22级数据结构与算法每周一测题目汇总

开始时间	12/02/2023 4:44:00 PM	结束时间	12/08/2023 12:00:00 PM	答题时长	8356分钟
答卷类型	标准答案	总分	1000		

判断题

得分：暂无　总分：100

1-1 在具有N个结点的单链表中，访问结点和增加结点的时间复杂度分别对应为$O(1)$和$O(N)$。(5分)

○ T　● F

1-2 链表 - 存储结构 (5分)

链表中逻辑上相邻的元素，其物理位置也一定相邻。

○ T　● F

1-3 链表 - 存储结构 (5分)

链表是一种随机存取的存储结构。

○ T　● F

1-4 线性表L如果需要频繁地进行不同下标元素的插入、删除操作，此时选择顺序存储结构更好。(5分)

○ T　● F

1-5 对于顺序存储的长度为N的线性表，访问结点和增加结点的时间复杂度分别对应为$O(1)$和$O(N)$。(5分)

● T　○ F

1-6 若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算，则利用顺序表存储最节省时间。(5分)

● T　○ F

1-7 对于顺序存储的长度为N的线性表，删除第一个元素和插入最后一个元素的时间复杂度分别对应为$O(1)$和$O(N)$。(5分)

○ T　● F

1-8 序列{1,2,3,4,5}依次入栈，则不可能得到{3,4,1,2,5}的出栈序列。(5分)

● T　○ F

1-9 栈结构不会出现溢出现象。(5分)

○ T　● F

1-10 两个栈共享一片连续空间，可以将两个栈的栈底分别设在这片空间的两端。(5分)

● T　○ F

1-11 栈的特性 (5分)

栈是后进先出的线性表。

● T　○ F

1-12 堆栈适合解决处理顺序与输入顺序相反的问题。(5分)

● T　○ F

1-13 在一棵二叉搜索树上查找63，序列39、101、25、80、70、59、63是一种可能的查找时的结点值比较序列。(5分)

○ T　● F

1-14 在一棵由包含4、5、6等等一系列整数结点构成的二叉搜索树中，如果结点4和6在树的同一层，那么可以断定结点5一定是结 (5分)
点4和6的父亲结点。

○ T　● F

1-15 二叉搜索树的查找和折半查找的时间复杂度相同。(5分)

○ T ◉ F

1-16 无向连通图边数一定大于顶点个数减1。(5分)

○ T ◉ F

1-17 用邻接矩阵法存储图，占用的存储空间数只与图中结点个数有关，而与边数无关。(5分)

◉ T ○ F

1-18 在一个有向图中，所有顶点的入度与出度之和等于所有边之和的2倍。(5分)

◉ T ○ F

1-19 用一维数组`G[]`存储有4个顶点的无向图如下：(5分)

```
G[] = { 0, 1, 0, 1, 1, 0, 0, 0, 1, 0 }
```

则顶点2和顶点0之间是有边的。

◉ T ○ F

1-20 图是表示多对多关系的数据结构。(5分)

◉ T ○ F

单选题

得分：暂无　总分：450

2-1 将两个结点数都为N且都从小到大有序的单向链表合并成一个从小到大有序的单向链表，那么可能的最少比较次数是：(6分)

○ A. 1
◉ B. N
○ C. $2N$
○ D. $NlogN$

2-2 已知表头元素为`c`的单链表在内存中的存储状态如下表所示：(6分)

地址	元素	链接地址
1000H	a	1010H
1004H	b	100CH
1008H	c	1000H
100CH	d	NULL
1010H	e	1004H
1014H		

现将`f`存放于`1014H`处，并插入到单链表中，若`f`在逻辑上位于`a`和`e`之间，则`a`、`e`、`f`的“链接地址”依次是：

○ A. `1010H`，`1014H`，`1004H`
○ B. `1010H`，`1004H`，`1014H`
○ C. `1014H`，`1010H`，`1004H`
◉ D. `1014H`，`1004H`，`1010H`

2-3 链表 - 存储密度 (6分)

链表的存储密度 _____ 。

○ A. 大于 1　○ B. 等于 1　◉ C. 小于 1　○ D. 不能确定

2-4 双链表 - 插入结点 (6分)

在双链表中，将 s 所指新结点插入到 p 所指结点之后，其语句应该为 _____ 。

○ A. p->next = s; s->prev = p; s->next = p->next; p->next->prev = s;
○ B. s->next = p->next; s->prev = p; p->next = s; p->next->prev = s;
◉ C. s->next = p->next; p->next->prev = s; p->next = s; s->prev = p;

D. p->next = s; s->prev = p; p->next->prev = s; s->next = p->next;

2-5 链表的适用场合 (6分)

线性表在 _____ 情况下适合采用链式存储结构。

A. 线性表中数据元素的值需经常修改
B. 线性表需经常插入或删除数据元素
C. 线性表包含大量的数据元素
D. 线性表的数据元素包含大量的数据项

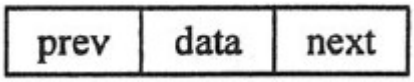
2-6 假设某个带头结点的单链表的头指针为head，则判定该表为空表的条件是（ ） (6分)

A. head==NULL
B. head->next==NULL
C. head!=NULL
D. head->next==head

2-7 若某线性表最常用的操作是在表头进行插入和删除，则利用哪种存储方式最合适? (6分)

A. 单链表
B. 双向链表
C. 数组
D. 广义表

2-8 现有非空双向链表 L，其结点结构为： (6分)



，`prev` 是指向直接前驱结点的指针，`next` 是指向直接后继结点的指针。若要在 L 中指针 `p` 所指向的结点（非尾结点）之后插入指针 `s` 指向的新结点，则在执行了语句序列 `s->next = p->next; p->next = s;` 后，下列语句序列中还需要执行的是：

A. `s->next->prev = p; s->prev = p;`
B. `p->next->prev = s; s->prev = p;`
C. `s->prev = s->next->prev; s->next->prev = s;`
D. `p->next->prev = s->prev; s->next->prev = p;`

2-9 数组A[1..5,1..6]每个元素占5个单元，将其按行优先次序存储在起始地址为1000的连续的内存单元中，则元素A[5,5]的地址为： (6分)

A. 1120
B. 1125
C. 1140
D. 1145

2-10 对于顺序存储的长度为N的线性表，访问结点和增加结点的时间复杂度为： (6分)

A. $O(1), O(1)$
B. $O(1), O(N)$
C. $O(N), O(1)$
D. $O(N), O(N)$

2-11 若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算，则利用哪种存储方式最节省时间? (6分)

A. 双链表
B. 单循环链表
C. 带头结点的双循环链表
D. 顺序表

2-12 顺序表中第一个元素的存储地址是100，每个元素的长度为2，则第5个元素的地址是（ ）。 (6分)

A. 100
B. 105
C. 108
D. 110

2-13 顺序表 - 地址计算 (6分)

假设顺序表第 1 个元素的地址是 100，每个元素占用 2 字节内存空间，则第 5 个元素的地址是 _____ 。

○ A. 104　○ B. 105　◉ C. 108　○ D. 110

2-14 顺序表 - 时间复杂度 (6分)

在包含 n 个数据元素的顺序表中，_____ 的时间复杂度为 $O(1)$。

◉ A. 访问第 $i(1 \leq i \leq n)$ 个数据元素　○ B. 在位序 $i(1 \leq i \leq n+1)$ 处插入一个新结点

○ C. 删除位序 $i(1 \leq i \leq n)$ 处的结点　○ D. 将 n 个元素按升序排序

2-15 已知二维数组 A 按行优先方式存储，每个元素占用 1 个存储单元。若元素 A[0][0] 的存储地址是 100，A[3][3] 的存储地址是 220，则元素 A[5][5] 的存储地址是: (6分)

○ A. 295

◉ B. 300

○ C. 301

○ D. 306

2-16 若某线性表最常用的操作是在表尾进行插入和删除，则利用哪种存储方式最合适? (6分)

◉ A. 数组　○ B. 单链表　○ C. 双向链表　○ D. 广义表

2-17 下列对顺序存储的有序表（长度为 n）实现给定操作的算法中，平均时间复杂度为 $O(1)$ 的是: (6分)

○ A. 查找包含指定值元素的算法

○ B. 插入包含指定值元素的算法

○ C. 删除第 i（$1 \leq i \leq n$）个元素的算法

◉ D. 获取第 i（$1 \leq i \leq n$）个元素的算法

2-18 设栈S和队列Q的初始状态均为空，元素a、b、c、d、e、f、g依次进入栈S。若每个元素出栈后立即进入队列Q，且7个元素出队的顺序是b、d、c、f、e、a、g，则栈S的容量至少是: (6分)

○ A. 1

○ B. 2

◉ C. 3

○ D. 4

2-19 若元素a、b、c、d、e、f依次进栈，允许进栈、退栈操作交替进行，但不允许连续三次进行退栈工作，则不可能得到的出栈序列是? (6分)

○ A. b c a e f d

○ B. c b d a e f

○ C. d c e b f a

◉ D. a f e d c b

2-20 有六个元素以6、5、4、3、2、1的顺序进栈，问哪个不是合法的出栈序列? (6分)

○ A. 2 3 4 1 5 6

◉ B. 3 4 6 5 2 1

○ C. 5 4 3 6 1 2

○ D. 4 5 3 1 2 6

2-21 若一个栈的入栈序列为1、2、3、…、N，其输出序列为p_1、p_2、p_3、…、p_N。若$p_1 = N$，则p_i为: (6分)

○ A. i

○ B. $n - i$

◉ C. $n - i + 1$

○ D. 不确定

2-22 设栈S和队列Q的初始状态均为空，元素{1, 2, 3, 4, 5, 6, 7}依次进入栈S。若每个元素出栈后立即进入队列Q，且7个元素出队的顺序是{2, 5, 6, 4, 7, 3, 1}，则栈S的容量至少是: (6分)

○ A. 1

○ B. 2

C. 3

D. 4

2-23 对空栈 S 进行 Push 和 Pop 操作，入栈序列为 a, b, c, d, e，经过 Push, Push, Pop, Push, Pop, Push, Push, Pop 操作后，得到的出栈序列是: (6分)

