PHẦN 1: CÂY NHỊ PHÂN TÌM KIẾM BST

MỤC TIÊU

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

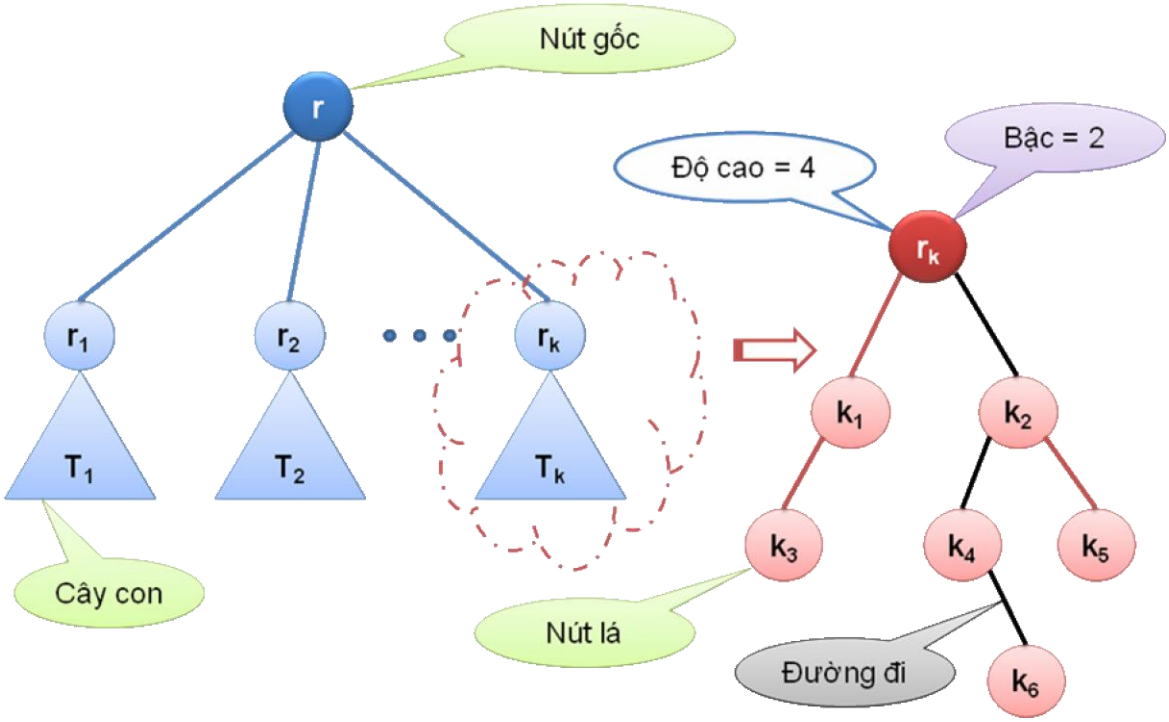
- Hiểu được các thành phần của cây nhị phân tìm kiếm.
- Thành thạo các thao tác trên cây nhị phân tìm kiếm: tạo cây, thêm phần tử, xóa phần tử, duyệt cây nhị phân tìm kiếm.
- Áp dụng cấu trúc dữ liệu cây nhị phân tìm kiếm vào việc giải quyết một số bài toán đơn giản.

Thời gian thực hành: **từ 120 phút đến 400 phút**

TÓM TẮT

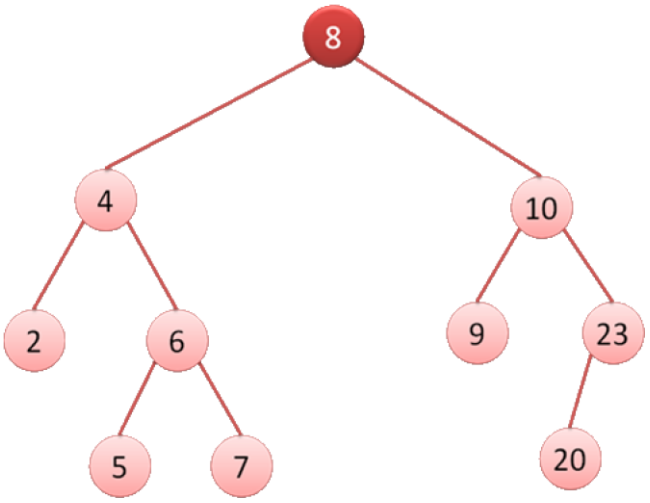
Cây nhị phân tìm kiếm là cây có tối đa 2 nhánh (cây con), nhánh trái và nhánh phải. Cây nhị phân tìm kiếm có các tính chất sau:

- Khóa của tất cả các nút thuộc cây con trái nhỏ hơn khóa nút gốc.
- Khóa của nút gốc nhỏ hơn khóa của tất cả các nút thuộc cây con phải.
- Cây con trái và cây con phải của nút gốc cũng là cây nhị phân tìm kiếm Một số khái niệm:



- Nút lá có độ cao bằng 1

Ví dụ cây nhị phân tìm kiếm:



Trong mỗi nút của cây nhị phân tìm kiếm, thông tin liên kết là vô cùng quan trọng. Chỉ cần một xử lý không cẩn thận có thể làm mất phần liên kết này thì cây sẽ bị '**gãy**' cây con liên quan ứng với liên kết đó (không thể truy xuất tiếp tất cả các nút của nhánh con bị mất).

Các thao tác cơ bản trên cây nhị phân tìm kiếm:

- Thêm 1 nút: dựa vào tính chất của cây nhị phân tìm kiếm để tìm vị trí thêm nút mới.
    - o Tạo cây: từ cây rỗng, lần lượt thêm các nút vào cây bằng phương thức thêm nút vào cây nhị phân tìm kiếm
- Xóa 1 nút: là nút lá, là nút có 1 nhánh con, là nút có 2 nhánh con.
- Duyệt cây nhị phân tìm kiếm: để có thể đi được hết các phần tử trên cây nhị phân tìm kiếm: duyệt trước (NLR), duyệt giữa (LNR), duyệt sau (LRN). Do tính chất của cây nhị phân tìm kiếm, phép duyệt giữa cho phép duyệt các khóa của cây theo thứ tự tăng dần

NỘI DUNG THỰC HÀNH

Cơ bản

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

Tổ chức một cây nhị phân tìm kiếm trong đó mỗi phần tử chứa thông tin dữ liệu là số nguyên.

Phân tích

- Cây nhị phân tìm kiếm có mỗi nút chứa dữ liệu nguyên. Thông tin của mỗi nút được khai báo theo ngôn ngữ C/C++ như sau:

```
#ifndef NODE_H
#define NODE_H


class Node
{
   public:
      Node();
      virtual ~Node();
```

```
        Node *Getleft() { return left; }
        void Setleft(Node val) { left = val; }
        Node *Getright() { return right; }
        void Setright(Node val) { right = val; }
        Node *Getparent() { return parent; }
        void Setparent(Node val) { parent = val; }
        int Getkey() { return key; }
        void Setkey(int val) { key = val; }

    protected:

    private:
        Node *left;
        Node *right;
        Node *parent;
        int key;
};

#endif // NODE_H
```

- Thao tác cần thực hiện:
  - Khai báo, khởi tạo cây
  - (lập) **thêm** nút có khóa nguyên vào **cây nhị phân tìm kiếm** (**Insert**),
  - **in** các nút của cây nhị phân tìm kiếm (**NLR**),
  - **tìm** 1 giá trị, nếu có: tính độ **cao** của nút đó (**Height**)
  - **xóa** nút khỏi cây (**RemoveNode**)
  - **in** các nút của cây sau khi xóa (**NLR**)

Chương trình mẫu

```
#ifndef NODE_H
#define NODE_H


class Node
{
   public:
      Node();
      Node(int);
      virtual ~Node();

      Node *Getleft() { return left; }
      void Setleft(Node *val) { left = val; }
      Node *Getright() { return right; }
```

```
        void Setright(Node *val) { right = val; }
        Node *Getparent() { return parent; }
        void Setparent(Node *val) { parent = val; }
        int Getkey() { return key; }
        void Setkey(int val) { key = val; }

    protected:

    private:
        Node *left;
        Node *right;
        Node *parent;
        int key;
};

#endif // NODE_H

//////////////////////////////////////////////////////////////////////
#include "Node.h"

Node::Node()
{

    //ctor
    this->key=0;
    this->left=nullptr;
    this->right=nullptr;
    this->parent=nullptr;
}
Node::Node(int k){
     //ctor
    this->key=k;
    this->left=nullptr;
    this->right=nullptr;
    this->parent=nullptr;
}
Node::~Node()
{
    //dtor
}

//////////////////////////////////////////////////////////////////////
#ifndef BST_H
#define BST_H
#include <Node.h>

class BST
```

```
{
   public:
      BST();
      virtual ~BST();

      Node* Getroot() { return root; }
      void Setroot(Node* val) { root = val; }
      bool InsertNode(Node*);
      bool InsertNodeRe(Node*,Node*);
      void deleteNode(Node*);
      void TravelNLR();
      void TravelLNR();
      void TravelLRN();
      void NLR(Node*);
      void LNR(Node*);
      void LRN(Node*);
      Node* search_x(int);

   protected:

   private:
      Node* root;
};

#endif // BST_H
//////////////////////////////////////////////////////////////////////////
#include "BST.h"
#include <iostream>

using namespace std;
BST::BST()
{
   //ctor
   this->root=nullptr;
}

BST::~BST()
{
   //dtor
}
bool BST::InsertNode(Node* n){
   Node *p=this->root;
   Node *T;
   if(root==nullptr)
   {
      this->root=n;
      return true;
```

```
    }
    while(p!=nullptr){
        T=p;

        if(p->Getkey()>n->Getkey())
            p=p->Getleft();
    else
        if(p->Getkey()<n->Getkey())
            p=p->Getright();
    else
        if(p->Getkey()==n->Getkey())
            return false;
    }

    if(T->Getkey()>n->Getkey())
        T->Setleft(n);

    else T->Setright(n);

     n->Setparent(T);
    return true;

}
bool BST::InsertNodeRe(Node* root,Node*p){
    if(root==nullptr){
        root=p;
        return true;
    }
    if(root->Getkey()==p->Getkey())
            return false;
    else if(root->Getkey()>p->Getkey())
            return InsertNodeRe(root->Getleft(),p);
            else return InsertNodeRe(root->Getright(),p);
}

void BST::NLR(Node*r){
    if(r!=nullptr){
        cout<<r->Getkey()<<"\n";
        NLR(r->Getleft());
        NLR(r->Getright());
    }
}
void BST::LNR(Node*r){
     //sinh vien code
}
void BST::LRN(Node*r){
    //sinh vien code
```

```
}
void BST::TravelNLR(){
   NLR(this->root);
}
void BST::TravelLNR(){
       //sinh vien code
}
void BST::TravelLRN(){
   //sinh vien code
}
Node* BST::search_x(int k){
   //sinh vien code
}
void BST::deleteNode(Node* n){
   Node* p=n;
   if(p->Getleft()==nullptr&&n->Getright()==nullptr){
      // sinh vien code
       delete n;
}
   else{
      if(p->Getright()!=nullptr){
        p=p->Getright();
         while(p->Getleft()!=nullptr)//
            p=p->Getleft();
         n->Setkey(p->Getkey());
   //sinh vien code


      }else{
           p=p->Getleft();
         while(p->Getright()!=nullptr)//
             p=p->Getright();

         n->Setkey(p->Getkey());
   //sinh vien code


      }
   }
}
////////////////////////////////////////////////////////////////////////
#include <iostream>
#include <BST.h>
using namespace std;

int main()
{
```

```
    BST *tree=new BST();
    Node *n;
    n=new Node(10);
    tree->InsertNode(n);
    n=new Node(19);
    tree->InsertNode(n);
     n=new Node(9);
     tree->InsertNode(n);
      n=new Node(3);
     tree->InsertNode(n);
      n=new Node(19);
     tree->InsertNode(n);
      n=new Node(8);
     tree->InsertNode(n);
      n=new Node(4);
     tree->InsertNode(n);
      n=new Node(1);
     tree->InsertNode(n);
      n=new Node(15);
     tree->InsertNode(n);

    tree->TravelNLR();
    return 0;
}
```

Yêu cầu

1. Biên dịch đoạn chương trình nêu trên.
2. Vẽ hình cây nhị phân tìm kiếm theo dữ liệu được câu 1.
3. Thực hiện hoàn thiện các hàm: có chú thích //sinh vien code

Áp dụng – Nâng cao

1. Bổ sung chương trình mẫu cho phép tính **tổng giá trị** các nút trên cây nhị phân gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **NLR** để viết hàm **SumTree**.

2. Bổ sung chương trình mẫu cho phép tìm **giá trị nguyên lớn nhất và nhỏ nhất** trong số các phần tử nguyên trên cây nhị phân tìm kiếm gồm các giá trị nguyên. Gợi ý: dựa vào tính chất 1, 2 của cây nhị phân tìm kiếm.
3. Bổ sung chương trình mẫu cho phép tính **số lượng các nút** của cây nhị phân gồm các giá trị nguyên.

Gợi ý: tham khảo hàm **NLR** để viết hàm **CountNode**.

