CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 4 – CÂY TÌM KIẾM NHỊ PHÂN CÂN BẰNG AVL (4 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững các khái niệm về cây cân bằng, cây cân bằng hoàn toàn.
- Cài đặt được kiểu dữ liệu cây tìm kiếm nhị phân cân bằng (AVL) và các thao tác, phép toán trên cây AVL.
- Vận dụng kiến thức đã học để giải một số bài toán thực tế.

II. Yêu cầu

- Sinh viên phải hoàn thành tối thiểu ***2 bài tập 1 và 2*** (trong phần V). Mỗi bài tập tạo một project, xóa các thư mục debug của project này. Sau đó chép cả 2 project vào thư mục: Lab4_CTK39_HoTen_MSSV_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab4_CTK39_HoTen_MSSV_Nhom#.rar. Ví dụ: Lab4_CTK39_NguyenVanA_1012345_Lab2.rar.
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. *Cây cân bằng và chỉ số cân bằng*

Cây tìm kiếm nhị phân cân bằng là cây tìm kiếm nhị phân mà tại mỗi nút của nó, độ cao của cây con trái và độ cao của cây con phải chênh lệch nhau không quá 1.

Chỉ số cân bằng (Balance factor) của một nút p là hiệu của chiều cao cây con phải và chiều cao cây con trái của nó. Nghĩa là:

BalFactor(p) = height(p.RightChild) – height(p.LeftChild)

Việc thêm hay hủy một nút trên cây AVL có thể làm cây tăng hay giảm chiều cao, khi đó, ta cần phải cân bằng lại cây. Để giảm tối đa chi phí cân bằng lại cây, ta chỉ cân bằng lại ở phạm vi cục bộ. Vì thế, với mỗi nút của cây, ngoài các thuộc tính thông thường như cây nhị phân, ta cần lưu trữ thêm thông tin về chỉ số cân bằng.

Để đơn giản, ta quy ước: EH = 0, RH = 1, LH = -1

BalFactor(p) = EH ⇔ 2 cây con của p cao bằng nhau

BalFactor(p) = RH ⇔ cây lệch phải hay cây con phải của p cao hơn cây con trái

BalFactor(p) = LH ⇔ cây lệch trái hay cây con trái của p cao hơn cây con phải

2. *Các trường hợp mất cân bằng*

Xét trường hợp chèn thêm một nút mới **v** vào một nút **u** trên cây cân bằng T. Do **v** chỉ có thể được chèn vào đúng một trong hai cây con của **u** nên nhiều nhất là **v** có thể làm tăng chiều cao của một trong hai cây con đó. Nếu v không làm tăng chiều cao của cây con nào hoặc v làm tăng chiều cao của một cây con nhưng trước đó cây con này có chiều cao nhỏ hơn hoặc bằng chiều cao của cây con kia thì tính cân bằng AVL tại đỉnh **u** vẫn giữ nguyên.

Tính cân bằng AVL tại **u** chỉ có thể bị phá vỡ khi **v** làm tăng chiều cao của cây con có chiều cao lớn hơn trong hai cây con của **u**.Cũng như vậy, nếu **v** không làm tăng chiều cao của chính **u** thì **v** không làm thay đổi hệ số cân bằng tại các đỉnh tiền bối của **u**.

Mặt khác, nút **v** luôn được thêm vào với tư cách là con của một nút trước đó là lá hoặc nửa lá. Nếu cha của **v** trước khi thêm **v** là nửa lá thì chiều cao của cây con gốc cha của **v** không thay đổi sau khi thêm **v** còn hệ số cân bằng tại đỉnh cha này bằng 0. Khi đó tất cả các nút tiền bối của cha của v không thay đổi hệ số cân bằng. Tính cân bằng AVL được giữ vững trên toàn bộ cây T.

Nếu đỉnh cha của **v** trước khi chèn **v** là lá, goi **u** là đỉnh tiền bối của **v** có mức cao nhất mà tính cân bằng AVL bị phá vỡ. Như vậy bốn trường hợp sau có thể phá vỡ tính cân bằng AVL tại **u**

