


Trường: ĐH CNTP TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm Môn học: TH Cấu trúc dữ liệu & giải thuật	BÀI 5. CÂY NHỊ PHÂN	
--	--------------------------------------	--

A. MỤC TIÊU:

- Hiểu được cấu trúc dữ liệu động.
- Lập trình và vận dụng được cấu trúc dữ liệu cây nhị phân vào từng bài toán cụ thể.
- Làm được các bài tập áp dụng cây nhị phân.

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV:

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

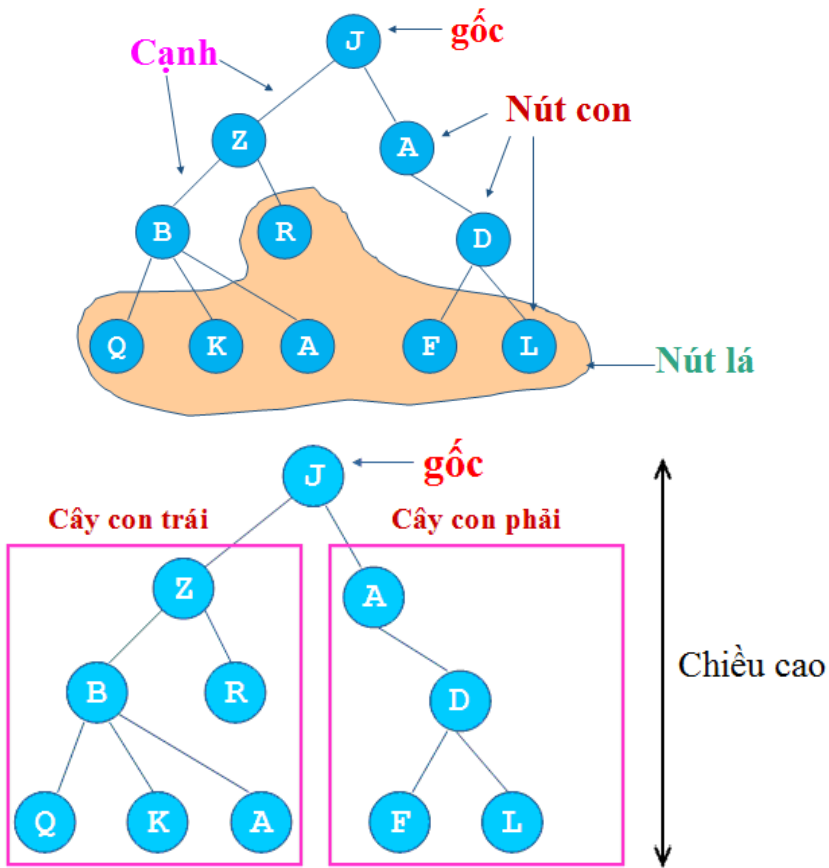
C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Khái niệm cây

Là một tập hợp T các phần tử (gọi là nút của cây) trong đó có 1 nút đặc biệt được gọi là nút gốc, các nút còn lại được chia thành những tập rời nhau T_1, T_2, \dots, T_n theo quan hệ phân cấp trong đó T_i cũng là 1 cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp $i + 1$. Quan hệ này người ta gọi là quan hệ cha-con.

- Cây là tập hợp các nút và cạnh nối các nút đó.
- Có một nút gọi là gốc.
- Các nút còn lại được chia thành những tập rời nhau T_1, T_2, \dots, T_n theo quan hệ phân cấp trong đó T_i cũng là 1 cây.
- Quan hệ one-to-many giữa các nút.
- Có duy nhất một đường đi từ gốc đến nút con.



Thuật ngữ:

- Bậc của một nút: là số cây con của nút đó.
- Bậc của một cây: là bậc lớn nhất của các nút trong cây (số cây con tối đa của một nút trong cây). Cây có bậc n thì gọi là cây n-phân.
- Nút gốc: không có nút cha.
- Nút lá: không có nút con, hay nút có bậc bằng 0.
- Nút nhánh: không phải nút lá và nút gốc, nút có bậc khác 0.
- Các nút có cùng một nút cha gọi là nút anh em (nút đồng cấp).
- Độ dài đường đi từ gốc đến nút x: là số nhánh cần đi qua kể từ gốc đến x.

- Độ dài đường đi của một cây: được định nghĩa là tổng các độ dài đường đi của tất cả các nút của cây.
- Mức của một nút: là độ dài đường đi từ gốc đến nút đó.
- Chiều cao của một nút: là mức của nút đó cộng thêm 1.
- Chiều cao của một cây: là chiều cao lớn nhất của các nút trong cây.
- Rừng là tập hợp các cây. Như vậy, nếu một cây bị loại bỏ nút gốc có thể cho ta một rừng.

