**实 验 报 告 （ 2 ）**

|  |  |
| --- | --- |
| **实验名称**：树的应用 | **实验地点**：宿舍，信息楼211 |
| **所使用的工具软件及环境：**  windows 11版本 10.0.22631 版本 22631:  GNU gdb (GDB) 7.6.1,  gcc version 6.3.0 (MinGW.org GCC-6.3.0-1) win32,  Vscode 1.94.2  Ubuntu 24.04 LTS :  gcc version 13.2.0 (Ubuntu 13.2.0-23ubuntu4) posix,  GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git,  valgrind-3.22.0 | |
| **一、实验目的：**  1、掌握二叉树的结构特征，以及各种存储结构的特点及使用范围。  2、掌握用指针类型描述、访问和处理二叉树的运算。  3、掌握树的应用算法。 | |
| **二、实验内容及要求：**（填写题目内容及输入输出要求）  **1. 编写程序，根据给定的一棵二叉树的后序遍历和中序遍历结果：**  **（1）输出该二叉树的前序遍历结果；**  **（2）判断给定的二叉树是否是二叉搜索树。**  **1）设计思路**  //构造二叉树   1. 利用scanf读取后序和中序遍历结果、二叉树结点个数。 2. 在后续遍历中找到根节点。 3. 从中序遍历中找到根结点，切出左右子树，对根节点所在位置（数组下标）进行标记 4. 再重复2.3步骤在左右子树中找到根节点和其孩子。   //前序遍历   1. 先访问根节点数据 2. 利用递归相继访问左子树和右子树的数据   //判断BST   1. 利用递归进行中序遍历，并在中序遍历过程中时刻判断中序遍历序列是否是递增序列 2. 如果中序遍历完之后仍然是递增序列，则表明是BST二叉搜索树   //其他设计   1. 人性化的输入输出，以空格分隔序列的输入 2. 添加了程序错误运行日志，并将其重定向到txt文件（利用freopen） 3. 完善了内存回收机制并利用valgrind进行内存检查，确保程序无内存泄漏，利用gdb进行调试 4. 利用const进行常量定义而不是用宏定义。因为编译器不会对宏定义进行类型检查，只做纯粹的代换，但是会对const修饰的变量进行类型检查。 5. 对程序进行模块化设计，将声明统一写在头文件内并加以详细注释（使头文件有较好的描述数据结构的效果），定义统一写在源文件内，变量命名使用下划线法增加可读性。   **2）数据结构的描述** //bintree.h  #pragma once  #include "err.h"  typedef int ElementType;    //元素类型  typedef struct TNode \*position;  typedef position BinTree;      //二叉树  struct TNode{       //二叉树节点      ElementType data;       //节点数据      BinTree left;       //指向左子树      BinTree right;       //指向右子树  };  /\*  @brief 根据中序遍历和后序遍历创建二叉树  @param int[] 中序遍历  @param int[] 后序遍历  @param int  最大结点个数  23010341  @return Bintree 创建完成的BinTree  \*/  BinTree build\_tree(int inorder[],int postorder[],int N);  /\*  @brief 先序遍历二叉树  @param BinTree 二叉树表头  @return None  \*/  void pre\_order\_traversal(BinTree T);  /\*  @brief 清空二叉树   23010341  @param BinTree 二叉树表头  @return None  \*/  void free\_bt(BinTree T);  /\*  @brief 判断二叉树是否是二叉搜索树  @param BinTree 二叉树表头  @param ElementType  根节点数据min,max  @return bool  \*/  bool isBST(BinTree T,ElementType \*min,ElementType \*max);  /\*  @brief 前序创建二叉树  @param int[]   前序序列  @param int\*  节点个数  @return BinTree 返回二叉树表头  \*/  BinTree pre\_build\_tree(int preorder[],int \*N);    /\*  @brief 生成每层的结点数(宽度)  @param BinTree 二叉树  @param int[] 当前层书拥有的结点数  \*/  void MaxWidth(BinTree T,int width[]);       //优化了 i  没必要;可以直接操作width指针  /\*  @brief 获取二叉树宽度  @param Bintree 二叉树  @return int 二叉树宽度  \*/  int get\_width(BinTree T);  /\*  @brief 获取二叉树高度  @param BinTree 二叉树  @return int 二叉树高度  \*/  int get\_height(BinTree T);  **3）程序代码**  //bintree.c  #include "bintree.h"  BinTree build\_tree(int inorder[], int postorder[], int N)  {    BinTree T;    int p;    if (!N)      return NULL;                             // 终止    T = (BinTree)malloc(sizeof(struct TNode)); // 分配空间    if (!T)    {      print\_err("build\_tree() malloc failed.");      exit(-1);    }    T->data = postorder[N - 1];    T->left = T->right = NULL;    for (p = 0; p < N; p++)      if (inorder[p] == postorder[N - 1])        break; // 在中序中寻找根节点    T->left = build\_tree(inorder, postorder, p);    T->right = build\_tree(inorder + p + 1, postorder + p, N - p - 1);    return T;  }  void pre\_order\_traversal(BinTree T)  {    if (T)    {      printf("%d ", T->data);        // 访问节点数据      pre\_order\_traversal(T->left);  // 访问左子树      pre\_order\_traversal(T->right); // 访问右子树    }    return;  }  void free\_bt(BinTree T) // 利用2个工作指针,利用递归先完成左右子树释放再释放根节点  {    BinTree left = T->left, right = T->right; // 获取左右子树    if (left)                                 // 左子树不为空    {      free\_bt(left); // 释放左子树    }    if (right) // 右子树不为空    {      free\_bt(right);    }    free(T); // 左右子树都空 释放当前节点    return;  // 到头  }  bool isBST(BinTree T, int \*min, int \*max)  {    int lmin, lmax, rmin, rmax, Left\_flag, Right\_flag;    if (!T)      return true;    if (!T->left && !T->right)    {      (\*min) = (\*max) = T->data;      return true;    }    Left\_flag = Right\_flag = false;    if ((T->left &&         isBST(T->left, &lmin, &lmax) &&         T->data > lmax) ||        !T->left)      Left\_flag = true;    if ((T->right &&         isBST(T->right, &rmin, &rmax) &&         T->data < rmin) ||        !T->right)      Right\_flag = true;    if (Left\_flag && Right\_flag)    {      if (T->left)        (\*min) = lmin;      else        (\*min) = T->data;      if (T->right)        (\*max) = rmax;      else        (\*max) = T->data;      return true;    }    else      return false;  }  BinTree pre\_build\_tree(int preorder[], int \*N)  {    //ABDE000F00CG000    BinTree bt;    if(!(\*N)) return NULL;   //创建完毕    if (\*preorder == 0)    {      bt = NULL; // 空子树    }    else    {      bt = (BinTree)malloc(sizeof(struct TNode)); // 开辟空间      (bt)->data = \*preorder;      preorder++;(\*N)--;    //后移输入      (bt)->left = pre\_build\_tree(preorder,N); // 创建左子树      (bt)->right = pre\_build\_tree(preorder,N); // 创建右子树    }    return bt;  }    void MaxWidth(BinTree T, int width[])  {      if (T == NULL)      {          return ;      }      else      {        //移除了i 因为可以直接操作指针进行数组赋值          (\*width)++;     //该层结点数+1          MaxWidth(T->left,width+1);    //遍历左子树          MaxWidth(T->right,width+1);   //右子树      }  }  int get\_height(BinTree BT)  {    int HL, HR, MaxH;      if( BT ) {          HL = get\_height(BT->left);   /\* 求左子树的高度 \*/          HR = get\_height(BT->right);  /\* 求右子树的高度 \*/          MaxH = HL > HR ? HL : HR;   /\* 取左右子树较大的高度 \*/          return ( MaxH + 1 );        /\* 返回树的高度 \*/    }    else  return 0; /\* 空树高度为0 \*/  }  int get\_width(BinTree T)  {      int height=get\_height(T);      int \*width=(int\*)malloc(sizeof(int)\*height);   //分配width数组空间      if(!width)      {        print\_err("get\_width malloc failed....");        exit(-2);      }      MaxWidth(T,width);    //获取每层结点个数      int max\_width=-1;   //最大宽度      for(int i=0;i<height;i++)      {        if(max\_width<\*(width+i)) max\_width=\*(width);      }    free(width);    //释放内存    return max\_width;   //返回宽度  }  //bintree.h  #pragma once  #include "err.h"  typedef int ElementType;    //元素类型  typedef struct TNode \*position;  typedef position BinTree;      //二叉树  struct TNode{       //二叉树节点      ElementType data;       //节点数据      BinTree left;       //指向左子树      BinTree right;       //指向右子树  };  /\*  @brief 根据中序遍历和后序遍历创建二叉树  @param int[] 中序遍历  @param int[] 后序遍历  @param int  最大结点个数  23010341  @return Bintree 创建完成的BinTree  \*/  BinTree build\_tree(int inorder[],int postorder[],int N);  /\*  @brief 先序遍历二叉树  @param BinTree 二叉树表头  @return None  \*/  void pre\_order\_traversal(BinTree T);  /\*  @brief 清空二叉树   23010341  @param BinTree 二叉树表头  @return None  \*/  void free\_bt(BinTree T);  /\*  @brief 判断二叉树是否是二叉搜索树  @param BinTree 二叉树表头  @param ElementType  根节点数据min,max  @return bool  \*/  bool isBST(BinTree T,ElementType \*min,ElementType \*max);  /\*  @brief 前序创建二叉树  @param int[]   前序序列  @param int\*  节点个数  @return BinTree 返回二叉树表头  \*/  BinTree pre\_build\_tree(int preorder[],int \*N);    /\*  @brief 生成每层的结点数(宽度)  @param BinTree 二叉树  @param int[] 当前层书拥有的结点数  \*/  void MaxWidth(BinTree T,int width[]);       //优化了 i  没必要;可以直接操作width指针  /\*  @brief 获取二叉树宽度  @param Bintree 二叉树  @return int 二叉树宽度  \*/  int get\_width(BinTree T);  /\*  @brief 获取二叉树高度  @param BinTree 二叉树  @return int 二叉树高度  \*/  int get\_height(BinTree T);    //err.h  //用于错误输出 23010341杳泽  #pragma once  #include "headers.h"  /\*  @brief perror(str) 并写入stderr  @param char\* str要写入的错误信息  @return None  \*/  void print\_err(char \*str);  /\*  @brief 重定向stderr至 str.log  @param  char\* 重定向后的文件主名  @return None  \*/  void redirect\_stderr(char \*str) ;    //err.c  #include "err.h"    void print\_err(char \*str)        //perror(str)    并写入stderr  {      perror(str);      fprintf(stderr,"errno:%d\n\n",errno) ;      fprintf(stdout,"%s.errno:%d\n",str,errno);      return ;  }  void redirect\_stderr(char \*str)      //重定向stderr 至str.log  {      char location[20];      sprintf(location,"./%s.log",str);       //生成存储文件名      if(freopen(location, "a", stderr) == NULL)fprintf(stdout,"error redirecting stdout\n");      return ;  }    //headers.h  //类似于 #include <std/bitsc++.h>       集合了所需的所有头文件的头文件  #pragma once  #include <errno.h>  #include <stdio.h>  #include <stdbool.h>  #include <stdlib.h>  #include <string.h>  #include <math.h>  #include <time.h>  #include <ctype.h>    //main.c  #include "bintree.h"  const int MAXN =30;     //最大元素个数  //const比宏定义更安全,因为宏定义只是单纯的替换,编译器并不会检查  int main(int argc,char \*argv[])  {      /\*      bool echo=false;      if(argc>1)      //带输入参数      {          if(strcmp(argv[1],"-echo"))     //开启输入回显          {              echo=true;          }      }      \*/      BinTree T;      int inorder[MAXN],postorder[MAXN],N,i;      printf("输入N:");      scanf("%d",&N);      /\*      int preorder[MAXN];      printf("preorder:");       //输入先序遍历      for(int j=0;j<N;j++)      {          scanf("%c",&preorder[j]);      }      T=pre\_build\_tree(preorder,&N);      //先序创建      pre\_order\_traversal(T);     //先序遍历      \*/        printf("postorder:");       //输入后序遍历      for(int j=0;j<N;j++)      {          scanf("%d",&postorder[j]);      }      printf("inorder:");      for(int j=0;j<N;j++)    //中序遍历      {          scanf("%d",&inorder[j]);      }      T=build\_tree(inorder,postorder,N);      printf("preorder:\n");      pre\_order\_traversal(T);      int min=-1,max=-1;      printf("\n是否BST:%d\n",isBST(T,&min,&max));        free\_bt(T);     //经valgrind检查无内存泄漏,free\_bt 有效      return 0;  }  **2.编写算法计算二叉树最大的宽度（二叉树的最大宽度是指二叉树所有层中结点个数的最大值）。**  **1）设计思路**  //构造二叉树   1. 利用scanf读取后序和中序遍历结果、二叉树结点个数。 2. 在后续遍历中找到根节点。 3. 从中序遍历中找到根结点，切出左右子树，对根节点所在位置（数组下标）进行标记 4. 再重复2.3步骤在左右子树中找到根节点和其孩子。   //前序遍历   1. 先访问根节点数据 2. 利用递归相继访问左子树和右子树的数据   //获取树的宽度   1. 先获取树的深度，利用递归依次获得左右子树中的最大高度，进而得到整棵树的最大高度 2. 利用树的高度分配width[]空间，并初始化为全0，使用动态内存。 3. 利用递归统计每一层的结点数，并存入width[]数组，遍历完每一层后，width[]数组内最大的值即为树的宽度；利用指针直接操作，省去了递归的中间变量。   //其他设计   1. 人性化的输入输出，以空格分隔序列的输入 2. 添加了程序错误运行日志，并将其重定向到txt文件（利用freopen） 3. 完善了内存回收机制并利用valgrind进行内存检查，确保程序无内存泄漏，利用gdb进行调试 4. 利用const进行常量定义而不是用宏定义。因为编译器不会对宏定义进行类型检查，只做纯粹的代换，但是会对const修饰的变量进行类型检查。 5. 对程序进行模块化设计，将声明统一写在头文件内并加以详细注释（使头文件有较好的描述数据结构的效果），定义统一写在源文件内，变量命名使用下划线法增加可读性。   **2）数据结构的描述**  //同1. //bintree.h  typedef int ElementType;    //元素类型  typedef struct TNode \*position;  typedef position BinTree;      //二叉树  struct TNode{       //二叉树节点      ElementType data;       //节点数据      BinTree left;       //指向左子树      BinTree right;       //指向右子树  };  //主要用到的操作函数（操作集）  /\*  @brief 生成每层的结点数(宽度)  @param BinTree 二叉树  @param int[] 当前层书拥有的结点数  \*/  void MaxWidth(BinTree T,int width[]);       //优化了 i  没必要;可以直接操作width指针  /\*  @brief 获取二叉树宽度  @param Bintree 二叉树  @return int 二叉树宽度  \*/  int get\_width(BinTree T);  /\*  @brief 获取二叉树高度  @param BinTree 二叉树  @return int 二叉树高度  \*/  int get\_height(BinTree T);  //更多参考bintree.h或1.（第1题）  **3）程序代码**  //bintree.c  #include "bintree.h"  BinTree build\_tree(int inorder[], int postorder[], int N)  {    BinTree T;    int p;    if (!N)      return NULL;                             // 终止    T = (BinTree)malloc(sizeof(struct TNode)); // 分配空间    if (!T)    {      print\_err("build\_tree() malloc failed.");      exit(-1);    }    T->data = postorder[N - 1];    T->left = T->right = NULL;    for (p = 0; p < N; p++)      if (inorder[p] == postorder[N - 1])        break; // 在中序中寻找根节点    T->left = build\_tree(inorder, postorder, p);    T->right = build\_tree(inorder + p + 1, postorder + p, N - p - 1);    return T;  }  void pre\_order\_traversal(BinTree T)  {    if (T)    {      printf("%d ", T->data);        // 访问节点数据      pre\_order\_traversal(T->left);  // 访问左子树      pre\_order\_traversal(T->right); // 访问右子树    }    return;  }  void free\_bt(BinTree T) // 利用2个工作指针,利用递归先完成左右子树释放再释放根节点  {    BinTree left = T->left, right = T->right; // 获取左右子树    if (left)                                 // 左子树不为空    {      free\_bt(left); // 释放左子树    }    if (right) // 右子树不为空    {      free\_bt(right);    }    free(T); // 左右子树都空 释放当前节点    return;  // 到头  }  bool isBST(BinTree T, int \*min, int \*max)  {    int lmin, lmax, rmin, rmax, Left\_flag, Right\_flag;    if (!T)      return true;    if (!T->left && !T->right)    {      (\*min) = (\*max) = T->data;      return true;    }    Left\_flag = Right\_flag = false;    if ((T->left &&         isBST(T->left, &lmin, &lmax) &&         T->data > lmax) ||        !T->left)      Left\_flag = true;    if ((T->right &&         isBST(T->right, &rmin, &rmax) &&         T->data < rmin) ||        !T->right)      Right\_flag = true;    if (Left\_flag && Right\_flag)    {      if (T->left)        (\*min) = lmin;      else        (\*min) = T->data;      if (T->right)        (\*max) = rmax;      else        (\*max) = T->data;      return true;    }    else      return false;  }  BinTree pre\_build\_tree(int preorder[], int \*N)  {    //ABDE000F00CG000    BinTree bt;    if(!(\*N)) return NULL;   //创建完毕    if (\*preorder == 0)    {      bt = NULL; // 空子树    }    else    {      bt = (BinTree)malloc(sizeof(struct TNode)); // 开辟空间      (bt)->data = \*preorder;      preorder++;(\*N)--;    //后移输入      (bt)->left = pre\_build\_tree(preorder,N); // 创建左子树      (bt)->right = pre\_build\_tree(preorder,N); // 创建右子树    }    return bt;  }    void MaxWidth(BinTree T, int \*width)  {      if (T == NULL)      {          return ;      }      else      {        //移除了i 因为可以直接操作指针进行数组赋值          (\*width)++;     //该层结点数+1          MaxWidth(T->left,width+1);    //遍历左子树          MaxWidth(T->right,width+1);   //右子树      }  }  int get\_height(BinTree BT)  {    int HL, HR, MaxH;      if( BT ) {          HL = get\_height(BT->left);   /\* 求左子树的高度 \*/          HR = get\_height(BT->right);  /\* 求右子树的高度 \*/          MaxH = HL > HR ? HL : HR;   /\* 取左右子树较大的高度 \*/          return ( MaxH + 1 );        /\* 返回树的高度 \*/    }    else  return 0; /\* 空树高度为0 \*/  }  int get\_width(BinTree T)  {      int height=get\_height(T);      int \*width=(int\*)malloc(sizeof(int)\*height);   //分配width数组空间      if(!width)      {        print\_err("get\_width malloc failed....");        exit(-2);      }      for(int \*p=width;p<width+height;p++)    //初始化width      {        \*p=0;      }      MaxWidth(T,width);    //获取每层结点个数      int max\_width=-1;   //最大宽度      for(int i=0;i<height;i++)      {        if(max\_width<\*(width+i)) max\_width=\*(width+i);      }    free(width);    //释放内存    return max\_width;   //返回宽度  }    //bintree.h  #pragma once  #include "err.h"  typedef int ElementType;    //元素类型  typedef struct TNode \*position;  typedef position BinTree;      //二叉树  struct TNode{       //二叉树节点      ElementType data;       //节点数据      BinTree left;       //指向左子树      BinTree right;       //指向右子树  };  /\*  @brief 根据中序遍历和后序遍历创建二叉树  @param int[] 中序遍历  @param int[] 后序遍历  @param int  最大结点个数  23010341  @return Bintree 创建完成的BinTree  \*/  BinTree build\_tree(int inorder[],int postorder[],int N);  /\*  @brief 先序遍历二叉树  @param BinTree 二叉树表头  @return None  \*/  void pre\_order\_traversal(BinTree T);  /\*  @brief 清空二叉树   23010341  @param BinTree 二叉树表头  @return None  \*/  void free\_bt(BinTree T);  /\*  @brief 判断二叉树是否是二叉搜索树  @param BinTree 二叉树表头  @param ElementType  根节点数据min,max  @return bool  \*/  bool isBST(BinTree T,ElementType \*min,ElementType \*max);  /\*  @brief 前序创建二叉树  @param int[]   前序序列  @param int\*  节点个数  @return BinTree 返回二叉树表头  \*/  BinTree pre\_build\_tree(int preorder[],int \*N);    /\*  @brief 生成每层的结点数(宽度)  @param BinTree 二叉树  @param int[] 当前层书拥有的结点数  \*/  void MaxWidth(BinTree T,int width[]);       //优化了 i  没必要;可以直接操作width指针  /\*  @brief 获取二叉树宽度  @param Bintree 二叉树  @return int 二叉树宽度  \*/  int get\_width(BinTree T);  /\*  @brief 获取二叉树高度  @param BinTree 二叉树  @return int 二叉树高度  \*/  int get\_height(BinTree T);    //err.h  //用于错误输出 23010341杳泽  #pragma once  #include "headers.h"  /\*  @brief perror(str) 并写入stderr  @param char\* str要写入的错误信息  @return None  \*/  void print\_err(char \*str);  /\*  @brief 重定向stderr至 str.log  @param  char\* 重定向后的文件主名  @return None  \*/  void redirect\_stderr(char \*str) ;    //err.c  #include "err.h"    void print\_err(char \*str)        //perror(str)    并写入stderr  {      perror(str);      fprintf(stderr,"errno:%d\n\n",errno) ;      fprintf(stdout,"%s.errno:%d\n",str,errno);      return ;  }  void redirect\_stderr(char \*str)      //重定向stderr 至str.log  {      char location[20];      sprintf(location,"./%s.log",str);       //生成存储文件名      if(freopen(location, "a", stderr) == NULL)fprintf(stdout,"error redirecting stdout\n");      return ;  }    //headers.h  //类似于 #include <std/bitsc++.h>       集合了所需的所有头文件的头文件  #pragma once  #include <errno.h>  #include <stdio.h>  #include <stdbool.h>  #include <stdlib.h>  #include <string.h>  #include <math.h>  #include <time.h>  #include <ctype.h>    //main.c  #include "bintree.h"  const int MAXN =30;     //最大元素个数  //const比宏定义更安全,因为宏定义只是单纯的替换,编译器并不会检查    int main(int argc,char \*argv[])  {      /\*      bool echo=false;      if(argc>1)      //带输入参数      {          if(strcmp(argv[1],"-echo"))     //开启输入回显          {              echo=true;          }      }      \*/      BinTree T;      int inorder[MAXN],postorder[MAXN],N,i;      printf("输入N:");      scanf("%d",&N);      /\*      int preorder[MAXN];      printf("preorder:");       //输入先序遍历      for(int j=0;j<N;j++)      {          scanf("%c",&preorder[j]);      }      T=pre\_build\_tree(preorder,&N);      //先序创建      pre\_order\_traversal(T);     //先序遍历      \*/        printf("postorder:");       //输入后序遍历      for(int j=0;j<N;j++)      {          scanf("%d",&postorder[j]);      }      printf("inorder:");      for(int j=0;j<N;j++)    //中序遍历      {          scanf("%d",&inorder[j]);      }      T=build\_tree(inorder,postorder,N);      printf("preorder:\n");      pre\_order\_traversal(T);     //先序序列      printf("\n");      /\*      int min=-1,max=-1;      printf("\n是否BST:%d\n",isBST(T,&min,&max));      \*/      printf("树的宽度为%d",get\_width(T));      free\_bt(T);     //经valgrind检查无内存泄漏,free\_bt 有效      return 0;  }  **3.设顺序存储的二叉树中有编号为i和j的两个结点，请设计算法求出它们最近的公共祖先结点的编号。**  **1）设计思路**  //这题因为是顺序存储所以相对简单  本质上该题不涉及对节点数据的访问和检查，空节点标记为（#）即可，因为利用的是完全二叉树中孩子节点和父亲节点的编号关系。  //int joint\_ancestor(int len,int i,int j);   1. 先对输入的i，j进行合法性检查，并与len进行比较，错误则返回-1 2. 比较i j的大小，将大的那一个除以2并向下取整（大的说明再更深的一层，而共同的祖先节点肯定是同一层的一个节点） 3. 比较i和j是否相等，相等则输出该节点编号（双方都找到了同一个祖先节点，表明就是共同的）；不相等则重复2.3. 4. 理论上只要i和j的值合法，除2除到最后i和j都会是0（即根节点在没有其他共同祖先节点的情况下，一定是共同的祖先节点）   **2）数据结构的描述**  //顺序存储的二叉树      char BT[] = {'a', 'b', 'c', 'd', 'e', '#', 'f'};        //下标+1就是序号  完全二叉树  /\*  @brief 寻找顺序存储二叉树编号为i和j的共同的祖先节点编号  @param int 二叉树存储的数据个数  @return int 编号  \*/  int joint\_ancestor(int len,int i,int j);  //实际上顺序存储二叉树里找祖先节点和二叉树本身没啥关系(如果认为空节点也算结点的话)  **3）程序代码**  //main.c  #include <stdio.h>  #include <stdbool.h>  /\*  @brief 寻找顺序存储二叉树编号为i和j的共同的祖先节点编号  @param int 二叉树存储的数据个数  @return int 编号  \*/  int joint\_ancestor(int len,int i,int j);  //实际上顺序存储二叉树里找祖先节点和二叉树本身没啥关系(如果认为空节点也算结点的话)  int main()  {      //顺序存储的二叉树      char BT[] = {'a', 'b', 'c', 'd', 'e', '#', 'f'};        //下标+1就是序号  完全二叉树      printf("请输入i和j.\n");      int i=0,j=0;      scanf("%d %d",&i,&j);      printf("编号为%d和%d的节点的共同祖先节点编号为%d.\n",i,j,joint\_ancestor(7,i,j));      return 0;  }  /\*         1        / \       2   3      / \ / \     4  5 6  7    / \ / \ / \   8  9 10 11 12 13  / \ / \ / \ / \  14 15 16 17 18 19 20        10       /  \      5    15     / \   / \    3   7 12 18  \*/  int joint\_ancestor(int len,int i,int j)  {      if (i\*j<=0||i>len||j>len) return -1;      //i和j invalid      //如果i和j合理 且二叉树正常,i==j作为终止条件 最后一定会有结果(Root)      while(1)      {      if(i>j)     i/=2;      else j/=2;      if(i==j)break;      }      return i;  } | |

|  |
| --- |
| **三、程序运行结果（分析计算时空复杂度，运行界面截图）**  **1.**    yaoze@yaoze-VMware-Virtual-Platform:/mnt/code/c/data\_structure-exp/exp-2/exp-2-1$ ls  bintree.c err.c exp-2-1.exe headers.h test1.txt  bintree.h err.h exp-2-1.out main.c  yaoze@yaoze-VMware-Virtual-Platform:/mnt/code/c/data\_structure-exp/exp-2/exp-2-1$ gcc \*.c -o exp-2-1.out -g  **//算法中大量涉及树的遍历，但每一节点只遍历1次，所以时间复杂度应为O(n)；**  **堆栈深度和树的高度有关，如果树完全平衡则S(n）=O(logn），最坏情况下（线性表）为**  **O(n)**  2.    yaoze@yaoze-VMware-Virtual-Platform:/mnt/code/c/data\_structure-exp/exp-2/exp-2-2$ ls  bintree.c err.c exp-2-2.exe headers.h test1.txt  bintree.h err.h exp-2-2.out main.c  yaoze@yaoze-VMware-Virtual-Platform:/mnt/code/c/data\_structure-exp/exp-2/exp-2-2$ gcc \*.c -o exp-2-2.out -g  //（理论上和第1题的时空复杂度一致）**//算法中大量涉及树的遍历，但每一节点只遍历1次，所以时间复杂度应为O(n)；**  **堆栈深度和树的高度有关，如果树完全平衡则S(n）=O(logn），最坏情况下（线性表）为**  **O(n)**  3.        **//显然S(n)=O(1) 因为是确定的顺序结构存储的二叉树（也可以认为数据越多，静态存储的树据越多则S(n)=O(n)）;**  **T(n)=O(logn) //每次进行除2取整操作** |

**成绩：\_\_\_\_\_\_\_\_\_\_\_\_** **任课教师签名：\_\_\_\_\_\_\_\_\_\_\_\_** 2024年 月 日