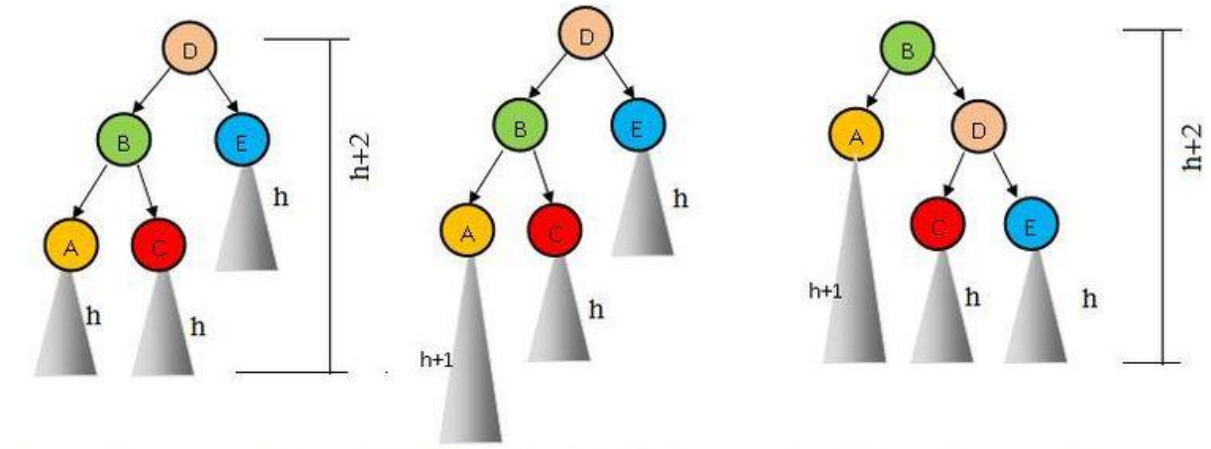
a. Trước khi chèn cây con gốc **u** lệch trái và **v** làm tăng chiều cao của cây con trái.
   - Sau khi chèn cây con trái lệch trái (Case LL)
   - Sau khi chèn cây con trái lệch phải (Case LR)

b. Trước khi chèn cây con gốc **u** lệch phải và **v** làm tăng chiều cao của cây con phải.
   - Sau khi chèn cây con phải lệch phải (Case RR)
   - Sau khi chèn cây con phải lệch trái. (Case RL)

Xét tương tự cho trường hợp xóa một nút khỏi cây. Lưu ý, việc cân bằng lại có thể xảy ra theo phản ứng dây chuyền.

3. *Cân bằng lại cây*

Trường hợp cây con trái lệch trái: Sử dụng phép quay đơn Left-Left (RotateLL)

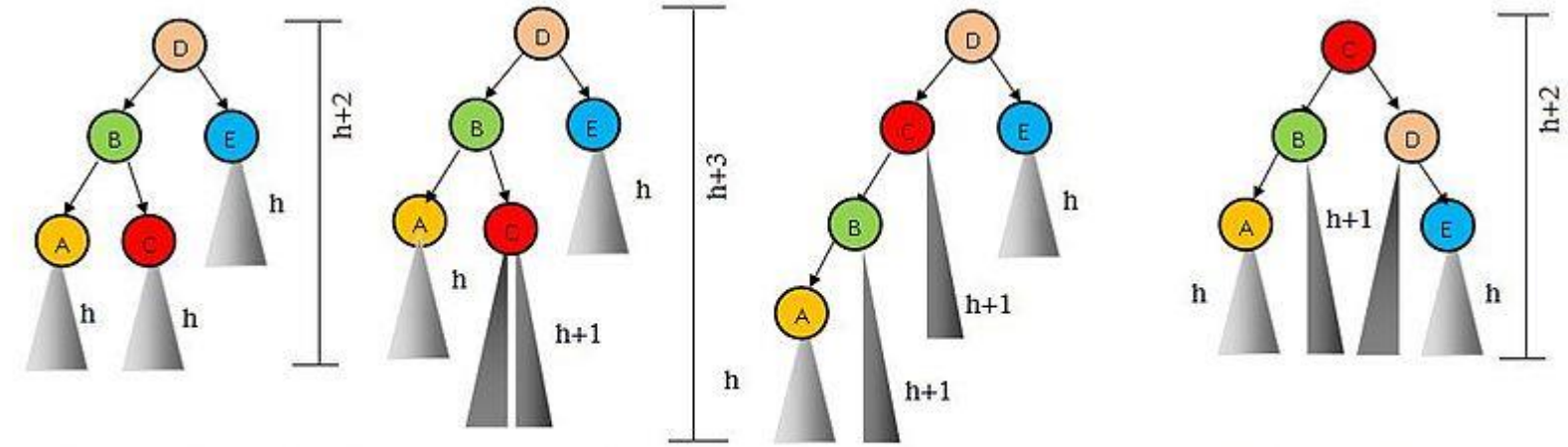


1. Cây lệch trái trước khi chèn . Chiều cao cây là h+2,

2. Chèn một phần tử vào cây con trái của cây con trái làm cho cây mất cân bằng

3. Sau phép quay phải cây trở thanh cân bằng, chiều cao cây vẫn là h+2

Trường hợp cây con trái lệch phải: Sử dụng phép quay Left-Right (RotateLR)



