HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

LẬP TRÌNH C CƠ BẢN

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

LẬP TRÌNH C CƠ BẢN

CÂY – PHẦN 2

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Bài toán thao tác và duyệt cây nhị phân (P.04.10.01)
- Bài toán kiểm tra cây nhị phân cân bằng (P.04.10.02)

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN (P.04.10.01)

- Mỗi nút của một cây nhị phân T có trường id (định danh của nút, id không trùng lặp nhau). Thực hiện chuỗi các thao tác sau đây trên cây T (ban đầu, T là cây rỗng)
  - MakeRoot u: tạo một nút gốc có id bằng u
  - AddLeft u v: tạo một nút có id = u và chèn vào vị trí con trái của nút có id bằng v trên T (không thực hiện hành động chèn nếu nút có id bằng u đã tồn tại hoặc nút có id bằng v không tồn tại hoặc nút có id bằng v đã có nút con trái rồi)
  - AddRight u v: tạo một nút có id = u và chèn vào vị trí con phải của nút có id bằng v trên T (không thực hiện hành động chèn nếu nút có id bằng u đã tồn tại hoặc nút có id bằng v không tồn tại hoặc nút có id bằng v đã có nút con phải rồi)
  - PreOrder: đưa ra trên 1 dòng mới dãy id của các nút trong phép duyệt cây T theo thứ tự trước (các phần tử cách nhau bởi đúng 1 ký tự cách SPACE)
  - InOrder: đưa ra trên 1 dòng mới dãy id của các nút trong phép duyệt cây T theo thứ tự giữa (các phần tử cách nhau bởi đúng 1 ký tự cách SPACE)
  - PostOrder: đưa ra trên 1 dòng mới dãy id của các nút trong phép duyệt cây T theo thứ tự sau (các phần tử cách nhau bởi đúng 1 ký tự cách SPACE)

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN

- **Input**: Mỗi dòng là 1 trong số cách thao tác với định dạng được mô tả ở trên (thao tác MakeRoot chỉ xuất hiện đúng 1 lần và luôn ở ngay dòng đầu tiên). Kết thúc của dữ liệu input là dòng chứa duy nhất ký tự *
- **Output:** Ghi ra trên 1 dòng kết quả của 1 trong số 3 thao tác InOrder, PreOrder, PostOrder mô tả ở trên

stdin	stdout
MakeRoot 1 AddLeft 2 1 AddRight 3 1 AddLeft 4 3 AddRight 5 2 PreOrder AddLeft 6 5 AddRight 7 5 InOrder *	1 2 5 3 4 2 6 5 7 1 4 3

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN

- Cấu trúc dữ liệu:

```
typedef struct Node{
    int id;
    struct Node* leftChild;
    struct Node* rightChild;

}Node;
```

- Tạo một node mới với id =u:

```
Node* makeNode(int u){
    Node* p = (Node*)malloc(sizeof(Node));
    p->leftChild = NULL;
    p->rightChild = NULL;
    p->id = u;
    return p;
}
```

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN – MÃ GIẢ

- Thêm một node mới có id =u vào con trái hoặc phải của node có id = v trên cây

```
void addLeft(int u, int v, Node* r){
     p = find(v, r)
    if p is NULL then
        return
    end if
    if p.leftChild is not NULL then
        return
    end if
    q = find(u, r)
    if q is not NULL then
        return
    end if
    p.leftChild := makeNode(u)
}
```

```
void addRight(int u, int v, Node*
r){
    p = find(v, r)
    if p is NULL then
        return
    end if
    if p.rightChild is not NULL then
        return
    end if
    q = find(u, r)
    if q is not NULL then
        return
   end if
    p.rightChild = makeNode(u)
}
```

```
Node* find(int u, Node* r){
    if r is NULL then
        return NULL
    end if

    if r.id equals u then
        return r
    end if
    p = find(u, r.leftChild)
    if p is not NULL then
        return p
    end if
    return find(u, r.rightChild)
}
```

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN – MÃ GIẢ

- Duyệt cây theo thứ tự trước, giữa, sau

```
void preOrder(Node* r){
     if r is NULL then
        return
    end if

    print(r.id)
    preOrder(r.leftChild)
    preOrder(r.rightChild)
}
```

```
void inOrder(Node* r){
     if r is NULL then
        return
    end if

    inOrder(r.leftChild)
    print(r.id)
    inOrder(r.rightChild)
}
```

```
void postOrder(Node* r){
    if r is NULL then
        return
    end if

    postOrder(r.leftChild)
    postOrder(r.rightChild)
    print(r.id)
}
```

