BÀI TẬP TUẦN 8

Môn học: Cấu Trúc Dữ Liệu và Giải Thuật.
GV hướng dẫn thực hành: Nguyễn Khánh Toàn (ktoan271199@gmail.com).
Nội dung chính:

- Cây AVL - một loại cây nhị phân tìm kiếm cây bằng

Thời gian thực hiện: **1** tuần.

Bài tập được biên soạn lại và tham khảo từ tài liệu thực hành môn Cấu Trúc Dữ Liệu và Giải Thuật của quý Thầy Cô Trợ Giảng khóa trước của bộ môn Khoa Học Máy Tính và quý Thầy Cô của bộ môn Công Nghệ Tri Thức. Trân trọng cảm ơn quý Thầy Cô của các bộ môn.

1 Giới thiệu cây AVL

Cây nhị phân tìm kiếm (Binary Search Tree) có thể sẽ bị suy biến thành mảng trong trường hợp xấu nhất, lúc này chiều cao cây bằng với số lượng phần tử của mảng khiến cho chi phí của thao tác tìm kiếm không được như mong muốn (độ phức tạp của quá trình tìm kiếm là $O(h)$ với h là chiều cao của cây). Chính vì vậy mà một trong những tính chất quan trọng cần bổ sung cho cây nhị phân tìm kiếm là *tính cân bằng*, để đảm bảo chiều cao của cây là $h = log(n)$ với n là tổng số nút của cây. Để đạt được tính chất này, cây AVL đã bổ sung hai điều:

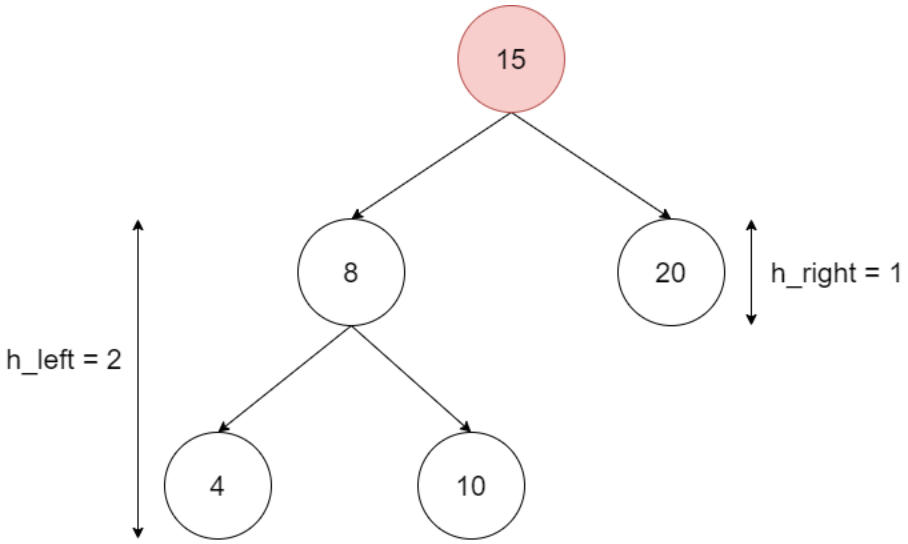
- Thêm dữ liệu để quản lý chiều cao tại mỗi nút (lưu ý lại rằng chiều cao tại mỗi nút là độ dài đường đi từ nút hiện tại đi xuống đến nút lá cao nhất) (Hình 1). Một khi đã lưu thuộc tính chiều cao cho mỗi nút, các bạn cần phải đảm bảo rằng giá trị của nó luôn đúng. Các thao tác có thể ảnh hưởng đến chiều cao tại mỗi nút: thêm nút, xóa nút.

```
struct Node{
    int key;
    //Luu y: Phan key nay se la phan de quan ly du lieu tai moi nut, no co the la 1 so nguyen, hoac 1 vector, v.v.

    int height; //chieu cao tai moi nut
    Node* left;
    Node* right;
}
```

Hình 1: Định nghĩa một nút trong cây AVL.

