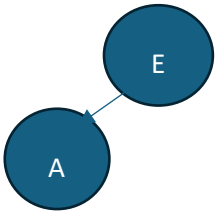
Câu 1 :

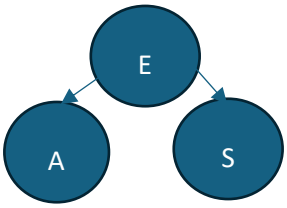
*CÂY AVL :

Chèn E :

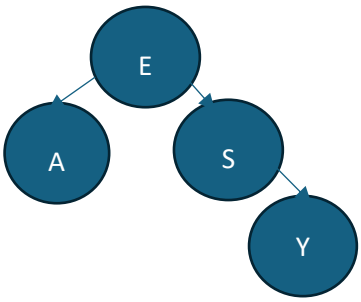
Chèn A, A nhỏ hơn E nên nằm trái E :



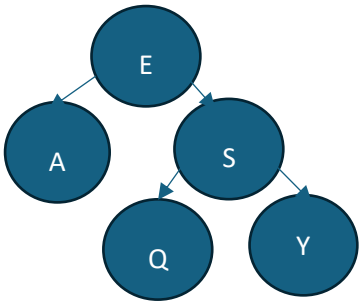
Chèn S, S lớn hơn E nên nằm phải E :



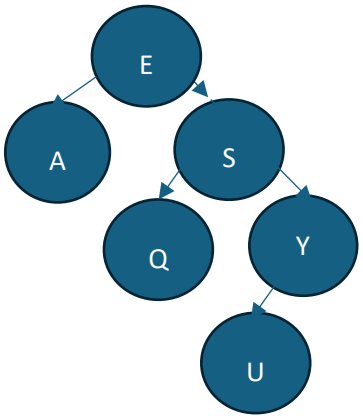
Chèn Y, Y lớn hơn E và lớn hơn S, nên nằm phải S :



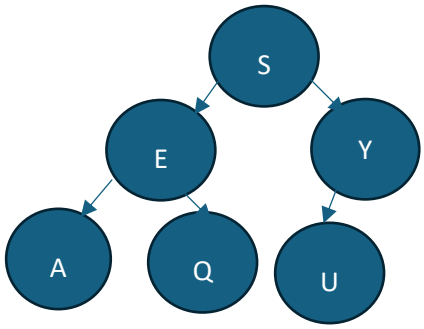
Chèn Q, Q lớn hơn E và nhỏ hơn S, nên nằm trái S :



Chèn U, U lớn hơn E, lớn hơn S, nhỏ hơn Y, nên nằm trái Y, nhưng khi này cây bị mất cân bằng tại E trường hợp phải – phải, vì chiều cao bên phải lớn hơn bên trái :



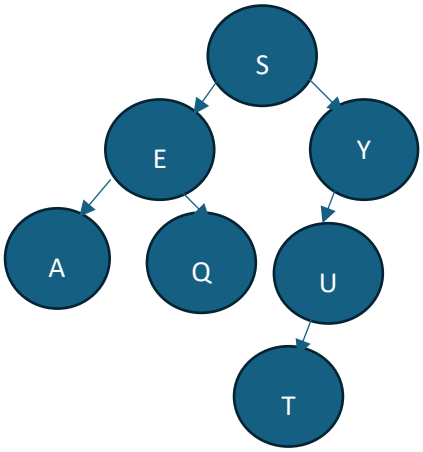
Quay trái để cân bằng :



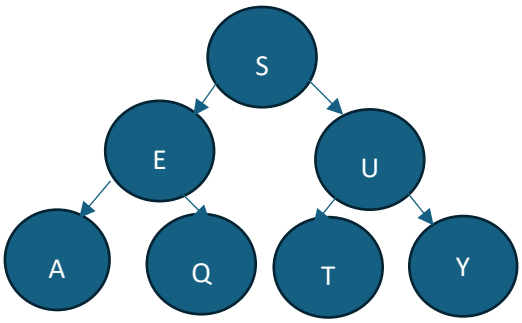
Chèn E, thực hiện tìm kiếm vị trí cho E, thấy E đã tồn tại, nên không thực hiện chèn.

