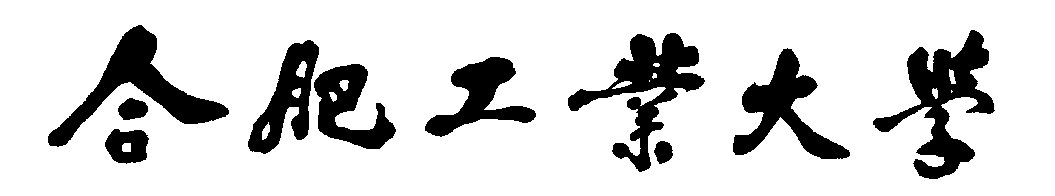
****

计算机与信息学院

数据结构实验报告

|  |  |
| --- | --- |
| 专 业 班 级 | 计算机科学与技术21-1 |
| 学生姓名及学号 | 杨程锦 2021214710 |
| 课程教学班号 |  |
| 任 课 教 师 | 胡学刚 |
| 实验指导教师 | 李培培 |
| 实验地点 | C201 |
| 2021 ~2022 学年第 二 学期 | |

实验序号及名称：实验 七 查找与排序综合运用

实验时间∶ 年 月 日

|  |
| --- |
| 预习内容 |
| 一、实验目的和要求∶  **（1）掌握不同存储方式的查找与排序算法原理；**  **（2）掌握查找与排序算法在大数据上的应用；**  **（3）掌握算法的时间和空间性能的分析方法。** |
| 二、实验任务∶  **<1> 设计一个查找类，并基于此类实现以下功能：**  **（1）设计实现基于顺序表的查找算法（1~2种）；**  **（2）设计实现基于树表的查找算法（1~2种）；**  **（3）设计实现2种基于散列表的查找算法：**  **（4）利用随机函数产生长度不少于N（N=10000）的数据，进行上述算法的验证，并给出时间消耗对比。**  **<2> 设计一个排序类，并基于此类实现以下功能：**  **（1）设计实现基于顺序表的排序算法（1~2种） ；**  **（2）设计实现基于树表的堆排序算法；**  **（3）设计实现基于链式基数排序算法 ；**  **（4）利用随机函数产生长度不少于N（N=10000）的数据，进行上述算法的验证，并给出时间消耗对比。** |
| 三、实验准备方案，包括以下内容：  （硬件类实验：实验原理、实验线路、设计方案等）  （软件类实验：所采用的核心方法、框架或流程图及程序清单）  任务一——基于顺序表的查找与排序  顺序表：  查找：通过直接遍历  排序：通过快速排序    任务二——基于树表的查找与排序  树表：先用一个一维数组储存输入的数据，递归插入平衡二叉树中  查找：中序遍历树表  排序：将待排序序列构造成一个大顶堆,此时,整个序列的最大值就是堆顶的根节点。将其与末尾元素进行交换,此时末尾就为最大值。然后将剩余n-1个元素重新构造成一个堆,这样会得到n个元素的次小值。递归执行得到一个有序序列。  任务三——基于散列表的查找与排序  散列表：  1.拉链法：定义node结构体，包括data和next，构建一维数组，按照数的最后一位数，使对应的一维数组的  手机屏幕截图  中度可信度描述已自动生成  查找方式：  图示  描述已自动生成  2.线性探测法  表格  描述已自动生成  查找方法：先通过线性探测，大致定位数据，后向后遍历数组  图示  描述已自动生成  任务四——链式基数排序  先按照个位数划分区间，在小区间内排序，后按照十位，百位进行划分，最后一轮排序获得的就是有序序列 |

