

DSA 数据结构与算法B笔试资料

Updated 1826 GMT+8 May 27, 2025

2024 spring, Compiled by Hongfei Yan

全是选择（Python）

1.1 绪论（5个题）

（1）在数据结构中，从逻辑上可以把数据结构分成（ C ）。

- A. 动态结构和静态结构
- B. 紧凑结构和非紧凑结构
- C. 线性结构和非线性结构
- D. 内部结构和外部结构

（2）与数据元素本身的形式、内容、相对位置、个数无关的是数据的（ C ）。

- A. 存储结构
- B. 存储实现
- C. 逻辑结构
- D. 运算实现

（3）通常要求同一逻辑结构中的所有数据元素具有相同的特性，这意味着(B)。

- A. 数据具有同一特点
- B. 不仅数据元素所包含的数据项的个数要相同，而且对应数据项的类型要一致
- C. 每个数据元素都一样
- D. 数据元素所包含的数据项的个数要相等

（4）以下说法正确的是（ D ）。

- A. 数据元素是数据的最小单位
- B. 数据项是数据的基本单位
- C. 数据结构是带有结构的各数据项的集合
- D. 一些表面上很不相同的数据可以有相同的逻辑结构

这道题考查的是对数据结构基本概念的理解。我们来逐项分析选项：

A. 数据元素是数据的最小单位

 错误。

解析：数据元素是数据的基本单位，但不是最小单位。数据元素可以由多个数据项组成，而数据项才是数据的最小单位。

B. 数据项是数据的基本单位

 错误。

解析：数据项是数据的最小单位，而不是基本单位。例如，“学生”这个数据元素可能包含“姓名”、“学号”、“年龄”等数据项。

C. 数据结构是带有结构的各数据项的集合

✗ 错误。

解析：数据结构是指带有结构的数据元素的集合，而不是数据项的集合。数据结构描述的是数据元素之间的相互关系和组织方式。

D. 一些表面上很不不同的数据可以有相同的逻辑结构

✓ 正确。

解析：数据的逻辑结构反映的是数据元素之间的逻辑关系，与具体的数据内容无关。例如，两个不同的应用（比如一个表示城市交通图，一个表示计算机网络）虽然数据不同，但都可能使用图结构来表示数据之间的连接关系。

(5) 以下与数据的存储结构无关的术语是（C）。

A. 顺序队列 B. 链表 C. 有序表 D. 链栈

有序表是指数据元素按某种顺序排列的逻辑结构，强调的是数据之间的顺序关系，而不是如何存储。

它可以采用不同的存储结构来实现，比如顺序存储（如数组）或链式存储（如链表）。

1.2 线性表（14个题）

(1) 在n个结点的顺序表中，算法的时间复杂度是O(1)的操作是（A）。

- A. 访问第i个结点 ($1 \leq i \leq n$) 和求第i个结点的直接前驱 ($2 \leq i \leq n$)
B. 在第i个结点后插入一个新结点 ($1 \leq i \leq n$)
C. 删除第i个结点 ($1 \leq i \leq n$)
D. 将n个结点从小到大排序

(2) 向一个有127个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动的元素个数为（B）。

A. 8 B. 63.5 C. 63 D. 7

这道题考查的是顺序表插入操作的平均移动次数。

已知条件：

- 顺序表中有 127个元素。
- 要插入一个新元素，并保持原有顺序不变。
- 插入位置是随机的，即在表中任何一个位置插入的可能性相同。

插入操作分析：

在顺序表中插入一个元素时，需要将插入位置之后的所有元素都向后移动一位，以腾出空间。

假设表中有 n 个元素，则插入位置可以是：

- 第 0 个位置（插在第一个元素前面）
- ...
- 第 n 个位置（插在最后一个元素后面）

所以一共有 $n + 1$ 个可能的插入位置。

对于每个插入位置 i （从 0 到 n ），需要移动的元素个数为：

$$n - i \quad (1)$$

因此，平均移动次数为所有插入位置所需移动次数的平均值：

$$\text{平均移动次数} = \frac{1}{n+1} \sum_{i=0}^n (n-i) = \frac{1}{n+1} \sum_{k=0}^n k = \frac{1}{n+1} \cdot \frac{n(n+1)}{2} = \frac{n}{2} \quad (2)$$