Chèn S, thực hiện tìm kiếm vị trí cho S, thấy S đã tồn tại, nên không thực hiện chèn.

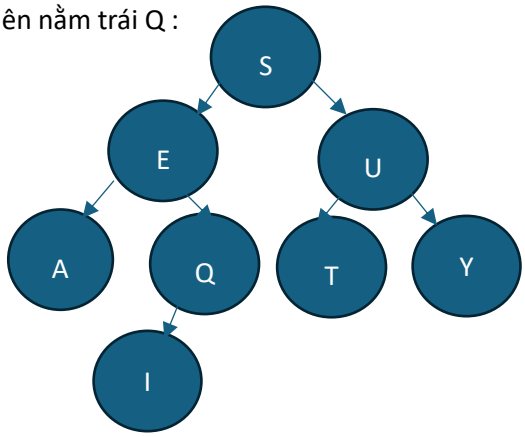
Chèn T, T lớn hơn S, nhưng nhỏ hơn Y và U, nên nằm trái U, gây mất cân bằng tại Y, trường hợp trái -trái , vì chiều cao bên trái lớn hơn bên phải :



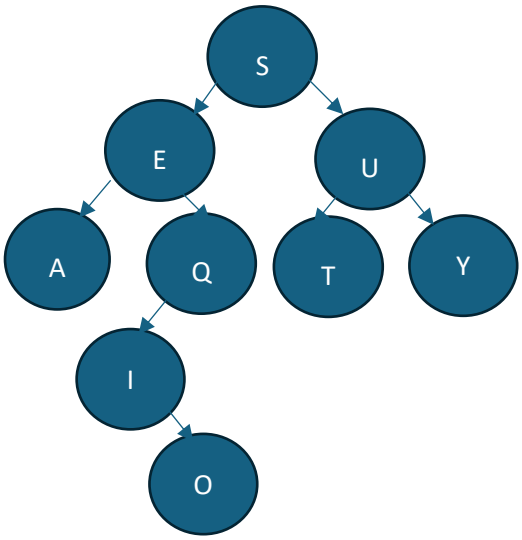
Quay phải tại Y để cân bằng :



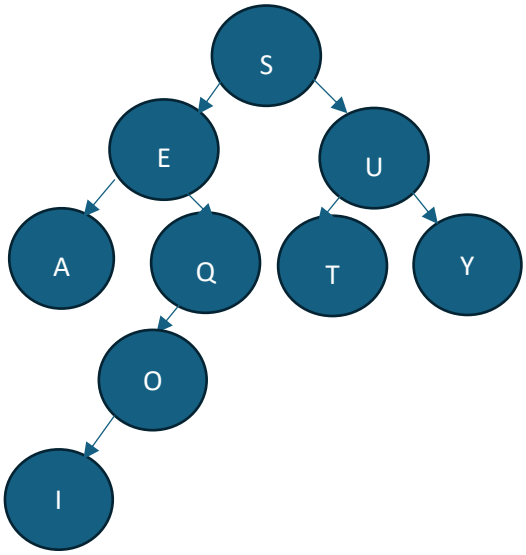
Chèn I vào, I nhỏ hơn S, lớn hơn E và nhỏ hơn Q nên nằm trái Q :



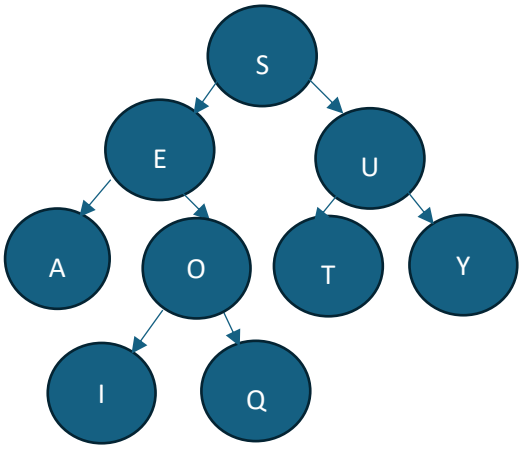
Chèn O vào, O nhỏ hơn S, lớn hơn E, nhỏ hơn Q, lớn hơn I nên nằm phải I, gây mất cân bằng tại Q, trường hợp trái – phải:



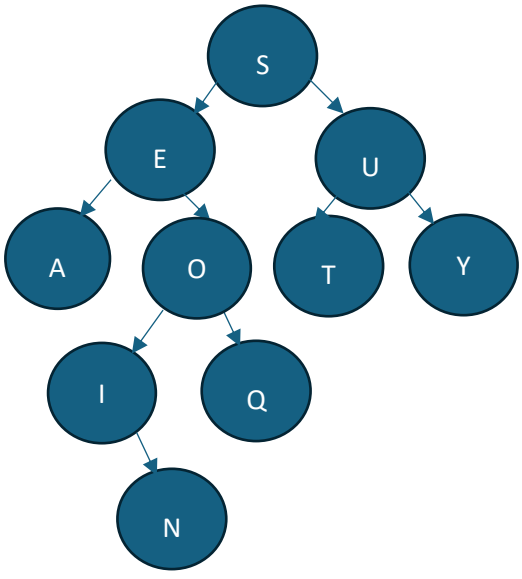
Quay trái tại I trước :



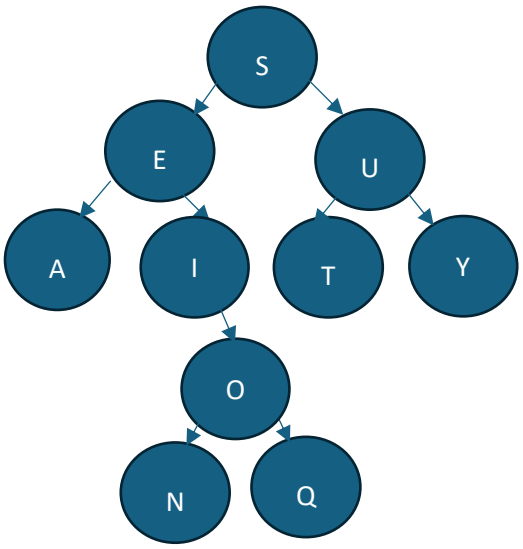
Quay phải tại Q :



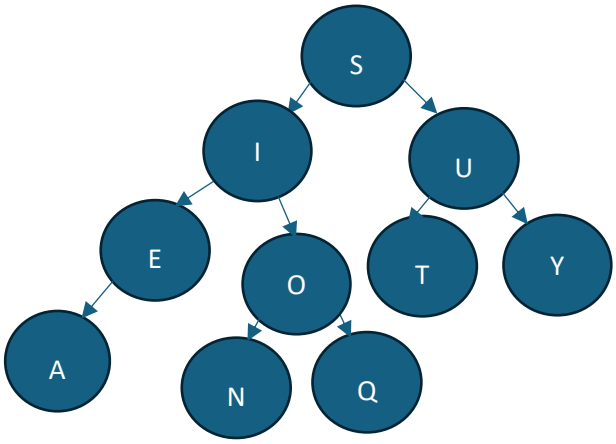
Chèn N vào, N nhỏ hơn S, lớn hơn E, nhỏ hơn O, lớn hơn I nên nằm phải I gây mất cân bằng tại E, trường hợp phải – trái, do chiều cao bên phải lớn hơn bên trái :



Quay phải tại O :



Quay trái tại E :



Vậy là đã hoàn thành cây AVL rồi ~

Xuất dãy từ khoá trên Cây AVL ra theo thứ tự tăng dần :

Ta sẽ duyệt lần lượt từ cây con trái nhất cho đến cây con phải nhất.

Dùng đệ quy.

Duyệt nhánh con trái của một nút :

Bắt đầu từ nút gốc, tìm nhánh con trái của nút hiện tại và tiếp tục đi sâu xuống nhánh con trái đó.

Nếu nhánh con không có nút nào (là lá) thì in ra.

Hoặc nếu không có nhánh trái (chỉ có nhánh phải) thì đi tiếp qua nhánh phải.

Nếu nút đã được in rồi thì di chuyển lại lên trên nút gần kề.

Cho đến khi tìm được lá, hoặc nút không phải lá nhưng các nút nhánh của nó đã được in ra rồi, thì in nút đó ra.

