

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 6. CÂY NHỊ PHÂN TÌM KIẾM	
---	---	--

A. MỤC TIÊU:

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

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	

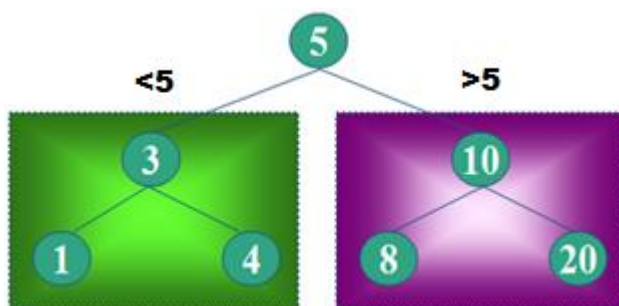
C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Khái niệm cây nhị phân tìm kiếm

Cây nhị phân tìm kiếm là cây nhị phân mà mỗi nút phải thỏa điều kiện sau:

- Giá trị của tất cả nút con trái < nút gốc.
- Giá trị của tất cả nút con phải > nút gốc.



2. Cấu trúc của một nút



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

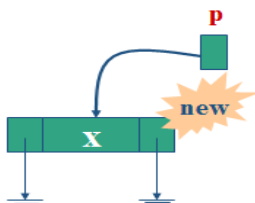
- Thông tin (dữ liệu) lưu trữ tại nút: **Info**.
- Địa chỉ nút gốc của cây con trái trong bộ nhớ: **Left**.
- Địa chỉ nút gốc của cây con phải trong bộ nhớ: **Right**.

3. Các thao tác trên cây nhị phân tìm kiếm

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 BSTree
{
    /* Định nghĩa kiểu dữ liệu cho cây nhị phân
    (Cây NPTK) */
    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 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);
}

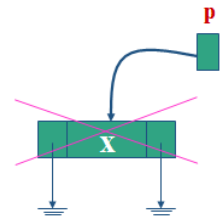
```

d. Hủy nút

```

void deleteTNode(TNode* &p)
{
    if(p == NULL) return;
    delete p;
}

```



e. Khởi tạo cây

```

void initBSTree(BSTree &bst)
{ //initialize BTree
    bst.Root = NULL;
}

```



f. Kiểm tra cây rỗng

```

void isEmpty(BSTree bst)
{ //Kiểm tra cây Btree có rỗng hay không?
    return (bst.Root == NULL)? 1 : 0;
}

```

g. 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 vào gốc
        root = p;
        return 1; //Thực hiện thành công
    }
    if(root→Info == p→Info)
        return 0; //Bị trùng nút
    if(p→Info < root→Info)
        insertTNode(root→Left, p); //Them trai
    else
        insertTNode(root→Right, p); //Them phải
    return 1; //Thực hiện thành công
}
```

h. Tạo cây từ một file chứa n số nguyên ($n > 0$)

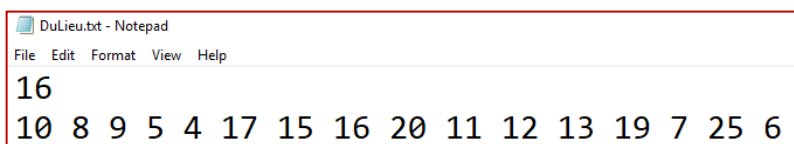
```
void createBSTree_FromFile(BSTree &bst, char
fileName[])
{ //Ham tao cay NPTK tu file
    FILE *f;
    f = fopen(fileName, "rt");
    if(!f) return;
    int n;
    fscanf(f, "%d", &n);
    ItemType x;
    initBSTree(bst);
    for(int i = 1; i <= n; i++)
    {
        fscanf(f, "%d", &x);
```

```

        TNode* p = createTNode(x);
        insertTNode(bst.Root, p);
    }
    fclose(f);
}