4. Bổ sung chương trình mẫu cho biết **số lượng các nút lá** trên cây nhị phân.

Gợi ý: tham khảo thao tác duyệt cây nhị phân **NLR**.

5. Sử dụng cây nhị phân tìm kiếm để giải bài toán:

   a. Đếm có bao nhiêu giá trị phân biệt trong dãy số cho trước

   b. Với mỗi giá trị phân biệt, cho biết số lượng phần tử

BÀI TẬP ỨNG DỤNG

1. Sử dụng cây nhị phân tìm kiếm để giải bài toán đếm (thống kê) số lượng ký tự có trong văn bản (Không dấu).
   a. Xây dựng cây cho biết mỗi ký tự có trong văn bản xuất hiện mấy lần
   b. Nhập vào 1 ký tự. Kiểm tra ký tự đó xuất hiện bao nhiêu lần trong văn bản
2. Bài toán tương tự như trên nhưng thống kê số lượng tiếng có trong văn bản (không dấu)

   Ví dụ:

   Văn bản có nội dung như sau: “hoc sinh di hoc mon sinh hoc”
   Kết quả cho thấy như sau:

   di: 1

   hoc: 3

   mon: 1

   sinh: 2

PHẦN 2: CÂY NHỊ PHÂN 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.

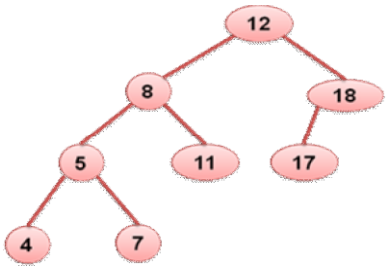
Thời gian thực hành: 120 phút – 360 phút

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

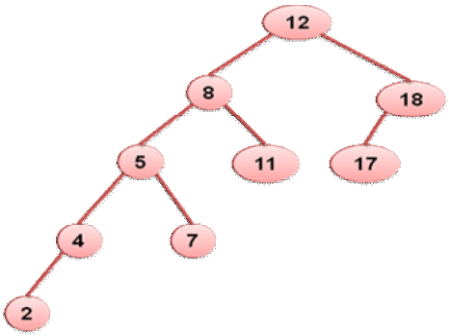
TÓM TẮT

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

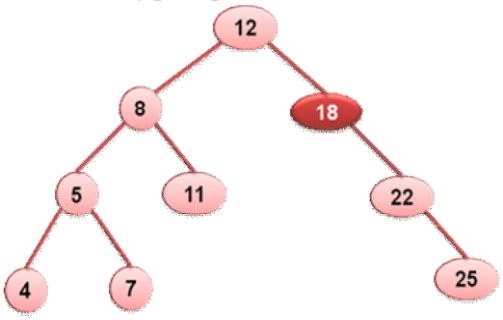


Ví dụ 2: cây không cân bằng

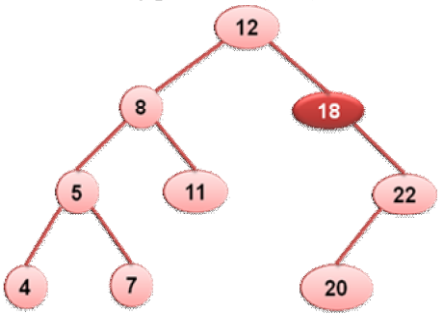


Khi thêm node mới vào cây AVL có thể xảy ra các trường hợp mất cân bằng như sau:

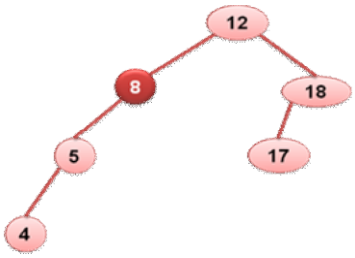
Mất cân bằng phải-phải (R-R)



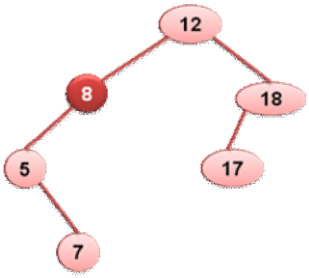
Mất cân bằng phải-trái (R-L)



Mất cân bằng trái-trái (L-L)

