BÁO CÁO ĐỒ ÁN MÔN HỌC
(Đồ án tìm hiểu CTDL/Giải thuật)
Lớp: IT003.O21.CNTN

SINH VIÊN THỰC HIỆN

Mã sinh viên: 23520295 **Họ và tên:** Võ Hoàng Doanh

TÊN ĐỀ TÀI: **Cấu trúc dữ liệu Splay Tree**

CÁC NỘI DUNG CẦN BÁO CÁO: kết quả tìm hiểu được về CTDL/GT đã chọn

1. **Giới thiệu đồ án:**
   - Splay Tree là một cấu trúc dữ liệu được giới thiệu lần đầu bởi 2 nhà khoa học là Daniel Dominic Sleator và Robert Endre Tarjan vào năm 1985. Cây Splay là một cây nhị phân tìm kiếm được thiết kế cho việc truy xuất dữ liệu một cách hiệu quả.
2. **Quá trình thực hiện:**
   a. Tuần 1: Tìm hiểu, đọc tài liệu, nghiên cứu về Splay Tree, tổng hợp, lọc ra những nguồn tài liệu quan trọng.
   b. Tuần 2: Tổng hợp lại những thông tin quan trọng và cần thiết, viết báo cáo và xây dựng chương trình minh họa trên máy tính.

3. **Kết quả đạt được**
   a. **Các định nghĩa/khái niệm cơ bản**

➢ Định nghĩa[1][3]:

- Splay Tree là một cấu trúc dữ liệu thuộc lớp cây nhị phân tìm kiếm tự cân bằng (Self-Adjusting Binary Search Tree), có nghĩa là cây sẽ tự điều chỉnh lại chính nó dựa trên mỗi lần thực hiện 1 thao tác trên

cây. Với cây Splay, cây sẽ tự động tái cấu trúc lại chính nó sao cho các phần tử thường xuyên được truy cập hoặc chèn trở nên gần nút gốc hơn. Đây là tính chất bổ sung so với cây nhị phân tìm kiếm cơ bản giúp cho các phần tử truy cập gần đây có thể được truy cập lại 1 lần nữa hiệu quả hơn.

- Ý tưởng cơ bản về cách hoạt động cũng như phương pháp tự cân bằng của cây Splay được dựa trên một thao tác gọi là Splaying.
- Với 1 cây Splay có N-Node, tất cả các thao tác tiêu chuẩn chuẩn của 1 cây nhị phân tìm kiếm như ***tìm kiếm***, ***chèn*** và ***xóa*** đều có thời gian thực thi trung bình là O(logN) và thời gian thực thi cho trường hợp tệ nhất là O(N). Độ phức tạp về không gian là O(N).
- Khác với các cây nhị phân tìm kiếm tự cân bằng khác, cây Splay không đặt điều kiện cao về việc cây phải trở nên cân bằng mà chỉ quan tâm đến việc truy cập vào các phần tử đã sử dụng gần đây sao cho nhanh nhất có thể nên nó sẽ đơn giản hơn ở thao tác cân bằng cây so với 1 số loại cây cùng họ khác như AVL Tree hay RedBlack Tree. Điều này khiến cho Splay Tree có hiệu quả rất cao trong việc truy xuất những dữ liệu gần nhưng lại có thể kém hiệu quả hơn ở những trường hợp xấu do cây có thể trở nên mất cân đối khi chiều cao của cây trở thành tuyến tính.

➢ Các khái niệm cơ bản[3]:

❖ **Splaying:** Splaying là quá trình cấu trúc lại cây bằng cách đưa phần tử được truy cập hoặc chèn gần đây nhất thành nút gốc thông qua 1 hoặc 1 chuỗi các phép quay.

