**Câu 1:**

*Cây AVL:*

* Chèn E: Cây đang rỗng: E thành gốc

E

* Chèn A: A < E (theo mã ascii) nên di chuyển A sang trái E. Do cây con trái E là NULL nên ta chèn A vào cây con trái E

E

A

\* Chèn S: S > E, di chuyển sang cây con phải E. Do E không có cây con phải, chèn S vào cây con phải E

E

A S

* Chèn Y: Y > E, di chuyển sang cây con phải E. Y > S, di chuyển Y sang cây con phải S. Cây con phải Y là NULL, chèn Y vào cây con phải S

E

A S

Y

* Chèn Q: Q > E, di chuyển sang cây con phải E. Q < S, di chuyển sang cây con trái S. Cây con trái S là NULL, chèn Q vào vị trí NULL này

E

A S

Q Y

\* Chèn U: U > E, duyệt sang cây con phải E. U > S, duyệt sang cây con phải S. U < Y, duyệt sang cây con trái Y. Cây con trái Y là NULL chèn U vào vị trí này.

E

A S

Q Y

U

Duyệt từ U về gốc thấy E mất cân bằng phải – phải, thực hiện RR gốc E

S

E Y

A Q U

* Chèn E: duyệt đệ quy từ root thấy E đã tồn tại trong cây nên không chèn E
* Chèn S: duyệt đệ quy từ root thấy S đã tồn tại trong cây nên không chèn S
* Chèn T: T > S, di chuyển sang cây con phải S. T < Y, di chuyển sang cây con trái Y. T < U, di chuyển sang cây con trái U. cây con trái U là NULL, chèn T vào đây

S

E Y

A Q U

T

Cây mất câm bằng trái trái tại Y, thực hiện LL tại Y

S

E U

A Q T Y

\* Chèn I: I < S, di chuyển sang cây con trái S. I > E, di chuyển sang cây con phải E. I < Q, di chuyển sang cây con trái Q. Cây con trái Q là NULL, chèn I vào đây.

S

E U

A Q T Y

I

* Chèn O: tương tự bước tìm ví trí NULL và chèn ở trên

S

E U

A Q T Y

I

O

Cây mất cân bằng trái phải tại Q, thực hiện LR tại Q (quay trái tại nút I, quay phải tại nút Q)

S

E U

A O T Y

I Q

* Chèn N: Tìm vị trí NULL thích hợp và chèn N vào

S

E U

A O T Y

I Q

N

Cây mất cân bằng phải trái tại E, thực hiện RL tại E (quay phải O(con phải), quay trái E)

S

I U

E O T Y

A N Q

-Mô tả:

+ lần lượt chèn từng nút vào cây:

>nếu cây là rỗng thì nút vừa chèn sẽ là gốc của cây.

> nếu cây không rỗng thì dùng đệ quy xét giá trị của nút cần chèn và nút đang duyệt:

= Nếu giá trị nút chèn lớn hơn thì di chuyển sang nút con phải nút đang xét

= Nếu giá trị nút bé lớn hơn thì di chuyển sang nút con trái nút đang xét

= Nếu giá trị nút chèn bằng giá trị nút đang xét thì kết thúc việc chèn

* Nếu tìm được vị trí NULL thì chèn nút vào đó
* Duyệt ngược từ nút vừa chèn lên nút gốc để kiểm tra tính cân bằng của cây. Nếu tìm thấy nút mất cân bằng thì ta thực hiện xoay cây theo 1 trong các phương pháp: LL, LR, RR, RL tùy vào tình trạng lệch của cây sau
* Muốn xuất dãy từ khóa theo thứ tự tăng dần: duyệt trung tự

+ Nếu nút gốc khác NULL, dùng đệ quy duyệt từ nút gốc lần lượt thăm và xuất giá trị cây con trái, thăm và xuất giá trị nút đang xét, thăm và xuất giá trị cây con phải

void PrintTree\_InOrder(TNode\* root) {

if (root != NULL) {

PrintTree\_InOrder(root->pLeft);

cout << "\n" << root->key;

PrintTree\_InOrder(root->pRight);

}

}

* Xóa node E

S

I U

A O T Y

N Q

Cây cân bằng nên không cần xoay

* Xóa T:

S

I U

A O Y

N Q

Cây cân bằng nên không cần xoay

+ Mô tả cách làm:

>Nếu root == NULL kết thúc việc tìm kiếm phần tử xóa

>Nếu root != NULL

>> Dùng đệ quy duyệt từ gốc tìm vị trí xóa

>>> Nếu tìm thấy thì xét:

>>>> Nếu nút xóa là lá thì xóa bình thường, không ảnh hưởng các nút khác

>>>>Nếu nút xóa có 1 cây con duy nhất thì lưu địa chỉ của nút con nó cho cây con trái hoặc cây con phải cây cha

>>>> Nếu nút xóa có 2 cây con thì tìm phần tử thế mạng( nút con trái nhất bên phải hoặc phải nhất bên trái) chuyển phần tử thế mạng này lên vị trí xóa rồi xóa nút cần xóa ban đầu như bình thường

>> Nếu giá trị cần xóa lớn hơn giá trị nút đang xét -> duyệt sang phải

>> Nếu giá trị cần xóa bé hơn giá trị nút đang xét -> duyệt sang phải

*Cây Red – Black*

* Cây đang NULL, chèn 1 node vào -> node đó sẽ là root

E

* Chèn A: A < E, di chuyển sang nút con trái E. Nút con trái E là NULL, chèn A vào đây, do E là cha đen nên A thành con đỏ

E

A

Chèn S: S > E, chuyển sang node con phải E. Node con phải E là NULL, chèn S vào đây, cập nhật màu S là đỏ do S là con của E nhưng vi phạm 2 màu đỏ liên tiếp nên cập nhật màu lại

EE

A S E

Y Y(đỏ)

nhưng vi phạm 2 màu đỏ liên tiếp nên cập nhật màu lại

E(đen)

A(đen) S(đen)

Y(đỏ)

E(đen)

A(đen) S(đen)

Q(đỏ) Y(đỏ)

* Chèn Y: Y > E, chuyển sang con phải E. Y > S, chuyển sang con phải E. Con phải E là NULL, chèn Y vào đây, cập nhật màu Y là đen do Y là con S đỏ

**Câu 2:**

int HighTree(TNode\* root)

{

if (!root) return 0;

int hl = HighTree(root->pLeft);

int hr = HighTree(root->pRight);

if (hl > hr)

return (1 + hl);

else

return (1 + hr);

}

bool isAVL(TNode\* p) {

if (p != NULL) {

if (abs(HighTree(p->pLeft) - HighTree(p->pRight)) == 2) return false;

else

{

return isAVL(p->pLeft) && isAVL(p->pRight);

}

}

return true;

}