18-19DSA期末考答案

本答案由群笨猫自制，由于做这份卷的时候被打断了n多次，所以正确率堪忧喵。如果做出来跟笨猫的不一样，很可能是笨猫有问题，记得在群里说一下。

选择题

选择题一般是15-20分，前面的简单题1分，后面的困难题2分。

1.C　注意顺序存储的线性表，在插入时还得挪位置，所以是O(n)
2.D　由森林转换二叉树时，第一棵树构成二叉树根及左子树，其余树构成右子树。所以二叉树根结点右子树上结点个数是第二、三棵树结点个数之和，即 M2 + M3
3.C　1+3+9+27+81+243=364，所以是C
4.A　堆是完全二叉树，大顶堆根结点最大，用顺序表存储时满足此特性，A 正确；堆排序时间复杂度平均为 O (nlogn)，最坏也是 O (nlogn)，空间复杂度为 O (1)，B、C、D 错误
5.D　完全有向图，所以是n(n-1)，如果是无向图则要/2
6.C　应该是以该事件为头的，这个事件和活动的讲法，真tm傻逼啊，而且一般不是用点算弧吗？太反人类了吧
7.B　无向图的邻接矩阵是对称的，AOV是顶点表示活动图（有向无环），AOE是边表示活动图（有向无环），所以是B
8.B　用邻接表表示图，拓扑排序需遍历所有顶点和边，时间复杂度为 O (n + e)，n 是顶点数，e 是边数。
9.C　8+7+6+...+2+1=36,36+1=37，我感觉是得加个'\0'，然后变成37的，同时要注意一下，software这里每个字母都不重复，所以可以直接算，但如果是genshin这种，有两个n，那就要考虑'n'的子串重复了。
10.C　归并，还是那句话，好好看表
11.C　除了哈希表，其它都跟表长度有关
12.A　前序为ABCDEF，中序为CBAEDF，前序第一个是A，说明根节点是A，所以后序最后一个是A；看中序为CBAEDF，说明CB在左子树，EDF在右子树，所以后序开头是CB，再看前序是DEF，中序是EDF，说明D是子树的根节点，EF分别是左右孩子，所以后序为CBEFDA。(总体来说，就是前序/后序找根，中序看划分，然后不断分解这个过程)
13.A　17-18一猫一样的题

填空题

填空题一般是15-20分，前面的简单题每空1分，后面的困难题每空2分。
1.由于无头结点，所以链栈为空的判断条件是S==NULL
2.直接看表，O(n)，详细一点可以写O(d(n+r))
3.邻接矩阵主要还是看结点数量，所以是$O(n^2)$
4.普通二叉链表有n+1个空指针，自己举几个例子就行；中序线索二叉链表应该是只有2个空指针来着？中序第一个的左孩子和最后一个的右孩子为空
5.看while(i*i<=n)就知道是$O(\sqrt{(n)})$了
6.4x2+3x1+1x5=2+1+5+n-1,n=9，或者你自己画个图
7.1+2+...+1024=2047,2018介于2047和1023之间，最后一层是1024+2018-2047=995个节点，所以右子树一共有1+2+...+256+995-512=994个节点
8.可以考虑一个9顶点完全强连通图+一个孤立点，此时有9x8=72条边，这时候再让孤立点指向其它顶点，这样就有72+9=81条边了。
9.先序是中左右，中序是左中右，先序和中序相同，说明没有左子树，只有5个节点，一路往右
10.第一次查91，第二次234，第三次123，一共3次比较

应用题

应用题一般是40-50分，每题分值不等
1.
(1)n(1*4*x+2*x+3)=180,n(2*4*x+3*x+4)=324，所以6x+3和11x+4的比是5/9，54x+27=55x+20,x=7
(2)45n=180,n=4
(3)按照列优先，那就是1000+4(2+3x5+4x5x4)=1388
2.这题就纯作业题了
线性探测法就是1234这样试，没什么好说的

