|  |  |  |  |
| --- | --- | --- | --- |
| **《数据结构与算法》实验报告** | | | |
| **学生姓名** | 郭茁宁 | **院（系）** | 计算机科学与技术 |
| **学 号** | 1183710109 | **专 业** | 软件工程 |
| **实验时间** | 2019年12月20日（周五） | **实验地点** | 格物213室 |
| **实验项目** | **实验4/5：**查找结构的实验比较**（3学时）** | | |
| **实验目的：**将课程的基本原理、技术和方法与实际应用相结合，训练和提高学生组织、存储和处理信息的能力，以及复杂问题的数据结构设计能力和程序设计能力，培养软件设计与开发所需要的实践能力。  **实验要求：**灵活运用基本的数据结构和算法知识，对实际问题进行分析和抽象；结合程序设计的一般过程和方法为实际问题设计数据结构和有效算法；用高级语言对数据结构和算法进行编程实现、调试，测试其正确性和有效性。 | | | |
| **实验内容：BST查找结构与折半查找方法的实现与实验比较**  **本实验要求编写程序实现BST 存储结构的建立（插入）、删除、查找和排序算法；实现折半查找算法；比较BST查找结构与折半查找的时间性能。**  1． 设计BST 的左右链存储结构，并实现BST插入（建立）、删除、查找和排序算法。  2． 实现折半查找算法。  3． 实验比较：设计并产生实验测试数据，考察比较两种查找方法的时间性能，并与理论结果进行比较。以下具体做法可作为参考：  （1）第1组测试数据： n=1024个已排序的整数序列（如0至2048之间的奇数）；第2组测试数据：第1组测试数据的随机序列。  （2）按上述两组序列的顺序作为输入顺序，分别建立BST。  （3）编写程序计算所建的两棵BST的查找成功和查找失败的平均查找长度（主要是改造Search算法，对“比较”进行计数），并与理论结果比较。  （4）以上述BST的中序遍历序列作为折半查找的输入，编写程序分别计算折半查找的查找成功和查找失败的平均查找长度，并与理论结果比较。  （5）以上实验能否说明：就平均性能而言，BST的查找与折半查找差不多，为什么？ | | | |
| 数据结构定义：  class BSTree：二叉搜索树类，拥有add连接、sorted输出排序数列、insert插入结点、del删除结点、find\_count查找并计数等成员函数； | | | |
| 算法设计与分析（要求画出核心内容的程序流程图）：   1. BST插入（建立）   读入n个数后建立一棵二叉搜索树，即把n个数不断插入到已有（一开始是空树）。插入操作：     1. BST删除   每读入一个要删除的数，就先找到该点，判断是如下哪种类型：   * 1. 左右儿子都为空：删除该点，并将其父亲的对应儿子指针指向空；   2. 左儿子为空：删除该点，该点右儿子变为该点父亲的对应新儿子；   3. 右儿子为空：删除该点，该点左儿子变为该点父亲的对应新儿子；   4. 左右儿子都不为空：该点左儿子的最右儿子的的左儿子代替其位置，其值代替删除点的值；  1. BST查找   int find\_count(int *value*, int *cnt*)  {      if (value == *this*->val) return cnt;      return value < val ? (son[0] == NULL ? -cnt : son[0]->find\_count(value, cnt + 1)) : (son[1] == NULL ? -cnt : son[1]->find\_count(value, cnt + 1));  }     1. BST排序   void sorted() // 打印中序遍历，排序结果  {      if (son[0] != NULL) son[0]->sorted();      printf("%-4d", val);      if (son[1] != NULL) son[1]->sorted();  }   1. 折半查找   int Half\_Search(int *x*, int *n*, int *a*[]) // 折半查找  {      int l = 1, r = n, mid, length = 1;      bool flag = false;      while (l <= r)      {          mid = (l + r) / 2;          if (x == a[mid])          {              flag = true;              break;          }          (x < a[mid]) ? r = mid - 1 : l = mid + 1;          length++;      }      return flag ? length : -length;  }     1. 记录平均查找成功/失败长度   int find\_count(int *value*, int *cnt*)  {      if (value == *this*->val) return cnt;      return value < val ? (son[0] == NULL ? -cnt : son[0]->find\_count(value, cnt + 1)) : (son[1] == NULL ? -cnt : son[1]->find\_count(value, cnt + 1));  }  int Half\_Search(int *x*, int *n*, int *a*[]) // 折半查找  {      int l = 1, r = n, mid, length = 0;      bool flag = false;      while (l <= r)      {          length++;          mid = (l + r) / 2;          if (x == a[mid])          {              flag = true;              break;          }          (x < a[mid]) ? r = mid - 1 : l = mid + 1;      }      return flag ? length : -length;  }  在比较中记录次数  length > 0 ? (tree\_succ\_times += 1, tree\_succ\_length += length) : (tree\_fail\_times += 1, tree\_fail\_length += length);  printf("Success: %10d %8.3lf\n", tree\_succ\_times, 1.0 \* tree\_succ\_length / tree\_succ\_times);  printf("Failure: %10d %8.3lf\n", tree\_fail\_times, 1.0 \* tree\_fail\_length / tree\_fail\_times);  length > 0 ? (half\_succ\_times += 1, half\_succ\_length += length) : (half\_fail\_times += 1, half\_fail\_length += length);  printf("Success: %10d %8.3lf\n", half\_succ\_times, 1.0 \* half\_succ\_length / half\_succ\_times);  printf("Failure: %10d %8.3lf\n", half\_fail\_times, 1.0 \* half\_fail\_length / half\_fail\_times);  输出平均查找长度   1. 比较性能   通过计算1亿次查找的成功/失败查找长度/用时，比较性能  sta = clock();      for (int i = 1, x; i <= m; i++)  {  ……      }      end = clock();  printf("It takes %.4lfs totally, %.6lfus per cmp.\n", (double)(end - sta) / CLOCKS\_PER\_SEC, 1000.0 \* (double)(end - sta) / (tree\_succ\_length - tree\_fail\_length)); | | | |
|  | | | |
| 实验测试结果及结果分析：    Tree:  Success: 49999971 12.490  Failure: 50000029 -13.480  It takes 12.5780s totally, 0.009687us per cmp.  Half:  Success: 49999912 9.012  Failure: 50000088 -11.002  It takes 11.7700s totally, 0.011762us per cmp.  可以看出：   1. 二叉搜索树中平均成功查找长度为12.490，平均失败查找长度为13.480 2. 二叉搜索树1亿次查找用时共12.5780秒，平均每次比较用时0.009687微秒 3. 折半查找中平均成功查找长度为9.012，平均失败查找长度为11.002 4. 折半查找1亿次查找用时共11.7700秒，平均每次比较用时0.011762秒   结论：   * 折半查找总的效率高于二叉搜索树，包括查找长度和用时； * 查找过程中的单次比较，二叉搜索树的用时比折半查找要短； * 总体来说，折半查找性能更优，因为它相对于二叉搜索树更为平衡； | | | |
| 问题及解决方法：   1. 二叉搜索树中删除操作，取左边最右儿子，能删除但不够完美； 2. 折半查找中计算查找次数，在每次比较前计数，符合定义； 3. 比较二叉搜索树（指针）和折半查找（数组）的效率，需要计算每次比较的时间，而不是总的时间； | | | |
| 源程序名称：lab4.cpp | | | |

注意：正文文字为宋体小4号，图中文字为宋体5号。行距为多倍行距1.25。

源程序与此报告打包提交，压缩包采用学号命名。