**《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3班 | 学生姓名 | 豆赫婧 |
| 实验  周次 | 10-11 | 实验  日期 | 2023.5.4 | 学时 | 4 | 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：92 | |
| 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：  （一）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。  （二）在二叉排序树中查找某一结点。  二、仪器设备和工具  硬设备：PC机  软件环境：Windows xp Visual C++6.0  三、实验内容、步骤及实验数据记录  1.定义结构  typedef  struct  node  { int  key;  int  other;  struct  node  \*lchild, \*rchild;    } bstnode;  void  inorder ( t )    { if  (t!=Null)  { inorder(t→lchild);    printf(“%4d”, t→key);  inorder(t→rchild);  } }  bstnode  \*insertbst(t, s)  bstnode  \*s,  \*t;  { bstnode  \*f,  \*p;  p=t;   while(p!=Null)  { f=p;  if (s→key= =p→key)  return  t;  if (s→key<p→key)  p=p→lchild;  else  p=p→rchild;   }   if(t= =Null)  return  s;  if (s→key<f→key)  f→lchild=s;  else   f→rchild=s;  return  t;   }    bstnode  \*creatord(  )  { bstnode  \*t, \* s;    int  key;   t=Null;  scanf(“%d”,&key);  while (key!=0)  { s=malloc(sizeof (bitree));   s→key=key;   s→lchild=Null;   s→rchild=Null;  scanf(“%d”, &data);  s→other=data;  t=insertbst(t, s);  scanf(“%d”,&key);    }  return  t;  }  2.折半查找  #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(" %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("%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("%d",&v.length);  printf("请从小到大输入%d个元素（整形数）：\n",v.length);  getchar();  for(i=1;i<=v.length;i++) scanf("%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");  }  3.二叉排序树的建立与查找  #include<stdio.h>  #include<stdlib.h>  #define TRUE 1  #define FALSE 0  #define ERROR 0  #define OK 1  typedef int Status;  typedef int KeyType;  typedef int ElemType;  typedef struct BiTNode {  KeyType data;  struct BiTNode\* lchild, \* rchild;  } BiTNode, \* BiTree;  Status SearchBST(BiTree T, KeyType key, BiTree f, BiTree\* p) {  //二叉排序树查找算法  //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的数据元素，若查找成功  //则根指针p指向该数据元素结点，并返回TRUE，否则指针p指向查找路径上访问的  //最后一个结点并返回TRUE，指针f指向T的双亲，其初始调用值为NULL  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)  return SearchBST(T->lchild, key, T, p);//在左子树中继续查找  else  return SearchBST(T->rchild, key, T, p);//在右子树中继续查找  }  Status InsertBST(BiTree\* T, ElemType e)  {  BiTree p, s;  if (!SearchBST(\*T, e, NULL, &p))  {  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;  }  else  {  return FALSE; // 树中已有关键字相同的结点，不再插入  }  }  Status Delete(BiTree\* p)  {//从二叉排序树中删除结点p ，并重接他的左或者右子树  BiTree q, s;  if (!(\*p)->rchild)  {//右子树为空则只需重接他的左或者右子树  q = \*p;  \*p = (\*p)->lchild;  free(q);  }  else if (!(\*p)->lchild)//左子树为空，重接右子树  {  q = \*p;  \*p = (\*p)->rchild;  free(q);  }  else//左右子树都不空  {  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->lchild;//重接\*p的右子树  else  q->lchild = s->lchild;//重接\*p的左子树  free(s);  }  return OK;  }  Status DeleteBST(BiTree\* T, KeyType key) {  //若二叉排序树上存在关键字等于key的元素，删除该数据元素结点  //并返回true,否则返回FALSE  if (!\*T)  return FALSE;//不存在关键字等于key的元素  else {  if (key == (\*T)->data)  return Delete(T);  else if (key < (\*T)->data)  return DeleteBST(&(\*T)->lchild, key);  else return DeleteBST(&(\*T)->rchild, key);  }  }  Status PrintElement(ElemType e)  {//输出元素e的值  printf("%4d", e);  return OK;  }  Status InOrderTraverse(BiTree T, Status(\*Visit)(ElemType e))  {  if (T)  {  if (InOrderTraverse(T->lchild, Visit))//访问左子树  if (Visit(T->data))//访问根结点  if (InOrderTraverse(T->rchild, Visit))//访问右子树  return OK;  return ERROR;  }  else  return OK;  }  int main()  {  int e, length;  BiTree T, f, p;  T = NULL;  f = NULL;  printf("输入要构造的二叉排序树的长度：");  scanf\_s("%d", &length);  printf("输入元素：");  while (length--)  {  scanf\_s("%d", &e);  InsertBST(&T, e);  }  printf("二叉排序树的中序遍历结果是:");  InOrderTraverse(T, PrintElement);  printf("\n");  printf("输入要查找的元素：");  scanf\_s("%d", &e);  if (SearchBST(T, e, f, &p))  printf("该元素在此二叉排序树中\n");  else  printf("该元素不在此二叉排序树中\n");  printf("输入要插入的元素：");  scanf\_s("%d", &e);  InsertBST(&T, e);  printf("插入元素后二叉排序树的中序遍历结果是:");  InOrderTraverse(T, PrintElement);  printf("\n");  printf("输入要删除的元素:");  scanf\_s("%d", &e);  DeleteBST(&T, e);  printf("插入元素后二叉排序树的中序遍历结果是:");  InOrderTraverse(T, PrintElement);  printf("\n");  system("pause");  return 0;  }   1. 实验结果及分析   折半查找  sd  二叉排序树的建立与查找  批注 2023-04-27 090334  通过本次实验掌握二叉树的特点及其存储方式，二叉树的创建和显示方法，复习二叉树遍历的概念，二叉树遍历的基本方法，求二叉树的叶子节点数、树的总结点数和树的深度等基本算法。 | | | | | | | |

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。