套用公式：

给定 $n = 127$ ，代入得：

$$\text{平均移动次数} = \frac{127}{2} = 63.5 \quad (3)$$

总结：

- 插入操作平均要移动一半的元素。
- 删除操作平均也要移动一半的元素。
- 这也是顺序表效率较低的原因之一。

(3) 链接存储的存储结构所占存储空间（A）。

- A. 分两部分，一部分存放结点值，另一部分存放表示结点间关系的指针
- B. 只有一部分，存放结点值
- C. 只有一部分，存储表示结点间关系的指针
- D. 分两部分，一部分存放结点值，另一部分存放结点所占单元数

(4) 线性表若采用链式存储结构时，要求内存中可用存储单元的地址（D）。

- A. 必须是连续的
- B. 部分地址必须是连续的
- C. 一定是不连续的
- D. 连续或不连续都可以

(5) 线性表 L 在 (B) 情况下适用于使用链式结构实现。

- A. 需经常修改 L 中的结点值 B. 需不断对 L 进行删除插入
C. L 中含有大量的结点 D. L 中结点结构复杂

(6) 单链表的存储密度 (C)。

- A. 大于1 B. 等于1 C. 小于1 D. 不能确定

(7) 将两个各有 n 个元素的有序表归并成一个有序表，其最少的比较次数是 (A)。

- A. n B. $2n-1$ C. $2n$ D. $n-1$

(8) 在一个长度为 n 的顺序表中，在第 i 个元素 ($1 \leq i \leq n+1$) 之前插入一个新元素时须向后移动 (B) 个元素。

- A. $n-i$ B. $n-i+1$ C. $n-i-1$ D. i

(9) 线性表 $L=(a_1, a_2, \dots, a_n)$ ，下列说法正确的是 (D)。

- A. 每个元素都有一个直接前驱和一个直接后继
B. 线性表中至少有一个元素
C. 表中诸元素的排列必须是由小到大或由大到小
D. 除第一个和最后一个元素外，其余每个元素都有一个且仅有一个直接前驱和直接后继。

(10) 若指定有 n 个元素的向量，则建立一个有序单链表的时间复杂性的量级是 (C)。

- A. $O(1)$ B. $O(n)$ C. $O(n^2)$ D. $O(n \log_2 n)$

一、什么是“建立一个有序单链表”？

“有序单链表”是指每次插入新节点时，都要保持链表整体有序（比如从小到大排列）。

给定的是一个含有 n 个元素的向量（数组），我们要将这些元素依次插入到链表中，并保证链表始终有序。

二、逐个分析插入操作的时间复杂度

对于单链表来说，插入一个元素要找到合适的位置，这需要从头遍历链表，直到找到插入位置。在最坏情况下，每个元素都要遍历当前整个链表才能找到插入点。

第1个元素： $O(0)$ 或 $O(1)$

- 链表为空，直接插入。

第2个元素：最多查找1次

- 找到插入位置。

第3个元素：最多查找2次

...

第i个元素：最多查找 i - 1 次

三、总的比较次数是多少？

总共进行的查找次数为：

$$0 + 1 + 2 + \cdots + (n - 1) = \frac{n(n - 1)}{2} = O(n^2) \quad (4)$$

四、补充说明

如果先对向量排序 ($O(n \log n)$)，然后用排序后的数据构造链表 ($O(n)$)，则总时间复杂度为 $O(n \log n)$ ，但这不是题干所指的做法。

题干强调的是“边插入边保持有序”，也就是典型的插入排序在链表上的实现，所以复杂度是 $O(n^2)$ 。

正确答案：C. $O(n^2)$

(11) 以下说法错误的是 (D)。

- A. 求表长、定位这两种运算在采用顺序存储结构时实现的效率不比采用链式存储结构时实现的效率低
- B. 顺序存储的线性表可以随机存取
- C. 由于顺序存储要求连续的存储区域，所以在存储管理上不够灵活
- D. 线性表的链式存储结构优于顺序存储结构

(12) 在单链表中，要将s所指结点插入到p所指结点之后，其语句应为 (D)。

- A. $s.next = p.next + 1$; $p.next = s$;
- B. $p.next = s$; $s.next = p.next$;
- C. $s.next = p.next$; $p.next = s.next$;
- D. $s.next = p.next$; $p.next = s$;

