CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT 2

LAB 3 – CÂY TÌM KIẾM NHỊ PHÂN (8 TIẾT)

I. Mục tiêu

Sau khi thực hành, sinh viên cần:

- Nắm vững các khái niệm về cây, cây nhị phân, cây tìm kiếm nhị phân (BST).
- Cài đặt được kiểu dữ liệu cây tìm kiếm nhị phân và các thao tác, phép toán trên cây.
- 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: Lab3_CTK39_HoTen_MSSV_Nhom#. Nén thư mục, đặt tên tập tin nén theo dạng sau: Lab3_CTK39_HoTen_MSSV_Nhom#.rar. Ví dụ: Lab3_CTK39_NguyenVanA_141111_Nhom1.rar.
- Sinh viên sẽ nộp bài Lab qua mạng tại phòng lab theo hướng dẫn của giáo viên.

III. Ôn tập lý thuyết

1. Tìm kiếm một nút có khóa K cho trước trên cây BST

Để tìm kiếm một nút có khóa K trên cây BST, ta tiến hành xét từ nút gốc, bằng cách so sánh khóa của nút đang xét với khóa K.

- Nếu nút đang xét bằng NULL thì không có khóa K trên cây.
- Nếu K bằng khóa của nút đang xét thì giải thuật dừng và ta đã tìm được nút chứa khóa K.
- Nếu K lớn hơn khóa của nút đang xét thì ta tiến hành (một cách đệ quy) việc tìm khóa K trên cây con bên phải.
- Nếu K nhỏ hơn khóa của nút đang xét thì ta tiến hành (một cách đệ quy) việc tìm khóa K trên cây con bên trái.

2. Thêm một nút có khóa K cho trước vào cây BST

Trong cây BST, không có hai nút trùng khóa. Vì vậy, khi thêm một nút mới, cần tìm kiếm để xác định có tồn tại nút chứa giá trị K hay chưa. Nếu có thì thuật toán kết thúc (không thêm). Ngược lại, nó sẽ thêm một nút mới chứa K.

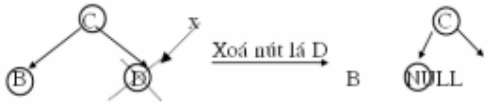
Ta tiến hành từ nút gốc bằng cách so sánh khóa của nút gốc với khóa K.

- Nếu nút đang xét bằng NULL thì khóa K chưa có trên cây, do đó ta thêm nút mới chứa khóa K.
- Nếu K bằng khóa của nút đang xét thì thuật toán dừng, không cần thêm nút mới để đảm bảo cấu trúc cây.
- Nếu K lớn hơn khóa của nút đang xét thì ta tiến hành thuật toán này trên cây con bên phải.
- Nếu K nhỏ hơn khóa của nút đang xét thì ta tiến hành thuật toán này trên cây con bên trái.

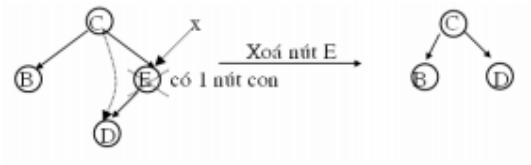
3. Xóa một nút có khóa K cho trước ra khỏi cây BST

Để xóa một nút có chứa khóa K, ta phải tìm kiếm xem nút chứa khóa K đó có trên cây hay không. Nếu không tìm thấy, thuật toán kết thúc. Nếu tìm gặp nút N có chứa khóa K, có 3 trường hợp xảy ra:

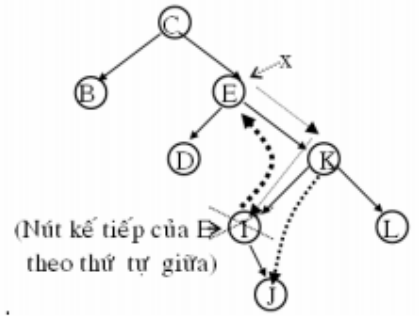
- TH1 - Nút N là nút lá: Ta thay nút này bởi giả trị NULL.
- TH2 - Nút N có đúng 1 con (trái hoặc phải): Ta thay nút N bởi nút con của nó.
- TH3 - N có đủ 2 nút con: Ta thay nó bằng nút lớn nhất bên cây con trái của nó (chính là nút bên phải nhất của cây con trái) hoặc nút bé nhất bên cây con phải của nó (chính là nút trái nhất của cây con phải). Trong thuật toán được cài đặt bên dưới (phần hướng dẫn thực hành), ta sẽ thay K bởi khóa trong nút con trái nhất của cây con bên phải rồi xóa nút cực trái này. Việc xóa nút cực trái sẽ rơi vào 1 trong 2 trường hợp trên.



