**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO TIỂU LUẬN MÔN HỌC**

**TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI:**

**XÂY DỰNG CHƯƠNG TRÌNH MINH HỌA TỪNG BƯỚC CÁC THUẬT TOÁN TRONG MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT.**

Giảng viên hướng dẫn: Ths. Nguyễn Đình Hiển

Nhóm thực hiện: Nhóm 13

VÕ VĂN KHA 5951071040

TRẦN THỌ HIỀN 5951071026

NGUYỄN CÔNG HẬU 5951071025

LÊ NGUYỄN THÁI KHANG 5951071043

NGUYỄN HOÀNG NHẬT 5951071069

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá :K59

Tp. Hồ Chí Minh, năm 2022

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO TIỂU LUẬN MÔN HỌC**

**TRÍ TUỆ NHÂN TẠO**

**ĐỀ TÀI:**

**XÂY DỰNG CHƯƠNG TRÌNH MINH HỌA TỪNG BƯỚC CÁC THUẬT TOÁN TRONG MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT.**

Giảng viên hướng dẫn: Ths. Nguyễn Đình Hiển

Nhóm thực hiện: Nhóm 13

VÕ VĂN KHA 5951071040

TRẦN THỌ HIỀN 5951071026

NGUYỄN CÔNG HẬU 5951071025

LÊ NGUYỄN THÁI KHANG 5951071043

NGUYỄN HOÀNG NHẬT 5951071069

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá :K59

Tp. Hồ Chí Minh, năm 2022

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI **PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**CỘNG HÒA XÃ HỘI CHỦ NGHIÃ VIỆT NAM**

Độc lập – Tự do – Hạnh phúc

**THIẾT KẾ TỔNG QUAN ĐỀ TÀI**

Sinh viên thực hiện:

VÕ VĂN KHA 5951071040

TRẦN THỌ HIỀN 5951071026

NGUYỄN CÔNG HẬU 5951071025

LÊ NGUYỄN THÁI KHANG 5951071043

NGUYỄN HOÀNG NHẬT 5951071069

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : K59

**1. Tên đề tài:** Xây dựng chương trình minh họa từng bước các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật.

**2. Mục đích, yêu cầu:** Xây dựng được chương trình minh họa biểu diễn từng bước các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật.

**3. Nội dung và phạm vi đề tài:** Giới hạn trong 3 thuật toán

* Các thuật toán sắp xếp (ít nhất 3 thuật toán)
* Các thủ tục của cấu trúc danh sách liên kết: AddHead, AddTail, AddAfterQ, Duyệt, DeleteHead.
* Các thủ tục của cấu trúc Cây nhị phân tìm kiếm

**4. Công nghệ, công cụ và ngôn ngữ lập trình:**

* Microsoft Visual Studio 2019
* Ngôn ngữ lập trình C#

**5. Các kết quả chính dự kiến sẽ đạt được và ứng dụng:**

* Xây dựng được chương trình hỗ trợ học tập nghiên cứu 3 thuật toán thông dụng trong môn Cấu trúc dữ liệu và Giải thuật.

**LỜI CẢM ƠN**

Trong thời gian làm đồ án tốt, nhóm em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô và bạn bè.

Em xin gửi lời cảm ơn chân thành đến Th.S Nguyễn Đình Hiển, giảng viên Bộ môn Công Nghệ Thông Tin - trường ĐH Giao Thông Vận Tải Phân Hiệu Tại TP.HCM người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình làm đồ án.

Em cũng xin chân thành cảm ơn các thầy cô giáo trong trường ĐH Giao Thông Vận Tải Phân Hiệu Tại TP.HCM nói chung, các thầy cô trong Bộ môn Công Nghệ Thông Tin nói riêng đã dạy dỗ cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập.

Cuối cùng, em xin chân thành cảm ơn gia đình và bạn bè, đã luôn tạo điều kiện, quan tâm,giúp đỡ, động viên em trong suốt quá trình học tập và hoàn thành đồ án.

**NHẬN XÉT CỦA GIẢNG VIÊN**

|  |
| --- |
| ***Tp. Hồ Chí Minh, ngày ….… tháng ….… năm ….…***  **Giảng viên** |

**MỤC LỤC**

*[THIẾT KẾ TỔNG QUAN ĐỀ TÀI 1](#_Toc30855)*

*[LỜI CẢM ƠN 2](#_Toc14413)*

*[NHẬN XÉT CỦA GIẢNG VIÊN 3](#_Toc6843)*

*[MỤC LỤC 4](#_Toc20324)*

*[BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ 5](#_Toc6651)*

*[PHÂN CÔNG NHIỆM VỤ 6](#_Toc19246)*

*[CHƯƠNG 1. MỞ ĐẦU 7](#_Toc25761)*

*[1.1 Tổng quan 7](#_Toc12632)*

*[1.1.1. Lý do chọn đề tài 7](#_Toc24432)*

*[1.1.2. Yêu cầu 7](#_Toc14880)*

*[1.2. Phạm vi 7](#_Toc22034)*

*[CHƯƠNG 2. TÌM HIỂU VỀ CÁC THUẬT TOÁN TRONG MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 8](#_Toc11623)*