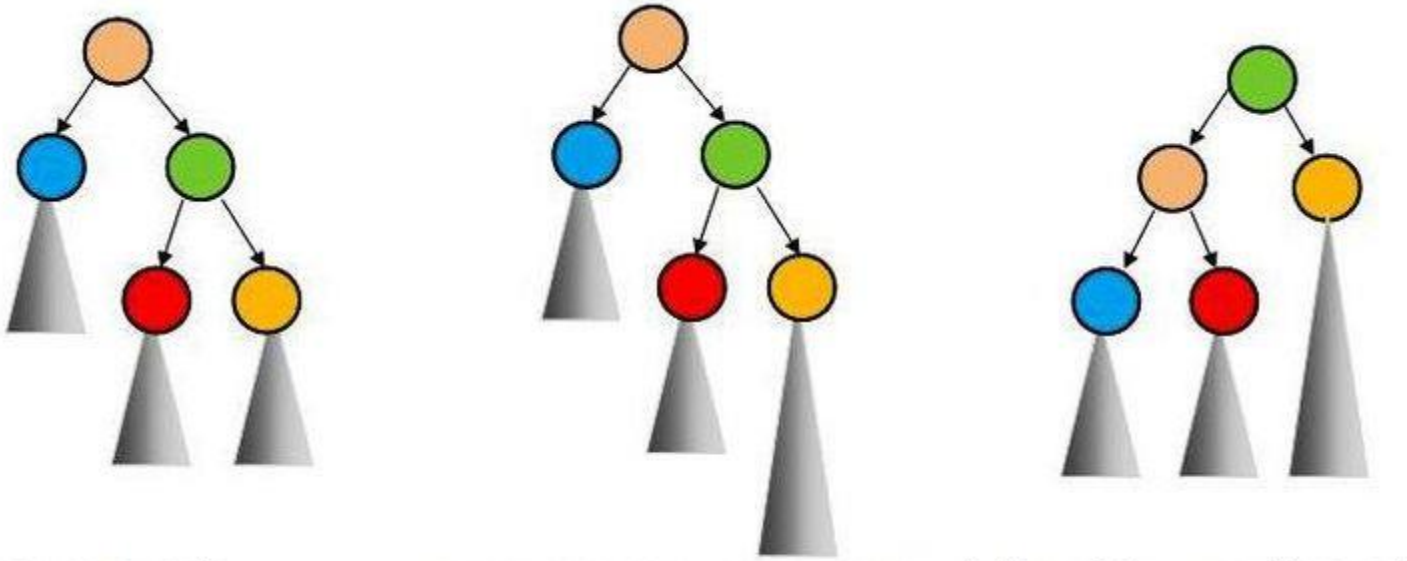
1. Cây ban đầu lệch trái, cây con trái cân bằng. Chiều cao cây là h+2

2. Phép chèn vào cây con trái làm cho cây con trái lệch phải.

3. Phép xoay trái cây con trái đưa cây con trái thành cây lệch trái

4.Sau phép xoay phải cây, cây trở thành cân bằng AVL. Chiều cao vẫn là h+2

Trường hợp cây con phải lệch phải: Sử dụng phép quay Right-Right (RotateRR)

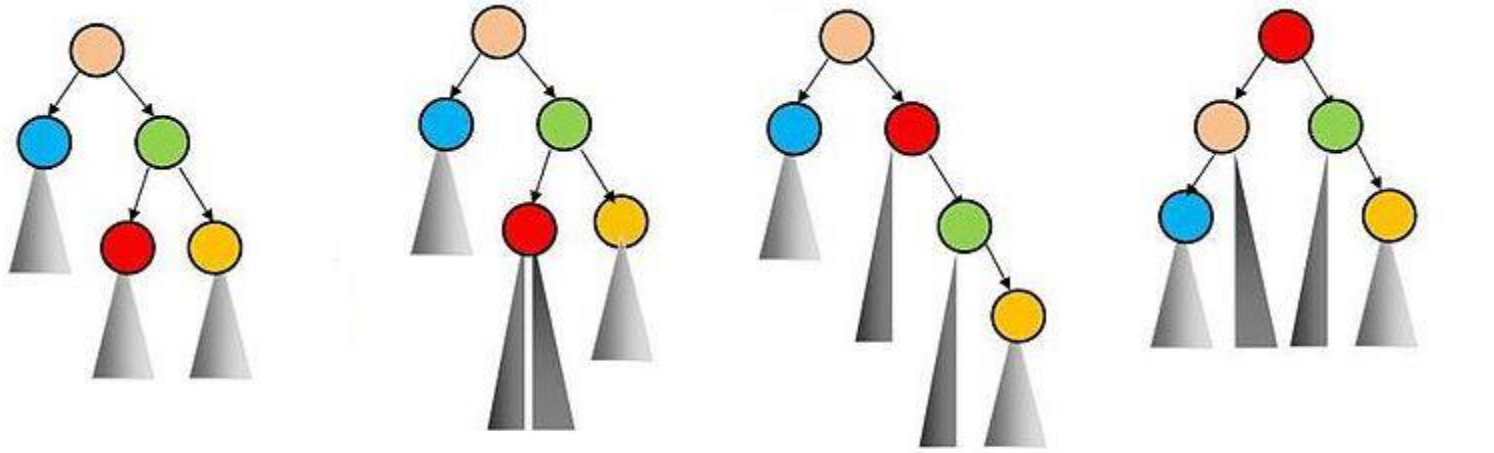


1. Cây lệch phải

2. Sau khi chèn, cây con phải lệch phải

3. Sau phép quay trái cây trở thnhà cân bằng AVL

Trường hợp cây lệch phải, cây con phải lệch trái: Sử dụng phép quay Right-Left (RotateRL)



1. Cây AVL lệch phải

2. Sau phpé chèn vào cay con phải, cây con phải lệch trái.

3. Sau phép quay phải cây con phải. Cây con phải trở thành lệch phải

4. Sau phép quay trái, cây trở thành cân bằng AVL.

IV. Hướng dẫn thực hành

1. Tạo dự án

Sinh viên có thể chọn cài đặt cây tìm kiếm nhị phân cân bằng (AVL) theo ngôn ngữ C# hoặc C++. Các phần sau minh họa cách cài đặt, hướng dẫn bằng mã giả trên cả ngôn ngữ C# (bên trái) và C++ (bên phải).

