CÂY CÂN BẰNG AVL

MỤC TIÊU

Hoàn tất bài thực hành này, sinh viên có thể:

- Hiểu được các thao tác quay cây (quay trái, quay phải) để hiệu chỉnh cây thành cây cân bằng.
- Cài đặt hoàn chỉnh cây cân bằng AVL.

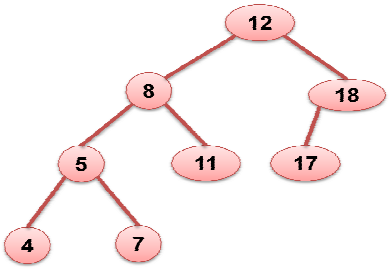
Thời gian thực hành: 120 phút – 360 phút

Lưu ý: Sinh viên phải thực hành bài tập về Cây nhị phân và Cây nhị phân tìm kiếm trước khi làm bài này.

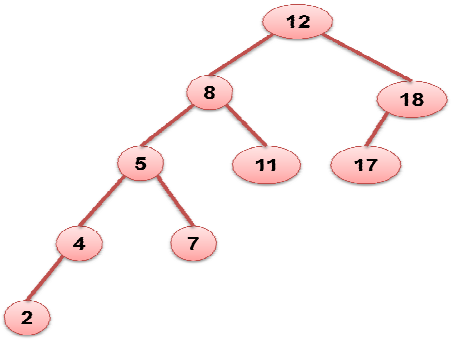
TÓM TẮT

Cây cân bằng AVL là ***cây nhị phân tìm kiếm*** (NPTK) mà tại mỗi đỉnh của cây, **độ cao** của cây con trái và cây con phải ***khác nhau không quá 1***.

Ví dụ 1: cây cân bằng AVL

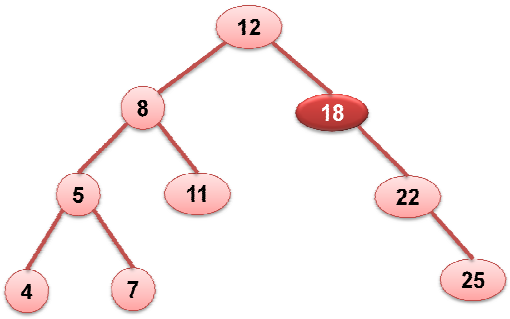


Ví dụ 2: cây không cân bằng

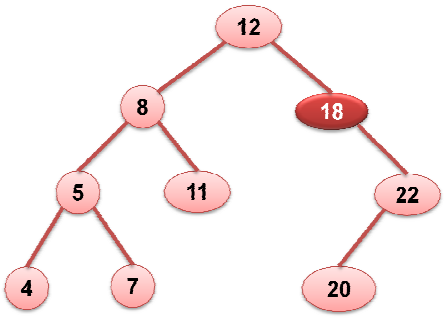


Khi thêm node mới vào cây AVL có thể xảy ra các trường hợp mất cân bằng như sau:

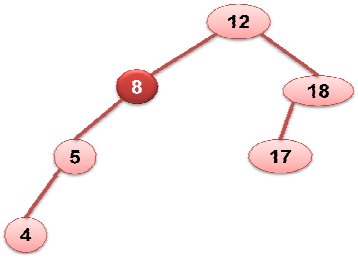
Mất cân bằng phải-phải (R-R)



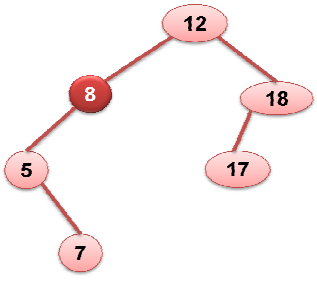
Mất cân bằng phải-trái (R-L)



Mất cân bằng trái-trái (L-L)

