作业8

1. 答：

（1）CAB

（2）14

（3）a(n-i+1)

（4）n-1

2. 答：

（1）对

（2）错，只需要移动top指针即可

3. 答：这句话正确，但是并不好，原因是增加了代码量负担的同时并没有显著地让时间空间复杂度变好。但需要指明的是，这种思想是有实际作用的，在查询大量矩阵乘积方面可以降低原本树形存储结构的的时间复杂度;这种做法在纯函数式编程语言中也是有用的。（本题实现即力扣232）

作业9

1. 答：

中缀：A+B\*C-D/E

后缀：ABCDE/-\*+

2. 答：

前缀：-\*+ABC/DE

中缀：A+B\*C-D/E

3. 答：

1

4. 答：

（1）curLen == 0; curLen==innerArray.length;

（2）this.isFull()?(rear+1):0

作业10

1. 答：不对，单个结点也可以是一棵树，该树的度为0，但不是空树。

2. 答：不对，二叉树的度有可能为1（由两个结点构成的树）或0（单一节点的树或空树）。

3. 答：二叉树的形态为，规模为3 对应5，规模为4对应14.(这里的规模我认为就是节点数)

有序树为

4. 答：这是国内发表的一篇关于二叉树一种特殊的遍历方式的文章。该算法是一种旨在利用计算机科学的树形数据结构描述生活中的实际问题的遍历方法。遍历的过程是：先访问左节点，将最左边的一支访问完全后再从底下往上遍历它们的右孩子以及左分支。（这也是为什么要用栈）。该算法与常见的先序中序后序遍历一样拥有O(n)的时间复杂度和空间复杂度。总体来讲这是一个切入点巧妙的遍历算法，但实际情况多为多叉树，需要对本算法进行一定的改进才可以用于真实问题的解决。

作业11

1. 答：

这是一篇总结现有的二叉树遍历算法，并给出一部分算法的流程化生成器的文章。文章利用R和T运算规则将常见的先序中序后序算法进行了拓展，并证明了两件事：1.经过两次R变换和两次C变换可以回到原遍历规则，即不同遍历算法之间会形成多个环；2.R(C(t))是有意义的，C(R(t))是没有意义的，(t指的是某一种遍历算法),即转换遍历算法时都使用C(t)->R(C(t))即可。这样为作者后面创造出遍历序列生成器提供了基础。接下来，作者显式地调用堆/栈/队列来消除已知遍历算法中的递归部分，便于编写统一生成器。最后，作者通过简单的编写思想，推荐使用函数式编程的方法构建该生成器，证明了生成算法可以遍历七个图，恰好能覆盖我们常见的遍历算法。我认为这项研究对于我们发现树和图这类较复杂的数据结构最底层的某些统一规律方面有着重要意义。

作业12

1. 答：先序序列是ABCDEFG，中序序列不可能是:C

(二叉树的先序序列和中序序列是进栈出栈的关系)

2. 答：

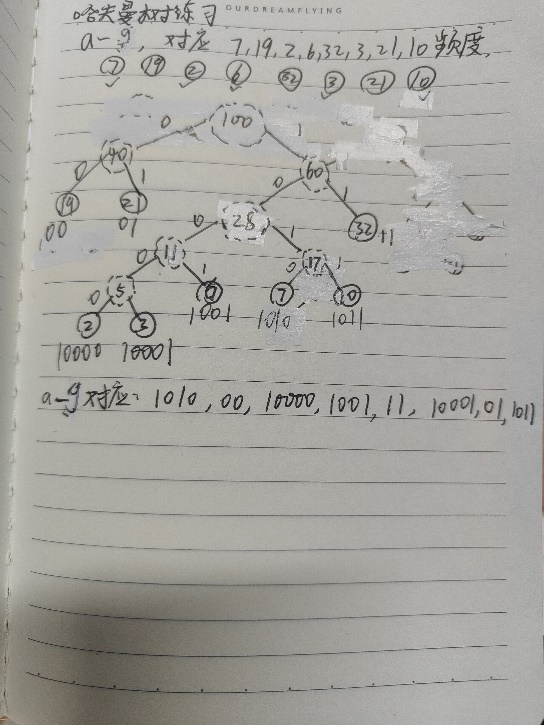
（1）把根左右和左根右的左字删掉，都成了根右，因此是任何结点的左子树全为空

（2）与（1）类似，是右子树空掉

（3）只有唯一结点的二叉树或空树

（4）（说明：可不做）层数小于等于2 的二叉树或只有左子树的二叉树.