A. b, a, c

B. b, a, e

C. b, c, a

D. b, c, e

2-24 在作进栈运算时，应先判别栈是否(①)；在作退栈运算时应先判别栈是否(②)。当栈中元素为n个，作进栈运算时发生上溢，则说明该栈的最大容量为(③)。 (6分)

①: A. 空 B. 满 C. 上溢 D. 下溢
②: A. 空 B. 满 C. 上溢 D. 下溢
③: A. n-1 B. n C. n+1 D. n/2

A. ① C ② D ③ B

B. ① B ② A ③ B

C. ① B ② B ③ A

D. ① B ② A ③ A

2-25 已知普通表达式 `c/(e-f)*(a+b)`，对应的后缀表达式是（ ） (6分)

A. `cef/-ab*+`

B. `ef-c/ab+*`

C. `cef-/ab+*`

D. `c/e-f*a+b`

2-26 给定有限符号集 S , in 和 out 均为 S 中所有元素的任意排列。 对于初始为空的栈 ST, 下列叙述中，正确的是: (6分)

A. 若 in 是 ST 的入栈序列， 则不能判断 out 是否为其可能的出栈序列

B. 若 out 是 ST 的出栈序列，则不能判断 in 是否为其可能的入栈序列

C. 若 in 是 ST 的入栈序列，out 是对应 in 的出栈序列， 则 in 与 out 一定不同

D. 若 in 是 ST 的入栈序列，out 是对应 in 的出栈序列， 则 in 与 out 可能互为倒序

2-27 检查表达式中的括号是否匹配的问题需要借助_____来解决。 (6分)

A. 队列　　B. 堆栈　　C. 二叉搜索树　　D. 有向无环图

2-28 若已知一队列用单向链表表示，该单向链表的当前状态（含3个对象）是： `1->2->3`，其中 `x->y` 表示 `x` 的下一节点是 `y`。此时，如果将对象 `4` 入队，然后队列头的对象出队，则单向链表的状态是: (6分)

A. `1->2->3`

B. `2->3->4`

C. `4->1->2`

D. 答案不唯一

2-29 如果循环队列用大小为 `m` 的数组表示，且用队头指针 `front` 和队列元素个数 `size` 代替一般循环队列中的 `front` 和 `rear` 指针来表示队列的范围，那么这样的循环队列可以容纳的元素个数最多为: (6分)

A. `m-1`

B. `m`

C. `m+1`

D. 不能确定

2-30 循环顺序队列中是否可以插入下一个元素（）。 (6分)

A. 与队头指针和队尾指针的值有关

B. 只与队尾指针的值有关，与队头指针的值无关

C. 只与数组大小有关，与队首指针和队尾指针的值无关

D. 与曾经进行过多少次插入操作有关

2-31 最不适合用作链队的链表是（）。 (6分)

A. 只带队头指针的非循环双链表

B. 只带队头指针的循环双链表

C. 只带队尾指针的循环双链表

D. 只带队尾指针的循环单链表

2-32 现有队列 Q 与栈 S，初始时 Q 中的元素依次是{ 1, 2, 3, 4, 5, 6 }（1在队头），S 为空。若允许下列3种操作：（1）出队并输出出队元素；（2）出队并将出队元素入栈；（3）出栈并输出出栈元素，则不能得到的输出序列是: (6分)

A. 1, 2, 5, 6, 4, 3

B. 2, 3, 4, 5, 6, 1

C. 3, 4, 5, 6, 1, 2

D. 6, 5, 4, 3, 2, 1

2-33 循环队列的引入，目的是为了克服( )。 (6分)

A. 假溢出问题

B. 真溢出问题

C. 空间不够用

D. 操作不方便

2-34 栈和队列的共同点是（ ）。 (6分)

A. 都是先进先出

B. 都是先进后出

C. 只允许在端点处插入和删除元素

D. 没有共同点

2-35 已知初始为空的队列 Q 的一端仅能进行入队操作，另外一端既能进行入队操作又能进行出队操作。若 Q 的入队序列是 1、2、3、4、5，则不能得到的出队序列是: (6分)

A. 5、4、3、1、2

B. 5、3、1、2、4

C. 4、2、1、3、5

D. 4、1、3、2、5

2-36 对包含N个元素的散列表进行查找，平均查找长度为: (6分)

A. $O(1)$

B. $O(logN)$

C. $O(N)$

D. 不确定

2-37 将M个元素存入用长度为S的数组表示的散列表，则该表的装填因子为: (6分)

A. $S+M$

B. $M-S$

C. $M \times S$

D. M/S

2-38 下列因素中， 影响散列（哈希）方法平均查找长度的是 (6分)

I. 装填因子

II.散列函数

III. 冲突解决策略

A. 仅 I、II

B. 仅 I、III

C. 仅 II、III

D. I、II、III

2-39 若每个月要为全市八十岁以上的老人按年龄段发补助，即需要找到这些老人的信息并更新补助发放数据，则以下哪种是最合适的人口数据存储方式？注意：人口数据每天都在更新。 (6分)

A. 将居民信息按年龄从大到小存在线性表里

B. 将居民信息按年龄大小存在一棵平衡的二叉搜索树里

C. 将居民信息按年龄大小存在最大堆里

D. 将居民信息以年龄为键值存在哈希表里，用平方探测法解决冲突。

2-40 现有长度为 5、初始为空的散列表 HT，散列函数 $H(k)=(k+4)\%5$，用线性探查再散列法解决冲突。若将关键字序列 2022，12，25 依次插入 HT 中，然后删除关键字 25，则 HT 中查找失败的平均查找长度为: (6分)

A. 1　B. 1.6　C. 1.8　D. 2.2

2-41 将关键字序列{ 7，8，30，11，18，9，14 }，散列存储到散列列表中，散列表的存储空间是一个下标从 0 开始的一维数组。处理冲突采用线性探测法。散列函数为 $h(key)=(key\times 3)\%$ 表长，要求装入因子为 0.7。则成功查找的平均查找长度为 __ (6分)

A. 1.57

B. 1.00

C. 1.14

D. 1.29

2-42 如果二叉树的前序遍历结果是12345，后序遍历结果是32541，那么该二叉树的中序遍历结果是什么? (6分)

A. 23145

B. 23154

C. 24135

D. 无法确定

2-43 设 T 是非空二叉树，若 T 的后序遍历和中序遍历序列相同，则 T 的形态是 __ (6分)

A. 只有一个根结点

B. 没有度为 1 的结点

C. 所有结点只有左孩子

D. 所有结点只有右孩子

2-44 对一棵二叉树的结点从 1 开始顺序编号。要求每个结点的编号都小于其子树所有结点的编号，且左子树所有结点的编号都小于右子树所有结点的编号。可采用 _____ 实现编号。 (6分)

A. 先序遍历　B. 后序遍历　C. 中序遍历　D. 层次遍历

2-45 对于先序遍历与中序遍历结果相同的二叉树为（ ） (6分)

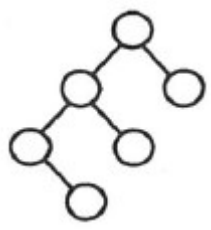
A. 一般二叉树

B. 任一结点均无右孩子的二叉树

C. 任一结点均无左子树的二叉树

D. 以上都不是

2-46 已知一棵二叉树的树形如下图所示，若其后序遍历序列为 f, d, b, e, c, a，则其先（前）序遍历序列是: (6分)



A. a, e, d, f, b, c　B. a, c, e, b, d, f

C. c, a, b, e, f, d　D. d, f, e, b, a, c

2-47 将d叉树堆存储在数组中。则对任意下标为i的单元，其父结点、第一个孩子结点、最后一个孩子结点的下标为: (6分)

A. $\lceil (i+d-2)/d \rceil$、$(i-2)d+2$、$(i-1)d+1$

B. $\lceil (i+d-1)/d \rceil$、$(i-2)d+1$、$(i-1)d$

C. $\lfloor (i+d-2)/d \rfloor$、$(i-1)d+2$、$id+1$

D. $\lfloor (i+d-1)/d \rfloor$、$(i-1)d+1$、$id$

2-48 设最小堆（小根堆）的层序遍历结果为{1, 3, 2, 5, 4, 7, 6}。用线性时间复杂度的算法将该堆调整为最大堆（大根堆），则该树的中序遍历结果为： (6分)

- A. 3, 5, 4, 2, 6, 1, 7
- B. 1, 4, 3, 7, 2, 6, 5
- C. 3, 5, 4, 7, 2, 6, 1
- D. 4, 1, 3, 7, 6, 2, 5

2-49 在一个有2333个元素的最小堆中，下列哪个下标不可能是最大元的位置？ (6分)

- A. 1116
- B. 1167
- C. 2047
- D. 2232

2-50 在下述结论中，正确的是： (6分)

① 只有2个结点的树的度为1；

② 二叉树的度为2；

③ 二叉树的左右子树可任意交换；

④ 在最大堆（大顶堆）中，从根到任意其它结点的路径上的键值一定是按非递增有序排列的。

- A. ①④
- B. ②④
- C. ①②③
- D. ②③④

2-51 设最小堆（小根堆）的层序遍历结果为 {8, 38, 25, 58, 52, 82, 70, 60}。用线性时间复杂度的算法将该堆调整为最大堆（大根堆），然后连续执行两次删除最大元素操作（DeleteMax）。则该树的中序遍历结果为： (6分)

- A. 60, 58, 8, 52, 38, 25
- B. 8, 58, 52, 60, 25, 38
- C. 38, 58, 52, 60, 8, 25
- D. 8, 58, 60, 32, 25, 38

2-52 下列关于大根堆（至少含 2 个元素）的叙述中，正确的是： (6分)

- (I). 可以将堆看成一棵完全二叉树
- (II). 可以采用顺序存储方式保存堆
- (III). 可以将堆看成一棵二叉排序树
- (IV). 堆中的次大值一定在根的下一层

- A. 仅 I、II
- B. 仅 II、III
- C. 仅 I、II、IV
- D. 仅 I、III、IV

2-53 将关键字 6、9、1、5、8、4、7 依次插入到初始为空的大根堆 H 中，得到的 H 是： (6分)

