


Trường: ĐH CNTT TP.HCM Khoa: Công nghệ thông tin Bộ môn: Công nghệ phần mềm. Môn học: TH Cấu trúc dữ liệu & giải thuật	BÀI 7. CÂY ĐỎ ĐEN	
---	------------------------------------	--

A. MỤC TIÊU:

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

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	

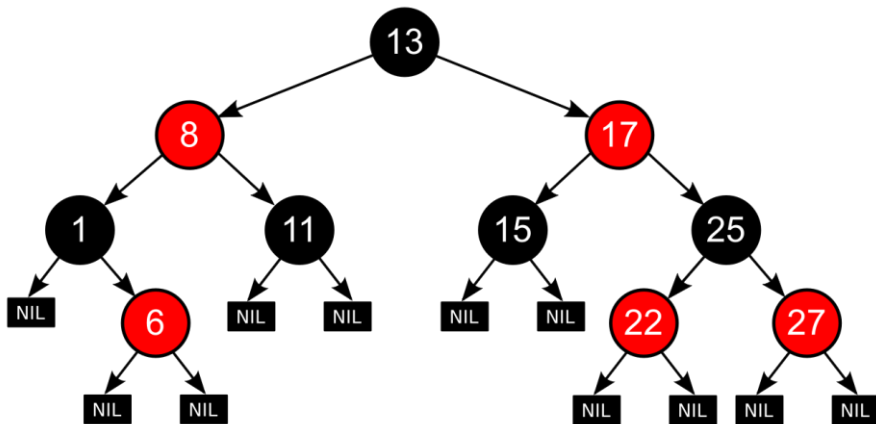
C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Khái niệm cây đỏ đen

Cây đỏ-đen là một nhị phân tìm kiếm (BST) cân bằng thỏa các điều kiện sau:

- Mọi node đều được tô màu đỏ hoặc màu đen.
- Node gốc luôn đen.
- Node lá (NULL) luôn đen.
- Cả hai con của mọi nút đỏ là đen (và suy ra mọi nút đỏ có nút cha là đen).
- Tất cả các đường đi từ một nút bất kỳ tới các lá có số nút đen bằng nhau. Số các nút đen trên một đường đi từ gốc tới mỗi lá được gọi là độ dài đen của đường đi đó.



2. Cấu trúc của một nút



Mỗi nút của cây nhị phân tìm kiếm (Cây NPTK) ứng với một biến động gồm năm thành phần:

- Thông tin (dữ liệu) lưu trữ tại nút: **Info**.
- Màu của nút là Đỏ hay Đen: **Color**
- Địa chỉ nút gốc của cây con trái trong bộ nhớ: **Left**.
- Địa chỉ nút gốc của cây con phải trong bộ nhớ: **Right**.
- Địa chỉ của nút cha trong bộ nhớ: **Parent**.

3. Các thao tác trên cây đỏ đen

a. Khai báo nút

```
/* Khai báo kiểu dữ liệu của Node */
typedef int ItemType;
/* Khai báo thuộc tính color */
enum EColor { RED, BLACK };
/* Khai báo cấu trúc node */
struct BRTNode
{
    ItemType Info;
    EColor Color;
    BRTNode *Left, *Right, *Parent;
```

```
};  
/* Khai báo cấu trúc cây Đỏ-Đen */  
struct BRTree  
{  
    BRTNode *Root;  
};
```

b. Tạo nút mới chứa giá trị x

```
BRTNode* createBRTNode(ItemType x) {  
    BRTNode* p = new BRTNode;  
    if(p == NULL)  
    {  
        printf("\nKhong the cap phat nut moi!");  
        getch();  
        return NULL;  
    }  
    p->Info = x;        // Gán dữ liệu mới cho nút  
    p->Color = RED;     // Gán màu đỏ (Red) mặc định  
    p->Left = NULL;     // Chưa có nút con trái  
    p->Right = NULL;    // Chưa có nút con phải  
    p->Parent = NULL;   // Chưa có nút cha  
    return p;  
}
```

c. Xuất nội dung của nút

```
void setColor(int colorBackground, int colorText) {  
    HANDLE hColor = GetStdHandle(STD_OUTPUT_HANDLE);  
    SetConsoleTextAttribute(hColor, colorBackground  
* 16 + colorText);  
    /* 0 = Black      8 = Gray  
       1 = Blue       9 = Light Blue  
       2 = Green      A = Light Green  
       3 = Aqua       B = Light Aqua  
       4 = Red        C = Light Red  
       5 = Purple     D = Light Purple  
       6 = Yellow     E = Light Yellow  
       7 = White      F = Bright White
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
    => set_Color(X);  
    → X = a*16 + b, a (background) và b (character)  
    */  
}  
void showBRTNode(BRTNode *p) {  
    if(p->Color == RED)  
        //Light Red=12 (C), Bright White=15 (F)  
        setColor(15, 12);  
    else if(p->Color == BLACK)  
        //Black=0, Bright White=15 (F)  
        setColor(15, 0);  
    cprintf("%4d", p->Info);  
    //Light Yellow=14 (E), Green=2  
    setColor(14, 2);  
}
```