```

Lưu ý: Cấu trúc file text như sau: dòng đầu tiên là một số nguyên n cho biết số lượng phần tử, dòng thứ hai là danh sách n số nguyên cách nhau bằng khoảng trắng, ví dụ như sau:



DuLieu.txt - Notepad
File Edit Format View Help
16
10 8 9 5 4 17 15 16 20 11 12 13 19 7 25 6

i. Duyệt cây theo traverseNLR

```

void traverseNLR(TNode* root)
{
    ...
}

```

j. Duyệt cây theo traverseLNR

```

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

```

k. Duyệt cây theo traverseLRN

```

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

```

l. Tìm kiếm nút chứa giá trị x

```

TNode* findTNode(TNode* root, ItemType x)
{
    ...
}

```

m. Xóa nút có giá trị x

```
TNode* findTNodeReplace(TNode* &p)
{ //Hàm tìm nút q thế mạng cho nút p, f là nút
  cha của nút q.
    TNode* f = p;
    TNode* q = p→Right;
    while(q→Left != NULL)
    {
        f = q; //Lưu nút cha của q
        q = q→Left; //q qua bên trái
    }
    p→Info = q→Info;
    //Tìm được phần tử thế mạng cho p là q
    if(f == p) //Nếu cha của q là p
        f→Right = q→Right;
    else
        f→Left = q→Right;
    return q;
    //trả về nút q là nút thế mạng cho p
}

//=====
int deleteTNodeX(TNode* &root, ItemType x)
{ //Hàm xóa nút có giá trị là x
    if(root == NULL) //Khi cây rỗng
        return 0; //Xóa không thành công
    if(root→Info > x)
        return deleteTNodeX(root→Left, x);
    else if(root→Info < x)
        return deleteTNodeX(root→Right, x);
    else
    {
        //root→Info = x, tìm nút thế mạng cho root
    }
}
```

```

TNode* p = root;
if(root→Left == NULL)
{ //khi cây con không có nhánh trái
    root = root→Right;
    delete p;
}
else if(root→Right == NULL)
{ //khi cây con không có nhánh phải
    root = root→Left;
    delete p;
}
else
{
    /* khi cây con có cả 2 nhánh, chọn min
    của nhánh phải để thế mạng */
    TNode* q = findTNodeReplace(p);
    //Xóa nút q là nút thế mạng cho p
    delete q;
}
return 1; //Xóa thành công
}
}

```

n. Tìm nút có giá trị lớn nhất trong cây nhị phân tìm kiếm

```

int maxTNode(TNode* root)
{
    //Ham tìm nút có giá trị lớn nhất của cây
    ...
}

```

Gợi ý: Nút lớn nhất là nút bên phải cùng của cây.

- Gán p = root.
- Trong khi p->Right ≠ NULL thì p=p->Right.
- Hàm trả về p->Info.

- o. Xuất ra màn hình nội dung các nút ở mức thứ k của cây

```
void showTNodeOfLevelK(TNode *root, int k)
{
    ...
}
```

Gợi ý: Theo nguyên tắc của phép duyệt cây NLR

- Nếu root = NULL thì dừng (bằng lệnh *return*).
- Nếu k = 0 thì xuất thông tin của root→Info.
- Giảm k xuống 1 đơn vị.
- Gọi đệ quy cho nhánh con trái với root→Left ở mức k.
- Gọi đệ quy cho nhánh con phải với root→Right ở mức k.

- p. Xuất ra màn hình nội dung các nút lá ở mức thứ k của cây

```
void showTNodeIsLeafOfLevelK(TNode *root, int k)
{
    ...
}
```

Gợi ý: Tương tự như hàm **showTNodeOfLevelK**

- Nếu root = NULL thì dừng (bằng lệnh *return*).
- Nếu k = 0 và root→Left = NULL và root→Right = NULL thì xuất thông tin của root→Info.
- Giảm k xuống 1 đơn vị.
- Gọi đệ quy cho nhánh con trái với root→Left ở mức k.
- Gọi đệ quy cho nhánh con phải với root→Right ở mức k.

