BÀI 6: CÂY ĐỎ ĐEN

1. GIỚI THIỆU

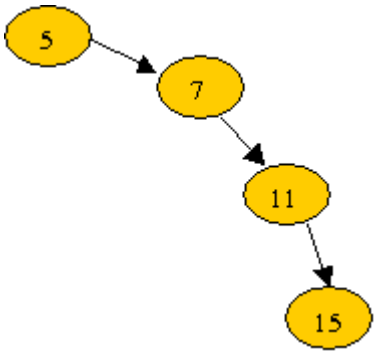
Cây tìm kiếm nhị phân là một cấu trúc lưu trữ dữ liệu tốt với tốc độ tìm kiếm nhanh.

Tuy nhiên trong một số trường hợp cây tìm kiếm nhị phân có một số hạn chế. Nó hoạt động tốt nếu dữ liệu được chèn vào cây theo thứ tự ngẫu nhiên. Tuy nhiên, nếu dữ liệu được chèn vào theo thứ tự đã được sắp xếp sẽ không hiệu quả. Khi các trị số cần chèn đã được sắp xếp thì cây nhị phân trở nên không cân bằng. Khi cây không cân bằng, nó mất đi khả năng tìm kiếm nhanh (hoặc chèn hoặc xóa) một phần tử đã cho.

Chúng ta khảo sát một cách giải quyết vấn đề của cây không cân bằng: đó là cây đỏ đen, là cây tìm kiếm nhị phân có thêm một vài đặc điểm .

Có nhiều cách tiếp cận khác để bảo đảm cho cây cân bằng: chẳng hạn cây 2-3-4. Tuy vậy, trong phần lớn trường hợp, cây đỏ đen là cây cân bằng hiệu quả nhất, ít ra thì khi dữ liệu được lưu trữ trong bộ nhớ chứ không phải trong những tập tin.

Trước khi khảo sát cây đỏ đen, hãy xem lại cây không cân bằng được tạo ra như thế nào.



Hình 1. Các node được chèn theo thứ tự tăng dần

Những node này tự sắp xếp thành một đường không phân nhánh. Bởi vì mỗi node lớn hơn node đã được chèn vào trước đó, mỗi node là con phải của nút trước đó. Khi ấy, cây bị mất cân bằng hoàn toàn.

Độ phức tạp:

Khi cây một nhánh, sẽ trở thành một danh sách liên kết, dữ liệu sẽ là một chiều thay vì hai chiều. Trong trường hợp này, thời gian truy xuất giảm về O(N), thay vì $O(\log_2 N)$ đối với cây cân bằng.

Để bảo đảm thời gian truy xuất nhanh của cây, chúng ta cần phải bảo đảm cây luôn luôn cân bằng (ít ra cũng là cây gần cân bằng). Điều này có nghĩa là mỗi node trên cây phải có xấp xỉ số node con bên phải bằng số node con bên trái.

2. ĐỊNH NGHĨA CÂY ĐỎ ĐEN

Cây đỏ đen là một cây nhị phân tìm kiếm (BST) tuân thủ các quy tắc sau: (hình 2)

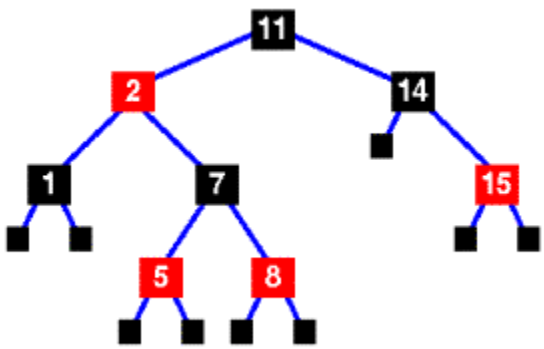
(1) Mọi node phải là đỏ hoặc đen.

(2) Node gốc và các node lá (NIL) phải luôn luôn đen.

(3) Nếu một node là đỏ, những node con của nó phải đen.

(4) Mọi đường dẫn từ gốc đến một lá phải có cùng số lượng node đen.

Khi chèn (hay xóa) một node mới, cần phải tuân thủ các quy tắc trên -gọi là quy tắc đỏ đen. Nếu được tuân thủ, cây sẽ được cân bằng.