❖ **Các trường hợp về các phép quay trong Splay Tree:**

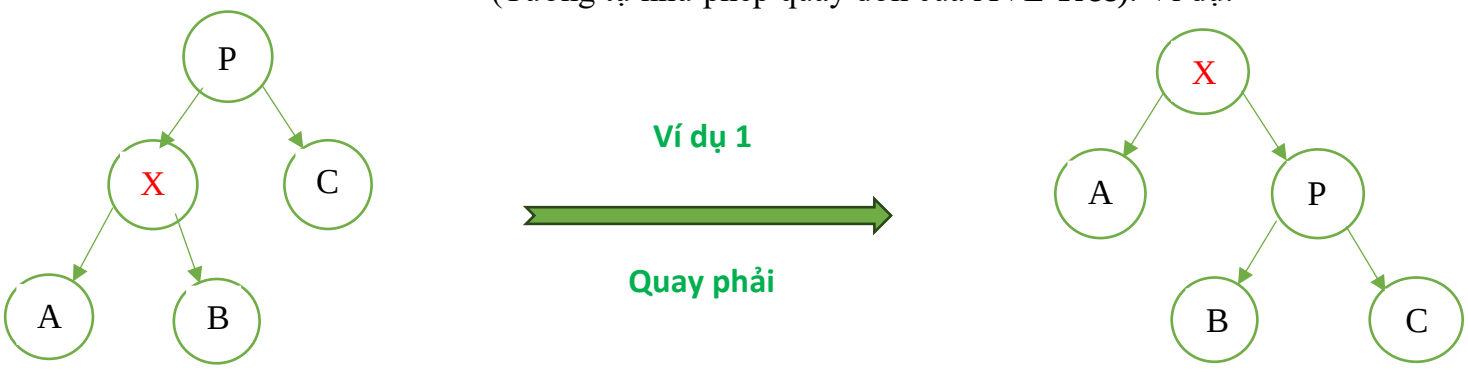
Ở đây ta quy ước:

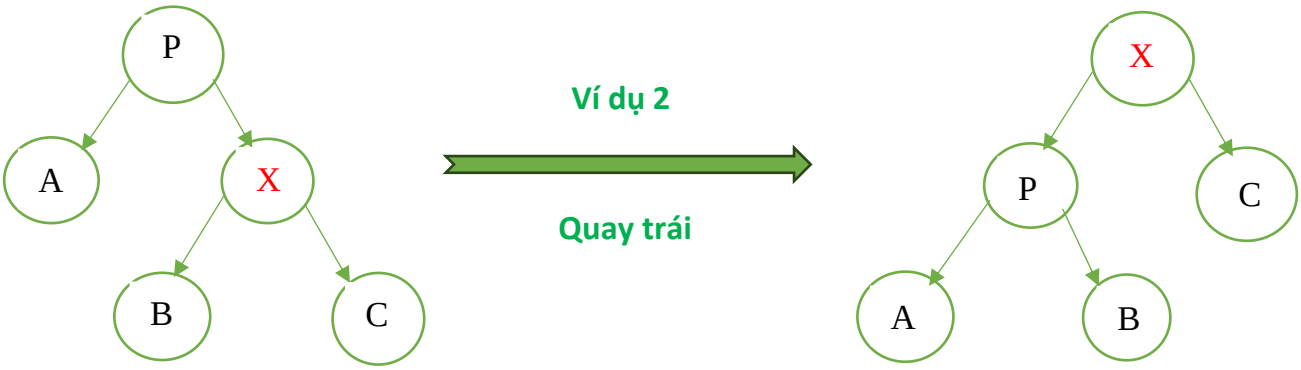
- X là nút đang được chỉ định
- P là nút Parent của X.

- G là nút Parent của P (Grandparent của X).