3. 答：假设是a-g八个字符，使用频率先换成整数频度：7,19,2,6,32,3,21,10



4. 答：B

5. 答：

**两种建堆算法：威廉姆斯建堆算法和弗洛伊德建堆算法  
\* 1.威廉姆斯的建堆算法分为两步：  
\* 1.1和父元素比，如果比父元素小就放在父元素尾部  
\* 1.2如果比父元素大，要把父元素向下调整，再与更上层父元素比较，直到比父元素小或者到达最顶部  
\* 时间复杂度为O(n\*logn)  
\* 2.弗洛伊德的建堆算法也分为两步：  
\* 2.1找到最后一个非叶子节点  
\* 2.2从后向前，对每个结点执行<下潜>  
\* 时间复杂度为O(n)  
\* <下潜>是指把一个结点和它的两个子节点中较大的进行交换,然后看有没有更多的左右孩子，如果有，继续下潜**

威廉姆斯建堆算法：

public int[] heapify(int[] arr){  
 for (int i = 1; i < arr.length; i++) {  
 int parent = arr[(i-1)>>1];  
 int child = arr[i];  
 int p = i;  
 int q = (i-1)>>1;  
 while(child>parent){  
 swap(p,q);  
 p = q;  
 q = (p-1)>>1;  
 parent = arr[q];  
 child = arr[p];  
 }  
 }  
   
}

弗洛伊德建堆算法：

public MaxHeap(int[] arr){   
 this.array = arr;  
 this.size = arr.length;  
 heapify();  
}

public void heapify(){  
 for (int i = (size>>1)-1; i >= 0 ; i--) {   
 down(i);  
 }  
}

//下潜

private void down(int parent) {  
 int left = (parent<<1)+1;  
 int right = left + 1;  
 int max = parent;//先假定父亲的更大  
 if(left<size&&array[left]>array[max]){  
 max = left;  
 }  
 if(right<size&&array[right]>array[max]){  
 max = right;  
 }  
 if(max!=parent){//找到了更大的孩子  
 swap(parent, max);  
 down(max);  
 }  
}

作业13

1. 答：n-1

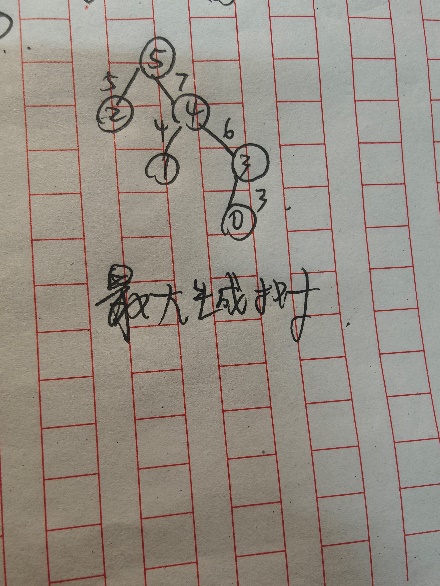
2. 答：n

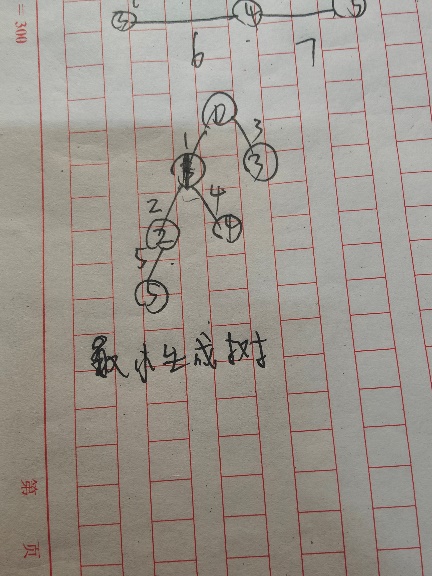
3. 答：21

完全图是条边，令该式大于200，n最小是21。

4. 答：图片在下面：

制作过程就是求最大删最小边使之恰好不连通，求最小删最大边使之恰好不连通。





作业14

1. 答：

（1）空树规定成-1，单个节点规定成0，这样往后方便+1.

（2）这里是无向图，如果是有向图可以用动态规划。对无向图使用记录父节点的手段换成有向图：

List<Integer>[] g;

int ans;

/\*\*

@param parent：parent[i]即为i的父节点

\*/

    public int longestPath(int[] parent) {

        this.s = s;

        var n = parent.length;

        g = new ArrayList[n];

        Arrays.setAll(g, e -> new ArrayList<>());

        for (var i = 1; i < n; i++) g[parent[i]].add(i);

        dfs(0);

        return ans + 1;

    }

    int dfs(int x) {

        var maxLen = 0;

        for (var y : g[x]) {

            var len = dfs(y) + 1；

                ans = Math.max(ans, maxLen + len);

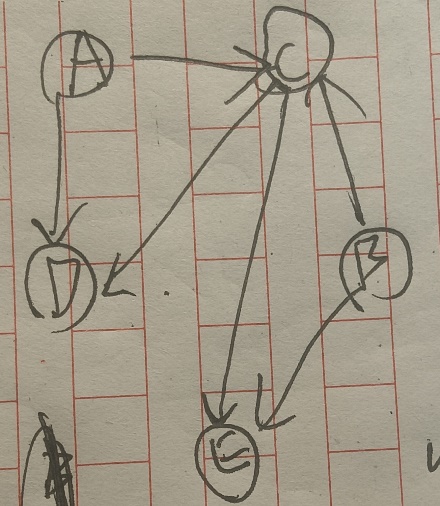
                maxLen = Math.max(maxLen, len);

        }

        return maxLen;

    }

2. 答：有向图见下，拓扑有序序列在更下方。



拓扑有序序列为:BACDE,ABCDE,ABCED,BACED