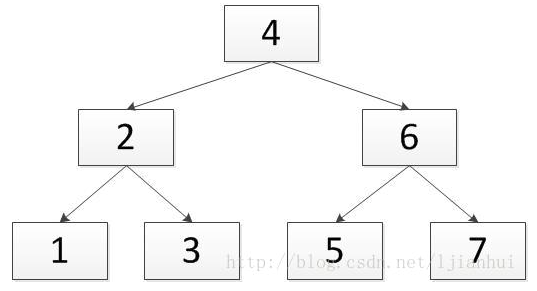
一、问题描述

输入一棵二叉搜索树，现在要将该二叉搜索树转换成一个排序的双向链表。而且在转换的过程中，不能创建任何新的结点，只能调整树中的结点指针的指向来实现。

二、实现思路

在二叉搜索树中，每个结点都有两个分别指向其左、右子树的指针，左子树结点的值总是小于父结点的值，右子树结点的值总是大于父结点的值。而在双向链表中，每个结点也有两个指针，它们分别指向前一个结点和后一个结点。所以这两种数据结构的结点是一致，二叉搜索树之所以为二叉搜索树，双向链表之所以为双向链表，只是因为两个指针的指向不同而已，通过改变其指针的指向来实现是完全可能的。

例如如下的二叉搜索树，



若采用中序遍历，其遍历顺序为1-2-3-4-5-6-7，通过适当的指针变换操作，可变成的双向有序链表如下：



从上图，我们可以看出，为了减少指针的变换次数，并让操作更加简单，在转换成排序双向链表时，

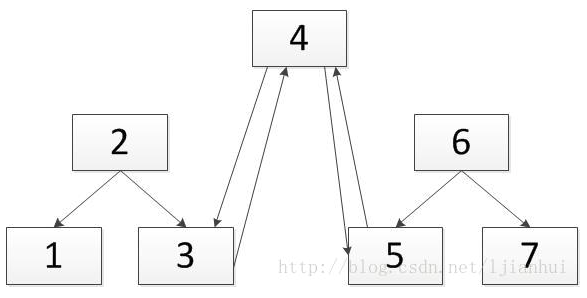
原先指向左子结点的指针调整为链表中指向前一个结点的指针，

原先指向右子结点的指针调整为链表中指向下一个结点的指针。

例如对于上面的值为2的指点，调整后，它的前一个结点为1，后一个结点为3，而结点2的左子结点本来就为1，右子结点本来就为3.

对于树的操作，通常是在遍历树的各个结点的过程中，通过对结点实施某些操作来完成的，这个算法也不例外。由于要求转换后的双向链表也是有序的，而我们从上面也可以看到，当我们以中序遍历二叉搜索树时，其遍历的结点就是有序的，所以在这里我位采用的遍历顺序应该是中序。

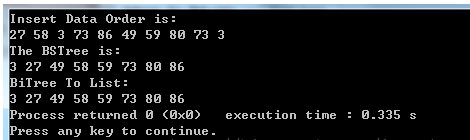
那么我们应该如何调整指针，让二叉搜索树变成一个双向有序链表呢？当遍历到根结点时，我们可以把树看成三个部分：根结点，根的左子树和根的右子树。如上图的二叉排序树，就分成了根结点4、以结点2为根的左子对和以结点6为根的右子树。从变换的链表中我们可以看到，应当把结点4的left指针指向结点3，把结点3的right指针指向结点4，而由于我们采用的是中序遍历，所以当我们遍历到结点4时，结点4的左子树已经转化为一个有序的双向链表，而结点3是这个已经转化的双向链表的尾结点，所以我们应该用一个变量last\_node来保存最后一个结点的指针，以便在与根结点连接时使用。然后把这个变量last\_node的值更新为指向根结点4。对于结点4的右子树，采取相似的操作。至于具体的实现，我们只需要对所有的子树递归地执行上述操作即可。其操作过程如下：



