**39-Combination Sum**

**难度**：Medium

**题目**：

|  |
| --- |
| Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.  The same repeated number may be chosen from C unlimited number of times.  Note:  All numbers (including target) will be positive integers.  Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).  The solution set must not contain duplicate combinations.  For example, given candidate set 2,3,6,7 and target 7,  A solution set is:  [7]  [2, 2, 3] |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  vector<vector<int>> res;  vector<vector<int>> combinationSum(vector<int>& candidates, int target) {  vector<int> tmp;  sort(candidates.begin(),candidates.end());  dfs(candidates,target,0,tmp,0);  return res;  }  void dfs(vector<int>& can,int target,int curSum,vector<int>& tmp,int dep){  if(curSum == target){res.push\_back(tmp);}  for(int i = dep;i<can.size();i++){  if(i ==0 || can[dep]!=can[dep-1]){  if(curSum+can[i]<=target){  tmp.push\_back(can[i]);  dfs(can,target,curSum+can[i],tmp,i);  tmp.pop\_back();  }  }  }  }  }; |

**40-** **Combination Sum II**

**难度**：Medium

**题目**：

|  |
| --- |
| Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.  Each number in C may only be used once in the combination.  Note:  All numbers (including target) will be positive integers.  Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).  The solution set must not contain duplicate combinations.  For example, given candidate set 10,1,2,7,6,1,5 and target 8,  A solution set is:  [1, 7]  [1, 2, 5]  [2, 6]  [1, 1, 6] |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  private:  vector<vector<int> >res;  public:  vector<vector<int> > combinationSum2(vector<int> &candidates, int target) {  sort(candidates.begin(), candidates.end());  vector<int> tmpres;  helper(candidates, 0, target, tmpres, 0);  return res;  }    //从数组candidates[index,...]寻找和为target的组合,times为前一个数字candidates[index-1]重复出现的次数  void helper(vector<int> &candidates, const int index, const int target, vector<int>&tmpres, int times)  {  if(target == 0)  {  res.push\_back(tmpres);  return;  }  for(int i = index; i < candidates.size() && target >= candidates[i]; i++)  {  if(i > 0 && candidates[i] == candidates[i-1])times++;  else times = 1;  if(times == 1 || (tmpres.size() >= times-1 && tmpres[tmpres.size()-times+1] == candidates[i]))  {  tmpres.push\_back(candidates[i]);  helper(candidates, i+1, target - candidates[i], tmpres, times);  tmpres.pop\_back();  }  }  }  }; |

**98-Validate Binary Search Tree**

**难度**：Medium

**题目**：

|  |
| --- |
| Given a binary tree, determine if it is a valid binary search tree (BST).  Assume a BST is defined as follows:  The left subtree of a node contains only nodes with keys less than the node's key.  The right subtree of a node contains only nodes with keys greater than the node's key.  Both the left and right subtrees must also be binary search trees.  confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ. |

**题解**：

|  |
| --- |
| **CPP:**  /\*\*  \* Definition for binary tree  \* struct TreeNode {  \* int val;  \* TreeNode \*left;  \* TreeNode \*right;  \* TreeNode(int x) : val(x), left(NULL), right(NULL) {}  \* };  \*/  class Solution {  public:  bool isValidBST(TreeNode \*root) {  // IMPORTANT: Please reset any member data you declared, as  // the same Solution instance will be reused for each test case.  //注意题目要求是 less than和greater than;  stack<TreeNode\*> S;  TreeNode \*pre = NULL, \*p = root;  while(p || S.empty() == false)  {  while(p)  {  S.push(p);  p = p->left;  }  if(S.empty() == false)  {  p = S.top();  S.pop();  if(pre && p->val <= pre->val)return false;  pre = p;  p = p->right;  }  }  return true;  }  }; |

