

109. Convert Sorted List to Binary Search Tree

109 Convert Sorted List to Binary Search Tree

- Depth-first Search + Linked list

Description

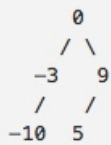
Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1.

Example:

Given the sorted linked list: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST:



1. Thought line

- Height-balanced BST
- Find the middle node in Linked List

```
1 //find the middle node of linked list
2 ListNode* dummyHeadLinkedList = new ListNode(0);
3 dummyHeadLinkedList->next = nodeList;
4 ListNode* ptr0 = dummyHeadLinkedList;
5 ListNode* ptr1 = dummyHeadLinkedList->next; //mid spot
6 ListNode* ptr2 = dummyHeadLinkedList->next->next;
7
8 while(ptr2 != nullptr && ptr2->next != nullptr){
9     ptr1 = ptr1->next;
10    ptr2 = ptr2->next->next;
11    ptr0 = ptr0->next;
12 }
```

- Get left half list and right half list.

```
1 // first half
2 ptr0->next = nullptr;
3 ListNode* firstHalf = dummyHeadLinkedList->next;
4
5 // second half
6 ListNode* secondHalf = ptr1->next;
7 ptr1->next = nullptr;
```

2. Depth-first Search + Linked list

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8  */
9 /**
10 * Definition for a binary tree node.
11 * struct TreeNode {
12 *     int val;
13 *     TreeNode *left;
14 *     TreeNode *right;
15 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
16 * };
17 */
18 void linkedListRoodFind(ListNode* nodeList, TreeNode* nodeTree, string str = "toRightChild"){
19     if (nodeList == nullptr) return;
20
21     //find the middle node of linked list
22     ListNode* dummyHeadLinkList = new ListNode(0);
23     dummyHeadLinkList->next = nodeList;
24     ListNode* ptr0 = dummyHeadLinkList;
25     ListNode* ptr1 = dummyHeadLinkList->next; //mid spot
26     ListNode* ptr2 = dummyHeadLinkList->next->next;
27
28     while(ptr2 != nullptr && ptr2->next != nullptr){
29         ptr1 = ptr1->next;
30         ptr2 = ptr2->next->next;
31         ptr0 = ptr0->next;
32     }
33
34     // first half
35     ptr0->next = nullptr;
36     ListNode* firstHalf = dummyHeadLinkList->next;
37
38     // second half
39     ListNode* secondHalf = ptr1->next;
40     ptr1->next = nullptr;
41
42     if (str == "toRightChild"){
43         nodeTree->right = new TreeNode(ptr1->val);
44         linkedListRoodFind(firstHalf, nodeTree->right, "toLeftChild");
45         linkedListRoodFind(secondHalf, nodeTree->right, "toRightChild");
46     }
47     else if (str == "toLeftChild"){
48         nodeTree->left = new TreeNode(ptr1->val);
49         linkedListRoodFind(firstHalf, nodeTree->left, "toLeftChild");
50         linkedListRoodFind(secondHalf, nodeTree->left, "toRightChild");
51     }
52 }
53
54
55 class Solution {
56 public:
57     TreeNode* sortedListToBST(ListNode* head) {
58         if (head==nullptr) return nullptr;
59         TreeNode* dummyHead = new TreeNode(INT_MIN);
60         linkedListRoodFind(head,dummyHead);
61         return dummyHead->right;
62     }
63 };
```