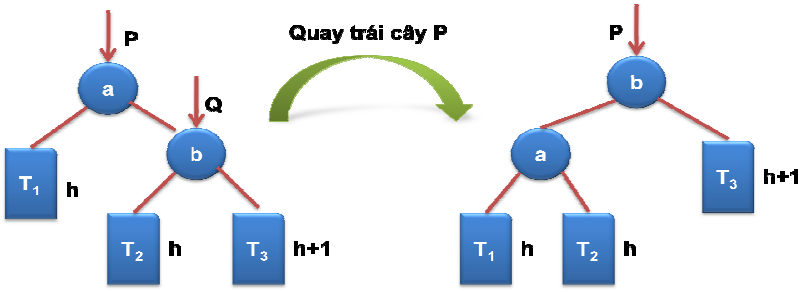


Mất cân bằng trái-phải (L-R)

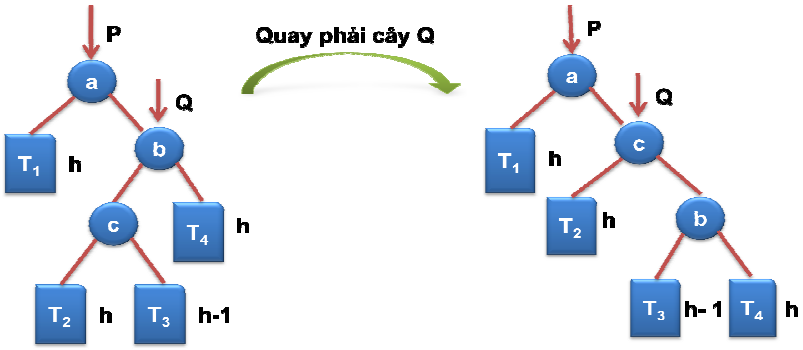


Xử lý mất cân bằng bằng cách sử dụng các phép quay cây

a. Quay trái



b. Quay phải



Xử lý cụ thể cho các trường hợp mất cân bằng như sau:

MẤT CÂN BẰNG PHẢI	
Mất cân bằng phải-phải (R-R) - Quay trái tại node bị mất cân bằng	Mất cân bằng phải-trái (R-L) - Quay phải tại node con phải của node bị mất cân bằng - Quay trái tại node bị mất cân bằng
MẤT CÂN BẰNG TRÁI	
Mất cân bằng trái-trái (L-L) - Quay phải tại node bị mất cân bằng	Mất cân bằng trái-phải (L-R) - Quay trái tại node con trái của node bị mất cân bằng - Quay phải tại node bị mất cân bằng

Giống với cây NPTK, các thao tác trên cây cân bằng bao gồm:

- Thêm phần tử vào cây
- Tìm kiếm 1 phần tử trên cây
- Duyệt cây
- Xóa 1 phần tử trên cây

NỘI DUNG THỰC HÀNH

Cơ bản

Sinh viên đọc kỹ phát biểu bài tập và thực hiện theo hướng dẫn:

Tổ chức một cây cân bằng AVL trong đó mỗi node trên cây chứa thông tin dữ liệu nguyên.

Người dùng sẽ nhập các giá trị nguyên từ bàn phím. Với mỗi giá trị nguyên được nhập vào, phải tạo cây AVL theo đúng tính chất của nó. Nếu người dùng nhập -1 quá trình nhập dữ liệu sẽ kết thúc.

Sau đó, xuất thông tin các node trên cây.

*Khi chương trình kết thúc, tất cả các node trên cây bị **xóa bỏ** khỏi bộ nhớ.*

Phân tích

- Các node trên cây cân bằng cũng giống như các node trên cây NPTK. Tuy nhiên, do mỗi lần thêm node vào cây chúng ta cần kiểm tra độ cao của node vừa thêm để kiểm soát tính cân bằng của cây nên cần bổ sung thêm giá trị cho biết sự cân bằng tại node đó vào cấu trúc của node. Cụ thể như sau:

```
struct AVLNODE
{
	int key;
	int bal; // thuộc tính cho biết giá trị cân bằng
	// 0: cân bằng, 1: lệch trái, 2: lệch phải
	NODE* pLeft;
	NODE* pRight;
};
```