**99-Recover Binary Search Tree**

**难度**：Hard

**题目**：

|  |
| --- |
| Two elements of a binary search tree (BST) are swapped by mistake.  Recover the tree without changing its structure.  Note:  A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?  confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  void recoverTree(TreeNode \*root) {  // IMPORTANT: Please reset any member data you declared, as  // the same Solution instance will be reused for each test case.  TreeNode \*pre = NULL, \*first = NULL, \*second = NULL;  inorder(root, pre, first, second);  if(first != NULL)  {  if(second == NULL)second = pre;//树{0,1}就可能出现这种情况  int tmp = first->val;  first->val = second->val;  second->val = tmp;  }  }  //pre是中序序列中当前节点的前驱，first、second分别是要找的两个乱序节点  void inorder(TreeNode \*root, TreeNode\* &pre, TreeNode\* &first, TreeNode\* &second)  {  if(root == NULL)return;  if(root->left)inorder(root->left, pre, first, second);  if(pre != NULL)//前驱不为空  {  if(first == NULL && root->val < pre->val)//如果还没找到第一个  first = pre;  else if(first && root->val > first->val)//已经找到第一个  {second = pre; return;}//两个错误位置都找到就退出  }  pre = root;  if(root->right)inorder(root->right, pre, first, second);  }  }; |

**100-Same Tree**

**难度**：Easy

**题目**：

|  |
| --- |
| Given two binary trees, write a function to check if they are equal or not.  Two binary trees are considered equal if they are structurally identical and the nodes have the same value. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  bool isSameTree(TreeNode\* p, TreeNode\* q) {  if(p==NULL && q==NULL) return true;  if(p==NULL || q==NULL) return false;  if(p->val != q->val) return false;  return isSameTree(p->left,q->left)&&isSameTree(p->right,q->right);  }  }; |

**101-Symmetric Tree**

**难度**：Easy

**题目**：

|  |
| --- |
| Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).  For example, this binary tree is symmetric:  1  / \  2 2  / \ / \  3 4 4 3  But the following is not:  1  / \  2 2  \ \  3 3  Note:  Bonus points if you could solve it both recursively and iteratively. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  bool check(TreeNode \*leftNode,TreeNode \*rightNode){  if(leftNode==NULL && rightNode==NULL) return true;  if(leftNode==NULL || rightNode==NULL) return false;  return (leftNode->val == rightNode->val)&&  check(leftNode->left,rightNode->right)&&  check(leftNode->right,rightNode->left);  }  bool isSymmetric(TreeNode\* root) {  if(root == NULL) return true;  return check(root->left,root->right);  }  }; |

**104-Maximum Depth of Binary Tree**

**难度**：Easy

**题目**：

|  |
| --- |
| Given a binary tree, find its maximum depth.  The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  int maxDepth(TreeNode\* root) {  if(root == NULL) return 0;  return 1+max(maxDepth(root->left),maxDepth(root->right));  }  }; |

**105-Construct Binary Tree from Preorder and Inorder Traversal**

**难度**：Medium

**题目**：

|  |
| --- |
| Given preorder and inorder traversal of a tree, construct the binary tree. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {  int n = preorder.size();  if(n==0) return NULL;  TreeNode\* root = build(preorder,0,n-1,inorder,0,n-1);  return root;  }  TreeNode\* build(vector<int>& pre,int a1,int a2, vector<int>& in,int b1,int b2){  TreeNode\* root = new TreeNode(pre[a1]);  int pos = 0;  for(int i=b1;i<=b2;i++){  if(in[i] == pre[a1]){  pos = i;  break;  }  }  int leftLen = pos - b1;  int rightLen = b2 - pos;  if(leftLen>0){  root->left = build(pre,a1+1,a1+leftLen,in,b1,b1+leftLen-1);  }else{  root->left=NULL;  }  if(rightLen>0){  root->right = build(pre,a1+leftLen+1,a1+leftLen+rightLen-1,in,pos+1,pos+rightLen);  }else{  root->right=NULL;  }  return root;  }  }; |