(13) 在双向链表存储结构中，删除p所指的结点时须修改指针 (A)。

- A. $p.next.prior = p.prior$; $p.prior.next = p.next$;

- B. $p.next = p.next.next; p.next.prior = p;$
C. $p.prior.next = p; p.prior = p.prior.prior;$
D. $p.prior = p.next.next; p.next = p.prior.prior;$

(14) 在双向循环链表中, 在 p 指针所指的结点后插入 q 所指向的新结点, 其修改指针的操作是 (C)。

- A. $p.next = q; q.prior = p; p.next.prior = q; q.next = q;$
B. $p.next = q; p.next.prior = q; q.prior = p; q.next = p.next;$
C. $q.prior = p; q.next = p.next; p.next.prior = q; p.next = q;$
D. $q.prior = p; q.next = p.next; p.next = q; p.next.prior = q;$

1.3 栈和队列 (15个题)

(1) 若让元素1, 2, 3, 4, 5依次进栈, 则出栈次序不可能出现在 (C) 种情况。

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

(2) 若已知一个栈的入栈序列是1, 2, 3, ..., n , 其输出序列为 $p_1, p_2, p_3, \dots, p_n$, 若 $p_1=n$, 则 p_i 为 (C)。

- A. i B. $n-i$ C. $n-i+1$ D. 不确定

这道题考查的是栈的特性和输出序列的规律。

题干:

若已知一个栈的入栈序列是1, 2, 3, ..., n , 其输出序列为 $p_1, p_2, p_3, \dots, p_n$, 若 $p_1=n$, 则 p_i 为 (C)

- A. i B. $n-i$ C. $n-i+1$ D. 不确定

分析:

栈是一种 **后进先出 (LIFO)** 的数据结构。

给定:

- 入栈序列为: 1, 2, 3, ..., n
- 输出序列为: p_1, p_2, \dots, p_n
- 已知第一个出栈元素 $p_1 = n$

这意味着最后一个入栈的元素 (也就是 n) 最先出栈。

由于栈是后进先出的, 所以如果 n 第一个出栈, 那说明:

- 所有元素 1 到 n 都已经入栈了 (因为 n 是最后入栈的)
- 然后从栈顶开始依次弹出: $n, n-1, \dots, 2, 1$

因此，输出序列为：

$$p_1 = n, \quad p_2 = n - 1, \quad p_3 = n - 2, \quad \dots, \quad p_i = n - i + 1 \quad (5)$$

示例验证（n=5）：

入栈顺序：1 → 2 → 3 → 4 → 5

全部入栈后，出栈顺序为：5, 4, 3, 2, 1

那么：

- $p_1 = 5 = 5 - 1 + 1$
- $p_2 = 4 = 5 - 2 + 1$
- $p_3 = 3 = 5 - 3 + 1$
- ...

符合公式： $p_i = n - i + 1$

 正确答案：C. $n-i+1$

 总结：

当栈的输入序列为 1 到 n，且第一个输出是 n（即最后一个入栈的元素），则整个输出序列就是逆序的 1 到 n，由此可得 $p_i = n - i + 1$ 。

(3) 数组 Q [n] 用来表示一个循环队列，f 为当前队列头元素的前一位置，r 为队尾元素的位置，假定队列中元素的个数小于 n，计算队列中元素个数的公式为（D）。

A. $r-f$ B. $(n+f-r)\%n$ C. $n+r-f$ D. $(n+r-f)\%n$

在循环队列中，由于队列是“循环”的，因此不能简单地用 $r-f$ 来计算元素个数。必须通过取模运算来处理循环情况

(4) 链式栈结点为：(data,link)，top 指向栈顶。若想摘除栈顶结点，并将删除结点的值保存到 x 中，则应执行操作（A）。

A. $x=top.data; top=top.link;$ B. $top=top.link; x=top.link;$
C. $x=top; top=top.link;$ D. $x=top.link;$

1 | top → 结点A → 结点B → 结点C → null

top 指向栈顶结点 A

A 的 link 指向下一个结点 B

(5) 设有一个递归算法如下


```

1  def fact(n):  #n大于等于0
2      if n <= 0:
3          return 1
4      else:
5          return n * fact(n - 1)