2. Cây nhị phân

a. Khái niệm

Cây nhị phân là cây mà mỗi nút có tối đa 2 cây con.

b. Cấu trúc của một nút



Mỗi nút (phần tử) của cây nhị phân ứng với một biến động gồm ba thành phần:

- Thông tin (dữ liệu) lưu trữ tại nút: **Info**.
- Địa chỉ nút gốc của cây con trái trong bộ nhớ: **Left**.
- Địa chỉ nút gốc của cây con phải trong bộ nhớ: **Right**.

3. Các thao tác trên cây nhị phân

a. Khai báo nút

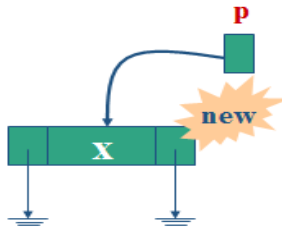
```
typedef int ItemType;
//Định nghĩa kiểu dữ liệu của một phần tử
struct TNode
{
    /* Định nghĩa kiểu dữ liệu cho 1 nút của
       cây nhị phân là Tnode */
    ItemType Info;
    TNode* Left;
    TNode* Right;
};
```

```

struct BTree
{
    /* Định nghĩa kiểu dữ liệu cho Cây NP*/
    TNode* Root;
};

```

b. Tạo nút mới chứa giá trị x



```

TNode* createTNode(ItemType x)
{
    TNode* p = new TNode;
    if(p == NULL)
    {
        printf("Khong du bo nho de cap phat!");
        getch();
        return NULL;
    }
    p->Info = x;
    p->Left = NULL;
    p->Right = NULL;
    return p;
}

```

c. Xuất nội dung của nút

```

void showTNode(TNode* p)
{
    printf("%4d", p->Info);
}

```

d. Hủy nút

```

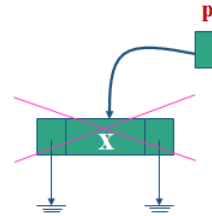
void deleteTNode(TNode* &p)

```

```

{
    if(p == NULL) return;
    delete p;
}

```



e. Khởi tạo cây

```

void initBTree(BTree &bt)
{
    //initialize BTree
    bt.Root = NULL;
}

```



f. Kiểm tra cây rỗng

```

int isEmpty(BTree bt)
{ // Kiểm tra cây có rỗng hay không?
    return (bt.Root == NULL) ? 1 : 0;
}

```

g. Thêm nút p làm nút con bên trái nút T

```

int insertTNodeLeft(TNode* &T, TNode* p)
{
    if(T == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    //Đã tồn tại nút con trái của T
    if(T->Left != NULL) return 0;
    //Gán p làm con trái cho T
    T->Left = p;
    return 1; //Thực hiện thành công
}

```

h. Thêm nút p làm nút con bên phải nút T

```

int insertTNodeRight(TNode* &T, TNode* p)
{
    if(T == NULL || p == NULL)
        return 0; //Thực hiện không thành công
    //Đã tồn tại nút con phải của T

```

```

if(T→Right != NULL) return 0;
//Gán p làm con phải cho T
T→Right = p;
return 1; //Thực hiện thành công
}

```

a. Thêm nút p vào cây

```

int insertTNode(TNode* &root, TNode* p)
{
    if(p == NULL)
        return 0; //Thực hiện không thành công
    if(root == NULL)
    { //Cây rỗng, nên thêm p vào gốc
        root = p;
        return 1; //Thực hiện thành công
    }
    if(root→Left == NULL) //Chưa có con trái
        insertTNode(root→Left, p); //Thêm bên trái
    else if(root→Right == NULL) //Chưa có con phải
        insertTNode(root→Right, p); //Thêm bên phải
    else
    { //p có đủ 2 con trái, phải → Thêm vào cây con
        int x = rand()%2; //Tạo số chẵn hoặc lẻ
        if(x == 0) //Thêm vào nhánh bên trái nếu x=0
            insertTNode(root→Left, p);
        else //Thêm vào nhánh bên phải nếu x=1
            insertTNode(root→Right, p);
    }
    return 1; //Thực hiện thành công
}

```

- b. Duyệt cây theo Node – Left – Right (*traverseNLR*)

```
void traverseNLR(TNode* root)
{
    if(root == NULL) return;
    showTNode(root→Info);
    traverseNLR(root→Left);
    traverseNLR(root→Right);
}
```

- c. Duyệt cây theo Left – Node – Right (*traverseLNR*)

```
void traverseLNR(TNode* root)
{
    ...
}
```

- d. Duyệt cây theo Left – Right – Node (*traverseLRN*)

```
void traverseLRN(TNode* root)
{
    ...
}
```

- e. Tìm kiếm nút chứa giá trị x

```
TNode* findTNode(TNode* root, ItemType x)
{
    if(root == NULL) return NULL;
    if(root→Info == x)
        return root; //Tìm được khóa x. Dừng
    TNode* p = findTNode(root→Left, x);
    if(p != NULL)
        return p; // x có bên nhánh trái. Dừng
    //Sẽ tiếp tục tìm x bên nhánh phải
    return findTNode(root→Right, x);
}
```

f. Kiểm tra nút T có phải là nút lá hay không

```
int isLeaf(TNode* T)
{
    //Trả về: 1 nếu nút là Lá, 0 nếu ngược lại
    ...
}
```

Gợi ý: Nút lá là nút có cả 2 con trỏ Left và Right đồng thời **NULL**.