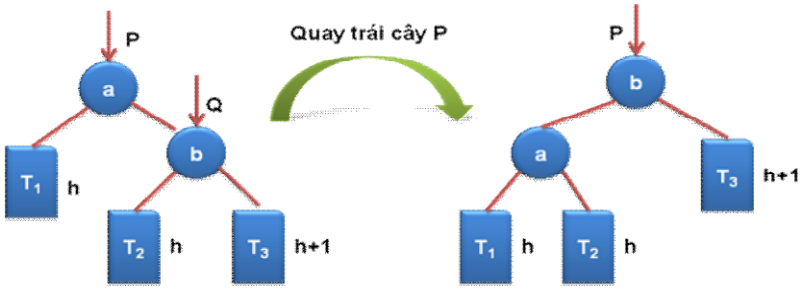


Mất cân bằng trái-phải (L-R)

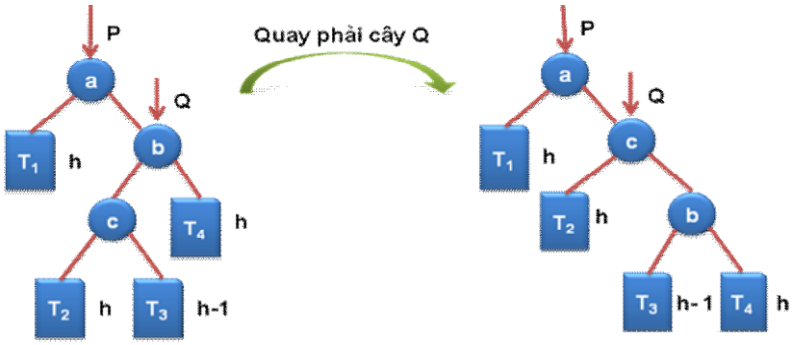


Xử lý mất cân bằng bằng cách sử dụng các phép quay cây

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.

Sau đó, xuất thông tin các node trên cây.