- Tạo dự án mới, đặt tên là Lab2_AVL_Tree
- Tạo ra hai lớp, đặt tên lần lượt là TreeNode và AVLTree

- Tạo dự án mới, đặt tên là Lab2_AVL_Tree
- Trong thư mục Header Files, tạo ra 4 files sau:
  - common.h: Định nghĩa các hằng số và kiểu dữ liệu AVLTree
  - avltree.h: Định nghĩa các thao tác trên cây AVL
  - stack.h: Định nghĩa kiểu dữ liệu Stack và các thao tác
  - queue.h: Định nghĩa kiểu dữ liệu Queue và các thao tác
- Trong thư mục Source Files, tạo tập tin: program.cpp

2. Định nghĩa kiểu dữ liệu cây tìm kiếm nhị phân (BST)

- Nhấp đôi chuột vào file TreeNode.cs, định nghĩa kiểu dữ liệu TreeNode (thể hiện một nút trên cây AVL) như sau:

```
// Kiểu dữ liệu thể hiện 1 nút của cây AVL
class AVLNode
{
```

- Nhấp đôi chuột vào file common.h, định nghĩa các hằng số và kiểu dữ liệu như sau:

```
#ifndef _DATATYPE_
#define _DATATYPE_
```

```
public int Key;   // Dữ liệu chứa trong nút
public int BalFactor;      // Chỉ số cân bằng
public AVLNode LeftChild;  // Nút con trái
public AVLNode RightChild; // Nút con phải

// Kiểm tra có phải là nút lá ko?
public bool IsLeaf
{
      get   {
            return !(HasLeft || HasRight);
      }
}
// Kiểm tra có nút con bên trái ko?
public bool HasLeft
{
      get { return LeftChild != null; }
}
// Kiểm tra có nút con bên phải ko?
public bool HasRight
{
      get { return RightChild != null; }
}
// Khởi tạo một nút với giá trị cho trước
public AVLNode(int data)
{
      this.Key = data;
      this.BalFactor = AVLTree.EH;
      this.LeftChild = null;
      this.RightChild = null;
}
public AVLNode(int data, AVLNode left,
                         AVLNode right)
{
      this.Key = data;
      this.BalFactor = AVLTree.EH;
      this.LeftChild = left;
      this.RightChild = right;
}
// Chuyển một nút sang chuỗi để hiển thị
public override string ToString()
{
      return string.Format("{0} - {1}",
                        Key, BalFactor);
}
```

- Nhấp đôi chuột vào file AVLTree.cs, định nghĩa kiểu dữ liệu cây tìm kiếm nhị phân cân bằng như sau:

```
#define     LH    -1    // Cây lệch trái
#define     EH    0     // Cây cân bằng
#define     RH    1     // Cây lệch phải
#define     BAL   2     // Kết quả cây cân bằng
#define     DEV   1     // Kết quả: lệch trái or phải
#define     FAIL  -1    // Thao tác thất bại
#define     ZERO  0

// Kiểu của dữ liệu lưu trong cây
typedef int DataType;

// Kiểu dữ liệu nút của cây BST
struct AVLNode
{
      DataType    Key;              // Dữ liệu
      int         BalFactor;  // Chỉ số cân bằng
      AVLNode*    LeftChild;        // Nút con trái
      AVLNode*    RightChild;       // Nút con phải
};
// Định nghĩa kiểu con trỏ
typedef AVLNode* NodePointer;

// Kiểu dữ liệu cây nhị phân
typedef NodePointer AVLTree;

// Kiểu dữ liệu phần tử của Queue, Stack
struct Entry
{
      // Dữ liệu chứa trong một nút của Queue
      NodePointer Data;       // hoặc Stack
      Entry*      Next;       // Con trỏ Next
};

// Định nghĩa kiểu con trỏ tới một Entry
typedef Entry*    EntryPtr;

// Tạo một nút của Queue hoặc Stack
EntryPtr CreateEntry(NodePointer node)
{
      EntryPtr item = new Entry;
      if (item)
      {
            item->Data = node;
            item->Next = NULL;
      }
      return item;
}
```

```
class AVLTree
{
	public static readonly int EH = 0;
	public static readonly int LH = 1;
	public static readonly int RH = -1;
	public static readonly int BAL = 2;
	public static readonly int DEV = 1;
	public static readonly int FAIL = -1;
	public static readonly int ZERO = 0;

	// Nút gốc của cây
	public AVLNode Root;

	// Kiểm tra có phải cây rỗng hay ko?
	public bool IsEmpty
	{
		get { return Root == null; }
	}

	// Khởi tạo một cây rỗng
	public AVLTree()
	{
		this.Root = null;
	}
}
```

```
#endif
```

