**BỘ GIÁO DỤC VÀ ĐÀO TẠO**



**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM KHOA CÔNG NGHỆ THÔNG TIN**

🙞🙞🙜🙜

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



**GVHD: TS. HUỲNH XUÂN PHỤNG**

**Đề Tài Cuối Kỳ**

**MÔ PHỎNG HÌNH ẢNH BIỂU DIỄN CÂY NHỊ PHÂN THEO DẠNG THREADED VỚI 3 CÁCH DUYỆT CÂY, TÌM KIẾM VÀ CHÈN NODE VÀO CÂY THREADED**

**Mã môn học: DASA230179\_22\_1\_08**

**NHÓM SỐ: 7**

**TÊN THÀNH VIÊN NHÓM-MSSV:**

Bùi Thiên Kim 21110521

Nguyễn Quỳnh My 21110548

Phạm Khương Tâm 21110638

**HỌC KỲ 1 - NĂM HỌC: 2022-2023**

**DANH SÁCH NHÓM**

**HỌC KÌ I, NĂM HỌC 2022 – 2023**

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | Mssv | Tỉ lệ hoàn thành |
| 1 | Bùi Thiên Kim | 21110521 | 100% |
| 2 | Nguyễn Quỳnh My | 21110548 | 100% |
| 3 | Phạm Khương Tâm | 21110638 | 100% |

***Nhận xét của giáo viên***

*Ngày 28 tháng 12 năm 2022*

**Giáo viên chấm điểm**

**GV.TS. Huỳnh Xuân Phụng**

**Mục lục**

[**MỞ ĐẦU** 1](#_Toc123142526)

[**1. Lý do chọn đề tài:** 1](#_Toc123142527)

[**2. Mục tiêu nghiên cứu:** 1](#_Toc123142528)

[**3. Phương pháp nghiên cứu:** 1](#_Toc123142529)

[**CƠ SỞ LÝ THUYẾT** 2](#_Toc123142530)

[**NỘI DUNG** 3](#_Toc123142531)

[**1.** **Khởi tạo cấu trúc NODE** 3](#_Toc123142532)

[**2.** **Xây dựng Threaded binary tree** 3](#_Toc123142533)

[**3.** **Tìm kiếm** 5](#_Toc123142534)

[**4.** **Duyệt cây:** 5](#_Toc123142535)

# **MỞ ĐẦU**

## **1. Lý do chọn đề tài:**

Chúng ta biết mỗi nút của cây nhị phân lưu trữ giá trị dữ liệu của nút và con trỏ địa chỉ của nút con trái và nút con phải. Nếu một nút trong cây nhị phân không có nút con trái hoặc phải hoặc là nút lá, thì nút con sẽ được biểu thị bằng một con trỏ null.

Nhiều nút hiện diện trong cây này giữ giá trị null trong con trỏ con bên trái hoặc bên phải của chúng. Không gian bị chiếm bởi các giá trị null này có thể được sử dụng để lưu trữ một số loại thông tin có giá trị.

Một cách khả thi để sử dụng không gian này là có một con trỏ đặc biệt trỏ tới các nút cao hơn trong cây (tức là tổ tiên). Những con trỏ đặc biệt này được gọi là threaded và cây nhị phân có các con trỏ như vậy được gọi là threaded binary tree.

Threaded binary tree là một biến thể của Cây nhị phân thông thường, nó tạo điều kiện cho việc duyệt cây nhanh hơn và không yêu cầu Ngăn xếp hoặc Đệ quy. Nó làm giảm sự lãng phí bộ nhớ bằng cách đặt các con trỏ null của nút lá thành nút tiền nhiệm theo thứ tự hoặc nút kế tiếp theo thứ tự.

## **2. Mục tiêu nghiên cứu:**

Nhóm sẽ nghiên cứu sự vận hành, chức năng của một Threaded binary tree. Từ đó, bằng kiến thức cấu trúc dữ liệu và giải thuật sẽ xây dựng mô phỏng quá trình chèn 1 node vào cây, quá trình duyệt cây theo 3 cách và quá trình tìm kiếm node dựa trên những gì mà nhóm thu thập được.

