**实验报告**

**学院（系）名称：**计算机科学与工程学院

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 王帆 | | **学号** | | 20152180 | | | **专业** | | 计算机科学与技术 | |
| **班级** | | 2015级1班 | | **实验项目** | | 实验三：二叉树的操作 | | | | | | |
| **课程名称** | | | | 数据结构与算法 | | | | | **课程代码** | | 0668016 | |
| **实验时间** | | | | 2017年 4月28日 第 1节 | | | | | **实验地点** | | 7-216 | |
| 考核标准 | 实验过程  25分 | | 程序运行  20分 | | 回答问题  15分 | | 实验报告  30分 | 特色  功能  5分 | | 考勤违纪情况  5分 | **成绩** |  |
| 成绩栏 |  | |  | |  | |  |  | |  | 其它批改意见:  教师签字： | |
| 考核内容 | 评价在实验课堂中的表现，包括实验态度、编写程序过程等内容等。 | | □功能完善,  □功能不全  □有小错  □无法运行 | | ○正确  ○基本正确  ○有提示  ○无法回答 | | ○完整  ○较完整  ○一般  ○内容极少  ○无报告 | ○有  ○无 | | ○有  ○无 |
|  | | | | | | | | | | | | |
| **一、实验目的**  通过实验深刻理解二叉树性质，验证二叉树的遍历算法，并能在遍历算法基础上实现较复杂算法设计。  **二、实验题目与要求**  每位同学从下面题目中至少选择2题实现：其中第1题必做，2-5题中至少选择1题。  1. 以二叉链表为存储结构，实现二叉树的创建、遍历  1）问题描述：在主程序中设计一个简单的菜单，分别调用相应的函数功能：  1…建立树  2…前序遍历树  3…中序(非递归)遍历树  4…后序遍历树  0…结束  2）实验要求：在程序中定义下述函数，并实现要求的函数功能：  CreateTree()：按从键盘输入的前序序列，创建树  PreOrderTree()：前序遍历树（递归）  InOrderTree()：中序(非递归)遍历树  LaOrderTree(): 后序遍历树（递归）  3）注意问题:   * 注意理解递归算法的执行步骤。 * 注意字符类型数据在输入时的处理。 * 重点理解如何利用栈结构实现非递归算法   2.编写算法交换二叉树中所有结点的左、右子树   1. 问题描述：编写一算法，交换二叉树中所有结点的左、右子树 2. 实验要求：以二叉链表作为存储结构   3.试编写按层次顺序遍历二叉树的算法   1. 问题描述：编写按层次顺序遍历二叉树的算法 2. 实验要求：以二叉链表作为存储结构   4.编写算法求二叉树高度及宽度。  1） 问题描述：二叉树高度是指树中所有节点的最大层数，二叉树宽度是指在二叉树的各层上，具有节点数最多的那一层上的节点总数。  2） 实验要求：以二叉链表作为存储结构  5.实现一个哈夫曼编/译码系统  1）问题描述：利用哈夫曼编码进行信息通信可以大大提高信道利用率，缩短信息传输时间，降低传输成本。但是，这要求在发送端通过一个编码系统对待传输数据预先编码，在接收端将传来的数据进行译码。对于双工信道，每端都需要一个完整的编码/译码系统。试为这样的信息收发站写一个哈夫曼的编/译码系统。  2）实验要求：一个完整的系统应具有以下功能：  (1) I：初始化(Initialization)。从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存于文件hfmTree中。  (2) E：编码(Encoding)。利用已建好的哈夫曼树对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。  (3) D：译码(Decoding)。利用已建好的哈夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。  (4) P：打印代码文件(Print)。将文件CodeFile以紧凑格式显示在终端上，每行50个代码。同时将此字符形式的编码文件写入文件CodePrin中。  (5) T：打印哈夫曼树(Tree printing)。将已在内存中的哈夫曼树以直观的方式显示在终端上，同时将此字符形式的哈夫曼树写入文件TreePrint中。  3) 实现提示：  (1) 文件CodeFile的基类型可以设为字节型。  (2) 用户界面可以设计为“菜单”方式：显示上述功能符号，再加上“Q”，表示退出运行Quit。请用户键入一个选择功能符。此功能执行完毕后再显示此菜单，直至某次用户选择了“E”为止。  (3) 在程序的一次执行过程中，第一次执行I、D或C命令之后，哈夫曼树已经在内存了，不必再读入。每次执行中不一定执行I命令，因为文件hfmTree可能早已建好。   1. **实验过程与实验结果**   **实验3-01：以二叉链表为存储结构，实现二叉树的创建、遍历**  **数据结构定义：**  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **算法设计思路简介：**  本实验需要实现以下操作：  二叉树的初始化、前中后序遍历等基本操作  1.利用递归实现前后序遍历，思路简洁，仅需要调整递归体的执行顺序即可实现。  2.利用非递归实现中序遍历，需要利用栈操作，按照中序遍历规则将节点依次入栈后出栈实现。  **算法描述：**  **图1 中序遍历（非递归）实现**  **算法的实现和测试结果（参考OJ）**    **实验3-02：编写算法交换二叉树中所有结点的左、右子树**  **数据结构定义：**  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **算法设计思路简介：**  本实验需要实现以下操作：  二叉树的初始化、前中后序遍历等基本操作  1.利用递归实现前后序遍历，思路简洁，仅需要调整递归体的执行顺序即可实现。  2.利用非递归实现中序遍历，需要利用栈操作，按照中序遍历规则将节点依次入栈后出栈实现。  **算法描述：**  **图2 交换二叉树子树的实现**  **算法的实现和测试结果（参考OJ）**    **实验3-03：二叉树层次遍历**  **数据结构定义：**  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **算法设计思路简介：**  本实验需要实现以下操作：  1.二叉树的前序创建  思路同前题，此处略。  2.二叉树的层次遍历  利用递归调用实现对二叉树各层各元素的调用，并做记录。  C:\Users\Du\AppData\Local\Microsoft\Windows\INetCacheContent.Word\3-3.png**算法描述：**  **图3 层次遍历实现**  **算法的实现和测试结果（参考OJ）**  **实验3-04：二叉树高度宽度求解**  **数据结构定义：**  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **算法设计思路简介：**  本实验需要实现以下操作：  1.二叉树的前序创建  思路同前题，此处略  2.二叉树的高度（深度）求解  利用递归调用，实现对二叉树左右字数深度的判断，直到  C:\Users\Du\AppData\Local\Microsoft\Windows\INetCacheContent.Word\3-4.png**算法描述：**  C:\Users\Du\AppData\Local\Microsoft\Windows\INetCacheContent.Word\未命名文件.png**图4-1 二叉树高度（深度）求解实现**  **图4-2 二叉树宽度求解实现**  **算法的实现和测试结果（参考OJ）**  **实验3-05：哈夫曼编码**  **数据结构定义：**  typedef struct HTNode{  int weight;  int parent;  int lchild;  int rchild;  char data;  }HTNode;  typedef struct Leaf{  char data;  int weight;  }Leaf;  typedef struct HCode{  int bit[maxbit];  int start;  }HCode;  **算法设计思路简介：**  本实验需要实现以下操作：  1.哈夫曼树的构造  思路同前题，此处略  2.哈夫曼树的编码  利用递归调用，实现对二叉树左右字数深度的判断，直到  3.哈夫曼树的译码  C:\Users\Du\AppData\Local\Microsoft\Windows\INetCacheContent.Word\3-4.png**算法描述：**  C:\Users\Du\AppData\Local\Microsoft\Windows\INetCacheContent.Word\未命名文件.png**图4-1 二叉树高度（深度）求解实现**  **图4-2 二叉树宽度求解实现**  **算法的实现和测试结果（参考OJ）**   1. **收获与体会**   通过本次实验   1. **源代码清单**   **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///实验3-01：二叉树遍历**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  #define MaxNode 100  using namespace std;  #include <queue>  #include <stack>  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **//1.创建二叉树（先序）**  BiTree CreateTree(){  char ch;  BiTree T;  cin>>ch;  if(ch=='#') T=NULL;  else if(!(T=(BiTree)malloc(sizeof(BiTNode)))) exit(1);  else{  T->data=ch;  T->lchild=CreateTree();  T->rchild=CreateTree();  }  return T;  }  **//2.先序遍历（递归实现）**  void PreOrderTree(BiTree root){  if(root==NULL) return;  cout<<root->data;  PreOrderTree(root->lchild);  PreOrderTree(root->rchild);  }  **//3.中序遍历（非递归实现）**  void InOrderTree(BiTree root){  BiTree p,stack[MaxNode];  int top=0;  if(root==NULL) return;  p=root;  while(!(p==NULL&&top==0)){  while(p!=NULL){  if(top<MaxNode-1){  stack[top]=p;  top++;  }  else return;  p=p->lchild;  }  if(top<=0) return;  else{  top--;  p=stack[top];  cout<<p->data;  p=p->rchild;  }  }  }  **//4.后序遍历（递归实现）**  void LaOrderTree(BiTree root){  if(root==NULL) return;  PostOrder(root->lchild);  PostOrder(root->rchild);  cout<<root->data;  }  int main(){  int n,T;  BiTree root;  cin>>n;  for(i=0;i<n;i++){  cin>>T;  switch(T){  case 1:{  root=CreateTree();  break;  }  case 2:{  PreOrderTree(root);  break;  }  case 3:{  InOrderTree(root);  break;  }  case 4:{  LaOrderTree(root);  break;  }  case 0:{  return 0;  }  default:break;  }  }  return 0;  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///实验3-02：交换左右子树**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  #define MaxNode 100  using namespace std;  #include <queue>  #include <stack>  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **//1.创建二叉树（先序）**  BiTree Create\_Tree(){  char ch;  BiTree T;  cin>>ch;  if(ch=='#') T=NULL;  else if(!(T=(BiTree)malloc(sizeof(BiTNode)))) exit(1);  else{  T->data=ch;  T->lchild=Create\_Tree();  T->rchild=Create\_Tree();  }  return T;  }  **//2.中序遍历（递归）**  void InOrder(BiTree root){  if(root==NULL) return;  InOrder(root->lchild);  cout<<root->data;  InOrder(root->rchild);  }  **//3.交换子树（递归）**  void Change\_LR(BiTree root){  BiTree temp=NULL;  if(root->lchild==NULL&&root->rchild==NULL) return;  else{  temp=root->lchild;  root->lchild=root->rchild;  root->rchild=temp;  }  if(root->lchild) Change\_LR(root->lchild);  if(root->rchild) Change\_LR(root->rchild);  }  int main(){  BiTree root,root2,root3;  root=Create\_Tree();  Change\_LR(root);  InOrder(root);  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **//实验3-03：二叉树层次遍历**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  #define MaxNode 100  using namespace std;  #include <queue>  #include <stack>  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **//1.创建二叉树（先序）**  BiTree Create\_Tree(){  char ch;  BiTree T;  cin>>ch;  if(ch=='#') T=NULL;  else if(!(T=(BiTree)malloc(sizeof(BiTNode)))) exit(1);  else{  T->data=ch;  T->lchild=Create\_Tree();  T->rchild=Create\_Tree();  }  return T;  }  **//2.层次遍历**  int Level\_Traversal(BiTree T, int level){  if(!T||level<0) return 0;  if(level==0){  cout<<T->data;  return 1;  }  return Level\_Traversal(T->lchild,level-1)+Level\_Traversal(T->rchild,level-1);  }  void Level\_Traversal(BiTree T){  int i=0;  for(i=0;;i++)  if(!Level\_Traversal(T,i)) break;  return;  }  int main(){  BiTree root,root2,root3;  root=Create\_Tree();  Level\_Traversal(root);  return 0;  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **//实验3-04：二叉树高度宽度求解**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  #include <queue>  #define MaxNode 100  using namespace std;  typedef struct BiTNode{  char data;  BiTNode \*lchild, \*rchild;  }BiTNode;  typedef BiTNode \*BiTree;  **//1.创建二叉树（先序）**  BiTree Create\_Tree(){  char ch;  BiTree T;  cin>>ch;  if(ch=='#') T=NULL;  else if(!(T=(BiTree)malloc(sizeof(BiTNode)))) exit(1);  else{  T->data=ch;  T->lchild=Create\_Tree();  T->rchild=Create\_Tree();  }  return T;  }  **//2.求二叉树高度（深度）**  int Get\_Height(BiTree T){  if(T==NULL) return 0;  return Get\_Height(T->lchild)>Get\_Height(T->rchild)?(Get\_Height(T->lchild)+1):(Get\_Height(T->rchild)+1);  }  **//3.求二叉树宽度**  int Get\_Width(BiTree pNode){  if (pNode == NULL)  {  return 0;  }  std::deque<BiTree> dequeTreeNode;**//双端队列**  int maxWidth = 1;**//最大的宽度,用于当只有一个节点时候返回1**  dequeTreeNode.push\_back(pNode);//头结点入队  while (true)  {    int length = dequeTreeNode.size();**//当前层节点的个数**  if (length == 0)**//当前层没有节点，跳出循环**  {  break;  }  while (length > 0)**//如果当前层还有节点**  {  BiTree pTemp = dequeTreeNode.front();  dequeTreeNode.pop\_front();**//出队**  length--;**//长度减一**  if (pTemp->lchild)  {  dequeTreeNode.push\_back(pTemp->lchild);**//下一层节点入队**  }  if (pTemp->rchild)  {  dequeTreeNode.push\_back(pTemp->rchild);**//下一层节点入队**  }  }  maxWidth = maxWidth > dequeTreeNode.size() ? maxWidth : dequeTreeNode.size();  **//得到最大宽度**  }  return maxWidth;  }  int main(){  BiTree root;  root=Create\_Tree();  cout<<Get\_Height(root)<<" "<<Get\_Width(root);  return 0;  }  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  **//实验3-05：哈夫曼编码**  **/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**  #include <iostream>  #include <cstdlib>  #include <string>  using namespace std;  #define MaxWeight 10000  #define MaxLeaf 10000  #define maxnode MaxLeaf \* 2-1  #define maxbit 100  typedef struct HTNode{  int weight;  int parent;  int lchild;  int rchild;  char data;  }HTNode;  typedef struct Leaf{  char data;  int weight;  }Leaf;  typedef struct HCode{  int bit[maxbit];  int start;  }HCode;  HTNode H[maxnode];  Leaf L[MaxLeaf];  HCode C[maxnode];  int n;  int whole = 0;  void init\_Haffman() {  int i,j,m1,m2,x1,x2;  for(i=0; i<2\*n-1; i++) {  H[i].weight=0;  H[i].parent=-1;  H[i].lchild=-1;  H[i].rchild=-1;  }  for(i=0; i<n; i++) {  H[i].data=L[i].data;  H[i].weight=L[i].weight;  }  for(i=0; i<n-1; i++) {  m1=m2=MaxWeight;  x1=x2=0;  for(j=0; j<n+i; j++) {  if(H[j].weight<m1&&H[j].parent==-1) {  m2=m1;  x2=x1;  m1=H[j].weight;  x1=j;  } else if(H[j].weight<m2&&H[j].parent==-1) {  m2=H[j].weight;  x2=j;  }  }  H[x1].parent=n+i;  H[x2].parent=n+i;  H[n+i].weight=H[x1].weight+H[x2].weight;  H[n+i].lchild=x1;  H[n+i].rchild=x2;  }  }  void init\_Code(){  HCode cd;  int i,j,c,p;  for(i=0;i<n;i++){  cd.start=n-1; c=i;  p=H[c].parent;  while(p!=-1){  if(H[p].lchild==c) cd.bit[cd.start]=0;  else cd.bit[cd.start]=1;  cd.start--; c=p;  p=H[c].parent;  }  for(j=cd.start+1;j<n;j++){  C[i].bit[j]=cd.bit[j];  }  C[i].start=cd.start;  }  }  void Code\_Now(){  string str;  cin>>str;  for(int i=0;str[i]!=NULL;i++){  for(int j=0;j<n;j++){  if(str[i]==H[j].data){  for(int k=C[j].start+1;k<n;k++){  cout<<C[j].bit[k];  }  break;  }  }  }  cout<<endl;  }  void Decode\_Now(){  string str;  int i,j,k,l;  cin>>str;  int len=str.length();  whole = len;  while(whole!=0){  for(i=0;i<n;i++){  int m=0;  for(k=C[i].start+1,j=0;k<n&&j<whole;k++,j++){  if((str[j]-'0')==C[i].bit[k])  m++;  }  if(m==j){  cout<<H[i].data;  for(int x=0;(x+j)<whole;x++){  str[x]=str[x+j];  }  whole -= j;  break;  }  }  }  }  int main(){  int Q,x;  cin>>Q;  for(int i=0;i<Q;i++){  cin>>x;  switch(x){  case 0:  cin>>n;  for(int i=0; i<n; i++){  cin>>L[i].data;  }  for(int j=0; j<n; j++){  cin>>L[j].weight;  }  init\_Haffman();  init\_Code();  break;  case 1:  Code\_Now();  break;  case 2:  Decode\_Now();  break;  }  }  return 0;  } | | | | | | | | | | | | |