实验四

|  |  |  |
| --- | --- | --- |
| 姓名：周兴 | | 学号：202318140413 |
| 班级：人工智能2302班 | | 专业：人工智能 |
| 实验名称： | 线性表的查找 | |
| 实验目的： | 1．掌握查找的定义；  2．掌握线性表的查找操作。 | |
| 实验要求： | | |
| (1) 程序要添加适当的注释，程序的书写要采用缩进格式。  (2) 程序要具在一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如插入删除时指定的位置不对等等。  (3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。  （4）每项任务给出代码及运行结果截图 | | |
| 实验内容： | | |
| 用程序实现以下算法：  1．设置监视哨的顺序查找；  2．折半查找；  查找表可以用整型数组代替。 | | |
| 实验代码及运行结果截图 | | |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  typedef int KeyType;  typedef char Infotype[100];  typedef struct {  KeyType key;  Infotype otherinfo;  } ElemType;  // 静态搜索表类型定义  typedef struct {  ElemType\* R;  int length;  int capacity; // 添加容量字段，用于记录数组的大小  } SSTable;  // 初始化静态搜索表  void InitSSTable(SSTable\* table, int capacity) {  table->capacity = capacity;  table->R = (ElemType\*)malloc(sizeof(ElemType) \* (capacity + 1)); // 多分配一个位置用于监视哨  if (table->R == NULL) {  exit(EXIT\_FAILURE); // 分配内存失败时退出程序  }  table->length = 0;  }  // 插入元素到静态搜索表  bool InsertElem(SSTable\* table, KeyType key, const Infotype otherinfo) {  if (table->length >= table->capacity) {  return false; // 表已满，无法插入  }  table->R[table->length].key = key;  strncpy(table->R[table->length].otherinfo, otherinfo, sizeof(Infotype) - 1);  table->R[table->length].otherinfo[sizeof(Infotype) - 1] = '\0'; // 确保字符串以null结尾  table->length++;  return true;  }  // 无序数列设置监视哨的顺序查找(把所找元素放入监视哨，后往前依次比较,所以时间复杂度O(n))  bool SeqSearch(SSTable\* table, KeyType key, Infotype otherinfo) {  table->R[0].key = key; // 设置监视哨  int i;  for (i = table->length; table->R[i].key != key; i--);  if (i > 0) {  strncpy(otherinfo, table->R[i].otherinfo, sizeof(Infotype) - 1);  otherinfo[sizeof(Infotype) - 1] = '\0'; // 确保字符串以null结尾  return true;  }  return false;  }  // 有序数列使用折半查找，分一半，如果大于向右查找，小于向左查找，继续二分（O(log n)）  bool BinarySearch(const SSTable\* table, KeyType key, Infotype otherinfo) {  int low = 0, high = table->length - 1;  while (low <= high) {  int mid = (low + high) / 2;  if (table->R[mid].key == key) {  strncpy(otherinfo, table->R[mid].otherinfo, sizeof(Infotype) - 1);  otherinfo[sizeof(Infotype) - 1] = '\0'; // 确保字符串以null结尾  return true;  } else if (table->R[mid].key < key) {  low = mid + 1;  } else {  high = mid - 1;  }  }  return false;  }  // 释放静态搜索表占用的内存  void FreeSSTable(SSTable\* table) {  free(table->R);  table->R = NULL;  table->length = 0;  table->capacity = 0;  }  // 比较函数，用于qsort排序  int CompareElemType(const void\* a, const void\* b) {  return ((ElemType\*)a)->key - ((ElemType\*)b)->key;  }  int main() {  SSTable table;  InitSSTable(&table, 10);  printf("=======202318140413========\n");  InsertElem(&table, 5, "Math");  InsertElem(&table, 3, "AI Intro");  InsertElem(&table, 8, "Algebra");  InsertElem(&table, 1, "Zhouxing");  InsertElem(&table, 6, "TOEFL");  Infotype info;  // 顺序查找测试  if (SeqSearch(&table, 1, info)) { //这里查找key为3对应的数据  printf("Found: %s\n", info);  } else {  printf("Not found.\n");  }  // 需要在折半查找前对表进行快速排序  qsort(table.R + 1, table.length, sizeof(ElemType), CompareElemType);  // 折半查找测试  if (BinarySearch(&table, 6, info)) {  printf("Found: %s\n", info);  } else {  printf("Not found.\n");  }  FreeSSTable(&table);  return 0;  } | | |
| 实验结论：  学习了静态顺序表的顺序查找和折半查找 | | |
|  | | |