# 1. input:123 output:321

## C++

class Solution {

public:

int reverse(int x) {

int b=0;

while (x != 0)

{

b = b \* 10;

b = b + x%10;

x = x/10;

}

return b;

}

};

## Java

class Solution {

public int reverse(int x) {

int b=0;

while(x!=0)

{

b=b\*10;

b=b+x%10;

x=x/10;

}

return b;

}

}

# 2. Input: 1->2->4, 1->3->4 Output: 1->1->2->3->4->4

## Java

class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode root = new ListNode(-1);

ListNode dummyRoot = root;

while(l1!= null && l2!=null) {

if(l1.val>l2.val){

root.next = l2;

l2 = l2.next;

} else {

root.next = l1;

l1 = l1.next;

}

root = root.next;

}

if(l1==null) {

root.next = l2;

}

if(l2==null) {

root.next = l1;

}

return dummyRoot.next;

}

}

# 3.TWO SUM

Example:

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

## Java

class Solution {

public int[] twoSum(int[] nums, int target) {

int i ;

int j ;

for(i=0;i<nums.length;i++)

{

for(j=i+1;j<nums.length;j++)

{

if(nums[i]+nums[j]==target)

return new int[]{i,j};

}

}

throw new IllegalArgumentException("No two sum solution");

}

}

# 4. Search Insert Position

**Example 1:**

**Input:** [1,3,5,6], 5

**Output:** 2

**Example 2:**

**Input:** [1,3,5,6], 2

**Output:** 1

## Java

class Solution {

public int searchInsert(int[] nums, int target) {

for(int i=0;i<nums.length;i++){

if(nums[i]==target)

return i;

}

for(int mid=0;mid<nums.length;mid++)

{

if(nums[mid]>target)

return mid;

}

return nums.length;

}

}

# 5.Given a binary tree, find its maximum depth.

**Example:**

Given binary tree [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

return its depth = 3.

## Java

public int maxDepth(TreeNode root) {

if(root==null){

return 0;

}

return 1+Math.max(maxDepth(root.left),maxDepth(root.right));

}

# 6. Single Number

Given a **non-empty** array of integers, every element appears twice except for one. Find that single one.

**Example 1:**

**Input:** [2,2,1]

**Output:** 1

**Example 2:**

**Input:** [4,1,2,1,2]

**Output:** 4

## Java

class Solution {

public int singleNumber(int[] nums) {

int result=0;

for(int num : nums) {

result=result^num;

}

return result;

}

}

(注：^ is the XOR operator. Let's assume we have an integer 'a'.  
So, a^a = 0 and a^0 = a. Hence the number that occurs twice, will make the result 0 and the number that occurs once the result will equal the number itself using the XOR(^) operator.)

## Python

### Approach #1 List operation [Time Limit Exceeded]

class Solution(object):

def singleNumber(self, nums):

no\_duplicate\_list = []

for i in nums:

if i not in no\_duplicate\_list:

no\_duplicate\_list.append(i)

else:

no\_duplicate\_list.remove(i)

return no\_duplicate\_list.pop()

### Approach #2 Hash Table [Accepted]

class Solution(object):

def singleNumber(self, nums):

hash\_table = {}

for i in nums:

try:

hash\_table.pop(i)

except:

hash\_table[i] = 1

return hash\_table.popitem()[0]

# 7.Linked List Cycle

Given a linked list, determine if it has a cycle in it.

## Java

### Approach #1 (Hash Table) [Accepted]

**Intuition**

To detect if a list is cyclic, we can check whether a node had been visited before. A natural way is to use a hash table.

**Algorithm**

We go through each node one by one and record each node's reference (or memory address) in a hash table. If the current node is null, we have reached the end of the list and it must not be cyclic. If current node’s reference is in the hash table, then return true.

public boolean hasCycle(ListNode head) {

Set<ListNode> nodesSeen = new HashSet<>();

while (head != null) {

if (nodesSeen.contains(head)) {

return true;

} else {

nodesSeen.add(head);

}

head = head.next;

}

return false;

}

### Approach #2 (Two Pointers) [Accepted]

**Intuition**

Imagine two runners running on a track at different speed. What happens when the track is actually a circle?

**Algorithm**

The space complexity can be reduced to O(1)O(1)O(1) by considering two pointers at **different speed** - a slow pointer and a fast pointer. The slow pointer moves one step at a time while the fast pointer moves two steps at a time.

If there is no cycle in the list, the fast pointer will eventually reach the end and we can return false in this case.