Trường hợp 1



Trường hợp 2



Trường hợp 3

4. *Duyệt qua các nút trên cây*

Duyệt theo thứ tự trước (tiền tự, Preorder, NLR)	Duyệt theo thứ tự giữa (trung tự, Inorder, LNR)	Duyệt theo thứ tự sau (hậu tự, Postorder, LRN)
Thăm nút gốc (xuất, thay đổi giá trị..)	Thăm cây con trái theo thứ tự giữa	Thăm cây con trái theo thứ tự sau
Thăm cây con trái theo thứ tự trước	Thăm nút gốc (xuất, thay đổi giá trị, …)	Thăm cây con phải theo thứ tự sau
Thăm cây con phải theo thứ tự trước	Thăm cây con phải theo thứ tự giữa	Thăm nút gốc (xuất, thay đổi giá trị, …)

Ngoài ra, ta có thể duyệt cây theo chiều rộng (tức duyệt cây theo chiều ngang, liệt kê các nút theo từng mức). Để cài đặt thuật toán này, ta cần dùng đến một hàng đợi.

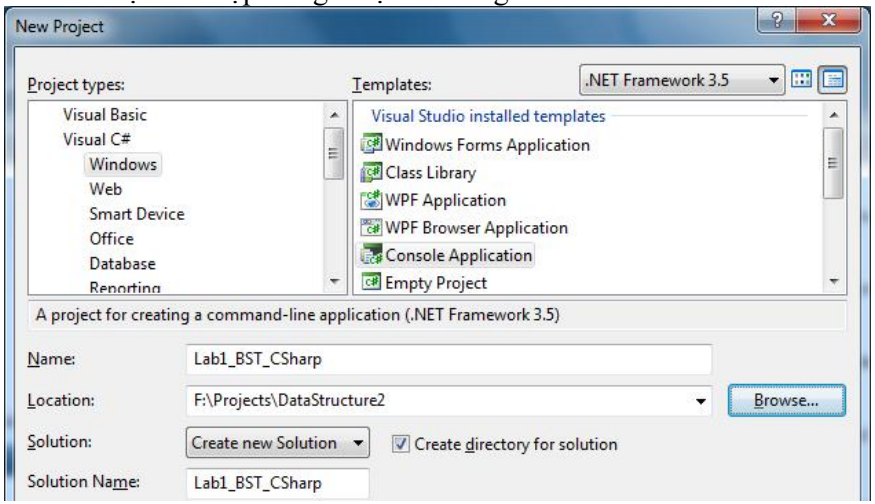
```
Khởi tạo hàng đợi chứa nút gốc của cây
Trong khi hàng đợi khác rỗng
{
   Lấy nút được lưu trong hàng đợi
   Xử lý nút đó
   Đưa các con của nút đang xét vào hàng đợi.
}
```

IV. Hướng dẫn thực hành

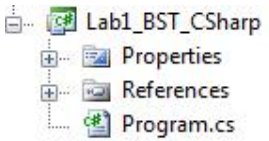
1. *Tạo dự án*

Sinh viên có thể chọn cài đặt cây tìm kiếm nhị phân 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).

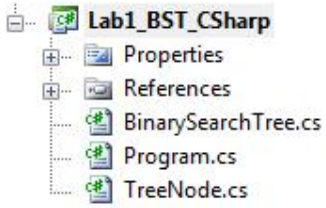
- Vào menu File > chọn New Project
- Chọn và nhập các giá trị như trong hình



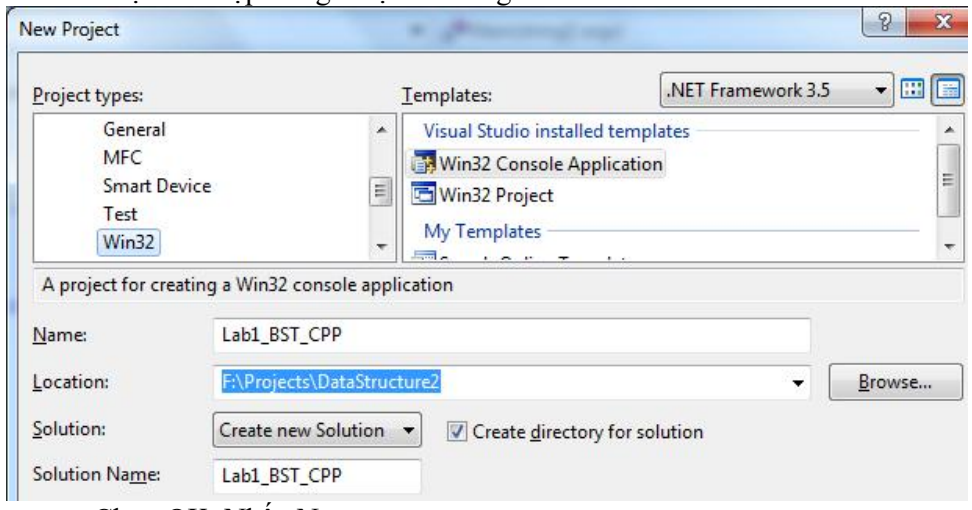
- Chọn OK. Ta có một project như sau:



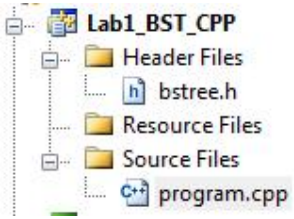
- Nhấp phải chuột vào project Lab1_BST_CSharp, chọn Add > Class … Đặt tên cho lớp đó là TreeNode.
- Tương tự, ta tạo thêm lớp BinarySearchTree.
- Sau khi tạo xong, ta được project như hình sau:



- Vào menu File > chọn New Project
- Chọn và nhập các giá trị như trong hình



- Chọn OK. Nhấn Next.
- Trong mục Additional options, chọn Empty project. Nhấn Finish.
- Tiếp theo, nhấp phải vào thư mục Header files, chọn Add > New Item. Trong cửa sổ Add New Item, chọn mục Code (khung bên trái), rồi chọn Header file (.h) (khung bên phải). Đặt tên (mục Name) cho nó là bstree.h
- Nhấp phải chuột vào thư mục Source files, chọn Add > New Item. Chọn mục Code, sau đó chọn C++ file (.cpp). Đặt tên cho tập tin này là program.cpp.
- Xem phần phụ lục để tạo thêm file stack.h và queue.h
- Sau khi tạo xong, ta có được project như sau:



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 BST) như sau:

- Nhấp đôi chuột vào file bstree.h, định nghĩa các kiểu dữ liệu và hàm như sau:

```csharp
namespace Lab1_BST_CSharp
{
    // Kiểu dữ liệu thể hiện 1 nút của cây BST
    class TreeNode
    {
        public int Key;      // Dữ liệu chứa trong nút
        public TreeNode LeftChild;  // Nút con trái
        public TreeNode 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 TreeNode(int data)
        {
            this.Key = data;
            this.LeftChild = null;
            this.RightChild = null;
        }
        public TreeNode(int data, TreeNode left,
                                  TreeNode right)
        {
            this.Key = data;
            this.LeftChild = left;
            this.RightChild = right;
        }
```

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

// Kiểu dữ liệu nút của cây BST
struct TreeNode
{
    DataType Key;            // Dữ liệu
    TreeNode* LeftChild;     // Nút con trái
    TreeNode* RightChild;    // Nút con phải
}

// Định nghĩa kiểu con trỏ
typedef TreeNode* NodePointer;

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

// Hàm khởi tạo một nút
NodePointer CreateNode(DataType item)
{
    NodePointer p = new TreeNode;
    if (p != NULL)
    {
        p->Key = item;
        p->LeftChild = NULL;
        p->RightChild = NULL;
    }
    return p;
}
// Hàm khởi tạo một nút
NodePointer CreateNode(DataType item,
        NodePointer left, NodePointer right)
{
    NodePointer p = new TreeNode;
    if (p != NULL)
    {
        p->Key = item;
        p->LeftChild = left;
        p->RightChild = right;
    }
```

```
        // Chuyển một nút sang chuỗi để hiển thị
        public override string ToString()
        {
            return this.Key.ToString();
        }
    }
}
```

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

```
namespace Lab1_BST_CSharp
{
    // Kiểu dữ liệu cây tìm kiếm nhị phân
    class BinarySearchTree
    {
        public int NodeCount;   // Số nút của cây
        public TreeNode Root;   // Gốc của cây

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

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

```
    return p;
}
// Kiểm tra có nút con trái hay không?
bool HasLeft(NodePointer p)
{
    return (p->LeftChild != NULL);
}
// Kiểm tra có nút con phải hay không?
bool HasRight(NodePointer p)
{
    return (p->RightChild != NULL);
}
// Kiểm tra có phải là nút lá ko?
bool IsLeaf(NodePointer p)
{
    return !(HasLeft(p) || HasRight(p));
}
// Khởi tạo cây BST rỗng
void CreateTree(BSTree &root)
{
    root = NULL;
}
// Kiểm tra có phải là cây rỗng?
bool IsEmpty(BSTree root)
{
    return (root == NULL);
}
```

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

```
#include <iostream>
using namespace std;

#include "bstree.h"

void main()
{
}
```

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.

