# 实验三

## 1. 问题定义及需求分析

### 1.1 问题描述

##### 互联网域名系统是一个典型的树形层次结构。从根节点往下的第一层是顶层域，如cn、com等，最底层（第四层）是叶子结点，如www等。因此，域名搜索可以看成是树的遍历问题。

### 1.2 实验要求

##### 设计搜索互联网域名的程序。

###### （1）采用树的孩子兄弟链表等存储结构。

###### （2）创建树形结构。

###### （3）通过深度优先遍历搜索。

###### （4）通过层次优先遍历搜索。

### 1.3 输入

#### 1.3.1 输入的形式

##### 域名字符串，包含英文大小写和数字或单独数字

### 1.4 输出形式

##### 字符串，包含英文大小写或数字

### 1.5 程序的功能

##### 采用孩子兄弟链表存储结构完成二叉树的创建，并且能实现添加、查找、深度优先遍历、层次优先遍历

### 1.6 测试数据

#### 1.6.1 输入

##### 1 www.baidu.com 1 www.google.com 1 neu.edu.cn 3

#### 1.6.2 输出

##### www.baidu.com

##### www.google.com

##### neu.edu.cn

## 2.概要设计

### 2.1 抽象数据类型定义

##### ADT Tree

##### {

##### Data 字符串

##### 基本操作 int FindLastDot(const char \*URL) //定位最后一个dot位置

##### TreeNode AddChild(TreeNode root,const char \*data) //建立新的孩子结点

##### void Add(TreeNode root,const char URL) //添加新的数据元素

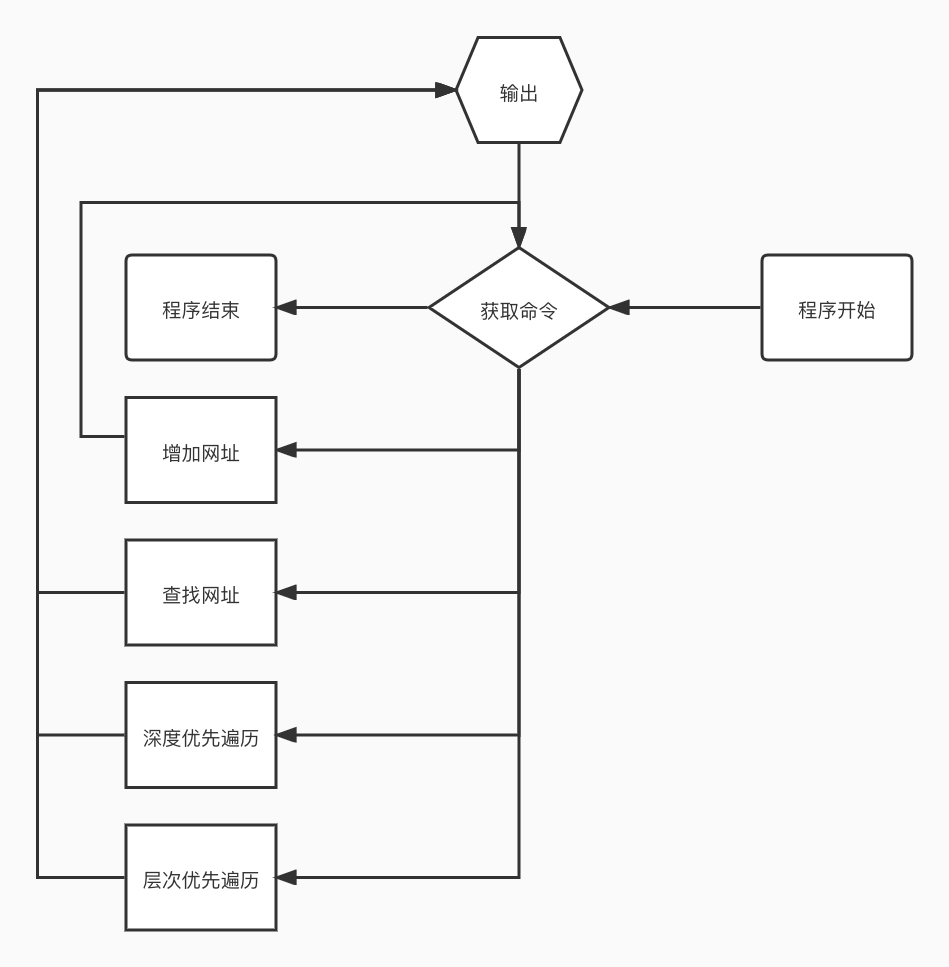
##### TreeNode\* Find(TreeNode root,const char URL) //查询数据元素