*[2.1. THUẬT TOÁN SẮP XẾP 8](#_Toc22619)*

*[2.1.1. Sắp xếp nổi bọt 8](#_Toc27992)*

*[2.1.2. Sắp xếp chèn 10](#_Toc12998)*

*[2.1.3. Sắp xếp chọn 12](#_Toc17276)*

*[2.2. Các thủ tục của cấu trúc danh sách liên kết: 14](#_Toc25550)*

*[2.2.1. Thêm vào đầu (AddHead) 14](#_Toc27434)*

*[2.2.2. Thêm vào cuối (AddTail) 15](#_Toc15629)*

*[2.2.3.AddAfterQ 16](#_Toc27827)*

*[2.2.4. Xóa đầu (deleteHead) 17](#_Toc5698)*

*[2.2.5. Duyệt (Traverse) 18](#_Toc14017)*

*[2.3. Các thủ tục của cấu trúc Cây nhị phân tìm kiếm 18](#_Toc27061)*

*[2.3.1.Thêm phần tử vào cây tìm kiếm nhị phân 19](#_Toc4020)*

*[2.3.2.Tìm kiếm trên BST 20](#_Toc11411)*

*[2.3.3.Xóa Node trong BST 21](#_Toc1810)*

*[2.3.3.1.Node cần xóa là Node lá(không có child nào cả) 21](#_Toc2862)*

*[2.3.3.2. Node cần xóa có 1 child 22](#_Toc3904)*

*[2.3.3.3. Node cần xóa có đủ 2 child 22](#_Toc12364)*

*[2.3.4.Duyệt cây tìm kiếm nhị phân 23](#_Toc27154)*

*[2.3.4.1. Duyệt PreOrder 24](#_Toc14316)*

*[2.3.4.2. Duyệt InOrder 24](#_Toc9993)*

*[2.3.4.3. Duyệt PostOrder 25](#_Toc12581)*

**BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ**

*[Hình: 1 . Ảnh minh họa cấu trúc danh sách liên kết 14](#_Toc2888)*

*[Hình: 2 . Ảnh mô phỏng AddHead 15](#_Toc1339)*

*[Hình: 3 . Ảnh mô phỏng AddTail 16](#_Toc4179)*

*[Hình: 4 . Ảnh mô phỏng AddAfterQ 17](#_Toc11889)*

*[Hình: 5 . Các bước thêm phần tử vào cây nhị phân 20](#_Toc6772)*

*[Hình: 6 . Các bước tìm kiếm trên cây nhị phân tìm kiếm 21](#_Toc32723)*

*[Hình: 7 . Xóa node lá trên cây nhị phân tìm kiếm 22](#_Toc27613)*

*[Hình: 8 . Xóa node có 1 child trên cây nhị phân tìm kiếm 22](#_Toc11840)*

*[Hình: 9 . Xóa node có 2 child trên cây nhị phân tìm kiếm 22](#_Toc19078)*

*[Hình: 10 . Duyệt PreOrder trên cây nhị phân tìm kiếm 24](#_Toc27298)*

*[Hình: 11 .Duyệt InOrder trên cây nhị phân tìm kiếm 25](#_Toc711)*

*[Hình: 12 . Duyệt PostOrder trên cây nhị phân tìm kiếm 26](#_Toc6433)*

**PHÂN CÔNG NHIỆM VỤ**

|  |  |  |
| --- | --- | --- |
| **HỌ VÀ TÊN** | **MÃ SINH VIÊN** | **NHIỆM VỤ** |
| Võ Văn Kha | 5951071040 | Thuật toán Decision tree |
| Trần Thọ Hiền | 5951071026 | Thuật toán Decision tree |
| Nguyễn Công Hậu | 5951071025 | Thuật toán Linked List |
| Lê Nguyễn Thái Khang | 5951071043 | Thuật toán Linked List |
| Nguyễn Hoàng Nhật | 5951071069 | Thuật toán sắp xếp, minh họa |

**CHƯƠNG 1. MỞ ĐẦU**

**1.1 Tổng quan**

**1.1.1. Lý do chọn đề tài**

Đầu tiên phải khẳng định cấu trúc dữ liệu và giải thuật rất quan trọng trong lập trình, không riêng ngôn ngữ Java, PHP, Python.. tất cả ngôn ngữ lập trình khác điều cần cấu trúc dữ liệu và giải thuật.

Sau khi học lập trình một thời gian, bạn sẽ nhận ra việc chuyển đổi ngôn ngữ không quá khó.Ngôn ngữ có thể thay đổi, nhưng cái sẽ theo bạn mãi là cấu trúc dữ liệu và thuật toán.

Khi nắm vững 2 thứ này, bạn sẽ có cách tổ chức dữ liệu tốt, giải quyết vấn đề tốt.Khi đó bạn có khả năng lớn hơn là sẽ không cần phải viết code nữa, mà sẽ trở thành người thiết kế phần mềm hoặc team leader để quản lý dự án. Cấu trúc dữ liệu và giải thuật là phần cần phải luyện tập nhiều để có thể vững vàng, khó có thể một sớm một chiều. Nhưng kiên trì và đúng phương pháp thì sẽ thành công.

Vì vậy để đáp ứng nhu cầu học tập và nghiên cứu các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật, vẫn dụng các kiến thức đã học ở môn Trí tuệ nhân tạo xây dựng chương trình minh họa từng bước một số thuật toán cơ bản trong môn Cấu trúc dữ liệu và Giải thuật.

**1.1.2. Yêu cầu**

* Xây dựng chương trình mô phỏng các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật một cách dễ hiểu.
* Hiển thị từng bước tính toán và ý nghĩa của các câu lệnh.
* Mô hình dễ sử dụng thân thiện với người dùng, hoạt động nhanh và mượt mà.

**1.2. Phạm vi**

Chương trình hỗ trợ học tập và nghiên cứu hướng đến các đối tượng có nhu cầu học tập các thuật toán trong môn Cấu trúc dữ liệu và Giải thuật

Giới hạn trong 3 thuật toán

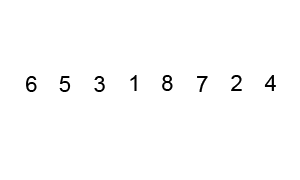
* Các thuật toán sắp xếp (ít nhất 3 thuật toán)
* Các thủ tục của cấu trúc danh sách liên kết: AddHead, AddTail, AddAfterQ, Duyệt, DeleteHead.
* Các thủ tục của cấu trúc Cây nhị phân tìm kiếm

**CHƯƠNG 2. TÌM HIỂU VỀ CÁC THUẬT TOÁN TRONG MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**2.1. THUẬT TOÁN SẮP XẾP**

**2.1.1. Sắp xếp nổi bọt**

Sắp xếp nổi bọt (tiếng Anh: bubble sort) là một thuật toán sắp xếp đơn giản, với thao tác cơ bản là so sánh hai phần tử kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ (swap). Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang). Sắp xếp nổi bọt còn có tên là sắp xếp bằng so sánh trực tiếp. Nó sử dụng phép so sánh các phần tử nên là một giải thuật sắp xếp kiểu so sánh.



**Giải thuật**

**Sắp xếp từ trên xuống**

Giả sử dãy cần sắp xếp có n phần tử. Khi tiến hành từ trên xuống, ta so sánh hai phần tử đầu, nếu phần tử đứng trước lớn hơn phần tử đứng sau thì đổi chỗ chúng cho nhau. Tiếp tục làm như vậy với cặp phần tử thứ hai và thứ ba và tiếp tục cho đến cuối tập hợp dữ liệu, nghĩa là so sánh (và đổi chỗ nếu cần) phần tử thứ n-1 với phần tử thứ n. Sau bước này phần tử cuối cùng chính là phần tử lớn nhất của dãy.

Sau đó, quay lại so sánh (và đổi chố nếu cần) hai phần tử đầu cho đến khi gặp phần tử thứ n-2....

Ghi chú: Nếu trong một lần duyệt, không phải đổi chỗ bất cứ cặp phần tử nào thì danh sách đã được sắp xếp xong.

**Sắp xếp từ dưới lên**

Sắp xếp từ dưới lên so sánh (và đổi chỗ nếu cần) bắt đầu từ việc so sánh cặp phần tử thứ n-1 và n. Tiếp theo là so sánh cặp phần tử thứ n-2 và n-1,... cho đến khi so sánh và đổi chỗ cặp phần tử thứ nhất và thứ hai. Sau bước này phần tử nhỏ nhất đã được nổi lên vị trí trên cùng (nó giống như hình ảnh của các "bọt" khí nhẹ hơn được nổi lên trên). Tiếp theo tiến hành với các phần tử từ thứ 2 đến thứ n.