- A. 9、8、7、6、5、4、1
- B. 9、8、7、5、6、1、4
- C. 9、8、7、5、6、4、1
- D. 9、6、7、5、8、4、1

2-54 对二叉搜索树进行什么遍历可以得到从小到大的排序序列？ (6分)

- A. 前序遍历
- B. 后序遍历
- C. 中序遍历
- D. 层次遍历

2-55 将{28, 15, 42, 18, 22, 5, 40}依次插入初始为空的二叉搜索树。则该树的后序遍历结果是: (6分)

- A. 5, 15, 18, 22, 40, 42, 28
- B. 5, 22, 15, 40, 18, 42, 28
- C. 28, 22, 18, 42, 40, 15, 5
- D. 5, 22, 18, 15, 40, 42, 28

2-56 若一棵二叉树的前序遍历序列是{ 4, 2, 1, 3, 6, 5, 7 }，中序遍历序列是{ 1, 2, 3, 4, 5, 6, 7 }，则下列哪句是错的? (6分)

- A. 这是一棵完全二叉树
- B. 所有的奇数都在叶子结点上
- C. 这是一棵二叉搜索树
- D. 2是5的父结点

2-57 将{ 6, 9, 12, 3, 4, 8 }依次插入初始为空的二叉搜索树。则该树的后序遍历结果是: (6分)

- A. 4, 3, 6, 8, 12, 9
- B. 3, 4, 9, 8, 12, 6
- C. 3, 4, 6, 8, 12, 9
- D. 4, 3, 8, 12, 9, 6

2-58 若二叉搜索树是有N个结点的完全二叉树，则不正确的说法是: (6分)

- A. 所有结点的平均查找效率是$O(logN)$
- B. 最小值一定在叶结点上
- C. 最大值一定在叶结点上
- D. 中位值结点在根结点或根的左子树上

2-59 若一棵二叉树的后序遍历序列是{ 1, 3, 2, 6, 5, 7, 4 }，中序遍历序列是{ 1, 2, 3, 4, 5, 6, 7 }，则下列哪句是错的? (6分)

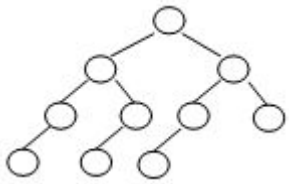
- A. 这是一棵完全二叉树
- B. 2是1和3的父结点
- C. 这是一棵二叉搜索树
- D. 7是5的父结点

2-60 将{ 32, 2, 15, 65, 28, 10 }依次插入初始为空的二叉搜索树。则该树的前序遍历结果是: (6分)

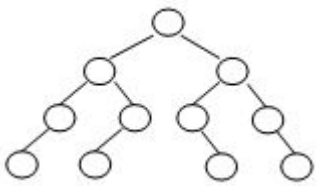
- A. 2, 10, 15, 28, 32, 65
- B. 32, 2, 10, 15, 28, 65
- C. 10, 28, 15, 2, 65, 32
- D. 32, 2, 15, 10, 28, 65

2-61 下列二叉树中，可能成为折半查找判定树（不含外部结点）的是: (6分)

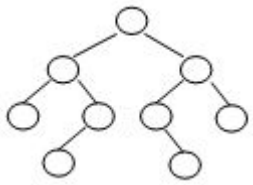
- A.



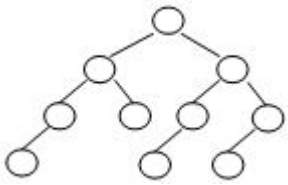
- B.



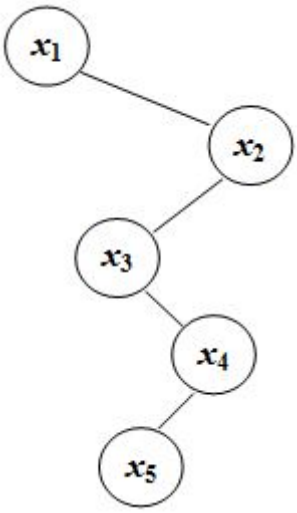
- C.



D.



2-62 已知二叉排序树如下图所示，元素之间应满足的大小关系是: (6分)



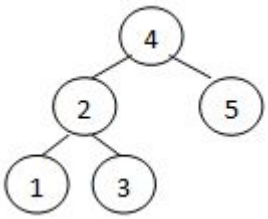
A. $x_1 < x_2 < x_5$

B. $x_1 < x_4 < x_5$

C. $x_3 < x_5 < x_4$

D. $x_4 < x_3 < x_5$

2-63 下列给定的关键字输入序列中，不能生成如下二叉排序树的是: (6分)



A. 4, 5, 2, 1, 3

B. 4, 5, 1, 2, 3

C. 4, 2, 5, 3, 1

D. 4, 2, 1, 3, 5

2-64 将 2, 1, 4, 5, 9, 3, 6, 7 顺序插入一棵初始为空的AVL树。下列句子中哪句是错的? (6分)

A. 4 是根结点

B. 3 和 7 是兄弟

C. 2 和 6 是兄弟

D. 9 是 7 的父结点

2-65 将一系列数字顺序一个个插入一棵初始为空的AVL树。下面哪个系列的第一次旋转是“右-左”双旋? (6分)

A. 1,2,3,4,5,6

B. 6,5,4,3,2,1

C. 4,2,5,6,3,1

D. 3,1,4,6,5,2

2-66 若一棵AVL树有 28 个结点，则该树的最大深度为__。空树的深度定义为0。 (6分)

A. 4

B. 5

C. 6

D. 7

2-67 首先将 28, 23, 54, 61, 98, 37 插入一棵初始为空的平衡二叉树（AVL树），然后马上插入下列选项中的一个键值。哪个键值将引起 RL 旋转? (6分)

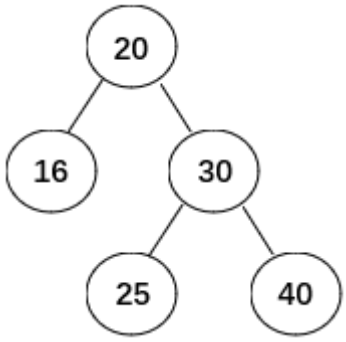
A. 10

B. 30

C. 60

D. 70

2-68 给定平衡二叉树如下图所示，插入关键字 23 后，根中的关键字是: (6分)



A. 16

B. 20

C. 23

D. 25

2-69 对于一个具有N个顶点的无向图，若采用邻接矩阵表示，则该矩阵的大小是: (6分)

A. $N-1$

B. N

C. $(N-1)^2$

D. N^2

2-70 下面关于图的存储的叙述中，哪一个是正确的? (6分)

A. 用相邻矩阵法存储图，占用的存储空间数只与图中结点个数有关，而与边数无关

B. 用相邻矩阵法存储图，占用的存储空间数只与图中边数有关，而与结点个数无关

C. 用邻接表法存储图，占用的存储空间数只与图中结点个数有关，而与边数无关

D. 用邻接表法存储图，占用的存储空间数只与图中边数有关，而与结点个数无关

2-71 将一个 10×10 的对称矩阵 M 的上三角部分的元素 $m_{i,j}$（$1 \leq i \leq j \leq 10$）按列优先存入C语言的一维数组 N 中，元素 $m_{7,2}$ 在 N在中的下标是: (6分)

A. 15

B. 16

C. 22

D. 23

2-72 一个有n个顶点的简单有向图最多有 （） 条边 (6分)

A. n

B. n(n-1)

C. n(n-1)/2

D. 2n

2-73 在一个具有n个顶点的有向图中，构成强连通图时至少有 （） 条边 (6分)

A. n

B. n+1
C. n-1
D. n/2

2-74 一个图的邻接矩阵中非0非∞的元素个数为奇数，则该图可能是（ ） (6分)

A. 有向图
B. 无向图
C. 无向图或有向图
D. 以上都不对

2-75 对于无向图 $G=(V,E)$, 下列选项中， 正确的是: (6分)

A. 当 $|V|>|E|$ 时，G 一定是连通的
B. 当 $|V|<|E|$ 时，G 一定是连通的
C. 当 $|V|=|E|-1$ 时，G 一定是不连通的
D. 当$|V|>|E|+1$ 时，G 一定是不连通的

函数题

得分：暂无　总分：150

6-1 链表逆置（15分）

本题要求实现一个函数，将给定单向链表逆置，即表头置为表尾，表尾置为表头。链表结点定义如下:

```
struct ListNode {
    int data;
    struct ListNode *next;
};
```

函数接口定义:

```
struct ListNode *reverse( struct ListNode *head );
```

其中 `head` 是用户传入的链表的头指针；函数 `reverse` 将链表 `head` 逆置，并返回结果链表的头指针。

裁判测试程序样例:

```
#include <stdio.h>
#include <stdlib.h>

struct ListNode {
    int data;
    struct ListNode *next;
};

struct ListNode *createlist(); /*裁判实现，细节不表*/
struct ListNode *reverse( struct ListNode *head );
void printlist( struct ListNode *head )
{
     struct ListNode *p = head;
     while (p) {
           printf("%d ", p->data);
           p = p->next;
     }
     printf("\n");
}

int main()
```

```
{
    struct ListNode  *head;

    head = createlist();
    head = reverse(head);
    printlist(head);

    return 0;
}

/* 你的代码将被嵌在这里 */
```

输入样例:

```
1 2 3 4 5 6 -1
```

输出样例:

```
6 5 4 3 2 1
```

编译器 GCC

代码

6-2 求链表的倒数第m个元素（15分）

请设计时间和空间上都尽可能高效的算法，在不改变链表的前提下，求链式存储的线性表的倒数第m（>0）个元素。

函数接口定义:

```
ElementType Find( List L, int m );
```

其中`List`结构定义如下:

```
typedef struct Node *PtrToNode;
struct Node {
    ElementType Data; /* 存储结点数据 */
    PtrToNode   Next; /* 指向下一个结点的指针 */
};
typedef PtrToNode List; /* 定义单链表类型 */
```

`L`是给定的带头结点的单链表；函数`Find`要将`L`的倒数第`m`个元素返回，并不改变原链表。如果这样的元素不存在，则返回一个错误标志`ERROR`。

裁判测试程序样例:

```
#include <stdio.h>
#include <stdlib.h>

#define ERROR -1

typedef int ElementType;
typedef struct Node *PtrToNode;
struct Node {
    ElementType Data;
    PtrToNode   Next;
};
```

```
typedef PtrToNode List;

List Read(); /* 细节在此不表 */
void Print( List L ); /* 细节在此不表 */

ElementType Find( List L, int m );

int main()
{
    List L;
    int m;
    L = Read();
    scanf("%d", &m);
    printf("%d\n", Find(L,m));
    Print(L);
    return 0;
}

/* 你的代码将被嵌在这里 */
```

输入样例:

```
5
1 2 4 5 6
3
```

输出样例:

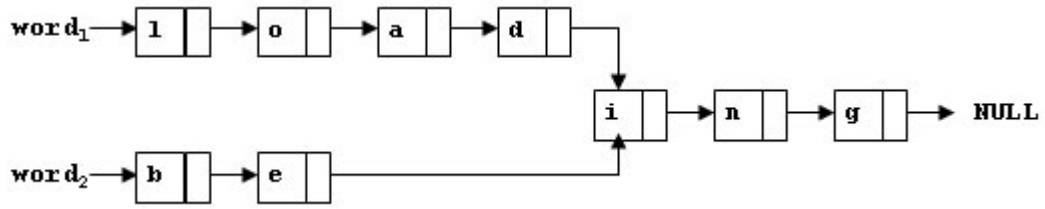
```
4
1 2 4 5 6
```

编译器　　GCC

代码

6-3 共享后缀的链表（15分）

有一种存储英文单词的方法，是把单词的所有字母串在一个单链表上。为了节省一点空间，如果有两个单词有同样的后缀，就让它们共享这个后缀。下图给出了单词“loading”和“being”的存储形式。本题要求你找出两个链表的公共后缀。



函数接口定义:

```
PtrToNode Suffix( List L1, List L2 );
```

其中`List`结构定义如下:

```
typedef struct Node *PtrToNode;
struct Node {
    ElementType Data; /* 存储结点数据 */
    PtrToNode   Next; /* 指向下一个结点的指针 */
```

```
};
typedef PtrToNode List; /* 定义单链表类型 */
```

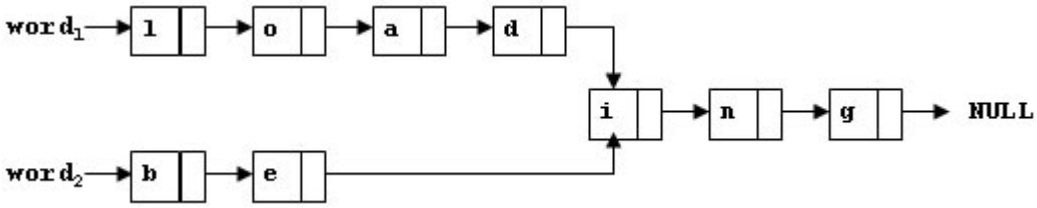
`L1`和`L2`都是给定的带头结点的单链表。函数`Suffix`应返回`L1`和`L2`的公共后缀的起点位置。

裁判测试程序样例：

```
#include <stdio.h>
#include <stdlib.h>

typedef char ElementType;

typedef struct Node *PtrToNode;
struct Node {
    ElementType Data; /* 存储结点数据 */
    PtrToNode   Next; /* 指向下一个结点的指针 */
};
typedef PtrToNode List; /* 定义单链表类型 */

void ReadInput( List L1, List L2 ); /* 裁判实现，细节不表 */
void PrintSublist( PtrToNode StartP ); /* 裁判实现，细节不表 */
PtrToNode Suffix( List L1, List L2 );

int main()
{
    List L1, L2;
    PtrToNode P;

    L1 = (List)malloc(sizeof(struct Node));
    L2 = (List)malloc(sizeof(struct Node));
    L1->Next = L2->Next = NULL;
    ReadInput( L1, L2 );
    P = Suffix( L1, L2 );
    PrintSublist( P );

    return 0;
}

/* 你的代码将被嵌在这里 */
```

输入样例：

如图存储的链表



输出样例：

```
ing
```

编译器　　GCC

代码

6-4 线性表元素的区间删除（15分）

给定一个顺序存储的线性表，请设计一个函数删除所有值大于min而且小于max的元素。删除后表中剩余元素保持顺序存储，并且相对位置不能改变。

函数接口定义：

```
List Delete( List L, ElementType minD, ElementType maxD );
```

其中`List`结构定义如下：

```
typedef int Position;
typedef struct LNode *List;
struct LNode {
    ElementType Data[MAXSIZE];
    Position Last; /* 保存线性表中最后一个元素在数组中的位置 */
};
```

`L`是用户传入的一个线性表，其中`ElementType`元素可以通过＞、＝＝、＜进行比较；`minD`和`maxD`分别为待删除元素的值域的下、上界。函数`Delete`应将`Data[]`中所有值大于`minD`而且小于`maxD`的元素删除，同时保证表中剩余元素保持顺序存储，并且相对位置不变，最后返回删除后的表。

裁判测试程序样例：

```
#include <stdio.h>

#define MAXSIZE 20
typedef int ElementType;

typedef int Position;
typedef struct LNode *List;
struct LNode {
    ElementType Data[MAXSIZE];
    Position Last; /* 保存线性表中最后一个元素的位置 */
};

List ReadInput(); /* 裁判实现，细节不表。元素从下标0开始存储 */
void PrintList( List L ); /* 裁判实现，细节不表 */
List Delete( List L, ElementType minD, ElementType maxD );

int main()
{
    List L;
    ElementType minD, maxD;
    int i;

    L = ReadInput();
    scanf("%d %d", &minD, &maxD);
    L = Delete( L, minD, maxD );
    PrintList( L );

    return 0;
}

/* 你的代码将被嵌在这里 */
```

输入样例:

```
10
4 -8 2 12 1 5 9 3 3 10
0 4
```

输出样例:

```
4 -8 12 5 9 10
```

编译器　　GCC

代码

6-5 双端队列（15分）

双端队列（deque，即double-ended queue的缩写）是一种具有队列和栈性质的数据结构，即可以（也只能）在线性表的两端进行插入和删除。若以顺序存储方式实现双端队列，请编写例程实现下列操作：

- `Push(X,D)`：将元素`X`插入到双端队列`D`的头；
- `Pop(D)`：删除双端队列`D`的头元素，并返回；
- `Inject(X,D)`：将元素`X`插入到双端队列`D`的尾部；
- `Eject(D)`：删除双端队列`D`的尾部元素，并返回。

函数接口定义:

```
bool Push( ElementType X, Deque D );
ElementType Pop( Deque D );
bool Inject( ElementType X, Deque D );
ElementType Eject( Deque D );
```

其中`Deque`结构定义如下:

```
typedef int Position;
typedef struct QNode *PtrToQNode;
struct QNode {
    ElementType *Data;      /* 存储元素的数组   */
    Position Front, Rear;   /* 队列的头、尾指针 */
    int MaxSize;            /* 队列最大容量     */
};
typedef PtrToQNode Deque;
```

注意：`Push`和`Inject`应该在正常执行完操作后返回true，或者在出现非正常情况时返回false。当`Front`和`Rear`相等时队列为空，`Pop`和`Eject`必须返回由裁判程序定义的`ERROR`。

裁判测试程序样例:

```
#include <stdio.h>
#include <stdlib.h>

#define ERROR -1
typedef int ElementType;
typedef enum { push, pop, inject, eject, end } Operation;
typedef enum { false, true } bool;
typedef int Position;
typedef struct QNode *PtrToQNode;
struct QNode {
    ElementType *Data;      /* 存储元素的数组   */
    Position Front, Rear;   /* 队列的头、尾指针 */
```

```
    int MaxSize;            /* 队列最大容量     */
};
typedef PtrToQNode Deque;

Deque CreateDeque( int MaxSize )
{   /* 注意：为区分空队列和满队列，需要多开辟一个空间 */
    Deque D = (Deque)malloc(sizeof(struct QNode));
    MaxSize++;
    D->Data = (ElementType *)malloc(MaxSize * sizeof(ElementType));
    D->Front = D->Rear = 0;
    D->MaxSize = MaxSize;
    return D;
}

bool Push( ElementType X, Deque D );
ElementType Pop( Deque D );
bool Inject( ElementType X, Deque D );
ElementType Eject( Deque D );

Operation GetOp();          /* 裁判实现，细节不表 */
void PrintDeque( Deque D ); /* 裁判实现，细节不表 */

int main()
{
    ElementType X;
    Deque D;
    int N, done = 0;

    scanf("%d", &N);
    D = CreateDeque(N);
    while (!done) {
        switch(GetOp()) {
        case push:
            scanf("%d", &X);
            if (!Push(X, D)) printf("Deque is Full!\n");
            break;
        case pop:
            X = Pop(D);
            if ( X==ERROR ) printf("Deque is Empty!\n");
            else printf("%d is out\n", X);
            break;
        case inject:
            scanf("%d", &X);
            if (!Inject(X, D)) printf("Deque is Full!\n");
            break;
        case eject:
            X = Eject(D);
            if ( X==ERROR ) printf("Deque is Empty!\n");
            else printf("%d is out\n", X);
            break;
        case end:
            PrintDeque(D);
            done = 1;
            break;
        }
    }
    return 0;
}
```

```
/* 你的代码将被嵌在这里 */
```

输入样例:

```
3
Pop
Inject 1
Pop
Eject
Push 2
Push 3
Eject
Inject 4
Inject 5
Inject 6
Push 7
Pop
End
```

输出样例:

```
Deque is Empty!
1 is out
Deque is Empty!
2 is out
Deque is Full!
Deque is Full!
3 is out
Inside Deque: 4 5
```