## **3. Phương pháp nghiên cứu:**

Phương pháp phân loại, hệ thống hóa kiến thức: thông qua các tài liệu thu thập được nhằm hệ thống và sắp xếp các tài liệu khoa học theo chủ đề, theo đơn vị kiến thức để dễ dàng nghiên cứu.

Phương pháp thu thập, phân tích tài liệu: thông qua các tài liệu trên internet, sách tham khảo, các bài báo cáo để làm rõ nội dung của mục tiêu nghiên cứu.

# **CƠ SỞ LÝ THUYẾT**

Threaded binary tree là gì?

Trong Cây nhị phân có luồng, các nút sẽ lưu trữ tiền thân/kế thừa theo thứ tự thay vì lưu trữ NULL trong các con trỏ con trái/phải.

Vì vậy, ý tưởng cơ bản của cây nhị phân luồng là đối với các nút có con trỏ bên phải là null, chúng tôi lưu trữ phần kế tiếp theo thứ tự của nút (nếu tồn tại) và đối với các nút có con trỏ bên trái là null, chúng tôi lưu trữ phần tử kế tiếp theo thứ tự của nút đó. -thứ tự tiền thân của nút (nếu tồn tại).

Một điều cần lưu ý là con trỏ con ngoài cùng bên trái và ngoài cùng bên phải của cây luôn trỏ tới null vì người tiền nhiệm và người kế vị theo thứ tự của chúng không tồn tại.

# **NỘI DUNG**

* + - 1. **Khởi tạo cấu trúc NODE**

Trong Threaded binary tree con trỏ null bên trái của một nút được tạo để trỏ tới nút tiền nhiệm theo thứ tự và con trỏ null bên phải được tạo để trỏ tới nút kế tiếp theo thứ tự.

Cấu trúc nút của cây nhị phân hai luồng:  Đối với cây nhị phân hai luồng, chúng tôi sử dụng hai biến boolean:  *RTAG* và *LTAG*.

struct NODE {

    bool LTAG, RTAG;

    int INFO;

    int level;

    NODE\* LLINK;

    NODE\* RLINK;

    int x, y;

}\*root;

Ở đây, các biến boolean *LTAG* và *RTAG*giúp chúng ta phân biệt liệu con trỏ trái/phải lưu trữ tiền thân/kế thừa theo thứ tự hay con trái/con phải.

* + - 1. **Xây dựng Threaded binary tree**

- Trước hết chúng ta xây dựng một cây Threaded rỗng

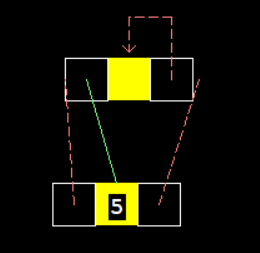
void init() {

        root = nullptr;