```

则计算fact(n)需要调用该函数的次数为（ A ）。

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

（6）栈在（ D ）中有所应用。

A. 递归调用 B. 函数调用 C. 表达式求值 D. 前三个选项都有

（7）为解决计算机主机与打印机间速度不匹配问题，通常设一个打印数据缓冲区。主机将要输出的数据依次写入该缓冲区，而打印机则依次从该缓冲区中取出数据。该缓冲区的逻辑结构应该是（ A ）。

A. 队列 B. 栈 C. 线性表 D. 有序表

（8）设栈S和队列Q的初始状态为空，元素e1、e2、e3、e4、e5和e6依次进入栈S，一个元素出栈后即进入Q，若6个元素出队的序列是e2、e4、e3、e6、e5和e1，则栈S的容量至少应该是（ B ）。

A. 2 B. 3 C. 4 D. 6

（9）在一个具有n个单元的顺序栈中，假设以地址高端作为栈底，以top作为栈顶指针，则当作进栈处理时，top的变化为（ C ）。

A. top不变 B. top=0 C. top -= 1 D. top += 1

（10）设计一个判别表达式中左，右括号是否配对出现的算法，采用（ D ）数据结构最佳。

A. 线性表的顺序存储结构 B. 队列

C. 线性表的链式存储结构 D. 栈

（11）用链接方式存储的队列，在进行删除运算时（ D ）。

A. 仅修改头指针 B. 仅修改尾指针

C. 头、尾指针都要修改 D. 头、尾指针可能都要修改

#特别是在删除队列中最后一个元素时，也需要更新尾指针。

（12）循环队列存储在数组A[0..m]中，则入队时的操作为（ D ）。

A. rear=rear+1 B. rear=(rear+1)%(m-1)

C. rear=(rear+1)%m D. rear=(rear+1)%(m+1)

（13）最大容量为n的循环队列，队尾指针是rear，队头是front，则队空的条件是（ B ）。

A. (rear+1)%n == front B. rear == front

C. rear+1 == front D. (rear-1)%n == front

条件	含义
<code>rear == front</code>	队列为空
<code>(rear + 1) % n == front</code>	队列为满

(14) 栈和队列的共同点是 (C)。

- 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

(15) 一个递归算法必须包括 (B)。

- A. 递归部分 B. 终止条件和递归部分
C. 迭代部分 D. 终止条件和迭代部分

1.4 串和数组 (6个题)

(1) 串是一种特殊的线性表，其特殊性体现在 (B)。

- A. 可以顺序存储 B. 数据元素是一个字符
C. 可以链式存储 D. 数据元素可以是多个字符

(2) 下面关于串的叙述中，(B) 是不正确的？

- A. 串是字符的有限序列 B. 空串是由空格构成的串
C. 模式匹配是串的一种重要运算 D. 串既可以采用顺序存储，也可以采用链式存储

(3) 串的长度是指 (B)。

- A. 串中所含不同字母的个数 B. 串中所含字符的个数
C. 串中所含不同字符的个数 D. 串中所含非空格字符的个数

(4) 若对 n 阶对称矩阵 A 以行序为主序方式将其下三角形的元素(包括主对角线上所有元素)依次存放于一维数组 $B[1..(n(n+1))/2]$ 中，则在 B 中确定 a_{ij} ($i < j$) 的位置 k 的关系为 (B)。

- A. $i*(i-1)/2+j$ B. $j*(j-1)/2+i$ C. $i*(i+1)/2+j$ D. $j*(j+1)/2+i$

题目给出的是一个对称矩阵 A 的下三角部分按行序优先(即先行后列)存入一维数组 $B[1..n(n+1)/2]$ ，要求我们求在这种存储方式下，上三角元素 a_{ij} (其中 $i < j$) 对应应在 B 中的位置 k 的计算公式。

分析：

由于矩阵是对称的，满足 $a_{ij}=a_{ji}$ 。所以对于 $i < j$ ，元素 a_{ij} 虽然不在下三角部分，但我们可以通过 a_{ji} 来找到它在 B 中的位置。