3. *Cài đặt các thao tác (phép toán) trên cây BST*

```
// Tìm nút chứa giá trị key cho trước (Đệ quy)
// Ví dụ cách gọi hàm:
// int key = 123;
// BinarySearchTree bst = new BinarySearchTree();
// TreeNode parent = null;
// TreeNode result = bst.Search(bst.Root, key, ref parent);
public TreeNode Search(TreeNode node, int key,
                       ref TreeNode parent)
{
    // Không tìm thấy giá trị key trong cây
    if (node == null) return null;

    if (node.Key == key)    // Tìm thấy key
        return node;
    else
    {
        parent = node;
        if (node.Key > key)
            // Tìm key trong cây con bên trái
            return Search(node.LeftChild, key, ref parent);
        else
            // Tìm key trong cây con bên phải
            return Search(node.RightChild, key, ref parent);
    }
}

// Tìm nút chứa giá trị key cho trước (Dạng lặp)
// Ví dụ cách gọi hàm:
// int key = 123;
// BinarySearchTree bst = new BinarySearchTree();
// TreeNode parent = null;
// TreeNode result = bst.Search(key, ref parent);
```

```
// Tìm nút chứa giá trị key cho trước (Đệ quy)
// Ví dụ cách gọi hàm:
// int key = 123;
// BSTree bst;
// NodePointer parent = null;
// NodePointer result = Search(bst, key, parent);
NodePointer Search(BSTree root, DataType key,
                   NodePointer &parent)
{
    // Không tìm thấy giá trị key trong cây
    if (root == NULL) return NULL;

    if (root->Key == key)   // Tìm thấy key
        return root;
    else
    {
        parent = root;
        if (root->Key > key)
            // Tìm key trong cây con bên trái
            return Search(root->LeftChild, key, parent);
        else
            // Tìm key trong cây con bên phải
            return Search(root->RightChild, key, parent);
    }
}

// Tìm nút chứa giá trị key cho trước (Dạng lặp)
// Ví dụ cách gọi hàm:
// int key = 123;
// BSTree bst;
// NodePointer parent = null;
// NodePointer result = Search(bst, key, parent);
```

```C#
public TreeNode SearchLoop(int key, ref TreeNode parent)
{
    TreeNode node = this.Root;  // Bắt đầu từ gốc
    while (node != null)        // Duyệt qua các nút
    {
        // Nếu tìm thấy, dừng việc tìm kiếm
        if (node.Key == key) break;
        else
        {
            parent = node;      // Lưu lại nút cha
            if (node.Key > key) // Nhảy qua cây con trái
                node = node.LeftChild;
            else                // Nhảy qua cây con phải
                node = node.RightChild;
        }
    }
    return node;
}

// Hàm thêm một nút vào cây (dạng đệ quy)
// Ví dụ cách gọi hàm
// int key = 123;
// BinarySearchTree bst = new BinarySearchTree();
// bool result = bst.Insert(ref bst.Root, key);
public bool Insert(ref TreeNode node, int key)
{
    if (node == null)
    {
        node = new TreeNode(key);
        return true;
    }
    else if (node.Key > key)
        return Insert(ref node.LeftChild, key);
    else if (node.Key < key)
        return Insert(ref node.RightChild, key);
    else return false;
}

// Hàm thêm một nút vào cây (dạng lặp)
// Ví dụ cách gọi hàm
// int key = 123;
// BinarySearchTree bst = new BinarySearchTree();
// bool result = bst.Insert(key);
```

```C++
NodePointer SearchLoop(BSTree root, DataType key,
                       NodePointer &parent)
{
    NodePointer node = root;
    while (node != NULL)
    {
        if (node->Key == key) break;
        else
        {
            parent = node;
            if (node->Key > key)
                node = node->LeftChild;
            else
                node = node->RightChild;
        }
    }
    return node;
}

// Hàm thêm một nút vào cây (dạng đệ quy)
// Ví dụ cách gọi hàm
// DataType key = 123;
// BSTree bst;
// CreateTree(bst);
// bool result = Insert(bst, key);
bool Insert(BSTree &root, DataType key)
{
    // Nếu nút đang xét là rỗng
    if (root == NULL)
    {
        // Tạo nút và gán dữ liệu cho nó
        root = CreateNode(key);
        return true;            // Thêm thành công
    }
    else if (root->Key > key)   // Thêm vào cây con trái
        return Insert(root->LeftChild, key);
    else if (root->Key < key)   // Thêm vào cây con phải
        return Insert(root->RightChild, key);
    else return false;          // Dữ liệu đã tồn tại
}

// Hàm thêm một nút vào cây (dạng lặp)
// Ví dụ cách gọi hàm
```

```
public bool InsertLoop(int key)
{
    TreeNode node, parent = null;
    if (SearchLoop(key, ref parent) != null)
        return false;
    else
    {
        node = new TreeNode(key);
        if (parent == null)
            this.Root = node;
        else if (parent.Key > key)
            parent.LeftChild = node;
        else
            parent.RightChild = node;
        return true;
    }
}

// Xóa nút trái (nhỏ) nhất trong các con của node
// Đồng thời trả về giá trị được lưu trong nút đó
private int DeleteMin(ref TreeNode node)
{
    // Nếu không có con trái => nút hiện hành
    // là nút trái nhất (nhỏ nhất)
    if (node.LeftChild == null)
    {
        int key = node.Key; // Lưu lại khóa
        node = node.RightChild; // Xóa nút
        return key;             // Trả về khóa
    }
    else
        // Nếu có con trái, tiếp tục xét con trái
        return DeleteMin(ref node.LeftChild);