I. Zig Situation

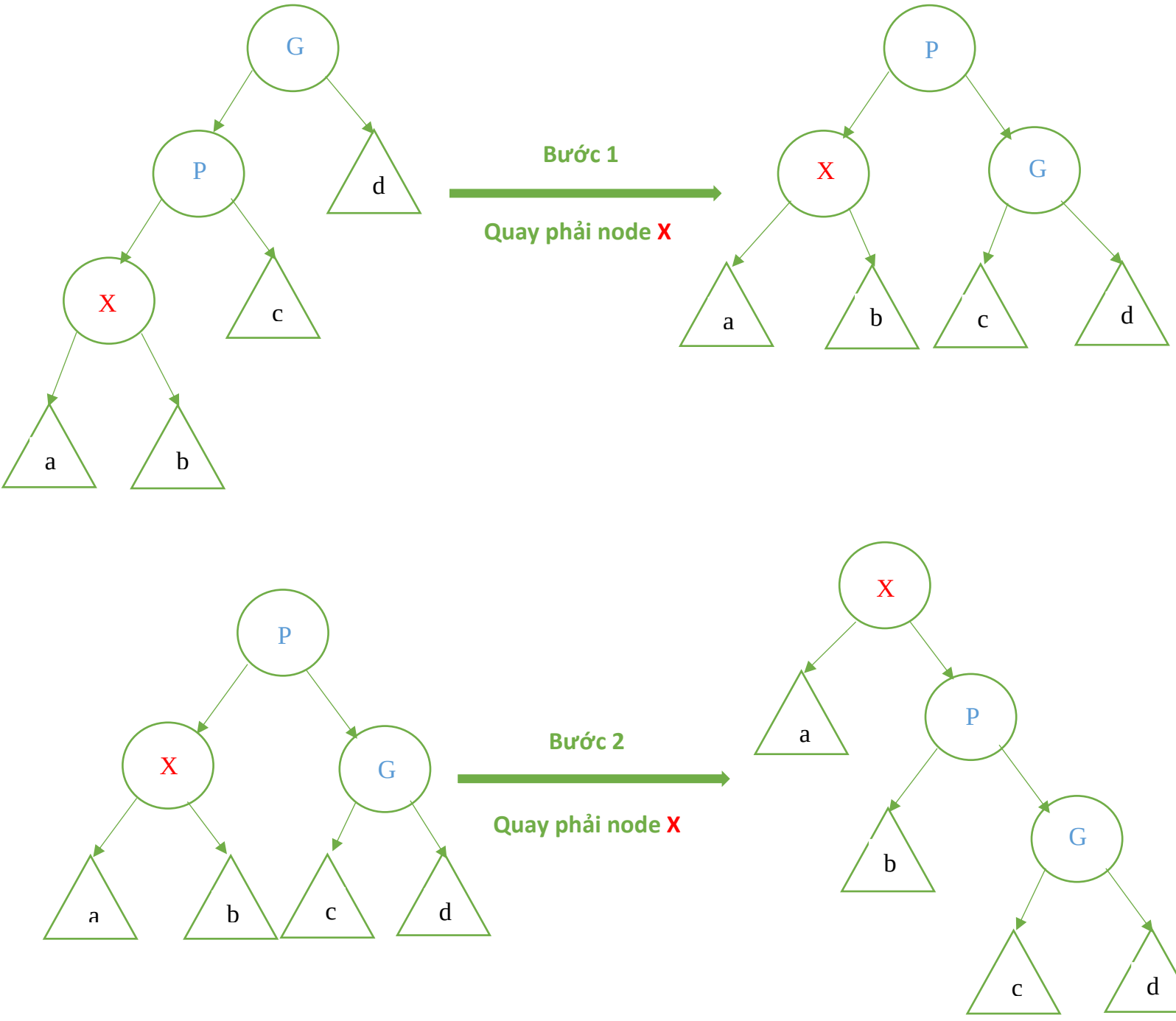
- Khi X là node con trực tiếp của ***Root***, thực hiện 1 phép quay đưa X lên làm nút gốc. Nếu X là con bên trái của ***Root***, thực hiện 1 phép quay phải. Ngược lại, thực hiện phép quay trái. (Tương tự như phép quay đơn của AVL Tree). Ví dụ:





II. Zig – Zig Situation

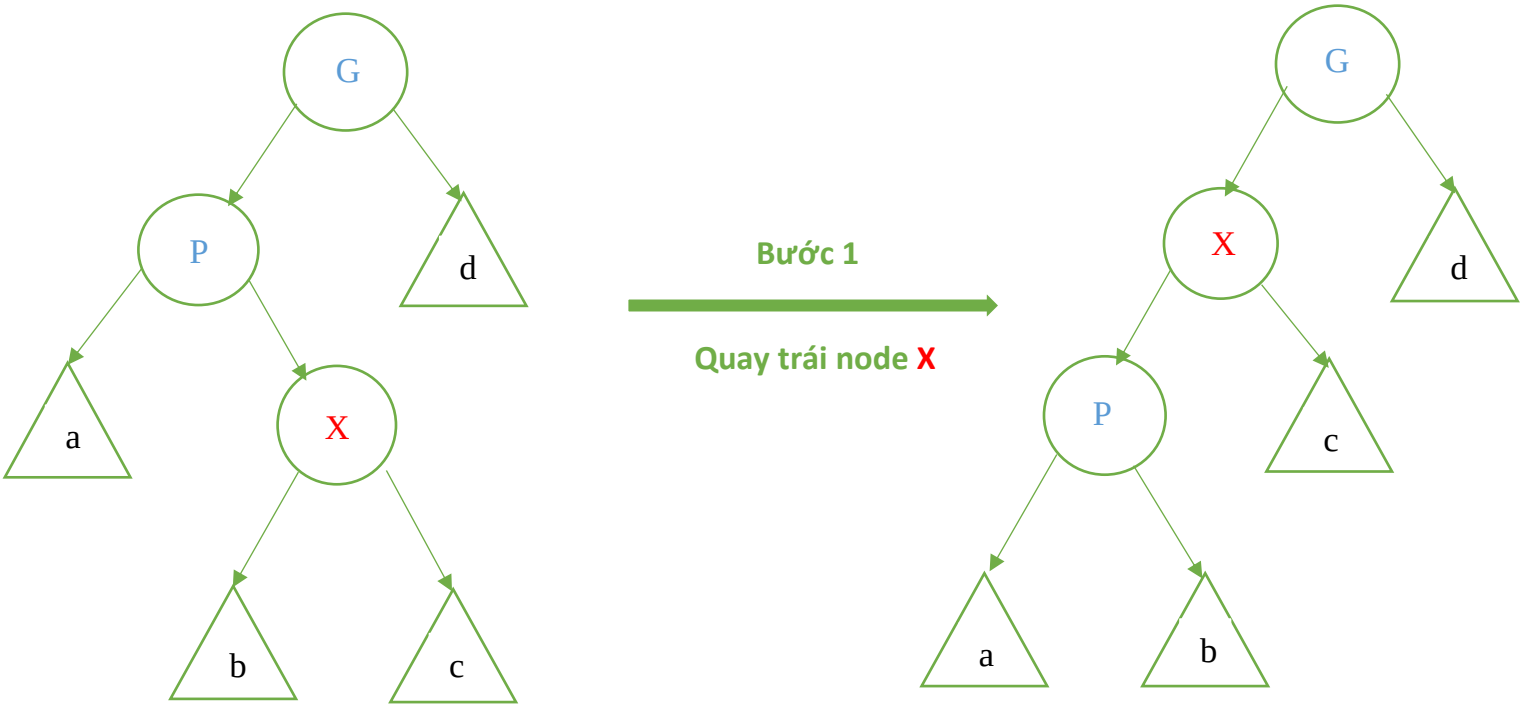
- Khi X và P cùng là node con bên trái hoặc cùng là node con bên phải. Khi đó thực hiện 2 phép quay đơn cùng hướng (quay phải 2 lần hoặc quay trái 2 lần). Ví dụ:

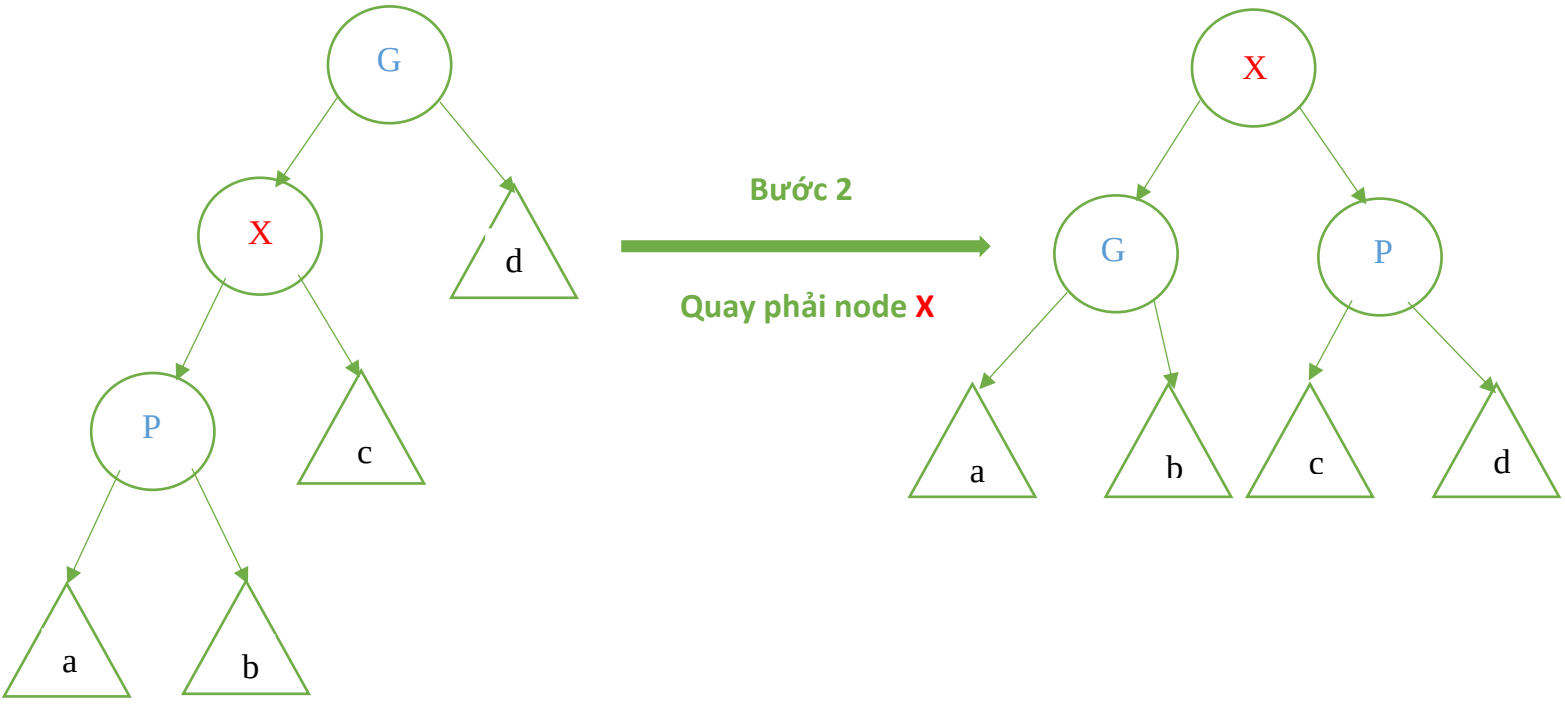

G
P
d
X
c
a
b
Bước 1
Quay phải node X
P
X
G
a
b
c
d

P
X
G
a
b
c
d
Bước 2
Quay phải node X
X
a
P
b
G
c
d

III. Zig – Zag Situation

- Khi X và P nằm khác hướng, thực hiện 2 phép quay đơn ngược hướng. Nếu X là node con bên phải của P và P là node con bên trái của G, thực hiện quay trái rồi sau đó quay phải. Nếu X là node con bên trái của P và P là node con bên phải của G, thực hiện quay phải rồi sau đó quay trái. Ví dụ:





b. Các đặc trưng nổi bật[1][2]