Sau đó, truy cập nút gốc :

Sau khi đã duyệt hết các nút con bên trái, quay lại và truy cập nút gốc hiện tại và in ra

.

Cuối cùng, duyệt nhánh con phải của nút đó

Sau khi truy cập nút gốc, tiếp tục sang nhánh con phải của nút hiện tại và lặp lại các bước :

Tiếp tục đi sâu xuống nhánh con trái.

Nếu nhánh con không có nút nào (là lá) thì in ra.

Hoặc nếu không có nhánh trái (chỉ có nhánh phải) thì đi tiếp qua nhánh phải.

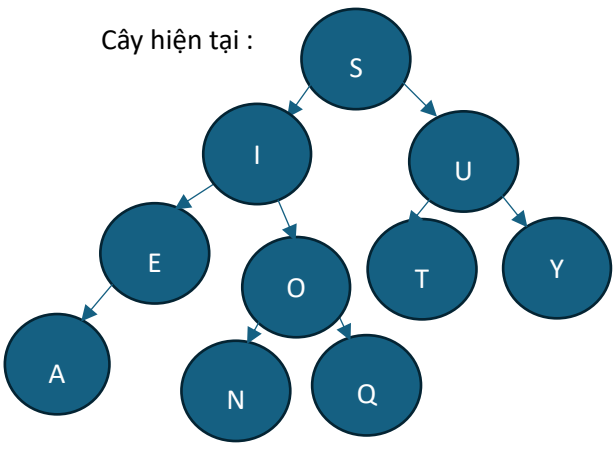
Nếu nút đã được in rồi thì di chuyển lại lên trên nút gần kề.

Cho đến khi tìm được lá, hoặc nút không phải lá nhưng các nút nhánh của nó đã được in ra rồi, thì in nút đó ra.

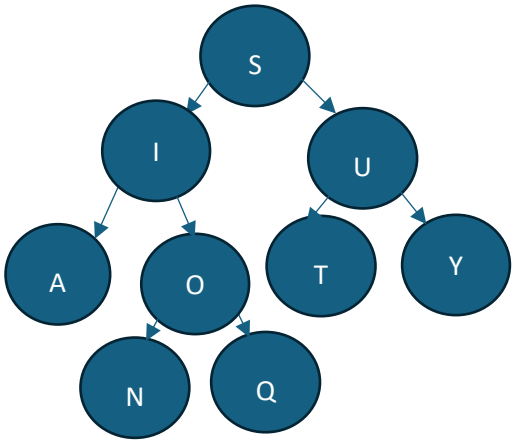
```
void inOrder(Node* root)
{
	if (root != nullptr)
	{
		inOrder(root->left);
		cout << root->key << " ";
		inOrder(root->right);
	}
}
```

Thao tác xoá Node chứ khoá kí tự E và T, hiệu chỉnh cây để đạt trạng thái bền vững :

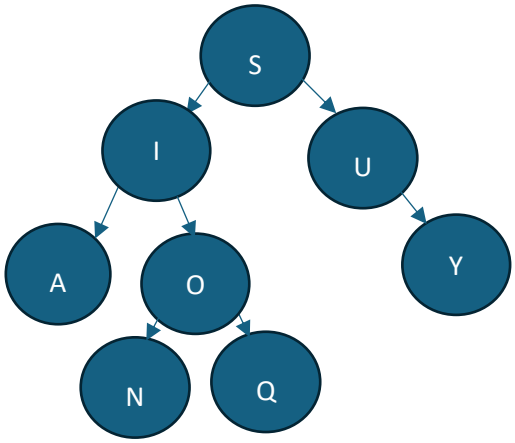
Cây hiện tại :



Khi xoá E, nhánh trái I hiện tại là A, chiều cao trái phải của I chênh lệnh 1, nên vẫn cân bằng :



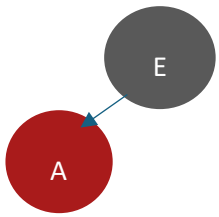
Khi xoá T, nhánh trái của U không còn T, U chỉ còn nhánh phải chứa Y, chiều cao trái phải của S chênh lệch nhau 1, nên vẫn cân bằng :



