苏州大学实验报告

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | | 20计科 | | 姓名 | 柯骅 | 学号 | 2027405033 |
| 课程名称 | | 数据结构实践 | | | | | | | 成绩 |  |
| 指导教师 | | 孔芳 | | 同组实验者 | | 无 | | 实验日期 | 2021.12.11 | |

1. 题目回顾

产生一个菜单驱动的演示程序，用以说明二叉树的使用。元素由单个键组成，键为单个 字符。用户能演示的二叉树基本操作至少包括：构造二叉树，按先序、中序、后序、 层序遍历这棵二叉树，求二叉树的深度、宽度，统计度为 0，1，2 的结点数等。二 叉树采用链式存储结构。对二叉查找树做上述工作，且增加以下操作：插入、删除给 定键的元素、查找目标键。

1. 数据结构设计
2. 对于每个结点，定义一种结构体Node/BNode，其中有存放数据的一个变量，和两个指向结构体本身的两个指针，用于存放左孩子和右孩子的地址。
3. 对于空结点，只需要让指向空结点的指针为nullptr即可
4. 定义两种数据类型，Tree二叉树和BTree二叉查找树，
   1. 数据以Node/BNode链式存储，
   2. 操作有创建、删除、各种序输出、统计深度、宽度、各个度共有多少结点、插入、删除、查找等
5. 对于树的宽度width，定义width\_arr数组，其中width\_arr[i]代表第i层共有多少个节点
6. 对于树的深度depth，定义depth变量存储
7. 对于度数的统计，由于是二叉树，所以度数只可能有0，1，2三种情况，定义degree\_arr数组，其中degree\_arr[i]代表度数为i的结点的个数。
8. 对于层序使用队列，先进先出，使每一层的结点依次进队出队输出，得到层序
9. 各个操作用法：
10. Tree()//根据类型创建树
11. ~Tree()//删除整棵树
12. void preorder()//输出先根顺序
13. void inorder()//输出中根顺序
14. void postorder()//输出后根顺序
15. void levelorder()//输出层序
16. int getdepth()//返回int类型的树的深度
17. int getwidth()//返回int类型的树的最大宽度
18. void getdegree(int &d0,int &d1,int &d2)//引用返回树中度为0，1，2的结点共有多少个
19. void insert(int new\_data)//在树中插入值为new\_data的结点
20. bool search(int target)//返回树中是否存在值为target的点
21. bool remove(int target)//删除树中值为target的点
22. 算法设计
23. 对于类中的函数，大都使用主调函数（public类型）与被调函数（private类型）来实现使得程序更加清晰，合理。
24. 二叉树
    1. 构建：采取先根遍历的顺序读入，在空结点处使用#代替，保证根一定比左右孩子先出现，这样可以使得建树是自上而下，自左而右的顺序，建树较为容易。
    2. 先序/中序/后序遍历：保证对于每一棵子树，它的根都处于左孩子和右孩子的前面/中间/后面，在被调函数中使用递归，根据顺序进行
    3. 层序：使用队列，宽度搜索，使同一层的顶点一定是一起输出的。
    4. 求深度：深度搜索，传递cnt变量记录当前为多少层，并不断更新depth，使得depth一定为cnt本次扫描历史最大值，扫描完毕后，私有成员变量depth即为当前深度。
    5. 求宽度：类似于求深度。深度搜索，使用cnt标记当前层数，并在相应的width\_arr[cnt]处+1，最终扫描完毕后，得到的width\_arr[i]即为第i层有多少个结点，再次扫描width\_arr中每一个值，取其中最大值，更新为width，即为当前树的宽度。
    6. 统计度为0，1，2的结点数：同样类似于求深度。深度搜索每一个点，统计它的度数，并在相应的degree\_arr[i]中+1，最后以引用的方式返回。
25. 二叉查找树
    1. 构建：不同于二叉树。事先传递参数n表示树中结点个数，然后在读入n个数，不断的insert到已有的树中，直到n个数添加完成。
    2. 插入：传递新结点的data：new\_data，从树的根结点开始，不断比较new\_data与当前结点的值，如果new\_data<当前结点的data，那么插入至左边，即以左孩子为新的当前结点不断比较，同理，如果new\_data>当前结点的data，那么以右孩子为新的当前结点不断比较，直到当前结点为空结点，将new\_data插入至那个空结点即可。
    3. 删除：
       1. 要删除的结点没有孩子：直接删除即可
       2. 要删除的结点只有一个孩子：删除该结点，且父亲结点指向被删结点的孩子结点即可。
       3. 要删除的结点有两个孩子：在它的右子树中寻找中序下的第一个节点（关键码最小）， 用它的值填补到被删节点中，再将那个中序下的第一个节点删除即可。
    4. 查找：类似于构建与插入的过程，不断比较target与当前结点的大小，根据大小关系来确定下一步的当前结点是谁，直到找到与target相等的结点查找成功，或空结点查找失败为止。
    5. 其余操作同二叉树。
26. 算法分析
    1. 时间复杂度
27. 二叉树的创建，遍历统计深度，度数为0，1，2的结点数：

由于他们都是扫描一遍树的每个结点即可，所以时间复杂度为O(n)

1. 求树的宽度：除了对每个点扫描一遍之外，还要对于每层宽度进行比较，而最多有n层（即一条链），所以时间复杂度也为O(n),
2. 二叉查找树的构建：由于每次读入进一个新的数，都要对已有树进行一边查找，所以可能产生时间退化：

①当读入的数据随机化时，即当生成的数的层数比较小时，每个数进树的时间复杂度为O(logn)，所以总时间复杂度为O(nlogn).

②当读入的数是有序的，即生成的树是一条链时，每次插入的效率为O(n),所以总复杂度为O(n\*n).

1. 同理，二叉查找树的插入与删除主要费时于查找到相应的位置，时间复杂度在O(logn)到O(n)之间。
   1. 空间复杂度

不论是链式存储每个结点，还是width\_arr都不会产生超过O(n)的空间复杂度，所以空间复杂度是O(n).

1. 运行实例





















