**《数据结构综合设计》实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 学院 | 人工智能与大数据学院 | 专业 | 虚拟现实技术 | 班级 | 21级2班 | 学生姓名 | 周泽萍 |
| 实验  周次 | 10-12 | 实验  日期 | 2023.5.5 | 学时 | 6 | 教师姓名 | 李昊康 |
| 项目名称 | | 查找的应用 | | | | | |
| 实验  类别 | 🗹验证型实验 🞎设计型实验 🞎综合型实验 🞎其它 | | | | | 成绩：94 | |
| 1. 实验目的及具体要求   实验目的：  1. 掌握查找的不同方法，并能用高级语言实现查找算法；  2. 熟练掌握二叉排序树的构造和查找方法。  3. 熟练掌握静态查找表及哈希表查找方法。  具体要求：  从空的二叉树开始，每输入一个结点数据，就建立一个新结点插入到当前已生成的二叉排序树中。  在二叉排序树中查找某一结点。   1. 实验仪器、设备和材料   装有并能运行VS2019的电脑。   1. 实验内容、步骤及实验数据记录   二叉排序树的建立与查找  #include <conio.h>  #include <math.h>  #include <stdio.h>  #include <stdlib.h>  enum BOOL { False, True };  typedef struct BiTNode //定义二叉树节点结构  {  char data; //为了方便，数据域只有关键字一项  struct BiTNode\* lchild, \* rchild; //左右孩子指针域  }BiTNode, \* BiTree;  BOOL SearchBST(BiTree, char, BiTree, BiTree&); //在二叉排序树中查找元素  BOOL InsertBST(BiTree&, char); //在二叉排序树中插入元素  BOOL DeleteBST(BiTree&, char); //在二叉排序树中删除元素  void Delete(BiTree&); //删除二叉排序树的根结点  void InorderBST(BiTree); //中序遍历二叉排序树，即从小到大显示各元素  int main()  {  BiTree T, p;  char ch, keyword, j = 'y';  BOOL temp;  T = NULL;  while (j != 'n')  {  printf("1.display\n");  printf("2.search\n");  printf("3.insert\n");  printf("4.delete\n");  printf("5.exit\n");  scanf\_s(" %c", &ch); //输入操作选项  switch (ch)  {  case '1':if (!T) printf("The BST has no elem.\n");  else { InorderBST(T); printf("\n"); }  break;  case '2':printf("Input the keyword of elem to be searched(a char):");  scanf\_s(" %c", &keyword); //输入要查找元素的关键字  temp = SearchBST(T, keyword, NULL, p);  if (!temp) printf("%c isn't existed!\n", keyword); //没有找到  else printf("%c has been found!\n", keyword); //成功找到  break;  case '3':printf("Input the keyword of elem to be inserted(a char):");  scanf\_s(" %c", &keyword); //输入要插入元素的关键字  temp = InsertBST(T, keyword);  if (!temp) printf("%c has been existed!\n", keyword); //该元素已经存在  else printf("Sucess to inert %c!\n", keyword); //成功插入  break;  case '4':printf("Input the keyword of elem to be deleted(a char):");  scanf\_s(" %c", &keyword); //输入要删除元素的关键字  temp = DeleteBST(T, keyword);  if (!temp) printf("%c isn't existed!\n", keyword); //该元素不存在  else printf("Sucess to delete %c\n", keyword); //成功删除  break;  default: j = 'n';  }  }  printf("The program is over!\nPress any key to shut off the window!\n");  int n= getchar();  }  void InorderBST(BiTree T)  {//以中序方式遍历二叉排序树T，即从小到大显示二叉排序树的所有元素  if (T->lchild) InorderBST(T->lchild);  printf("%2c", T->data);  if (T->rchild) InorderBST(T->rchild);  }  BOOL SearchBST(BiTree T, char key, BiTree f, BiTree& p)  {//在根指针T所指二叉排序树中递归的查找其关键字等于key的元素，若查找成功  //则指针p指向该数据元素，并返回True,否则指针指向查找路径上访问的最后一  //个结点并返回False,指针f指向T的双亲，其初始调用值为NULL  BOOL tmp1, tmp2;  tmp1 = tmp2 = False;  if (!T) { p = f; return False; } //查找不成功  else if (key == T->data) { p = T; return True; } //查找成功  else if (key < T->data) tmp1 = SearchBST(T->lchild, key, T, p); //在左子树中继续查找  else tmp2 = SearchBST(T->rchild, key, T, p); //在右子树中继续查找  if (tmp1 || tmp2) return True; //若在子树中查找成功，向上级返回True  else return False; //否则返回False  }  BOOL InsertBST(BiTree& T, char e)  {//当二叉排序树T中不存在元素e时，插入e并返回True,否则返回False  BiTree p, s;  if (!SearchBST(T, e, NULL, p)) //查找不成功  {  s = (BiTree)malloc(sizeof(BiTNode));  s->data = e;  s->lchild = s->rchild = NULL;  if (!p) T = s; //被插结点\*s为新的根结点  else if (e < p->data) p->lchild = s; //被插结点\*s为左孩子  else p->rchild = s; //被插结点\*s为右孩子  return True; //成功插入  }  else return False; //树中已存在关键字为e的数据元素  }  BOOL DeleteBST(BiTree& T, char key)  {//若二叉排序树T中存在关键字等于key的数据元素时，则删除该数据元素结点  //并返回True,否则返回False  BOOL tmp1, tmp2;  tmp1 = tmp2 = False;  if (!T) return False; //不存在关键字等于key的数据元素  else  {  if (key == T->data) { Delete(T); return True; }  //找到关键字等于key的数据元素并删除它  else if (key < T->data) tmp1 = DeleteBST(T->lchild, key); //继续在左子树中删除  else tmp2 = DeleteBST(T->rchild, key); //继续在右子树中删除  if (tmp1 || tmp2) return True; //在子树中删除成功，返回True  else return False; //不存在该元素  }  }  void Delete(BiTree& p)  {//在二叉排序树中删除结点p,并重接它的左或右子树  BiTree s, q;  if (!p->rchild) //右子树空，只需重接它的左子树  {  q = p;  p = p->lchild;  free(q);  }  else if (!p->lchild) //左子树空，只需重接它的右子树  {  q = p;  p = p->rchild;  free(q);  }  else //左右子树均不空  {  q = p;  s = p->lchild;  while (s->rchild)  {  q = s; s = s->rchild;  } //转左，然后向右走到尽头  p->data = s->data; //s指向被删结点的“前驱”  if (q != p) q->rchild = s->rchild; //重接\*q的右子树  else q->lchild = s->lchild; //重接\*q的左子树  free(s);  }  }  折半排序  #include <conio.h>  #include <stdio.h>  #define MAX 30 //定义有序查找表的最大长度  typedef struct {  char elem[MAX]; //有序查找表  int length; //length指示当前有序查找表的长度  }SSTable;  void initial(SSTable&); //初始化有序查找表  int search(SSTable, int); //在有序查找表中查找元素  void print(SSTable); //显示有序查找表中所有元素  void main()  {  SSTable ST; //ST为一有序查找表  int ch, loc, flag = 1;  char j;  initial(ST); //初始化有序查找表  while (flag)  {  printf("请选择：\n");  printf("1.显示所有元素\n");  printf("2.查找一个元素\n");  printf("3.退出\n");  scanf\_s(" %c", &j);  switch (j)  {  case '1':print(ST); break; //显示所有元素  case '2': {printf("请输入要查找的元素：");  scanf\_s("%d", &ch); //输入要查找的元素的关键字  loc = search(ST, ch); //查找  if (loc != 0) printf("该元素所在位置是：%d\n", loc); //显示该元素位置  else printf("%d 不存在!\n", ch);//当前元素不存在  break;  }  default:flag = 0;  }  }  printf("程序运行结束!按任意键退出!\n");  }  void initial(SSTable& v)  {//初始化有序查找表  int i;  printf("请输入静态表的元素个数："); //输入有序查找表初始化时的长度  scanf\_s("%d", &v.length);  printf("请从小到大输入%d个元素（整形数）：\n", v.length);  int n =getchar();  for (i = 1; i <= v.length; i++) scanf\_s("%d", &v.elem[i]); //从小到大输入有序查找表的各元素  }  int search(SSTable v, int ch)  {//在有序查找表中查找ch的位置，成功返回其位置，失败返回0  int low, high, mid;  low = 1; high = v.length; //置区间初值  while (low <= high)  {  mid = (low + high) / 2;  if (v.elem[mid] == ch) return mid; //找到待查元素  else if (v.elem[mid] > ch) high = mid - 1; //继续在前半区间进行查找  else low = mid + 1; //继续在后半区间进行查找  }  return 0; //找不到时，i为0  }  void print(SSTable v) //显示当前有序查找表所有元素  {  int i;  for (i = 1; i <= v.length; i++) printf("%d ", v.elem[i]);  printf("\n");  }  分析： | | | | | | | |

说明：1. 实验周次：填写实际上课周，如第5-8周上课填“5-8”或第10周上课填“10”。

1. 实验报告各部分内容需详实填写，按实验指导书上的评分标准给出分数。
2. 实验类型参考实验类型说明文件。