TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BÁO CÁO BÀI TẬP:

CÂY NHỊ PHÂN VÀ CÂY NHỊ PHÂN TÌM KIẾM

SINH VIÊN THỰC HIỆN: Nguyễn Đỗ Huy – 3121411085

LỚP: DCT124C7

GVHD: ĐỖ NHƯ TÀI

Thành phố Hồ Chí Minh , Tháng 3 Năm 2025

MỤC LỤC

CÂY NHỊ PHÂN

Bài 001.

Cho một cây nhị phân có nút gốc là Root, mỗi nút trong cây chứa một số nguyên:

a) Viết chương trình tính trung bình cộng các nút trong cây.

b) Viết chương trình tính trung bình cộng các số dương trong cây.

c) Viết chương trình tính trung bình cộng các số âm trong cây.

d) Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

Bài làm

a) Viết chương trình tính trung bình cộng các nút trong cây.

```
int DemNode(TREE Root)
{
    if(Root==NULL)
        return 0;
    int a=DemNode(Root->pLeft);
    int b=DemNode(Root->pRight);
    return (a+b+1);
}

int TongNode(TREE Root)
{
    if(Root==NULL)
        return 0;
    int a=TongNode(Root->pLeft);
    int b=TongNode(Root->pRight);
    return (a+b+Root->info);
}

float TrungBinhCong(TREE Root)
{
```

```
    int s = TongNode(Root);
    int dem = DemNode(Root);
    if(dem==0)
        return 0;
    return (float)s/dem;
}
```

b) Viết chương trình tính trung bình cộng các số dương trong cây.

```
int DemDuong(TREE Root)
{
    if(Root==NULL)
        return 0;
    int a=DemDuong(Root->pLeft);
    int b=DemDuong(Root->pRight);
    if(Root->info>0)
        return (a+b+1);
    return (a+b);
}

int TongDuong(TREE Root)
{
    if(Root==NULL)
        return 0;
    int a=TongDuong(Root->pLeft);
    int b=TongDuong(Root->pRight);
    if(Root->info>0)
        return (a+b+Root->info);
    return (a+b);
}
```

```
float TrungBinhDuong(TREE Root)
{
    int s = TongDuong(Root);
    int dem=DemDuong(Root);
    if(dem==0)
        return 0;
    return (float)s/dem;
}
```

c) Viết chương trình tính trung bình cộng các số âm trong cây.

```
int DemAm(TREE Root)
{
    if(Root==NULL)
        return 0;
    int a=DemAm(Root->pLeft);
    int b=DemAm(Root->pRight);
    if(Root->info<0)
        return (a+b+1);
    return (a+b);
}

int TongAm(TREE Root)
{
    if(Root==NULL)
        return 0;
    int a=TongAm(Root->pLeft);
    int b=TongAm(Root->pRight);
    if(Root->info<0)
```

```
        return (a+b+Root->info);
    return (a+b);
}

float TrungBinhCongAm(TREE Root)
{
    int s = TongAm(Root);
    int dem = DemAm(Root);
    if(dem==0)
        return 0;
    return (float)s/dem;
}
```

d) Viết chương trình tính tính tỉ số R=a/b. Với a là tổng các nút có giá trị dương, b là tổng các nút có giá trị âm.

```
float TinhTySo(TREE Root)
{
    int a = TongDuong(Root);
    int b = TongAm(Root);
    if(b==0)
        return 0;
    return (float)a/b;
}
```

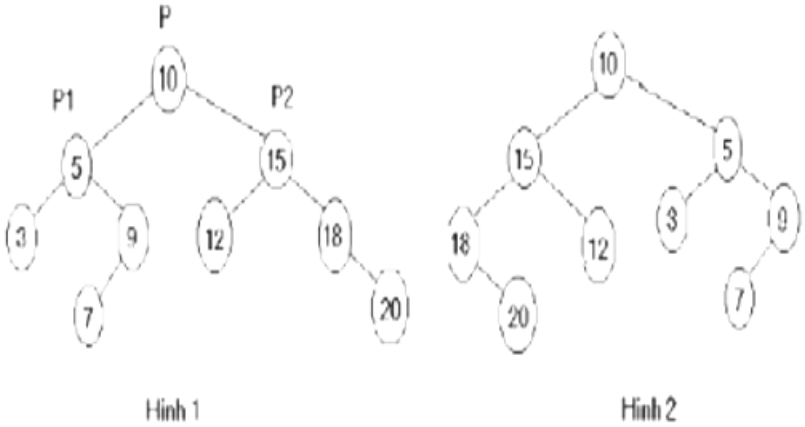
Kết quả

INPUT	OUTPUT
1 -8 2 9 -3 4 7 3 6 -5	So node: 10 Tong gia tri cac node: 16

	Trung binh cong: 1.6 So node duong: 7 Tong node duong: 32 Trung binh duong: 4.57143 So node am: 3 Tong node am: -16 Trung binh am: -5.33333 Ty so tong duong/tong am: -2

Bài 002.