Hình 2. Một ví dụ về cây đỏ đen

Số lượng node đen trên một đường dẫn từ gốc đến lá được gọi là *chiều cao đen* (black height). Ta có thể phát biểu quy tắc (4) theo một cách khác là mọi đường dẫn từ gốc đến lá phải có cùng chiều cao đen .

Khai báo cấu trúc:

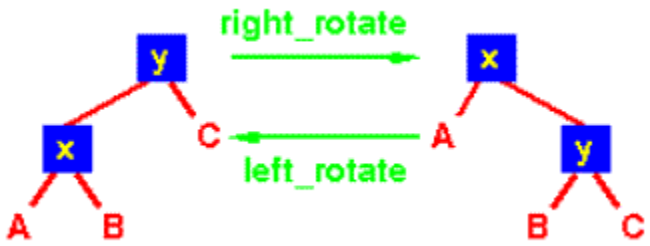
```
typedef int Data; /* Kiểu dữ liệu khoá */
typedef enum { BLACK, RED } nodeColor;
typedef struct NodeTag {
    nodeColor color; /* Màu node (BLACK, RED) */
    Data info; /* Khoá sử dụng tìm kiếm */
    struct NodeTag *left; /* Con trái */
    struct NodeTag *right; /* Con phải */
    struct NodeTag *parent; /* Cha */
} NodeType;
typedef NodeType *iterator;
```

Bổ đề:

Một cây đỏ đen n-node có chiều cao $h <= 2 \log_2(n+1)$

3. PHÉP QUAY

Thực ra quay không có nghĩa là các node bị quay mà để chỉ sự thay đổi quan hệ giữa chúng. Một node được chọn làm "đỉnh" của phép quay. Nếu chúng ta đang thực hiện một phép quay qua phải, node "đỉnh" này sẽ di chuyển xuống dưới và về bên phải, vào vị trí của node con bên phải của nó. Node con bên trái sẽ đi lên để chiếm lấy vị trí của nó.



Hình 3. Quay trái và quay phải

Phải đảm bảo trong phép quay phải, node ở đỉnh phải có node con trái. Nếu không chẳng có gì để quay vào điểm đỉnh. Tương tự, nếu làm phép quay trái, node ở đỉnh phải có node con phải.

4. THÊM NODE MỚI

Chúng ta sẽ xem xét việc mô tả qui trình chèn. Gọi X, P, và G để chỉ định nhãn những node liên quan. X là node vi phạm quy tắc (X có thể là một node mới được chèn, hoặc node con khi node cha và node con xung đột đỏ-đỏ, nghĩa là có cùng màu đỏ).

– X là một node cho trước.

– P là node cha của X.

– G là node ông bà của X (node cha của P).

Trong quá trình thêm vào node mới có thể vi phạm các quy tắc của cây đỏ đen, chúng ta sẽ thực hiện các thao tác sau đây:

– Các phép lật màu trên đường đi xuống.

– Các phép quay khi node đã được chèn.

– Các phép quay trên đường đi xuống.

4.1 Các phép lật màu trên đường đi xuống

Phép thêm vào trong cây đỏ đen bắt đầu như trên cây tìm kiếm nhị phân thông thường: đi theo một đường dẫn từ node gốc đến vị trí cần chèn, đi qua phải hay trái tùy vào giá trị của khóa node và khóa tìm kiếm.