Now consider a cyclic list and imagine the slow and fast pointers are two runners racing around a circle track. The fast runner will eventually meet the slow runner. Why? Consider this case (we name it case A) - The fast runner is just one step behind the slow runner. In the next iteration, they both increment one and two steps respectively and meet each other.

How about other cases? For example, we have not considered cases where the fast runner is two or three steps behind the slow runner yet. This is simple, because in the next or next's next iteration, this case will be reduced to case A mentioned above.

public boolean hasCycle(ListNode head) {

if (head == null || head.next == null) {

return false;

}

ListNode slow = head;

ListNode fast = head.next;

while (slow != fast) {

if (fast == null || fast.next == null) {

return false;

}

slow = slow.next;

fast = fast.next.next;

}

return true;

}

# 8.Linked List Cycle II

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Java

public class Solution {

public ListNode detectCycle(ListNode head) {

Set<ListNode> testnode = new HashSet<>();

while(head!=null){

if(testnode.contains(head)){

return head; }

else{

testnode.add(head); }

head=head.next;

}

return null;

}

}

# 9.输入一个链表，按链表值从尾到头的顺序返回一个ArrayList。

使用栈：Java中提供了Stack栈，可以直接使用。

|  |  |
| --- | --- |
| boolean | [empty](https://blog.csdn.net/qq_27703417/article/details/70946406)()        测试堆栈是否为空。 |
| [E](https://blog.csdn.net/qq_27703417/article/details/70946406) | [peek](https://blog.csdn.net/qq_27703417/article/details/70946406)()         查看堆栈顶部的对象，但不从堆栈中移除它。 |
| [E](https://blog.csdn.net/qq_27703417/article/details/70946406) | [pop](https://blog.csdn.net/qq_27703417/article/details/70946406)()          移除堆栈顶部的对象，并作为此函数的值返回该对象。 |
| [E](https://blog.csdn.net/qq_27703417/article/details/70946406) | [push](https://blog.csdn.net/qq_27703417/article/details/70946406)([E](https://blog.csdn.net/qq_27703417/article/details/70946406) item)      把项压入堆栈顶部。 |
| int | [search](https://blog.csdn.net/qq_27703417/article/details/70946406)([Object](https://blog.csdn.net/qq_27703417/article/details/70946406) o)    返回对象在堆栈中的位置，以 1 为基数。 |

/\*\*

\* public class ListNode {

\* int val;

\* ListNode next = null;

\* ListNode(int val) {

\* this.val = val;

\* }

\* }

\*/

import java.util.Stack;

import java.util.ArrayList;

public class Solution {

public ArrayList<Integer> printListFromTailToHead(ListNode listNode) {

Stack<Integer> stack = new Stack<>();

while (listNode != null) {

stack.push(listNode.val);

listNode = listNode.next;

}

ArrayList<Integer> list = new ArrayList<>();

while (!stack.isEmpty()) {

list.add(stack.pop());

}

return list;

}

}

# 10请实现一个函数，将一个字符串中的空格替换成“%20”。

例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

package com.offer.cn;

//请实现一个函数，将一个字符串中的空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

public class Solution {

public static String replaceSpace(StringBuffer str) {

String str1=str.toString();

char[] charArray = str1.toCharArray();

StringBuilder sBuilder = new StringBuilder();

for (char c : charArray) {

if(c==' ') {

sBuilder.append("%20");

}else {

sBuilder.append(c);

}

}

String string = sBuilder.toString();

return string;

}

public static void main(String[] args) {

System.out.println(replaceSpace(new StringBuffer("We Are Happy.")));

}

}

# 11.重建二叉树

输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

public class Solution {

    public TreeNode reConstructBinaryTree(int [] pre,int [] in) {

        TreeNode root=reConstructBinaryTree(pre,0,pre.length-1,in,0,in.length-1);

        return root;

    }

    //前序遍历{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}

    private TreeNode reConstructBinaryTree(int [] pre,int startPre,int endPre,int [] in,int startIn,int endIn) {

        if(startPre>endPre||startIn>endIn)

            return null;

        TreeNode root=new TreeNode(pre[startPre]);

        for(int i=startIn;i<=endIn;i++)

            if(in[i]==pre[startPre]){

                root.left=reConstructBinaryTree(pre,startPre+1,startPre+i-startIn,in,startIn,i-1);

                root.right=reConstructBinaryTree(pre,i-startIn+startPre+1,endPre,in,i+1,endIn);

                      break;

            }

        return root;

    }

}