HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

C Programming Basic

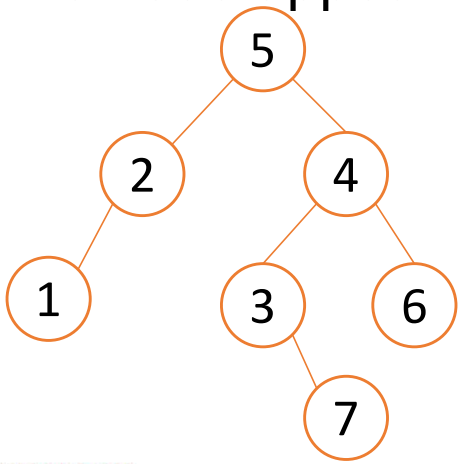
Trees – part 2

Manipulation with binary trees

- Each node of the binary tree has the following data structure

```
typedef struct Node{
    int id; // identifier of the node
    struct Node* leftChild;// pointer to the left child
    struct Node* rightChild;// pointer to the right child
}Node;
```

- The data of a binary tree is stored in an external text file with the format
  - Each line contains 3 integers *t, u, v* in which *u* and *v* (if different from -1) are the left child and the right child of *t* (***note***: the value *t* in each line (except line 1) is a child of some node appearing in previous lines)
  - The file is terminated with -2




```
5 2 4
2 1 -1
4 3 6
3 -1 7
-2
```

Manipulation with binary trees

- Write a program running in an interactive mode with commands
  - Load <filename>: load the data from <filename> to build a tree
  - Print: print the tree to the screen
  - AddLeftChild <cur_id> <child_id>: add a left child (if not exists) with identifier <child_id> to the node with identifier <cur_id> in the current tree if exists
  - AddRightChild <cur_id> <child_id>: add a right child (if not exists) with identifier <child_id> to the node with identifier <cur_id> in the current tree if exists
  - Find <id>: find the node having identifier <id>
  - Count: print number of nodes of the current tree
  - FindLeaves: print the leaf nodes of the current tree
  - Height <id>: print the height of the node with identifier <id> (if exists)
  - Store <filename>: store the tree to <filename>
  - Quit: terminate the program

Manipulation with binary trees

```
#include <stdio.h>
typedef struct Node{
    int id;
    struct Node* leftChild;
    struct Node* rightChild;
}Node;
Node* root;

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

Manipulation with binary trees

```
Node* find(Node* r, int id){
    if(r == NULL) return NULL;
    if(r->id == id) return r;
    Node* p = find(r->leftChild,id);
    if(p != NULL) return p;
    return find(r->rightChild,id);
}
void addLeftChild(int u, int left){
    Node* pu = find(root,u);
    if(pu == NULL){
        printf("Not found %d\n",u); return;
    }
    if(pu->leftChild != NULL){
        printf("Node %d has already leftChild\n",u); return;
    }
    pu->leftChild = makeNode(left);
}
```

Manipulation with binary trees

```
void addRightChild(int u, int right){
    Node* pu = find(root,u);
    if(pu == NULL){
        printf("Not found %d\n",u); return;
    }
    if(pu->rightChild != NULL){
        printf("Node %d has already rightChild\n",u); return;
    }
    pu->rightChild = makeNode(right);
}
```

Manipulation with binary trees

```
void load(char* filename){
    FILE* f = fopen(filename,"r");
    root = NULL;
    while(1){
        int u;
        fscanf(f,"%d",&u);
        if(u == -2) break;// termination
        if(root == NULL) root = makeNode(u);// create the root
        int l,r;
        fscanf(f,"%d%d",&l,&r);
        if(l > -1) addLeftChild(u,l);
        if(r > -1) addRightChild(u,r);
    }
    fclose(f);
}
```

Manipulation with binary trees

```
void printTree(Node* r){
    if(r == NULL) return;
    printf("%d: ",r->id);
    if(r->leftChild == NULL)  printf("leftChild = NULL");
    else printf("leftChild = %d",r->leftChild->id);
    if(r->rightChild == NULL)  printf(", rightChild = NULL");
    else printf(", rightChild = %d",r->rightChild->id);
    printf("\n");

    printTree(r->leftChild);
    printTree(r->rightChild);
}
```

Manipulation with binary trees

```
void printTreeF(Node* r, FILE* f){
    if(r == NULL) return;
    fprintf(f,"%d ",r->id);
    if(r->leftChild == NULL)  fprintf(f,"-1 ");
    else fprintf(f,"%d ",r->leftChild->id);
    if(r->rightChild == NULL)  fprintf(f,"-1 ");
    else fprintf(f,"%d ",r->rightChild->id);
    fprintf(f,"\n");

    printTreeF(r->leftChild,f);
    printTreeF(r->rightChild,f);
}
```

Manipulation with binary trees