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN - CODE

- Thêm một node mới có id =u vào con trái hoặc phải của node có id = v trên cây

```
void addLeft(int u, int v, Node* r){
    Node* p = find(v,r);
    if(p == NULL) return;

    if(p->leftChild != NULL) return;
    Node* q = find(u,r);
    if(q != NULL) return;// node having
id = u exists -> do not insert more
    p->leftChild = makeNode(u);
}
```

```
void addRight(int u, int v, Node*
r){
    Node* p = find(v,r);
    if(p == NULL) return;
if(p->rightChild != NULL) return;
    Node* q = find(u,r);
    if(q != NULL) return;// node
having id = u exists -> do not
insert more
    p->rightChild = makeNode(u);
}
```

```
Node* find(int u, Node* r){
    if(r == NULL) return NULL;
    if(r->id == u) return r;
    Node* p = find(u,r->leftChild);
    if(p != NULL) return p;
    return find(u,r->rightChild);
}
```

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN - CODE

- Duyệt cây theo thứ tự trước, giữa, sau

```
void preOrder(Node* r){
    if(r == NULL) return;
    printf("%d ",r->id);
    preOrder(r->leftChild);
    preOrder(r->rightChild);
}
```

```
void inOrder(Node* r){
    if(r == NULL) return;
    inOrder(r->leftChild);
    printf("%d ",r->id);
    inOrder(r->rightChild);
}
```

```
void postOrder(Node* r){
    if(r == NULL) return;
    postOrder(r->leftChild);
    postOrder(r->rightChild);
    printf("%d ",r->id);
}
```

BÀI TOÁN THAO TÁC VÀ DUYỆT CÂY NHỊ PHÂN - CODE

- Một số hàm triển khai:

```
int main(){
    char cmd[100];
    Node* root = NULL;
    while(1){
        scanf("%s",cmd);
        if(strcmp(cmd,"*")==0) break;
        else if(strcmp(cmd,"MakeRoot") == 0){
            int u;
            scanf("%d",&u);
            root = makeNode(u);
        }else if(strcmp(cmd,"AddLeft")==0){
            int u,v;
            scanf("%d%d",&u,&v);
            addLeft(u,v,root);
        }
        else if(strcmp(cmd,"AddRight")==0){
            int u,v;
            scanf("%d%d",&u,&v);
            addRight(u,v,root);
        }else if(strcmp(cmd,"InOrder")==0){
            inOrder(root);
            printf("\n");
        }else if(strcmp(cmd,"PreOrder")==0){
            preOrder(root);
            printf("\n");
        }else if(strcmp(cmd,"PostOrder")==0){
            postOrder(root);
            printf("\n");
        }
    }
}
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG (P.04.10.02)

- Mỗi nút của một cây nhị phân có một trường id là định danh của nút. Xây dựng một cây nhị phân và kiểm tra xem cây đó có phải là cây cân bằng không, tính chiều cao của cây đã cho (số nút của cây có thể lên đến 50000).
- Đầu vào
  - Dòng 1 chứa MakeRoot u: tạo nút gốc của cây có id = u
  - Mỗi dòng tiếp theo chứa các lệnh AddLeft hoặc AddRight với định dạng
    - AddLeft u v: tạo một nút có id = u, thêm nút này vào làm con bên trái của nút có id = v (nếu không tồn tại)
    - AddRight u v: tạo một nút có id = u, thêm nút này vào làm con bên phải của nút có id = v (nếu không tồn tại)
  - Dòng cuối cùng chứa * để đánh dấu kết thúc đầu vào
- Đầu ra Ghi hai số nguyên z và h (cách nhau bởi một KHOẢNG TRẮNG) trong đó:
  - h là chiều cao (số nút của đường dẫn dài nhất từ gốc đến một lá)
  - z = 1 nếu cây là cây cân bằng và z = 0, ngược lại.

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG

- Dữ liệu input và output

stdin	stdout
MakeRoot 1 AddLeft 2 1 AddRight 3 1 AddLeft 9 2 AddRight 4 2 AddLeft 6 3 AddRight 5 3 AddLeft 7 4 AddRight 8 4 *	1 4

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG

- Cấu trúc dữ liệu:

```
#define N 1000001
typedef struct TNode{
    int id;
    struct TNode* left;
    struct TNode*right;
}Node;
Node* root;
Node* nodes[N]; //Mảng chứa các node của cây và chỉ số
                  của mảng trùng với id của node
```