- Sở hữu đặc tính xuất sắc trong việc truy xuất dữ liệu: Các mục thường xuyên được truy cập trở nên dễ dàng cho việc tìm kiếm trong khi các mục ít được truy cập sẽ không gây cản trở.
- Độ phức tạp về thời gian: Mọi thao tác trên cây Splay đều có có thời gian trung bình là O(logN), giúp chúng nhanh hơn nhiều so với một số cấu trúc cây cân bằng khác trong một số trường hợp.
- Hiệu suất tương đương: Hiệu suất trung bình tương đương với các cây khác cùng họ.
- Bộ nhớ: Tiêu tốn ít bộ nhớ vì không cần yêu cầu thông tin phụ trên mỗi node để phục vụ cho việc cân bằng cây.
- Độ phức tạp trong cài đặt: Cây Splay đơn giản hơn 1 số loại cây cân bằng phổ biến khác như AVL Tree hay Red-Black Tree do không yêu cầu thông tin bổ sung trên mỗi nút và không đặt yêu cầu cao về việc cây phải trở nên cân đối.

- Thay đổi chỉ với thao tác đọc: Không giống như những loại cây khác, cây Splay sẽ tự tái cấu trúc lại chính nó ngay cả với thao tác chỉ cần đọc như tìm kiếm.

c. Các thao tác cơ sở (nếu là CTDL)[1]

1. Splaying:

I. Trường hợp cơ sở: Nếu cây rỗng hoặc đã là gốc, return về nút gốc.

II. Nếu key nằm ở cây con bên trái:

- Zig – Zig Situation(Left left): Key nằm ở cây con bên trái của cây con bên trái, thực hiện đệ quy splay cây con bên trái của cây con bên trái. Cuối cùng quay phải tại gốc
- Zig – Zag Situation(Left right): Key nằm ở cây con bên phải của cây con bên trái, thực hiện đệ quy splay cây con bên phải của cây con bên trái. Quay trái tại cây con bên trái và sau đó quay phải tại gốc.

III. Nếu key nằm ở cây con bên phải:

- Zig – Zig Situation(Right right): Key nằm trong cây con bên phải của cây con bên phải, thực hiện đệ quy splay cây con bên phải của cây con bên phải. Cuối cùng quay trái tại nút gốc.
- Zig – Zag Situtation(Right left): Key nằm trong cây con bên trái của cây con bên phải, thực hiện đệ quy splay cây con bên trái của cây con bên phải. Quay phải tại cây con bên phải và sau đó quay trái tại gốc.

⇒ ***Kết quả thu được sau khi kết thúc hàm Splay***: Cây tái cấu trúc lại chính nó với khóa "key" làm nút gốc, nếu "key" không tồn tại, nút gần nhất với khóa "key" được đưa lên làm nút gốc.

2. Chèn

I. Trường hợp cơ sở: Gốc là rỗng, cấp phát một nút mới và return nó về như 1 nút gốc.

II. Splay khóa "key": gọi hàm Splay với khóa "key". Nếu "key" có trong cây, hàm Splay sẽ đưa "key" lên làm gốc. Ngược lại, nếu cây không có khóa "key", hàm splay sẽ đưa nút gần nhất với "key" lên làm gốc.

III. Sau khi gọi hàm Splay, nếu gốc mới có khóa là "key", có nghĩa là "key" đã tồn tại trong cây, không làm gì thêm.

IV. Nếu "key" chưa tồn tại trong cây, chúng ta cần tạo một nút mới với khóa "key", so sánh khóa "key" với gốc hiện tại:

a) Nếu "key" nhỏ hơn gốc:

- Làm cho gốc hiện tại trở thành con phải của nút mới.
- Sao chép con trái của gốc hiện tại(nếu có) làm con trái của nút mới.
- Đặt con trái của gốc hiện tại thành rỗng.

b) Nếu "key" lớn hơn gốc:

- Làm cho gốc hiện tại trở thành con trái của nút mới.
- Sao chép con phải của gốc hiện tại(nếu có) làm con phải của nút mới.
- Đặt con phải của gốc hiện tại thành rỗng.

3. Xóa

I. Trường hợp cơ sở: Gốc là rỗng, return nút gốc.

II. Nếu không, Splay khóa "key" đã cho. Nếu "key" tồn tại, nó trở thành gốc mới, nếu không có khóa "key" trong cây, nút lá được truy cập cuối cùng được đưa lên làm gốc.