```
void processLoad(){
    char filename[256];
    scanf("%s",filename);
    load(filename);
}
void printChildren(Node* p){
    if(p->leftChild == NULL) printf(" Node %d does not has leftChild",p->id);
    else printf(", LeftChild = %d",p->leftChild->id);
    if(p->rightChild == NULL) printf(" Node %d does not has rightChild\n",p->id);
    else printf(", RightChild = %d\n",p->rightChild->id);
}
```

Manipulation with binary trees

```
void processFind(){
    int id;
    scanf("%d",&id);
    Node* p = find(root,id);
    if(p == NULL) printf("Not found %d\n",id);
    else {
            printf("Found node %d: ",id);
            printChildren(p);
    }
}
void processPrint(){
    printTree(root);
}
```

Manipulation with binary trees

```
void processAddLeftChild(){
    int id,u;
    scanf("%d%d",&id,&u);
    addLeftChild(id,u);
}
void processAddRightChild(){
    int id,u;
    scanf("%d%d",&id,&u);
    addRightChild(id,u);
}
```

Manipulation with binary trees

```
int height(Node* p){
    if(p == NULL) return 0;
    int maxH = 0;
    int hl = height(p->leftChild);
    if(maxH < hl) maxH = hl;
    int hr = height(p->rightChild);
    if(maxH < hr) maxH = hr;
    return maxH + 1;
}
void processHeight(){
    int id;
    scanf("%d",&id);
    Node* p = find(root,id);
    if(p == NULL) printf("Not found %d\n",id);
    else printf("Height of %d is %d\n",height(p));
}
```

Manipulation with binary trees

```
int count(Node* p){
    if(p == NULL) return 0;
    return 1 + count(p->leftChild) + count(p->rightChild);
}
void printLeaves(Node* p){
    if(p == NULL) return;
    if(p->leftChild == NULL && p->rightChild == NULL)
        printf("%d ",p->id);
    printLeaves(p->leftChild);
    printLeaves(p->rightChild);
}
void processFindLeaves(){
    printLeaves(root); printf("\n");
}
```

Manipulation with binary trees

```
void processCount(){
    printf("Number of nodes = %d\n",count(root));
}
void processStore(){
    char filename[256];
    scanf("%s",filename);
    FILE* f = fopen(filename,"w");
    printTreeF(root,f);
    fprintf(f,"-2");
    fclose(f);
}
void freeTree(Node* r){
    if(r == NULL) return;
    freeTree(r->leftChild);
    freeTree(r->rightChild);
    free(r); r = NULL;
}
```

Manipulation with binary trees

```
void main(){
    while(1){
        char cmd[256]; // representing the input command
        printf("Enter a command: ");
        scanf("%s",cmd);
        if(strcmp(cmd,"Quit") == 0) break;
        else if(strcmp(cmd,"Load")==0) processLoad();
        else if(strcmp(cmd,"Print")==0) processPrint();
        else if(strcmp(cmd,"Find")==0) processFind();
        else if(strcmp(cmd,"Height")==0) processHeight();
        else if(strcmp(cmd,"Count")==0) processCount();
        else if(strcmp(cmd,"FindLeaves")==0) processFindLeaves();
        else if(strcmp(cmd,"AddLeftChild")==0) processAddLeftChild();
        else if(strcmp(cmd,"AddRightChild")==0) processAddRightChild();
        else if(strcmp(cmd,"Store")==0) processStore();
    }
    freeTree(root);
}
```

Manipulation with binary trees

- **Exercise** Each node of a binary tree has a field id which is the identifier of the node. Build a binary tree and check if the tree is a balanced tree, compute the height of the given tree
- **Input**
  - Line 1 contains MakeRoot u: make the root of the tree having id = u
  - Each subsequent line contains: AddLeft or AddRight commands with the format
    - AddLeft u v: create a node having id = u, add this node as a left-child of the node with id = v (if not exists)
    - AddRight u v: create a node having id = u, add this node as a right-child of the node with id = v (if not exists)
  - The last line contains * which marks the end of the input
- **Output**
  - Write two integer z and h in which h is the height (the number of nodes of the longest path from the root to a leaf) and z = 1 if the tree is balanced and z = 0, otherwise

Manipulation with binary trees

- **Example**

Input	Output
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

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

soict.hust.edu.vn/ fb.com/groups/soict