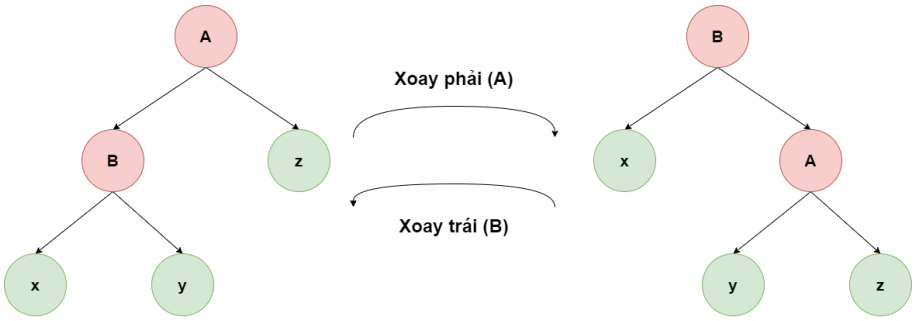
- Một khi đã quản lý được chiều cao tại mỗi nút, ta có thể biết rằng khi nào mỗi một nút bị mất cân bằng, bằng việc tính độ chêch lệch chiều cao giữa cây con trái và phải của nó (Hình 2). Nếu độ chênh lệch lớn hơn 1, nghĩa là tại nút đó đang xảy ra mất cân bằng. Lúc này ta sẽ phải thực hiện các thao tác xoay cây để cây quay trở lại trạng thái cân bằng tại nút đó.



Hình 2: Ví dụ về cây AVL và chiều cao của các nút con của nút gốc.

2 Thao tác xoay cây

Đối với cây AVL, có hai thao tác xoay cây: xoay trái và xoay phải. Các bạn quan sát hình vẽ xoay cây như sau (Hình 3)

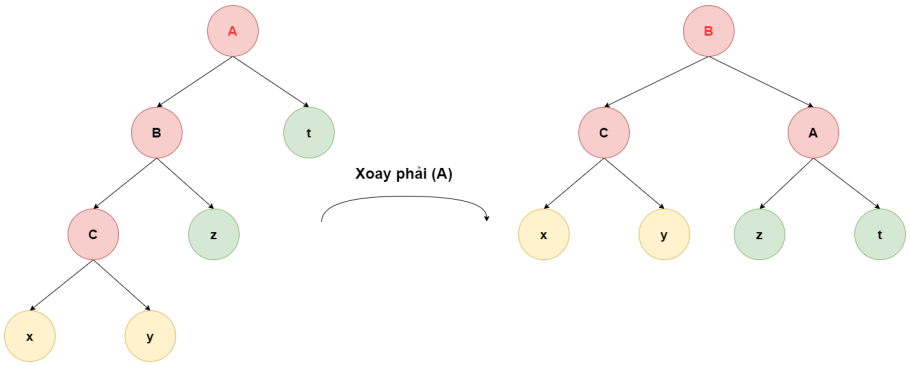


Hình 3: Các thao tác xoay cây.

Làm thế nào để cài đặt hàm xoay cây tại một nút? Ví dụ với hàm xoay trái,

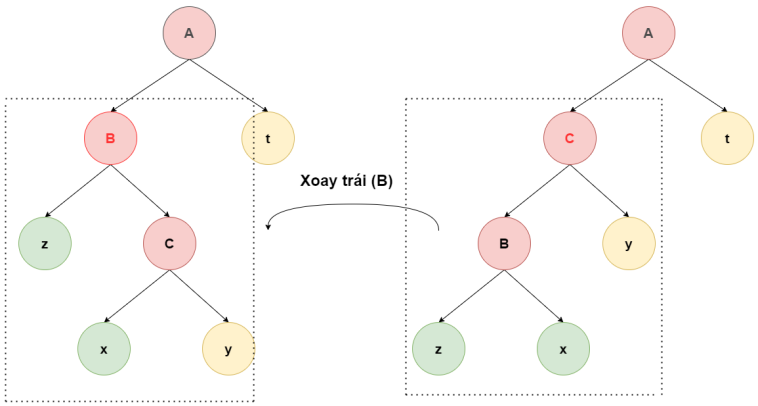
nút cần truyền vào sẽ là vị trí nút cần xoay, vậy nút trả về sẽ là gì? Chính là nút gốc trong Hình 3 (các bạn hình dung là xoay ở nút gốc và sẽ trả về nút gốc mới). Tại sao phải trả về? Để nút cha của (nút gốc - nút bị xoay) gán đúng nút con của nó với giá trị nút gốc mới. Ví dụ gọi nút cha của gốc là **PAR**, nút gốc hiện tại là A. Khi đó chẳng hạn ban đầu ta có **PAR** $\rightarrow$ **left** $= A$. Thực hiện xoay phải tại A, hàm này sẽ phải trả ra nút B, để khi gọi đệ quy quay trở lại việc gán giá trị nút **left** cho nút **PAR** thì ta có **PAR** $\rightarrow$ **left** $= B$.

