**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.

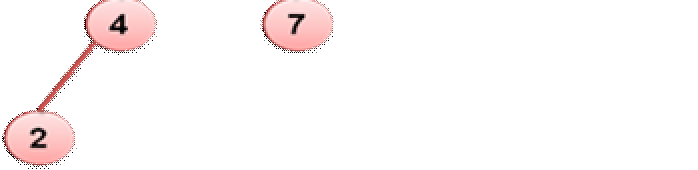
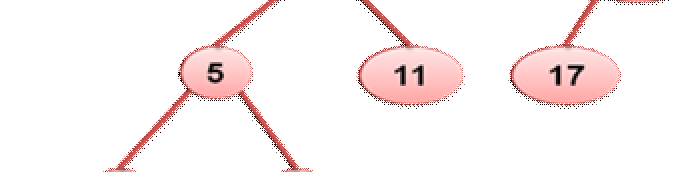
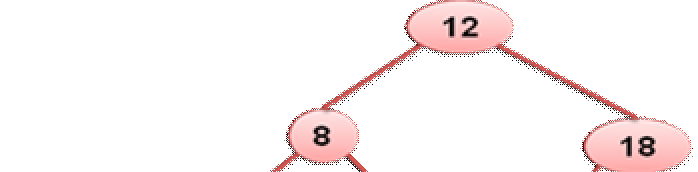
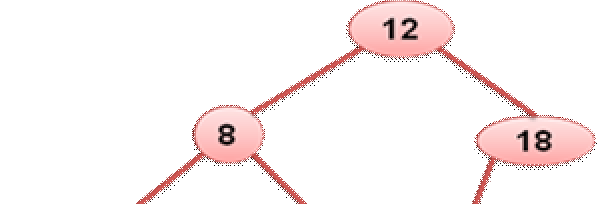
Thời gian thực hành: 120 phút – 360 phút

***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

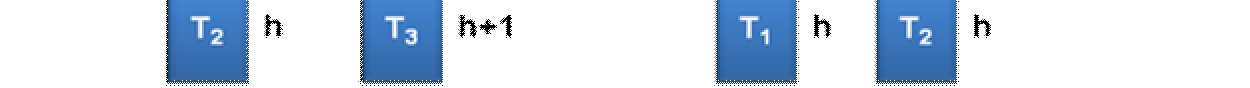
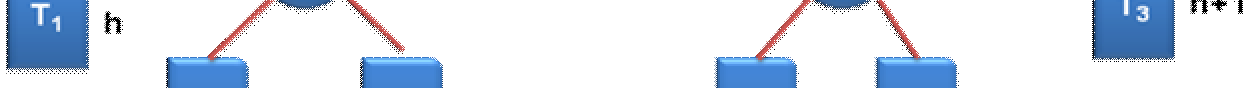
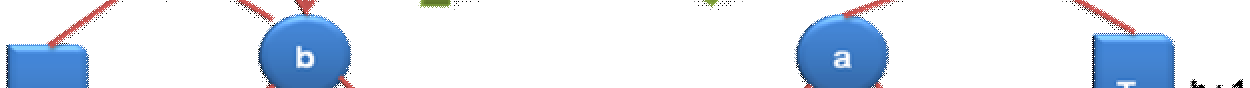
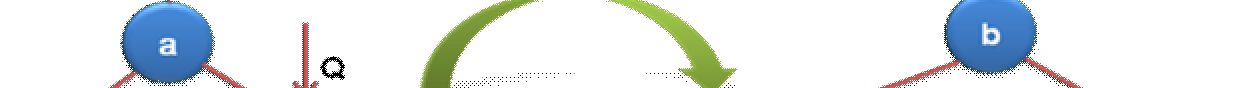


Khi thêm node mới vào cây AVL có thể xảy ra các trường hợp mất cân bằng như sau:

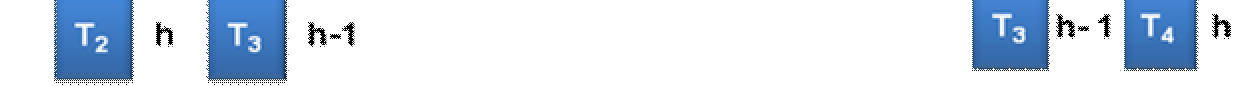
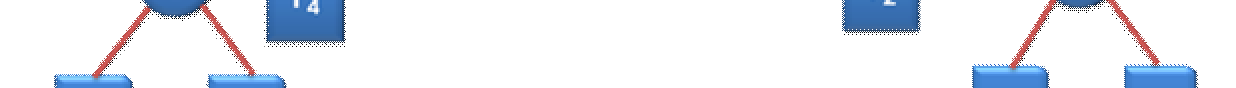
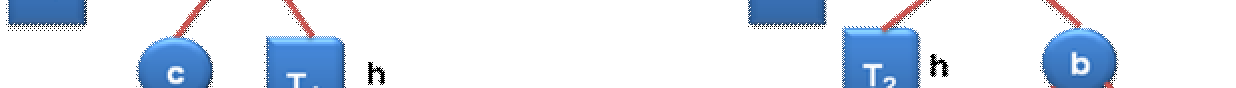
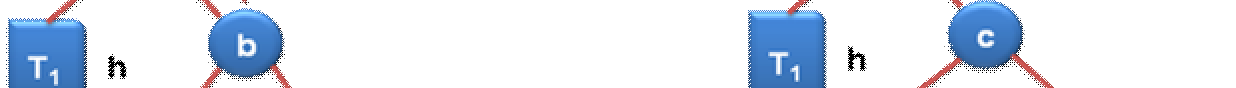
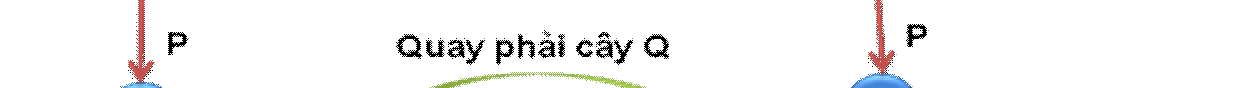
|  |  |
| --- | --- |
| Mất cân bằng phải-phải (R-R) | Mất cân bằng phải-trái (R-L) |
| Mất cân bằng trái-trái (L-L) | Mất cân bằng trái-phải (L-R) |