编译器 GCC

代码

6-6 线性探测法的查找函数（15分）

试实现线性探测法的查找函数。

函数接口定义:

```
Position Find( HashTable H, ElementType Key );
```

其中`HashTable`是开放地址散列表，定义如下:

```
#define MAXTABLESIZE 100000  /* 允许开辟的最大散列表长度 */
typedef int ElementType;     /* 关键词类型用整型 */
typedef int Index;           /* 散列地址类型 */
typedef Index Position;      /* 数据所在位置与散列地址是同一类型 */
/* 散列单元状态类型，分别对应：有合法元素、空单元、有已删除元素 */
typedef enum { Legitimate, Empty, Deleted } EntryType;

typedef struct HashEntry Cell; /* 散列表单元类型 */
struct HashEntry{
    ElementType Data; /* 存放元素 */
    EntryType Info;   /* 单元状态 */
};
```

```
typedef struct TblNode *HashTable; /* 散列表类型 */
struct TblNode {   /* 散列表结点定义 */
    int TableSize; /* 表的最大长度 */
    Cell *Cells;   /* 存放散列单元数据的数组 */
};
```

函数`Find`应根据裁判定义的散列函数`Hash( Key, H->TableSize )`从散列表`H`中查到`Key`的位置并返回。如果`Key`不存在，则返回线性探测法找到的第一个空单元的位置；若没有空单元，则返回`ERROR`。

裁判测试程序样例：

```
#include <stdio.h>

#define MAXTABLESIZE 100000  /* 允许开辟的最大散列表长度 */
typedef int ElementType;     /* 关键词类型用整型 */
typedef int Index;           /* 散列地址类型 */
typedef Index Position;      /* 数据所在位置与散列地址是同一类型 */
/* 散列单元状态类型，分别对应：有合法元素、空单元、有已删除元素 */
typedef enum { Legitimate, Empty, Deleted } EntryType;

typedef struct HashEntry Cell; /* 散列表单元类型 */
struct HashEntry{
    ElementType Data; /* 存放元素 */
    EntryType Info;   /* 单元状态 */
};

typedef struct TblNode *HashTable; /* 散列表类型 */
struct TblNode {   /* 散列表结点定义 */
    int TableSize; /* 表的最大长度 */
    Cell *Cells;   /* 存放散列单元数据的数组 */
};

HashTable BuildTable(); /* 裁判实现，细节不表 */
Position Hash( ElementType Key, int TableSize )
{
    return (Key % TableSize);
}

#define ERROR -1
Position Find( HashTable H, ElementType Key );

int main()
{
    HashTable H;
    ElementType Key;
    Position P;

    H = BuildTable();
    scanf("%d", &Key);
    P = Find(H, Key);
    if (P==ERROR)
        printf("ERROR: %d is not found and the table is full.\n", Key);
    else if (H->Cells[P].Info == Legitimate)
        printf("%d is at position %d.\n", Key, P);
    else
        printf("%d is not found.  Position %d is returned.\n", Key, P);

    return 0;
```

```
}

/* 你的代码将被嵌在这里 */
```

输入样例1：（注：-1表示该位置为空。下同。）

```
11
11 88 21 -1 -1 5 16 7 6 38 10
38
```

输出样例1：

```
38 is at position 9.
```

输入样例2：

```
11
11 88 21 -1 -1 5 16 7 6 38 10
41
```

输出样例2：

```
41 is not found.  Position 3 is returned.
```

输入样例3：

```
11
11 88 21 3 14 5 16 7 6 38 10
41
```

输出样例3：

```
ERROR: 41 is not found and the table is full.
```

编译器　　GCC

代码

6-7 分离链接法的删除操作函数（15分）

试实现分离链接法的删除操作函数。

函数接口定义：

```
bool Delete( HashTable H, ElementType Key );
```

其中`HashTable`是分离链接散列表，定义如下：

```
typedef struct LNode *PtrToLNode;
struct LNode {
    ElementType Data;
    PtrToLNode Next;
};
typedef PtrToLNode Position;
```

```
typedef PtrToLNode List;

typedef struct TblNode *HashTable; /* 散列表类型 */
struct TblNode {   /* 散列表结点定义 */
    int TableSize; /* 表的最大长度 */
    List Heads;    /* 指向链表头结点的数组 */
};
```

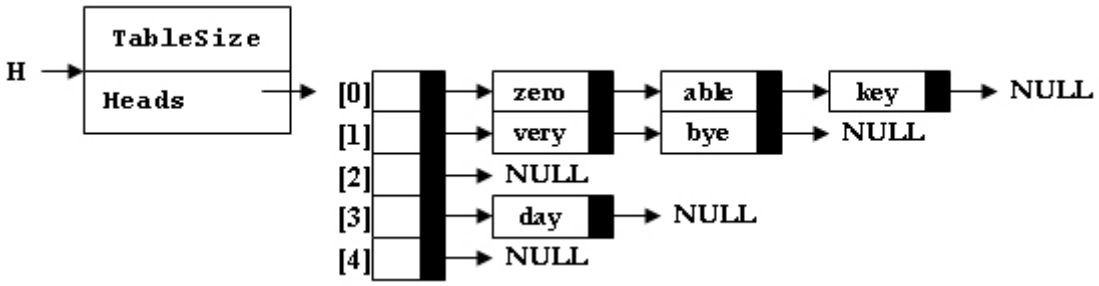
函数`Delete`应根据裁判定义的散列函数`Hash( Key, H->TableSize )`从散列表`H`中查到`Key`的位置并删除之，然后输出一行文字：`Key is deleted from list Heads[i]`，其中`Key`是传入的被删除的关键词，`i`是`Key`所在的链表的编号；最后返回true。如果`Key`不存在，则返回false。

裁判测试程序样例：

```
#include <stdio.h>
#include <string.h>

#define KEYLENGTH 15                  /* 关键词字符串的最大长度 */
typedef char ElementType[KEYLENGTH+1]; /* 关键词类型用字符串 */
typedef int Index;                    /* 散列地址类型 */
typedef enum {false, true} bool;

typedef struct LNode *PtrToLNode;
struct LNode {
    ElementType Data;
    PtrToLNode Next;
};
typedef PtrToLNode Position;
typedef PtrToLNode List;

typedef struct TblNode *HashTable; /* 散列表类型 */
struct TblNode {   /* 散列表结点定义 */
    int TableSize; /* 表的最大长度 */
    List Heads;    /* 指向链表头结点的数组 */
};

Index Hash( ElementType Key, int TableSize )
{
    return (Key[0]-'a')%TableSize;
}

HashTable BuildTable(); /* 裁判实现，细节不表 */
bool Delete( HashTable H, ElementType Key );

int main()
{
    HashTable H;
    ElementType Key;

    H = BuildTable();
    scanf("%s", Key);
    if (Delete(H, Key) == false)
        printf("ERROR: %s is not found\n", Key);
    if (Delete(H, Key) == true)
        printf("Are you kidding me?\n");
    return 0;
}
```

```
/* 你的代码将被嵌在这里 */
```

输入样例1：散列表如下图




```
able
```

输出样例1：

```
able is deleted from list Heads[0]
```

输入样例2：散列表如样例1图

```
date
```

输出样例2：

```
ERROR: date is not found
```

编译器　　GCC

代码

6-8 二叉树的遍历（15分）

本题要求给定二叉树的4种遍历。

函数接口定义：

```
void InorderTraversal( BinTree BT );
void PreorderTraversal( BinTree BT );
void PostorderTraversal( BinTree BT );
void LevelorderTraversal( BinTree BT );
```

其中`BinTree`结构定义如下：

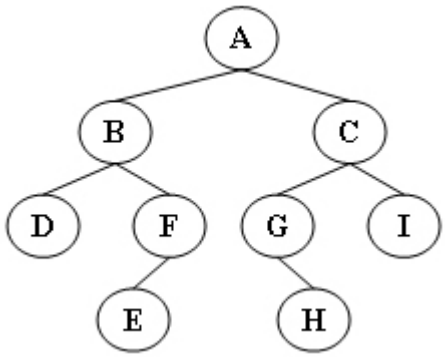
```
typedef struct TNode *Position;
typedef Position BinTree;
struct TNode{
    ElementType Data;
    BinTree Left;
    BinTree Right;
};
```

要求4个函数分别按照访问顺序打印出结点的内容，格式为一个空格跟着一个字符。

裁判测试程序样例：

```
#include <stdio.h>
#include <stdlib.h>

typedef char ElementType;
typedef struct TNode *Position;
typedef Position BinTree;
struct TNode{
    ElementType Data;
    BinTree Left;
    BinTree Right;
};

BinTree CreatBinTree(); /* 实现细节忽略 */
void InorderTraversal( BinTree BT );
void PreorderTraversal( BinTree BT );
void PostorderTraversal( BinTree BT );
void LevelorderTraversal( BinTree BT );

int main()
{
    BinTree BT = CreatBinTree();
    printf("Inorder:");    InorderTraversal(BT);    printf("\n");
    printf("Preorder:");   PreorderTraversal(BT);   printf("\n");
    printf("Postorder:");  PostorderTraversal(BT);  printf("\n");
    printf("Levelorder:"); LevelorderTraversal(BT); printf("\n");
    return 0;
}
/* 你的代码将被嵌在这里 */
```

输出样例（对于图中给出的树）：




```
Inorder: D B E F A G H C I
Preorder: A B D F E C G H I
Postorder: D E F B H G I C A
Levelorder: A B C D F G I E H
```

编译器　　GCC

代码

6-9 先序输出叶结点（15分）

本题要求按照先序遍历的顺序输出给定二叉树的叶结点。

函数接口定义：

```
void PreorderPrintLeaves( BinTree BT );
```

其中`BinTree`结构定义如下：

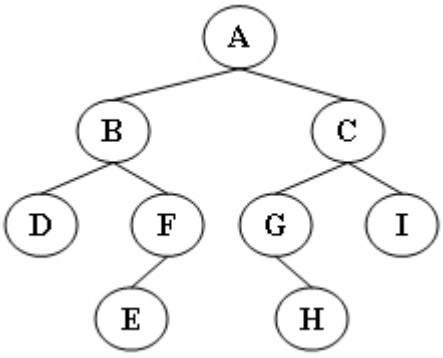
```
typedef struct TNode *Position;
typedef Position BinTree;
struct TNode{
    ElementType Data;
    BinTree Left;
    BinTree Right;
};
```

函数`PreorderPrintLeaves`应按照先序遍历的顺序输出给定二叉树`BT`的叶结点，格式为一个空格跟着一个字符。

裁判测试程序样例：

```
#include <stdio.h>
#include <stdlib.h>

typedef char ElementType;
typedef struct TNode *Position;
typedef Position BinTree;
struct TNode{
    ElementType Data;
    BinTree Left;
    BinTree Right;
};

BinTree CreatBinTree(); /* 实现细节忽略 */
void PreorderPrintLeaves( BinTree BT );

int main()
{
    BinTree BT = CreatBinTree();
    printf("Leaf nodes are:");
    PreorderPrintLeaves(BT);
    printf("\n");

    return 0;
}
/* 你的代码将被嵌在这里 */
```

输出样例（对于图中给出的树）：




```
Leaf nodes are: D E H I
```

编译器　　GCC

代码

6-10 二叉排序树查找操作（15分）

本题要求实现二叉排序树的查找操作。

函数接口定义：

```
BSTree SearchBST(BSTree T,ElemType e);
```

其中BSTree结构定义如下：

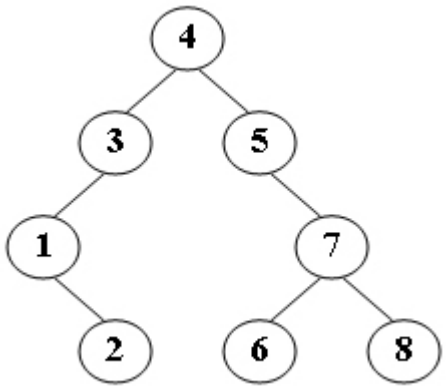
```
typedef int ElemType;
typedef struct BSTNode
{
    ElemType data;
    struct BSTNode *lchild,*rchild;
}BSTNode,*BSTree;
```

裁判测试程序样例：

```
#include <stdio.h>
#include <stdlib.h>
typedef int ElemType;
typedef struct BSTNode
{
    ElemType data;
    struct BSTNode *lchild,*rchild;
}BSTNode,*BSTree;
BSTree CreateBST();/* 二叉排序树创建，由裁判实现，细节不表 */
BSTree SearchBST(BSTree T,ElemType e);
void Inorder(BSTree T);/* 中序遍历，由裁判实现，细节不表 */

int main()
{
    BSTree T,result;
    ElemType n,e;
    T = CreateBST();
    printf("Inorder:");    Inorder(T);    printf("\n");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
    {
        scanf("%d",&e);
        result = SearchBST(T,e);
        if(result) printf("%d is found\n",result->data);
        else printf("%d is not found\n",e);
    }
    return 0;
}
/* 你的代码将被嵌在这里 */
```

输入样例：

输入第一行以-1结束的整数，为二叉排序树中元素初始序列，第二行为一个整数n，代表要查询的元素个数，第三行为n个整数，代表要查询的元素值。
例如创建如下二叉排序树，输入为：




```
4 3 5 1 2 7 6 8 -1
4
1 8 0 9
```

输出样例:

```
Inorder: 1 2 3 4 5 6 7 8
1 is found
8 is found
0 is not found
9 is not found
```

编译器 GCC

代码

编程题

得分：暂无　总分：300

7-1 两个有序链表序列的交集（20分）

已知两个非降序链表序列S1与S2，设计函数构造出S1与S2的交集新链表S3。

输入格式:

输入分两行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列）。数字用空格间隔。

输出格式:

在一行中输出两个输入序列的交集序列，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出`NULL`。

输入样例:

```
1 2 5 -1
2 4 5 8 10 -1
```

输出样例:

```
2 5
```

7-2 数组循环左移（20分）

本题要求实现一个对数组进行循环左移的简单函数：一个数组a中存有n（>0）个整数，在不允许使用另外数组的前提下，将每个整数循环向左移m（≥ 0）个位置，即将a中的数据由（$a_0a_1\cdots a_{n-1}$）变换为（$a_m\cdots a_{n-1}a_0a_1\cdots a_{m-1}$）（最前面的$m$个数循环移至最后面的$m$个位置）。如果还需要考虑程序移动数据的次数尽量少，要如何设计移动的方法？

输入格式:

输入第1行给出正整数n（≤ 100）和整数m（≥ 0）；第2行给出n个整数，其间以空格分隔。

输出格式:

在一行中输出循环左移m位以后的整数序列，之间用空格分隔，序列结尾不能有多余空格。

输入样例:

```
8 3
1 2 3 4 5 6 7 8
```

输出样例:

```
4 5 6 7 8 1 2 3
```

7-3 今天我要赢（20分）

2018 年我们曾经出过一题，是输出“2018 我们要赢”。今年是 2022 年，你要输出的句子变成了“我要赢！就在今天！”然后以比赛当天的日期落款。

输入格式:

本题没有输入。

输出格式:

输出分 2 行。在第一行中输出 `I'm gonna win! Today!`，在第二行中用 `年年年年-月月-日日` 的格式输出比赛当天的日期。已知比赛的前一天是 `2022-04-22`。

输入样例:

```
无
```

输出样例（第二行的内容要你自己想一想，这里不给出）:

```
I'm gonna win! Today!
这一行的内容我不告诉你…… 你要自己输出正确的日期呀~
```

7-4 种钻石（20分）

2019年10月29日，中央电视台专题报道，中国科学院在培育钻石领域，取得科技突破。科学家们用金刚石的籽晶片作为种子，利用甲烷气体在能量作用下形成碳的等离子体，慢慢地沉积到钻石种子上，一周“种”出了一颗 1 克拉大小的钻石。

本题给出钻石的需求量和人工培育钻石的速度，请你计算出货需要的时间。

输入格式:

输入在一行中给出钻石的需求量 N（不超过 10^7 的正整数，以微克拉为单位）和人工培育钻石的速度 v（$1 \le v \le 200$，以微克拉/天为单位的整数）。

输出格式:

在一行中输出培育 N 微克拉钻石需要的整数天数。不到一天的时间不算在内。

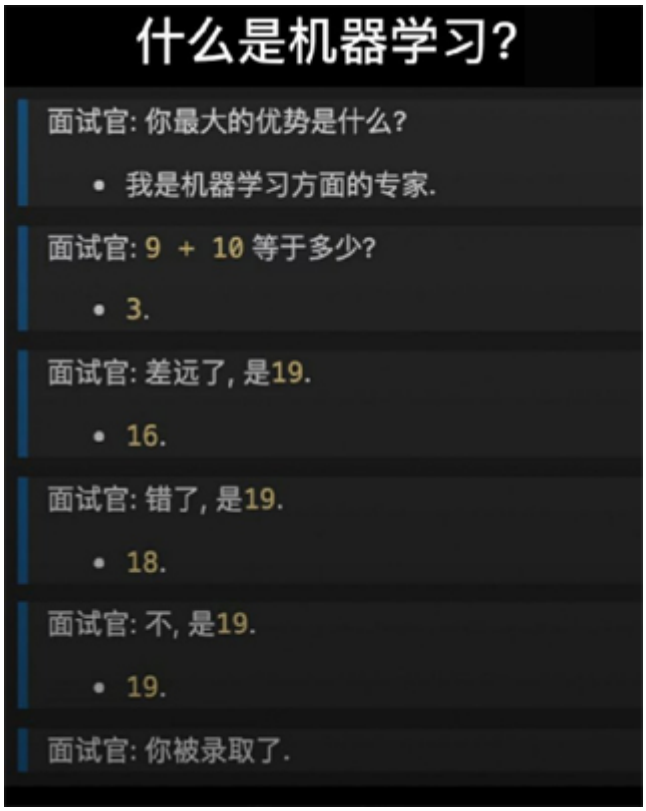
输入样例:

```
102000 130
```

输出样例:

```
784
```

7-5 什么是机器学习（20分）



什么是机器学习？上图展示了一段面试官与“机器学习程序”的对话：

```
面试官：9 + 10 等于多少？
答：3
面试官：差远了，是19。
答：16
面试官：错了，是19。
答：18
面试官：不，是19。
答：19
```

本题就请你模仿这个“机器学习程序”的行为。

输入格式：

输入在一行中给出两个整数，绝对值都不超过 100，中间用一个空格分开，分别表示面试官给出的两个数字 A 和 B。

输出格式：

要求你输出 4 行，每行一个数字。第 1 行比正确结果少 16，第 2 行少 3，第 3 行少 1，最后一行才输出 A+B 的正确结果。

输入样例：

```
9 10
```

输出样例：

```
3
16
18
19
```

7-6 包装机（20分）

一种自动包装机的结构如图 1 所示。首先机器中有 N 条轨道，放置了一些物品。轨道下面有一个筐。当某条轨道的按钮被按下时，活塞向左推动，将轨道尽头的一件物品推落筐中。当 0 号按钮被按下时，机械手将抓取筐顶部的一件物品，放到流水线上。图 2 显示了顺序按下按钮 3、2、3、0、1、2、0 后包装机的状态。

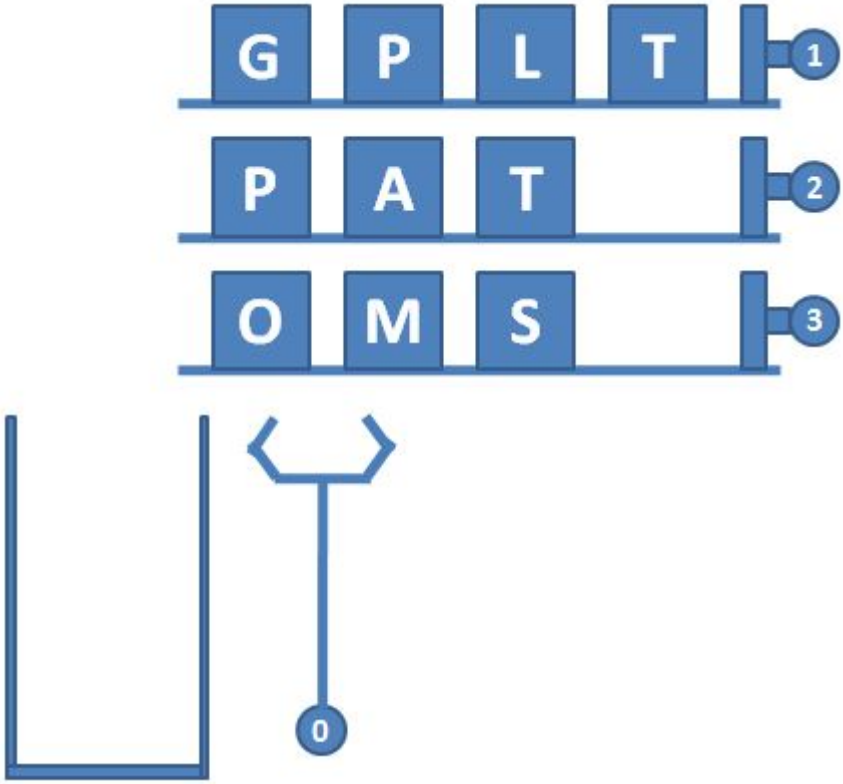


图1 自动包装机的结构

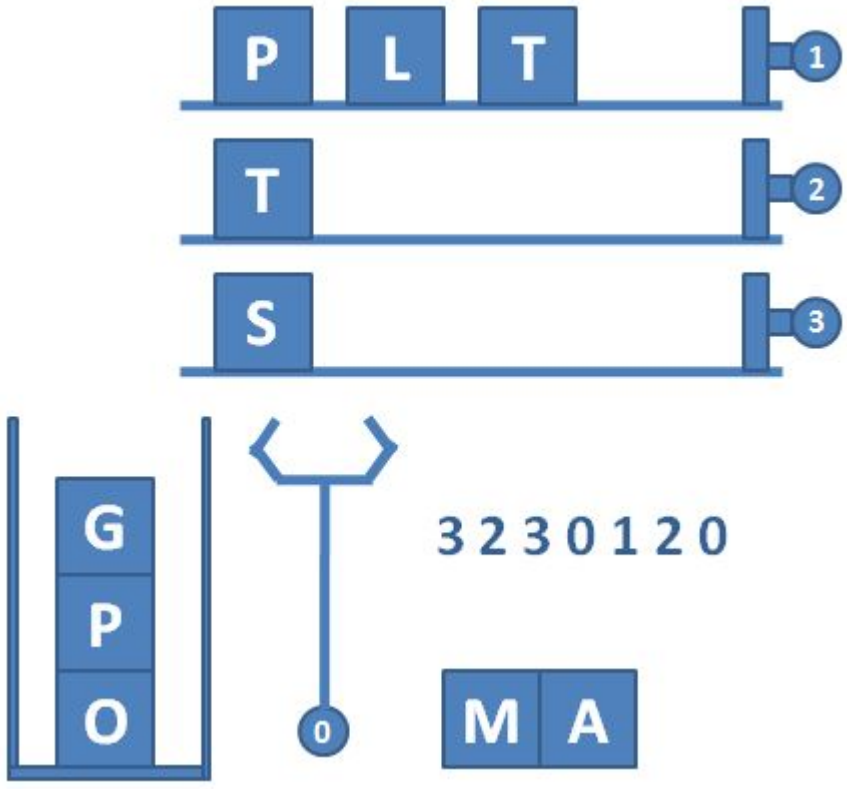


图 2 顺序按下按钮 3、2、3、0、1、2、0 后包装机的状态

一种特殊情况是，因为筐的容量是有限的，当筐已经满了，但仍然有某条轨道的按钮被按下时，系统应强制启动 0 号键，先从筐里抓出一件物品，再将对应轨道的物品推落。此外，如果轨道已经空了，再按对应的按钮不会发生任何事；同样的，如果筐是空的，按 0 号按钮也不会发生任何事。

现给定一系列按钮操作，请你依次列出流水线上的物品。

输入格式:

输入第一行给出 3 个正整数 N（≤ 100）、M（≤ 1000）和 S_{max}（≤ 100），分别为轨道的条数（于是轨道从 1 到 N 编号）、每条轨道初始放置的物品数量、以及筐的最大容量。随后 N 行，每行给出 M 个英文大写字母，表示每条轨道的初始物品摆放。

最后一行给出一系列数字，顺序对应被按下的按钮编号，直到 -1 标志输入结束，这个数字不要处理。数字间以空格分隔。题目保证至少会取出一件物品放在流水线上。

输出格式:

在一行中顺序输出流水线上的物品，不得有任何空格。

输入样例:

```
3 4 4
GPLT
PATA
OMSA
3 2 3 0 1 2 0 2 2 0 -1
```

输出样例:

```
MATA
```

7-7 插松枝（20分）

人造松枝加工场的工人需要将各种尺寸的塑料松针插到松枝干上，做成大大小小的松枝。他们的工作流程（并不）是这样的:

- 每人手边有一只小盒子，初始状态为空。
- 每人面前有用不完的松枝干和一个推送器，每次推送一片随机型号的松针片。
- 工人首先捡起一根空的松枝干，从小盒子里摸出最上面的一片松针 —— 如果小盒子是空的，就从推送器上取一片松针。将这片松针插到枝干的最下面。
- 工人在插后面的松针时，需要保证，每一步插到一根非空松枝干上的松针片，不能比前一步插上的松针片大。如果小盒子中最上面的松针满足要求，就取之插好；否则去推送器上取一片。如果推送器上拿到的仍然不满足要求，就把拿到的这片堆放到小盒子里，继续去推送器上取下一片。注意这里假设小盒子里的松针片是按放入的顺序堆叠起来的，工人每次只能取出最上面（即最后放入）的一片。

- 当下列三种情况之一发生时，工人会结束手里的松枝制作，开始做下一个：

（1）小盒子已经满了，但推送器上取到的松针仍然不满足要求。此时将手中的松枝放到成品篮里，推送器上取到的松针压回推送器，开始下一根松枝的制作。

（2）小盒子中最上面的松针不满足要求，但推送器上已经没有松针了。此时将手中的松枝放到成品篮里，开始下一根松枝的制作。

（3）手中的松枝干上已经插满了松针，将之放到成品篮里，开始下一根松枝的制作。

现在给定推送器上顺序传过来的 N 片松针的大小，以及小盒子和松枝的容量，请你编写程序自动列出每根成品松枝的信息。

输入格式:

输入在第一行中给出 3 个正整数：N（$\leq 10^3$），为推送器上松针片的数量；M（≤ 20）为小盒子能存放的松针片的最大数量；K（≤ 5）为一根松枝干上能插的松针片的最大数量。

随后一行给出 N 个不超过 100 的正整数，为推送器上顺序推出的松针片的大小。

输出格式:

每支松枝成品的信息占一行，顺序给出自底向上每片松针的大小。数字间以 1 个空格分隔，行首尾不得有多余空格。

输入样例:

```
8 3 4
20 25 15 18 20 18 8 5
```

输出样例:

```
20 15
20 18 18 8
25 5
```

7-8 汉诺塔的非递归实现（20分）

借助堆栈以非递归（循环）方式求解汉诺塔的问题（n, a, b, c），即将N个盘子从起始柱（标记为“a”）通过借助柱（标记为“b”）移动到目标柱（标记为“c”），并保证每个移动符合汉诺塔问题的要求。

输入格式:

输入为一个正整数N，即起始柱上的盘数。

输出格式:

每个操作（移动）占一行，按`柱1 -> 柱2`的格式输出。

输入样例:

```
3
```

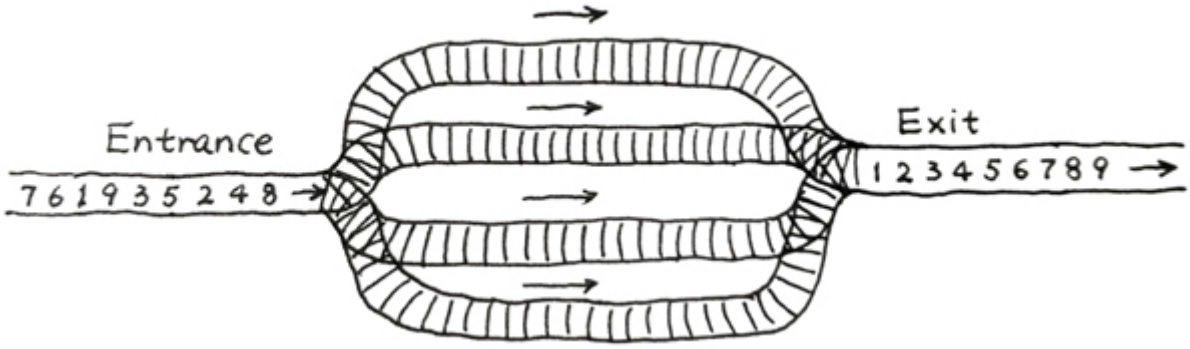
输出样例:

```
a -> c
a -> b
c -> b
a -> c
```

```
b -> a
b -> c
a -> c
```

7-9 列车调度（20分）

火车站的列车调度铁轨的结构如下图所示。



两端分别是一条入口（Entrance）轨道和一条出口（Exit）轨道，它们之间有`N`条平行的轨道。每趟列车从入口可以选择任意一条轨道进入，最后从出口离开。在图中有9趟列车，在入口处按照{8，4，2，5，3，9，1，6，7}的顺序排队等待进入。如果要求它们必须按序号递减的顺序从出口离开，则至少需要多少条平行铁轨用于调度？

输入格式:

输入第一行给出一个整数`N` ($2 \le$ `N` $\le 10^5$)，下一行给出从1到`N`的整数序号的一个重排列。数字间以空格分隔。

输出格式:

在一行中输出可以将输入的列车按序号递减的顺序调离所需要的最少的铁轨条数。

输入样例:

```
9
8 4 2 5 3 9 1 6 7
```

输出样例:

```
4
```

7-10 Windows消息队列（20分）

消息队列是Windows系统的基础。对于每个进程，系统维护一个消息队列。如果在进程中有特定事件发生，如点击鼠标、文字改变等，系统将把这个消息加到队列当中。同时，如果队列不是空的，这一进程循环地从队列中按照优先级获取消息。请注意优先级值低意味着优先级高。请编辑程序模拟消息队列，将消息加到队列中以及从队列中获取消息。