Cho cây như hình 1, cho trước nút p



Hình 1 Hình 2

Hãy viết các câu lệnh cần thiết để chuyển cây sang dạng biểu diễn của hình 2

Bài làm

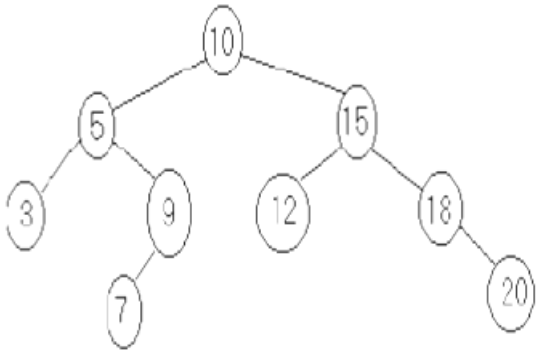
Các câu lệnh cần thiết

1. NODE *temp=p->pLeft;
2. p->pLeft = p->pRight;
3. p->pRight=temp;
4. temp = p->pLeft->pLeft;
5. p->pLeft->pLeft=p->pLeft->pRight;

6. p->pLeft->pRight=temp;

Bài 003.

Cho cây nhị phân tìm kiếm như hình vẽ. Hãy cho biết thứ tự các nút thêm vào cây sao cho để có được cấu trúc này? (Giả sử lúc đầu cây rỗng).



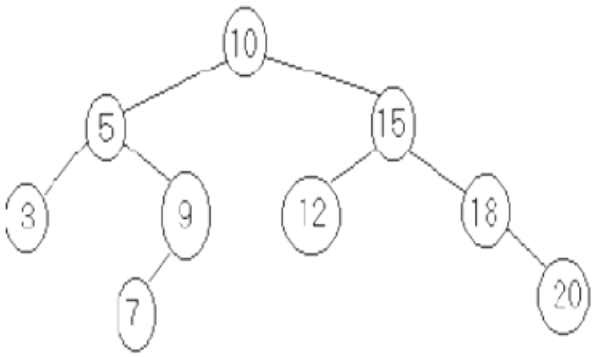
Nếu kết quả của phép duyệt cây trên là: 3, 7, 9, 5, 12, 20, 18, 15, 10. Hãy cho biết người ta đã áp dụng phép duyệt nào?

Bài làm

- Thứ tự các nút thêm vào cây để có được cấu trúc như trên là:
  - Cách 1: 10, 5, 15, 3, 9, 12, 18, 7, 20.
  - Cách 2: 10, 5, 3, 9, 7, 15, 12, 18, 20.
  - Cách 3: 10, 15, 18, 20, 12, 5, 9, 7, 3.
- Đó là phép duyệt cây theo phương pháp: LRN.

Bài 004.

Cho một cây nhị phân tìm kiếm với nút gốc là Root, giá trị lưu trữ tại mỗi nút là một số nguyên (int). Hãy viết hàm tìm phần tử nhỏ nhất và lớn nhất trong cây.



Bài làm

```
struct node
{
    int info;
    struct node*pLeft;
    struct node*pRight;
};
typedef struct node NODE;
typedef NODE*TREE;

NODE* NhoNhat(TREE Root)
{
    if(Root==NULL)
        return NULL;
    NODE*lc = Root;
    while(lc->pLeft)
        lc = lc->pLeft;
    return lc;
}

NODE* LonNhat(TREE Root)
{
    if(Root==NULL)
        return NULL;
    NODE*lc = Root;
    while(lc->pRight)
        lc = lc->pRight;
    return lc;
}
```

Kết quả

INPUT	OUTPUT
10 5 15 3 9 12 18 7 20	Phan tu nho nhat: 3 Phan tu lon nhat: 20

Bài 005.

Cho một cây nhị phân có cấu trúc nút là NODE hãy:

• Viết hàm để tính tổng số nút có một nhánh con (con trái HAY con phải) bằng cách dùng thuật toán duyệt nút gốc giữa NLR;

• Thiết lập một công thức đệ quy để thực hiện yêu cầu của câu trên. Cài đặt công thức này thành hàm.