- q. Đếm số lượng nút ở mức thứ k của cây

```
int countTNodeOfLevelK(TNode* root, int k)
{
    ...
}
```

Gợi ý:

- Nếu root = NULL thì hàm trả về 0.

- Nếu $k = 0$ thì hàm trả về 1.
- Giảm k xuống 1 đơn vị.
- Tính **cnl** = Số lượng nút của nhánh con trái với $\text{root} \rightarrow \text{Left}$ ở mức k .
- Tính **cnr** = Số lượng nút của nhánh con phải với $\text{root} \rightarrow \text{Right}$ ở mức k .
- Hàm trả về **cnl** + **cnr**.

r. Xóa toàn bộ cây

```
void deleteTree(TNode* &root)
{
    if(!root) return;
    deleteTree(root→Left);
    //đệ quy xóa cây con trái
    deleteTree(root→Right);
    //đệ quy xóa cây con phải
    delete root;
    //hoặc có thể dùng lệnh deleteTNode(root);
}
```

s. Nút có khoảng cách về giá trị gần nhất với phần tử x trong cây

```
int minDistance(TNode* root, ItemType x)
{
    if(!root) return -1;
    int min = root→Info;
    int mindis = abs(x - min);
    while(root != NULL)
    {
        if(root→Info == x)
            return x;
        if(mindis > abs(x - root→Info))
        {
```

```

        min = root→Info;
        mindis = abs(x - min);
    }
    if(x > root→Info)
        root = root→Right;
    else
        root = root→Left;
    }
    return min;
}

```

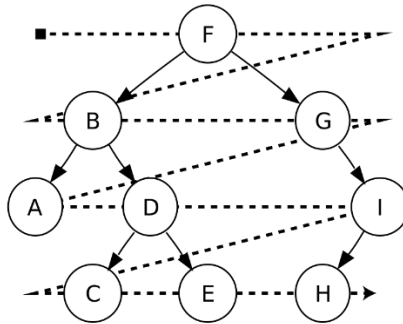
- t. Nút có khoảng cách về giá trị xa nhất với phần tử x trong cây

```

int maxDistance(TNode* root, ItemType x)
{
    if(!root) return -1;
    TNode* minLeft = root;
    //Tìm nút trái nhất
    while(minLeft→Left != NULL)
        minLeft = minLeft→Left;
    TNode* maxRight = root;
    //Tìm nút phải nhất
    while(maxRight→Right != NULL)
        maxRight = maxRight→Right;
    int dis1 = abs(x - minLeft→Info);
    int dis2 = abs(x - maxRight→Info);
    if(dis1 > dis2)
        return minLeft→Info;
    else
        return maxRight→Info;
}

```

u. Duyệt cây theo chiều rộng (Breadth - first search) cho NLR



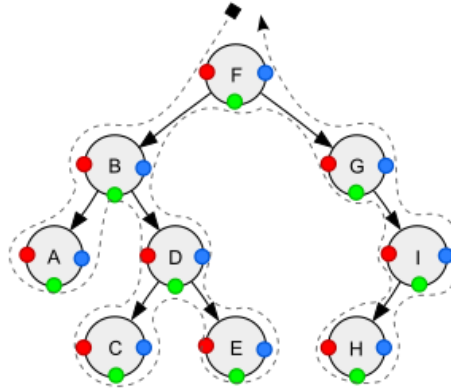
```
void traverseBreadthNLR(TNode *root)
{ /* In theo thứ tự cấp */
    if (root == NULL) return;

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

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

Tham khảo: https://en.wikipedia.org/wiki/Tree_traversal#Breadth-first_search,_or_level_order

v. Duyệt cây theo chiều sâu (Depth - first search) cho NLR