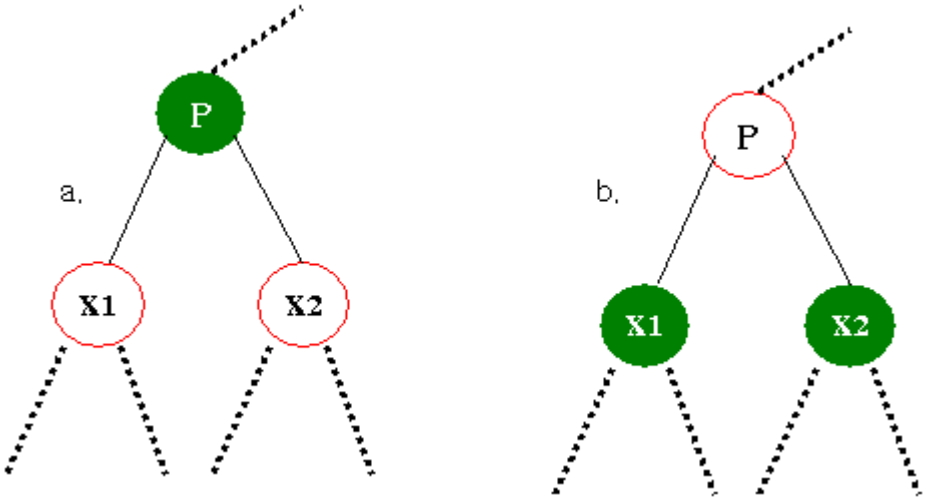
Tuy nhiên, trong cây đỏ đen, đến được điểm chèn là phức tạp bởi các phép lật màu và quay.

Để bảo đảm không vi phạm các quy tắc màu, cần phải tiến hành các phép lật màu khi cần theo quy tắc như sau:

Nếu phép thêm vào làm xuất hiện tình trạng một node đen có hai node con đỏ, chúng ta đổi các node con thành đen và node cha thành đỏ (trừ khi node cha là node gốc, nó vẫn vẫn giữ màu là đen).

Một phép lật màu ảnh hưởng đến các quy tắc đỏ-đen ra sao? chúng ta gọi node ở đỉnh tam giác, node có màu đen trước phép lật là P (P thay cho node cha). Chúng ta gọi hai node con trái và phải của P là X1 và X2. Xem hình 4a.

Hình 4. Lật màu



Hình 4a. trước khi lật màu, Hình 4b sau khi lật màu.

Chúng ta nhận thấy sau khi lật màu chiếu cao đen của cây không đổi. Như vậy phép lật màu không vi phạm quy tắc (4).

Mặc dù quy tắc (4) không bị vi phạm qua phép lật, nhưng quy tắc 3 (một node con và node cha không thể đồng màu đỏ) lại có khả năng bị vi phạm. Nếu node cha của P là đen, không có vấn đề vi phạm khi P được đổi từ đen sang đỏ, nhưng nếu node cha của P là đỏ, thì sau khi đổi màu, ta sẽ có hai node đỏ trên một hàng.

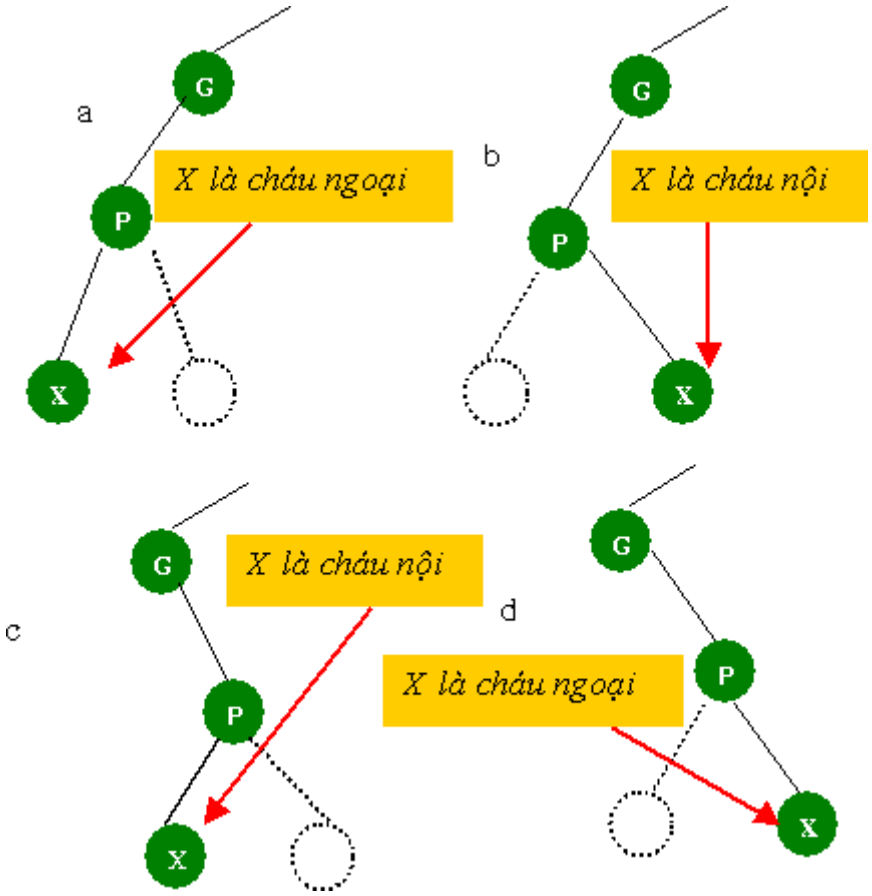
Điều này cần phải được chuẩn bị trước khi đi xuống theo cây để chèn node mới. Chúng ta có thể giải quyết trường hợp này bằng một phép quay.