	0	1	2	3	4	5	6	7	8	9	10
HT	33	11	35	13	01	48			19	20	30

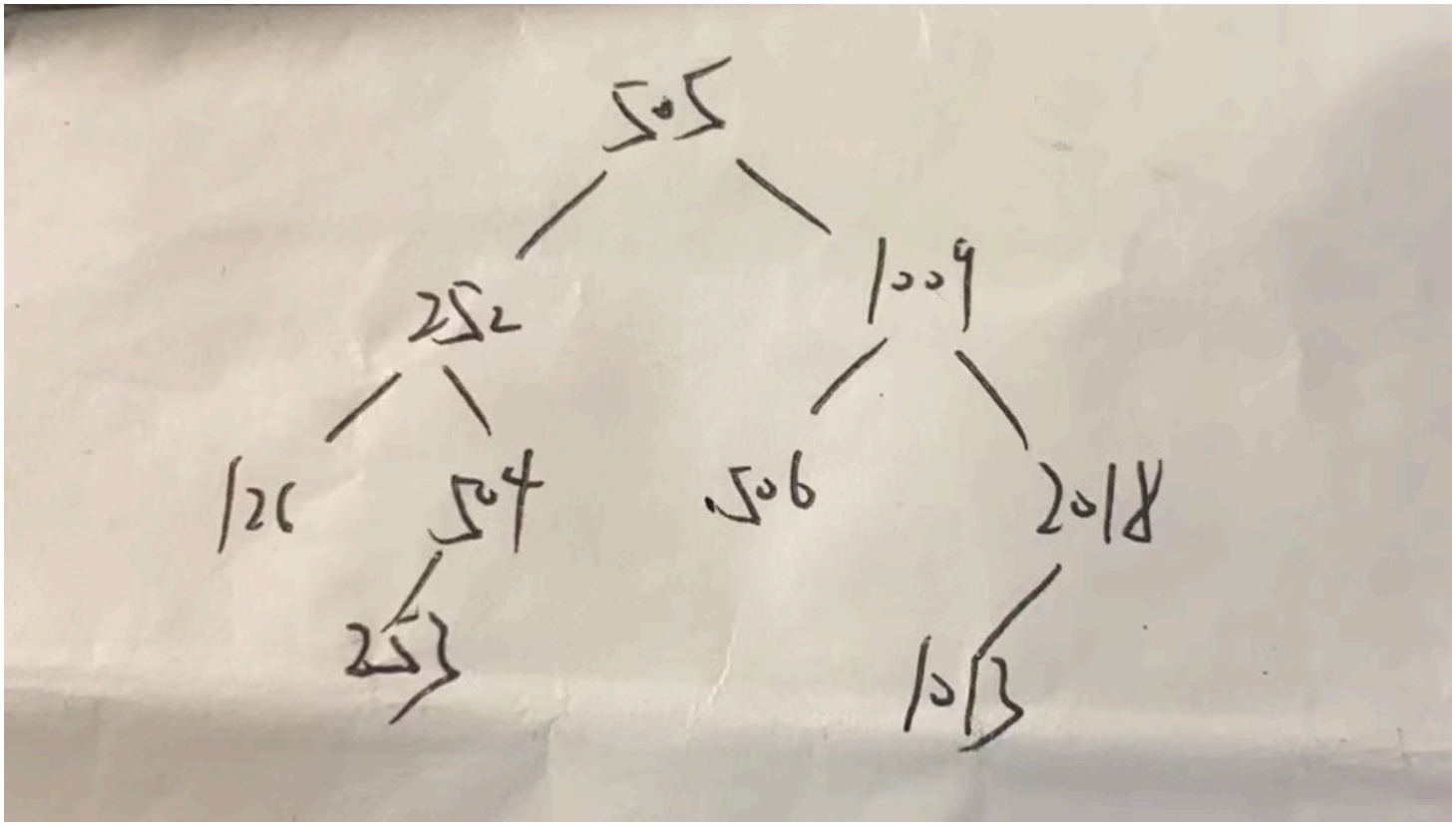
删掉35之后，ASL=(1+2+2+4+2+1+1+3)/8=2

3.

