**《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 | 冯艺婷 |
| 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：93 | |
| 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   1.顺序查找：  #pragma warning(disable : 4996)  #pragma warning(disable : 6031)  #include <stdio.h>  #define MAX\_NUM 50 //查找表中数据元素最大数量  typedef struct {  int key; //关键字域  }ElemType; //定义数据元素类型  typedef struct {  ElemType elem[MAX\_NUM]; //从下标为1的分量开始存储(0位置设置监视哨)  int length; //表长  }SST; //Sequence Search Table-顺序查找表  /\*顺序查找算法,ST为目标查找表,key为给定的关键字\*/  int Seq\_Search(SST ST, int key)  {  ST.elem[0].key = key; //设置监视哨  int i;  for (i = ST.length; ST.elem[i].key != key; i--)  {  ; //for循环内为空语句  }  return i; //返回0:查找失败;返回值>0:查找成功  }  int main()  {  int i, n, key;  printf("请输入查找表中记录个数n：\n");  scanf("%d", &n);  SST ST;  ST.length = n;  printf("\n请输入查找表中数据记录：\n");  for (i = 1; i < ST.length + 1; i++)  scanf("%d", &(ST.elem[i].key));  printf("\n请输入查找的关键字key：\n");  scanf("%d", &key);  /\*顺序查找\*/  int m = Seq\_Search(ST, key);  if (m)  printf("\n查找成功！该记录的位置为：%d", m);  else  printf("\n查找失败！查找表中没有该记录");  return 0;  }  批注 2023-05-05 085746  2.折半查找：  #pragma warning(disable : 4996)  #pragma warning(disable : 6031)  #include <stdio.h>  #define MAX\_NUM 50 //查找表中数据元素最大数量  typedef struct {  int key; //关键字域  }ElemType; //定义数据元素类型  typedef struct {  ElemType elem[MAX\_NUM]; //从下标为1的分量开始存储(0位置设置监视哨)  int length; //表长  }SST; //Sequence Search Table-顺序查找表  /\*折半查找算法,ST为目标查找表,key为给定的关键字\*/  int Bin\_Search(SST ST, int key)  {  int low = 1;  int high = ST.length;  int mid;  while (low <= high)  {  mid = (low + high) / 2;  if (key == ST.elem[mid].key)  return mid;  if (key < ST.elem[mid].key)  high = mid - 1;  if (key > ST.elem[mid].key)  low = mid + 1;  }  return 0; //返回0:查找失败;返回值>0:查找成功  }  /\*折半查找的递归算法\*/  //ST为目标查找表,key为给定的关键字,low为查找区间的开始下标,high为查找区间的结束下标,\*/  int Bin\_Search\_R(SST ST, int key, int low, int high)  {  if (low > high)  return 0; //查找失败  int mid;  mid = (low + high) / 2;  if (key == ST.elem[mid].key)  return mid;  if (key < ST.elem[mid].key)  Bin\_Search\_R(ST, key, low, high - 1);  if (key > ST.elem[mid].key)  Bin\_Search\_R(ST, key, low + 1, high);  }  int main()  {  int i, n, key;  printf("请输入查找表中记录个数n：\n");  scanf("%d", &n);  SST ST;  ST.length = n;  printf("\n请输入查找表中数据记录：\n");  for (i = 1; i < ST.length + 1; i++)  scanf("%d", &(ST.elem[i].key));  printf("\n请输入查找的关键字key：\n");  scanf("%d", &key);  /\*折半查找\*/  int m = Bin\_Search(ST, key);  if (m)  printf("\n折半查找：\n查找成功！该记录的位置为：%d", m);  else  printf("\n折半查找：\n查找失败！查找表中没有该记录");  /\*折半查找-递归版\*/  int k = Bin\_Search\_R(ST, key, 1, ST.length);  if (k)  printf("\n折半查找(递归算法)：\n查找成功！