Bài làm

Số lượng nút một con trong cây nhị phân bằng 1 cộng số lượng nút một con trong cây nhị phân con trái cộng số lượng nút một con trong cây nhị phân con phải nếu nút gốc có một con. Ngược lại số lượng nút một con trong cây nhị phân bằng số lượng nút một con trong cây nhị phân con trái cộng số lượng nút một con trong cây nhị phân con phải.

```
int DemMotCon(TREE t) {
    if (t == NULL)
        return 0;
    if ((t->pLeft && !t->pRight) || (!t->pLeft && t->pRight))
        return 1 + DemMotCon(t->pLeft) + DemMotCon(t->pRight);
    return DemMotCon(t->pLeft) + DemMotCon(t->pRight);
}
```

Kết quả

INPUT	OUTPUT
1 2 4 3 5	So nut co mot nhanh con: 2
0	So nut co mot nhanh con: 0

1 2	So nut co mot nhanh con: 1

Bài 006.

Hãy phát biểu công thức đệ quy để tính các giá trị sau đây:

- Số nút trong cây nhị phân tìm kiếm.
- Tổng giá trị các nút trong cây (giả sử mỗi phần tử là một số nguyên).

Cài đặt thành hàm các công thức đã nêu ở trên.

Bài làm

- Số nút trong cây nhị phân tìm kiếm bằng số nút trong cây nhị phân tìm kiếm trong cây con trái cộng số nút trong cây nhị phân tìm kiếm trong cây con phải cộng 1.
- Tổng giá trị các nút trong cây tìm kiếm bằng tổng giá trị các nút trong cây nhị phân tìm kiếm con trái cộng tổng giá trị các nút trong cây nhị phân tìm kiếm con phải cộng giá trị tại nút gốc.

```
struct node {
    int info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

int DemNode(TREE t) {
    if (t == NULL)
        return 0;
    int a = DemNode(t->pLeft);
    int b = DemNode(t->pRight);
    return (a + b + 1);
}
```

```
int TongNode(TREE t) {
    if (t == NULL)
        return 0;
    int a = TongNode(t->pLeft);
    int b = TongNode(t->pRight);
    return (a + b + t->info);
}
```

Kết quả

INPUT	OUTPUT
10 5 15 3 9 12 18 7 20	So nut trong cay: 9 Tong gia tri cac nut: 99
5	So nut trong cay: 1 Tong gia tri cac nut: 5
10 5 15	So nut trong cay: 3 Tong gia tri cac nut: 30

Bài 007.

Hãy mô tả ngắn gọn sự giống và khác nhau giữa hai cấu trúc Cây Nhị Phân tìm kiếm và Danh Sách Liên Kết Đơn.

Bài làm

Giống nhau

- CTDL động.
- Các thao tác cơ bản Thêm, Xóa, Cập Nhật được thực hiện một cách linh hoạt.

Khác nhau

- Dữ liệu trên cây NPTK được tổ chức và dslk đơn thì không.
- Chi phí tìm kiếm, thêm trên cây nhanh hơn trên dslk đơn.

Bài 008.

Định nghĩa hàm duyệt và xuất cây nhị phân các số thực ra tập tin nhị phân data.out theo phương pháp LNR.

Bài làm

```
struct node {
    float info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

void LNR(TREE t, FILE* fp) {
    if (t == NULL)
        return;
    LNR(t->pLeft, fp);
    fwrite(&t->info, sizeof(float), 1, fp);
    LNR(t->pRight, fp);
}

int Xuat(const char *filename, TREE t) {
    FILE* fp = fopen(filename, "wb");
    if (fp == NULL)
        return 0;
    LNR(t, fp);
    fclose(fp);
    return 1;
}
```

Kết quả

INPUT	OUTPUT
10.5 5.2 15.8 3.1 9.7 12.4 18.9 7.6 20.3	3.1 5.2 9.7 10.5 12.4 15.8 7.6 18.9 20.3
5.5	5.5
10.2 5.1 15.3	5.1 10.2 15.3

Bài 009.

Định nghĩa hàm duyệt và xuất cây nhị phân các số thực ra tập tin nhị phân data.out theo phương pháp NLR.

Bài làm

```
struct node {
    float info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

void NLR(TREE t, FILE* fp) {
    if (t == NULL)
        return;
    fwrite(&t->info, sizeof(float), 1, fp);
    NLR(t->pLeft, fp);
    NLR(t->pRight, fp);
}

int Xuat(char *filename, TREE t) {
    FILE* fp = fopen(filename, "wb");
    if (fp == NULL)
        return 0;
```

```
    NLR(t, fp);
    fclose(fp);
    return 1;
}
```

Kết quả

INPUT	OUTPUT
10.5 5.2 15.8 3.1 9.7 12.4 18.9 7.6 20.3	10.5 5.2 3.1 9.7 15.8 12.4 18.9 7.6 20.3
5.5	5.5
10.2 5.1 15.3	10.2 5.1 15.3

Bài 010.

Định nghĩa hàm duyệt và xuất cây nhị phân các số thực ra tập tin nhị phân data.out theo phương pháp LRN.

Bài làm

```
struct node {
    float info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

void LRN(TREE t, FILE* fp) {
    if (t == NULL)
        return;
    LRN(t->pLeft, fp);
    LRN(t->pRight, fp);
    fwrite(&t->info, sizeof(float), 1, fp);
```

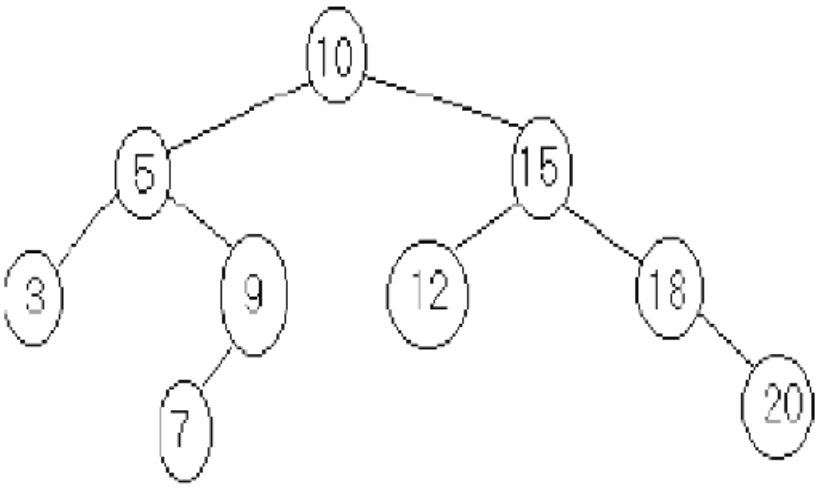
```
}


int Xuat(char *filename, TREE t) {
    FILE* fp = fopen(filename, "wb");
    if (fp == NULL)
        return 0;
    LRN(t, fp);
    fclose(fp);
    return 1;
}
```

Kết quả

INPUT	OUTPUT
10.5 5.2 15.8 3.1 9.7 12.4 18.9 7.6 20.3	3.1 9.7 5.2 12.4 7.6 20.3 18.9 15.8 10.5
5.5	5.5
10.2 5.1 15.3	5.1 15.3 10.2

Bài 011.

Cho cây nhị phân tìm kiếm các số nguyên t. Hãy cho biết cách thức duyệt cây như thế nào để ta được thứ tự các giá trị tăng dần.

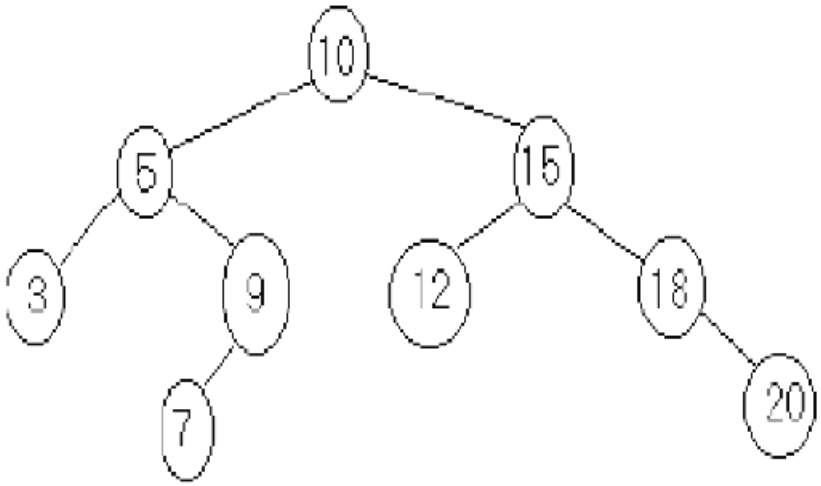


Bài làm

- Duyệt cây theo phương pháp LNR ta sẽ được các giá trị tăng dần.