    }

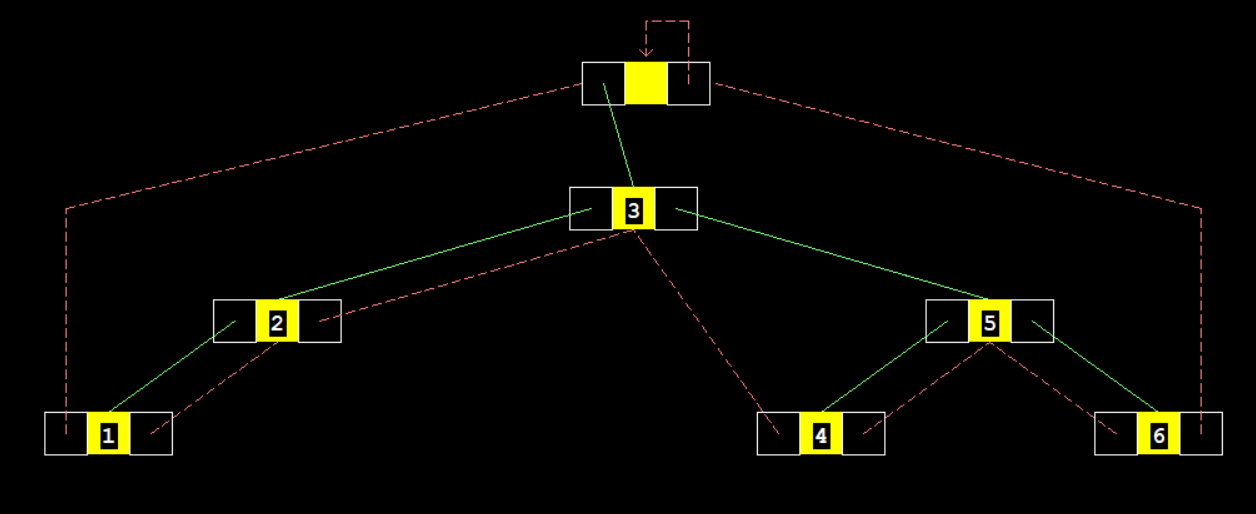
- Sau khi đã xây dựng được một cây Threaded rỗng, chúng ta cần xác định được thông tin của 1 node bất kỳ của cây bao gồm các thông tin cấp bậc, thông tin chứa trong node, vị trí node trong cây nên xây dựng hàm NodeInfo (32-47).

- Để thực hiện việc mô phỏng cây Threaded, chúng ta cần vẽ các node của cây với 2 hàm DrawNodeColor (50-66) – để vẽ màu và ghi thông tin mà các node mang và hàm DrawSquareNode (69-153) – để vẽ các node của cây Threaded.



- Để thuận tiện cho việc so sánh giá trị được chứa trong các node thì hàm CompareNode (156-189) giúp chúng ta dễ dàng hơn trong việc so sánh sau này. Chúng ta sẽ lấy key cần so sánh đem so sánh với các giá trị được chứa trong các node.

- Xây dựng hàm MoveNode (192-226) đây là hàm giúp di chuyển các node. Để thực hiện được di chuyển node chúng ta cần xác định vị trí đích mà node cần được chuyển đến vì vậy chúng ta cần dùng hàm NodeInfo ở trên để giúp xác định vị trí node cần chuyển đến.

****- Hàm DrawThreadTree(229-246) là hàm thực hiện vẽ Threaded binary tree. Tại đây hàm vẽ node threaded là DrawSquareNode (đã xây dựng bên trên) để thực hiện vẽ và nối các node lại với nhau tạo thành Threaded binary tree hoàn chỉnh.

*(hình ảnh minh họa)*

- Xây dựng hàm insert(249-311) để thực hiện thêm node vào cây theo quy trình:

1. Để chèn một nút, nhiệm vụ đầu tiên của chúng ta là tìm vị trí để chèn nút.

2. Đầu tiên, kiểm tra xem cây có trống không, nghĩa là cây chỉ có một nút giả, sau đó chèn nút mới vào cây con bên trái của nút giả.

3. Nếu cây không rỗng thì tìm chỗ chèn nút vào, giống như trong BST bình thường.

4. Nếu nút mới nhỏ hơn hoặc bằng nút hiện tại thì kiểm tra xem *LTAG* = true, nếu có thì ta đã tìm được vị trí chèn nút, nút sẽ nằm bên trái cây con và nếu *LTAG* = false thì sang trái .

5. Nếu nút mới lớn hơn nút hiện tại thì kiểm tra xem *RTAG* = true, nếu đúng thì ta đã tìm được vị trí chèn nút vào, nó sẽ nằm bên phải cây con còn nếu *RTAG* = false thì sang phải.

6. Lặp lại các bước 4 và 5 cho đến khi không tìm thấy vị trí được chèn.

7. Khi đã quyết định vị trí nút sẽ được chèn, nhiệm vụ tiếp theo sẽ là chèn nút. đầu tiên, chúng ta sẽ xem nút sẽ được chèn vào nút con bên trái như thế nào.