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:

```
#pragma once
#ifndef NODE_H
#define NODE_H



class Node
{
public:
    Node();
    Node(int);
    virtual ~Node();

    Node* Getleft() { return left; }
    void Setleft(Node* val) { left = val; }
    Node* Getright() { return right; }
    void Setright(Node* val) { right = val; }
    Node* Getparent() { return parent; }
    void Setparent(Node* val) { parent = val; }
    int Getkey() { return key; }
    void Setkey(int val) { key = val; }
    int Getheight() { return height; }
    void Setheight(int val) { height = val; }



protected:


private:
```

```
    Node* left;
    Node* right;
    Node* parent;
    // int bal; // -1 0 1
    int key;
    int height;
};

#endif // NODE_H



#include "Node.h"

Node::Node()
{

    //ctor
    this->key = 0;
    this->left = nullptr;
    this->right = nullptr;
    this->parent = nullptr;
    this->height = 0;
}
Node::Node(int k) {
    //ctor
    this->key = k;
    this->left = nullptr;
    this->right = nullptr;
    this->parent = nullptr;
    this->height = 0;
}
Node::~Node()
{
    //dtor
}
```

```
#pragma once
#ifndef AVL_TREE_H
#define AVL_TREE_H
#include "Node.h"

class AVL_tree
{
public:
    AVL_tree();
    virtual ~AVL_tree();
    Node* Getroot() { return root; }
    void Setroot(Node* val) { root = val; }
    bool InsertNode(Node*);
    Node* InsertNode(Node*, Node*);
    void InsertNodeRe(Node*);
    void deleteNode(Node*);
    void TravelNLR();
    void TravelLNR();
    void TravelLRN();
    void NLR(Node*);
    void LNR(Node*);
    void LRN(Node*);
    void LeftRotate(Node*&);
    void RightRotate(Node*&);
    int CheckBal(Node*);
    int GetHeight(Node*);
    Node* search_x(int);

protected:

private:
    Node* root;

    int nNum; // Node number of tree
```

```
    int height; //height of tree

};

#endif // AVL_TREE_H



#include "AVL_tree.h"
#include <iostream>
#include "Node.h"
using namespace std;
AVL_tree::AVL_tree()
{
    //ctor
    this->root = nullptr;
}

AVL_tree::~AVL_tree()
{
    //dtor
}
bool AVL_tree::InsertNode(Node* n) {
    Node* p = this->root;
    Node* T=nullptr;
    if (root == nullptr)
    {
        this->root = n;

        return true;
    }
    while (p != nullptr) {
        T = p;

        if (p->Getkey() > n->Getkey())
            p = p->Getleft();
```

```
    else
        if (p->Getkey() < n->Getkey())
            p = p->Getright();
        else
            if (p->Getkey() == n->Getkey())
                return false;
}

if (T->Getkey() > n->Getkey())
    T->Setleft(n);
else T->Setright(n);
n->Setparent(T);
////////////////////////////////////////
Node* x = n;
Node* parentX = x->Getparent();
while (x!= nullptr) {
    int bal = this->CheckBal(x);
    switch (bal) {
    case 0:    break;     // Can bang
    case 1:    break;     //
    case -1:   break;   //

    case 2:if (Sinh vien code)// LEFT-LEFT
    {
        parentX = x->Getparent();
        this->RightRotate(x);
        x->Setparent(parentX);
        if (parentX != nullptr)
        if (x->Getkey() > parentX->Getkey())
            parentX->Setright(x);
        else parentX->Setleft(x);
    }
         else // Left-Right
    {
```

```
    }
        break;     //
    case -2: if (x->Getright()->Getright() != nullptr)  // RIGHT-RIGHT
       {
       parentX = x->Getparent();
       this->LeftRotate(x);
       x->Setparent(parentX);
       if(parentX!=nullptr)
        if (x->Getkey() > parentX->Getkey())
          parentX->Setright(x);
        else parentX->Setleft(x);
       }
       else // Right-left
          {


          }
        break;   //
    }
    if (x->Getparent() == nullptr)
       this->root = x;
    x = x->Getparent();

  }

  return true;

}
Node* AVL_tree::InsertNode(Node* r, Node* p) {
  if (r == nullptr) {
    r = p;
    return r;
  }
  if (r->Getkey() == p->Getkey())
    return nullptr;
  else if (r->Getkey() > p->Getkey()) {
```

```
        r->Setleft(InsertNode(r->Getleft(), p));
        return r->Getleft();
    }
    else {
        r->Setright(InsertNode(r->Getright(), p));
        return r->Getright();
    };
    //
 /*   r->Setheight ( 1 + max(r->Getleft()->Getheight(),r->Getright()->Getheight()));

    int valBalance = r->Getleft()->Getheight() - r->Getright()->Getheight();

    if(valBalance>1&&r->Getleft()->Getkey()>p->Getkey())
        this->RightRotate(r);*/

}



void AVL_tree::InsertNodeRe(Node* p) {
    this->root = InsertNode(this->root, p);
}
void AVL_tree::NLR(Node* r) {
    if (r != nullptr) {
        cout << r->Getkey() << "\n";
        NLR(r->Getleft());
        NLR(r->Getright());
    }
}
void AVL_tree::LNR(Node* r) {
    //sinh vien code
}
void AVL_tree::LRN(Node* r) {
    //sinh vien code
}
void AVL_tree::TravelNLR() {
```

```
    NLR(this->root);
}
void AVL_tree::TravelLNR() {
    //sinh vien code
}
void AVL_tree::TravelLRN() {
    //sinh vien code
}
Node* AVL_tree::search_x(int k) {
    //sinh vien code
    return nullptr;
}
void AVL_tree::deleteNode(Node* n) {
    Node* p = n;
    if (n->Getleft() == nullptr && n->Getright() == nullptr)
        delete n;
    else {
        if (p->Getright() != nullptr) {
            while (p->Getleft() != nullptr)//
                p = p->Getleft();
            n->Setkey(p->Getkey());
            // Code
            delete p;
        }
        else {
            while (p->Getright() != nullptr)//
                p = p->Getright();

            n->Setkey(p->Getkey());
            //code
            delete p;
        }
    }
}
void AVL_tree::LeftRotate(Node*& P) {
```

```
    Node* Q;
    Node* T;
    Q = P->Getright();
    T = Q->Getleft();
    P->Setright(T);
    Q->Setleft(P);
    P->Setparent(Q);
    if(T!=nullptr)
    T->Setparent(P);
    P = Q;
}
void AVL_tree::RightRotate(Node*& P) {
    // sinh vien code
}
int AVL_tree::CheckBal(Node* p) {
    int bal = this->GetHeight(p->Getleft()) - this->GetHeight(p->Getright());
    return bal;
}
int AVL_tree::GetHeight(Node* p) {
    if (p == nullptr) return 0;
    else
    return 1 + max(GetHeight(p->Getleft()), GetHeight(p->Getright()));
}
```

- 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**)

Yêu cầu

1. Xây dựng cấu trúc cây AVL
2. Xây dựng cây AVL, khi người dùng nhập vào các dữ liệu sau:

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

3. Vẽ hình cây AVL được tạo ra từ phần nhập liệu ở câu 2.
4. 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,.

5. 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.