}


// Xóa một nút ra khỏi cây (dạng đệ quy)
// Ví dụ cách gọi hàm
// DataType key = 123;
// BinarySearchTree bst = new BinarySearchTree();
// bool result = bst.Delete(bst.Root, key);
public bool Delete(ref TreeNode root, int key)
{
```

```
// DataType key = 123;
// BSTree bst;
// CreateTree(bst);
// bool result = InsertLoop(bst, key);
bool InsertLoop(BSTree &root, DataType key)
{
    NodePointer node, parent = NULL;

    // Nếu tồn tại nút chứa khóa key
    if (Search(root, key, parent) != NULL)
        return false;       // Cho biết thêm thất bại
    else
    {
        // Tạo nút mới chứa khóa key
        node = CreateNode(key);
        if (parent == NULL)     // Cây đang rỗng
            root = node;        // => tạo nút gốc
        else if (parent->Key > key) // Thêm vào con trái
            parent->LeftChild = node;
        else                        // Thêm vào con phải
            parent->RightChild = node;
        return true;
    }
}

// Xóa nút trái (nhỏ) nhất trong các con của node
// Đồng thời trả về giá trị được lưu trong nút đó
DataType DeleteMin(NodePointer &node)
{
    // Nếu không có con trái => nút hiện hành
    // là nút trái nhất (nhỏ nhất)
    if (node->LeftChild == NULL)
    {
        DataType key = node->Key;   // Lưu lại khóa
        node = node->RightChild;    // Xóa nút
        return key;                 // Trả về khóa
    }
    else
        // Nếu có con trái, tiếp tục xét con trái
        return DeleteMin(node->LeftChild);
}

// Xóa một nút ra khỏi cây (dạng đệ quy)
```

```
    // root == null : không tìm thấy nút nào có chứa
    // giá trị key => không cần phải xóa
    if (root != null)
    {
        if (root.Key > key)
            // Xóa nút key trong cây con bên trái
            return Delete(ref root.LeftChild, key);
        else if (root.Key < key)
            // Xóa nút key trong cây con bên phải
            return Delete(ref root.RightChild, key);
        else
        {
            // Nút cần xóa là nút lá
            if (root.LeftChild == null &&
                 root.RightChild == null)
            {
                root = null;    // xóa nút lá
            }
            // Trường hợp có 1 con bên phải
            else if (root.LeftChild == null)
                root = root.RightChild;
            // Trường hợp có 1 con bên trái
            else if (root.RightChild == null)
                root = root.LeftChild;
            // Trường hợp có đủ 2 con, Xóa
            else root.Key = DeleteMin(ref root.RightChild);
            return true;
        }
    }
    else return false;
}

// Xóa một nút ra khỏi cây (dạng đệ quy)
// Ví dụ cách gọi hàm
// DataType key = 123;
// BinarySearchTree bst = new BinarySearchTree();
// bool result = bst.DeleteLoop(key);
public bool DeleteLoop(int key)
{
    TreeNode x, parent = null, leftTree, rightTree;

    // Tìm nút chứa key và cha của nó
```

```
// Ví dụ cách gọi hàm
// DataType key = 123;
// BSTree bst; CreateTree(bst);
// bool result = Delete(bst, key);
bool Delete(BSTree &root, DataType key)
{
    // root == NULL : không tìm thấy nút nào có chứa
    // giá trị key => không cần phải xóa
    if (root != NULL)
    {
        if (root->Key > key)
            // Xóa nút key trong cây con bên trái
            return Delete(root->LeftChild, key);
        else if (root->Key < key)
            // Xóa nút key trong cây con bên phải
            return Delete(root->RightChild, key);
        else
        {
            // Nút cần xóa là nút lá
            if (root->LeftChild == NULL &&
                 root->RightChild == NULL)
            {
                root = NULL;    // xóa nút lá
            }
            // Trường hợp có 1 con bên phải
            else if (root->LeftChild == NULL)
                root = root->RightChild;
            // Trường hợp có 1 con bên trái
            else if (root->RightChild == NULL)
                root = root->LeftChild;
            // Trường hợp có đủ 2 con, Xóa
            else root->Key = DeleteMin(root->RightChild);
            return true;
        }
    }
    else return false;
}

// Xóa một nút ra khỏi cây (dạng đệ quy)
// Ví dụ cách gọi hàm
// DataType key = 123;
// BSTree bst; CreateTree(bst);
// bool result = DeleteLoop(bst, key);
```

```
    x = Search(this.Root, key, ref parent);
    if (x == null)
        return false;
    else
    {
        // Nếu nút có 2 con
        if (x.HasLeft && x.HasRight)
        {
            rightTree = x.RightChild;
            parent = x;
            // Tìm nút trái nhất của cây con phải
            while (rightTree.LeftChild != null)
            {
                parent = rightTree;
                rightTree = rightTree.LeftChild;
            }
            // Hoán vị dữ liệu của nút trái nhất với
            // nút x chứa key.
            x.Key = rightTree.Key;
            x = rightTree;
        }
        // Nếu nút có một con hoặc ko có con
        leftTree = x.LeftChild;
        if (x.HasRight)
            leftTree = x.RightChild;

        // Nếu nút bị xóa là nút gốc
        if (parent == null)     // Thì nút con của nút bị
            this.Root = leftTree;   // xóa sẽ trở thành nút gốc
        else // Nút bị xóa nằm bên trái nút cha
