1、所有的数据结构都是基于数组（顺序存储方式）和链表（链式存储方式）两种

2、在数据结构上的基本操作无非就是遍历和访问两种

遍历又分为线性(**for**)和非线性(**递归实现**)

2.1链表具有顺序和链式的遍历方式

```
顺序方式
while(p->next!=null){

}
链式(递归)方式
void tranverse(ListNode head){
 //head.val-前序遍历
 tranverse(head->next);
 //head.val-后序遍历
}
如果采用前序遍历方法则为正序打印，如果采用后序遍历则为反序打印
```

对于遍历方式的泛化：

基本二叉树遍历->基本N叉树遍历->图的遍历（由于图中可能出现环所以可以使用一个数组visited来表标记）

```
基本N叉树的遍历：
void tranverse(ListNode head){
 for(ListNode child:head.children){
 tranverse(child);
 }
}
```

二叉树的解题框架(very important)！！！

```
void tranverse(TreeNode root){
 //前序遍历
 tranverse(root->left);
 //中序遍历
 tranverse(root->right);
 //后序遍历
}
```

知识点：

1、我们可以通过（前序遍历结果/后续遍历结果）+中序遍历结果来建立一个二叉树

例题1已知前序遍历和后续遍历的数组，重新构建二叉树

```
TreeNode buildTree(int[] preorder,int preStart,int preEnd,
```

```
int[] inorder,int inStart,int inEnd,Map<Integer,Integer> inMap){
 if(preStart>preEnd||inStart>inEnd) return null;

 TreeNode root=new ThreeNode(preorder[preStart]));
 int inRoot=inMap(root.val);
 int numleft=inRoot-inStart;//表示inRoot的左子树中节点个数
 //构建左子树
 ThreeNode left=buildTree(preorder,preStart+1,preStart+numleft,
  inorder,inStart,inRoot-1,inMap);
  //构建右子树
  ThreeNode right=buildTree(preorder,preStart+numleft+1,preEnd,
  inorder,inRoot+1,inEnd,inMap);
  root->left=left;
  root->right=right;
  return root;
}
本质就是一个前序遍历加一个二分搜索
```

只要涉及递归的问题，基本都是树的问题