Đồng thời kết hợp với các hàm DrawNodeColor và MoveNode để thực hiện việc mô phỏng quá trình thêm 1 node mới, kiểm tra node, di chuyển node vào vị trí của nó trong cây Threaded.

- Hàm CheckInput(562-567) và hàm Validate(570-574) để kiểm tra giá trị nhập vào từ bàn phím có phù hợp với yêu cầu để chứa trong node hay không. Giúp bổ trợ quá quá trình nhập dữ liệu trong hàm main.

* + - 1. **Tìm kiếm**

- Hàm search(314-342) chúng ta kiểm tra cả hai biến LTAG và RTAG để xác định xem key cần tìm có bằng giá trị tồn tại trong node hay không. Quy trình tìm kiếm có thể được tóm tắt như sau:

1. Đi qua cây từ gốc và so sánh key với khóa của từng node dọc theo cây

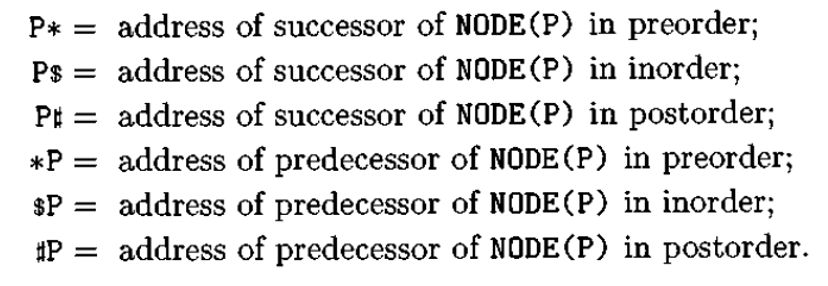
2. Nếu một khóa khớp, xuất ra đã tìm thấy node.

3. Nếu không có khóa nào khớp khi LTAG hoặc RTAG là True, thì nút đó không tồn tại trong cây.

Việc sử dụng hàm DrawNodeColor giúp cho việc quan sát quá trình tìm kiếm dễ dàng hơn. Cây sẽ được kiểm tra lần lượt từ node gốc đến các node con để tìm kiếm node con chứa giá trị key cần tìm. Khi duyệt toàn bộ cây mà không tìm thấy thì sẽ xuất ra màn hình BGI dòng “not found”. Khi duyệt cây xuất hiện node có giá trị bằng cây cần tìm thì sẽ xuất ra màn hình BGI “found” và đường đi đến vị trí node cần tìm được tô màu xanh sáng.

* + - 1. **Duyệt cây:**

Chúng ta duyệt cây theo 3 dạng Inorder, Preorder, Postorder:

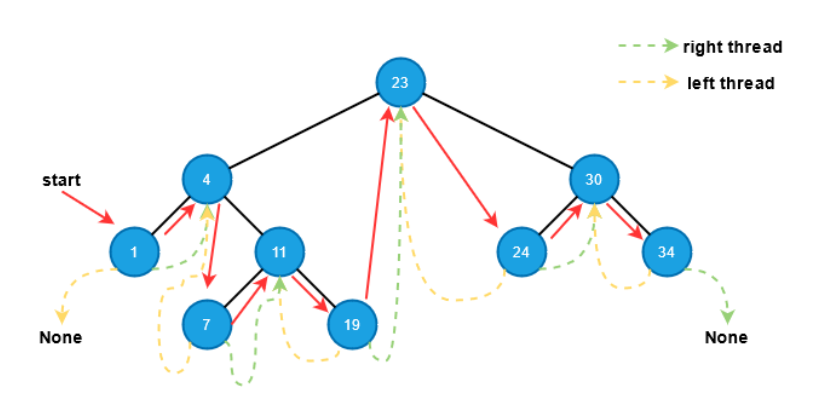


**+ Duyệt cây theo *In-Order* (trái-gốc-phải)** (344-408)

1. Bắt đầu từ nút ngoài cùng bên trái của toàn bộ cây.

2. Nếu node bên phải là Thread, xét theo thuộc tính bên phải; nếu node bên phải không phải là một Thread, hãy chuyển đến nút ngoài cùng bên trái của cây con.