Bài 012.

Cho cây nhị phân tìm kiếm các số nguyên t. Hãy cho biết cách thức duyệt cây như thế nào để ta được thứ tự các giá trị giảm dần.



Bài làm

- Duyệt cây theo phương pháp RNL ta sẽ được các giá trị giảm dần.

Bài 013.

Định nghĩa hàm lưu cây nhị phân tìm kiếm các số thực xuống tập tin nhị phân sao cho khi đọc dữ liệu từ file ta có thể tạo lại được cây ban đầu.

Bài làm

```
struct node {
    float info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

NODE* CreateNode(int x) {
    NODE* p = new NODE;
```

```
    if (p == NULL)
        return NULL;
    p->info = x;
    p->pLeft = NULL;
    p->pRight = NULL;
    return p;
}

void Insert(TREE &p, float theKey) {
    if (p == NULL) {
        NODE* newNode = CreateNode(theKey);
        p = newNode;
    }
    else if (p->info > theKey) {
        Insert(p->pLeft, theKey);
    }
    else {
        Insert(p->pRight, theKey);
    }
}

void NLR(TREE t, FILE* fp) {
    if (t == NULL)
        return;
    fwrite(&t->info, sizeof(float), 1, fp);
    NLR(t->pLeft, fp);
    NLR(t->pRight, fp);
}

int Xuat(char *filename, TREE t) {
```

```
    FILE* fp = fopen(filename, "wb");
    if (fp == NULL)
        return 0;
    NLR(t, fp);
    fclose(fp);
    return 1;
}

void DocVaTaoBST(TREE &t, char *filename) {
    FILE* fp = fopen(filename, "rb");
    if (fp == NULL) {
        return;
    }

    int x;
    t = NULL;
    while (fread(&x, sizeof(float), 1, fp) == 1) {
        Insert(t, x);
    }
    fclose(fp);
}

void NLR_Print(TREE t) {
    if (t == NULL)
        return;
    cout << t->info << " ";
    NLR_Print(t->pLeft);
    NLR_Print(t->pRight);
}
```

Kết quả

INPUT	OUTPUT
10.5 5.2 15.8 3.1 9.7 12.4 18.9 7.6 20.3	10.5 5.2 3.1 9.7 15.8 12.4 18.9 7.6 20.3
5.5	5.5
10.2 5.1 15.3	10.2 5.1 15.3

Bài 014.

Hãy viết một hàm tạo danh sách liên kết đơn từ cây nhị phân tìm kiếm sao cho giá trị các phần tử trong danh sách có thứ tự giảm dần. Biết nút gốc của cây là Root.

Bài làm

```
struct nodetree {
    int info;
    struct nodetree* pLeft;
    struct nodetree* pRight;
};
typedef struct nodetree NODETREE;
typedef NODETREE* TREE;

struct nodelist {
    int info;
    struct nodelist* pNext;
};
typedef struct nodelist NODELIST;

struct list {
    NODELIST* pHead;
    NODELIST* pTail;
};
typedef struct list LIST;
```

```
void Init(LIST &l) {
    l.pHead = l.pTail = NULL;
}

NODELIST* GetNode(int x) {
    NODELIST* p = new NODELIST;
    if (p == NULL)
        return NULL;
    p->info = x;
    p->pNext = NULL;
    return p;
}

void AddTail(LIST &l, NODELIST* p) {
    if (l.pHead == NULL)
        l.pHead = l.pTail = p;
    else {
        l.pTail->pNext = p;
        l.pTail = p;
    }
}
void RNL(TREE Root, LIST &l) {
    if (Root == NULL)
        return;
    RNL(Root->pRight, l);
    NODELIST* p = GetNode(Root->info);
    if (p != NULL)
        AddTail(l, p);
    RNL(Root->pLeft, l);
}
```

```
void BuildList(TREE Root, LIST &l) {
    Init(l);
    RNL(Root, l);
}
```

Kết quả

INPUT	OUTPUT
10 5 15 3 9 12 18 7 20	20 18 15 12 10 9 7 5 3
5	5
10 5 15	15 10 5

CÂY NHỊ PHÂN TÌM KIẾM

Bài 015.

Giữa cấu trúc cây nhị phân tìm kiếm và cấu trúc mảng các phần tử được sắp thứ tự tăng dần có những điểm giống và khác nhau như thế nào.

