
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 1

ONE LOVE. ONE FUTURE.

NỘI DUNG

- Bài toán tìm độ sâu và độ cao của cây (P.04.09.01)
- Bài toán duyệt cây theo thứ tự trước, giữa, sau (P.04.09.02)
- Bài toán cây gia phả (P.04.09.03)

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY (P.04.09.01)

- Mỗi nút trên 1 cây có trường id (identifier) là một số nguyên (id của các nút trên cây đôi một khác nhau)
- Thực hiện 1 chuỗi các hành động sau đây bao gồm các thao tác liên quan đến xây dựng cây và duyệt cây
  - MakeRoot u: Tạo ra nút gốc u của cây
  - Insert u v: tạo mới 1 nút u và chèn vào cuối danh sách nút con của nút v (nếu nút có id bằng v không tồn tại hoặc nút có id bằng u đã tồn tại thì không thêm mới)
  - Height u: Tính và trả về độ cao của nút u
  - Depth u: Tính và trả về độ sâu của nút u
- Biết rằng dữ liệu đầu vào có 1 lệnh duy nhất là MakeRoot và luôn ở dòng đầu tiên
- Dữ liệu: bao gồm các dòng, mỗi dòng có định dạng như mô tả ở trên, trong đó dòng cuối dùng ghi * (dấu hiệu kết thúc dữ liệu)
- Kết quả: ghi ra mỗi dòng kết quả của các lệnh Height và Depth tương ứng đọc được từ đầu vào

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – VÍ DỤ

- Dữ liệu input và output

stdin	stdout
MakeRoot 10 Insert 11 10 Insert 1 10 Insert 3 10 Insert 5 11 Insert 4 11 Depth 4 Insert 8 3 Insert 2 3 Insert 7 3 Insert 6 4 Insert 9 4 Height 10 Height 11 Height 4 *	3 4 3 2

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY

- Cấu trúc dữ liệu:

```
typedef struct Node{
          int id;
          struct Node* leftMostChild; // pointer to the left-most child
          struct Node* rightSibling; // pointer to the right sibling
          struct Node* parent;
}Node;
```

- Tạo một node mới với id =u:

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

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – MÃ GIẢ

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

```
void insert(Node* r, int u, int v){
    p = find(r, v)
    if p is NULL then return
    q = makeNode(u)
    if p.leftMostChild is NULL then
        p.leftMostChild = q
        q.parent = p
        return
    h = p.leftMostChild
    while h.rightSibling is not NULL
        h = h.rightSibling
    h.rightSibling = q
    q.parent = p
}
```

```
Node* find(Node* r, int u){
    if r is NULL then
        return NULL
    if r.id is equal to u then
        return r
    p = r.leftMostChild
    while p is not NULL do
        q = find(p, u)
        if q is not NULL then
            return q
        end if
        p = p.rightSibling
    end while
    return NULL
}
```

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – MÃ GIẢ

- Tìm độ sâu và độ cao của một cây

```
int depth(Node* r){
    p = r
    d = 0

    while p is not NULL do
        d = d + 1
        p = p.parent
    end while

    return d
}
```

```
int height(Node* r){
    maxH = 0
    if r is NULL then
        return 0
    end if

    for each p in r.leftMostChild to NULL do
        h = height(p)
        if h > maxH then
            maxH = h
        end if
    end for
    return maxH + 1
}
```

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – CODE

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

```
void insert(Node* r, int u, int v){
Node* p = find(r,v);
    if(p == NULL) return;
    Node* q = makeNode(u);
    if(p->leftMostChild == NULL){
        p->leftMostChild = q;
        q->parent = p;
        return; }
    Node* h = p->leftMostChild;
    while(h->rightSibling != NULL)
        h = h->rightSibling;
    h->rightSibling = q;
    q->parent = p;
}
```

```
Node* find(Node* r, int u){
    if(r == NULL) return NULL;
    if(r->id == u) return r;
    Node* p = r->leftMostChild;
    while(p != NULL){
        Node* q = find(p,u);
        if(q != NULL) return q;
        p = p->rightSibling;
    }
    return NULL;
}
```

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – CODE

- Tìm độ sâu và độ cao của một cây

```
int depth(Node* r){
    Node* p = r;    int d = 0;
    while(p != NULL){
        d += 1;
        p = p->parent;
    }
    return d;
}
```

```
int height(Node* r){
    int maxH = 0;
    if(r == NULL) return 0;
    for(Node* p = r->leftMostChild; p != NULL; p = p->rightSibling){
        int h = height(p);
        if(h > maxH) maxH = h;
    }
    return maxH + 1;
}
```