输入格式:

输入首先给出正整数N（$\le 10^5$），随后N行，每行给出一个指令——`GET`或`PUT`，分别表示从队列中取出消息或将消息添加到队列中。如果指令是`PUT`，后面就有一个消息名称、以及一个正整数表示消息的优先级，此数越小表示优先级越高。消息名称是长度不超过10个字符且不含空格的字符串；题目保证队列中消息的优先级无重复，且输入至少有一个`GET`。

输出格式:

对于每个`GET`指令，在一行中输出消息队列中优先级最高的消息的名称和参数。如果消息队列中没有消息，输出`EMPTY QUEUE!`。对于`PUT`指令则没有输出。

输入样例:

```
9
PUT msg1 5
PUT msg2 4
GET
PUT msg3 2
PUT msg4 4
GET
GET
GET
GET
```

输出样例:

```
msg2
msg3
msg4
msg1
EMPTY QUEUE!
```

7-11 关于堆的判断（20分）

将一系列给定数字顺序插入一个初始为空的小顶堆`H[]`。随后判断一系列相关命题是否为真。命题分下列几种：

- `x is the root`：`x`是根结点；
- `x and y are siblings`：`x`和`y`是兄弟结点；
- `x is the parent of y`：`x`是`y`的父结点；
- `x is a child of y`：`x`是`y`的一个子结点。

输入格式:

每组测试第1行包含2个正整数`N`（$\le$ 1000）和`M`（$\le$ 20），分别是插入元素的个数、以及需要判断的命题数。下一行给出区间$[-10000, 10000]$内的`N`个要被插入一个初始为空的小顶堆的整数。之后`M`行，每行给出一个命题。题目保证命题中的结点键值都是存在的。

输出格式:

对输入的每个命题，如果其为真，则在一行中输出`T`，否则输出`F`。

输入样例:

```
5 4
46 23 26 24 10
24 is the root
26 and 23 are siblings
46 is the parent of 23
23 is a child of 10
```

输出样例:

```
F
T
F
T
```

7-12 树种统计（20分）

随着卫星成像技术的应用，自然资源研究机构可以识别每一棵树的种类。请编写程序帮助研究人员统计每种树的数量，计算每种树占总数的百分比。

输入格式:

输入首先给出正整数N（$\leq 10^5$），随后N行，每行给出卫星观测到的一棵树的种类名称。种类名称由不超过30个英文字母和空格组成（大小写不区分）。

输出格式:

按字典序递增输出各种树的种类名称及其所占总数的百分比，其间以空格分隔，保留小数点后4位。

输入样例:

```
29
Red Alder
Ash
Aspen
Basswood
Ash
Beech
Yellow Birch
Ash
Cherry
Cottonwood
Ash
Cypress
Red Elm
Gum
Hackberry
White Oak
Hickory
Pecan
Hard Maple
White Oak
Soft Maple
Red Oak
Red Oak
White Oak
Poplan
Sassafras
Sycamore
Black Walnut
Willow
```

输出样例:

```
Ash 13.7931%
Aspen 3.4483%
Basswood 3.4483%
Beech 3.4483%
Black Walnut 3.4483%
Cherry 3.4483%
Cottonwood 3.4483%
Cypress 3.4483%
Gum 3.4483%
```

```
Hackberry 3.4483%
Hard Maple 3.4483%
Hickory 3.4483%
Pecan 3.4483%
Poplan 3.4483%
Red Alder 3.4483%
Red Elm 3.4483%
Red Oak 6.8966%
Sassafras 3.4483%
Soft Maple 3.4483%
Sycamore 3.4483%
White Oak 10.3448%
Willow 3.4483%
Yellow Birch 3.4483%
```

7-13 完全二叉搜索树（20分）

一个无重复的非负整数序列，必定对应唯一的一棵形状为完全二叉树的二叉搜索树。本题就要求你输出这棵树的层序遍历序列。

输入格式:

首先第一行给出一个正整数 N（≤ 1000），随后第二行给出 N 个不重复的非负整数。数字间以空格分隔，所有数字不超过2000。

输出格式:

在一行中输出这棵树的层序遍历序列。数字间以 1 个空格分隔，行首尾不得有多余空格。

输入样例:

```
10
1 2 3 4 5 6 7 8 9 0
```

输出样例:

```
6 3 8 1 5 7 9 0 2 4
```

7-14 网红点打卡攻略（20分）

一个旅游景点，如果被带火了的话，就被称为“网红点”。大家来网红点游玩，俗称“打卡”。在各个网红点打卡的快（省）乐（钱）方法称为“攻略”。你的任务就是从一大堆攻略中，找出那个能在每个网红点打卡仅一次、并且路上花费最少的攻略。

输入格式:

首先第一行给出两个正整数：网红点的个数 N（$1 < N \leq 200$）和网红点之间通路的条数 M。随后 M 行，每行给出有通路的两个网红点、以及这条路上的旅行花费（为正整数），格式为“网红点1 网红点2 费用”，其中网红点从 1 到 N 编号；同时也给出你家到某些网红点的花费，格式相同，其中你家的编号固定为 `0`。

再下一行给出一个正整数 K，是待检验的攻略的数量。随后 K 行，每行给出一条待检攻略，格式为：

$n\ V_1\ V_2 \cdots V_n$

其中 $n(\leq 200)$ 是攻略中的网红点数，V_i 是路径上的网红点编号。这里假设你从家里出发，从 V_1 开始打卡，最后从 V_n 回家。

输出格式:

在第一行输出满足要求的攻略的个数。

在第二行中，首先输出那个能在每个网红点打卡仅一次、并且路上花费最少的攻略的序号（从 1 开始），然后输出这个攻略的总路费，其间以一个空格分隔。如果这样的攻略不唯一，则输出序号最小的那个。

题目保证至少存在一个有效攻略，并且总路费不超过 10^9。

输入样例:

```
6 13
0 5 2
6 2 2
6 0 1
3 4 2
1 5 2
2 5 1
3 1 1
4 1 2
1 6 1
6 3 2
1 2 1
4 5 3
2 0 2
7
6 5 1 4 3 6 2
6 5 2 1 6 3 4
8 6 2 1 6 3 4 5 2
3 2 1 5
6 6 1 3 4 5 2
7 6 2 1 3 4 5 2
6 5 2 1 4 3 6
```

输出样例:

```
3
5 11
```

样例说明:

第 2、3、4、6 条都不满足攻略的基本要求，即不能做到从家里出发，在每个网红点打卡仅一次，且能回到家里。所以满足条件的攻略有 3 条。

第 1 条攻略的总路费是：(0->5) 2 + (5->1) 2 + (1->4) 2 + (4->3) 2 + (3->6) 2 + (6->2) 2 + (2->0) 2 = 14；

第 5 条攻略的总路费同理可算得：1 + 1 + 1 + 2 + 3 + 1 + 2 = 11，是一条更省钱的攻略；

第 7 条攻略的总路费同理可算得：2 + 1 + 1 + 2 + 2 + 2 + 1 = 11，与第 5 条花费相同，但序号较大，所以不输出。

7-15 哲哲打游戏（20分）

哲哲是一位硬核游戏玩家。最近一款名叫《达诺达诺》的新游戏刚刚上市，哲哲自然要快速攻略游戏，守护硬核游戏玩家的一切！

为简化模型，我们不妨假设游戏有 N 个剧情点，通过游戏里不同的操作或选择可以从某个剧情点去往另外一个剧情点。此外，游戏还设置了一些**存档**，在某个剧情点可以将玩家的游戏进度保存在一个档位上，读取存档后可以回到剧情点，重新进行操作或者选择，到达不同的剧情点。

为了追踪硬核游戏玩家哲哲的攻略进度，你打算写一个程序来完成这个工作。假设你已经知道了游戏的全部剧情点和流程，以及哲哲的游戏操作，请你输出哲哲的游戏进度。

输入格式:

输入第一行是两个正整数 N 和 M ($1 \leq N, M \leq 10^5$)，表示总共有 N 个剧情点，哲哲有 M 个游戏操作。

接下来的 N 行，每行对应一个剧情点的发展设定。第 i 行的第一个数字是 K_i，表示剧情点 i 通过一些操作或选择能去往下面 K_i 个剧情点；接下来有 K_i 个数字，第 k 个数字表示做第 k 个操作或选择可以去往的剧情点编号。

最后有 M 行，每行第一个数字是 0、1 或 2，分别表示：

- 0 表示哲哲做出了某个操作或选择，后面紧接着一个数字 j，表示哲哲在当前剧情点做出了第 j 个选择。我们保证哲哲的选择永远是合法的。
- 1 表示哲哲进行了一次存档，后面紧接着是一个数字 j，表示存档放在了第 j 个档位上。
- 2 表示哲哲进行了一次读取存档的操作，后面紧接着是一个数字 j，表示读取了放在第 j 个位置的存档。

约定：所有操作或选择以及剧情点编号都从 1 号开始。存档的档位不超过 100 个，编号也从 1 开始。游戏默认从 1 号剧情点开始。总的选项数（即 $\sum K_i$）不超过 10^6。

输出格式：

对于每个 1（即存档）操作，在一行中输出存档的剧情点编号。

最后一行输出哲哲最后到达的剧情点编号。

输入样例：

```
10 11
3 2 3 4
1 6
3 4 7 5
1 3
1 9
2 3 5
3 1 8 5
1 9
2 8 10
0
1 1
0 3
0 1
1 2
0 2
0 2
2 2
0 3
0 1
1 1
0 2
```

输出样例：

```
1
3
9
10
```

样例解释：

简单给出样例中经过的剧情点顺序：

1 -> 4 -> 3 -> 7 -> 8 -> 3 -> 5 -> 9 -> 10。

档位 1 开始存的是 1 号剧情点；档位 2 存的是 3 号剧情点；档位 1 后来又存了 9 号剧情点。