##### void DFS(TreeNode root,const char URL) //深度优先遍历

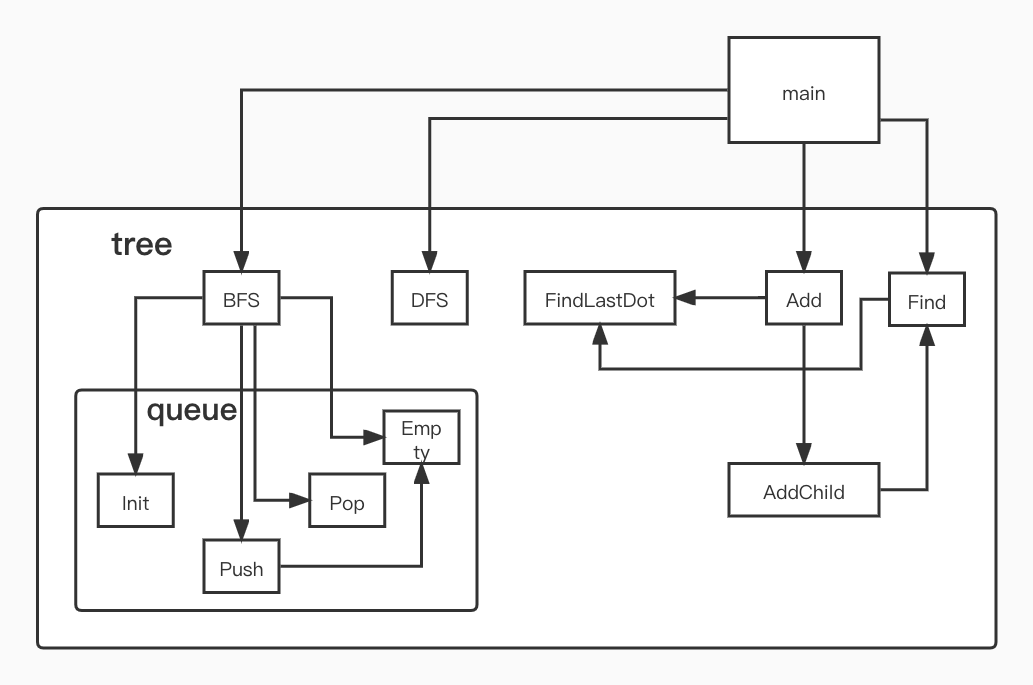
##### void BFS(TreeNode \*root) //层次优先遍历

##### }

### 2.2 主程序流程



### 2.3 模块调用关系



## 3.详细设计

### 3.1 定义数据类型及存储结构

typedef struct TREENODE  
{  
 char data[max\_string\_length];  
 struct TREENODE \*firstchild,\*nextsibling;  
 int terminate;  
}TreeNode;//节点结构  
typedef struct  
{  
 void\* data[max\_elem\_cnt];  
 int front,end;  
}Queue;//队列结构

### 3.2 每个函数及操作的代码

#### 3.2.1 队列相关

void Init(Queue\* Q)  
{  
 Q->front=1;  
 Q->end=0;  
}  
int Push(Queue\* Q,void \*root)  
{  
 if(Q->end+1>=max\_elem\_cnt)  
 return ERROR;  
 ++Q->end;  
 Q->data[Q->end]=root;  
 return OK;  
}  
void \*Pop(Queue\* Q)  
{  
 if(Empty(Q))return NULL;  
 ++Q->front;  
 return Q->data[Q->front-1];  
}  
int Empty(Queue\* Q)  
{  
 if(Q->front>Q->end)return 1;  
 return 0;  
}

#### 3.2.2 树相关

int FindLastDot(const char \*URL) //定位最后一个dot位置  
{  
 for(int i=strlen(URL);i>=0;--i)  
 if(URL[i]=='.')return i;  
 return -1;  
}   
TreeNode \*AddChild(TreeNode \*root,const char \*data) //建立新的孩子结点  
{  
 printf(" data: %s\n",data);  
 TreeNode\* NewNode=Find(root,data); //寻找添加的域名  
 if(NewNode==NULL) //不存在则添加  
 {  
 NewNode=(TreeNode\*)malloc(sizeof(TreeNode));  
 strcpy(NewNode->data,data);   
 NewNode->firstchild=NULL;  
 NewNode->nextsibling=NULL;  
 NewNode->terminate=0;   
 if(root->firstchild) //根节点有孩子  
 {  
 TreeNode \*NowNode=root->firstchild;   
 while(NowNode->nextsibling)   
 NowNode=NowNode->nextsibling;  
 NowNode->nextsibling=NewNode; //新结点加入根节点的孩子结点的兄弟  
 }  
 else //根节点无孩子  
 {  
 root->firstchild=NewNode; //作为根节点的孩子  
 }  
 }  
 return NewNode;  
}  
void Add(TreeNode \*root,const char \*URL) //添加新的数据元素  
{  
 printf("Adding:\n remaining\_url: %s\n",URL);  
 int pos=FindLastDot(URL);  
 char NextURL[max\_string\_length];  
 if(pos!=-1) //域名中存在dot  
 {  
 TreeNode \*child=AddChild(root,URL+pos+1);//将dot后的域名作为孩子插入  
 memcpy(NextURL,URL,pos\*sizeof(char));   
 NextURL[pos]='\0';  
 Add(child,NextURL); //递归插入域名  
 }  
 else //不存在dot  
 {  
 TreeNode \*child=AddChild(root,URL+pos+1);  
 child->terminate=1;   
 }  
}  
TreeNode\* Find(TreeNode \*root,const char \*URL) //查询数据元素  
{  
 int pos=FindLastDot(URL);   
 char NextURL[max\_string\_length];  
 if(pos!=-1) //链接中有dot  
 {  
 TreeNode \*NowNode=root->firstchild;   
 while(NowNode)  
 {  
 if(strcmp(NowNode->data,URL+pos+1)==0)   
 {  
 memcpy(NextURL,URL,pos\*sizeof(char));  
 NextURL[pos]='\0';  
 return Find(NowNode,NextURL);  
 }  
 NowNode=NowNode->nextsibling; //遍历兄弟寻找dot后字符串的元素  
 }   
 return NULL;  
 }  
 else //输入链接无dot  
 {  
 TreeNode \*NowNode=root->firstchild;   
 while(NowNode)  
 {  
 if(strcmp(NowNode->data,URL)==0)  
 {  
 return NowNode;  
 }  
 NowNode=NowNode->nextsibling; //遍历兄弟寻找输入字符串中的元素  
 }  
 return NULL;  
 }  
}  
void DFS(TreeNode \*root,const char \*URL) //深度优先遍历  
{  
 char NewURL[max\_string\_length];  
 if(strlen(URL)==0)  
 strcpy(NewURL,root->data);  
 else  
 sprintf(NewURL,"%s.%s",root->data,URL);  
 if(root->terminate)  
 printf("%s\n",NewURL);  
 TreeNode \*child=root->firstchild;  
 while(child)  
 {  
 DFS(child,NewURL);  
 child=child->nextsibling; //递归遍历兄弟节点  
 }  
}  
void BFS(TreeNode \*root) //层次优先遍历  
{  
 Queue \*q=(Queue\*)malloc(sizeof(Queue));  
 Init(q);  
 Push(q,root); //根节点入队列  
 int lvl=0;  
 int now\_lvl\_cnt=1;  
 int next\_lvl\_cnt=0;  
 while(!Empty(q)) //队列不空时执行  
 {  
 TreeNode \*NowNode=(TreeNode\*)Pop(q); //取队列首元素  
 printf("%s ",NowNode->data);--now\_lvl\_cnt;  
 TreeNode \*child=NowNode->firstchild;  
 while(child)   
 {  
 ++next\_lvl\_cnt;  
 Push(q,child);  
 child=child->nextsibling;  
 } //出队列节点孩子不空则入队列  
 if(now\_lvl\_cnt==0&&next\_lvl\_cnt)  
 {  
 printf("\n---level:%d---\n",++lvl);  
 now\_lvl\_cnt=next\_lvl\_cnt;  
 next\_lvl\_cnt=0;  
 }  
 }  
 free(q);  
}

