**LeetCode 知识点整理**

# Bit  Manipulation

## 求二进制数中1的位数

1. Soulution 1: 依次遍历二进制数中的每一位，统计1的个数。
2. Solution 2: 每次统计一个1的个数，采用x &= x-1 将最末尾的1置为0。
3. Solution 3:很巧妙的方法，首先是将每2位的1的个数放在这2位中，依次类推到4位，8位，16位，32位。
4. 使用gcc编译器内建的\_\_builtin\_popcount(n)函数。

参考资料：<https://en.wikipedia.org/wiki/Hamming_weight>

## ffs(find first set bit),clz(count leading zeros),ctz(count trailing zeros)

1. **比较巧妙的求解**

**使用ctz来求clz。For循环中，通过传导，将最高位的1之后的位全部设置为1。**

**function** clz(x):

**for each** y **in** {1, 2, 4, 8, 16}: x ← x | (x >> y)

**return** 32 − ctz(x + 1)

**类似于上面的方法。**

**function** clz(x):

**for each** y **in** {1, 2, 4, 8, 16}: x ← x | (x >> y)

**return** 32 − pop(x)

**采用二分法。**

**function** ctz (x)

**if** x = 0 **return** 32

n ← 0

**if** (x & 0x0000FFFF) = 0: n ← n + 16, x ← x >> 16

**if** (x & 0x000000FF) = 0: n ← n + 8, x ← x >> 8

**if** (x & 0x0000000F) = 0: n ← n + 4, x ← x >> 4

**if** (x & 0x00000003) = 0: n ← n + 2, x ← x >> 2

**if** (x & 0x00000001) = 0: n ← n + 1

**return** n

**采用二分法**

**function** clz(x)

**if** x = 0 **return** 32

n ← 0

**if** (x & 0xFFFF0000) = 0: n ← n + 16, x ← x << 16

**if** (x & 0xFF000000) = 0: n ← n + 8, x ← x << 8

**if** (x & 0xF0000000) = 0: n ← n + 4, x ← x << 4

**if** (x & 0xC0000000) = 0: n ← n + 2, x ← x << 2

**if** (x & 0x80000000) = 0: n ← n + 1

**return** n

1. **几种方法的转换**

ctz(x) = ffs(x) – 1

lg(x) = w − 1 − clz(x)

ctz(x) = lg(x & (−x))

ffs(x) = w − clz(x & (−x)).

ctz(x) = pop((x & (−x)) − 1)

ffs(x) = pop(x ^ (~(−x)))

参考资料：<https://en.wikipedia.org/wiki/Find_first_set>

## 关于bit Manipulation的一些小知识点

x-1：从高位到低位，将最后一个1之后的全部取反（包括这个1），其余位不变。

x & (x-1):从高位到低位，将最后一个1置0，其余位不变。

-x: 从高位到低位,最后一个1之后（保证这个1）的位不变，其余位取反。

x & (-x): 最后一个1之前的所有位置为0.

-x 和x-1都是以最后一个1位分界线，一边取反，一边不变。

# [Moore’s Voting Algorithm](http://www.cs.utexas.edu/~moore/best-ideas/mjrty/index.html) 和 [Iceberg Query Algorithm](http://blog.csdn.net/pi9nc/article/details/9355293)

参考资料：

<http://blog.csdn.net/pi9nc/article/details/9355293>

C:\Users\killerxwj\Documents\GitHub\Leetcode\karp.pdf

<https://en.wikipedia.org/wiki/Online_algorithm>

具体实现自己还得写代码！！！！！！

# 二叉树

## 线索二叉树

n各结点的二叉链表共有2n个链域，非空链域为n-1个，但其中的空链域却有n+1个，利用这空链域来指向其它节点，如果lchild为NULL，用它来指向前驱节点；如果rchild为空，则用它指向后继节点。

以中序遍历为例，采用递归的方式建立中序遍历的线索树， 和中序遍历的流程基本一样，只不过数据的访问变成了和当前节点相关的线索的建立。

而遍历线索树，空间复杂度变为了1，因为它不需要使用栈来存储，可以直接通过线索回到后继节点。

参考资料：<http://blog.chinaunix.net/uid-26548237-id-3476920.html>

## 二叉树的遍历

### 中序遍历

一般有三种方法可以解决：

1. 递归，这个方法相对简单。
2. 利用栈来辅助，具体思路找一个节点的后继节点，初始节点为NULL，在搜索过程中需要将经过的节点都压栈，为了后面回到该节点。具体代码可以见Github下Leetcode中的Binary Tree Inorder Traversal
3. 利用Morris Traversal,空间复杂度位O(1)，时间复杂度为O(n)。因为不能用栈作为辅助空间来保存付节点的信息，重点在于当访问到子节点的时候如何重新回到父节点。