该记录的位置为：%d", k);  else  printf("\n折半查找(递归算法)：\n查找失败！查找表中没有该记录");  return 0;  }  批注 2023-05-05 090448  3.哈希查找：  #pragma warning(disable : 4996)  #pragma warning(disable : 6031)  #include<stdio.h>  #include<malloc.h>  #define MAXSIZE 25 //哈希表中元素最大数目  #define P 13 //除留余数法的除数  #define OK 1  #define ERROR 0  #define DUPLICATE -1  #define TRUE 1  #define FALSE 0  typedef struct {  int key; /\*关键字值\*/  int flag; /\*是否存放元素\*/  }ElemType; /\*哈希表元素结构\*/  typedef struct {  ElemType data[MAXSIZE];  int count; /\*元素个数\*/  int sizeindex; /\*当前哈希表容量\*/  }HashTable; /\*哈希表\*/  int d1[15] = { 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14 }; /\*线性探测序列\*/  int d2[15] = { 0,1,-1,2 \* 2,-2 \* 2,3 \* 3,-3 \* 3,4 \* 4,-4 \* 4,5 \* 5,-5 \* 5,6 \* 6,-6 \* 6,7 \* 7,-7 \* 7 }; /\*二次探测序列\*/  void dataset(int ds[], int\* len); /\*输入查找表\*/  int InsertHash(HashTable\* H, int e, int d[]); /\*计算哈希地址，插入哈希表\*/  int CreateHash(HashTable\* H, int ds[], int len, int d[]); /\*构造哈希表\*/  int SearchHash(HashTable\* H, int e, int d[]); /\*在哈希表中查找\*/  void menu(); /\*演示菜单\*/  /\*输入查找表\*/  void dataset(int ds[], int\* len) { //查找表关键字通过ds数组返回,查找表长度通过指针len返回  int n, m;  n = 0;  printf("\n查找表输入(输入一个非整数结束)：");  //scanf()返回值表示正确输入参数的个数  while (scanf("%d", &m) == 1) /\*以输入一个非整数作为结束\*/  {  ds[n] = m; //ds数组存放查找表关键字  n++; //n记录个数  }  \*len = n;  }  /\*计算哈希地址，插入哈希表\*/  int InsertHash(HashTable\* H, int e, int d[]) { //H为目标哈希表,e为插入的关键字,d数组指向线性或二次探测序列  int k, i = 1;  k = e % P; //除留余数法计算地址  while (H->data[k].flag == TRUE || k < 0) //发生冲突or计算得到的位置<0  {  k = (e % P + d[i]) % MAXSIZE;  i++;  if (i >= 15)  return ERROR; //探测序列所有值都尝试后仍未找到合适地址,则插入失败,返回0  }  //成功找到空地址  H->data[k].key = e;  H->data[k].flag = TRUE;  H->count++; //插入成功,哈希表长度+1  return OK; //返回1  }  /\*构造哈希表\*/  int CreateHash(HashTable\* H, int ds[], int len, int d[]) { //ds数组存放查找表的关键字,len为记录个数,d数组指向线性或二次探测序列  int i;  for (i = 0; i < len; i++)  {  if (SearchHash(H, ds[i], d) != -1)  return DUPLICATE; //查找成功,说哈希表中已存在该关键字,故创建失败,返回-1  //查找失败,说明不存在重复关键字,可以插入  InsertHash(H, ds[i], d);  if (H->count >= MAXSIZE)  return ERROR; //记录个数超过MAXSIZE,返回0  }  return OK; //创建成功,返回1  }  /\*初始化哈希表\*/  void InitHash(HashTable\* H) {  int i;  for (i = 0; i < MAXSIZE; i++)  {  H->data[i].key = 0;  H->data[i].flag = FALSE;  }  }  /\*在哈希表中查找\*/  int SearchHash(HashTable\* H, int e, int d[]) { //e为查找的关键字,d数组指向线性或二次探测序列(查找和插入应使用同一个探测序列)  int k, i = 1;  k = e % P;  while (H->data[k].key != e || k < 0)  {  k = (e % P + d[i]) % MAXSIZE;  i++;  if (i >= 15)  return -1; //探测序列中所有值都尝试过仍未找到关键字e,说明哈希表中不存在该关键字,查找失败,返回-1  }  return k; //查找成功,返回该关键字在哈希表中的位置  }  /\*演示菜单\*/  void menu() {  int choice;  int\* p;  HashTable h;  h.count = 0;  h.sizeindex = MAXSIZE;  int a[MAXSIZE] = { 0 };  int i, n, e;  dataset(a, &n); /\*建立查找表\*/  getchar(); //消耗回车键  printf("\n");  do {  printf("\n----哈希查找演示----\n");  printf("\n1.