Đối với node gốc thì phép lật màu node gốc và hai node con của nó vẫn làm cho node gốc cũng như hai node con có màu đen. Điều này tránh sự vi phạm quy tắc 2 và quy tắc 3 (xung đột đỏ-đỏ). Trong trường hợp này, chiều cao đen trên mỗi đường đi từ node gốc tăng lên 1, do đó quy tắc 4 cũng không bị vi phạm.

4.2. Các phép quay khi chèn node

Thao tác chèn node mới có thể làm cho quy tắc đỏ-đen bị vi phạm. Do vậy sau khi chèn, cần phải kiểm tra xem có phạm quy tắc không và thực hiện những thao tác hợp lý.

Như đã xét ở trên, node mới được chèn mà ta gọi là node X, luôn luôn đỏ. Node X có thể nằm ở những vị trí khác nhau đối với P và G, như trong hình 5.

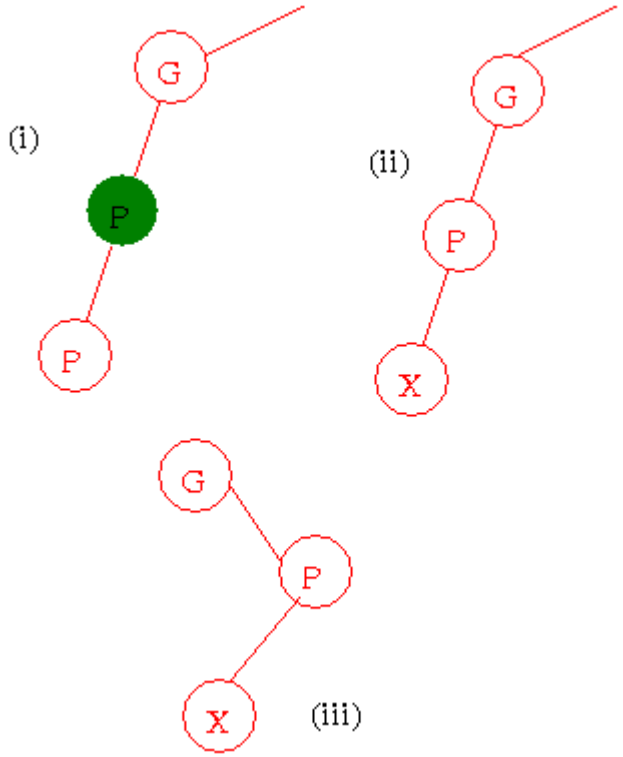


Hình 5. Các biến dạng của node được chèn

X là một node cháu ngoại nếu nó nằm cùng bên node cha P và P cùng bên node cha G. Điều này có nghĩa là, X là node cháu ngoại nếu hoặc nó là node con trái của P và P là node con trái của G, hoặc nó là node con phải của P và node P là node con phải của G. Ngược lại, X là một node cháu nội.

Nếu X là node cháu ngoại, nó có thể hoặc bên trái hoặc bên phải của P, tùy vào việc node P ở bên trái hay bên phải node G. Có hai khả năng tương tự nếu X là một node cháu nội. Bốn trường hợp này được trình bày trong hình 5.

