|  |  |  |  |
| --- | --- | --- | --- |
| **《数据结构与算法》实验报告** | | | |
| **学生姓名** |  | **院（系）** | 计算学部 |
| **学 号** |  | **专 业** | 计算机类 |
| **实验时间** | 2021年11月10日（周三） | **实验地点** | 格物213室 |
| **实验项目** | **实验3-3：**查找结构的实验比较**（2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：  ①二叉查找树的定义。结构体celltype包含的内容有：  Int型变量data，表示结点存储的关键字信息。  celltype型指针变量lchild，表示该结点的左孩子结点。  celltype型指针变量rchlid，表示该结点的右孩子结点。 | | | |
| 算法设计与分析（要求画出核心内容的程序流程图）：  **①任务一（BST的左右存储结构及实现BST插入（建立）、删除、查找和排序算法。）**   1. BST的左右存储结构   实现方法：  BST的左右存储结构的实现可通过结构体celltype进行实现。  结构体celltype包含的内容有：  Int型变量data，表示结点存储的关键字信息。  celltype型指针变量lchild，表示该结点的左孩子结点。  celltype型指针变量rchlid，表示该结点的右孩子结点。  这样便完成了BST的左右存储结构的实现。   1. BST插入（建立）算法   I BST插入算法  算法的设计思想：  使用递归，根据BST的结构特点，若要插入的关键字大于结点的关键字值，则说明其应该插入在该结点的右子树当中，就对该结点的右孩子结点再次递归调用该函数；若要插入的关键字值小于结点的关键字值，则说明其应该插入在该结点的左子树当中，就对该结点的左孩子结点再次递归调用该函数。  算法的实现流程：   1. 当前结点若为空，则直接将要插入的关键字放入，插入完成。 2. 若当前结点不为空且要插入的关键字大于该结点的关键字值，递归调用函数访问该结点的右孩子结点。 3. 若当前结点不为空且要插入的关键字小于该结点的关键字值，递归调用函数访问该结点的左孩子结点。   程序流程图：    II BST建立算法  算法设计：BST的建立算法可依靠BST的插入算法进行实现，对要用BST建立的序列，调用BST插入算法，将其依次插入到BST当中，即可完成BST的建立。  三.BST删除算法  算法的设计思想：  首先使用递归找到要删除结点的位置。这部分的算法实现可借鉴BST插入算法的设计，若要删除的关键字小于该结点的关键字值，则访问其左孩子结点。若要删除的关键字值大于该结点的关键字值，则访问其右孩子结点，直到访问到要删除的结点。在访问到结点后，开始进行删除操作。在删除时，分为以下四种情况进行处理：   1. 若被删除结点为叶子结点，直接删除即可 2. 若被删除结点只有左子树，则用左子树继承被删除结点 3. 若被删除结点只有右子树，则用右子树继承被删除结点 4. 若被删除结点既有左子树又有右子树，则用被删除结点右子树中的最小结点继承被删结点   过程4)的算法设计思路：使用递归，访问到被删结点右子树中最左的结点即可。  算法的实现流程：   1. 若删除关键字值k小于访问结点的关键字值，递归调用函数访问该结点的左孩子结点 2. 若删除关键字值k大于访问结点的关键字值，递归调用函数访问该结点的右孩子结点 3. 若删除关键字值k等于访问结点的关键字值，执行以下操作 4. 若该结点左右结点都为空，直接删除该结点，删除操作完成 5. 若该节点左结点不为空，右结点为空，用左孩子结点继承该结点，删除操作完成 6. 若该结点右结点不为空，左节点为空，用右孩子结点继承该结点，删除操作完成 7. 若该节点左结点和右结点均不为空，访问到该结点右子树的最左结点，用其继承被删除结点   程序流程图（在此只展示继承被删除结点过程的算法，查找被删除位置部分的流程图在后续的BST查找算法中）：     1. BST查找算法（添加了计数功能）   算法的设计思想：通过BST查找关键字所在结点的过程可借鉴BST插入算法的设计，利用递归，从根节点开始，将要查找的关键字与当前访问的结点的关键字进行比较，若大于则递归访问当前结点的右孩子结点，若小于则递归访问当前结点的左孩子结点，直到查找到关键字所在结点。借用递归这一算法，可以通过每次调用函数时，计数器加一来间接表示查找的次数。  算法的实现流程：   1. 从根节点开始访问，计数器cnt加一 2. 若访问的结点为空，返回查找失败 3. 若查找的关键字大于访问结点的关键字，递归调用函数访问该结点的右孩子结点 4. 若查找的关键字小于访问结点的关键字，递归调用函数访问该结点的右孩子结点 5. 若查找的关键字等于访问结点的关键字，返回查找成功及该结点   算法的流程图；     1. BST的排序算法   算法的设计思路：根据BST结构的特点，即结点的左孩子结点的关键字值小于结点，结点的右孩子结点的关键字值大于结点，将BST中序遍历，得到的序列即位BST存储的关键字值由小到大的排序，因此BST的排序算法即为BST的中序遍历算法。  