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – CODE

- Một số hàm triển khai:

```
void solve(){
    Node* root = NULL; char cmd[50];
    while(1){
          scanf("%s",cmd);
          if(strcmp(cmd,"*") == 0) break;
          else if(strcmp(cmd,"MakeRoot") == 0){
              int id;
              scanf("%d",&id);
              root = makeNode(id);
          }
         else if(strcmp(cmd,"Insert") == 0){
              int u,v; scanf("%d%d",&u,&v);
              insert(root,u,v);
          }
```

```
          else if(strcmp(cmd,"Height") == 0){
              int id; scanf("%d",&id);
              Node* p = find(root,id);
              printf("%d\n", height(p));
           }else if(strcmp(cmd,"Depth") == 0){
              int id; scanf("%d",&id);
              Node* p = find(root,id);
              int ans = depth(p);
              printf("%d\n",ans);
           }
    }
    freeTree(root);
}
```

BÀI TOÁN TÌM ĐỘ SÂU VÀ ĐỘ CAO CỦA CÂY – CODE

- Một số hàm triển khai:

```
int main(){
    solve();
}
```

```
void freeTree(Node* r){
    if(r == NULL) return;
    Node* p = r->leftMostChild;
    while(p != NULL){
        Node* np = p->rightSibling;
        free(p);
        p = np;
    }
    free(r);
}
```

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU

- Mỗi nút của cây có trường id (số nguyên duy nhất, không trùng lặp)
- Thực hiện 1 chuỗi các hành động sau đây bao gồm các thao tác liên quan đến xây dựng cây và duyệt cây:
  - MakeRoot u: Tạo ra nút gốc u của cây
  - Insert u v: tạo mới 1 nút u và chèn vào cuối danh sách nút con của nút v
  - PreOrder: in ra thứ tự các nút trong phép duyệt cây theo thứ tự trước
  - InOrder: in ra thứ tự các nút trong phép duyệt cây theo thứ tự giữa
  - PostOrder: in ra thứ tự các nút trong phép duyệt cây theo thứ tự sau
- **Dữ liệu**: bao gồm các dòng, mỗi dòng là 1 trong số các hành động được mô tả ở trên, dòng cuối dùng là * (đánh dấu sự kết thúc của dữ liệu).
- **Kết quả**: ghi ra trên mỗi dòng, thứ tự các nút được thăm trong phép duyệt theo thứ tự trước, giữa, sau của các hành động PreOrder, InOrder, PostOrder tương ứng đọc được từ dữ liệu đầu vào

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – VÍ DỤ

- Dữ liệu input và output

stdin	stdout
MakeRoot 10 Insert 11 10 Insert 1 10 Insert 3 10 InOrder Insert 5 11 Insert 4 11 Insert 8 3 PreOrder Insert 2 3 Insert 7 3 Insert 6 4 Insert 9 4 InOrder PostOrder *	11 10 1 3 10 11 5 4 1 3 8 5 11 6 4 9 10 1 8 3 2 7 5 6 9 4 11 1 8 2 7 3 10

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU

- Cấu trúc dữ liệu:

```
struct Node{
    int id;
    Node* leftMostChild;
    Node* rightSibling;
};
```

- Tạo một node mới với id =u:

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

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – MÃ GIẢ

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

```
void insert(Node* r, int u, int v){
    p = find(r, v)
    if p is NULL then return
    q = makeNode(u)
    if p.leftMostChild is NULL then
        p.leftMostChild = q
        return
    h = p.leftMostChild
    while h.rightSibling is not NULL
        h = h.rightSibling
    h.rightSibling = q
}
```

```
Node* find(Node* r, int u){
    if r is NULL then
        return NULL
    if r.id is equal to u then
        return r
    p = r.leftMostChild
    while p is not NULL do
        q = find(p, u)
        if q is not NULL then
            return q
        end if
        p = p.rightSibling
    end while
    return NULL
}
```

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – 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) // Visit the root r

    p = r.leftMostChild
    while p is not NULL do
        preOrder(p)
        p = p.rightSibling
    end while
}
```

```
void inOrder(Node* r){
    if r is NULL then return
    end if
    p = r.leftMostChild
    inOrder(p)
    print(r.id)
    if p is NULL then return
    end if
    p = p.rightSibling
    while p is not NULL do
        inOrder(p)
        p := p.rightSibling
    end while
}
```

```
void postOrder(Node* r){
     if r is NULL then
        return
    end if
    p = r.leftMostChild
    while p is not NULL do
        postOrder(p)
        p = p.rightSibling
    end while

    print(r.id)
}
```

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – CODE

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