Thao tác phục hồi quy tắc đỏ-đen được xác định bởi các màu và cấu hình của node X và những bà con của nó. Có 3 khả năng xảy ra được xem xét như sau:(hình 6)



Hình 6. Ba khả năng sau khi chèn nút

i) Khả năng 1: P đen

ii) Khả năng 2: P đỏ và X là cháu ngoại của G

iii) Khả năng 3: P đỏ và X là cháu nội của G

Chúng ta sẽ xét các khả năng trên một cách cụ thể như sau:

i) Khả năng 1: P đen

P đen là trường hợp đơn giản. Node thêm vào luôn đỏ. Nếu node cha đen, không có xung khắc đỏ-đỏ (quy tắc 3), và không có việc cộng thêm vào số node đen (quy tắc 4). Do vậy, không bị vi phạm quy tắc về màu. Thao tác chèn đã hoàn tất.

ii) Khả năng 2: P đỏ và X là cháu ngoại của G

Nếu node P đỏ và X là node cháu ngoại, ta cần một phép quay đơn giản và một vài thay đổi về màu. Bắt đầu với giá trị 50 tại node gốc, và chèn

các node 25, 75 và 12. Ta cần phải làm một phép lật màu trước khi chèn node 12.

Bây giờ, chèn node mới X là 6. (hình 7a.)xuất hiện lỗi: cha và con đều đỏ, vì vậy cần phải có các thao tác như sau: (hình 7)

Trong trường hợp này, ta có thể áp dụng ba bước để phục hồi tính đỏ-đen và làm cho cân bằng cây. Sau đây là các bước ấy:

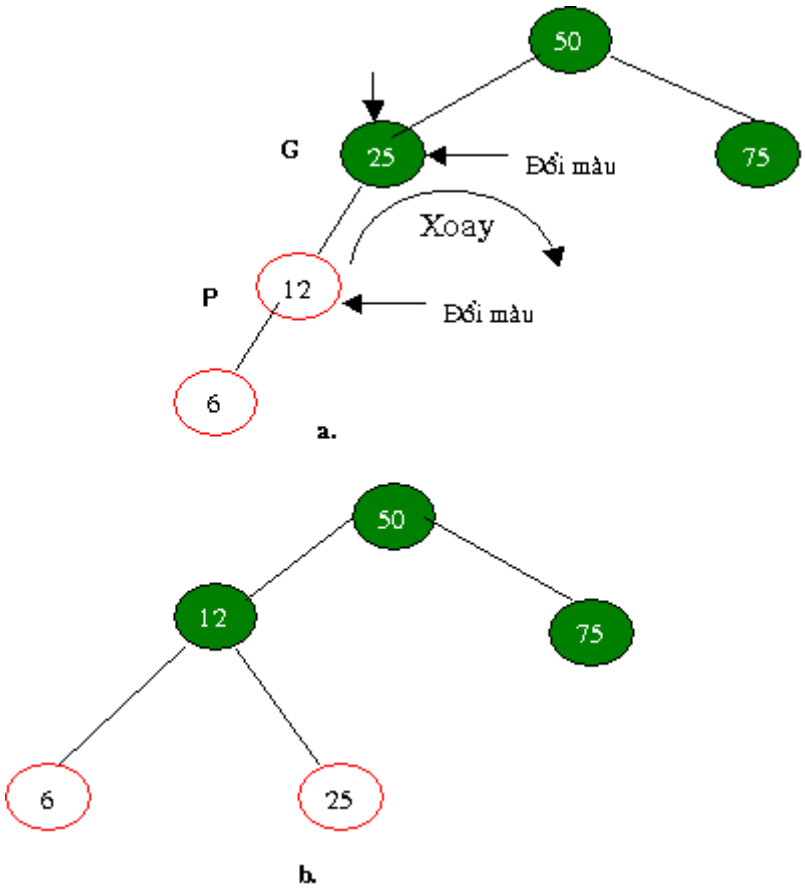
-Đổi màu node G - node ông bà của node X (trong thí dụ này là node 25).

-Đổi màu node P - node cha của node X (node 12)

-Quay với node G (25) ở vị trí đỉnh, theo hướng làm nâng node X lên (6). Đây là một phép quay phải.