**Mã giả**

**Sắp xếp từ trên xuống**

**procedure** bubble\_sort1(*list* L, *number* n) //n=listsize

**For** *number* i **from**  n **downto** 2

**for** *number* j **from** 1 **to** (i - 1)

**if** L[j] > L[j + 1] //nếu chúng không đúng thứ tự

swap(L[j], L[j + 1]) //đổi chỗ chúng cho nhau

**endif**

**endfor**

**endfor**

**endprocedure**

**Sắp xếp từ dưới lên**

**procedure** bubble\_sort2(*list* L, *number* n) //n=listsize

**Fornumber *i*** *from* ***1 to*** *n*-1

**for** *number* j **from** *n*-1 **downto** i

**if** L[j] > L[j + 1] //nếu chúng không đúng thứ tự

swap(L[j], L[j + 1]) //đổi chỗ chúng cho nhau

**endif**

**endfor**

**endfor**

**endprocedure**

**Giảm bớt vòng duyệt**

Nếu trong một lần duyệt nào đó với một i cố định khi vòng lặp j kết thúc mà không cần phải đổi chỗ cặp phần tử nào, nghĩa là mọi cặp phần tử kề nhau đã đứng đúng thứ tự thì dãy đã được sắp xếp và không cần tiến hành vòng lặp tiếp theo. Do đó có thể dùng một *cờ* để kiểm soát việc này. Ta có một đoạn mã giả của thuật toán nổi bọt như sau:

**procedure** bubble\_sort3(*list* L, *number* n)

i:= n

**while** i > 1 **do**

has\_swapped:= 0 //khởi tạo lại giá trị cờ

**for** *number* j **from** 1 **to** (i - 1)

**if** L[j] > L[j + 1] //nếu chúng không đúng thứ tự

swap(L[j], L[j + 1]) //đổi chỗ chúng cho nhau

has\_swapped:= 1 //có đổi chỗ ít nhất một lần, danh sách chưa sắp xếp xong

**endif**

**endfor**

**if** has\_swapped = 0 //nếu không có lần đổi chỗ nào, danh sách đã sắp xếp xong

**exit**

**else** //nếu có ít nhất một lần đổi chỗ, danh sách chưa sắp xếp xong

i = i - 1

**endif**

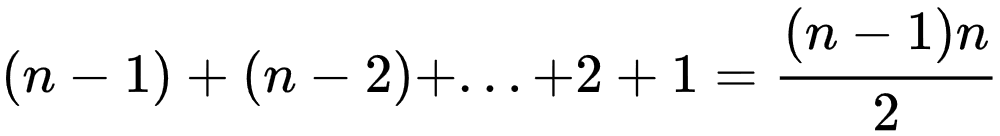
**enddo**

**endprocedure**

Ghi chú: Cũng có thể không cần dùng đến biến i, khi đó mỗi lần kiểm tra đều phải duyệt từ đầu đến cuối dãy.

**Thời gian tính**

Với mỗi i = 1,2,..,n-1 ta cần i phép so sánh. Do đó số nhiều nhất các lần so sánh và đổi chỗ trong giải thuật là



Do đó độ phức tạp của giải thuật cỡ O(N^2)

**2.1.2. Sắp xếp chèn**

**Sắp xếp chèn** (insertion sort) là một [thuật toán sắp xếp](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp" \o "Thuật toán sắp xếp) bắt chước cách sắp xếp quân bài của những người chơi bài. Muốn sắp một bộ bài theo trật tự người chơi bài rút lần lượt từ quân thứ 2, so với các quân đứng trước nó để chèn vào vị trí thích hợp.



**Thuật toán**

Cơ sở lập luận của sắp xếp chèn có thể mô tả như sau: Xét danh sách con gồm k phần tử đầu {\displaystyle a\_{1},...,a\_{k}},với k = 1, danh sách gồm một phần tử đã được sắp. Giả sử trong danh sách k-1 phần tử đầu {\displaystyle a\_{1},...,a\_{k-1}} đã được sắp. Để sắp xếp phần tử {\displaystyle a\_{k}=x}thứ k =x ta tìm vị trí thích hợp của nó trong dãy {\displaystyle a\_{1},...,a\_{k-1}}các phần tử trước k. Vị trí thích hợp đó là đứng trước phần tử lớn hơn nó và sau phần tử nhỏ hơn hoặc bằng nó.

|  |  |  |  |
| --- | --- | --- | --- |
| Các phần tử ≤x | Vị trí thích hợp | Các phần tử>x | Các phần tử chưa sắp |

**Giải thuật**

* Danh sách a bắt đầu từ chỉ số 1 tới length

**rocedure insert** (*array* a, *int* k, value) {

*int* i:= k-1;

**while** (i > 0 **and** a[i] > value) {

a[i+1]:= a[i];

i:= i - 1;

}

a[i+1]:= value;

}

**Procedure InsertSort** (*array* a, *int* length) {

*int* k:= 2;

**while** (k < length) {

insert(a, k, a[k]);

k:= k + 1;

}

}

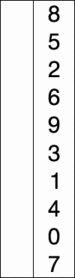
**Đánh giá**

* Thuật toán sử dụng trung bình n2/4 phép so sánh và n2/4 lần hoán vị, n2/2 phép so sánh và n2/2 lần hoán vị trong trường hợp xấu nhất, n-1 phép so sánh và 0 lần hoán vị trong trường hợp tốt nhất.
* Thuật toán thích hợp đối với mảng đã được sắp xếp một phần hoặc mảng có kích thước nhỏ

**2.1.3. Sắp xếp chọn**

**Sắp xếp chọn** là một [thuật toán sắp xếp](https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp" \o "Thuật toán sắp xếp) đơn giản, dựa trên việc so sánh tại chỗ.

Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu tiên của dãy hiện hành. Sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn n-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2. Lặp lại quá trình trên cho dãy hiện hành đến khi dãy hiện hành chỉ còn một phần tử. Dãy ban đầu có n phần tử, vậy tóm tắt ý tưởng thuật toán là thực hiện n-1 lượt việc đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí đúng ở đầu dãy.



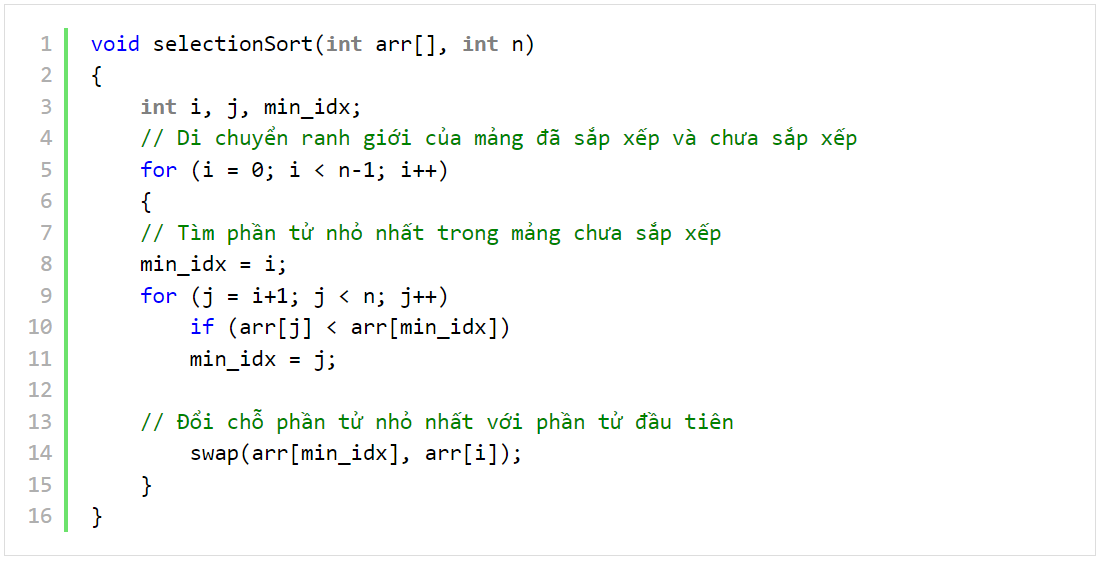
**Các bước thực hiện**

* Bước 1: i=1.
* Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[n].
* Bước 3: Hoán vị a[min] và a[i]
* Bước 4: Nếu i<=n-1 thì i=i+1; Lặp lại bước 2.
* Ngược lại: Dừng. n-1 phần tử đã nằm đúng vị trí.