Bài làm

Giống nhau

- Dữ liệu được tổ chức.
- Chi phí tìm kiếm một phần tử trên cả hai ctdl là như nhau.

Khác nhau

- Chi phí thêm và xoá phần tử vào mảng lớn hơn chi phí cây nhị phân tìm kiếm.

Bài 016.

Cho một cây nhị phân tìm kiếm t có cấu trúc nút là BST_NODE được khai báo như sau:

```
11.struct BST_NODE{
12.   int Key;      // Khóa của nút
13.   int So_lan; // Số lần xuất hiện.
14.   struct BST_NODE*Left,*Right;
15.};
16.struct BST_TREE
17.{
18.   struct BST_NODE *pRoot;
                  // Nút gốc của cây.
19.};
20.struct BST_TREE t; // Cây t
```

Hãy viết hàm thực hiện thao tác xoá phần tử có khoá X. Cách xoá như sau:

- Nếu phần tử X có tồn tại giảm field So_lan của nó một đơn vị.
- Nếu phần tử X không tồn tại. Thông báo.

Hãy viết hàm in lên màn hình giá trị của các phần tử đang tồn tại trong cây theo thứ tự NLR.

Bài làm

```
struct BST_NODE {
    int Key;
    int So_lan;
    struct BST_NODE* Left;
    struct BST_NODE* Right;
};

struct BST_TREE {
    struct BST_NODE* pRoot;
};
struct BST_TREE t;

int DeleteNode(BST_TREE &t, int x) {
    if (t.pRoot == NULL)
        return 0;
    if (t.pRoot->Key == x) {
        if (t.pRoot->So_lan > 0) {
            t.pRoot->So_lan--;
            return 1;
        }
        return 0;
    }
    if (x < t.pRoot->Key)
        return DeleteNode(*(BST_TREE*)&t.pRoot->Left, x);
    return DeleteNode(*(BST_TREE*)&t.pRoot->Right, x);
}

void XoaGiaTri(BST_TREE &t, int X) {
    int kq = DeleteNode(t, X);
    if (kq == 0)
```

```
        printf("Khong ton tai X hoac So_lan da la 0\n");
    else
        printf("Xoa thanh cong.\n");
}

void NLR(BST_NODE* Root) {
    if (Root == NULL)
        return;
    if (Root->So_lan > 0)
        printf("%4d (So_lan: %d)", Root->Key, Root->So_lan);
    NLR(Root->Left);
    NLR(Root->Right);
}

void LietKe(BST_TREE t) {
    NLR(t.pRoot);
    printf("\n");
}
```

Kết quả

INPUT	Thao tác	OUTPUT
10 5 15 5 10 3 9 15 20	Ban đầu	10(2) 5(2) 3(1) 9(1) 15(2) 20(1)
	XoaGiaTri(t, 5)	10(2) 5(1) 3(1) 9(1) 15(2) 20(1)
	XoaGiaTri(t, 5)	10(2) 3(1) 9(1) 15(2) 20(1)
	XoaGiaTri(t, 7)	10(2) 3(1) 9(1) 15(2) 20(1)

Bài 001.

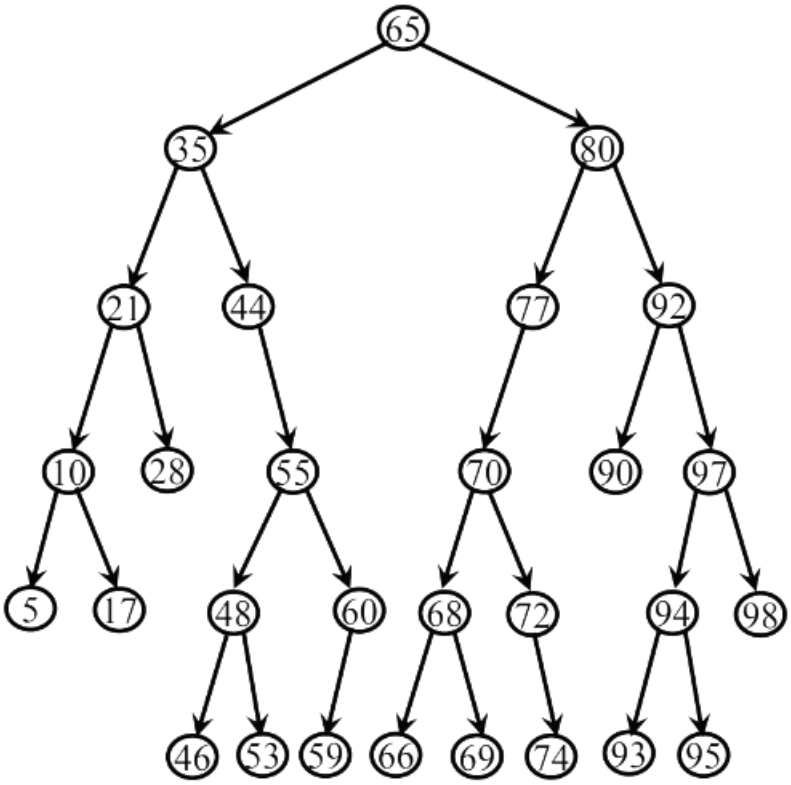
Vấn đề: Định nghĩa hàm thu hồi tất cả các bộ nhớ đã cấp phát cho cây nhị phân tìm kiếm các số nguyên.

