数据结构的存储方式

数据结构的存储只有两种方式：数组(顺序存储)和链表(链式存储)。

数据结构的基本操作

数据遍历框架：

1. 数组遍历框架,典型的迭代遍历结构：

```
void traverse(std::vector<int> &vec) {
  for(int i = 0;i < vec.size(); ++i){
    // 迭代访问vec[i]
  }
}
```

2. 链表遍历框架，兼具迭代和递归遍历结构：

```
// 基本的单链表节点
struct listNode{
  int val;
  listNode *next;
};

void traverse(listNode *head){
  for(listNode *ln = head; ln != nullptr; ln = ln->next){
    // 迭代访问
  }
}

void traverse(listNode *head){
  // 递归访问
  traverse(head->next);
}
```

3. 二叉树遍历,典型非线性递归遍历

```
// 基本二叉树节点
struct TreeNode{
  int val;
  TreeNode *left, *right;
};

void traverse(TreeNode *root){
  traverse(root->left);
  traverse(root->right);
}
```

4. N叉树遍历

```
// 基本N叉树节点
struct NTreeNode{
  int val;
  TreeNode*[] children;
};

void traverse(NTreeNode *root){
  for(auto child : children){
    traverse(child);
  }
}
```

N叉树遍历可以扩展为图遍历。

算法刷题指南

1. 从二叉树开始。

如：124. 二叉树中最大路径和

```
class Solution {
public:
  int maxPathSum(TreeNode *root) {
    _maxPathSum(root);
    return ans;
  }

private:
  int _maxPathSum(TreeNode *root) {
    if (!root) {
      return 0;
    }

    int left  = std::max(0, _maxPathSum(root->left));
    int right = std::max(0, _maxPathSum(root->right));
    ans       = std::max(ans, left + right + root->val);

    return std::max(left, right) + root->val;
  }

  int ans = INT_MIN;
};
```

利用后序遍历的思想，对一颗二叉树的左右子树分别进行递归，如果子树的和小于0，则不遍历该子树，(路径不一定从根节点开始).