d. Khởi tạo cây rỗng

```
/* Initalize BRTree */  
void initBRTree(BRTree &brt) {  
    brt.Root = NULL;  
}
```

e. Hoán vị màu của 2 nút

```
/* Hoán vị màu (Color) */  
void swapColors(IColor &color1, IColor &color2) {  
    IColor temp = color1;  
    color1 = color2;  
    color2 = temp;  
}
```

f. Hoán vị thông tin của 2 nút

```
/* Hoán vị màu (Color) */  
void swapInfos(Itemtype &info1, Itemtype &info2) {  
    Itemtype temp = info1;  
    info1 = info2;  
    info2 = temp;  
}
```

g. Duyệt cây theo chiều rộng (Breadth-first search)

```
/* Duyệt theo chiều rộng */
void levelOrder(BRTNode *root) {
    if(root == NULL) return;

    queue <BRTNode *> q;
    q.push(root);

    while (!q.empty()) {
        BRTNode *p = q.front();
        q.pop();
        showBRTNode(p);
        if(p->Left != NULL)
            q.push(p->Left);
        if(p->Right != NULL)
            q.push(p->Right);
    }
}

/* In theo thứ tự cấp */
void showLevelOrder(BRTNode *root) {
    printf("\nIn theo thu tu cap (Breadth-first
traverse): \n");
    if(root == NULL)
        printf("\nCay rong!");
    else
        levelOrder(root);
}
```

h. Thêm nút vào cây đồ đen

```
/* insert BRTNode */
BRTNode* BRTInsert(BRTNode* root, BRTNode *pNew) {
    /* Nếu cây trống thì trả về một BRTNode mới */
    if(root == NULL) return pNew;

    /* Nếu không thì tiếp tục duyệt xuống dưới cây */
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
if(pNew->Info < root->Info)
{
    root->Left = BRTInsert(root->Left, pNew);
    root->Left->Parent = root;
}
else if(pNew->Info > root->Info)
{
    root->Right = BRTInsert(root->Right, pNew);
    root->Right->Parent = root;
}
/* Trả về con trỏ BRTNode */
return root;
}
```

i. Xoay trái một nút

```
/* Thuật toán xoay trái */
void rotateLeft(BRTNode *&root, BRTNode *&p) {
    BRTNode *pRight = p->Right;

    p->Right = pRight->Left;

    if(p->Right != NULL)
        p->Right->Parent = p;

    pRight->Parent = p->Parent;

    if(p->Parent == NULL)
        root = pRight;
    else if(p == p->Parent->Left)
        p->Parent->Left = pRight;
    else
        p->Parent->Right = pRight;

    pRight->Left = p;
    p->Parent = pRight;
}
```

j. Xoay phải một nút

```
/* Thuật toán xoay phải */
void rotateRight(BRTNode *&root, BRTNode *&p) {
    BRTNode *pLeft = p->Left;

    p->Left = pLeft->Right;

    if(p->Left != NULL)
        p->Left->Parent = p;

    pLeft->Parent = p->Parent;

    if(p->Parent == NULL)
        root = pLeft;
    else if(p == p->Parent->Left)
        p->Parent->Left = pLeft;
    else
        p->Parent->Right = pLeft;

    pLeft->Right = p;
    p->Parent = pLeft;
}
```