Morris遍历方法用了线索二叉树，这个方法不需要为每个节点额外分配指针指向其前驱和后继结点，而是利用叶子节点中的右空指针指向中序遍历下的后继节点就可以了。

算法具体分情况如下：  
1. 如果当前结点的左孩子为空，则输出当前结点并将其当前节点赋值为右孩子。  
2. 如果当前节点的左孩子不为空，则寻找当前节点在中序遍历下的前驱节点（也就是当前结点左子树的最右孩子）。接下来分两种情况：  
 a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点（**做线索使得稍后可以重新返回父结点**）。然后将当前节点更新为当前节点的左孩子。  
 b) 如果前驱节点的右孩子为当前节点，表明左子树已经访问完（**英文通过线索回到了当前节点，这样就说明了左子树访问完成**），可以访问当前节点。将它的右孩子重新设为空（恢复树的结构）。输出当前节点。当前节点更新为当前节点的右孩子。

整个过程中每条边最多只走2次，一次是为了定位到某个节点，另一次是为了寻找上面某个节点的前驱节点，而n个结点的二叉树中有n-1条边，所以时间复杂度是O(2\*n)=O(n)，仍然是一个线性算法。空间复杂度的话我们分析过了，只是两个辅助指针，所以是O(1)。

参考资料：<http://www.cnblogs.com/AnnieKim/archive/2013/06/15/morristraversal.html>

### 前序遍历

1. 递归
2. 利用栈来辅助，使用栈辅助有两种方法：（1） 第一种和中序遍历类似，唯一的不同在于何时将值打印出来，就一行代码的区别，中序遍历是在出栈时打印，而前序遍历是在压栈时打印。（2） 初始时，将当前节点入栈，出栈时就直接打印该节点，然后将当前节点的右孩子和左孩子分别入栈。
3. Morris Traversal，和中序遍历算法一样，唯一的区别就是打印的时机，前序遍历的打印时机在当前节点确定后。而中序遍历的打印时间为左子树访问完成之后。

参考资料：<http://www.cnblogs.com/AnnieKim/archive/2013/06/15/morristraversal.html>

### 后续遍历

1. 递归
2. 直接使用栈来辅助。后序遍历结点的访问顺序是：左儿子 -> 右儿子 -> 自己。那么一个结点需要两种情况下才能够输出：第一，它已经是叶子结点；第二，它不是叶子结点，但是它的儿子已经输出过。那么基于此我们只需要记录一下当前输出的结点即可。对于一个新的结点，如果它不是叶子结点，儿子也没有访问，那么就需要将它的右儿子，左儿子压入。
3. 通过前序遍历来转换。遍历顺序为LRC,(LRC)T = CTRTLT,这样就转化为了前序遍历。所以后续遍历是前序遍历CRL的转置。而前序遍历一方面可以使用栈来辅助，另一方面，也可以使用线索。

参考资料：<http://www.cnblogs.com/changchengxiao/p/3416402.html>

# 二分搜索的一些小技巧

1. Mid的计算。

mid 一般选取为 (l+r)/2，或(l+r+1)/2.何时取为(l+r)/2? (l+r)/2是取在左边的中点的。如果判断条件中有l = mid，当r-l == 1时，l的值将不会改变，范围无法缩小，此时会进入死循环，所以在选取mid时要取在右边的中点，即mid=(l+r+1)/2.

1. 如果没有等于val值的，就选取第一个大于val的。

如果有等于val的就返回了。如果没有，就会返回一个比它大的，最后会有l==r的情况，在这个情况下，不知道当前num[l]是大于它的；但是接下来的情况是l-r == 1.这个时候，l在右边，r在左边。必然会有l所对应的就是大于val的索引。

所以有while(l <= r) {…} return l;

同理，对于选取离他最近的小于val的就返回r。

# 额外知识点

## 牛顿迭代法求平方根

double sqr(double n)

{

double k = 1.0; //可任取

while(fabs(k\*k-n)>1e-9) //double不能用==比较

{

k=(k+n/k)/2;

}

return k;

}