- Trong tập tin program.cpp, nhập đoạn mã sau:

```
#include <iostream>
#include <conio.h>
using namespace std;

#include "common.h"
#include "queue.h"
#include "stack.h"
#include "avltree.h"

// Hàm in dữ liệu trong nút của cây
void PrintNode(NodePointer node)
{
cout << node->Key << "\t" << node->BalFactor << endl;
}

void main()
{
	getch();
}
```

Vào menu Build > chọn Build Solution để biên dịch chương trình, kiểm tra lỗi. Nếu có lỗi, kiểm tra lại mã nguồn bạn đã viết. Nếu chương trình không có lỗi, ta sẽ cài đặt các phép toán trên cây tìm kiếm nhị phân.

V. Bài tập thực hành

Bài 1. Cài đặt các thao tác trên cây tìm kiếm nhị phân cân bằng (AVL) – Viết hàm trong tập tin avltree.h

a. Tìm nút trên cây chứa giá trị key cho trước (dạng đệ quy)

```
C#: TreeNode Search(AVLTree root, int key, ref TreeNode parent)
C++: NodePointer Search(AVLTree root, DataType key, NodePointer &parent)
{
	Nếu (root = null) thì trả về null;		// Không tìm thấy
	Nếu (root.Key = key) thì trả về root;		// Tìm thấy
	Ngược lại
	{
		Gán parent = root;			// Cập nhật nút cha
```

```
            Nếu (root.Key > key) thì
                  Trả về kết quả tìm kiếm key trong cây con trái của root.
            Ngược lại
                  Trả về kết quả tìm kiếm key trong cây con phải của root.
      }
}
```

b. Cài đặt lại hàm tìm kiếm trên đây theo dạng lặp (khử đệ quy)

c. Duyệt cây theo thứ tự LNR (Inorder) dạng đệ quy

```
C#: void LNR(AVLTree root, Action<TreeNode> XuLy)
C++: void LNR(AVLTree root, void (XuLy)(NodePointer node))
{
      Nếu (root khác null) thì
      {
            Duyệt cây con bên trái root theo thứ tự LNR
            XuLy( root );                  // Gọi hàm xử lý nút root
            Duyệt cây con bên phải root theo thứ tự LNR
      }
}
```

d. Cài đặt các hàm duyệt cây theo thứ tự LNR, NLR, LRN theo cả hai dạng đệ quy và lặp

e. Phép quay đơn Left-Left. Sử dụng khi cây con bên trái lệch trái hoặc cân bằng.

```
C#: void RotateLL(ref TreeNode T)          // Xem hình minh họa trong phần
C++: void RotateLL(NodePointer &T)         // ôn tập lý thuyết
{
      Gán T1 = Cây con trái của T;
      Gán Cây con trái của T = Cây con phải của T1;
      Gán Cây con phải của T1 = T;
      Nếu (Cây con trái T1 đang cân bằng) thì
      {                                                  // Sau khi quay
            Gán hệ số cân bằng của T = LH;               // T lệch trái
            Gán hệ số cân bằng của T1 = RH;              // T1 lệch phải
      }
      Ngược lại, nếu (Cây con trái T1 đang lệch trái) thì
      {
            Gán hệ số cân bằng của T = EH;               // T cân bằng
            Gán hệ số cân bằng của T1 = EH;              // T1 cân bằng
      }
      T = T1;           // Thay thế nút T bởi nút T1 (đổi vai trò của T1)
```

```
}
```

f. Phép quay đơn Right-Right. Sử dụng khi cây con bên phải lệch phải hoặc cân bằng.

```
C#: void RotateRR(ref TreeNode T)          // Xem hình minh họa trong phần
C++: void RotateRR(NodePointer &T)         // ôn tập lý thuyết
{
      Gán T1 = Cây con phải của T;
      Gán Cây con phải của T = Cây con trái của T1;
      Gán Cây con trái của T1 = T;
      Nếu (Cây con phải T1 đang cân bằng) thì
      {                                                  // Sau khi quay
            Gán hệ số cân bằng của T = RH;               // T lệch phải
            Gán hệ số cân bằng của T1 = LH;              // T1 lệch trái
      }
      Ngược lại, nếu (Cây con phải T1 đang lệch phải) thì
      {
            Gán hệ số cân bằng của T = EH;               // T cân bằng
            Gán hệ số cân bằng của T1 = EH;              // T1 cân bằng
      }
      T = T1;           // Thay thế nút T bởi nút T1 (đổi vai trò của T1)
}
```

