HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

C PROGRAMMING BASIC

C PROGRAMMING BASIC

TREE – PART 1

OUTLINE

- Tree manipulation query depth – height (P.04.09.01)
- Tree manipulation and traversal (P.04.09.02)
- Family Tree (P.04.09.03)

TREE MANIPULATION QUERY DEPTH – HEIGHT (P.04.09.01)

- Each node in a tree has a field called "id" (identifier), which is an integer (the ids of nodes in the tree are distinct). Perform a series of the following actions, including operations related to tree construction and traversal:
  - **MakeRoot u:** Create the root node with id u.
  - **Insert u v:** Create a new node u and insert it at the end of the list of children of node v (if the node with id v does not exist or the node with id u already exists, do not insert).
  - **Height u:** Calculate and return the height of node u.
  - **Depth u:** Calculate and return the depth of node u.
- It is known that there is only one MakeRoot command, and it always appears on the first line.
- **Input:** Consists of lines, each line formatted as described above, where the last line is marked by * (indicating the end of the input).
- **Output:** Write the result of each Height and Depth command, respectively, as read from the input.

TREE MANIPULATION QUERY DEPTH – HEIGHT

- Example: Input and 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

TREE MANIPULATION QUERY DEPTH – HEIGHT

- Data structure:

```
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;
```

- Create a new node with 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;
}
```

TREE MANIPULATION QUERY DEPTH – HEIGHT– PSEUDOCODE

- Insert a new node with the identifier (id) u as the leftmost child of the node with the identifier (id) v in the tree

```
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
}
```

TREE MANIPULATION QUERY DEPTH – HEIGHT– PSEUDOCODE

- Find the depth and height of a tree

```
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
}
```

TREE MANIPULATION AND TRAVERSAL (P.04.09.02)

- Each node of the tree has a unique integer identifier (id). Perform a series of the following actions, including operations related to tree construction and traversal:
    - **MakeRoot u:** Create the root node with id u.
    - **Insert u v:** Create a new node with id u and insert it at the end of the list of children of the node with id v.
    - **PreOrder:** Print the order of nodes during the pre-order tree traversal.
    - **InOrder:** Print the order of nodes during the in-order tree traversal.
    - **PostOrder:** Print the order of nodes during the post-order tree traversal.
- The input consists of lines, each line representing one of the described actions. The last line is marked by * to indicate the end of the data.
- The output should, on each line, display the order of nodes visited during the pre-order, in-order, and post-order traversals corresponding to the actions PreOrder, InOrder, PostOrder, respectively, as read from the input data.

TREE MANIPULATION AND TRAVERSAL

- Example: input and 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

TREE MANIPULATION AND TRAVERSAL

- Data structure:

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

- Create a new node with id =u:

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

TREE MANIPULATION AND TRAVERSAL - PSEUDOCODE

- Insert a new node with the identifier (id) u as the leftmost child of the node with the identifier (id) v in the tree.

```
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
}
```

TREE MANIPULATION AND TRAVERSAL - PSEUDOCODE

- Perform tree traversal in pre-order, in-order, and post-order.

```
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)
}
```

TREE MANIPULATION AND TRAVERSAL - PSEUDOCODE

- Insert a new node with the identifier (id) u as the leftmost child of the node with the identifier (id) v in the tree.

```
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;
}
```

FAMILY TREE (P.04.09.03)

- Given a family tree represented by child-parent (c,p) relations in which c is a child of p. Perform queries about the family tree:
  - descendants <name>: return number of descendants of the given <name>
  - generation <name>: return the number of generations of the descendants of the given <name>
- Note that: the total number of people in the family is less than or equal to 104
- **Input**
- Contains two blocks. The first block contains information about child-parent, including lines (terminated by a line containing ***), each line contains: <child> <parent> where <child> is a string represented the name of the child and <parent> is a string represented the name of the parent. The second block contains lines (terminated by a line containing ***), each line contains two string <cmd> and <param> where <cmd> is the command (which can be descendants or generation) and <param> is the given name of the person participating in the query.
- **Output**
- Each line is the result of a corresponding query.

FAMILY TREE

- Example: input and 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

FAMILY TREE

- Data structure:

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

- Create a new node with the parameter "name" passed into the function

```
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;
}
```

FAMILY TREE - PSEUDOCODE

- Insert a new child node as the leftmost child of the parent node in the tree and search by name.

```
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
}
```

FAMILY TREE - PSEUDOCODE

- Calculate the number of descendants (children and grandchildren) and the number of generations (maximum depth of child trees) of a 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}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !