框架刷通二叉树(二)

构造最大二叉树

> 给定一个数组，将数组的最大值作为根，最大值左侧为左节点，右侧为右节点，构造一颗二叉树。

```
class Solution {
public:
  TreeNode *constructMaximumBinaryTree(std::vector<int> &nums) {
    int len = nums.size();
    if (0 >= len) {
      return nullptr;
    }

    return constructMaximumBinaryTree(nums, 0, len - 1, len);
  }

private:
  TreeNode *constructMaximumBinaryTree(std::vector<int> &nums,
                                       int               lo,
                                       int               hi,
                                       int               len) {
    if (0 > lo || len <= hi || lo > hi) {
      return nullptr;
    }

    // 查找最大值
    int m     = nums[lo];
    int index = lo;
    for (int i = lo; i <= hi; i++) {
      if (m < nums[i]) {
        m     = nums[i];
        index = i;
      }
    }

    TreeNode *root = new TreeNode(m);
    root->left     = constructMaximumBinaryTree(nums, lo, index - 1, len);
    root->right    = constructMaximumBinaryTree(nums, index + 1, hi, len);

    return root;
  }
};
```

先序与中序序列构造二叉树

```
class Solution {
public:
  TreeNode *buildTree(std::vector<int> &preorder, std::vector<int>
&inorder) {
    int lenp = preorder.size();
    int leni = inorder.size();
    if (lenp <= 0 || leni <= 0) {
      return nullptr;
    }
    return buildTree(preorder,
                     0,
                     preorder.size() - 1,
                     inorder,
                     0,
                     inorder.size() - 1);
  }

private:
  TreeNode *buildTree(std::vector<int> &preorder,
                      int               plo,
                      int               phi,
                      std::vector<int> &inorder,
                      int               ilo,
                      int               ihi) {
    if (plo > phi) {
      return nullptr;
    }

    TreeNode *root = new TreeNode(preorder[plo]);
    // 查找preorder[plo]在inorder中的位置
    int index = ilo;
    for (int i = ilo; i <= ihi; i++) {
      if (inorder[i] == preorder[plo]) {
        index = i;
        break;
      }
    }

    // 数组左侧长度
    int leftlen = index - ilo;
    root->left =
        buildTree(preorder, plo + 1, plo + leftlen, inorder, ilo, index -
1);
    root->right =
        buildTree(preorder, plo + leftlen + 1, phi, inorder, index + 1,
ihi);

    return root;
  }
};
```

通过后序和中序遍历结果构造二叉树

```
class Solution {
public:
  TreeNode *buildTree(std::vector<int> &inorder, std::vector<int>
&postorder) {
    int leni = inorder.size();
    int lenp = postorder.size();

    if (leni <= 0 || lenp <= 0 || leni != lenp) {
      return nullptr;
    }

    return buildTree(inorder, 0, leni - 1, postorder, 0, lenp - 1);
  }

private:
  TreeNode *buildTree(std::vector<int> &inorder,
                      int               lo1,
                      int               hi1,
                      std::vector<int> &postorder,
                      int               lo2,
                      int               hi2) {
    if (lo1 > hi1) {
      return nullptr;
    }

    TreeNode *root  = new TreeNode(postorder[hi2]);
    int       index = hi2;
    for (int i = lo1; i <= hi1; i++) {
      if (inorder[i] == postorder[hi2]) {
        index = i;
        break;
      }
    }

    // 左子树长度
    int leftlen = index - lo1;
    root->left =
        buildTree(inorder, lo1, index - 1, postorder, lo2, lo2 + leftlen -
1);
    root->right =
        buildTree(inorder, index + 1, hi1, postorder, lo2 + leftlen, hi2 -
1);

    return root;
  }
};
```