三、实现代码

1. #include <iostream>
2. #include <stdlib.h>
3. #include <time.h>
5. using std::cout;
6. using std::cin;
7. using std::endl;
8. struct BSNode
9. {
10. //定义二叉搜索树的结点结构
11. BSNode \*left;
12. BSNode \*right;
13. int data;
14. };
15. //定义各种用到数据类型
16. typedef BSNode\* BSTree;
17. typedef BSNode\* DList;
18. typedef BSNode DLNode;
19. //往二叉搜索树tree中插入值为data的结点
20. BSTree InsertNode(BSTree tree, int data);
21. //把二叉搜索树tree转化成双向链表，返回头结点
22. DList BSTreeToList(BSTree tree);
23. //遍历二叉搜索树tree的各个结点,并进行指针调整
24. void ConvertNode(BSTree tree, BSNode \*\*last\_node);
25. //查找二叉搜索树tree的最左结点
26. BSNode\* FindLeftmostNode(BSTree tree);
27. //以中序输出二叉搜索树tree
28. void PrintBiTree(BSTree tree);
29. //输出链表
30. void PrintList(DList list);
31. BSTree InsertNode(BSTree tree, int data)
32. {
33. if(tree == NULL)
34. {
35. //找到插入点，则插入
36. tree = new BSNode;
37. tree->left = NULL;
38. tree->right = NULL;
39. tree->data = data;
40. }
41. //插入在其右子树中
42. else if(tree->data < data)
43. tree->right = InsertNode(tree->right, data);
44. //插入在其左子树中
45. else if(tree->data > data)
46. tree->left = InsertNode(tree->left, data);
47. return tree;
48. }
49. DList BSTreeToList(BSTree tree)
50. {
51. if(tree == NULL)
52. return NULL;
53. //找到最左边的结点，即转换后链表的头结点
54. DLNode \*head = FindLeftmostNode(tree);
55. BSNode \*last\_node = NULL;
56. //进行转换
57. ConvertNode(tree, &last\_node);
58. return head;
59. }
60. BSNode\* FindLeftmostNode(BSTree tree)
61. {
62. if(tree == NULL)
63. return NULL;
64. while(tree->left != NULL)
65. tree = tree->left;
66. return tree;
67. }
68. void ConvertNode(BSTree tree, BSNode \*\*last\_node)
69. {
70. if(tree == NULL)
71. return;
72. //对tree的左子树进行转换，last\_node是转换后链表最后一个结点的指针
73. if(tree->left != NULL)
74. ConvertNode(tree->left, last\_node);
75. //调整tree的left指针，指向上一个结点
76. tree->left = \*last\_node;
77. //调整指向最后一个结点，right指向下一个结点
78. if(\*last\_node != NULL)
79. (\*last\_node)->right = tree;
80. //调整指向最后链表一个结点的指针
81. \*last\_node = tree;
82. //对tree的右子树进行转换，last\_node是转换后链表最后一个结点的指针
83. if(tree->right != NULL)
84. ConvertNode(tree->right, last\_node);
85. }
86. void PrintBSTree(BSTree tree)
87. {
88. if(tree == NULL)
89. return;
90. PrintBSTree(tree->left);
91. cout << tree->data << " ";
92. PrintBSTree(tree->right);
93. }
94. void PrintList(DList list)
95. {
96. DLNode \*node = list;
97. while(node != NULL)
98. {
99. cout << node->data << " ";
100. node = node->right;
101. }
102. }
103. int main()
104. {
105. BSTree tree = NULL;
106. srand(time(NULL));
107. cout << "Insert Data Order is:" << endl;
108. for(int i = 0; i < 10; ++i)
109. {
110. //插入随机的10个数，生成二叉排序树
111. int data = rand()%100;
112. cout << data << " ";
113. tree = InsertNode(tree, data);
114. }
115. cout << "\nThe BSTree is: " << endl;
116. PrintBSTree(tree);
117. //进行转换
118. tree = BSTreeToList(tree);
119. cout << "\nBiTree To List: "<< endl;
120. PrintList(tree);
121. return 0;
122. }

运行结果如下：



四、代码分析

由于二叉排序树中不允许有相同的元素，在随机产生的10个数中，有两个是相同的3和73，所以实际插入到二叉排序树中的结点只有8个，然后我们以中序方式遍历输出二叉排序树中的数据，然后输出转换后的链表的数据，发现其顺序是一致的，从而证明算法的正确性。

该算法的实现的核心函数为BSTreeToList，ConvertNode和FindLeftMostNode.

我们可以看到在函数BSTreeToList中，我们有一个变量last\_node用来记录转换了的链表末结点，由于在惯例中，我们会返回链表的第1个结点（从1开始计数）的指针，而last\_node指向的却是末结点，我们可以通过该指针来从尾走到头来获取第一个结点的指针，但是在这里我却没有这样做，因为它需要对每个结点都遍历一次，时间复杂度为O（n）。而是在变换前，找到二叉排序树的最左结点的指针。因为排序二叉树是有序的，最左的结点即为最小的结点，而我们的算法也不会删除或新增结点，也就是说结点的地址是不会改变的，所以最左的结点就是转换后的链表的第1个结点，其时间复杂度为O（logN）。

五、时间复杂度与空间复杂度

该算法首先从根要点一直向左走，找到最左边的结点，其时间复杂度为O(logN)，然后对二叉排序树中的每个结点遍历一次，进行指针变换，其时间复杂度为O(N)，所以总的时间复杂度为O(N)。

至于空间复杂度，由于ConvertNode函数进行递归调用，其函数有两个开参，而函数栈中的函数调用层数不会超过树高，所以其空间复杂度为O(logN)。