            if (parent.LeftChild == x)
                parent.LeftChild = leftTree;
            else // Nút bị xóa nằm bên phải nút cha
                parent.RightChild = leftTree;
        return true;
    }
}

// Hàm xây dựng cây từ một tập có n phần tử cho trước
// Ví dụ cách gọi hàm
// DataType data[] = { 14, 20, 2, 4, 17, 11, 8, 1, 19 };
// BinarySearchTree bst = new BinarySearchTree();
```

```
bool DeleteLoop(BSTree &root, DataType key)
{
    NodePointer x, parent, leftTree, rightTree;

    // Tìm nút chứa key và cha của nó
    x = Search(root, key, parent);
    if (x == NULL)
        return false;
    else
    {
        // Nếu nút có 2 con
        if (HasLeft(x) && HasRight(x))
        {
            rightTree = x->RightChild;
            parent = x;

            // Tìm nút trái nhất của cây con phải
            while (rightTree->LeftChild != NULL)
            {
                parent = rightTree;
                rightTree = rightTree->LeftChild;
            }
            // Hoán vị dữ liệu của nút trái nhất với
            // nút x chứa key.
            x->Key = rightTree->Key;
            x = rightTree;
        }
        // Nếu nút có một con hoặc ko có con
        leftTree = x->LeftChild;
        if (x->RightChild != NULL)
            leftTree = x->RightChild;

        // Nếu nút bị xóa là nút gốc
        if (parent == NULL)     // Thì nút con của nút bị
            root = leftTree;    // xóa sẽ trở thành nút gố
        else // Nút bị xóa nằm bên trái nút cha
            if (parent->LeftChild == x)
                parent->LeftChild = leftTree;
        else // Nút bị xóa nằm bên phải nút cha
            parent->RightChild = leftTree;
    }
}
```

```
// bst.BuildTree(data);
public void BuildTree(int[] items)
{
    // Duyệt qua từng phần tử trong tập dữ liệu
    foreach (var key in items)
    {
        InsertLoop(key);        // chèn vào cây
    }
}

// Hàm duyệt qua các nút theo thứ tự trước (NLR)-Đệ quy
// Trong hàm này, đối số thứ hai là một con trỏ hàm. Mục
// đích là để thực hiện một thao tác bất kỳ trên nút mà
// chúng ta đang duyệt qua.
// Ví dụ cách gọi hàm: Đầu tiên ta phải tạo hàm xử lý
// void PrintNode(TreeNode node)
// {
//      Console.WriteLine(node.Key + "\t");
// }
// Tiếp theo, trong hàm main, ta bổ sung lệnh sau:
// Action<TreeNode> xuly = new Action<TreeNode)(PrintNode);
// bst.Preorder(bst.Root, xuly);
public void Preorder(TreeNode root, Action<TreeNode> XuLy)
{
    if (root != null)
    {
        XuLy(root);
        Preorder(root.LeftChild, XuLy);
        Preorder(root.RightChild, XuLy);
    }
}

// Hàm duyệt qua các nút theo thứ tự trước (NLR)- Lặp
public void PreOrder(Action<TreeNode> XuLy)
{
    if (!IsEmpty)
    {
        TreeNode node = this.Root;
        // Khởi tạo ngăn xếp
        Stack<TreeNode> s = new Stack<TreeNode>();
        s.Push(node);           // Đưa nút gốc vào Stack

        // Kiểm tra stack khác rỗng hay không?
```

```
// Hàm xây dựng cây từ một tập có n phần tử cho trước
// Ví dụ cách gọi hàm
// DataType data[] = { 14, 20, 2, 4, 17, 11, 8, 1, 19 };
// BSTree bst;  CreateTree(bst);
// BuildTree(bst, data, 9);
void BuildTree(BSTree &root, DataType items[], int n)
{
    CreateTree(root);       // Khởi tạo cây
    // Duyệt qua từng phần tử trong tập dữ liệu
    for (int i=0; i<n; i++)
    {
        Insert(root, items[i]);     // chèn vào cây
    }
}

// Hàm duyệt qua các nút theo thứ tự trước (NLR)-Đệ quy
// Trong hàm này, đối số thứ hai là một con trỏ hàm. Mục
// đích là để thực hiện một thao tác bất kỳ trên nút mà
// chúng ta đang duyệt qua.
// Ví dụ cách gọi hàm: Đầu tiên ta phải tạo hàm xử lý
// void PrintNodeKey(NodePointer node)
// {
//      cout << node->Key << "\t";
// }
// Tiếp theo, trong hàm main, ta bổ sung lệnh sau:
// Preorder(bst, PrintNodeKey);
void Preorder(BSTree root, void (*XuLy)(NodePointer node))
{
    if (root)
    {
        XuLy(root);
        Preorder(root->LeftChild, XuLy);
        Preorder(root->RightChild, XuLy);
    }
}

// Hàm duyệt qua các nút theo thứ tự trước (NLR)-Đệ quy
void PreOrder(BSTree root, void (*XuLy)(NodePointer node))
{
    if (root)
```

```
        while (s.Count > 0)
        {
            node = s.Pop();      // Lấy 1 nút từ Stack
            XuLy(node);          // Xử lý nút đó

            // Đưa nút con phải vào stack nếu khác rỗng
            if (node.HasRight)
                s.Push(node.RightChild);
            // Đưa nút con trái vào stack nếu khác rỗng
            if (node.HasLeft)
                s.Push(node.LeftChild);
        }
    } // endif
}
```

```
    {
        NodePointer node;
        Stack s = CreateStack();    // Khởi tạo ngăn xếp
        Push(s, root);              // Đưa nút gốc vào Stack

        // Kiểm tra stack khác rỗng hay không?
        while (!EmptyStack(s))
        {
            node = Pop(s);      // Lấy 1 nút từ Stack
            XuLy(node);         // Xử lý nút đó
            // Đưa nút con phải vào stack nếu khác rỗng
            if (node->RightChild != NULL)
                Push(s, node->RightChild);
            // Đưa nút con trái vào stack nếu khác rỗng
            if (node->LeftChild != NULL)
                Push(s, node->LeftChild);
        }
    } // endif
}
```