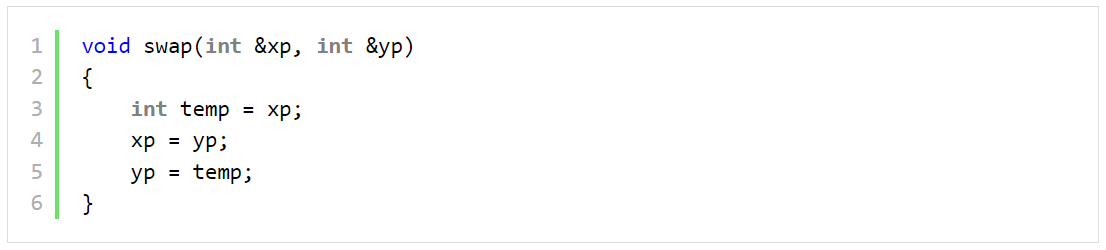
**Giải thích thuật toán**

1. Thiết lập giá trị nhỏ nhất (min) về vị trí số 0.
2. Tạo vòng lặp for để di chuyển ranh giới của sorted list và unsorted list.
3. Tìm phần tử nhỏ nhất trong list chưa được sắp xếp.
4. Sau khi tìm được phần tử nhỏ nhất thì đổi chỗ phần tử đó với phần tử đầu tiên. Ở bước này chúng ta cần viết một hàm Swap(), hàm này mình sẽ viết ở bên dưới.
5. Lặp lại cho tới khi list được sắp xếp xong.

**Thuật toán**



**Hàm swap():**

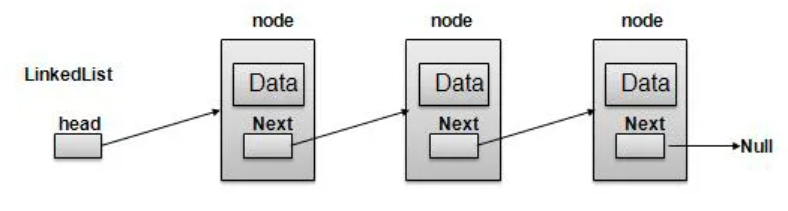


**Đánh giá**

* Thuật toán ít phải đổi chỗ các phần tử nhất trong số các thuật toán sắp xếp(n lần hoán vị) nhưng có độ phức tạp so sánh là O(n2) (n2/2 phép so sánh).
* Thuật toán tốn thời gian gần như bằng nhau đối với mảng đã được sắp xếp cũng như mảng chưa được sắp xếp.

**2.2. Các thủ tục của cấu trúc danh sách liên kết:**

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.



Hình: 1. Ảnh minh họa cấu trúc danh sách liên kết

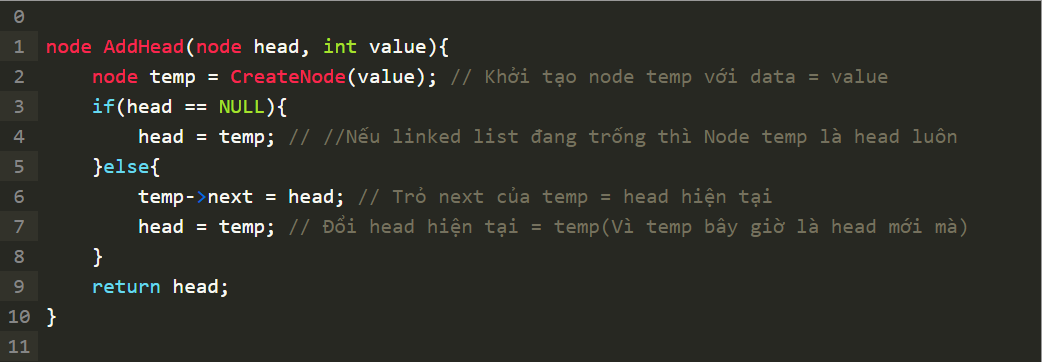
**các kiểu danh sách liên kết:**

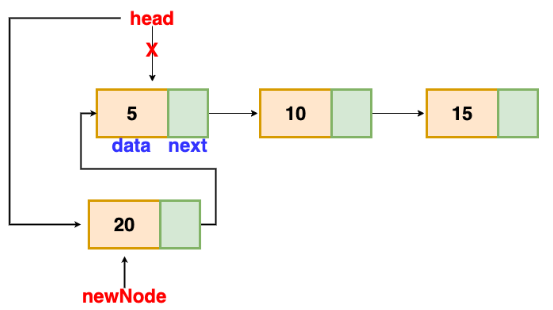
* Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau.
* Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía trước với phần tử phía sau
* Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.

### **2.2.1. Thêm vào đầu (AddHead)**

Việc thêm vào đầu chính là việc cập nhật lại thằng head. Ta gọi Node mới(temp), ta có:

* Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm head luôn
* Ngược lại, ta phải thay thế thằng head cũ bằng head mới. Việc này phải làm theo thứ tự như sau:
  + Cho next của temp trỏ tới head hiện hành
  + Đặt temp làm head mới

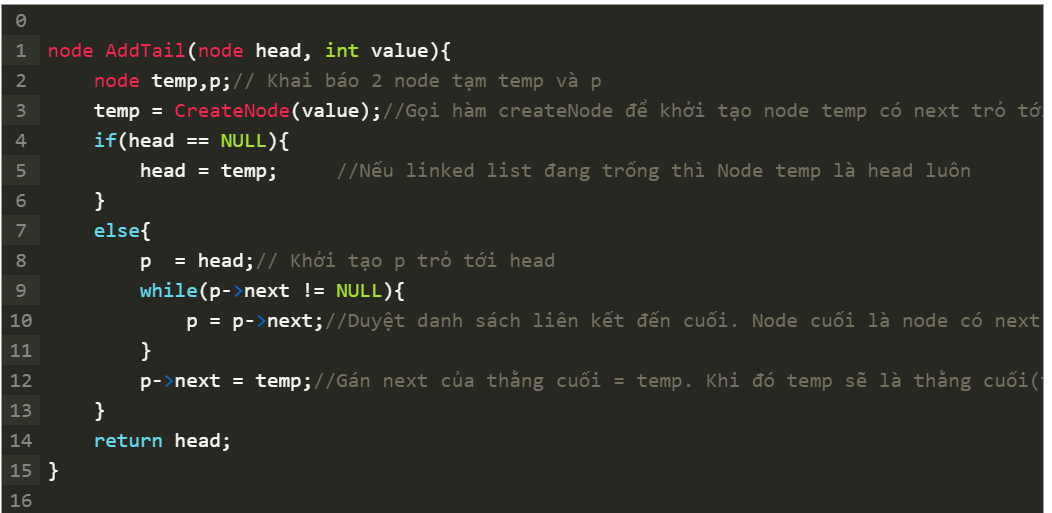


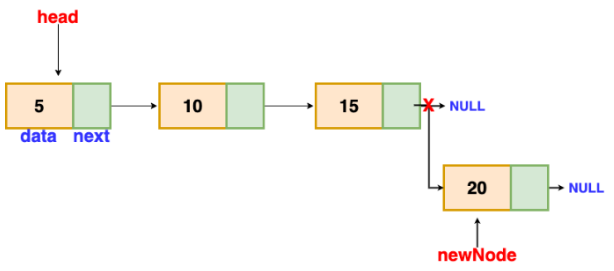


Hình: 2. Ảnh mô phỏng AddHead

**2.2.2. Thêm vào cuối (AddTail)**

* Trong trường hợp danh sách trống, chúng tôi sẽ trả về phần đầu đã cập nhật của danh sách liên kết vì trong trường hợp này, nút được chèn là nút đầu tiên cũng như nút cuối cùng của danh sách liên kết.
* Chúng ta sẽ xem qua danh sách cho đến khi chúng ta tìm thấy nút cuối cùng. Sau đó, chúng ta chèn nút mới vào cuối danh sách. Lưu ý rằng chúng ta phải xem xét các trường hợp đặc biệt như danh sách đang trống.