因此我们需要找到下三角中 a_{ji} (即 $j \geq i$) 在数组 B 中的位置。

下三角按行优先存储规则：

我们将矩阵的下三角（含主对角线）按行优先存入数组 B，即按照以下顺序：

a11; a21,a22; a31,a32,a33; ...; an1,an2,...,ann

每一行 i（从 1 到 n）中有 i 个元素：ai1,ai2,...,a_{ii}

要找的是 a_{ij} 的位置，假设 i < j，则找 a_{ji}

即我们要找下三角元素 a_{ji}（因为 j > i，所以 a_{ji} 在下三角），在数组 B 中的位置。

计算位置公式：

前 j-1 行共存了：

$$1 + 2 + \cdots + (j-1) = \frac{(j-1)j}{2}$$

所以第 j 行第 i 个元素（即 a_{ji}，其中 i ≤ j）在 B 中的位置为：

$$k = \frac{(j-1)j}{2} + i$$

(5) A[N, N]是对称矩阵，将下面三角（包括对角线）以行序存储到一维数组T[N(N+1)/2]中，则对任一上三角元素a[i][j]对应T[k]的下标k是（B）。重复了，同前一个题目(4)

A. i(i-1)/2+j B. j(j-1)/2+i C. i(j-i)/2+1 D. j(i-1)/2+1

(6) 设二维数组A[1.. m, 1.. n]（即m行n列）按行存储在数组B[1.. m*n]中，则二维数组元素A[i,j]在一维数组B中的下标为（A）。

A. (i-1) * n+j B. (i-1) * n+j-1 C. i * (j-1) D. j * m+i-1

1.5 树和二叉树（10个题）

(1) 把一棵树转换为二叉树后，这棵二叉树的形态是（A）。

A. 唯一的 B. 有多种
C. 有多种，但根结点都没有左孩子 D. 有多种，但根结点都没有右孩子

(2) 由3个结点可以构造出多少种不同的二叉树？（D）

A. 2 B. 3 C. 4 D. 5

构造不同的二叉树（Binary Tree）的数量，给定 n 个结点，是一个经典的组合数学问题。

对于 n 个不同的节点（这里如果没有特别说明节点是否相同，一般默认是无区别的节点），不同的二叉树结构数目由 卡特兰数（Catalan Number）决定：

$$C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)! \cdot n!} \quad (6)$$

如果是结构不同的二叉树，结点无区别，数量为卡特兰数第 n 项：

当 $n=3$ ：

$$C_3 = \frac{1}{4} \binom{6}{3} = \frac{20}{4} = 5 \quad (7)$$

答案：

D. 5

即：由 3 个结点可以构造出 **5 种不同的二叉树结构**。

(3) 一棵完全二叉树上有1001个结点，其中叶子结点的个数是（D）。

A. 250 B. 500 C. 254 D. 501

(4) 一个具有1025个结点的二叉树的高 h 为（D）。设独根树深度为0。

A. 11 B. 10 C. 11至1025之间 D. 10至1024之间

这道题考查的是完全二叉树的性质，尤其是叶子结点与总节点数之间的关系。

一、完全二叉树的基本性质

在完全二叉树中，有如下几个重要性质：

1. 若总节点数为 n ，则叶子结点个数为：

$$\left\lceil \frac{n}{2} \right\rceil \quad (8)$$

因为完全二叉树结构紧凑，下层叶子集中在最左边，所以一半左右是叶子。

2. 也可以从父子关系来理解：

- 每个非叶子结点（度为1或2）至少有一个孩子。
- 所以叶子结点的数量 = 总结点数 - 分支数（即父节点数量）

方法一：直接使用公式

对于完全二叉树，叶子结点个数公式为：

$$\text{叶子结点数} = \left\lceil \frac{n+1}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil \quad (9)$$

代入 $n = 1001$ ：

$$\left\lceil \frac{1001}{2} \right\rceil = \lceil 500.5 \rceil = 501 \quad (10)$$