```
void traverseDepthNLR(TNode *root)
{
    if (root == NULL) return;

    stack <TNode *> s;
    s.push(root);

    while(!s.empty())
    {
        TNode *p;
        p = s.top();
        s.pop();
        showTNode(p);
        // con phải được đẩy vào Stack trước, để
        // bên trái được xử lý trước
        if (p->Right != NULL)
            s.push(p->Right);
        if (p->Left != NULL)
            s.push(p->Left);
    }
}
```

Tham khảo: https://en.wikipedia.org/wiki/Tree_traversal#Depth-first_search_of_binary_tree

II. Bài tập hướng dẫn mẫu

Bài 1. Ứng dụng Cây NPTK để viết chương trình quản lý các số nguyên?

- **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 BSTree
```

```
{ //Định nghĩa kiểu dữ liệu cho Cây NPTK
```

```
    TNode* Root;
```

```
};
```

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

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

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

```
{
```

```
    ...
```

```
}
```

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

```
void initBSTree(BSTree &bst)
```

```

{ //initialize BSTree
    ...
}
//=====
int insertTNode(TNode* &T, TNode* p)
{
    ...
}
//=====
void createBSTree_FromArray(BTree &bt, ItemType
a[], int na)
{ //Hàm tạo cây NPTK từ mảng a
    ...
}
//=====
void showTNode(TNode* p)
{
    ...
}
//=====
void traverseNLR(TNode* root)
{ //Hàm duyệt cây theo thứ tự NLR
    ...
}
//=====
int countTNodeIsLeafOfLevelK(TNode *root, int k)
{
    ...
}
//=====

```

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

III. Bài tập ở lớp

Bài 1. Cho cây nhị phân tìm kiếm 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:

- a. Tạo cây NPTK 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).
- b. Duyệt cây NPTK bằng **6 cách**: traverseNLR, traverseLNR, traverseLRN, traverseNRL, traverseRNL, traverseRLN.
- c. Duyệt cây NPTK theo chiều rộng tương ứng với 2 phép duyệt cây NLR, NRL.
- d. Duyệt cây NPTK theo chiều sâu tương ứng với 6 phép duyệt cây NLR, NRL, LNR, RNL, LRN và RLN.
- e. Thêm 1 nút có giá trị **x** vào cây.
- f. Tìm kiếm 1 nút có giá trị **x** trên cây hay không.
- g. Xóa nút có giá trị **x** trên cây.
- h. Xuất các phần tử theo chiều giảm dần.
- i. Đếm số giá trị lớn hơn **x**, nhỏ hơn **x**, có giá trị trong đoạn [**x**, **y**].
- j. Tìm nút có giá trị lớn nhất, nhỏ nhất của cây.
- k. Xuất ra nội dung các nút ở mức **k**/nội dung các nút lá ở mức **k**/ nội dung các nút chỉ có 1 con ở mức **k**/
- l. Đếm số nút ở mức **k**/số nút lá ở mức **k**/ số nút chỉ có 1 con ở mức **k**/
- m. Tính tổng giá trị các nút dương/ giá trị các nút âm trên cây.
- n. Tìm phần tử có khoảng cách về giá trị gần nhất với phần tử **x** trong cây (nếu cây rỗng trả về -1).
- o. Tìm phần tử có khoảng cách về giá trị xa nhất với phần tử **x** trong cây (nếu cây rỗng trả về -1).
- p. Đếm số nút của cây (dùng đệ quy / không dùng đệ quy).
- q. Đếm số nút là số nguyên tố, là số chính phương, là số hoàn thiện,

là số thịnh vượng, là số yếu của cây.

- r. Tính tổng giá trị các nút của cây (dùng đệ quy / không dùng đệ quy).
- s. Tính tổng giá trị các nút là số nguyên tố, là số chính phương, là số hoàn thiện, là số thịnh vượng, là số yếu của cây.
- t. Xóa toàn bộ cây.