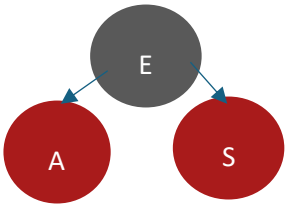
*CÂY RED-BLACK :

Chèn E (E là gốc nên là màu đen) :

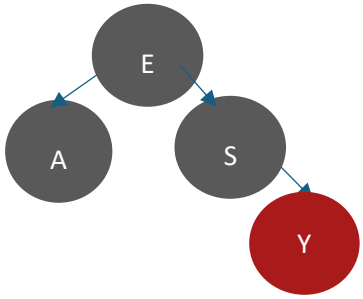
Chèn A, A nhỏ hơn E nên nằm trái E : (mới tạo nên sẽ là màu đỏ)



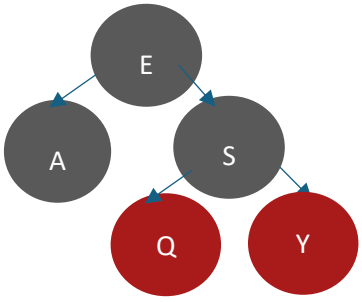
Chèn S, S lớn hơn E nên nằm phải E : (mới tạo nên sẽ là màu đỏ)



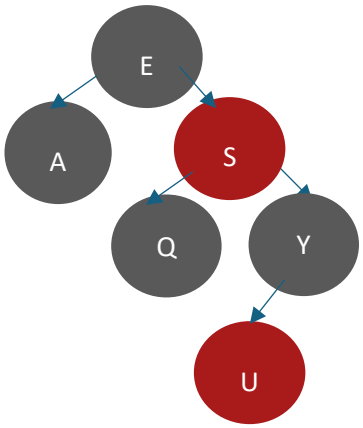
Chèn Y, Y lớn hơn E và lớn hơn S, nên nằm phải S , Y mới tạo nên là màu đỏ, gốc thì luôn có màu đen, đổi màu cho A, S thành màu đen để đảm bảo từ 1 nút đến lá NULL bất kì đều có số lượng nút đen bằng nhau là 2.



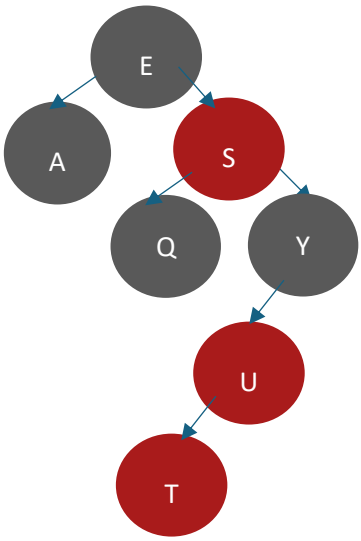
Chèn Q, Q lớn hơn E và nhỏ hơn S, nên nằm trái S , Q mới tạo nên là màu đỏ (đảm bảo lá sẽ màu đen theo quy tắc)



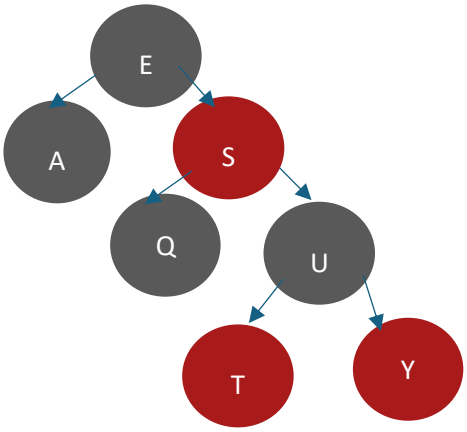
Chèn U, U lớn hơn E và lớn hơn S, nhỏ hơn Y, nên nằm trái Y, mới tạo nên có màu đỏ, S chuyển sang màu đỏ để đảm bảo từ 1 nút đến lá NULL bất kì đều có số lượng nút đen bằng nhau là 2.