(1)往上追溯吧，反正一路/2然后向下取整就行，2018-1009-504-252-126，1013-506-253-126，所以是126
(2)直接x2的是左孩子，x2后还要+1的是右孩子，所以中序遍历是504，1009，2018，252，505，

126，506，1013，253

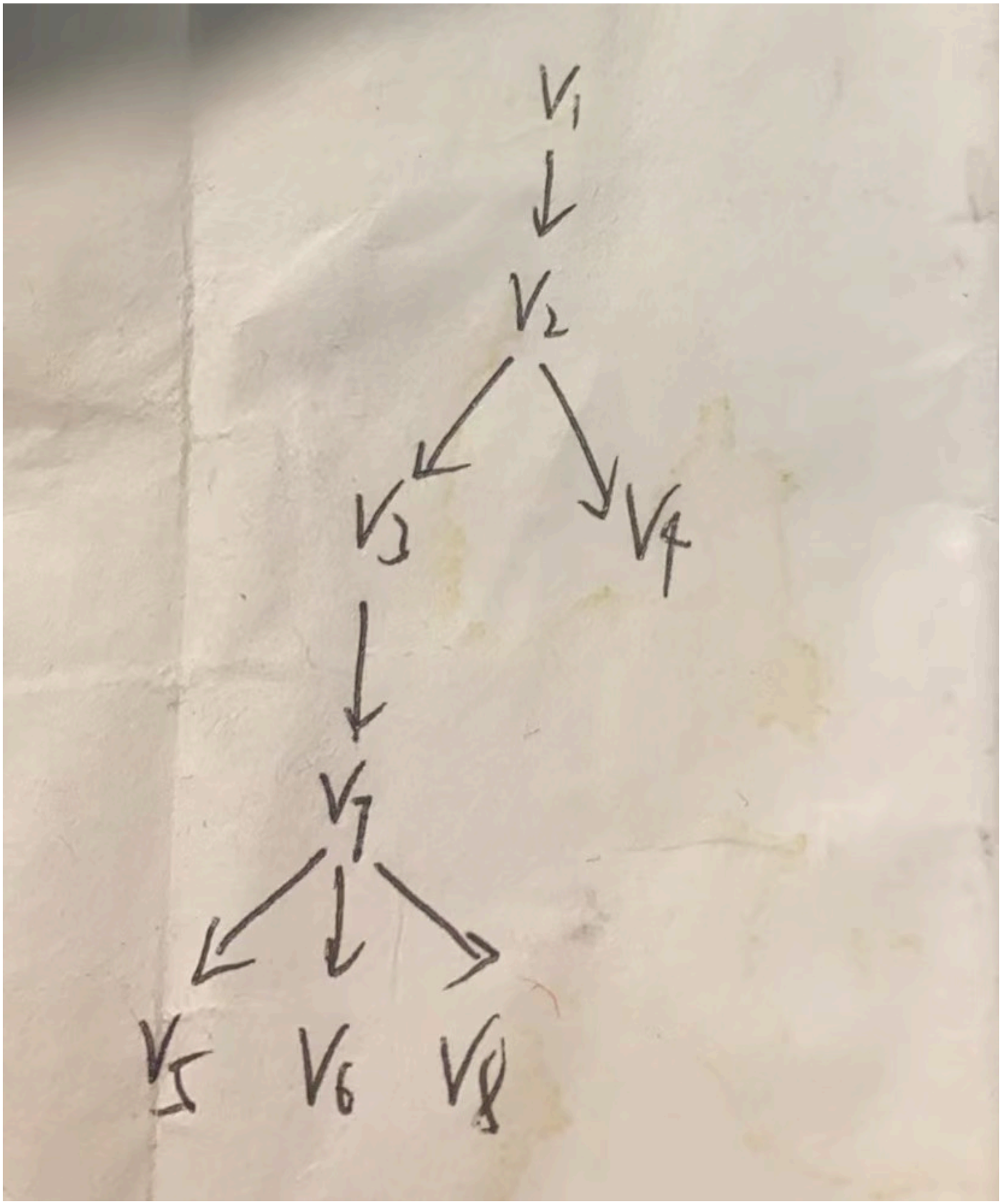
(3)按照上图的序列来逐个插入，二叉查找树要求左<中<右，最后的AVL二叉查找树如下：



说实话，我这题的思路有点怪的，反正有7个节点的AVL树应该是完全二叉树，再加上要满足二分查找，所以是固定的，最后插个1013和253就可以了。我是真烦AVL树，当年数据结构及其算法期末考唯一不太会的那题就是一道选择，问你图里的树要用什么旋转才能变成AVL树，选项是什么LL、LR、RL、RR的排列组合，我至今没想明白那题怎么弄。

4.强行算就是了，但是真的好恶心

(1)DFS的时候暂时不用考虑拓扑关系:v1-v2-v3-v7-v5-v6-v8-v4,DFS树长这样：


V_1
V_2
V_3
V_4
V_7
V_5
V_6
V_8

	v1	v2	v3	v4	v5	v6	v7	v8
VE	0	6	14	9	29	25	18	20
VL	0	6	14	10	29	25	18	20

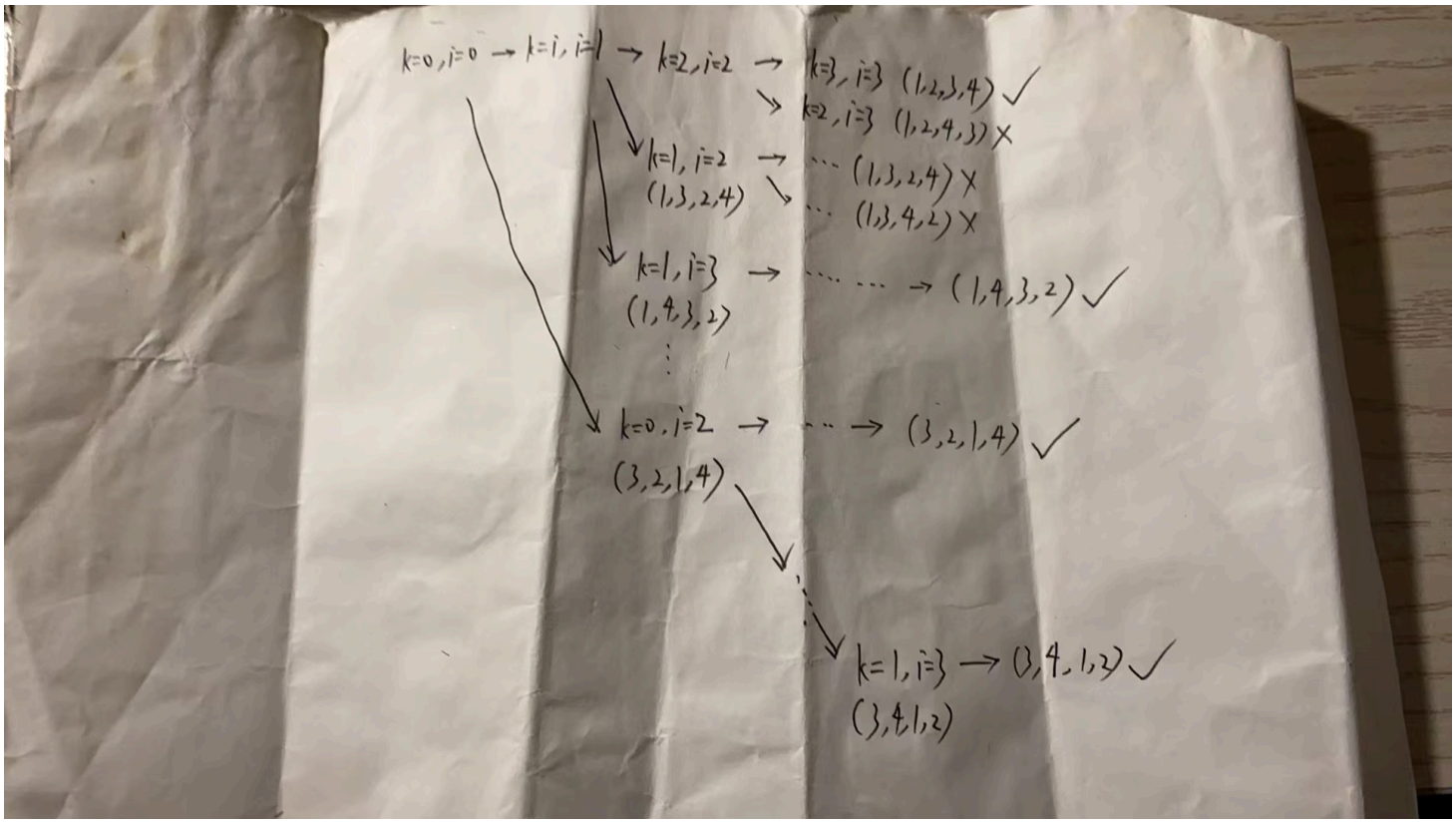
(3)