- Nếu con trỏ T = **NULL** thì hàm trả về 0.
- Nếu con trỏ T->Left = **NULL** và T->Right = **NULL** thì hàm trả về 1. Ngược lại hàm trả về 0.

g. Xóa nút con bên trái của nút T

```
int deleteTNodeLeft(TNode* T, ItemType &x)
{ //Nút con trái của T phải là nút lá.
  //Xóa thành công trả về 1, ngược lại trả về 0.
  ...
}
```

Gợi ý:

- Nếu con trỏ T = **NULL** thì hàm trả về 0.
- Gán p = T->Left.
- Nếu con trỏ p = **NULL** thì hàm trả về 0.
- Nếu con trỏ p->Left \neq **NULL** và p->Right \neq **NULL** thì hàm trả về 0.
- Gán x = T->Info.
- Xóa p.
- Hàm trả về 1.

h. Xóa nút con bên phải của nút T

```
int deleteTNodeRight(TNode* T, ItemType &x)
{
    ...
}
```


Gợi ý:

- Nếu con trỏ T = **NULL** thì hàm trả về 0.
- Gán p = T->Right.
- Nếu con trỏ p = **NULL** thì hàm trả về 0.
- Nếu con trỏ p->Left \neq **NULL** và p->Right \neq **NULL** thì hàm trả về 0.
- Gán x = T->Info.
- Xóa p.
- Hàm trả về 1.

i. Đếm số nút của cây

```
int countTNode(TNode* root)
{    //Ham dem so nut hien co trong cay
    if(!root) return 0;
    //đệ quy trái
    int nl = countTNode(root->Left);
    //đệ quy phải
    int nr = countTNode(root->Right);
    return (1 + nl + nr);
}
```

j. Đếm số nút lá của cây

```
int countTNodeIsLeaf(TNode* root)
{    //Ham dem so nut la hien cua cay
    ...
}
```

Gợi ý: Tương tự hàm **countTNode**

- Nếu root = **NULL** thì hàm trả về 0.
- Nếu root->Left = **NULL** và root->Right = **NULL** thì trả về 1.
- Tính **cnl** = Số lượng nút nhánh con trái.
- Tính **cnr** = Số lượng nút nhánh con phải.
- Hàm trả về **cnl + cnr**.

k. Đếm số nút có đúng 2 nút con của cây.

```
int countTNodeHaveTwoChild(TNode* root)
{
    //Ham dem so nut co du 2 con
    ...
}
```

Gợi ý:

- Nếu root = **NULL** thì hàm trả về 0.
- Nếu root->Left = **NULL** hoặc root->Right = **NULL** thì trả về 0.
- Tính **cnl** = Số lượng nút nhánh con trái.
- Tính **cnr** = Số lượng nút nhánh con phải.
- Hàm trả về **cnl + cnr + 1**.

l. Tính tổng giá trị các nút của cây

```
int sumTNode(TNode* root) //Tinh tong gia tri
cac nut hien co trong cay
{
    ...
}
```

Gợi ý:

- Nếu root = **NULL** thì trả về 0.
- Tính **suml** = Tổng giá trị các nút nhánh con trái.
- Tính **sumr** = Tổng giá trị các nút nhánh con phải.
- Hàm trả về **suml + sumr + root->Info**.

m. Tính chiều cao của cây

```
int highTree(TNode* root)
{ //Ham tinh chieu cao cua cay
    ...
}
```

Gợi ý:

- Nếu root = **NULL** thì trả về 0.

- Tính **hl** = Chiều cao nhánh con trái.
- Tính **hr** = Chiều cao nhánh con phải.
- Nếu **hl** > **hr** thì: Hàm trả về **hl + 1**.
- Ngược lại: Hàm trả về **hr + 1**.

II. Bài tập hướng dẫn mẫu

Bài 1. Viết chương trình quản lý các số nguyên bằng Cây NP?