```
void insert(Node* r, int u, int v){
    Node* p = find(r,v);
    if(p == NULL) return;
    Node* q = makeNode(u);
    if(p->leftMostChild == NULL){
        p->leftMostChild = q;
        return;
    }
    Node* h = p->leftMostChild;
    while(h->rightSibling != NULL)
        h = h->rightSibling;
    h->rightSibling = q;
}
```

```
Node* find(Node* r, int u){
    if(r == NULL) return NULL;
    if(r->id == u) return r;
    Node* p = r->leftMostChild;
    while(p != NULL){
        Node* q = find(p,u);
        if(q != NULL) return q;
        p = p->rightSibling;
    }
    return NULL;
}
```

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – CODE

- Duyệt cây theo thứ tự trước, giữa, sau

```
void preOrder(Node* r){
    if(r == NULL) return;
    printf("%d ",r->id); // visit the root r
    Node* p = r->leftMostChild;
    while(p != NULL){
        preOrder(p);
        p = p->rightSibling;
    }
}
```

```
void inOrder(Node* r){
    if(r == NULL) return;
    Node* p = r->leftMostChild;// the first (left-most) child of r: r1
    inOrder(p);
    printf("%d ",r->id);
    if(p == NULL) return;
    p = p->rightSibling; // p = the second child of r: r2
    while(p != NULL){
        inOrder(p);
        p = p->rightSibling;
    }
}
```

```
void postOrder(Node* r){
    if(r == NULL) return;
    Node* p = r->leftMostChild;// start with the first (left-most) child of r
    while(p != NULL){
        postOrder(p);
        p = p->rightSibling;
    }
    printf("%d ",r->id);// lastly, visit the root r
}
```

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – CODE

- Một số hàm triển khai:

```
void solve(){
    Node* root = NULL;
    char cmd[50];
    while(1){
        scanf("%s",cmd);
        if(strcmp(cmd,"*") == 0) break;
        else if(strcmp(cmd,"MakeRoot") == 0){
            int id;
            scanf("%d",&id);
            root = makeNode(id);
        }else if(strcmp(cmd,"Insert") == 0){
            int u,v;
            scanf("%d%d",&u,&v);
            insert(root,u,v);
        }
```

```
        else if(strcmp(cmd,"PreOrder") == 0){
            preOrder(root);
            printf("\n");
         }else if(strcmp(cmd,"InOrder") == 0){
            inOrder(root);
            printf("\n");
         }else if(strcmp(cmd,"PostOrder") == 0){
            postOrder(root);
            printf("\n");
         }
    }
    freeTree(root);
}
```

BÀI TOÁN DUYỆT CÂY THEO THỨ TỰ TRƯỚC, GIỮA, SAU – CODE

- Một số hàm triển khai:

```
int main(){
    solve();
}
```

```
void freeTree(Node* r){
    if(r == NULL) return;
    Node* p = r->leftMostChild;
    while(p != NULL){
        Node* np = p->rightSibling;
        free(p);
        p = np;
    }
    free(r);
}
```

BÀI TOÁN CÂY GIA PHẢ (P.04.09.03)

- Cho một cây gia phả được biểu diễn bằng các mối quan hệ con-cha (c,p) trong đó c là con của p.
- Thực hiện các truy vấn về cây gia phả như sau:
  - descendants <name>: trả về số lượng hậu duệ của <name> đã cho
  - generation <name>: trả về số lượng thế hệ của hậu duệ của <name> đã cho
- Lưu ý rằng: tổng số người trong gia đình không quá 10^4
- Đầu vào chứa hai khối:
  - Khối đầu tiên chứa thông tin về con - cha, bao gồm các dòng (kết thúc bằng một dòng chứa ***), mỗi dòng chứa: <child> <parent> trong đó <child> là một chuỗi biểu thị tên của đứa trẻ và <parent> là một chuỗi biểu thị tên của phụ huynh.
  - Khối thứ hai chứa các dòng (kết thúc bằng một dòng chứa ***), mỗi dòng chứa hai chuỗi <cmd> và <param> trong đó <cmd> là lệnh (có thể là descendants hoặc generation) và <param> là tên đã cho của người tham gia trong truy vấn.
- Đầu ra: Mỗi dòng là kết quả của một truy vấn tương ứng.

BÀI TOÁN CÂY GIA PHẢ - VÍ DỤ

- Dữ liệu input và output

stdin	stdout
Peter Newman Michael Thomas John David Paul Mark Stephan Mark Pierre Thomas Mark Newman Bill David David Newman Thomas Mark *** descendants Newman descendants Mark descendants David generation Mark ***	10 5 2 2

BÀI TOÁN CÂY GIA PHẢ

- Cấu trúc dữ liệu:

```
typedef struct Node{
    char name[MAX_LEN];
    struct Node* leftMostChild;
    struct Node* rightSibling;
    struct Node* parent;
}Node;
```

- Tạo một node mới với tham số name truyền vào cho hàm:

```
Node* makeNode(const char* name){
    Node* p = (Node*)malloc(sizeof(Node));
    strcpy(p->name,name);
    p->leftMostChild = NULL;
    p->rightSibling = NULL;
    p->parent = NULL;
    return p;
}
```

BÀI TOÁN CÂY GIA PHẢ– MÃ GIẢ

- Thêm một node child vào con trái nhất của node parent trên cây và tìm kiếm theo tên

```
void addChild(Node* child, Node* parent){
    child.parent = parent

    if parent.leftMostChild is NULL then
        parent.leftMostChild = child
    else
        p = parent.leftMostChild
        while p.rightSibling is not NULL do
            p = p.rightSibling
        end while
        p.rightSibling = child
    end if}
```

```
Node* findNode(char* name){
     for i from 0 to n - 1 do
         if strcmp(nodes[i].name, name) equals 0
then
             return nodes[i]
         end if
    end for

    return NULL
}
```

BÀI TOÁN CÂY GIA PHẢ– MÃ GIẢ

- Tính số lượng hậu duệ (các con cháu) và số lượng thế hệ (độ sâu tối đa các cây con) của một node:

```
int countNodes(Node* nod){
if nod is NULL then
        return 0
    end if

    p = nod.leftMostChild
    cnt = 1

    while p is not NULL do
        cnt = cnt + countNodes(p)
        p = p.rightSibling
    end while
    return cnt
}
```

```
int height(Node* nod){
    if nod is NULL then
        return 0
    end if
    maxH = 0
    p = nod.leftMostChild
    while p is not NULL do
        h = height(p)
        if h > maxH then
            maxH = h
        end if
        p = p.rightSibling
    end while
    return maxH + 1}
```

BÀI TOÁN CÂY GIA PHẢ– CODE

- Thêm một node child vào con trái nhất của node parent trên cây và tìm kiếm theo tên

```
void addChild(Node* child, Node* parent){
    child->parent = parent;
    if(parent->leftMostChild == NULL){
        parent->leftMostChild = child;
    }else{
        Node* p = parent->leftMostChild;
        while(p->rightSibling != NULL)
            p = p->rightSibling;
        p->rightSibling = child;
    }
}
```

```
Node* findNode(char* name){
    for(int i = 0; i < n; i++){
        if(strcmp(nodes[i]->name,name) == 0){
            return nodes[i];
        }
    }
    return NULL;
}
```

BÀI TOÁN CÂY GIA PHẢ– CODE

- Tính số lượng hậu duệ (các con cháu) và số lượng thế hệ (độ sâu tối đa các cây con) của một node:

```
int countNodes(Node* nod){
    if(nod == NULL) return 0;
    Node* p = nod->leftMostChild;
    int cnt = 1;
    while(p != NULL){
        cnt += countNodes(p);
        p = p->rightSibling;
    }
    return cnt;
}
```

```
int height(Node* nod){
    if(nod == NULL) return 0;
    int maxH = 0;
    Node* p = nod->leftMostChild;
    while(p != NULL){
        int h = height(p);
        if(h > maxH) maxH = h;
        p = p->rightSibling;
    }
    return maxH + 1;
}
```

BÀI TOÁN CÂY GIA PHẢ– CODE

- Một số hàm triển khai:

```
void sovle(){
    while(1){
        char name1[MAX_LEN], name2[MAX_LEN];
        scanf("%s",name1); if(strcmp(name1,"***")
== 0) break;
        scanf("%s",name2);
        Node* n1 = findNode(name1);
        if(n1 == NULL){
            n++; nodes[n-1] = makeNode(name1); n1 =
nodes[n-1]; }
        Node* n2 = findNode(name2);
        if(n2 == NULL){
            n++; nodes[n-1] = makeNode(name2); n2 =
nodes[n-1];}
        addChild(n1,n2);
    }
```

```
while(1){
        char cmd[MAX_LEN]; scanf("%s",cmd);
        if(strcmp(cmd,"***")==0) break;
        if(strcmp(cmd,"descendants")==0){
            char param[MAX_LEN];
            scanf("%s",param);
            Node* nod = findNode(param);
            int ans = countNodes(nod);
            printf("%d\n",ans-1);
        }else if(strcmp(cmd,"generation")==0){
            char param[MAX_LEN];
            scanf("%s",param);
            Node* nod = findNode(param);
            int ans = height(nod);
            printf("%d\n",ans-1); }
    }
}
```

THANK YOU !