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

- a. Tạo cây NPTK bằng 2 cách (nhập liệu từ bàn phím, tạo ngẫu nhiên tự động).
- b. Duyệt cây NPTK bằng 6 cách: *traverseNLR*, *traverseLNR*, *traverseLRN*, *traverseNRL*, *traverseRNL*, *traverseRLN*.
- c. Thêm 1 nút là phân số p vào cây.
- d. Tìm kiếm 1 phân số x có trên cây hay không?
- e. Xóa một phân số x trên cây.
- f. Xóa những phân số > 2 (*xét theo giá trị*).
- g. Xóa những phân số có mẫu số là số nguyên tố.
- h. Tính tổng các phân số.
- i. Tìm phân số nhỏ nhất.
- j. Tìm phân số lớn nhất.
- k. Liệt kê các phân số có tử số lớn hơn mẫu số.
- l. Liệt kê các phân số có tử số nhỏ hơn mẫu số.
- m. Liệt kê các phân số có tử số và mẫu số đồng thời là các số nguyên tố.
- n. Liệt kê các phân số ở mức k (k được nhập từ bàn phím).
- o. Đếm số lượng phân số ở mức k (k được nhập từ bàn phím).
- p. Tính tổng các phân số ở mức k (k được nhập từ bàn phím).
- q. Đếm có bao nhiêu phân số có cả tử số và mẫu số đều là các số nguyên tố.
- r. Xóa toàn bộ danh sách.

IV. Bài tập về nhà

Bài 3. Tiếp theo **Bài 1**. Hãy viết các hàm thực hiện các chức năng sau:

- a. Viết hàm xuất các số hoàn thiện trong cây.
- b. Viết hàm xuất tất cả các nút trên tầng thứ k của cây. (*)
- c. Viết hàm xuất tất cả các nút trên cây theo thứ tự từ tầng 0 đến tầng $h-1$ của cây (với h là chiều cao của cây). (*)
- d. Đếm số lượng nút lá mà thông tin tại nút đó là giá trị chẵn.
- e. Đếm số lượng nút có đúng 1 con mà thông tin tại nút đó là số nguyên tố.
- f. Đếm số lượng nút có đúng 2 con mà thông tin tại nút đó là số chính phương.
- g. Đếm số lượng nút nằm ở tầng thấp hơn tầng thứ k của cây.
- h. Đếm số lượng nút nằm ở tầng cao hơn tầng thứ k của cây.
- i. Tính tổng các nút lẻ.
- j. Tính tổng các nút lá mà thông tin tại nút đó là giá trị chẵn.
- k. Tính tổng các nút có đúng 1 con mà thông tin tại nút đó là số nguyên tố.
- l. Tính tổng các nút có đúng 2 con mà thông tin tại nút đó là số chính phương.
- m. Kiểm tra cây nhị phân T có phải là "cây nhị phân tìm kiếm" hay không?
- n. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng" hay không?
- o. Kiểm tra cây nhị phân T có phải là "cây nhị phân cân bằng hoàn toàn" hay không?

Bài 4. Cây NPTK 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, **mean** là nghĩa tiếng Việt). Hãy xây dựng cây NPTK với những chức năng sau:

- a. Tạo cây NPTK từ 1 file text lưu từ điển Anh-Việt.

- b. Duyệt cây NPTK để xem nội dung theo phép duyệt cây traverseLNR.
- c. Thêm một từ bất kỳ vào cây, duyệt lại cây để xem kết quả.
- d. Xóa một từ bất kỳ khỏi cây, duyệt lại cây để xem kết quả.
- e. Tra cứu nghĩa của 1 từ bất kỳ.
- f. Bổ sung hay chỉnh sửa nghĩa của 1 từ bất kỳ.
- g. Xóa toàn bộ cây.

Bài 5. Sinh viên tìm hiểu Cây AVL. Cài đặt cây AVL để lưu trữ dữ liệu là một từ điển cho phép việc tra cứu nghĩa Tiếng Việt khi biết từ Tiếng Anh.

-- HẾT --