Bài làm

```
struct node {
    int info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

void RemoveAll(TREE &t) {
    if (t == NULL)
        return;
    RemoveAll(t->pLeft);
    RemoveAll(t->pRight);
    delete t;
    t = NULL;
}
```

Bài 017.

Hãy viết hàm tìm phần tử thay thế trong thao tác "Xoá một phần tử P có 2 con trong cây BST", sử dụng nguyên tắc "tìm phần tử tận cùng bên trái của nhánh phải P".



Bài làm

```
void SearchStandFor(TREE &p, TREE &q) {
    if (q->pLeft) {
        SearchStandFor(p, q->pLeft);
    }
    else {
        p->info = q->info;
        TREE temp = q;
        q = q->pRight;
        delete temp;
    }
}
```

Kết quả

INPUT	Thao tác	OUTPUT
10 5 15 3 9 12 18 7 20	Delete(t, 15)	10 5 3 9 7 18 12 20

	Delete(t, 5)	10 5 3 9 7 18 12 20
10 5 15	Delete(t, 10)	15 5

Bài 018.

Cho một mảng a gồm n phần tử kiểu số nguyên int. Ta có thể sắp xếp mảng a bằng cách:

- Từ mảng a, tạo một cây nhị phân tìm kiếm T.
- Duyệt cây T và đưa các nút trở lại mảng a.

Yêu cầu

a) Cho biết phương pháp duyệt cây T để đưa các nút lên mảng sao cho mảng được sắp tăng dần.

b) Cho biết cấu trúc cây T.

c) Xây dựng hàm

- Tạo cnp T từ mảng a.
- Duyệt cây để đưa các phần tử trở lại mảng sao cho mảng được sắp thứ tự tăng dần.

Bài làm

- Câu a: Phương pháp duyệt là LNR.
- Câu b: Cấu trúc dữ liệu:

```
struct node {
    int info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;
```

- Câu c:

```
int TaoCay(TREE &t, int a[], int n) {
    Init(t);
    for (int i = 0; i < n; i++) {
```

```
        if (InsertNode(t, a[i]) == -1)
            return 0;
    }
    return 1;
}

void Init(TREE &t) {
    t = NULL;
}

NODE* GetNode(int x) {
    NODE* p = new NODE;
    if (p == NULL)
        return NULL;
    p->info = x;
    p->pLeft = p->pRight = NULL;
    return p;
}

int InsertNode(TREE &t, int x) {
    if (t) {
        if (t->info == x)
            return 0;
        if (t->info < x)
            return InsertNode(t->pRight, x);
        return InsertNode(t->pLeft, x);
    }
    t = GetNode(x);
    if (t == NULL)
        return -1;
```

```
    return 1;
}


void LNR(TREE t, int a[], int &n) {
    if (t == NULL)
        return;
    LNR(t->pLeft, a, n);
    a[n] = t->info;
    n++;
    LNR(t->pRight, a, n);
}
```

Kết quả

INPUT	OUTPUT
10 5 15 3 9 12 18 7 20	Mang truoc khi sap xep: 10 5 15 3 9 12 18 7 20 Cay sau khi tao: 10 5 3 9 7 15 12 18 20 Mang sau khi sap xep tang dan: 3 5 7 9 10 12 15 18 20

Bài 033.

• Hãy cài đặt thuật toán duyệt cây nhị phân (NLR) không dùng đệ quy (dùng stack).

• Hãy cài đặt thuật toán duyệt cây nhị phân theo mức không dùng đệ quy (dùng queue).