III. Nếu gốc mới không bằng khóa "key", return về gốc bởi vì "key" không tồn tại trong cây.

IV. Nếu "key" tồn tại(và đang là gốc):

- Chia cây thành hai cây Tree1 = cây con bên trái của gốc và Tree2 = cây con bên phải của gốc và xóa nút gốc.
- Đặt gốc của Tree1 và Tree2 là Root1 và Root2 tương ứng.
- Nếu Root1 là NULL: Trả về Root2.
- Ngược lại, Splay nút lớn nhất (nút có giá trị lớn nhất) của Tree1.
- Sau quy trình Splay, làm cho Root2 trở thành con phải của Root1 và return về Root1.

4. Tìm kiếm

- Thao tác tìm kiếm trong cây Splay thực hiện phép tìm kiếm BST tiêu chuẩn. Ngoài việc tìm kiếm, nó còn thực hiện splay. Nếu tìm kiếm thành công, thì nút được tìm thấy được splay và trở thành gốc mới. Ngược lại, nút cuối cùng được truy cập trước khi chạm đến NULL được splay và trở thành gốc mới.

d. Các thư viện hỗ trợ (C++/Python)[5]

- **C++:** Hiện tại trong thư viện chuẩn C++(STL) chưa có container hay CTDL nào được cài đặt sử dụng Splay Tree, chỉ có trong một số thư viện mã nguồn mở như thư viện Boost.

- **Python:** pybst (thư viện cung cấp các CTDL cây nhị phân tìm kiếm, bao gồm cả Splay Tree).

e. Vận dụng thực tế[1]

- **Bộ nhớ đệm(Cache):** Ứng dụng phổ biến nhất của SlayTree. Những phần tử được truy cập thường xuyên sẽ được di chuyển lên gần gốc, giúp cải thiện thời gian truy cập vào các phần tử này trong tương lai.
- **Database Indexing**: Cây Splay có thể được dùng để tạo chỉ mục cho cơ sở dữ liệu để tìm kiếm và truy xuất dữ liệu nhanh hơn.
- **Nén dữ liệu**: Cây Splay có thể được dùn để nén dữ liệu bằng cách xác định và mã hóa các chuỗi lặp lại.
- **File Systems**: Dùng cây Splay để lưu trữ metadata của file system.
- **Xử lý văn bản**: Cây Splay có thể được sử dụng trong các ứng dụng xử lý văn bản, như các chương trình kiểm tra chính tả, nơi mà từ ngữ được lưu trữ trong một cây Splay để tìm kiếm và truy xuất nhanh chóng.
- **Áp dụng trong thuật toán đồ thị**: Cây Splay có thể được sử dụng để triển khai một số thuật toán đồ thị.
- **Online Game:** Cây Splay có thể dùng để lưu trữ và quản lý điểm, bảng xếp hạng và thống kê của người chơi.

4. Tài liệu tham khảo

1. Geeksforgeeks :Introduction to Splay tree data structure - GeeksforGeeks
2. Wikipedia :Splay tree - Wikipedia
3. Youtube: Splay Tree Introduction – Global Software Support: https://youtu.be/IBY4NtxmGg8?si=RppBpbwkfBEt9LDn
4. StackOverflow: https://stackoverflow.com/posts/36810117/revisions

5. Copilot AI

5. Phụ lục: Chương trình minh họa:

- Chương trình minh họa: [Splay-Tree-Data-Structure/SplayTree.cpp at main · aslpk21/Splay-Tree-Data-Structure (github.com)]
- Cách hoạt động của chương trình: Nhập Input là 1 trong những lựa chọn có sẵn được cài đặt.

  Chương trình có 6 lựa chọn đầu vào:

  1. Chèn một nút
  2. Xóa một nút
  3. Tìm kiếm một phần tử
  4. In cây hiện tại ra màn hình
  5. Khởi tạo một cây mẫu
  6. Thoát khỏi chương trình