- Các thao tác cần cài đặt: **xoay trái** cây (**RotateLeft**), **xoay phải** cây (**RotateRight**), **thêm** 1 node mới vào cây (**InsertNode**), **duyệt** cây theo (**Traverse**), **xóa toàn bộ node** trên cây (**RemoveAll**)
- Để chương trình mạch lạc và rõ ràng hơn, chúng ta sẽ cài đặt 2 hàm xử lý cân bằng khi cây lệch trái và lệch phải theo bảng phân loại ở trang 2. Như vậy trong chương trình sẽ có thêm 2 hàm **BalanceLeft** và **BalanceRight**.

Chương trình tham khảo

```
#include <stdio.h>

struct AVLNODE
{
	int Key;
	int bal; // thuộc tính cho biết giá trị cân bằng
	// 0: cân bằng, 1: lệch trái, 2: lệch phải
	AVLNODE* pLeft;
	AVLNODE* pRight;
};

AVLNODE* CreateNode(int Data)
{
	AVLNODE* pNode;
	pNode = new AVLNODE; //Xin cấp phát bộ nhớ động để tạo một phần tử (node)
mới
	if (pNode == NULL){
		return NULL;
	}
	pNode->Key = Data;
	pNode->pLeft = NULL;
	pNode->pRight = NULL;
	pNode->bal = 0; //Ghi chú: giải thích ý nghĩa của thao tác này
	return pNode;

}

void LeftRotate(AVLNODE* &P)
{
```

```
	AVLNODE *Q;
	Q = P->pRight;
	P->pRight = Q->pLeft;
	Q->pLeft = P;
	P = Q;
}
void RightRotate(AVLNODE* &P)
{
	//Ghi chú: sinh viên tự code cho hàm này
}
void LeftBalance(AVLNODE* &P)
{
	switch(P->pLeft->bal){
	case 1: //mất cân bằng trái trái
		RightRotate(P);
		P->bal = 0;
		P->pRight->bal = 0;
		break;
	case 2: //Ghi chú: cho biết đây là trường hợp mất cân bằng nào?
		LeftRotate(P->pLeft);
		RightRotate(P);
		switch(P->bal){
		case 0:
			P->pLeft->bal= 0;
			P->pRight->bal= 0;
			break;
		case 1:
			P->pLeft->bal= 0;
			P->pRight->bal= 2;
			break;
		case 2:
			P->pLeft->bal= 1;
			P->pRight->bal= 0;
			break;
		}
		P->bal = 0;
		break;
	}
}

void RightBalance(AVLNODE* &P)
{
	switch(P->pRight->bal){
	case 1: //Ghi chú: cho biết đây là trường hợp mất cân bằng nào?
		RightRotate(P->pRight);
		LeftRotate(P);
		switch(P->bal){
		case 0:
			P->pLeft->bal= 0;
			P->pRight->bal= 0;
			break;
		case 1:
			P->pLeft->bal= 0;
			P->pRight->bal= 2;
			break;
		case 2:
			P->pLeft->bal= 1;
			P->pRight->bal= 0;
			break;
		}
		P->bal = 0;
		break;
	case 2: //Ghi chú: cho biết đây là trường hợp mất cân bằng nào?
		LeftRotate(P);
```

```
            P->bal = 0;
            P->pLeft->bal = 0;
            break;
      }
}
int InsertNode(AVLNODE* &tree, int x)
{
      int res;
      if(tree==NULL){ //Ghi chú: cho biết ý nghĩa của câu lệnh này
            tree = CreateNode(x);
            if(tree==NULL){
                  return -1; //thêm ko thành công vì thiếu bộ nhớ
            }
            return 2;//thêm thành công và làm tăng chiều cao cây
      }
      else {
            if(tree->Key==x){
                  return 0; //khóa này đã tồn tại trong cây
            }
            else if(tree->Key > x){
                  res = InsertNode(tree->pLeft,x);
                  if(res < 2) {
                        return res;
                  }
                  switch(tree->bal){ //Ghi chú: giải thích ý nghĩa của câu lệnh
switch này
                  case 0:
                        tree->bal = 1;
                        return 2;
                  case 1:
                        LeftBalance(tree);
                        return 1;
                  case 2:
                        tree->bal = 0;
                        return 1;
                  }
            }
            else{
                  res = InsertNode(tree->pRight,x);
                  if(res<2){
                        return res;
                  }
                  switch(tree->bal){
                  case 0:
                        tree->bal=2;
                        return 2;
                  case 1:
                        tree->bal = 0;
                        return 1;
                  case 2:
                        RightBalance(tree);
                        return 1;
                  }
            }
      }
}
void Traverse(AVLNODE* t)
{
      if(t!=NULL)
      {
            Traverse(t->pLeft);
            printf("Khoa: %d, can bang: %d\n", t->Key,t->bal);
            Traverse(t->pRight);
      }
```

```
}
void RemoveAll(AVLNODE* &t)
{
	if(t!=NULL){
		RemoveAll(t->pLeft);
		RemoveAll(t->pRight);
		delete t;
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	AVLNODE *tree;
	//Ghi chu: Tại sao lại phải thực hiện phép gán phía dưới?
	tree = NULL;
	int Data;
	do
	{
		printf("Nhap vao du lieu, -1 de ket thuc: ");
		scanf("%d", &Data);
		if (Data == -1)
			break;
		InsertNode(tree, Data);
	}while (Data != -1);

	printf("\nCay AVL vua tao: \n");

	Traverse(tree);

	RemoveAll(tree);

	return 0;
}
```