**106-Construct Binary Tree from Inorder and Postorder Traversal**

**难度**：Medium

**题目**：

|  |
| --- |
| Given inorder and postorder traversal of a tree, construct the binary tree. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  TreeNode\* buildTree(vector<int>& inorder, vector<int>& postorder) {  if(inorder.size()==0) return NULL;  TreeNode \* root = createTree(inorder,postorder,0,inorder.size()-1,0,postorder.size()-1);  return root;  }  TreeNode\* createTree(vector<int>& inorder, vector<int>& postorder,int inBeg,int inEnd,int poBeg,int poEnd){  if(inBeg>inEnd) return NULL;  TreeNode\* root = new TreeNode(postorder[poEnd]);  int idx;  for(int i=inBeg;i<=inEnd;i++){  if(inorder[i] == root->val){  idx = i;  break;  }  }  int len = idx - inBeg;  root->left = createTree(inorder,postorder,inBeg,idx-1,poBeg,poBeg+len-1);  root->right = createTree(inorder,postorder,idx+1,inEnd,poBeg+len,poEnd-1);  return root;  }  }; |

**108-Convert Sorted Array to Binary Search Tree**

**难度**：Medium

**题目**：

|  |
| --- |
| Given an array where elements are sorted in ascending order, convert it to a height balanced BST. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  TreeNode \*sortedArrayToBST(vector<int> &num) {  // IMPORTANT: Please reset any member data you declared, as  // the same Solution instance will be reused for each test case.  int len = num.size();  if(len == 0)return NULL;  return sortedArrayToBSTRecur(num, 0, len-1);  }  TreeNode \*sortedArrayToBSTRecur(vector<int> &num, int istart, int iend)  {  if(istart > iend)return NULL;  int middle = (istart+iend)/2;  TreeNode \*res = new TreeNode(num[middle]);  res->left = sortedArrayToBSTRecur(num, istart, middle-1);  res->right = sortedArrayToBSTRecur(num, middle+1, iend);  return res;  }  }; |

**109-Convert Sorted List To Binary Search Tree**

**难度**：Medium

**题目**：

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

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  TreeNode \*sortedListToBST(ListNode \*head) {  if (!head) return NULL;  if (!head->next) return new TreeNode(head->val);  ListNode \*slow = head;//慢指针  ListNode \*fast = head;//快指针  ListNode \*last = slow;//跟踪慢指针，最后成为中间点前面的那个点  while (fast->next && fast->next->next) {  last = slow;  slow = slow->next;  fast = fast->next->next;  }  fast = slow->next;//中间点后面的那个点  //注意这一步的顺序，当链表仅有两个元素时，不能先执行下一步  last->next = NULL;//使下次递归时，左侧以head为起点的链截止在上次递归是的根部  TreeNode \*cur = new TreeNode(slow->val);  //递归  if (head != slow) cur->left = sortedListToBST(head);  cur->right = sortedListToBST(fast);  return cur;  }  }; |

**110-Balanced Binary Tree**

**难度**：Medium

**题目**：

|  |
| --- |
| Given a binary tree, determine if it is height-balanced.  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. |

**题解**：

|  |
| --- |
| **CPP:**  class Solution {  public:  bool isBalanced(TreeNode \*root) {  // IMPORTANT: Please reset any member data you declared, as  // the same Solution instance will be reused for each test case.  if(root == NULL)return true;  if(height(root) == -1)return false;  else return true;  }  //若root是平衡树，那么返回树的高度，否则返回-1  int height(TreeNode \*root)  {  //if(root==NULL) return 0;此句可有可无，因为只有不为空才会进入height函数  if(root->left == NULL && root->right == NULL)return 1;//最小情形返回值  //中间递归过程  int leftHeight = 0, rightHeight = 0;  if(root->left)  leftHeight = height(root->left);  if(leftHeight == -1)return -1;  if(root->right)  rightHeight = height(root->right);  if(rightHeight == -1)return -1;  if(abs(leftHeight-rightHeight) > 1)return -1;  return 1+max(leftHeight, rightHeight);//中间过程返回值  }  }; |