k. Cấu trúc lại sự cân bằng khi nút bị vi phạm

```
/* Sửa lại cấu trúc khi chèn BRTNode vào hoặc xóa node */
void fixViolation(BRTNode *&root, BRTNode *&p) {
    BRTNode *pParent = NULL;
    BRTNode *pGrandParent = NULL;

    while ((p != root) && (p->Color != BLACK) &&
        (p->Parent->Color == RED))
    {
        pParent = p->Parent;
        pGrandParent = p->Parent->Parent;
    }
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
/* Trường hợp A: node cha của p là con trái
    của node cha của p */
if(pParent == pGrandParent->Left)
{
    BRTNode *pUncle = pGrandParent->Right;

    /* Trường hợp: 1
    node chú của p là node đỏ khi này
    chỉ cần đổi màu cho node đó thành đen */
    if(pUncle!=NULL && pUncle->Color==RED)
    {
        pGrandParent->Color = RED;
        pParent->Color = BLACK;
        pUncle->Color = BLACK;
        p = pGrandParent;
    }
    else
    {
        /* Trường hợp: 2
        p là node con phải của node cha nó
        ta thực hiện xoay trái */
        if(p == pParent->Right)
        {
            rotateLeft(root, pParent);
            p = pParent;
            pParent = p->Parent;
        }
        /* Trường hợp: 3
        p là con trái của node cha nó
        -> nên thực hiện xoay phải */
        rotateRight(root, pGrandParent);
        swapColors(pParent->Color,
                    pGrandParent->Color);
        p = pParent;
    }
}
```


Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
}
/* Trường hợp: B
node cha của p là con phải của node cha của p */
else
{
    BRTNode *pUncle = pGrandParent->Left;
    /* Trường hợp: 1
    node chú của p là node đỏ khi này
    chỉ cần đổi màu cho node đó thành đen */
    if(pUncle != NULL && pUncle->Color==RED)
    {
        pGrandParent->Color = RED;
        pParent->Color = BLACK;
        pUncle->Color = BLACK;
        p = pGrandParent;
    }
    else
    {
        /* Trường hợp: 2
        p là con trái của node cha nó
        -> nên thực hiện xoay phải */
        if(p == pParent->Left)
        {
            rotateRight(root, pParent);
            p = pParent;
            pParent = p->Parent;
        }
        /* Trường hợp: 3
        p là node con phải của node cha nó
        -> nên thực hiện xoay trái */
        rotateLeft(root, pGrandParent);
        swapColors(pParent->Color,
                    pGrandParent->Color);
        p = pParent;
    }
}
```

```
    }  
  }  
  root->Color = BLACK;  
}
```

l. Thêm một nút mới với dữ liệu đã cho

```
/* Chèn một node mới với dữ liệu đã cho */  
void insert(BRTNode *&root, ItemType x) {  
    BRTNode *pNew = createBRTNode(x);  
    /* Thực hiện chèn như bình thường */  
    root = BRTInsert(root, pNew);  
  
    /* Sửa lại lỗi của quy tắc cây đỏ đen */  
    fixViolation(root, pNew);  
}
```

m. Kiểm tra nút con trái của nút cha

```
/* Kiểm tra xem node hiện tại có phải là node con  
trái của node cha không */  
bool isOnLeft(BRTNode *p) {  
    return p == p->Parent->Left;  
}
```

n. Tìm nút chú (cậu) của một nút

```
/* Trả về con trở tới node chú (Uncle) */  
BRTNode *findUncle(BRTNode *p) {  
    /* Nếu không có node cha hoặc node ông,  
    thì không có node chú */  
    if(p->Parent == NULL ||  
        p->Parent->Parent == NULL)  
        return NULL;  
  
    if(isOnLeft(p->Parent))  
        // node chú bên phải  
        return p->Parent->Parent->Right;  
    else  
        // node chú bên trái  
        return p->Parent->Parent->Left;
```

}

o. Tìm nút anh em của một nút

```
/* Trả về con trỏ cho node anh em */  
BRTNode *findSibling(BRTNode *p) {  
    // node anh rỗng nếu không tồn tại node cha  
    if(p->Parent == NULL)  
        return NULL;  
  
    if(isOnLeft(p))  
        return p->Parent->Right;  
    else  
        return p->Parent->Left;  
}
```

p. Tìm nút không có nút con bên trái

```
/* Tìm nút không có nút con bên trái trong cây con  
của nút đã cho */  
BRTNode *findSuccessor(BRTNode *p) {  
    BRTNode *temp = p;  
    while (temp->Left != NULL)  
        temp = temp->Left;  
    return temp;  
}
```

q. Kiểm tra nút hiện tại có nút con là nút đỏ hay không?

```
/* Kiểm tra node hiện tại có node con là nút đỏ hay  
không */  
bool hasRedChild(BRTNode *p) {  
    return (p->Left!=NULL && p->Left->Color==RED) ||  
           (p->Right!=NULL && p->Right->Color==RED);  
}
```

r. Tìm nút thế mạng cho nút bị xóa

```
/* Tìm nút thay thế nút đã xóa trong BSTree */  
BRTNode *BSTReplace(BRTNode *p) {  
    // Khi nút có 2 con  
    if(p->Left != NULL && p->Right != NULL)  
        return findSuccessor(p->Right);  
}
```

```
// Khi node lá
if(p->Left == NULL && p->Right == NULL)
    return NULL;

// Khi node có một con
if(p->Left != NULL)
    return p->Left;
else
    return p->Right;
}
```

s. Xóa một nút bất kỳ

```
/* Xóa nút đã cho */
void deleteNode(BRTNode *&root, BRTNode *pDelete) {
    BRTNode *pReplace = BSTReplace(pDelete);

    // Đúng khi pReplace và pDelete đều đen
    bool flagDoubleBlack =
        ((pReplace == NULL || pReplace->Color == BLACK)
        && (pDelete->Color == BLACK));
    BRTNode *pParent = pDelete->Parent;

    if(pReplace == NULL) {
        // pReplace là NULL do đó pDelete là lá
        if(pDelete == root) {
            // pDelete là root, làm cho root là NULL
            root = NULL;
        }
    }
    else {
        if(flagDoubleBlack) {
            // pReplace và pDelete đều đen
            // pDelete là lá, sửa màu đen kép
            fixDoubleBlack(root, pDelete);
        }
        else {

```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
// pReplace hoặc pDelete là đỏ
if(findSibling(pDelete) != NULL)
    // node anh chị em không rỗng,
    // làm cho nó màu đỏ
    findSibling(pDelete)->Color = RED;
}
// Xóa pDelete khỏi cây
if(isOnLeft(pDelete)) {
    pParent->Left = NULL;
}
else {
    pParent->Right = NULL;
}
}
delete pDelete;
return;
}
if(pDelete->Left==NULL || pDelete->Right==NULL)
{
    // pDelete có 1 node con
    if(pDelete == root) {
        // Gán giá trị của pReplace cho pDelete
        // và xóa pReplace
        pDelete->Info = pReplace->Info;
        pDelete->Left = pDelete->Right = NULL;
        delete pReplace;
    }
    else {
        // Tách node pDelete khỏi cây và
        // di chuyển node pReplace lên
        if(isOnLeft(pDelete)) {
            pParent->Left = pReplace;
        }
        else {
            pParent->Right = pReplace;
        }
    }
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
    }
    delete pDelete;
    pReplace->Parent = pParent;
    if(flagDoubleBlack) {
        // pReplace và pDelete đều đen,
        // sửa hai màu đen ở pReplace
        fixDoubleBlack(root, pReplace);
    }
    else {
        // pReplace hoặc pDelete đỏ,
        // màu pReplace đen
        pReplace->Color = BLACK;
    }
}
return;
}

// pDelete có 2 con, hoán đổi giá trị với
// nút thế mạng và đệ quy
swapInfos(pReplace->Info, pDelete->Info);
deleteNode(root, pReplace);
}

void fixDoubleBlack(BRTNode *&root, BRTNode *p) {
    // p là node gốc thì return
    if(p == root) return;

    BRTNode *pSibling = findSibling(p);
    BRTNode *pParent = p->Parent;
    if(pSibling == NULL) {
        // Không có sibling,
        // màu đen kép được đẩy lên
        fixDoubleBlack(root, pParent);
    }
    else {
        if(pSibling->Color == RED) {
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
// Anh em màu đỏ
pParent->Color = RED;
pSibling->Color = BLACK;
if(isOnLeft(pSibling)) {
    // Trường hợp left
    rotateRight(root, pParent);
}
else {
    // Trường hợp right
    rotateLeft(root, pParent);
}
fixDoubleBlack(root, p);
}
else {
    // Anh em đen
    if(hasRedChild(pSibling)) {
        // Ít nhất 1 trẻ em màu đỏ
        if(pSibling->Left != NULL &&
            Sibling->Left->Color == RED) {
            if(isOnLeft(pSibling)) {
                // left - left
                pSibling->Left->Color =
                    pSibling->Color;
                pSibling->Color =
                    pParent->Color;
                rotateRight(root, pParent);
            }
            else {
                // right - left
                pSibling->Left->Color =
                    pParent->Color;
                rotateRight(root, pSibling);
                rotateLeft(root, pParent);
            }
        }
    }
}
```

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
        else {
            if(isOnLeft(pSibling)) {
                // left - right
                pSibling->Right->Color =
                    Parent->Color;
                rotateLeft(root, pSibling);
                rotateRight(root, pParent);
            }
            else {
                // right - right
                pSibling->Right->Color =
                    pSibling->Color;
                pSibling->Color =
                    pParent->Color;
                rotateLeft(root, pParent);
            }
        }
        pParent->Color = BLACK;
    }
    else {
        // Hai con đen
        pSibling->Color = RED;
        if(pParent->Color == BLACK)
            fixDoubleBlack(root, pParent);
        else
            pParent->Color = BLACK;
    }
}
}
```

t. Tìm kiếm một nút có chứa giá trị đã cho