Bài làm

- Duyệt cây nhị phân (NLR) dùng Stack:

Khởi tạo stack rỗng và push node gốc vào, lặp khi stack không rỗng:

- Pop node đầu stack, in giá trị (N).
- Push nhánh phải trước (R).

- Push nhánh trái sau (L).

Do stack là LIFO (Last In First Out), nhánh trái sẽ được xử lý trước nhánh phải, đúng thứ tự NLR.

```
void PreOrderNonRecursive(TREE t) {
    if (t == NULL) return;
    Stack s;
    InitStack(s);
    Push(s, t);

    cout << "Duyet NLR bang Stack: ";
    while (!IsEmptyStack(s)) {
        TREE p = Pop(s);
        cout << p->info << " ";

        // Push nhánh phải trước (vì stack LIFO, nhánh trái sẽ được xử lý trước)
        if (p->pRight) Push(s, p->pRight);
        if (p->pLeft) Push(s, p->pLeft);
    }
    cout << endl;
    ClearStack(s);
}
```

- Duyệt cây nhị phân theo mức dùng Queue:

Khởi tạo queue rỗng và enqueue node gốc, lặp khi queue không rỗng:

- Dequeue node đầu queue, in giá trị.
- Enqueue nhánh trái (nếu có).
- Enqueue nhánh phải (nếu có).

Do queue là FIFO (First In First Out), các node ở cùng mức được xử lý trước khi sang mức tiếp theo.

```
void LevelOrderNonRecursive(TREE t) {
```

```
    if (t == NULL) return;
    Queue q;
    InitQueue(q);
    Enqueue(q, t);

    cout << "Duyet theo muc dung Queue: ";
    while (!IsEmptyQueue(q)) {
        TREE p = Dequeue(q);
        cout << p->info << " ";

        // Thêm nhánh trái và phải vào queue (theo thứ tự từ trái sang phải)
        if (p->pLeft) Enqueue(q, p->pLeft);
        if (p->pRight) Enqueue(q, p->pRight);
    }
    cout << endl;
    ClearQueue(q);
}
```

Kết quả

INPUT	OUTPUT
10 5 15 3 9 12 18 7 20	Duyet NLR bang Stack: 10 5 3 9 7 15 12 18 20 Duyet theo muc bang Queue: 10 5 15 3 9 12 18 7 20

Bài 037.

Cho một cây nhị phân có nút gốc là Root. Hãy viết hàm kiểm tra xem cây này có phải là cây cân bằng không? (Giả sử đã có hàm tính chiều cao của nút p như sau:

int ChieuCao(NODE*p)

và thông tin tại mỗi nút trong cây là số nguyên).

Bài làm

```
struct node {
    int info;
    struct node* pLeft;
    struct node* pRight;
};
typedef struct node NODE;
typedef NODE* TREE;

int ChieuCao(TREE t) {
    if (t == NULL)
        return 0;
    int a = ChieuCao(t->pLeft);
    int b = ChieuCao(t->pRight);
    if (a > b)
        return a + 1;
    return b + 1;
}

NODE* LonNhat(TREE t) {
    if (t == NULL)
        return NULL;
    NODE* lc = t;
    NODE* a = LonNhat(t->pLeft);
    if (a && a->info > lc->info)
        lc = a;
    NODE* b = LonNhat(t->pRight);
    if (b && b->info > lc->info)
        lc = b;
    return lc;
```

```
}

NODE* NhoNhat(TREE t) {
    if (t == NULL)
        return NULL;
    NODE* lc = t;
    NODE* a = NhoNhat(t->pLeft);
    if (a && a->info < lc->info)
        lc = a;
    NODE* b = NhoNhat(t->pRight);
    if (b && b->info < lc->info)
        lc = b;
    return lc;
}

int ktCanBang(TREE Root) {
    if (Root == NULL)
        return 1;
    if (ktCanBang(Root->pLeft) == 0)
        return 0;
    if (ktCanBang(Root->pRight) == 0)
        return 0;
    NODE* a = LonNhat(Root->pLeft);
    if (a && a->info > Root->info)
        return 0;
    a = NhoNhat(Root->pRight);
    if (a && a->info < Root->info)
        return 0;
    int x = ChieuCao(Root->pLeft);
    int y = ChieuCao(Root->pRight);
```

```
    if (abs(x - y) > 1)
        return 0;
    return 1;
}
```

Kết quả

INPUT	OUTPUT
10 5 15 3 9 12 18	Cay: 10 5 3 9 15 12 18 Cay can bang
10 5 3 2	Cay: 10 5 3 2 Khong can bang