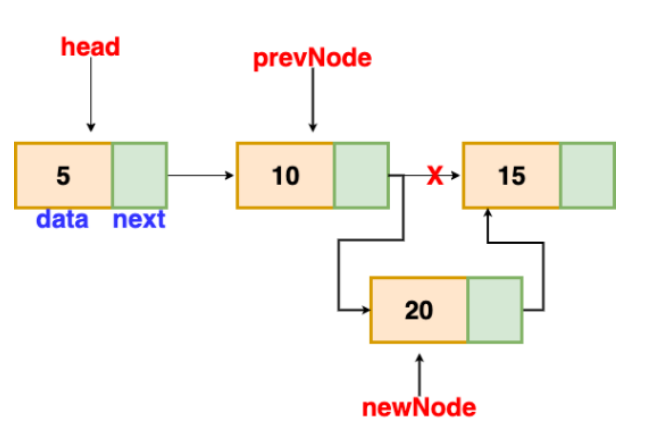
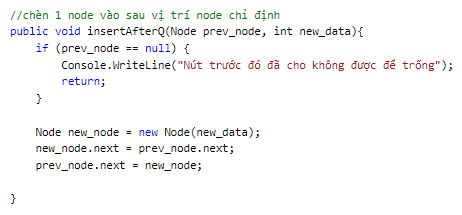




Hình: 3. Ảnh mô phỏng AddTail

**2.2.3.AddAfterQ**

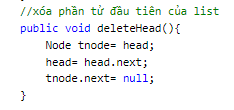
* Tạo một newNode cần thêm vào list
* Gán con trỏ next của newNode mới bằng con trỏ next của nó prevNode
* Trỏ con trỏ next của prevNode vào newNode



Hình: 4. Ảnh mô phỏng AddAfterQ

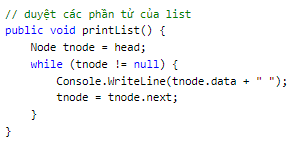
**2.2.4. Xóa đầu (deleteHead)**

* Khởi tạo 1 node tạm thời, gán node tạm thời cho node head
* Trỏ node Head đến node thứ hai
* Xóa node tạm thời
* Giải phóng bộ nhớ node đã xóa



**2.2.5. Duyệt (Traverse)**

* Khởi tạo 1 node tạm thời, gán node tạm thời cho node head
* Di chuyển node tạm thời sang node tiếp theo và hiển thị nội dung của nó
* Khi node tạm thời null ,thì chúng ta thoát ra khỏi vòng lặp



**2.3. Các thủ tục của cấu trúc Cây nhị phân tìm kiếm**

**Lý thuyết về cây tìm kiếm nhị phân:** Cây tìm kiếm nhị phân(Binary Search Tree – viết tắt: BST) – là một cây nhị phân và có thêm các ràng buộc sau đây:

1. Giá trị của tất cả các Node ở cây con bên trái phải <= giá trị của Node gốc.
2. Giá trị của tất cả các Node ở cây con bên phải phải > giá trị của Node gốc.
3. Tất cả các cây con(bao gồm bên trái và phải) cũng đều phải đảm bảo 2 tính chất trên.

Cây tìm kiếm nhị phân là một cấu trúc dữ liệu hiệu quả cho phép chúng ta xây dựng nên một danh sách mà dữ liệu trên đó được sắp xếp:

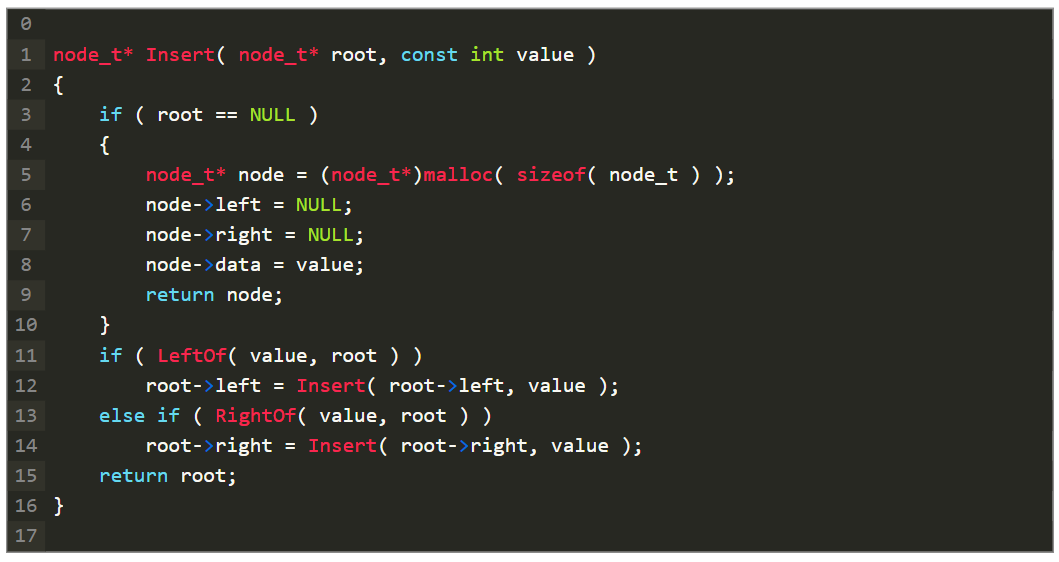
* Nó được gọi là cây nhị phân vì mỗi Node của cây chỉ có tối đa hai con
* Nó được gọi là cây tìm kiếm nhị phân vì nó có thể được sử dụng để tìm kiếm sự hiện diện của một phần tử trong thời gian O(log (n)).

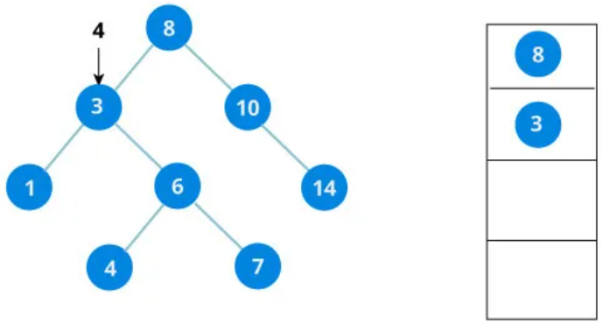
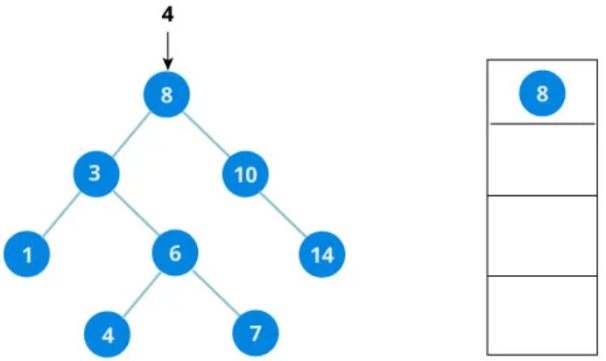
**2.3.1.Thêm phần tử vào cây tìm kiếm nhị phân**

Việc thêm 1 phần tử vào cây nhị phân tìm kiếm vẫn phải đảm bảo được các ràng buộc của một BST đã trình bày ở trên. Như vậy, bạn cần phải tìm kiếm vị trí thích hợp trong cây tìm kiếm nhị phân để lưu giữ nó.