线性探测构造哈希表\n");  printf("\n2.二分探测构造哈希表\n");  printf("\n3.退出\n");  printf("\n输入选择：");  scanf("%d", &choice);  if (choice == 1)  p = d1; //p指向线性探测序列  else if (choice == 2)  p = d2; //p指向二次探测序列  else  return;  InitHash(&h); /\*初始化哈希表\*/  i = CreateHash(&h, a, n, p); /\*构造哈希表\*/  if (!i)  printf("\n哈希表构造失败！\n");  else if (i == DUPLICATE)  printf("\n哈希表具有重复关键字！\n");  else  {  printf("\n哈希表：\n");  for (i = 0; i < h.sizeindex; i++)  printf("%3d", h.data[i].key);  printf("\n\n哈希查找\n输入要查找的key值：");  getchar(); //消耗回车键  scanf("%d", &e);  if ((i = SearchHash(&h, e, p)) == -1)  printf("\n%d未找到\n", e);  else  printf("\n%d在哈希表中下标为%d\n", e, i);  }  getchar();  } while (1);  }  int main()  {  menu();  return 0;  }  批注 2023-05-05 091233  批注 2023-05-05 091250  批注 2023-05-05 091304  4.二叉排序树的建立与查找:  #define \_CRT\_SECURE\_NO\_WARNINGS  #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  void main()  {  BiTree T, p;  char ch, keyword, j = 'y';  BOOL temp;  T = NULL;  while (j != 'n')  {  printf("1.陈列\n");  printf("2.查找\n");  printf("3.插入\n");  printf("4.删除\n");  printf("5.退出\n");  scanf(" %c", &ch); //输入操作选项  switch (ch)  {  case '1':if (!T) printf("列表没有元素.\n");  else { InorderBST(T); printf("\n"); }  break;  case '2':printf("输入要搜索的关键字:");  scanf(" %c", &keyword); //输入要查找元素的关键字  temp = SearchBST(T, keyword, NULL, p);  if (!temp) printf("%c 不存在！\n", keyword); //没有找到  else printf("%c 已找到!\n", keyword); //成功找到  break;  case '3':printf("输入要插入的关键字(a char):");  scanf(" %c", &keyword); //输入要插入元素的关键字  temp = InsertBST(T, keyword);  if (!temp) printf("%c 已经存在!\n", keyword); //该元素已经存在  else printf("成功插入 %c!\n", keyword); //成功插入  break;  case '4':printf("输入要删除元素的关键字:");  scanf(" %c", &keyword); //输入要删除元素的关键字  temp = DeleteBST(T, keyword);  if (!temp) printf("%c 该元素不存在!\n", keyword); //该元素不存在  else printf("成功删除 %c\n", keyword); //成功删除  break;  default: j = 'n';  }  }  printf("结束\n按任意键关闭窗口!\n");  getchar(); 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);  }  }  批注 2023-05-05 093040 批注 2023-05-05 093057 批注 2023-05-05 093120  批注 2023-05-05 093138 批注 2023-05-05 093153  分析：  顺序查找对查找表无任何要求，既适合无序查找表，又适合有序查找表，其查找成功的平均长度为（n+1）/2，时间复杂度为O（n）。  折半查找要求表中元素必按关键字有顺序，其平均查找长度为近似log2（n+1)-1，时间复杂度为O（log2n)。  常用的解决冲突的方法有：线性探测法、平方探测法、连地址法等。 | | | | | | | |

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。