## 4.调试分析

### 4.1 遇到的问题及分析

#### 4.1.1 问题

##### 在编写层次优先遍历算法的时候遍历结果总是不正确。

#### 4.1.2 分析

##### 原因是取完队首元素后没有将出队列，经过改正，在取队首元素后加上出队列函数将队首元素出队；这样便解决了问题。

### 4.2 算法时空分析

##### 经过孩子兄弟链表表示的树创建后便得到一棵二叉树；对于两个遍历函数，深度优先遍历是递归算法，在时间上，递归算法效率较低，尤其是运算次数较大的时候；层次优先遍历函数借助到队列，所以在内存占用上较多；而深度优先遍历算法的空间占用上更优于层次优先遍历；

### 4.3 经验和体会

##### 孩子兄弟链表表示的树与二叉树可以相互转化；它的深度优先遍历递归算法虽较为简单但相比非递归算法而言效率不高。

## 5.使用说明

##### (1)在主界面选择add选项，按照提示依次输入地址，程序内部自动完成树上结点的添加

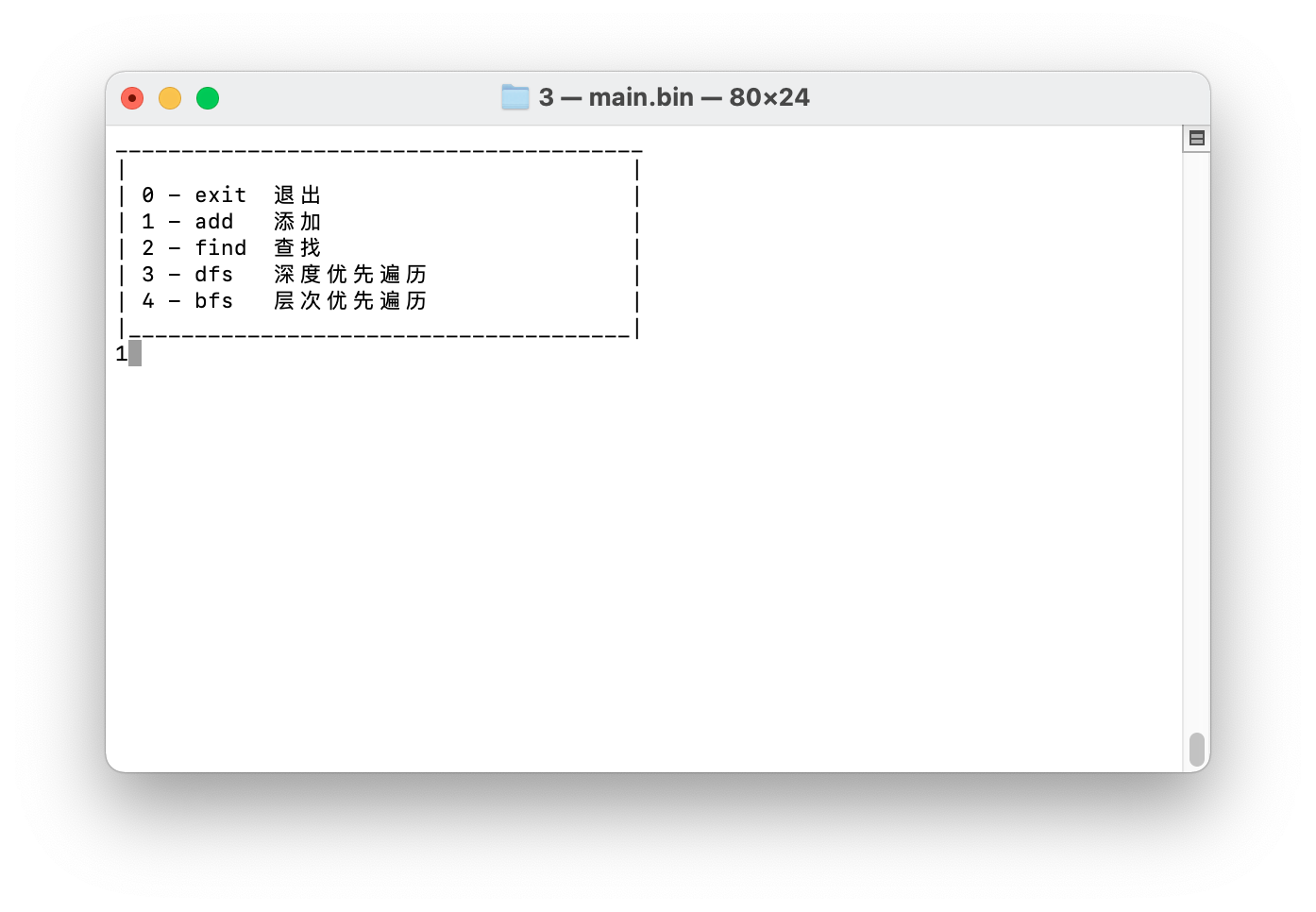
##### (2)可以选择：

###### (a)查找某个地址是否存在

###### (b)按照深度优先遍历输出域名

###### (c)按照层次优先遍历输出域名

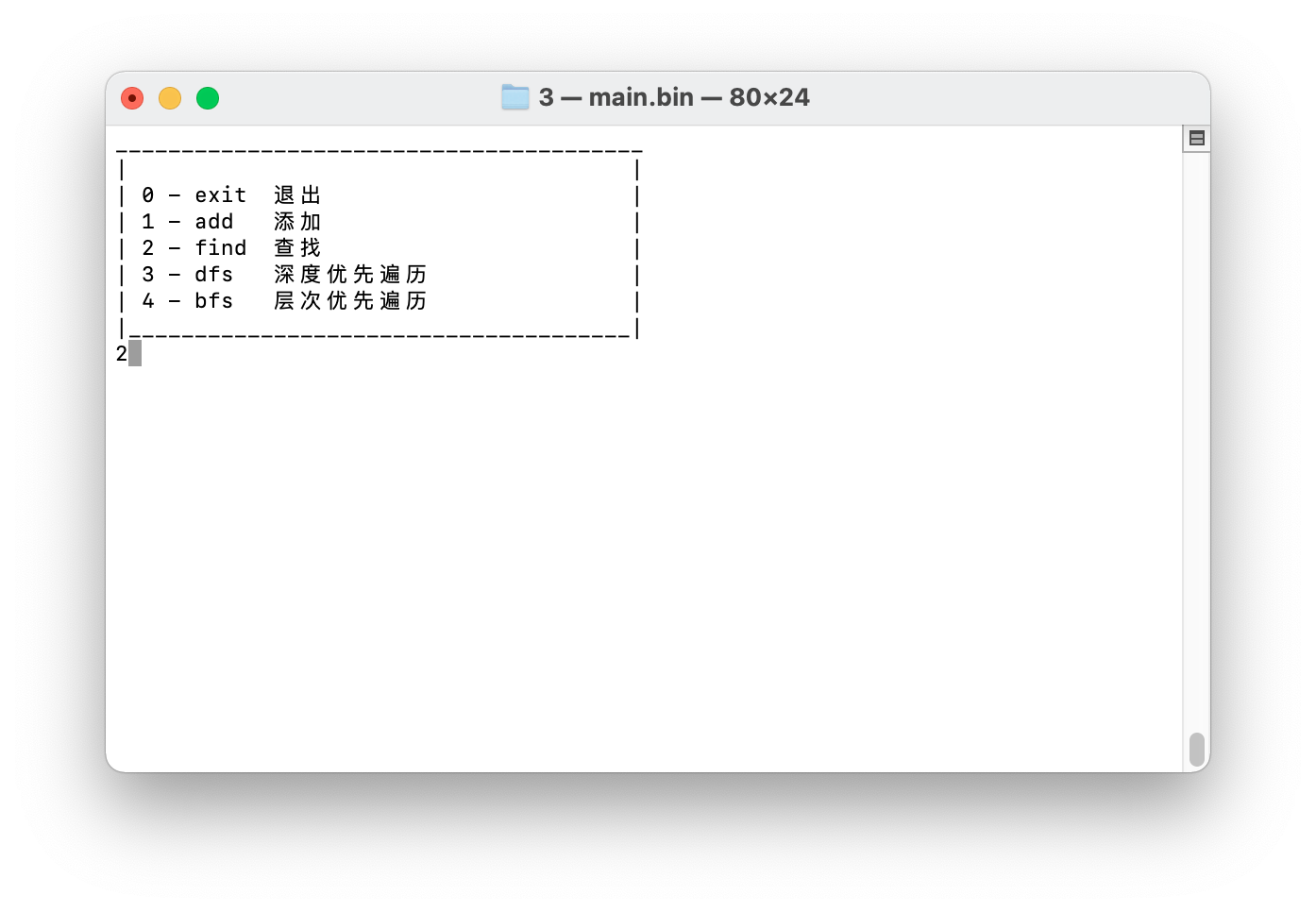
## 6.测试结果

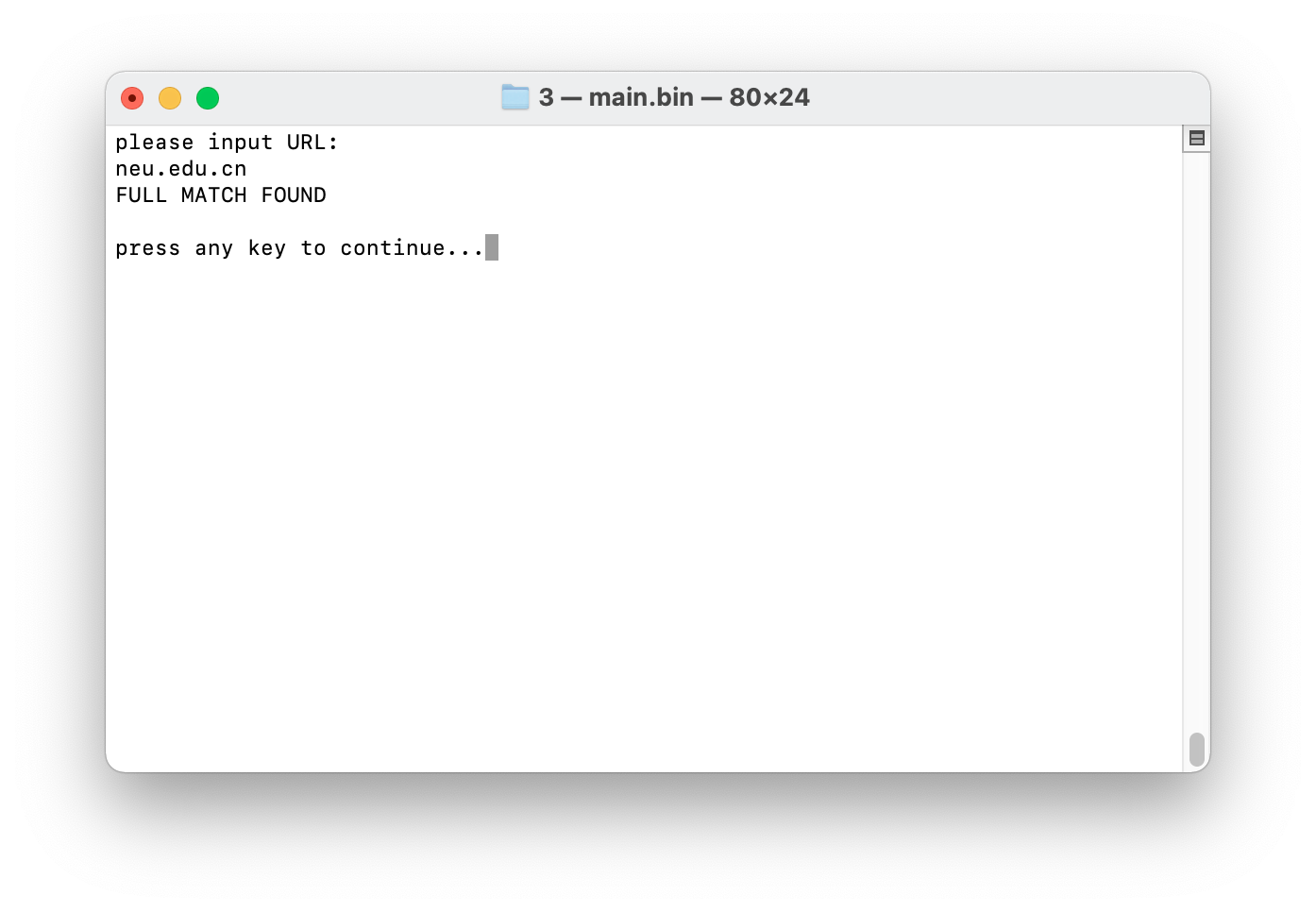




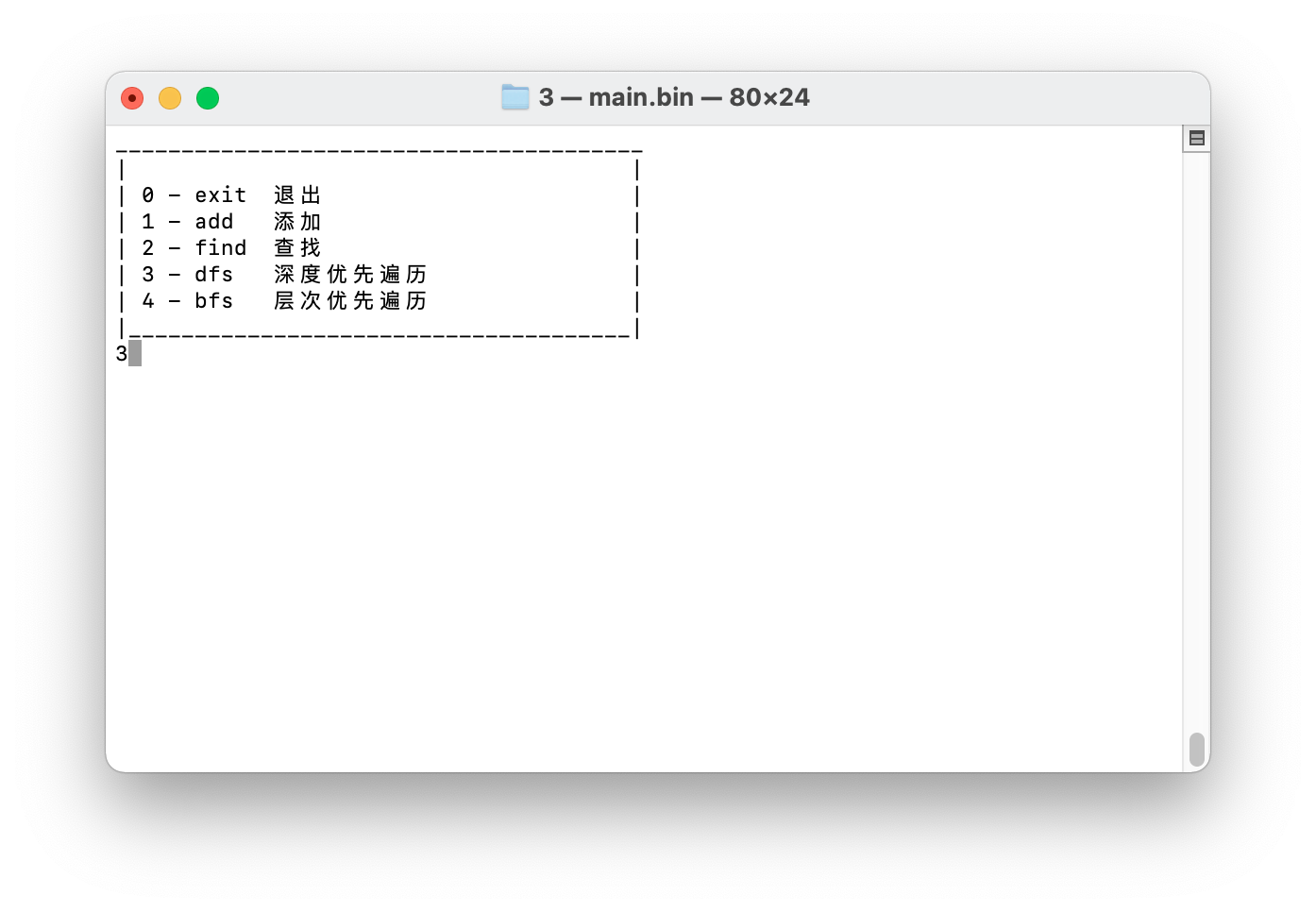




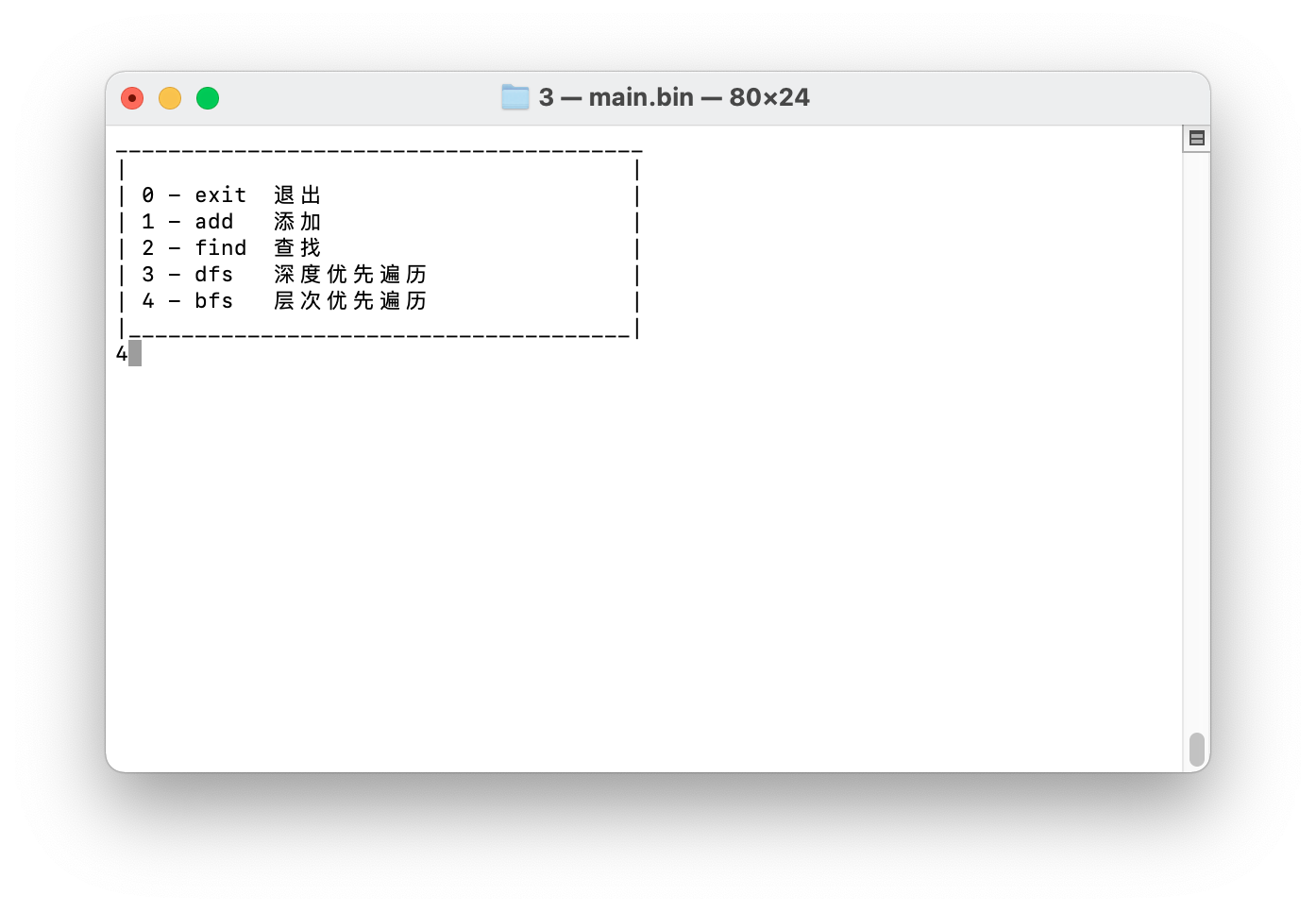


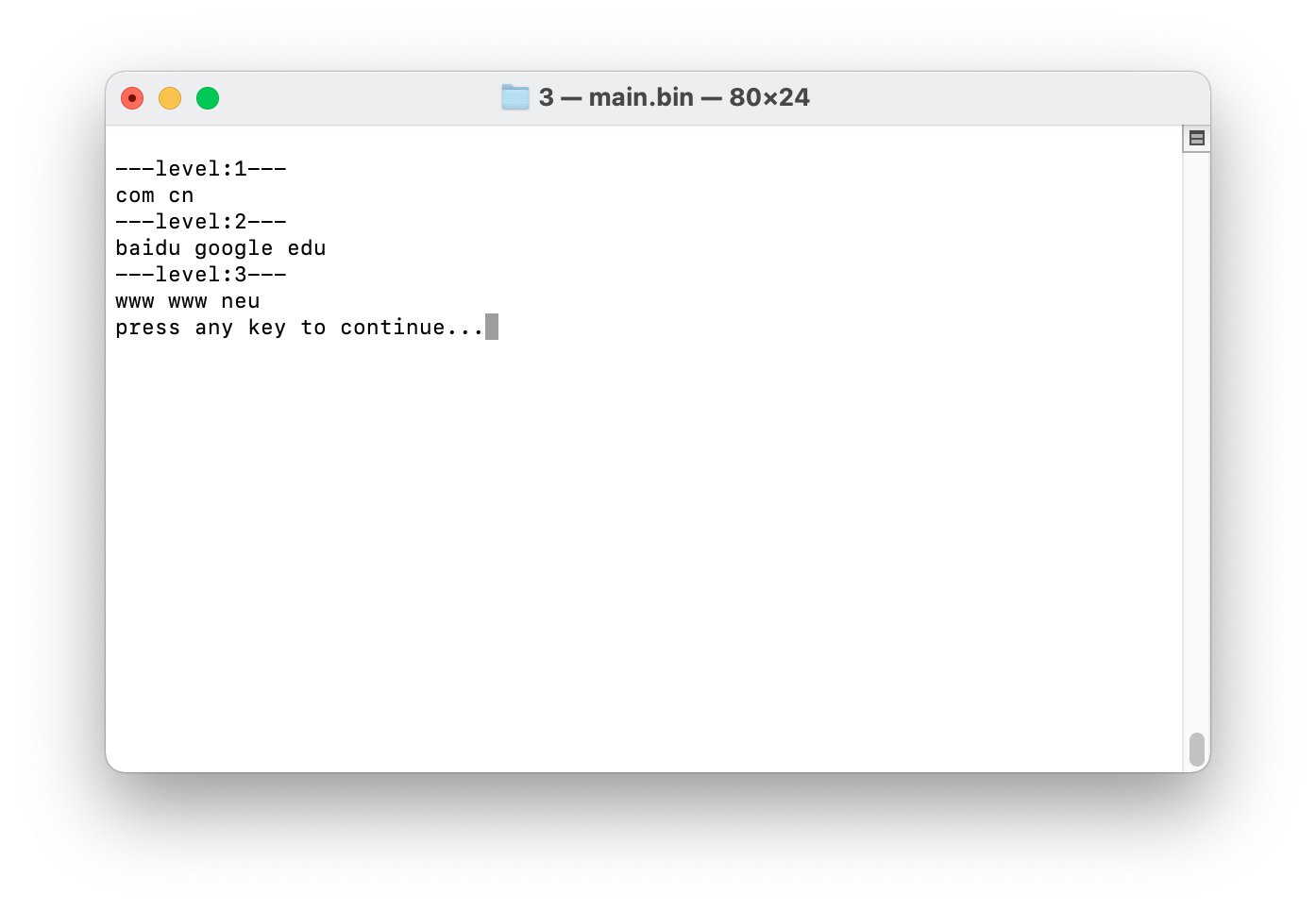












## 7.附录

### 7.1 个人负责的部分

##### 与树有关的遍历、查找操作。

### 7.2 整个程序

#### 7.2.0 makefile

main : main.c queue.c tree.c   