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

1. Quay trái



1. 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:

|  |
| --- |
| //Node.h  #pragma once  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;  };  //Node.cpp  #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  }  //AVL\_tree.h  #pragma once  #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);  int UpdateHeight(Node\*);  protected:  private:  Node\* root;  int nNum; // Node number of tree  int height; //height of tree  };  //AVL\_tree.cpp  //AVL\_tree.cpp  #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;  while (x != nullptr) {  Node\* left = x->Getleft();  Node\* right = x->Getright();  int bal = GetHeight(left) - GetHeight(right);  if (abs(bal) < 2 ){    }  else if (bal == 2) {  if (x->Getleft()->Getleft() != nullptr)// LEFT-LEFT  {  this->RightRotate(x);  /\*parentX = x->Getparent();  x->Setparent(parentX);  if (parentX != nullptr)  if (x->Getkey() > parentX->Getkey())  parentX->Setright(x);  else parentX->Setleft(x);\*/  }  else // Left-Right  {  Node\* xleft = x->Getleft();  this->LeftRotate(xleft);  this->RightRotate(x);  }  }  else {  if (x->Getright()->Getright() != nullptr) // RIGHT-RIGHT  {  this->LeftRotate(x);  /\*parentX = x->Getparent();  x->Setparent(parentX);  if (parentX != nullptr)  if (x->Getkey() > parentX->Getkey())  parentX->Setright(x);  else parentX->Setleft(x);\*/  }  else // Right-left  {  Node\* xright = x->Getright();  this->RightRotate(xright);  this->LeftRotate(x);  }  }  if (x == nullptr) 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) {  if (r != nullptr) {  LNR(r->Getleft());  cout << r->Getkey() << " ";  LNR(r->Getright());  }  //sinh vien code  }  void AVL\_tree::LRN(Node\* r) {  if (r != nullptr) {  LRN(r->Getleft());  LRN(r->Getright());  cout << r->Getkey() << "\n";  }  //sinh vien code  }  void AVL\_tree::TravelNLR() {  NLR(this->root);  }  void AVL\_tree::TravelLNR() {  LNR(this->root);  }  void AVL\_tree::TravelLRN() {  LRN(this->root);  }  Node\* AVL\_tree::search\_x(int k) {  //sinh vien code  Node\* r = this->root;  while (r->Getkey() != k || r != nullptr) {  if (k > r->Getkey()) {  r = r->Getright();  }  else {  r = r->Getleft();  }  }  return r;  }  void AVL\_tree::deleteNode(Node\* n) {  Node\* p = n;  if (p->Getleft() == nullptr && n->Getright() == nullptr) {  if (p->Getparent()->Getleft() == n) {  p->Getparent()->Setleft(nullptr);  }  else {  p->Getparent()->Setright(nullptr);  }  delete n;  }  else {  if (p->Getright() != nullptr) {  p = p->Getright();  while (p->Getleft() != nullptr)//  p = p->Getleft();  n->Setkey(p->Getkey());  if (p->Getright() != nullptr) p->Getparent()->Setleft(p->Getright());  delete p;  }  else {  p = p->Getleft();  while (p->Getright() != nullptr)//  p = p->Getright();  n->Setkey(p->Getkey());  if (p->Getleft() != nullptr) p->Getparent()->Setright(p->Getleft());  delete p;  }  }  }  void AVL\_tree::LeftRotate(Node\*& P) {  Node\* Q;  Node\* T;  Q = P->Getright();  T = Q->Getleft();  P->Setright(T);  Q->Setleft(P);  if (T != nullptr)  T->Setparent(P);  Q->Setparent(P->Getparent());  if (P->Getparent() != nullptr)  P->Getparent()->Setright(Q);  P->Setparent(Q);  }  void AVL\_tree::RightRotate(Node\*& P) {  Node\* Q;  Node\* T;  Q = P->Getleft();  T = Q->Getright();  P->Setleft(T);  Q->Setright(P);  if (T != nullptr)  T->Setparent(P);  Q->Setparent(P->Getparent());  if (P->Getparent() != nullptr)  P->Getparent()->Setleft(Q);  P->Setparent(Q);    // sinh vien code  }  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

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

1. 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

* 1. 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.
  2. 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.
  3. So sánh 2 kết quả trên và rút ra nhận xét?

1. 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
2. Cài đặt cây cân bằng AVL trong đó mỗi node trên cây lưu thông tin sinh viên.
3. 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.
2. Cài đặt lại các bài tập thêm của cây NPTK bằng cách dùng cây AVL