1. **Trường hợp mất cân bằng Left-Left**. Các bạn quan sát hình vẽ sau (Hình 4):



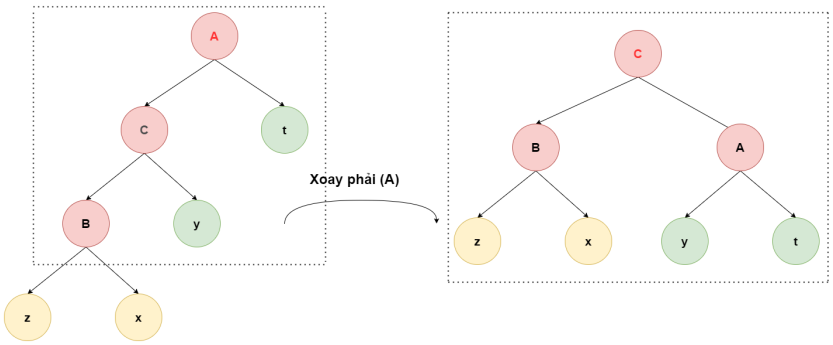
Hình 4: Xử lý cân bằng cây trong trường hợp mất cân bằng Left-Left.

2. **Trường hợp mất cân bằng Left-Right**. Ý tưởng sẽ là xoay cây để quay về trường hợp mất cân bằng Left-Left rồi áp dụng cân bằng cây cho trường hợp này (các bạn quan sát Hình 5).



Hình 5: Bước 1 trong xử lý trường hợp MCB Left-Right, cố gắng biến trường hợp Left-Right thành Left-Left.

Sau khi chuyển về được trường hợp Left-Left, các bạn sẽ thực hiện tương tự trường hợp 1 như sau (Hình 6):



Hình 6: Bước 2 trong xử lý trường hợp MCB Left-Right, thực hiện xoay phải tại nút gốc như trường hợp Left-Left.

3. **Trường hợp mất cân bằng Right-Right**. Tương tự như trường hợp MCB Left-Left, các bạn sẽ phải tự suy nghĩ để giải quyết.
4. **Trường hợp mất cân bằng Right-Left**.Tương tự như trường hợp MCB Left-Right, các bạn sẽ phải tự suy nghĩ để giải quyết.

Bây giờ các bạn sẽ cài đặt các hàm sau:

1. Xoay phải tại một nút:
   - **Node*** rightRotate(**Node*** A);
2. Xoay trái tại một nút:
   - **Node*** leftRotate(**Node*** B);

Lưu ý rằng các bạn nhớ thực hiện **cập nhật lại giá trị chiều cao tại các nút có thể bị thay đổi chiều cao** sau khi xoay cây.

3 Các hàm chức năng của cây AVL

Sau khi đã nắm được lý thuyết về xoay cây, các bạn thực hiện cài đặt các hàm chức năng sau của **cây AVL**. Lưu ý rằng các chữ ký hàm dưới đây chỉ là gợi ý, các bạn hoàn toàn có thể thay đổi, miễn hoàn thành đúng yêu cầu là được. Ngoài ra, các bạn cũng có thể sử dụng *class* để cài đặt cây AVL nếu đã có kiến thức về lập trình hướng đối tượng (OOP) (khuyến khích).

1. Khởi tạo một nút từ một giá trị cho trước, chiều cao nút mới bằng 1.
   - **NODE*** createNode(**int** data);

2. Thêm một nút mới với giá trị cho trước vào cây AVL (thông báo đã có giá trị nếu bị trùng).
    - **void** insert(**NODE*** &root, **int** x);
3. Xóa một nút với giá trị cho trước khỏi cây AVL (thông báo nếu giá trị đó không có trong cây).
    - **void** remove(**NODE*** &root, **int** x);
4. Tìm và trả về 1 NODE có giá trị cho trước trên cây AVL.
    - **NODE*** search(**NODE*** root, **int** x);
5. Kiểm tra xem 1 cây nhị phân cho trước có phải là 1 cây AVL hay không:
    - **bool** isAVL(**NODE*** root);
6. Xuất ra cây (gồm key và height) theo thao tác duyệt trước:
    - **void** NLR(**NODE*** root);
7. Xuất ra cây (gồm key và height) theo thao tác duyệt giữa:
    - **void** LNR(**NODE*** root);
8. Xuất ra cây (gồm key và height) theo thao tác duyệt sau:
    - **void** LRN(**NODE*** root);
9. Xuất ra cây (gồm key và height) theo thao tác duyệt từng tầng:
    - **void** levelOrder(**NODE*** root);
10. Đếm số lượng nút trong cây AVL mà có giá trị nhỏ hơn một giá trị cho trước.
    - **int** countLesser(**NODE*** root, int x);
11. Đếm số lượng nút trong cây AVL mà có giá trị lớn hơn một giá trị cho trước.
    - **int** countGreater(**NODE*** root, int x);
12. Đếm số lượng nút lá trong cây AVL.
    - **int** countLeaves(**NODE*** root);
13. Xóa toàn bộ cây AVL.
    - **int** removeTree(**NODE*** &root);

Hết