
* Tạo một từ điển trống node\_positions để lưu tọa độ của các đỉnh

A black background with colorful text

Description automatically generated

**Minh hoạ bài 2 :**

**5.3.Ý tưởng**

Mục tiêu : Xây dựng và minh họa trực quan hoạt động của cây nhị phân tìm kiếm với đối tượng là sách

Các thao tác chức năng :

* Thêm nút vào cây (đệ quy hoặc không đệ quy).
* Xóa nút khỏi cây (đệ quy hoặc không đệ quy).
* Tìm kiếm sách theo mã sách
* Bấm chuột vào nút nào hiển thị thông tin nút sách đó
* Hiển thị chiều cao hiện tại của cây.

**5.4.Nội dung**

**a.Lớp Sach:**

Gồm : mã sách, tên sách, tác giả, giá bán

A computer screen with white text

Description automatically generated

**b.Lớp Node:**

Biểu diễn một nút trong BST:

key: Thông tin sách

left, right: bên trái và phải của nút

A black background with white text

Description automatically generated

**c.Lớp BST:**

Chứa toàn bộ các phương thức xử lý BST

**Thêm nút vào cây (đệ quy)** : Thêm một nút mới vào BST

Điều kiện dừng : Nếu root is None, nghĩa là đã đến vị trí trống trong cây, tạo nút mới tại đây và trả về nút đó.

So sánh mã sách của nút mới với nút gốc : Nếu key < root.key: Gọi đệ quy với cây con bên trái , nếu > thì đệ quy cây con bên phải.

Sau khi thêm nút, cập nhật cây con trái hoặc cây con phải của root.

A computer screen shot of a code

Description automatically generated

**Thêm nút không dùng đệ quy :**

Tạo một nút mới new\_node với thông tin sách cần thêm.

Nếu cây rỗng (root is None), gán nút mới làm gốc.

Tìm vị trí thích hợp: Duyệt cây bằng vòng lặp, so sánh mã sách :

* Nếu nhỏ hơn, duyệt xuống cây con bên trái.
* Nếu lớn hơn, duyệt xuống cây con bên phải.
* Nếu bằng, kết thúc (không thêm).

Gán new\_node làm con trái hoặc con phải của nút cha (parent) tùy theo kết quả so sánh.

A screen shot of a computer program

Description automatically generated

**Xóa nút khỏi cây (đệ quy) :** Xóa một nút khỏi BST.

Điều kiện dừng : Nếu root is None, trả về None (cây rỗng).

Tìm nút cần xóa : Nếu key < root.key.maSach: Gọi đệ quy với cây con bên trái. Nếu > thì gọi đệ quy cây con phải.

Xử lý 3 trường hợp: Nút không có con thì rả về None. Nút có một con thì trả về con của nút cần xóa. Nút có hai con thì Tìm nút nhỏ nhất trong cây con bên phải (successor) rồi thay thế giá trị nút cần xóa bằng giá trị của successor, sau đó, gọi đệ quy để xóa successor.

A screen shot of a computer program

Description automatically generated

**Xóa nút không dùng đệ quy :**

Tìm nút cần xóa: Duyệt cây để tìm nút có khóa bằng key. Lưu cha của nút vào biến parent.

Nút cần xóa có 0 hoặc 1 con: Gán con của nút vào vị trí của nút đó (liên kết với cha).

Nút cần xóa có 2 con: Tìm nút kế nhiệm nhỏ nhất trong cây con bên phải (successor). Thay thế nút cần xóa bằng successor. Xóa nút successor khỏi cây.

Trả về cây sau khi xóa.

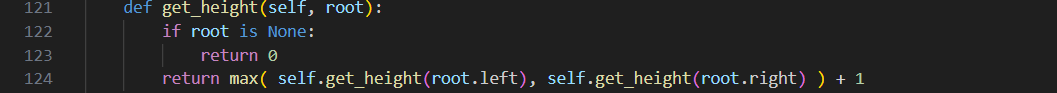
A computer screen shot of a program

Description automatically generated

**Hàm get\_height**

Dùng đệ quy tính chiều cao cây con trái và cây con phải.

Trả về chiều cao của cây = 1 + max ( chiều cao trái, chiều cao phải )



**Hàm search :**

Nếu root=None, cây rỗng hoặc không tìm thấy → trả về False.

Nếu key == root.key.maSach: Tìm thấy → trả về True. Nếu < thì gọi đệ quy vào cây con trái. Nếu > gọi đệ quy vào cây con phải.

A screen shot of a computer code

Description automatically generated

**d.Lớp giaodienBST** : Cung cấp giao diện đồ họa để thao tác với cây BST.

**Hàm \_\_init\_\_ :** Khởi tạo cửa sổ giao diện và các thành phần cần thiết cho chương trình.

self.master : Là cửa sổ chính của ứng dụng, được truyền vào từ Tk()

self.bst : Là đối tượng quản lý cây nhị phân tìm kiếm (lớp BST).

self.canvas : Vùng vẽ để hiển thị cây BST, với kích thước 800 x 600, nền trắng.