|  |
| --- |
| 实验内容 |
| 一、实验用仪器、设备：  Legion R9000p，cLion |
| 二、实验内容与步骤（过程及数据记录）：  任务一——基于顺序表的查找与排序  查找  void test1() {  cout << "请输入数据长度" << endl;  int\* x, a, b, c = 0;  cin >> a;  x = new int[a];  for (int o = 0; o < a; o++) {  x[o] = rand() % 1000;  cout << x[o]<<" ";  }  cout << "请输入待查找的数据" << endl;  cin >> b;  for (int o = 0; o < a; o++) {  if (x[o] == b) {  cout << "第" << o + 1 << "个数据=" << b << endl;  c++;  }  }  if (!c)cout << "未找到" << b << endl;  }    排序  void test01() {//////基于顺序表查找  int\* x, length;  cout << "请输入数据长度" << endl;  cin >> length;  x = new int[length];  for (int o = 0; o < length; o++) {  x[o] = rand() % 1000;  }  cout << "排序前" << endl;  for (int o = 0; o < length; o++) {  cout << setw(6) << x[o];  if ((o + 1) % 10 == 0)cout << endl;  }  for (int o = length - 1; o >= 0; o--) {  for (int p = 1; p <= o; p++) {  if (x[p] < x[p - 1]) {  int temp = x[p];  x[p] = x[p - 1];  x[p - 1] = temp;  }  }  }  cout << "排序后" << endl;  for (int o = 0; o < length; o++) {  cout << setw(6) << x[o] << " ";  if ((o + 1) % 10 == 0)cout << endl;  }  }    结果正常  任务二：树表的查找排序  查找：  void test2() {  class btree  {  public:  btree\* left;  btree\* right;  int data;  btree(int i) :left(NULL), right(NULL), data(i) {}  void insert(int a) {  if (a < data && left)  left->insert(a);  else if (a < data && !left)  left = new btree(a);  else if (a > data && right)  right->insert(a);  else if (a > data && !right)  right = new btree(a);  };  static void inorder(const btree\* b, int find) {  int mm = 0;  inorder\_(b, mm, find);  }  static void inorder\_(const btree\* b, int mm, int find) {  if (b != NULL)  {  if (find <= b->data)  inorder\_(b->left, mm, find);  //cout << setw(4) << b->data;  if (b->data == find)cout << "找到了" << endl;  if (find >= b->data)  inorder\_(b->right, mm, find);  }  };//中序遍历  ~btree()  {  if (left)  delete left;  if (right)  delete right;  }  };  int\* zu, mm, f;  cout << "请输入数据长度" << endl;  cin >> mm;  zu = new int[mm];  for (int i = 0; i < mm; ++i)  {  zu[i] = rand() % 1000;  }  btree\* root = new btree(zu[0]);  for (int i = 1; i < mm; ++i)  {  root->insert(zu[i]);  }  cout << "请输入要找的数据" << endl;  cin >> f;  btree::inorder(root, f);  cout << endl;  delete root;  }    结果正常  排序：  void test02() {  class btree  {  public:  btree\* left;  btree\* right;  int data;  btree(int i) :left(NULL), right(NULL), data(i) {}  void insert(int a) {  if (a < data && left)  left->insert(a);  else if (a < data && !left)  left = new btree(a);  else if (a > data && right)  right->insert(a);  else if (a > data && !right)  right = new btree(a);  };  static void inorder(const btree\* b) {  int mm = 0;  inorder\_(b, mm);  }  static void inorder\_(const btree\* b, int mm) {  if (b != NULL)  {  inorder\_(b->left, mm);  cout << setw(4) << b->data;  inorder\_(b->right, mm);  }  };//中序遍历  ~btree()  {  if (left)  delete left;  if (right)  delete right;  }  };  int\* zu, mm;  cout << "请输入数据长度" << endl;  cin >> mm;  zu = new int[mm];  for (int i = 0; i < mm; ++i)  {  zu[i] = rand() % 1000;  }  btree\* root = new btree(zu[0]);  for (int i = 1; i < mm; ++i)  {  root->insert(zu[i]);  }  btree::inorder(root);  cout << endl;  delete root;  }    结果正常  任务三：散列表的查找：  void test3() {  typedef int T;  struct node {  T data;  node\* next;  };  class ll {  private:  node x[10];  public:  ll() {  for (int o = 0; o < 10; o++) {  x[o].data = o;  }  }  void insert\_num(T data\_) {  int y = data\_ % 10;  node\* p = new node;  p->data = data\_;  p->next = NULL;  node\* q = x[y].next;  x[y].next = p;  p->next = q;  }  void find(int data\_) {  int y = data\_ % 10;  node\* p = &x[y];  p = p->next;  while (p != NULL) {  if (p->data == data\_) {  cout << "找到了" << endl;  return;  }  p = p->next;  }  cout << "未找到" << endl;  }  };  ll x;  cout << "请输入数据长度" << endl;  int length, num;  cin >> length;  for (int o = 0; o < length; o++) {  x.insert\_num(rand() % 1000);  }  cout << "请输入需要查找的数据" << endl;  cin >> num;  x.find(num);  }    void test3\_() {  const int p\_ = 100000;  cout << "请输入数据长度" << endl;  int length, \* node, temp;  cin >> length;  node = new int[length \* 12 / 10];  temp = length / 2;  for (int o = 0; o < length; o++) {  node[o] = p\_;  }  for (int p = 0; p < length; p++) {  int insert = rand() % 1000;  int insert\_ = insert % temp;  if (p\_ == node[insert\_]) {  node[p] = insert;  }  else {  for (int mm = insert + 1; mm < length; mm++) {  if (p\_ == node[mm]) {  node[p] = insert;  }  }  }  }  cout << "请输入需要查找的数据" << endl;  int find;  cin >> find;  for (int pp = find % temp; pp < length; pp++) {  if (node[pp] == find) {  cout << "找到了" << endl;  return;  }  }  cout << "没有找到" << endl;  }    结果正常  任务四——链式基数排序  void test03() {  struct Node  {  int data;//数据域  Node\* next;//指针域  };  //定义程序所需的特殊队列  class Queue  {  private:  Node\* front;//链式对列的头指针  Node\* rear;//链队的尾指针  public:  //构造函数，初始化队列（带头结点的链式队列）  Queue()  {  //开始先构造一个空结点，没有数据元素存储  Node\* p = new Node;  p->data = NULL;  p->next = NULL;  //开始是空链队，首尾指针分别去指向队头结点  front = p;  rear = p;  }  //析构函数，销毁链队的结点占据的内存  ~Queue()  {  //标记指针  Node\* p = front;  //辅助的标记指针，作用是删除结点  Node\* q;  //循环遍历整个队列，直到标记指针 p 为 null  while (p != NULL)  {  //比较常见的删除结点内存的写法  q = p;  //指向队列的下一个结点  p = p->next;  //销毁之  delete q;  }  }  //入队方法，从尾进入，节点不存在，需要自行创建结点的方法  void push(int e)  {  Node\* p = new Node;  p->data = e;  //本结点作为了队列的尾结点  p->next = NULL;  //然后连接结点到队尾  rear->next = p;  //最后尾指针指向新的末位结点  rear = p;  }  //入队方法，尾进入，节点原来就存在的方法，不需要再新建结点和存储结点的内容  void push(Node\* p)  {  //设置此结点为尾结点  p->next = NULL;  //链接结点  rear->next = p;  //尾指针指向新的尾结点  rear = p;  }  //求数据元素的最大位数的方法，也就是求出需要分配和收集的次数  int lengthData()  {  int length = 0;//保存数据元素的 最大位数  int n = 0; //单个数据元素具有的位数  int d; //用来存储待比较的数据元素  //指示指针  Node\* p = front->next;  //遍历  while (p != NULL)  {  //取出结点的数据，也就是代比较的数据元素  d = p->data;  //如果 d 为正数，很重要的一个技巧，必须是 d 大于 0 的判断  while (d > 0)  {  //数据位数分离算法  d /= 10;  //单个数据元素的位数存储在此  n++;  }  //沿着链队后移一个元素  p = p->next;  //找出数据元素的最大位数  if (length < n)  {  length = n;  }  //重新循环往复，n 设置为0  n = 0;  }  //返回最终位数  return length;  }  //判断队列是否为空  bool empty()  {  //队头指针和队尾指针重合，说明空  if (front == rear)  {  return true;  }  //否则为不空  return false;  }  //清除队列中的元素  void clear()  {  //直接把头结点之后的链接断开  front->next = NULL;  //设置尾指针指向头结点即可，回到了构造函数初始化的情景  rear = front;  }  //输出队列中的元素,传入引用参数比较好  void print(Queue& que)  {  //第一个结点是头结点，next 才是第一个存储元素的结点  Node\* p = que.front->next;  //直到尾结点为止  int m = 0;  while (p != NULL)  {  m++;  cout <<p->data << " ";  //遍历所有结点  if (m % 10 == 0)cout << endl;  p = p->next;  }  }  //基数排序过程  void RadixSort(Queue& que)  {  //声明一个指针数组，该指针数组中存放十个指针，这十个指针需要分别指向十个队列，这是模拟10个桶，因为是0-9的数字，取值范围为10  Queue\* arr[10];  //初始化这十个队列  for (int i = 0; i < 10; i++)  {  //初始化建立头结点  arr[i] = new Queue;  }  //取得待排序数据元素中的最大位数  int maxLen = que.lengthData();  //因为是 LSD 方式，从后到前，开始比较关键字，然后分配再收集，故开始设置数据分离算法中的除数为 1  int d = 1;  //将初始队列中的元素分配到十个队列中,maxlen 代表了需要分配和收集的次数  for (int i = 0; i < maxLen; i++)  {  Node\* p = que.front->next;  //辅助指针 q  Node\* q;  //余数为k，则存储在arr[k]指向的链式队列（桶）中  int k;  //遍历原始序列  while (p != NULL)  {  //重要的技巧，数据分离算法过程,最后勿忘模10，取余数，分离出需要的关键字位  k = (p->data / d) % 10;  q = p->next;  //把本结点 p 加入对应的队列中  arr[k]->push(p);  //指针后移，指向下一个结点  p = q;  }  //清空原始队列  que.clear();  //分配完毕，马上将十个队列中的数据收集到原始队列中  for (int i = 0; i < 10; i++)  {  if (!arr[i]->empty())  {  //从首节点开始遍历，不是头结点开始  Node\* p = arr[i]->front->next;  //辅助指针 q  Node\* q;  while (p != NULL)  {  q = p->next;  //收集到原始队列中，这就是为什么每次分配完毕，需要清除原始队列  que.push(p);  p = q;  }  }  }  //一趟的分配收集完毕，最后要清空十个队列  for (int i = 0; i < 10; i++)  {  arr[i]->clear();  }  //进行下一趟的分配和收集  d \*= 10;  }  //输出队列中排好序的元素  print(que);  }  };  Queue oldque;  int i;  cout << "请输入数据长度" << endl;  cin >> i;  //顺序输入元素  while (i != 0)  {  oldque.push(rand() % 1000);  cout << rand() % 1000 << " ";  i--;  }  cout << endl;  //基数排序  oldque.RadixSort(oldque);  return;  } |
|  |
| 三、实验结果分析、思考题解答∶  成功构造了顺序表、树表、哈希表等数据结构。完成了顺序表、树表的查找与排序，哈希表的查找，链式基数排序。 |
| 四、感想、体会、建议∶  感觉整体倒不是特别难，各类数据结构已经构造过类似的了，就是实验内容过多，需要很多时间。随机数貌似在构造的时候出了点问题，rand（）没法做到完全随机，搜了一下写了个随机数函数，用种子配合rand（）返回一个伪随机数  int fun()  {  srand(time(0) + rand());  return rand() % 1000;  }  其他感受不多，就是写的头大。  建议的话，这个实验还好，没啥明显的需要改进的地方 |
| 实验成绩∶  指导教师签名：  年 月 日 |