4. Biên dịch và kiểm tra chương trình

Trong hàm main của lớp Program, sửa đổi mã nguồn như sau:

Trong tập tin program.cpp, sửa đổi mã nguồn như sau:

```
class Program
{
    static void PrintNode(TreeNode node)
    {
        Console.Write(node.Key + "\t");
    }

    static void Main(string[] args)
    {
        BinarySearchTree bst = new BinarySearchTree();
        // Khai báo tập dữ liệu muốn chèn vào cây
        int[] data = { 14, 20, 2, 4, 17, 11, 8, 1, 19 };

        // Xây dựng cây BST từ tập data có 9 phần tử
        bst.BuildTree(data);

        // Xuất các nút của cây theo thứ tự trước
        Action<TreeNode> xuly = new Action<TreeNode>(PrintNode);
        bst.Preorder(bst.Root, xuly);   // Đệ quy
        Console.WriteLine();
        bst.PreOrder(PrintNode);    // Lặp

        // Khai báo một phần tử mới
        int x = 25;
        bool kq = false;

        // Tìm xem trong cây có phần tử X hay chưa
        TreeNode parent = null, node;
        node = bst.SearchLoop(x, ref parent);
```

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

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

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

void main()
{
    BSTree bst;     // Khai báo đối tượng cây BST

    // Khai báo tập dữ liệu muốn chèn vào cây
    DataType data[] = {14,20,2,4,17,11,8,1,19};

    // Xây dựng cây BST từ tập data có 9 phần tử
    BuildTree(bst, data, 9);

    // Xuất các nút của cây theo thứ tự trước
    Preorder(bst, PrintNode);   // Đệ quy
    cout << endl;
    PreOrder(bst, PrintNode);   // Lặp
```

```
        // Nếu không tìm thấy
        if (node == null)
        {
            kq = bst.Insert(ref bst.Root, x);
            if (kq)
            {
                Console.WriteLine();
                Console.WriteLine("Them phan tu {0} thanh cong", x);
                // Xuất lại các nút của cây
                bst.PreOrder(PrintNode);    // Dạng Lặp
            }
        }
        else // Nếu tìm thấy, xuất thông báo
        {
            Console.WriteLine();
            Console.Write("Tim thay phan tu {0}", x);
            if (parent != null)
                Console.WriteLine("\r\nNut cha : " + parent.Key);
            else
                Console.WriteLine(". Chinh la nut goc cua cay.");
        }

        // Hủy một nút trên cây
        x = 17;
        kq = bst.Delete(ref bst.Root, x);       // Đệ quy
        //kq = bst.DeleteLoop(x);               // Lặp

        // Nếu xóa thành công
        if (kq)
        {
            Console.WriteLine("Da xoa thanh cong");
            // Xuất lại các nút của cây
             bst.Preorder(bst.Root, xuly);   // Đệ quy
         }

         Console.ReadKey();
    }
}
```

```
    // Khai báo một phần tử mới
    DataType x = 25;
    bool kq = false;

    // Tìm xem trong cây có phần tử X hay chưa
    NodePointer parent = NULL, node;
    node = SearchLoop(bst, x, parent);

    // Nếu không tìm thấy
    if (node == NULL)
    {
        kq = Insert(bst, x);
        if (kq)
        {
            cout << endl << "Them phan tu " << x
                 << " thanh cong" << endl;
            // Xuất lại các nút của cây
            Preorder(bst, PrintNode);
        }
    }
    else // Nếu tìm thấy, xuất thông báo
    {
        cout << endl << "Tim thay phan tu " << x;
        if (parent)
            cout << endl << "Nut cha : " << parent->Key;
        else
            cout << ". Chinh la nut goc cua cay.";
    }

