实验 4　二叉树的应用(BST)

计科 1903 201914020128 陈旭

一.　问题分析

1). 问题与功能描述

1. 需要处理的数据是一串整型(int 型)数字组成的一个数组。

2. 实现的功能有:
   - 读入一个整数作为树中元素的值;
   - 读入这 n 个数;
   - 将这 n 个数进行排序;
   - 根据数的个数获取由这组数组成的既满足 BST 树又满足 CBT 树的树的层数，左子树子结点个数和右子树子结点个数。
   - 递归建树;
   - 层次遍历输出该树。

3. 使用标准输出即可。

2). 样例分析

1. **求解方法:** ① 将读入的数组进行排序。

   ② 根据数据个数求解层数和左右子树结点个数以及根所在位置。
   ③ 设置根结点，并设置左右子树，即对从根结点分开的每一子树的数组进行上述操作。

2. **样例求解:**

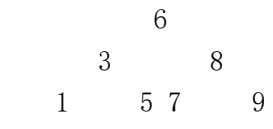
【样例输入】 10

1 2 3 4 5 6 7 8 9 0

【样例输出】 6 3 8 1 5 7 9 0 2 4

【求解过程】 求解过程如下

1. 先排序，得元素序列为 0 1 2 3 4 5 6 7 8 9
2. 计算层数：F=int($\log_2 n$),n 为元素个数。得 F=3；
3. 计算左子树结点个数：
   分为叶结点和分支结点：
   leafNode=n-2^F+1;
   leafNode=3;
   midNode=2^{F-1}-1
   判断左子树中能否存满叶节点
   2^{F-1}≥leafNode, leafNode=leafNode
   2^{F-1}<leafNode， leafNode=2^{F-1}.
   最终，leafNode=3, midNode=3;
   leftNodenum=6, rightNodenum=3.
4. 经上式得根结点元素下标为 6.再将 0-5，7-9 的元素分别执行 3 的操作，并记录根结点位置。
5. 得到一棵树



6. 得到层次遍历序列 6 3 8 1 5 7 9 0 2 4

数据结构分析

1) 数据对象 本题处理的数据为整型(int)数组。

数据关系 本题处理的数据为整型有限数据，题目要求使用树形结构，并且要求顺序存储，并将其构建为完全二叉树。故数据之间关系为链式关系，根元素对应 0-2 个子元素。

2) 基本操作

- **【功能描述】**求解层数

  【名字】 getFloor

  【输入】 整数 n,代表元素的个数

  【输出】 返回一个整数作为层数

- **【功能描述】**求解左子树结点数

  【名字】 leftNodeCount

  【输入】 整数 n,代表元素个数

  【输出】 返回一个整数，代表左子树结点数

- **【功能描述】**求解右子树结点数

  【名字】 rightNodeCount

  【输入】 整数 n,代表元素个数

  【输出】 返回一个整数，代表右子树结点数

- **【功能描述】**根据序列创建树

  【名字】 build_CBST

  【输入】 两个整数一个数组

  【输出】 一个整数，即线性表长度

- **【功能描述】**层次遍历并输出

  【名字】 levelorder

  【输入】 一个结点类类型指针，用来表示开始的位置

  【输出】 打印出层次序列。

3) 物理实现

```
int getFloor(int num)
{
    int count=0;
    while (num>1) {
        num/=2;
        count++;
    }
    return count;
}//返回二叉树的层数(深度，最长路径)
int leftNodeCount(int num)
{
    int leftnode=0, floor=0, result=0;
    floor=getFloor(num+1);
    leftnode=pow(2, floor);
    int temp=num-leftnode+1;
    leftnode=pow(2,floor-1);
    temp=(temp<leftnode)?temp:leftnode;
    leftnode=pow(2, floor-1);
    result=leftnode-1+temp;
    return result;
}//返回需要插入的左子树的节点个数
int rightNodeCount(int num)
{
    return num-1-leftNodeCount(num);
}//返回需要插入的左子树的节点个数
node<E>* build_CBST(int left, int right, E a[])
{
    int leftcount;
    int count=(right+1)-left;
    if (count<=0) {
        return NULL;
    }
    leftcount=leftNodeCount(count);
    node<E>* it;
    it=new realisemybitreenode<E>;
    it->setelement((a[left+leftcount]));
    it->setleft(build_CBST(left, left+leftcount-1, a));
    it->setright(build_CBST(left+leftcount+1, right, a));
    return it;
} //建树
```

```
void levelorder(node<E>* it)
{
    queue<node<E>*> q;
    if(it!=NULL)
    {
        q.push(it);
    }
    node<E>* b;
    while(!q.empty())
    {
        b=q.front();
        cout<<b->element<<' ';
        q.pop();
        if(b->left())
        {
            q.push(b->left());
        }
        if(b->right())
        {
            q.push(b->right());
        }
    }
}
```

二. 算法分析

- **算法思想**：先排序，然后根据元素个数，找出对应的根结点元素位置，从此处开始建树。根结点左右子树对应的的数组看作一棵新的满足 BST 和 CBT 的树，再递归建立左右子树。然后进行层次遍历

- **关键功能的算法步骤：**

1) 排序

```
sort(a, a+n);
```

2) 求左右子节点数

```
int leafNode=num-pow(2, getFloor(num))+1;
int midNode=pow(2, getFloor(num)-1); //getFloor():求根结点所
                                        在位置
leafNode=midNode>leafNode?leafNode:midNode;
leftNodeCount=leafNode+midNode-1;
rightNodeCount=num-1-leftNodeCount(num); //求左右子节点数
```

3) 建树函数

```
node<E>* build_CBST(int left, int right,E a[])
{
int leftcount, LeftRoot, RightRoot;
int NowCount=(right+1)-left;
if (NowCount==0) {
    return NULL;
}
leftcount=leftNodeCount(NowCount);
node<E>* it=new node<E> (a[left+leftcount]);
it->setleft(build_CBST(left, left+leftcount-1, a));
it->setright(build_CBST(left+leftcount+1, right, a));
    return it;
} //build_CBST 函数
```

- **性能分析**

【空间复杂度】每输入一个值便进行一次插入操作，本题中仅开辟一个数组的空间，得空间复杂度为 $\Theta(n)$。

【时间复杂度】先循环输入一个值，时间开销为 C_1n，再进行排序,开销为 $C_2n\log n$，然后递归建树，每次要进行一次计算左右子树结点个数并且执行建立左右子树时间开销为 $C_3n\log n$，综上时间复杂度为 $O(n\log n)$。