g. Phép quay kép Left-Right. Sử dụng khi cây con bên trái lệch phải.

```
C#: void RotateLR(ref TreeNode T)          // Xem hình minh họa trong phần
C++: void RotateLR(NodePointer &T)         // ôn tập lý thuyết
{
      Gán T1 = Cây con trái của T;
      Gán T2 = Cây con phải của T1;
      Gán Cây con phải của T1 = Cây con trái của T2;
      Gán Cây con trái của T = Cây con phải của T2;
      Gán Cây con trái của T2 = T1;
      Gán Cây con phải của T2 = T;
      Nếu (Cây con T2 đang lệch trái) thì
      {                                                  // Sau khi quay
            Gán hệ số cân bằng của T = RH;               // T lệch phải
            Gán hệ số cân bằng của T1 = EH;              // T1 cân bằng
      }
      Ngược lại, Nếu (Cây con T2 đang cân bằng) thì
      {                                                  // Sau khi quay
```

```
            Gán hệ số cân bằng của T = EH;              // T cân bằng
            Gán hệ số cân bằng của T1 = EH;             // T1 cân bằng
      }
      Ngược lại, nếu (Cây con T1 đang lệch phải) thì
      {
            Gán hệ số cân bằng của T = EH;              // T cân bằng
            Gán hệ số cân bằng của T1 = LH;             // T1 lệch trái
      }
      Gán hệ số cân bằng của T2 = EH;             // T2 cân bằng
      T = T2;           // Thay thế nút T bởi nút T2 (đổi vai trò của T2)
}
```

h. Phép quay kép Right-Left. Sử dụng khi cây con bên phải lệch trái.

```
C#: void RotateRL(ref TreeNode T)          // Xem hình minh họa trong phần
C++: void RotateRL(NodePointer &T)         // ôn tập lý thuyết
{
      Gán T1 = Cây con phải của T;
      Gán T2 = Cây con trái của T1;
      Gán Cây con trái của T1 = Cây con phải của T2;
      Gán Cây con phải của T = Cây con trái của T2;
      Gán Cây con trái của T2 = T;
      Gán Cây con phải của T2 = T1;
      Nếu (Cây con T2 đang lệch trái) thì
      {                                                 // Sau khi quay
            Gán hệ số cân bằng của T = EH;              // T cân bằng
            Gán hệ số cân bằng của T1 = RH;             // T1 lệch phải
      }
      Ngược lại, Nếu (Cây con T2 đang cân bằng) thì
      {                                                 // Sau khi quay
            Gán hệ số cân bằng của T = EH;              // T cân bằng
            Gán hệ số cân bằng của T1 = EH;             // T1 cân bằng
      }
      Ngược lại, nếu (Cây con T1 đang lệch phải) thì
      {
            Gán hệ số cân bằng của T = LH;              // T lệch trái
            Gán hệ số cân bằng của T1 = EH;             // T1 cân bằng
      }
      Gán hệ số cân bằng của T2 = EH;             // T2 cân bằng
```

```
        T = T2;             // Thay thế nút T bởi nút T2 (đổi vai trò của T2)
}
```

i. Cân bằng lại cây khi cây lệch trái

```
C#: int ReBalanceLeft(ref TreeNode T)            // Xem hình minh họa trong phần
C++: int ReBalanceLeft(NodePointer &T)           // ôn tập lý thuyết
{
        Gán T1 = Cây con bên trái của T;
        switch (Hệ số cân bằng của T)
        {
                LH: Quay đơn Left-Left tại nút T; Trả về BAL;
                EH: Quay đơn Left-Left tại nút T; Trả về DEV;
                RH: Quay kép Left-Right tại nút T; Trả về BAL;
                Default: Trả về BAL;
        }
}
```

j. Cân bằng lại cây khi cây lệch phải

```
C#: int ReBalanceRight(ref TreeNode T)           // Xem hình minh họa trong phần
C++: int ReBalanceRight(NodePointer &T)          // ôn tập lý thuyết
{
        Gán T1 = Cây con bên phải của T;
        switch (Hệ số cân bằng của T)
        {
                LH: Quay kép Right-Left tại nút T; Trả về BAL;
                EH: Quay đơn Right-Right tại nút T; Trả về DEV;
                RH: Quay kép Right-Right tại nút T; Trả về BAL;
                Default: Trả về BAL;
        }
}
```

k. Thêm một phần tử mới vào cây

```
C#: int Insert(TreeNode T, int item)
C++: int Insert(AVLTree &T, DataType item)
{
        Nếu (Cây T = rỗng) thì
        {
                Tạo nút chứa dữ liệu item và gán cho T;
                Nếu (Cây T = rỗng) thì trả về FAIL;       // Không thể chèn
                Ngược lại, trả về BAL;
```

```
        }
        Ngược lại
        {
                Gán result = ZERO;
                Nếu (T.Key = item) thì trả về FAIL;        // phần tử đã có
                Ngược lại, nếu (T.Key > item) thì
                {
                        Gán result = Insert(T.LeftChild, item);
                        Nếu (result khác BAL) thì trả về result;
                        Ngược lại
                        {
                                switch (Hệ số cân bằng của T)
                                {
                                        LH: Cân bằng lại vì T lệch trái; Trả về DEV;
                                        EH: Gán hệ số cân bằng của T = LH; Trả về BAL;
                                        RH: Gán hệ số cân bằng của T = EH; Trả về DEV;
                                }
                        }
                }
                Ngược lại
                {
                        Gán result = Insert(T.RightChild, item);
                        Nếu (result khác BAL) thì trả về result;
                        Ngược lại
                        {
                                switch (Hệ số cân bằng của T)
                                {
                                        LH: Gán hệ số cân bằng của T = EH; Trả về DEV;
                                        EH: Gán hệ số cân bằng của T = RH; Trả về BAL;
                                        RH: Cân bằng lại vì T lệch phải; Trả về DEV;
                                }
                        }
                }
        }
}
```