    // Hủy một nút trên cây
    x = 17;
    kq = Delete(bst, x);
    // Nếu xóa thành công
    if (kq)
    {
        cout << endl << "Da xoa thanh cong" << endl;
        // Xuất lại các nút của cây
        Preorder(bst, PrintNode);
    }

    getch();
}
```

Sau đó, vào menu Build > chọn Build Solution. Sửa lỗi nếu có. Nhấn F5 để chạy chương trình và kiểm tra kết quả.

V. Bài tập thực hành

Bài 1. Viết chương trình thực hiện các chức năng sau:

a. Nhập từ bàn phím các số nguyên và xây dựng một cây tìm kiếm nhị phân (BST).
b. Xây dựng cây BST với dữ liệu lấy từ file.
c. Lưu toàn bộ dữ liệu trên cây xuống file.
d. Hủy toàn bộ cây BST.
e. Xuất các phần tử trên cây BST theo thứ tự đầu (NLR), giữa (LNR), cuối (LRN).
f. Đếm số nút của cây (thỏa một điều kiện nào đó)
g. Tính tổng giá trị các nút trên cây
h. Đếm số nút của cây ở mức K
i. Đếm số nút lá của cây
j. Tính chiều cao của cây
k. Đếm số nút có đúng hai nút con khác rỗng
l. Đếm số nút có đúng một nút con khác rỗng
m. Đảo nhánh trái và nhánh phải của cây.
n. Duyệt cây theo chiều rộng (BFS)
o. Duyệt cây theo chiều sâu (DFS)
p. Kiểm tra xem cây T có phải là cây cân bằng hoàn toàn hay không?
q. Xuất đường đi từ nút gốc đến một nút bất kỳ (giá trị nhập từ bàn phím).
r. Tìm mức của một nút theo giá trị nhập.
s. Kiểm tra một cây T cho trước có phải là cây BST hay không?

Yêu cầu:

- Chương trình phải được tổ chức bằng menu để người dùng chọn chức năng cần thực hiện

Bài 2. Viết chương trình thực hiện các chức năng sau:

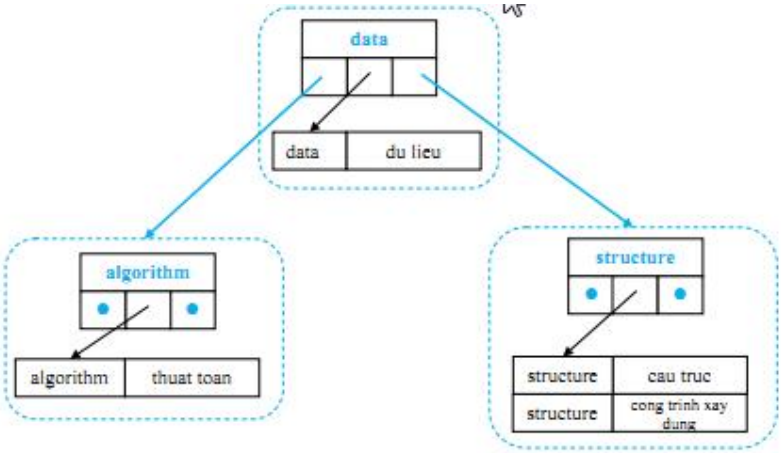
a. Nhập vào một biểu thức số học đơn giản. Biểu diễn biểu thức số học lên cây nhị phân. Kiểm tra cú pháp của biểu thức.
b. Xuất ra biểu thức số học đó dưới dạng: tiền tố, trung tố, hậu tố.
c. Tính giá trị của biểu thức.

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 điển
- 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.

VI. Bài làm thêm

Bài 1. Gia phả (Bài tập cây nhiều nhánh)

Hãy sử dụng cấu trúc dữ liệu thích hợp để lưu trữ gia phả của một dòng họ nào đó trong bộ nhớ của máy tính. Với cấu trúc dữ liệu này, hãy trình bày thuật toán và cài đặt chương trình thực hiện việc kiểm tra hai người có tên X, Y có phải là anh em ruột hay không? Nếu không phải thì ai có vai vế cao hơn? Trong hai trường hợp sau:

a. Mỗi cặp vợ chồng có không quá 2 người con.
b. Mở rộng: Mỗi cặp vợ chồng có không quá 5 người con.

Bài 2. Hệ thống menu (trình đơn) (Bài tập cây nhiều nhánh)

Hãy sử dụng cấu trúc dữ liệu thích hợp để lưu trữ một hệ thống menu có nhiều mục chọn, nhiều cấp trong bộ nhớ máy tính. Với cấu trúc dữ liệu này, hãy trình bày thuật toán và viết chương trình thực hiện việc cho menu xuất hiện trên màn hình và cho phép người sử dụng chọn một chức năng nào đó của menu.

=== HẾT ===