- **Bước 1:** Tạo một Project mới.
- **Bước 2:** Khai báo thêm các thư viện cơ bản cho chương trình.

```
#include <conio.h>
```

```
#include <stdio.h>
```

- **Bước 3:** Khai báo cấu trúc dữ liệu cho chương trình.

```
//=====
typedef int ItemType; //Định nghĩa kiểu dữ liệu
của một phần tử
```

```
struct TNode
```

```
{ //Định nghĩa kiểu dữ liệu cho 1 nút của cây
nhị phân là TNode
```

```
    ItemType Info;
```

```
    TNode* Left;
```

```
    TNode* Right;
```

```
};
```

```
struct BTree
```

```
{//Định nghĩa kiểu dữ liệu cho cây nhị phân
```

```
    TNode* Root;
```

```
};
```

- **Bước 4:** Viết các hàm cần thiết cho chương trình như sau:

```
//=====
```

```
TNode* createTNode(ItemType x)
```

```
{
```

```
    ...
```

```
}
```

```

//=====
void initBTree(BSTree &bt)
{ //initialize BSTree
    ...
}
//=====
int insertTNodeLeft(TNode* &T, TNode* p)
{
    ...
}
//=====
int insertTNodeRight(TNode* &T, TNode* p)
{
    ...
}
//=====
int insertTNode(TNode* &T, TNode* p)
{
    ...
}
//=====
void createBTree_FromArray(BTree &bt, ItemType
a[], int na)
{ //Hàm tạo cây NP từ mảng a
    ...
}
//=====
void showTNode(TNode* p)
{
    ...
}
//=====

```

```

void traverseNLR(TNode* root)
{ //Hàm duyệt cây theo thứ tự traverseNLR
    ...
}
//=====
TNode* findTNode(TNode* root, ItemType x)
{
    ...
}
//=====
int highTree(TNode* root)
{ //Hàm tính chiều cao của cây
    ...
}
//=====

```

– **Bước 5:** Viết hàm main để thực thi chương trình.

III. Bài tập ở lớp

Bài 1. Cho cây NP chứa các số nguyên (*mỗi nút là 1 số nguyên*) như Bài tập mẫu 1. Hãy hoàn thiện chương trình với những chức năng sau:

- Tạo cây NP bằng 3 cách (**Cách 1:** Cho trước 1 mảng **a** có **n** phần tử, hãy tạo một cây NP có **n** nút, mỗi nút lưu 1 phần tử của mảng. **Cách 2:** Nhập liệu từ bàn phím. **Cách 3:** Tạo ngẫu nhiên tự động).
- Duyệt cây NP bằng 6 cách: traverseNLR, traverseLNR, traverseLRN, traverseNRL, traverseRNL, traverseRLN.
- Thêm 1 nút có giá trị **x** làm con trái của nút có giá trị **y** của cây.
- Thêm 1 nút có giá trị **x** làm con phải của nút có giá trị **y** của cây.
- Đếm số nút trên cây.
- Tìm kiếm 1 nút có giá trị **x** có tồn tại trên cây hay không?

- g. Liệt kê các nút có giá trị có lớn hơn x .
- h. Thực hiện một số thao tác xử lý tính toán trên cây như: Đếm số nút trên cây/ số nút lá/ số nút có 1 con/ số nút có 2 con/..., Tính tổng các nút trên cây/ tổng nút lá/ tổng nút có 1 con/ tổng nút có 2 con/..., tính chiều cao, ...

Bài 2. Cho cây nhị phân mà mỗi nút là 1 phân số. Hãy viết chương trình để thực hiện những chức năng sau:

- a. Tạo cây NP bằng **3 cách** (từ một mảng, nhập liệu từ bàn phím, tạo ngẫu nhiên tự động).
- b. Duyệt cây NP bằng **6 cách**: `traverseNLR`, `traverseLNR`, `traverseLRN`, `traverseNRL`, `traverseRNL`, `traverseRLN`.
- c. Thêm 1 nút là phân số p làm con trái/ phải của nút T .
- d. Đếm số lượng những phân số lớn hơn 1.
- e. Tối giản tất cả các nút (*phân số*) của cây.
- f. Tìm kiếm trên cây có nút nào có giá trị bằng với phân số x hay không?

VI. Bài tập về nhà

Bài 3. Tiếp theo Bài tập 2. Hãy bổ sung thêm những chức năng sau:

- a. Tính tổng các phân số.
- b. Tìm phân số nhỏ nhất/ lớn nhất.
- c. Liệt kê các phân số có tử số lớn hơn mẫu số/ nhỏ hơn mẫu số.
- d. Liệt kê các phân số có tử số và mẫu số đồng thời là số nguyên tố.
- e. Liệt kê các phân số ở mức k (k được nhập từ bàn phím).
- f. Đếm số lượng phân số ở mức k (k được nhập từ bàn phím).
- g. Tính tổng các phân số ở mức k (k được nhập từ bàn phím).

Bài 4. Dựa vào 6 phép duyệt cây, hãy viết các hàm duyệt cây với việc khử đệ quy bằng cách ứng dụng **Stack** (*).

-- HẾT --