 gcc -Wall -O2 -o main.bin main.c queue.c tree.c

#### 7.2.1 universal\_header.h

#ifndef UNIVERSAL\_H  
#define UNIVERSAL\_H  
  
#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
#define max\_string\_length 100  
#define max\_elem\_cnt 100  
#define ERROR 1  
#define OK 0  
  
#endif // UNIVERSAL\_H

#### 7.2.2 queue.h

#ifndef QUEUE\_H  
#define QUEUE\_H  
  
#include"universal\_header.h"  
typedef struct  
{  
 void\* data[max\_elem\_cnt];  
 int front,end;  
}Queue;//队列结构  
void Init(Queue\*);  
int Push(Queue\*,void\*);  
void \*Pop(Queue\*);  
int Empty(Queue\*);  
  
#endif

#### 7.2.3 queue.c

#include"queue.h"  
void Init(Queue\* Q)  
{  
 Q->front=1;  
 Q->end=0;  
}  
int Push(Queue\* Q,void \*root)  
{  
 if(Q->end+1>=max\_elem\_cnt)  
 return ERROR;  
 ++Q->end;  
 Q->data[Q->end]=root;  
 return OK;  
}  
void \*Pop(Queue\* Q)  
{  
 if(Empty(Q))return NULL;  
 ++Q->front;  
 return Q->data[Q->front-1];  
}  
int Empty(Queue\* Q)  
{  
 if(Q->front>Q->end)return 1;  
 return 0;  
}

#### 7.2.4 tree.h

#ifndef QUEUE\_H  
#define QUEUE\_H  
  
#include"universal\_header.h"  
typedef struct  
{  
 void\* data[max\_elem\_cnt];  
 int front,end;  
}Queue;//队列结构  
void Init(Queue\*);  
int Push(Queue\*,void\*);  
void \*Pop(Queue\*);  
int Empty(Queue\*);  
  
#endif

#### 7.2.5 tree.c

#include"tree.h"  
int FindLastDot(const char \*URL) //定位最后一个dot位置  
{  
 for(int i=strlen(URL);i>=0;--i)  
 if(URL[i]=='.')return i;  
 return -1;  
}   
TreeNode \*AddChild(TreeNode \*root,const char \*data) //建立新的孩子结点  
{  
 printf(" data: %s\n",data);  
 TreeNode\* NewNode=Find(root,data); //寻找添加的域名  
 if(NewNode==NULL) //不存在则添加  
 {  
 NewNode=(TreeNode\*)malloc(sizeof(TreeNode));  
 strcpy(NewNode->data,data);   
 NewNode->firstchild=NULL;  
 NewNode->nextsibling=NULL;  
 NewNode->terminate=0;   
 if(root->firstchild) //根节点有孩子  
 {  
 TreeNode \*NowNode=root->firstchild;   
 while(NowNode->nextsibling)   
 NowNode=NowNode->nextsibling;  
 NowNode->nextsibling=NewNode; //新结点加入根节点的孩子结点的兄弟  
 }  
 else //根节点无孩子  
 {  
 root->firstchild=NewNode; //作为根节点的孩子  
 }  
 }  
 return NewNode;  
}  
void Add(TreeNode \*root,const char \*URL) //添加新的数据元素  
{  
 printf("Adding:\n remaining\_url: %s\n",URL);  
 int pos=FindLastDot(URL);  
 char NextURL[max\_string\_length];  
 if(pos!=-1) //域名中存在dot  
 {  
 TreeNode \*child=AddChild(root,URL+pos+1);//将dot后的域名作为孩子插入  
 memcpy(NextURL,URL,pos\*sizeof(char));   
 NextURL[pos]='\0';  
 Add(child,NextURL); //递归插入域名  
 }  
 else //不存在dot  
 {  
 TreeNode \*child=AddChild(root,URL+pos+1);  
 child->terminate=1;   
 }  
}  
TreeNode\* Find(TreeNode \*root,const char \*URL) //查询数据元素  
{  
 int pos=FindLastDot(URL);   
 char NextURL[max\_string\_length];  
 if(pos!=-1) //链接中有dot  
 {  
 TreeNode \*NowNode=root->firstchild;   
 while(NowNode)  
 {  
 if(strcmp(NowNode->data,URL+pos+1)==0)   
 {  
 memcpy(NextURL,URL,pos\*sizeof(char));  
 NextURL[pos]='\0';  
 return Find(NowNode,NextURL);  
 }  
 NowNode=NowNode->nextsibling; //遍历兄弟寻找dot后字符串的元素  
 }   
 return NULL;  
 }  
 else //输入链接无dot  
 {  
 TreeNode \*NowNode=root->firstchild;   
 while(NowNode)  
 {  
 if(strcmp(NowNode->data,URL)==0)  
 {  
 return NowNode;  
 }  
 NowNode=NowNode->nextsibling; //遍历兄弟寻找输入字符串中的元素  
 }  
 return NULL;  
 }  
}  
void DFS(TreeNode \*root,const char \*URL) //深度优先遍历  
{  
 char NewURL[max\_string\_length];  
 if(strlen(URL)==0)  
 strcpy(NewURL,root->data);  
 else  
 sprintf(NewURL,"%s.%s",root->data,URL);  
 if(root->terminate)  
 printf("%s\n",NewURL);  
 TreeNode \*child=root->firstchild;  
 while(child)  
 {  
 DFS(child,NewURL);  
 child=child->nextsibling; //递归遍历兄弟节点  
 }  
}  
void BFS(TreeNode \*root) //层次优先遍历  
{  
 Queue \*q=(Queue\*)malloc(sizeof(Queue));  
 Init(q);  
 Push(q,root); //根节点入队列  
 int lvl=0;  
 int now\_lvl\_cnt=1;  
 int next\_lvl\_cnt=0;  
 while(!Empty(q)) //队列不空时执行  
 {  
 TreeNode \*NowNode=(TreeNode\*)Pop(q); //取队列首元素  
 printf("%s ",NowNode->data);--now\_lvl\_cnt;  
 TreeNode \*child=NowNode->firstchild;  
 while(child)   
 {  
 ++next\_lvl\_cnt;  
 Push(q,child);  
 child=child->nextsibling;  
 } //出队列节点孩子不空则入队列  
 if(now\_lvl\_cnt==0&&next\_lvl\_cnt)  
 {  
 printf("\n---level:%d---\n",++lvl);  
 now\_lvl\_cnt=next\_lvl\_cnt;  
 next\_lvl\_cnt=0;  
 }  
 }  
 free(q);  
}

#### 7.2.6 main.h

#ifndef MAIN\_H  
#define MAIN\_H  
  
#include"universal\_header.h"  
#include"tree.h"  
  
#endif // MAIN\_H

#### 7.2.7 main.c

#include"main.h"  
void ClearWindow()  
{  
 #if defined(\_\_APPLE\_\_)||defined(\_\_linux\_\_)  
 system("clear");  
 #else  
 system("cls");  
 #endif  
}  
void ShowWindow()  
{  
 ClearWindow();  
 printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  
 printf("| |\n");  
 printf("| 0 - exit 退出 |\n");  
 printf("| 1 - add 添加 |\n");  
 printf("| 2 - find 查找 |\n");  
 printf("| 3 - dfs 深度优先遍历 |\n");  
 printf("| 4 - bfs 层次优先遍历 |\n");  
 printf("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");  
}  
int main()  
{  
 int cmd;  
 char buf[max\_string\_length];  
 TreeNode \*root=(TreeNode\*)malloc(sizeof(TreeNode));  
 root->data[0]='\0';  
 root->firstchild=NULL;  
 root->nextsibling=NULL;  
 root->terminate=0;  
  
 ShowWindow();  
 while(scanf("%d",&cmd),cmd)  
 {  
 ClearWindow();  
 switch(cmd)  
 {  
 case 1:  
 {  
 printf("please input URL:\n");  
 scanf("%s",buf);  
 Add(root,buf);  
 break;  
 }  
 case 2:  
 {  
 printf("please input URL:\n");  
 scanf("%s",buf);  
 TreeNode \*res=Find(root,buf);  
 if(res)  
 {  
 if(res->terminate)  
 printf("FULL MATCH ");  
 printf("FOUND\n");  
 }  
 else  
 printf("NOTFOUND\n");  
 break;  
 }  
 case 3:  
 {  
 DFS(root,"");  
 break;  
 }  
 case 4:  
 {  
 BFS(root);  
 break;  
 }  
 default:  
 break;  
 }  
 printf("\npress any key to continue...");  
 getchar();getchar();  
 ShowWindow();  
 }  
}