```
/* Tìm kiếm một giá trị trên cây */
BRTNode *search(BRTNode *root, ItemType x) {
    BRTNode *temp = root;
    while (temp != NULL) {
```


Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

```
if(x == temp->Info) {
    return temp;
}
else if(x < temp->Info) {
    temp = temp->Left;
}
else {
    temp = temp->Right;
}
}
return NULL; //Không tìm thấy x trong cây
}
```

II. Bài tập hướng dẫn mẫu

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

- **Bước 1:** Tạo một Project mới.
- **Bước 2:** Add vào project 2 file sau: **BRTree.h**, **BRTree.cpp**
- **Bước 3:** Chạy thử chương trình và kiểm tra kết quả thực hiện.

III. Bài tập ở lớp

Bài 1. Dựa vào Bài tập mẫu 1. Hãy hoàn thiện chương trình với những chức năng sau:

- Bổ sung dạng menu cho chương trình.
- Tạo lại cây từ 1 mảng **a** có **n** phần tử ($n > 50$) số nguyên bất kỳ.
- Tạo lại cây bằng cách đọc dữ liệu từ 1 file text.
- Duyệt cây bằng **6 cách**: traverseNLR, traverseLNR, traverseLRN, traverseNRL, traverseRNL, traverseRLN.
- Duyệt cây theo chiều rộng tương ứng với 2 phép duyệt cây NLR, NRL.
- Duyệt cây theo chiều sâu tương ứng với 6 phép duyệt cây NLR, NRL, LNR, RNL, LRN và RLN.
- Thêm 1 nút có giá trị **x** bất kỳ vào cây.
- Xóa 1 nút có giá trị **x** bất kỳ của cây.

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

- i. Tìm kiếm một giá trị **x** có trên cây hay không?
- j. Đếm số nút đỏ, đen trên cây.
- k. Đếm số nút đỏ, đen ở mức **k** của cây.
- l. Tính tổng giá trị các nút đỏ, đen trên cây.

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

- a. Tạo menu của chương trình.
- b. Tạo cây từ một mảng.
- c. Duyệt cây bằng 6 cách: `traverseNLR`, `traverseLNR`, `traverseLRN`, `traverseNRL`, `traverseRNL`, `traverseRLN`.
- d. Duyệt cây theo chiều rộng tương ứng với 2 phép duyệt cây `NLR`, `NRL`.
- e. Duyệt cây theo chiều sâu tương ứng với 6 phép duyệt cây `NLR`, `NRL`, `LNR`, `RNL`, `LRN` và `RLN`.
- f. Thêm 1 nút là phân số **p** bất kỳ vào cây.
- g. Tìm kiếm 1 phân số **p** có trên cây hay không?
- h. Xóa một phân số **p** trên cây.
- i. Xóa những phân số có giá trị > 2 .
- j. Xóa những phân số có mẫu số là số nguyên tố.
- k. Tính tổng các phân số.
- l. Đếm số lượng phân số ở mức **k** (*k được nhập từ bàn phím*).
- m. Xóa toàn bộ các nút của cây.

VI. Bài tập về nhà

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

- a. Liệt kê các phân số có tử số nhỏ hơn mẫu số.
- b. Liệt kê các phân số có mẫu số là một số chính phương.
- c. Liệt kê các phân số có tử số và mẫu số đồng thời là số hoàn thiện.

Bài 4. Cây đồ đen lưu trữ dữ liệu là một từ điển Anh-Việt (Mỗi nút của cây có dữ liệu gồm 2 trường: **word** là khóa chứa một từ tiếng anh,

Bài giảng: Thực hành Cấu trúc dữ liệu và Giải thuật

mean là nghĩa tiếng Việt). Hãy xây dựng cây đồ đen với những chức năng sau:

- a. Xây dựng chương trình dạng menu.
- b. Tạo cây từ 1 file text lưu từ điển Anh-Việt.
- c. Duyệt cây để xem nội dung theo chiều rộng và chiều sâu.
- d. Tra cứu nghĩa của 1 từ bất kỳ.
- e. Thêm một từ bất kỳ vào cây, duyệt lại cây để xem kết quả.
- f. Xóa một từ bất kỳ khỏi cây, duyệt lại cây để xem kết quả.
- g. Bổ sung hay chỉnh sửa nghĩa của 1 từ bất kỳ.
- h. Xóa toàn bộ cây.

-- HẾT --