Yêu cầu

1. Biên dịch đoạn chương trình nêu trên.

2. Cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:

-1

10 30 35 32 20 8 -1

30 40 50 -10 -5 -1

3. Nhận xét trình tự các node được xuất ra màn hình? Giải thích tại sao lại in ra được trình tự như nhận xét?

4. Sinh viên hoàn tất hàm **RightRotate** trong source code.

Gợi ý: RightRotate tương tự hàm LeftRotate.

5. Biên dịch lại chương trình sau khi hoàn thành câu 3 và cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:

50 20 30 10 -5 7 15 35 57 65 55 -1

6. Vẽ hình cây AVL được tạo ra từ phần nhập liệu ở câu 5.

7. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú (`//Ghi chú`) trong các hàm `InsertNode, BalanceLeft, BalanceRight, _tmain`.

8. Sinh viên cài đặt lại các hàm dùng cho cây nhị phân và cây NPTK để áp dụng cho cây AVL.

Áp dụng – Nâng cao

1. Sinh viên tự cài đặt thêm chức năng cho phép người dùng nhập vào khóa x và kiểm tra xem khóa x có nằm trong cây AVL hay không.
   Cho dãy A như sau:
   1 3 5 7 9 12 15 17 21 23 25 27
   a. Tạo cây AVL từ dãy A. Cho biết số phép so sánh cần thực hiện để tìm phần tử 21 trên cây AVL vừa tạo.
   b. Tạo cây nhị phân tìm kiếm từ dãy A dùng lại đoạn code tạo cây của bài thực hành trước). Cho biết số phép so sánh cần thực hiện để tìm phần tử 21 trên cây nhị phân tìm kiếm vừa tạo.
   c. So sánh 2 kết quả trên và rút ra nhận xét?
2. Cài đặt chương trình đọc các số nguyên từ tập tin input.txt (không biết trước số lượng số nguyên trên tập tin) và tạo cây AVL từ dữ liệu đọc được
3. Cài đặt cây cân bằng AVL trong đó mỗi node trên cây lưu thông tin sinh viên.
4. Tự tìm hiểu và cài đặt chức năng xóa một node ra khỏi cây AVL.

BÀI TẬP THÊM

1. Viết chương trình cho phép tạo, tra cứu và sửa chữa từ điển Anh-Việt (sinh viên liên hệ với GVLT để chép file từ điển Anh-Việt)
2. Cài đặt lại các bài tập thêm của cây NPTK bằng cách dùng cây AVL