Khi ta hoàn tất ba bước trên sẽ bảo toàn cây đỏ đen. Xem hình 7b.

Trong thí dụ này, node X là node cháu ngoại của một node con trái. Có một trường hợp đối xứng khi node X là node cháu ngoài nhưng của một node con phải. Thử làm điều này bằng cách tạo nên cây 50, 25, 75, 87, 93 (với phép lật màu khi cần). Chỉnh sửa cây bằng cách đổi màu node 75 và 87, và quay trái với node 75 là node đỉnh. Một lần nữa cây lại được cân bằng.

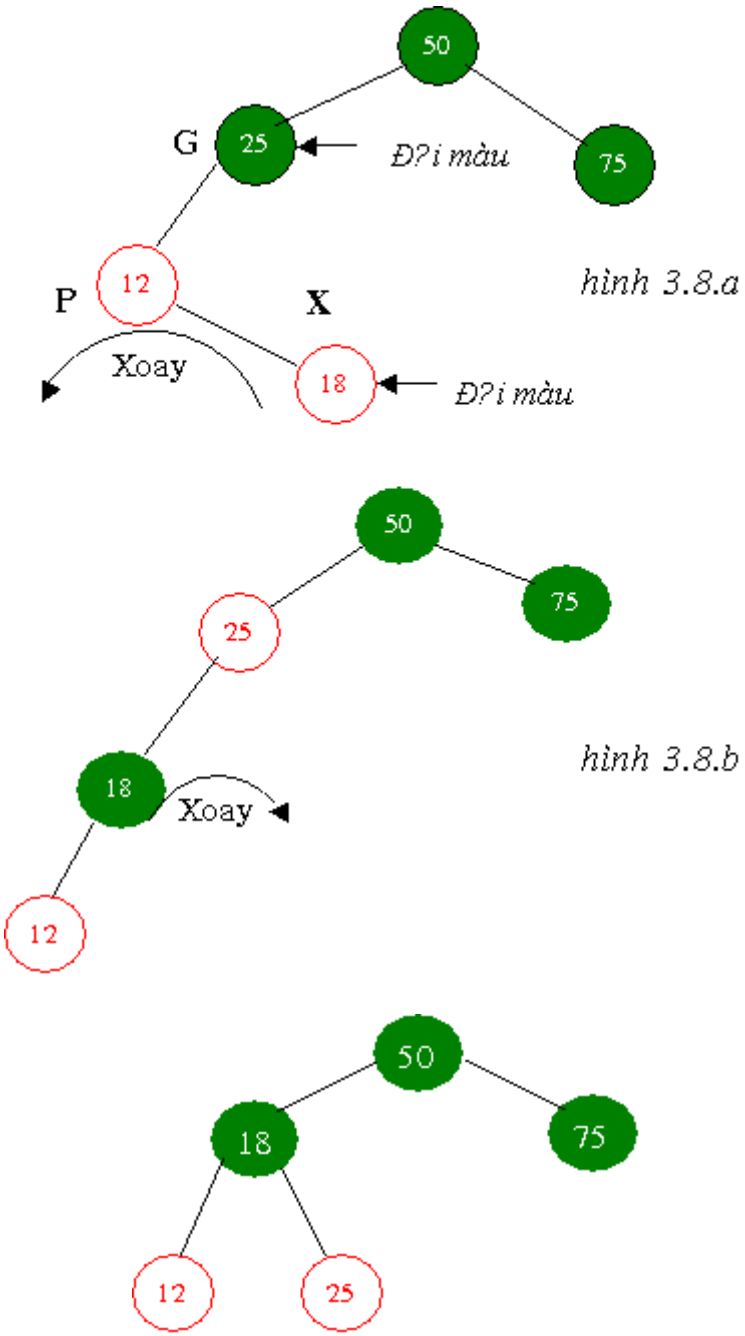


Hình 7. Node P đỏ và X là node cháu ngoại

iii) Khả năng 3: P đỏ và X là cháu nội của G

Nếu node P đỏ và X là node cháu nội, chúng ta cần thực hiện hai phép quay và một vài phép đổi màu. Cây đỏ đen được tạo thành từ các node 50, 25, 75, 12 và 18. (cần phải lật màu trước khi chèn node 12). Xem hình 8a.