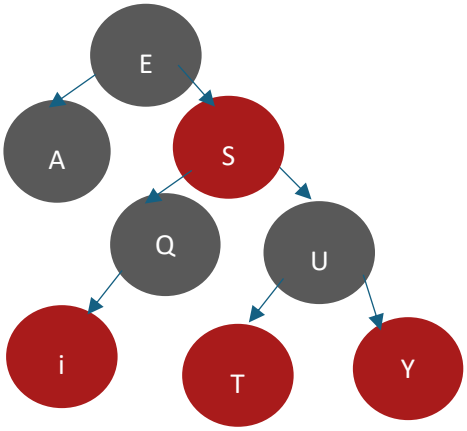
Chèn T, T lớn hơn E và lớn hơn S, nhỏ hơn Y, nhỏ hơn U nên nằm trái U, mới tạo nên có màu đỏ, gây mất cân bằng tại Y, do chiều cao trái phải lệch nhau 2, trường hợp trái - trái



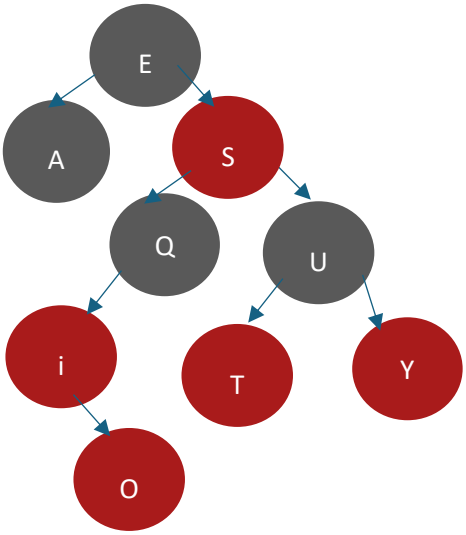
Quay phải tại Y, đổi màu U và Y để đảm bảo bảo từ 1 nút đến lá NULL bất kì đều có số lượng nút đen bằng nhau là 2.



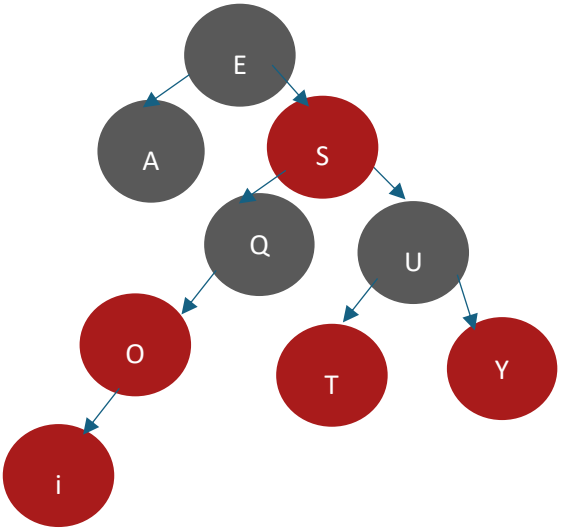
Chèn I vào, lớn hơn E, nhỏ hơn S, nhỏ hơn Q, nên nằm trái Q



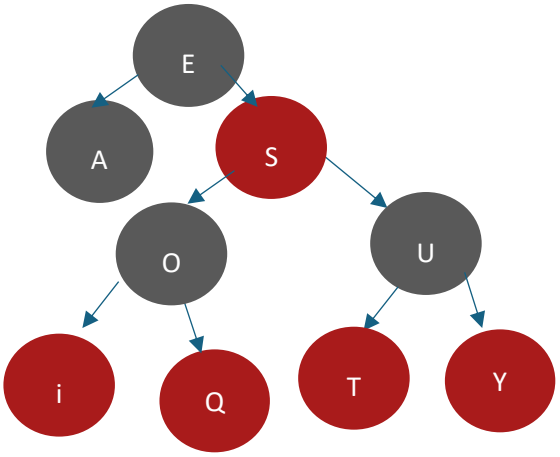
Chèn I vào, lớn hơn E, nhỏ hơn S, nhỏ hơn Q, lớn hơn I nên nằm phải I, không cân bằng tại Q vì chiều cao trái phải lệch 2 , mất cân bằng trường hợp trái- phải:



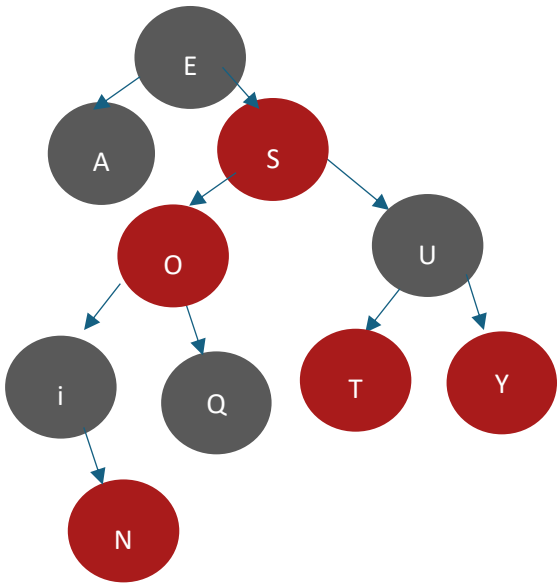
Quay trái tại I :



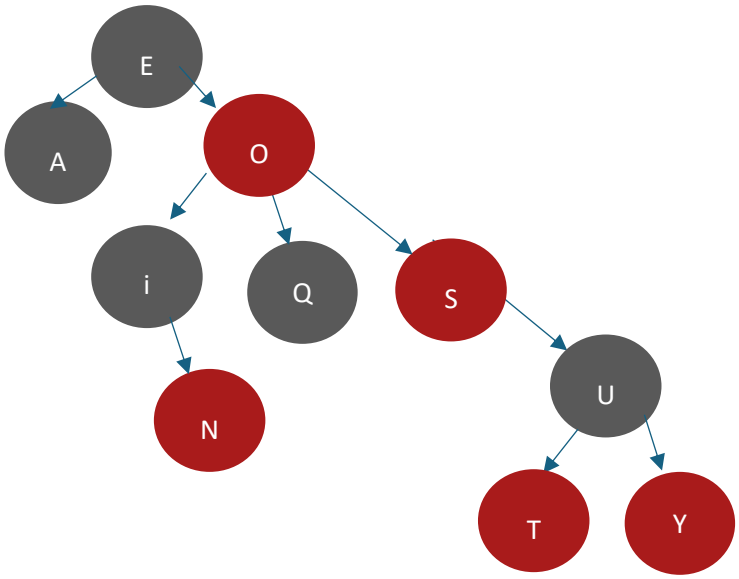
Quay phải tại Q , đổi màu O, Q đảm bảo bảo từ 1 nút đến lá NULL bất kì đều có số lượng nút đen bằng nhau là 2.



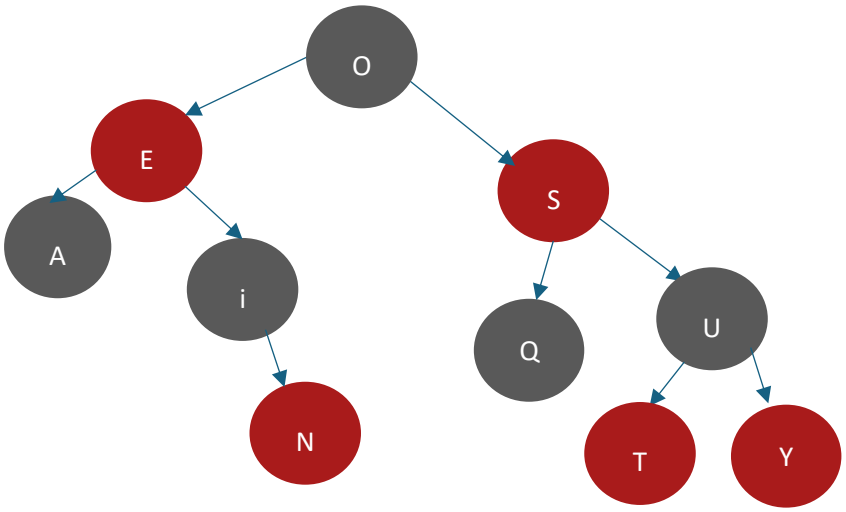
Chèn N, lớn hơn E, nhỏ hơn S, nhỏ hơn O, lớn hơn I nên nằm phải I, đổi màu sao cho đảm bảo từ 1 nút đến lá NULL bất kì đều có số lượng nút đen bằng nhau là 2.