l. Tìm nút trái nhất của cây con bên phải Q để thay thế cho P

```
C#: int SearchStandFor(ref TreeNode p, ref TreeNode q)
```

```
C++: int SearchStandFor(NodePointer &p, NodePointer &q)
{
      Nếu (q có cây con bên trái) thì
      {
            Gán kq = Tìm nút trong cây con bên trái Q để thay thế cho P;
            Nếu (kq < 2) thì trả về kq;          // Không thể chèn
            switch (Hệ số cân bằng của q)
            {
                  LH: Gán hệ số cân bằng của q = EH; Trả về BAL; // 2
                  EH: Gán hệ số cân bằng của q = RH; Trả về DEV; // 1
                  RH: Trả về kết quả sau khi Cân bằng lại cây tại nút q;
            }
      }
      Ngược lại
      {
            Gán dữ liệu từ nút q vào nút p;
            Gán p = q;                           // Đổi vai trò của p, q
            Gán q = Cây con trái của q;
            return BAL;
      }
}
```

m. Xóa một nút ra khỏi cây

```
C#: int Delete(TreeNode T, int item)
C++: int Delete(AVLTree &T, DataType item)
{
      Nếu (Cây T = rỗng) thì trả về FAIL;              // Không thể xóa
      Khởi tạo biến result = ZERO;
      Nếu (item nằm trong cây con trái của T) thì
      {
            Gán result = Delete(T.LeftChild, item);
            Nếu (result != BAL) thì trả về result;
            switch (Hệ số cân bằng của T)
            {
                  LH: Gán hệ số cân bằng của T = EH; Trả về BAL;
                  EH: Gán hệ số cân bằng của T = RH; Trả về DEV;
                  RH: Trả về kết quả sau khi Cân bằng lại vì T lệch phải;
            }
```

```
}
Ngược lại, Nếu (item nằm trong cây con phải của T) thì
{
	Gán result = Delete(T.RightChild, item);
	Nếu (result != BAL) thì trả về result;
	switch (Hệ số cân bằng của T)
	{
		LH: Trả về kết quả sau khi Cân bằng lại vì T lệch trái;
		EH: Gán hệ số cân bằng của T = LH; Trả về DEV;
		RH: Gán hệ số cân bằng của T = EH; Trả về BAL;
	}
}
Ngược lại // TRường hợp tìm thấy nút p chứa item
{
	Gán p = T;
	Nếu (T không có con trái) thì
	{
		Gán T = Cây con phải của T;
		Gán result = BAL;
	}
	Ngược lại, Nếu (T không có con phải) thì
	{
		Gán T = Cây con trái của T;
		Gán result = BAL;
	}
	Ngược lại
	{
		Gán result = SearchStandFor(p, p.RightChild);
		Nếu (result != BAL) thì trả về result;
		switch (Hệ số cân bằng của T)
		{
			LH: Trả về kết quả sau khi Cân bằng lại vì T lệch trái;
			EH: Gán hệ số cân bằng của T = LH; Trả về BAL;
			RH: Gán hệ số cân bằng của T = EH; Trả về BAL;
		}
	}
}
```

```
        Trả về result;
    }
```

n. Xây dựng cây AVL từ một tập dữ liệu cho trước

```
C#: void BuildTree(int[] items)
{
        foreach (int key in items)
        {
                Gọi hàm chèn phần tử key vào cây;
        }
}
C++: void BuildTree(AVLTree &T, DataType items[], int count)
{
        Tạo cây rỗng T;
        for (int i=0; i<count; i++)
        {
                Gọi hàm chèn phần tử items[i] vào cây T;
        }
}
```

o. Viết hàm xuất hệ số cân bằng và dữ liệu chứa trong các nút của cây theo thứ tự NLR

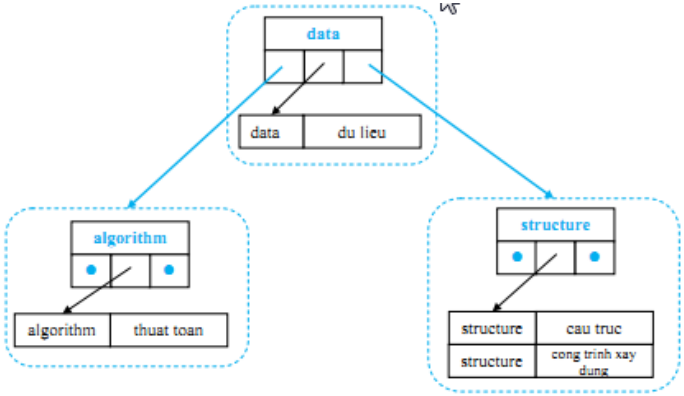
Bài 2. Viết chương trình thực hiện các thao tác trên cây AVL với dữ liệu có kiểu chuỗi và được lấy từ tập tin. Ngoài ra, bổ sung vào cấu trúc của nút một con trỏ trỏ đến nút cha của nó.

Bài 3. Kiểu dữ liệu trừu tượng: Từ điển

Cài đặt KDLTT từ điển Anh-Việt bằng cây tìm kiếm nhị phân theo mô tả như sau: Một từ điển có nhiều từ, một từ có thể có nhiều nghĩa. Với mỗi từ, ta cần lưu từ gốc tiếng Anh (key) và nghĩa tiếng Việt của từ đó (meaning). Sau đây là một số gợi ý để cài đặt.

```
struct Entry{
        string key;
        string meaning;
};
```

```
struct Node{
        string key;
        list<Entry> data;
        Node * left;
        Node * right;
};
```



Yêu cầu: Định nghĩa các phương thức (hàm) sau

- FindAll(k): trả về danh mục từ (Entry) có khóa k
- Insert(k, m): Thêm vào từ điển một mục từ có khóa k và nghĩa của nó là m
- Remove(k): Loại bỏ một mục từ (Entry) có khóa k khỏi từ điền
- Entries(): Trả về danh sách tất cả các mục từ trong từ điển, sắp xếp theo thứ tự tăng dần
- Size(): Trả về số mục từ trong từ điển
- LoadFile(fname): Đọc dữ liệu từ file có tên fname, đưa vào cây.

Bài 4. Viết chương trình quản lý danh sách lớp.

Mỗi sinh viên gồm các thành phần:

+Mã SV: char[10];
+Mã Lớp: int
+Tên SV: char[255];
+DiemToan
+DiemLy
+DiemHoa

Mỗi lớp chứa gồm các thông tin:

+Mã Lớp: int
+Tên Lớp: char[10];
+Khóa

Thành phần khóa chính (và Index) của danh sách sinh viên chính là mã SV.
Thành phần khóa chính (và Index) của danh sách lớp chính là mã lớp.

Xây dựng và quản lý danh sách lớp sử dụng cây nhị phân tìm kiếm (Binary Search Tree). Hiển thị menu thực hiện các chức năng sau (mỗi chức năng thực hiện bằng hàm). Thành phần dữ liệu trong mỗi Node là giá trị kiểu Sinh viên.

a. Thêm 1 lớp mới.
b. Thêm một sinh viên
   - Nếu mã SV đã có thì hiển thị sinh viên đó ra màn hình, cùng với thông báo không thể nhập sinh viên đã có
   - Mã lớp phải tồn tại trong danh sách lớp. Nếu chưa có, phải hiện thông báo lỗi.
c. Tìm một sinh viên theo mã SV
   - Khi tìm thấy, hiển thị mã, tên, điểm, mã lớp và tên lớp.
d. Lưu danh sách sinh viên-lớp vào file
e. Đọc danh sách sinh viện từ file.
f. Hiển thị danh sách sinh viên
   - Tăng dần theo mã SV
   - Giảm dần theo mã SV
   - Mỗi sinh viên hiển thị điểm toán, lý, hóa và điểm trung bình
g. Tìm tất cả sinh viên theo tên nhập vào
h. Hiển thị tất cả sinh viên theo mã lớp nhập vào

i. Hiển thị tất cả sinh viên theo tên lớp nhập vào
j. Xóa một sinh viên ra khỏi danh sách
k. Xóa một lớp ra khỏi danh sách
l. Tìm tất cả sinh viên có điểm trung bình lớn nhất
m. Tìm tất cả sinh viên có điểm trung bình lớn nhất trong một lớp

VI. Bài tập làm thêm:

Bài 1. Thống kê từ trong tập tin văn bản

Các văn bản được lưu trữ thành từng dòng trong các file văn bản, mỗi dòng có chiều dài không quá 127 ký tự. Hãy đề xuất cấu trúc dữ liệu thích hợp để lưu trữ trong bộ nhớ trong của máy tính tên từ và số lần xuất hiện của từ đó trong tập tin văn bản. Với cấu trúc dữ liệu này, hãy trình bày thuật toán và cài đặt chương trình thực hiện việc thống kê xem các từ trong file văn bản xuất hiện với tần suất như thế nào? Cho biết văn bản có bao nhiêu từ, bao nhiêu tên từ?

Bài 2. Tìm hiểu và cài đặt các thao tác trên kiểu dữ liệu trừu tượng cây đỏ - đen.

- vi.wikipedia.org/wiki/Cây_đỏ_đen
- en.wikipedia.org/wiki/Red–black_tree
- http://www.codeproject.com/Articles/8287/Red-Black-Trees-in-C

Bài 3. Tìm hiểu và cài đặt các thao tác trên kiểu dữ liệu trừu tượng B-Cây.

- http://violet.vn/c2phuan/present/show/entry_id/2046853
- vi.wikipedia.org/wiki/B-cây
- http://www.bluerwhite.org/btree/

=== HẾT ===