算法的实现流程：   1. 递归调用函数访问结点的左孩子结点 2. 输出该结点的关键字值 3. 递归调用函数访问结点的右孩子结点   **②任务二（折半查找算法）**  算法的设计思路：基于二分法的思想，如果中间元素正好是要查找的元素, 则搜索过程结束；如果某一特定元素大于或者小于中间元素, 则在数组大于或小于元素的那一半中查找, 而且跟开始一样从中间元素开始比较. 若某1个步骤中数组为空, 则代表找不到。  算法的实现流程：   1. 初始化low，up，mid结点 2. 访问mid结点，判断其与要查找的关键字的大小关系，若等于要查找的关键字，返回结果。若大于要查找的关键字，进行3)步骤；若小于要查找的关键字，进行4)步骤。 3. 用现在的mid更新up并计算出新的mid，进行步骤2)操作 4. 用现在的mid更新low并计算出新的mid，进行步骤2)操作   算法的流程图：    ③**任务三（实验比较）**  1.测试数据一：按顺序排列的0-2048间的奇数建立BST  计算BST查找成功和查找失败的平均查找长度：    结果分析：由于输入的数据是按顺序排列的0-2048间的奇数，因此所建立出来的树是一棵只有右子树的，高度为1024的树，且元素大小按照根节点往下依次升高，故理论上的查找成功的平均长度计算为：，查找失败的平均长度计算为：，均与测试所得结果理论相符。  2.测试数据二：按随机排列的0-2048间的奇数建立BST  计算BST查找成功和查找失败的平均查找长度：    结果分析：由于输入的数据是随机序列，故输入序列的顺序对查找的平均长度有较大影响，当输入的数据恰好为顺序输入即建立的BST平衡性较差时，其查找的平均长度最长，当输入的数据建立出的BST平衡性较好时，其查找的平均长度最短。当输入的数据够多，随机性够高时，其查找成功与查找失败的平均长度会稳定在一个范围，经测试后得到如图所示的结果。  3.测试数据三：按上述BST的中序遍历结果进行折半查找  计算折半查找成功和查找失败的平均查找长度：    结果分析：关于折半查找的平均查找长度，可借助树来进行理解，故查找成功的平均长度的理论值可表示为第一层到最后一层,层数\*对应层数的成功的结点数累加然后除以成功的结点总数，即=，与测试结果相同。查找失败的平均长度的理论值可表示为倒数第二层的层数\*下一层失败(没有)的结点数加上最后一层的层数\*下一层失败(没有)的结点数(和已有的结点互补)然后除以失败的结点总数，即10.00195，与测试结果相同。  4.BST与折半查找的比较  可以。BST查找和折半查找的时间复杂度都是O(logn),而输入的数据具有一定顺序时，BST的平衡性较差，导致其查找的长度增加。在测试数据一中，当使用顺序序列建树的时候，就会形成一个只有右子树的二叉树，导致查找数据性能下降。而当使用随机序列输入时，在测试数据二中得到的平均查找时间性能和折半查找时间性能相差很小，且数字越随机其查找的平均长度越少。因此，特别四当数据的输入顺序随机性较强时，有很强的理由认为折半查找和BST查找的时间性能相近。 | | | |
| 实验测试结果及结果分析：  ①BST树的建立  输入数据（顺序序列）：    建立BST的中序遍历结果：    输入数据（随机序列）：    建立BST的中序遍历结果：    结果分析：所设计的BST存储结构能够顺利完成树的建立。  ②BST的删除  输入要删除的元素：    删除后BST的中序遍历结果：    输入要删除的元素：    结果分析：可以观察到关键字3已经从BST中删除，说明程序能够顺利执行BST删除操作。且当要删除的关键字不存在时，能够返回提示消息。  ③BST查找  输入要查找的元素：      结果分析：可以观察到，当输入的关键字可被查找到时，会提示信息“查找成功”并返回比较次数，而当输入的关键字不存在时，会返回提示“查找失败”及比较次数。  ④BST插入  输入要插入的元素：    插入后的中序遍历结果：    结果分析：可以观察到关键字2被成功插入到BST中，说明程序能够顺利执行BST插入操作。  ⑤折半查找：  输入要查找的元素：    结果分析：可以观察到能够顺利查找到关键字5并返回其下标，查找的比较次数，说明程序可以顺利执行折半查找操作。 | | | |
| 问题及解决方法：  问题一：随机输入建立的BST平均查找长度过大  解决方法：经过对问题的仔细分析，发现是输入的随机序列其随机性不够高，从而造成BST的平衡性较差，进行导致查找的效率降低。通过对随机序列生成的改进，增强其随机性，最终得到了良好的平均查找长度，解决了问题。  问题二：一开始不知道如何完成查找长度的计数  解决方法：借助BST的搜索算法，每次进行搜索算法时，都在计数器上加一，这样便可以通过计数器的值来得知进行了多少次搜索。最终顺利完成这部分的算法设计，得到了查找成功平均长度，查找失败平均长度。 | | | |
| 源程序名称：实验3.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。