```
// Cấu trúc chứa thông tin của cây
typedef struct TINFO{
    int balanced; // cây có cân bằng không
    int hl; // Độ sâu của con trái
    int hr; // Độ sâu của con phải
    int h;  // Độ sâu của cây
}INFO;
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG

- Khởi tạo một node với khóa id

```
Node* makeNode(int id){
    Node* p = (Node*)malloc(sizeof(Node));
    p->id = id;
    p->left = NULL;
    p->right = NULL;
    return p;
}
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG– MÃ GIẢ

- Thêm một node mới có id =u vào con trái hoặc phải của node có id = v trên cây

```
int addLeft(int u, int v){
     if nodes[u] is not NULL or nodes[v] is NULL then
         return 0
    end if

    if nodes[v].left is not NULL then
         return 0
    end if

    p = makeNode(u)
    nodes[v].left = p
    nodes[u] = p
    return 1
}
```

```
int addRight(int u, int v){
     if nodes[u] is not NULL or nodes[v] is NULL
then
         return 0
    end if

    if nodes[v].right is not NULL then
         return 0
    end if

    p = makeNode(u)
    nodes[v].right := p
    nodes[u] = p
    return 1
}
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG– MÃ GIẢ

- Hàm visit được thiết kế để duyệt một cây nhị phân và trả về một cấu trúc INFO chứa hai thông tin: Chiều cao của cây có gốc và liệu cây con đó có cân bằng không (cân bằng khi balanced =1 và 0 nếu ngược lại).

```
INFO visit(Node * r){
if r is NULL then
        INFO i
        i.balanced := 1
        i.h := 0
        return i
    end if
    i1 := visit(r.left)
    i2 := visit(r.right)
    INFO i
    i.h := max(i1.h, i2.h) + 1
    if i1.balanced = 0 then
        i.balanced := 0
        return i
    end if
    if i2.balanced = 0 then
        i.balanced := 0
        return i
    end if

    if abs(i1.h - i2.h) >= 2 then
        i.balanced := 0
    else
        i.balanced := 1
    end if

    return i;
}
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG– CODE

- Thêm một node mới có id =u vào con trái hoặc phải của node có id = v trên cây

```
int addLeft(int u, int v){// add a new node id = u as a
left child of the node id = v (if not exists)


    if(nodes[u] != NULL || nodes[v] == NULL) return 0;
    if(nodes[v]->left != NULL) return 0;
    Node* p = makeNode(u);
    nodes[v]->left = p;
    nodes[u] = p;
    return 1;
}
```

```
int addRight(int u, int v){


    if(nodes[u] != NULL || nodes[v] == NULL)
           return 0;
    if(nodes[v]->right != NULL) return 0;
    Node* p = makeNode(u);
    nodes[v]->right = p;
    nodes[u] = p;
    return 1;
}
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG– CODE

- Hàm visit được thiết kế để duyệt một cây nhị phân và trả về một cấu trúc INFO chứa hai thông tin: Chiều cao của cây có gốc và liệu cây con đó có cân bằng không (cân bằng khi balanced =1 và 0 nếu ngược lại).

```
INFO visit(Node * r){
    if(r == NULL){
        INFO i;
        i.balanced = 1;
        i.h = 0;
        return i;
    }
    INFO i1 = visit(r->left);
    INFO i2 = visit(r->right);
    INFO i;
    i.h = (i1.h > i2.h ? i1.h : i2.h) + 1;

    if(i1.balanced == 0){
        i.balanced = 0; return i;
    }
    if(i2.balanced == 0){
        i.balanced = 0; return i;
    }
    if(abs(i1.h - i2.h) >= 2){
        i.balanced = 0;
    }else
        i.balanced = 1;

    return i;
}
```

BÀI TOÁN KIỂM TRA CÂY NHỊ PHÂN CÂN BẰNG– CODE

- Một số hàm triển khai:

```
void solve(){
    for(int i = 0; i < N; i++) nodes[i] = NULL;
    while(1){
        char s[20];
        scanf("%s",s);
        if(strcmp(s,"*")==0) break;
        if(strcmp(s,"MakeRoot")==0){
            int u; scanf("%d",&u);
            root = makeNode(u);
            nodes[u] = root;
        }
```

```
    else if(strcmp(s,"AddLeft")==0){
            int u,v;
            scanf("%d%d",&u,&v);
            addLeft(u,v);
        }else if(strcmp(s,"AddRight")==0){
            int u,v;
            scanf("%d%d",&u,&v);
            addRight(u,v);
        }
    }
    INFO i = visit(root);
    printf("%d %d",i.balanced,i.h);
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !