

**实 验 报 告**

**（2019 / 2020 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 数据结构 | | | | | |
| 实验名称 | 二叉树的基本操作  及哈夫曼编码译码系统的实现 | | | | | |
| 实验时间 | 2019 | 年 | 11 | 月 | 22 | 日 |
| 指导单位 | 计算机学院 | | | | | |
| 指导教师 | 邹志强 | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 | 张颖 | 班级学号 | B18030406 |
| 学院(系) | 计算机学院 | 专 业 | 计算机科学与技术 |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | **二叉树的基本操作**  **及哈夫曼编码译码系统的实现** | | | **指导教师** | **邹志强** |
| **实验类型** | **验证** | **实验学时** | **2** | **实验时间** | **2019-11-22** |
| 1. **实验目的和要求**   1.掌握二叉树的二叉链表存储表示及遍历操作实现方法。  2.实现二叉树遍历运算的应用：求二叉树中叶子结点个数、结点总数、二叉树的高度、交换二叉树的左右子树等。  3.掌握二叉树的应用——哈夫曼编码的实现。 | | | | | |
| 二、**实验环境(实验设备)**  硬件：微型计算机  软件：Windows 操作系统、Dev-C++ 5.11 | | | | | |
| **三、实验原理及内容**  1、算法设计  ADT BinaryTree{  数据：  二叉树是结点的有限集合，它或者为空集，或者由一个根结点、根结点的左子树和右子树组成。  运算：  InitBinaryTree(T)：构造空二叉树T。  DestroyBinaryTree(T)：销毁二叉树T。  CreateBinaryTree(T)：建立以T为根的二叉树。  ClearBinaryTree(T)：若二叉树T存在，则将二叉树T清空。  EmptyofBinaryTree(T)：若二叉树T为空树，则返回TRUE，否则返回FALSE。  DepthofBinaryTree(T)：返回二叉树T的高度。  Root(T)：返回二叉树T的根结点。  Value(T, e)：e是二叉树T中的一个结点，返回此结点的值。  Assign(T, e, value)：e是二叉树T中的一个结点，将e的值赋值为value。  Parent(T, e)：返回二叉树T中e结点的双亲；若e是根结点，则返回空。  LeftChild(T, e)：返回二叉树T中e结点的左孩子；若e是叶子或无左孩子，返回空。  RightChild(T, e)：返回二叉树T中e结点的右孩子；若e是叶子或无右孩子，返回空。  InsertChild(T, t, lr, i)：将结点i插入到t所指结点的左或右子树。lr值为0时插入到t的左子树，i成为左子树的根结点；lr值为1时插入到t的右子树，i成为右子树的根结点。  DeleteSubTree(T, t, lr)：删除二叉树T中t所指结点的左或右子树。其中，t指向二叉树T中的某个结点，lr值为0时删除t的左子树，lr值为1时删除t的右子树。  PreOrderTransverse(T)：按照先序次序对二叉树T的所有结点进行访问一次操作。  InOrderTransverse(T)：按照中序次序对二叉树T的所有结点进行访问一次操作。  PostOrderTransverse(T)：按照后序次序对二叉树T的所有结点进行访问一次操作。  }  2、算法实现与分析  1）参照程序5.1~程序5.4，编写程序，完成二叉树的先序创建、先序遍历、中序遍历、后序遍历等操作。  #include <stdio.h>  #include <stdlib.h>  typedef char ElemType;  typedef struct BinTree  {  ElemType data;  struct BinTree \*lchild, \*rchild;  } BinTree, \*BiTree;  void PreCreateBt(BiTree \*T)  {  ElemType c;  scanf("%c", &c);  if ('#' == c)  {  \*T = NULL;  }  else  {  \*T = (BinTree \*)malloc(sizeof(BinTree));  (\*T)->data = c;  PreCreateBt(&(\*T)->lchild);  PreCreateBt(&(\*T)->rchild);  }  }  void PreOrderTransverse(BiTree T)  {  if (T)  {  printf("%c", T->data);  PreOrderTransverse(T->lchild);  PreOrderTransverse(T->rchild);  }  }  void InOrderTransverse(BiTree T)  {  if (T)  {  PreOrderTransverse(T->lchild);  printf("%c", T->data);  PreOrderTransverse(T->rchild);  }  }  void PostOrderTransverse(BiTree T)  {  if (T)  {  PreOrderTransverse(T->lchild);  PreOrderTransverse(T->rchild);  printf("%c", T->data);  }  }  int main()  {  BiTree T = NULL;  printf("先序创建：");  PreCreateBt(&T);  printf("先序遍历：");  PreOrderTransverse(T);  printf("\n中序遍历：");  InOrderTransverse(T);  printf("\n后序遍历：");  PostOrderTransverse(T);  return 0;  }  2）以第一题所示二叉链表为存储结构，编写程序实现求二叉树结点个数、叶子结点个数、二叉树的高度以及交换二叉树所有左右子树的操作。  #include <stdio.h>  #include <stdlib.h>  typedef char ElemType;  typedef struct BinTree  {  ElemType data;  struct BinTree \*lchild, \*rchild;  } BinTree, \*BiTree;  void PreCreateBt(BiTree \*T)  {  ElemType c;  scanf("%c", &c);  if ('#' == c)  {  \*T = NULL;  }  else  {  \*T = (BinTree \*)malloc(sizeof(BinTree));  (\*T)->data = c;  PreCreateBt(&(\*T)->lchild);  PreCreateBt(&(\*T)->rchild);  }  }  void PreOrderTransverse(BiTree T)  {  if (T)  {  printf("%c", T->data);  PreOrderTransverse(T->lchild);  PreOrderTransverse(T->rchild);  }  }  void InOrderTransverse(BiTree T)  {  if (T)  {  PreOrderTransverse(T->lchild);  printf("%c", T->data);  PreOrderTransverse(T->rchild);  }  }  void PostOrderTransverse(BiTree T)  {  if (T)  {  PreOrderTransverse(T->lchild);  PreOrderTransverse(T->rchild);  printf("%c", T->data);  }  }  int CountLeaf(BiTree T)  {  if (T == NULL)  return 0;  else  return CountLeaf(T->lchild) + CountLeaf(T->rchild) + 1;  }  int Leaf(BiTree T)  {  if (T == NULL)  return 0;  if (!(T->lchild) && !(T->rchild))  return 1;  else  return (Leaf(T->lchild) + Leaf(T->rchild));  }  int Depth(BiTree T)  {  if (T == NULL)  return 0;  int left, right, depth;  left = Depth(T->lchild);  right = Depth(T->rchild);  depth = (left > right ? left : right) + 1;  return depth;  }  void Exchange(BiTree T)  {  BiTree p;  if (T != NULL)  {  p = T->lchild;  T->lchild = T->rchild;  T->rchild = p;  Exchange(T->lchild);  Exchange(T->rchild);  }  }  int main()  {  BiTree T = NULL;  printf("先序创建：");  PreCreateBt(&T);  printf("先序遍历：");  PreOrderTransverse(T);  printf("\n中序遍历：");  InOrderTransverse(T);  printf("\n后序遍历：");  PostOrderTransverse(T);  printf("\n结点个数：%d", CountLeaf(T));  printf("\n叶子结点个数：%d", Leaf(T));  printf("\n二叉树的高度：%d", Depth(T));  printf("\nProcessing:交换左右子树");  Exchange(T);  printf("\nProcessed: 交换左右子树");  printf("\n先序遍历：");  PreOrderTransverse(T);  printf("\n中序遍历：");  InOrderTransverse(T);  printf("\n后序遍历：");  PostOrderTransverse(T);  return 0;  }  3）编写程序，实现哈夫曼树的创建、哈夫曼编码以及解码的实现。  #include <iostream>  #include <vector>  #include <string>  using namespace std;  const int INF = 1000000000;  const int maxBit = 1 << 5;  const int maxNode = 1 << 10;  const int maxCode = 1 << 10;  struct Node  {  string value;  float weight;  int parent;  int lchild;  int rchild;  };  struct Code  {  int bit[maxBit];  int start;  string value;  };  Node huffman[maxNode];  Code huffmanCode[maxCode];  int n;  void initHuffmanTree()  {  for (int i = 0; i < (2 \* n) - 1; i++)  {  huffman[i].weight = 0;  huffman[i].value = "";  huffman[i].parent = -1;  huffman[i].lchild = -1;  huffman[i].rchild = -1;  }  }  void huffmanTree()  {  for (int i = 0; i < n - 1; i++)  {  int m1 = INF;  int m2 = INF;  int x1 = 0;  int x2 = 0;  for (int j = 0; j < n + i; j++)  {  if (huffman[j].weight < m1 && huffman[j].parent == -1)  {  m2 = m1;  x2 = x1;  m1 = huffman[j].weight;  x1 = j;  }  else if (huffman[j].weight < m2 && huffman[j].parent == -1)  {  m2 = huffman[j].weight;  x2 = j;  }  }  huffman[x1].parent = n + i;  huffman[x2].parent = n + i;  huffman[n + i].weight = huffman[x1].weight + huffman[x2].weight;  huffman[n + i].lchild = x1;  huffman[n + i].rchild = x2;  }  }  void huffmanEncoding()  {  Code cd;  int child, parent;  for (int i = 0; i < n; i++)  {  cd.value = huffman[i].value;  cd.start = n - 1;  child = i;  parent = huffman[child].parent;  while (parent != -1)  {  if (huffman[parent].lchild == child)  {  cd.bit[cd.start] = 0;  }  else  {  cd.bit[cd.start] = 1;  }  cd.start--;  child = parent;  parent = huffman[child].parent;  }  for (int j = cd.start + 1; j < n; j++)  {  huffmanCode[i].bit[j] = cd.bit[j];  }  huffmanCode[i].start = cd.start;  huffmanCode[i].value = cd.value;  }  }  void printHuffmanCode()  {  for (int i = 0; i < n; i++)  {  cout << "第" << i + 1 << "个字符" << huffmanCode[i].value << "的Huffman编码为：";  for (int j = huffmanCode[i].start + 1; j < n; j++)  {  cout << huffmanCode[i].bit[j];  }  if (i != n - 1)  cout << endl;  }  cout << "。" << endl;  }  void HuffmanDecoding(string s)  {  vector<string> v;  int ok = 1;  for (int i = 0; i < s.length();)  {  int x = (2 \* n) - 1 - 1;  while (huffman[x].lchild != -1 && huffman[x].rchild != -1)  {  if (s[i] == '0')  {  x = huffman[x].lchild;  }  else  {  x = huffman[x].rchild;  }  i++;  if (i == s.length() && huffman[x].lchild != -1)  {  ok = 0;  break;  }  }  if (ok)  {  v.push\_back(huffman[x].value);  }  }  if (ok)  {  for (int i = 0; i < v.size(); i++)  {  cout << v[i];  }  cout << endl;  }  else  {  cout << "解码有误。" << endl;  }  }  int main()  {  while (true)  {  cout << "字符串个数(0退出)：";  cin >> n;  if (!n)  {  break;  }  initHuffmanTree();  for (int i = 0; i < n; i++)  {  cout << i + 1 << "/" << n << ":";  cin >> huffman[i].value;  cin >> huffman[i].weight;  }  huffmanTree();  huffmanEncoding();  printHuffmanCode();  while (true)  {  cout << "0,1序列(q退出)：";  string s;  cin >> s;  if (s[0] == 'q')  {  cout << endl;  break;  }  cout << "解码后为：";  HuffmanDecoding(s);  }  }  return 0;  }  3、实验结果与结论  1）    2）    3） | | | | | |

**实 验 报 告**

|  |
| --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  (一)实验中遇到的主要问题及解决方法  问题：在题目一中遇到一些自己到现在都没看懂的问题  解决办法：在询问身边的同学后，将后序遍历更名为PostOrder、element换成Data等修改后解决了问题…  （二）实验心得  不能盲目相信教材  （三）意见与建议（没有可省略） |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. **支撑毕业要求指标点**   《数据结构》课程支撑毕业要求的指标点为：  1.2-M掌握计算机软硬件相关工程基础知识，能将其用于分析计算机及应用领域的相关工程问题。  3.2-H能够根据用户需求，选取适当的研究方法和技术手段，确定复杂工程问题的解决方案。  4.2-H能够根据实验方案，配置实验环境、开展实验，使用定性或定量分析方法进行数据分析与处理，综合实验结果以获得合理有效的结论。   |  |  |  |  | | --- | --- | --- | --- | | 实验内容 | 支撑点1.2 | 支撑点3.2 | 支撑点4.2 | | 线性表及多项式的运算 | √ |  |  | | 二叉树的基本操作及哈夫曼编码译码系统的实现 |  | √ | √ | | 图的基本运算及智能交通中的最佳路径选择问题 |  | √ | √ | | 各种内排序算法的实现及性能比较 | √ |  | √ | | | | | | |
| **六、指导教师评语 (含学生能力达成度的评价)**  如评分细则所示 | | | | | |
| **成 绩** | XX | **批阅人** | **XX** | **日 期** | **XXX** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **评 分 细 则** | **评分项** | **优秀** | **良好** | **中等** | **合格** | **不合格** |
| **遵守实验室规章制度** |  |  |  |  |  |
| **学习态度** |  |  |  |  |  |
| **算法思想准备情况** |  |  |  |  |  |
| **程序设计能力** |  |  |  |  |  |
| **解决问题能力** |  |  |  |  |  |
| **课题功能实现情况** |  |  |  |  |  |
| **算法设计合理性** |  |  |  |  |  |
| **算法效能评价** |  |  |  |  |  |
| **回答问题准确度** |  |  |  |  |  |
| **报告书写认真程度** |  |  |  |  |  |
| **内容详实程度** |  |  |  |  |  |
| **文字表达熟练程度** |  |  |  |  |  |
| **其它评价意见** |  | | | | |
| **本次实验能力达成评价（总成绩）** |  |  |  |  |  |