Nếu bạn để ý, bạn sẽ nhận ra vị trí của các Node được thêm vào sẽ luôn Node lá(không có Child nào hết). Như vậy, tại vị trí đó trước khi các Node mới tới ở thì nó là NULL. Ta có quy trình như sau:

1. Nếu Node hiện tại = NULL, đó là vị trí cần thêm. Thêm vào BST và kết thúc
2. Nếu giá trị cần thêm < giá trị root hiện tại, gọi đệ quy Insert vào cây con bên trái
3. Nếu giá trị cần thêm > giá trị root hiện tại, gọi đệ quy Insert vào cây con bên phải.

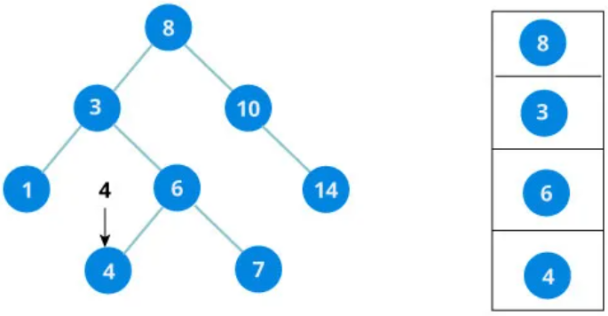
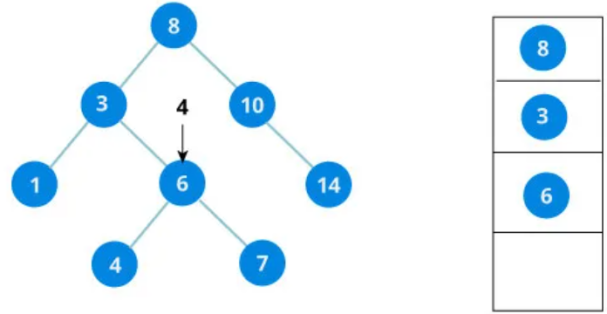




2)

1)

3)



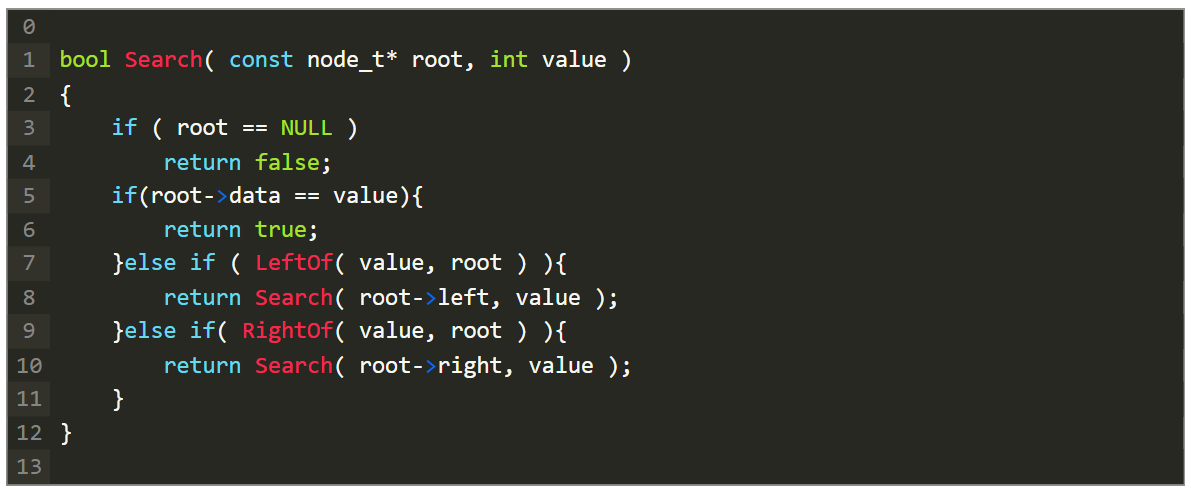
4)

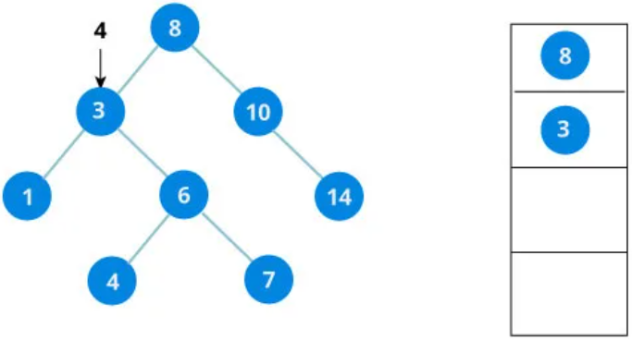
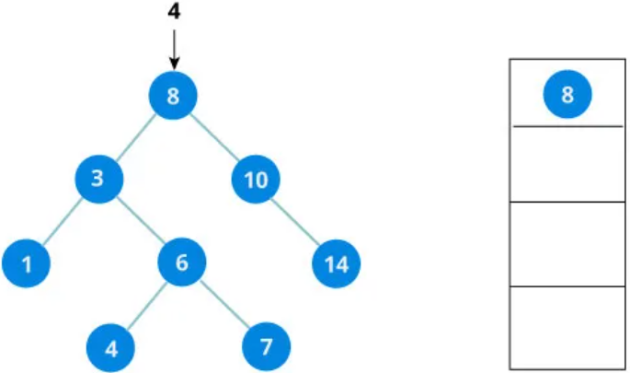
Hình: 5. Các bước thêm phần tử vào cây nhị phân

**2.3.2.Tìm kiếm trên BST**

Việc tìm kiếm cũng tương tự như việc thêm phần tử vào BST. Ta có quy trình như sau:

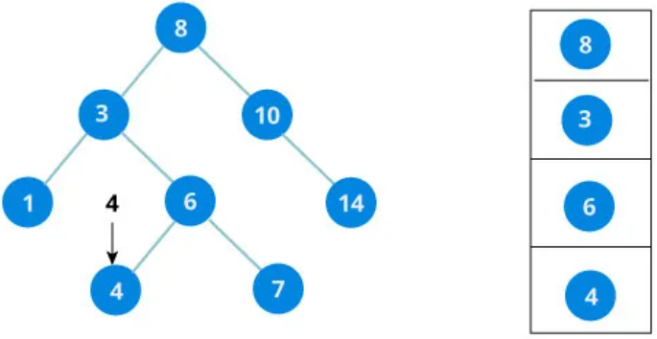
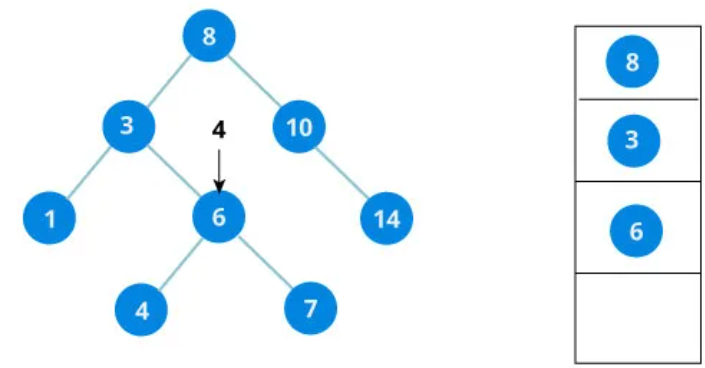
1. Nếu Node hiện tại có giá trị = giá trị cần tìm, trả về true và kết thúc.
2. Nếu Node hiện tại có giá trị > giá trị cần tìm, gọi đệ quy tìm ở cây con bên trái.
3. Nếu Node hiện tại có giá trị < giá trị cần tìm, gọi đệ quy tìm ở cây con bên phải
4. Nếu tìm đến hết cây(Node đó = NULL) mà không xảy ra (1), trả về false và kết thúc.





2)

1)



4)

3)

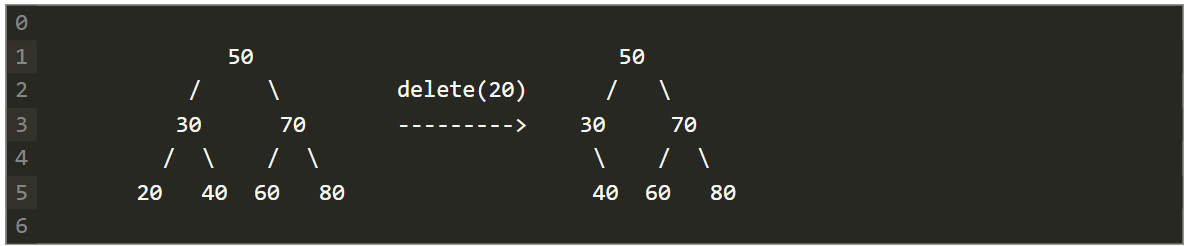
Hình: 6. Các bước tìm kiếm trên cây nhị phân tìm kiếm

**2.3.3.Xóa Node trong BST**

Việc xóa Node trong BST có lẽ là phức tạp nhất trong các chức năng của cây tìm kiếm nhị phân. Nếu Node bạn cần xóa là Node lá thì việc xóa rất đơn giản. Cái khó ở chỗ xóa một Node ở trung gian, khi đó, bạn phải tìm cách xóa và nối lại mà vẫn đảm bảo cây mới vẫn là BST.

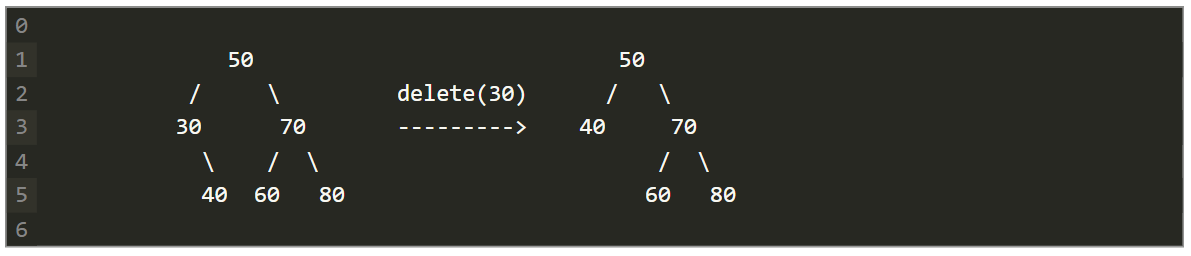
**2.3.3.1.Node cần xóa là Node lá(không có child nào cả)**

Giả sử ta cần xóa 20 trong hình dưới đây, đơn giản là chỉ cần giải phóng ô nhớ đó.



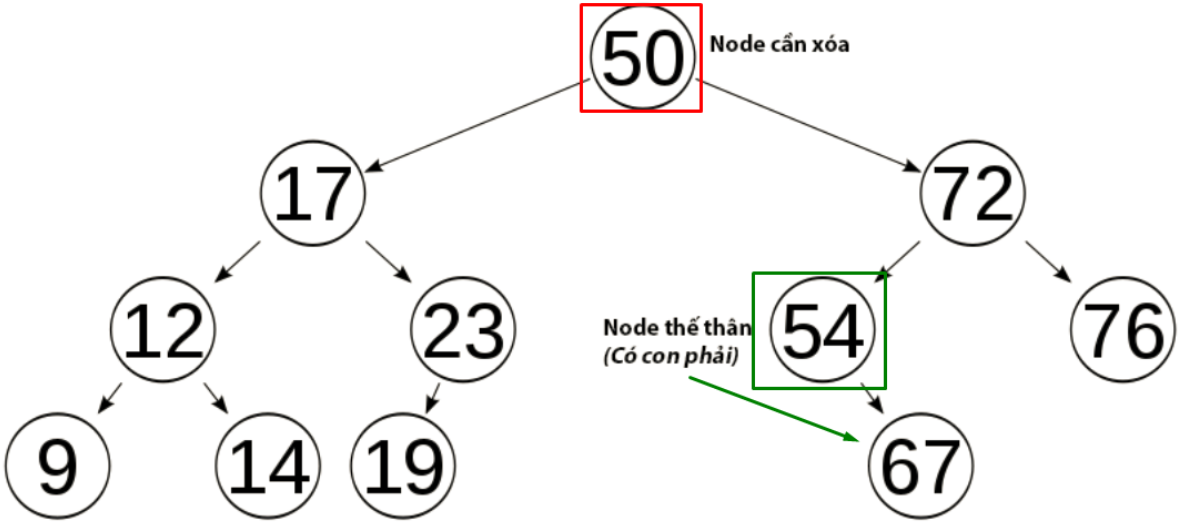
Hình: 7. Xóa node lá trên cây nhị phân tìm kiếm

**2.3.3.2. Node cần xóa có 1 child**



Hình: 8. Xóa node có 1 child trên cây nhị phân tìm kiếm

**2.3.3.3. Node cần xóa có đủ 2 child**



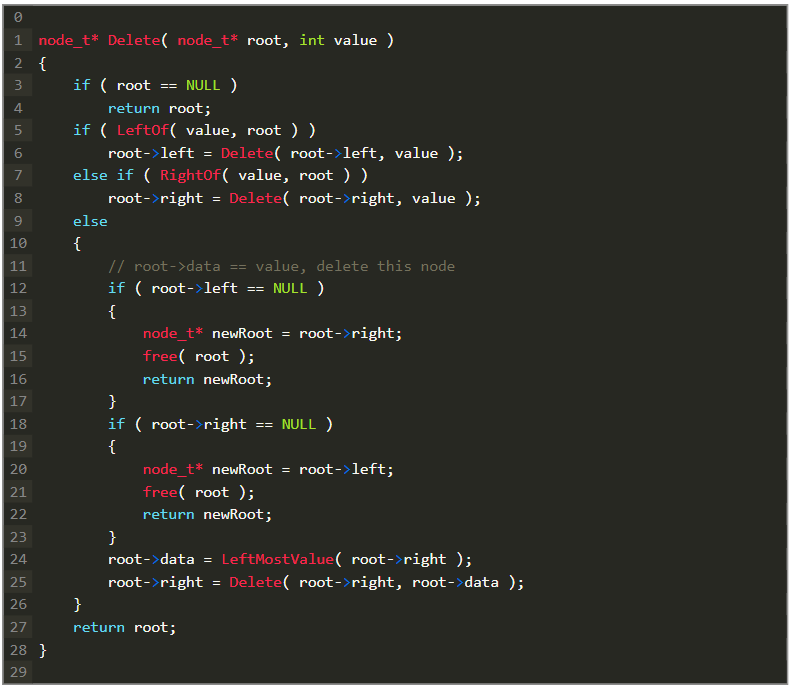
Hình: 9. Xóa node có 2 child trên cây nhị phân tìm kiếm

Đây là trường hợp nan giải nhất, nhưng chúng ta vẫn có cách làm như sau:

1. Tìm Node của con trái nhất(giả sử nó là leftmost) của cây con bên phải của Node cần xóa.
2. Cập nhật giá trị của Node cần xóa = giá trị của Node leftmost.
3. Gọi đệ quy hàm Delete xóa Node leftmost khỏi BST.

**Giải thích:**

* Khi muốn xóa Node có 2 con, ta cần tìm Node nào đó trong BST thỏa mãn tính chất lớn hơn lớn hơn tất cả các con bên trái và nhỏ hơn tất cả các con bên phái -> Node đó chính là LeftMostChild của con bên phải của Node cần xóa.
* Ta chỉ cần sửa giá trị của Node cần xóa, không cần xóa Node đó làm gì. Thay vào đó, xóa Node bị thế thân(LeftMostChild của con bên phải của Node cần xóa).



**2.3.4.Duyệt cây tìm kiếm nhị phân**

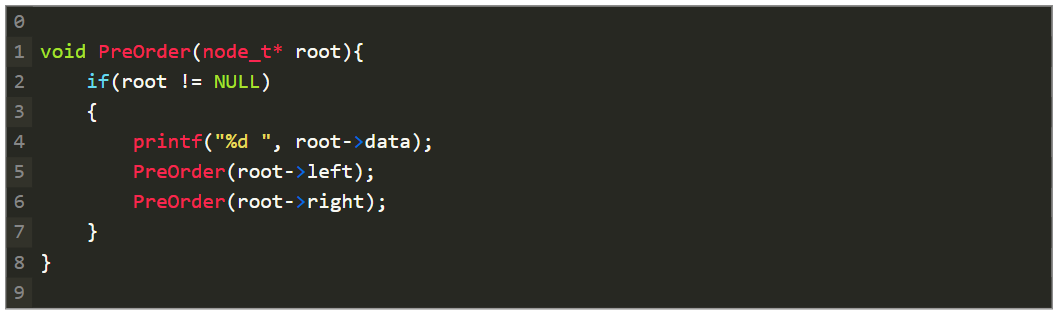
Thứ tự duyệt được đặt tên phụ thuộc vào vị trí của Node root trong quá trình duyệt:

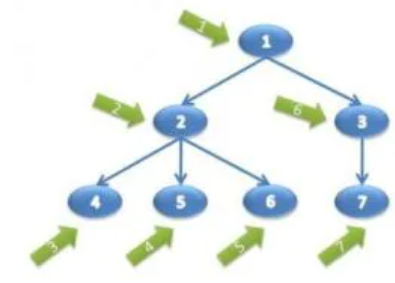
* PreOrder: Node -> Left -> Right
* InOrder: Left -> Node -> Right
* PostOrder: Left -> Right -> Node

**2.3.4.1. Duyệt PreOrder**

Quy trình duyệt PreOrder sẽ thực hiện theo thứ tự Node -> Left -> Right, cụ thể như sau:

1. Ghé thăm Node root
2. Gọi đệ quy duyệt qua cây con bên trái
3. Gọi đệ quy duyệt qua cây con bên phải



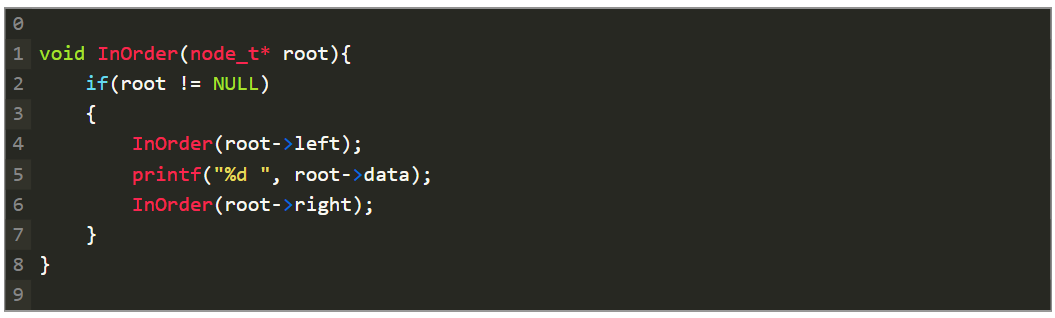


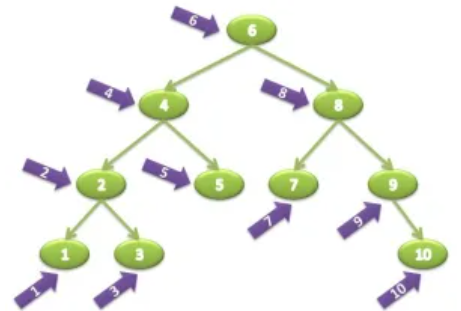
Hình: 10. Duyệt PreOrder trên cây nhị phân tìm kiếm

**2.3.4.2. Duyệt InOrder**

Quy trình duyệt PreOrder sẽ thực hiện theo thứ tự Left-> Node -> Right, cụ thể như sau:

1. Gọi đệ quy duyệt qua cây con bên trái
2. Ghé thăm Node root
3. Gọi đệ quy duyệt qua cây con bên phải



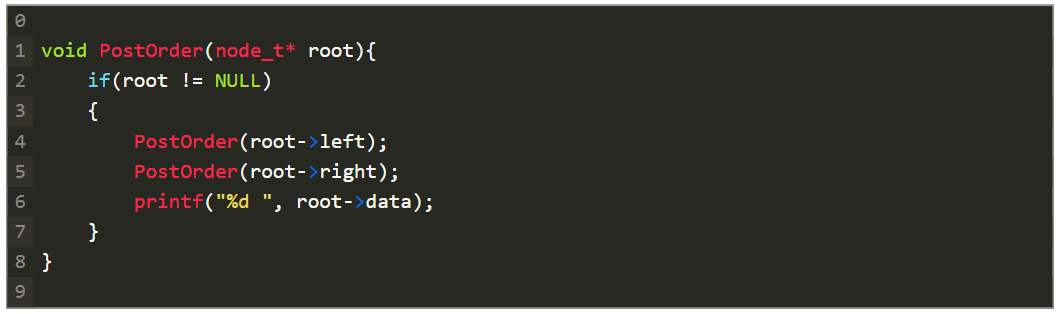


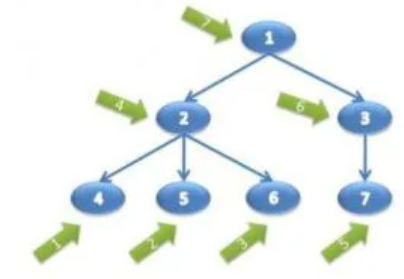
Hình: 11.Duyệt InOrder trên cây nhị phân tìm kiếm

**2.3.4.3. Duyệt PostOrder**

Quy trình duyệt PreOrder sẽ thực hiện theo thứ tự Left -> Right -> Node, cụ thể như sau:

1. Gọi đệ quy duyệt qua cây con bên trái
2. Gọi đệ quy duyệt qua cây con bên phải
3. Ghé thăm Node root





Hình: 12. Duyệt PostOrder trên cây nhị phân tìm kiếm