Lưu ý là node 18 là node cháu nội. Node này và node cha đều đỏ (cha và con đều đỏ).



hình 8.c

Hình 8. Khả năng 3: P đỏ và X là node cháu nội

Chỉnh lại sự sắp xếp này cũng khá rắc rối hơn. Nếu ta cố quay phải node ông bà G (25) ở đỉnh, như ta đã làm trong khả năng 2, node cháu trong X (18) đi ngang hơn là đi lên, như thế cây sẽ không còn cân bằng như trước.

(Thử làm điều này, rồi quay trở lại, với node 12 ở đỉnh, để phục hồi cây nhu cũ). Phải cần một giải pháp khác.

Thủ thuật cần dùng khi X là node cháu nội là tiến hành hai phép quay hơn là một phép. Phép quay đầu biến X từ một node cháu nội thành node cháu ngoại, như trong hình 8b. Bây giờ, trường hợp là tương tự như khả năng 1, và ta có thể áp dụng cùng một phép quay, với node ông bà ở đỉnh, như đã làm trước đây. Kết quả như trong hình 8c.

Chúng ta cũng cần tô màu lại các nút. Ta làm điều này trước khi làm bất cứ phép quay nào (thứ tự không quan trọng, nhưng nếu ta đợi đến khi sau khi quay mới tô màu lại node thì khó mà biết phải gọi chúng như thế nào). Các bước là:

- Đổi màu node ông bà của node X (node 25).
- Đổi màu node X (node X đây là node 18).
- Quay trái với node P - node cha của X - ở đỉnh (node cha đây là 12).
- Quay lần nữa với node ông bà của X (25) ở đỉnh, về hướng nâng X lên (quay phải).

5. LOẠI BỎ NODE

Trong cây BST chúng ta thấy rằng phép loại bỏ phức tạp hơn so với phép thêm vào. Trong cây đỏ đen phép loại bỏ càng phức tạp hơn rất nhiều so với phép thêm vào vì yêu cầu đảm bảo quy tắc đỏ đen. Chúng ta có thể tham khảo trong phần cài đặt.

- Nếu xóa một nút đỏ thì chiều cao đen của cây không đổi
- Nếu xóa một nút đen thì chúng ta phải cân bằng lại cây.

6. TÍNH HIỆU QUẢ CỦA CÂY ĐỎ ĐEN

Giống như cây tìm kiếm nhị phân thông thường, cây đỏ đen có thể cho phép việc tìm kiếm, chèn và xóa trong thời gian $O(\log_2 N)$. Thời gian tìm kiếm là gần như bằng nhau đối với hai loại cây, vì những đặc điểm của cây đỏ đen không sử dụng trong quá trình tìm kiếm. Điều bất lợi là việc lưu trữ cần cho mỗi node tăng chút ít để điều tiết màu đỏ-đen (một biến boolean).

Đặc thù hơn, theo Sedgewick, trong thực tế tìm kiếm trên cây đỏ đen mất khoảng $\log_2 N$ phép so sánh, và có thể chứng minh rằng nó không cần hơn $2*\log_2 N$ phép so sánh.

Thời gian chèn và xóa tăng dần bởi một hằng số vì việc phải thực thi phép lật màu và quay trên đường đi xuống và tại những điểm chèn. Trung bình một phép chèn cần khoảng chừng một phép quay. Do đó, chèn hày còn chiếm $O(\log_2 N)$ thời gian, nhưng lại chậm hơn phép chèn trong cây nhị phân thường.

Bởi vì trong hầu hết các ứng dụng, có nhiều thao tác tìm kiếm hơn là chèn và xóa, có lẽ không có nhiều bất lợi về thời gian khi dùng cây đỏ đen thay vì cây nhị phân thuờng. Dĩ nhiên, điều thuận lợi là trong cây đỏ đen, dữ liệu đã sắp xếp không làm giảm hiệu suất $O(N)$.

Một trở ngại trong cây đỏ đen là việc cài đặt các phép toán phức tạp hơn so với cây BST. Chúng ta có thể tham khảo các phép toán thêm vào và loại bỏ trong phần cài đặt.