Tạo nút thêm sách đệ quy : gọi hàm add\_node với tham số True

Tạo nút thêm sách không đệ quy : gọi hàm add\_node với tham số False

Tạo nút xoá sách đệ quy : gọi hàm delete\_node với tham số True

Tạo nút xoá sách không đệ quy : gọi hàm delete\_node với tham số False

Tạo nút tìm kiếm : gọi hàm search\_book

Tạo nút hiển thị chiều cao cây : gọi hàm show\_height

"<Button-1>" : nhấn chuột trái vào nút, và phương thức self.show\_node\_info sẽ được thực hiện

A computer code on a black background

Description automatically generated

**Hàm draw\_tree**

Xác định màu của nút dựa trên giá trị highlight. Lưu tọa độ (x, y) của nút trên Canvas và giá trị dữ liệu của nó (root.key) vào dictionary self.node\_positions.

Vẽ nút : Dùng create\_oval để vẽ hình tròn đại diện cho nút với bán kính 20. Ghi mã sách (maSach) trong nút bằng create\_text.

Vẽ nhánh : Nếu có cây con trái : Tính tọa độ của cây con trái, rồi vẽ đường nối từ nút hiện tại đến cây con trái (create\_line). Nếu có cây con phải : thì làm tương tự nhưng với cây con phải.

Gọi đệ quy với hàmdraw\_tree để vẽ cây con trái và cây con phải, với độ lệch ngang (dx) giảm dần.

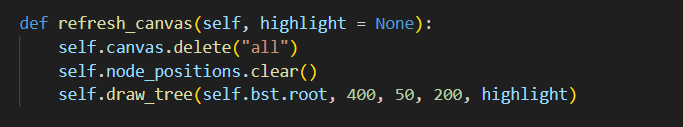
A computer screen shot of text

Description automatically generated

**Hàm refresh\_canvas :** Xóa toàn bộ canvas và vẽ lại cây hiện tại.

Dùng delete("all") để xóa toàn bộ nội dung.

Gọi draw\_tree để vẽ cây bắt đầu từ gốc (self.bst.root).



**Hàm add\_node**

Sử dụng hộp thoại simpledialog để nhập mã sách, tên sách, tác giả, và giá bán.

Nếu bất kỳ thông tin nào bị bỏ trống, hiển thị thông báo hủy thao tác.

Tạo nút : Nếu dequy=True thì gọi insert\_dequy. Nếu là False thì gọi insert\_khong\_dequy.

Gọi refresh\_canvas để vẽ lại cây.

A screen shot of a computer code

Description automatically generated

**Hàm delete\_node**

Sử dụng hộp thoại simpledialog để nhận mã sách cần xóa

Kiểm tra sự tồn tại, nếu sách không tồn tại, hiển thị báo lỗi.

Khi xoá, gọi delete\_dequy hoặc delete\_khongdequy tùy thuộc vào giá trị dequy, True thì delete\_dequy, False thì delete\_khongdequy

Gọi refresh\_canvas để vẽ lại cây.

A screen shot of a computer code

Description automatically generated

**Hàm search\_book :**

Hiển thị hộp thoại để người dùng nhập mã sách cần tìm.

Gọi phương thức search trên cây BST để tìm nút có mã sách bằng maSach.

Hiển thị thông tin sách tìm thấy

Tô màu đỏ nút tương ứng.

Nếu result là None (không tìm thấy sách), hiển thị thông báo lỗi.

A black screen with colorful text

Description automatically generated

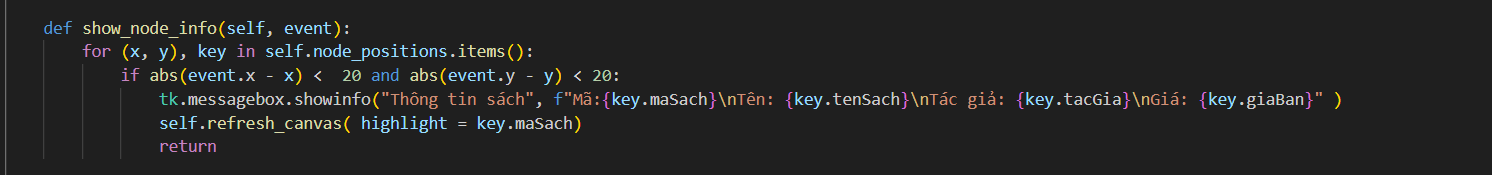
**Hàm show\_node\_info**

Duyệt qua tất cả các nút đã vẽ trên giao diện.

Kiểm tra xem tọa độ nhấp chuột (event.x, event.y) có nằm trong bán kính 20 pixel quanh nút tại (x, y) hay không.

Hiển thị thông tin sách trong một hộp thoại (messagebox).

Làm mới giao diện Canvas và tô màu đỏ nút tương ứng



**Hàm show\_height :**

Dùng hàm get\_height để lấy chiều cao, rồi dùng tk.messagebox.showinfo để hiển thị chiều cao.