3. Lặp lại bước 2 cho đến khi node bên phải là None.



*(hình ảnh minh họa)*

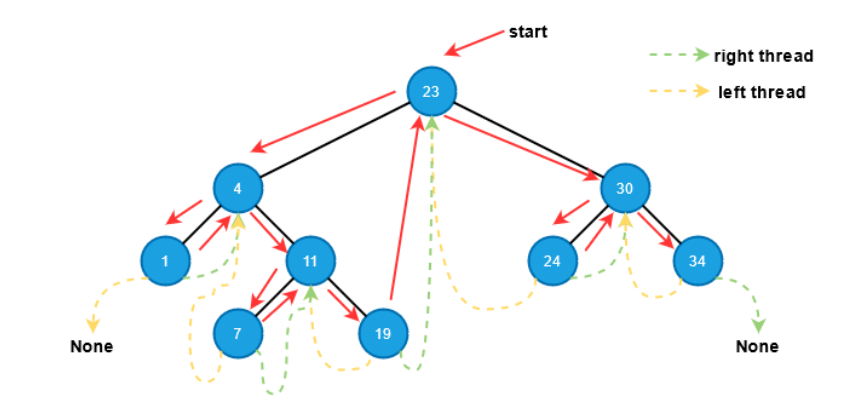
**+ Duyệt cây theo *Preorder*(gốc-trái-phải)** (410-468)

1. Bắt đầu từ gốc

2. Nếu node bên trái không trống, hãy chuyển đến phần con bên trái.

3. Nếu thuộc tính bên trái trống hoặc một thread, đi sang bên phải.

4. Lặp lại bước 2 và 3 cho đến khi node bên phải là None.



**+ Duyệt cây theo *Postorder*(trái-phải-gốc)** (470-558)

Việc duyệt cây theo Postorder sẽ gây ra rất nhiều khó khăn bởi vì trong một cây thì node kế tiếp theo Postorder sẽ không bao giờ ở bên dưới nó trong cây( node con bên phải duyệt trước node cha). Vì vậy việc sử dụng duyệt theo thứ tự ngược sẽ dễ dàng hơn:

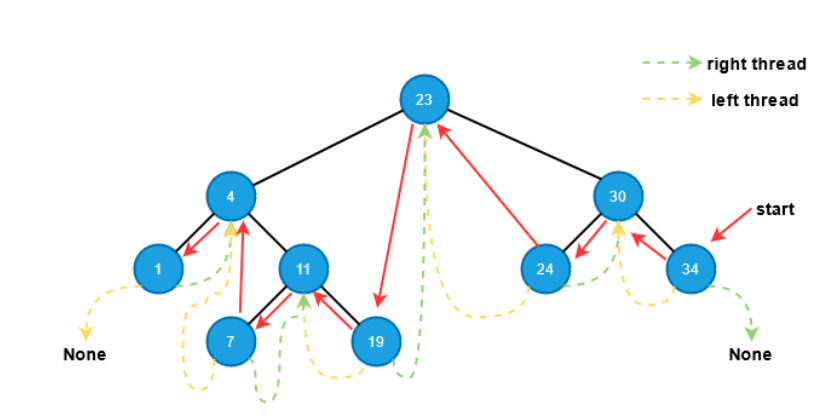
1. Bắt đầu từ gốc

2. Nếu node bên phải không trống, hãy chuyển đến phần con bên phải.

3. Nếu thuộc tính bên phải trống hoặc một thread, đi sang bên trái.

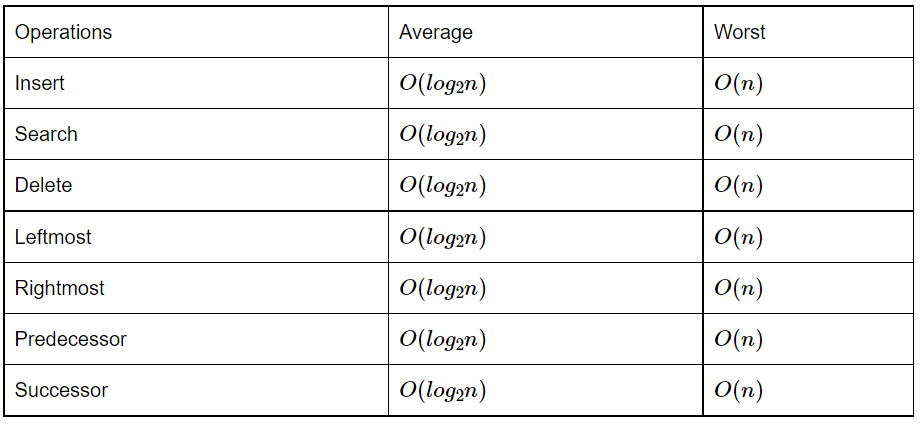
4. Lặp lại bước 2 và 3 cho đến khi node bên trái là None.

5. Xuất ngược lại dãy đã duyệt.

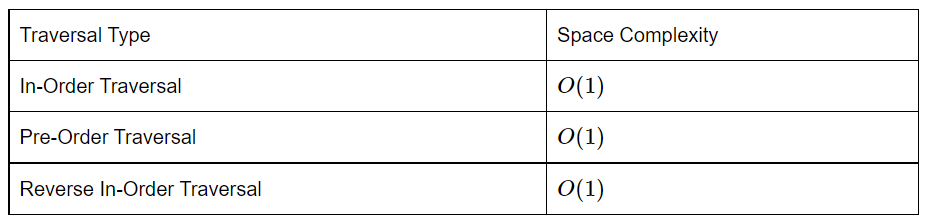


*(Hình ảnh minh họa quá trình duyệt ngược)*

**Kết quả thực hiện:**



Độ phức tạp không gian của các lần duyệt:



**Ưu, nhược điểm của thuật toán:**

**- Ưu điểm:**

+ Không cần ngăn xếp hoặc đệ quy: Không giống như cây nhị phân, cây nhị phân theo luồng không yêu cầu ngăn xếp hoặc đệ quy cho quá trình truyền tải của chúng.

+ Sử dụng bộ nhớ tối ưu: Một ưu điểm khác của cấu trúc dữ liệu cây nhị phân luồng là nó làm giảm lãng phí bộ nhớ. Trong cây nhị phân bình thường, bất cứ khi nào con trỏ trái/phải của nút là NULL, bộ nhớ sẽ bị lãng phí. Nhưng với các cây nhị phân theo luồng, chúng tôi đang khắc phục vấn đề này bằng cách lưu trữ tiền thân/kế thừa theo thứ tự của nó.

+ Độ phức tạp về thời gian: Quá trình truyền tải theo thứ tự trong cây nhị phân luồng diễn ra nhanh chóng vì nhận được nút tiếp theo trong thời gian O(1) so với cây nhị phân thông thường mất O(Chiều cao). Nhưng thao tác chèn và xóa mất nhiều thời gian hơn đối với Threaded binary tree.

+ Duyệt ngược: Trong cây nhị phân hai luồng, chúng ta thậm chí có thể thực hiện duyệt ngược.

**- Nhược điểm:**

+ Chèn và xóa phức tạp: Bằng cách lưu trữ nút tiền nhiệm/kế thừa theo thứ tự cho nút bằng con trỏ trái/phải null, chúng ta thực hiện việc chèn và xóa nút tốn nhiều thời gian hơn và là một quy trình rất phức tạp.

+ Sử dụng bộ nhớ bổ sung: Chúng tôi sử dụng bộ nhớ bổ sung dưới dạng các biến RTAG và LTAG để phân biệt giữa một chuỗi với một liên kết thông thường. (Tuy nhiên, có nhiều phương pháp hiệu quả hơn để phân biệt giữa một luồng và một liên kết thông thường).