2 nút S và O cùng màu đỏ, vi phạm điều kiện nên quay phải tại S, sau khi quay :

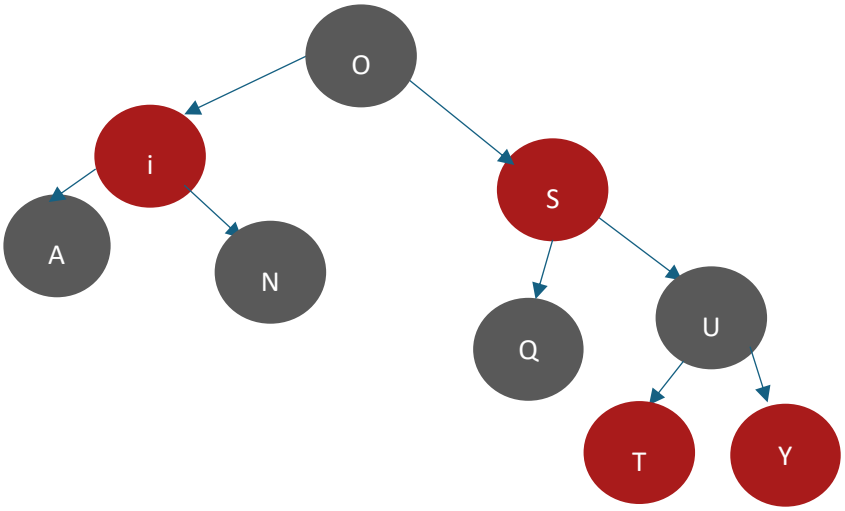


Nút O và S cùng màu. Mất cân bằng tại E, thực hiện quay trái và đổi màu để đảm bảo từ 1 nút đến lá NULL bất kì đều có số lượng nút đen bằng nhau là 2.

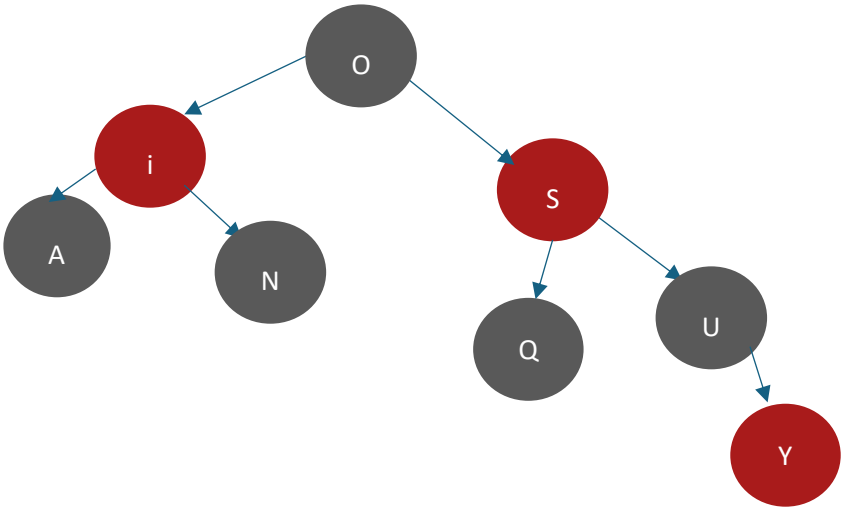


Xoá E :

Do xóa nút E làm giảm số lượng nút đen của một nhánh xoay trái tại A



Xoá T :



Câu 2 :

Kiểm tra cây AVL:

```
int checkAVL(Node* root, bool& check)
{
    if (root == nullptr)
        return 0;

    int l = checkAVL(root->trai, check);
    int r = checkAVL(root->phai, check);

  if (!check) return 0;

    if (abs(l - r) > 1)
        check = false;
    return 1 + max(l, r);
}
```

```
bool isAVL(Node* root)
{
   bool isBalanced = true;
   checkAVL(root, isBalanced);
   return isBalanced;
}
```