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14
e(i)	0	0	6	6	9	9	14	14	6	18	20	18	18	25
l(i)	0	5	7	6	10	13	15	14	6	18	20	18	20	25

(4)找e(i)=l(i)的情况，有a1,a4,a8,a9,a10,a11,a12,a14
所以关键路径有：
v1-v2-v7-v6-v5
v1-v2-v3-v7-v6-v5
v1-v2-v7-v8-v6-v5
v1-v2-v3-v7-v8-v6-v5

5.堆排序的过程，大家一定要理解啊
(1)56,53,40,47,35,38,25,41,23,17,27,31
(2)53,47,40,41,35,38,25,31,23,17,27,56

6.这题也给我整不会了
我直接跑了一下，结果是：
1 2 3 4
1 4 3 2
3 2 1 4
3 4 1 2
接下来让我们分析一下为什么是这个结果：
只有当state=0时才会输出东西，那什么时候state=0呢？当a[i]%2==(i+1)%2的时候state=1，说人话，就是当list的排序的奇偶性符合1234，也就是奇偶奇偶的时候会输出，所以显然是有四种可能的输出：1234、1324、3214、3412，那它们输出的顺序是什么呢，注意到k=n-1=3时才会输出，i=k时，for循环里的第一个swap实际上没什么效果，所以我们实际上是在不改动list的情况下一直递归到func(list,3,4)，这时候输出最原始的1 2 3 4；由于两次swap会相互抵消，类比这个操作，只要我们生成了符合输出的一组顺序，它就一定能迭代输出，所以谁先产生，谁就先输出。这里我的思路是把握住i和k的关系，首先，i一定大于等于k的，那我们要怎么变换呢？根据代码分析，最优先的变换是i和k同时+1，其次才是k不变，i+1，了解了这个规律，我就可以画一个有意思的图出来：



大概是这样，我有点不太会表述了，希望毅宝能提供更精妙的解法，实在不行我习题课扯一下。

算法设计

算法设计估计是20分，一共2题，一题10分的样子，感觉算法设计题反而是最简单的

1.

(1)遍历一遍就知道了

```
void GetMinVal(NODE *pHead, int &minx) {
    NODE *p = pHead->next;
    minx = p->val;
    while (p!= NULL) {
        if (p->val < minx)
            minx = p->val;
        p = p->next;
    }
}
```

(2)链表版的选择排序

```
void SortLinkedList(NODE *pHead) {
    NODE *p, *q, *min;
    int temp;
    p = pHead->next;
    while (p!= NULL) {
        min = p;
        q = p->next;
        while (q!= NULL) {
            if (q->val < min->val)
                min = q;
            q = q->next;
        }
        if (min!= p) {
            temp = p->val;
            p->val = min->val;
            min->val = temp;
        }
        p = p->next;
    }
}
```

2.跟17-18的最后一题一样，爽!

(1)二叉检索树：左<中<右

```
bool checkBST(bTree *pBTree) {
    if (pBTree->lchild && (pBTree->lchild->data >= pBTree->data || !check
 BST(pBTree->lchild))) return false;
    if (pBTree->rchild && (pBTree->rchild->data <= pBTree->data || !check
 BST(pBTree->rchild))) return false;
    return true;
 }
```

(2)等概率条件下，那就是总深度/节点数就行

```
int countDepthTotal(bTree *pBTree, int depth) {
    int ret = depth;
    if (pBTree->lchild) ret += countDepthTotal(pBTree->lchild, depth +
1);
    if (pBTree->rchild) ret += countDepthTotal(pBTree->rchild, depth +
1);
    return ret;
 }
 int countNodeTotal(bTree *pBTree) {
    int ret = 1;
    if (pBTree->lchild) ret += countNodeTotal(pBTree->lchild);
    if (pBTree->rchild) ret += countNodeTotal(pBTree->rchild);
    return ret;
 }
 double calcASL(bTree *pBTree) {
    return (double)countDepthTotal(pBTree,1) / countNodeTotal(pBTree)
 }
```