✅ 正确答案：D. 501

📌 方法二：通过层次分析法验证

- 完全二叉树每一层的节点数为 $2^0, 2^1, 2^2, \dots, 2^{h-1}$ ，直到最后一层可能不满。
- 树的高度 h 满足：

$$2^{h-1} \leq n < 2^h \quad (11)$$

计算高度：

$$2^9 = 512, \quad 2^{10} = 1024 \Rightarrow h = 10 \quad (12)$$

前 9 层是满的，共：

$$2^0 + 2^1 + \dots + 2^8 = 2^9 - 1 = 511 \quad (13)$$

第10层有：

$$1001 - 511 = 490 \text{ 个结点} \quad (14)$$

这些 490 个结点都是叶子，且分布在最后一层。

再看倒数第二层（第9层）也有部分叶子：

- 第9层有 $2^8 = 256$ 个结点
- 每个结点最多有两个子结点
- 已经用了 490 个子结点 \rightarrow 最多需要 $\lceil \frac{490}{2} \rceil = 245$ 个父结点
- 所以第9层中有 $256 - 245 = 11$ 个没有孩子的结点，它们也是叶子

\rightarrow 叶子总数：

$$490 \text{ (最后一层)} + 11 \text{ (倒数第二层未被使用的父结点)} = 501 \quad (15)$$

✅ 结论不变：叶子结点个数为 501

📌 小贴士：常用结论记忆

类型	公式
完全二叉树叶子数	$\lceil \frac{n}{2} \rceil$
度为0结点数	$n_0 = \lceil \frac{n+1}{2} \rceil$
度为1结点数	$n_1 = 0 \text{ 或 } 1$ （取决于是否为满二叉树）
度为2结点数	$n_2 = n_0 - 1$

(5) 深度为h的满m叉树的第k层有 (A) 个结点。(1= $k \leq h$)

A. m^{k-1} B. m^k-1 C. m^{h-1} D. m^h-1

(6) 对二叉树的结点从1开始进行连续编号，要求每个结点的编号大于其左、右孩子的编号，同一结点的左右孩子中，其左孩子的编号小于其右孩子的编号，可采用 (C) 遍历实现编号。

A. 先序 B. 中序 C. 后序 D. 从根开始按层次遍历

也就是说：编号必须是 自底向上、从左到右 的顺序分配。

父节点只有在它的所有子节点都编号之后才能获得一个比它们大的编号。

(7) 若二叉树采用二叉链表存储结构，要交换其所有分支结点左右子树的位置，利用 (C) 遍历方法最合适。

A. 前序 B. 中序 C. 后序 D. 按层次

~~#A、C、D应该都对。C是类似于递归自底向上，D是自顶向下。这种题目与实现有关，出成选择题不合适。~~

这道题考查的是二叉树遍历方式与操作的适配性，尤其是当我们需要对每个分支结点进行某种操作（如交换左右子树）时，选择哪种遍历方式更合适。

一、理解问题要求

我们要做的是：交换所有分支结点的左右子树。

也就是说：

- 对于每一个非叶子结点（即有孩子的结点），都要交换它的左右孩子指针。
- 最终结果是整棵树被“镜像翻转”。

二、为什么“后序遍历”最合适？

后序遍历顺序：左 → 右 → 根

这种访问顺序的特点是：

- 在处理当前结点之前，已经处理了它的左右子树
- 这样在交换左右子树的时候，可以先处理子树，再处理父节点，保证操作不会干扰后续处理

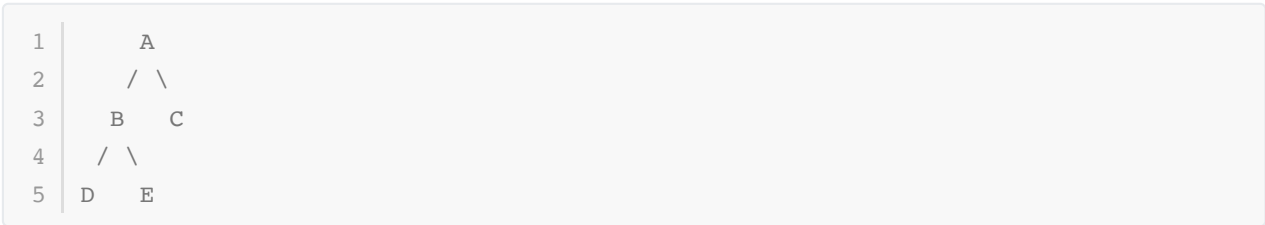
✅ 这正是我们想要的逻辑顺序！

三、对比其他遍历方式

遍历方式	访问顺序	是否适合
前序	根 → 左 → 右	✗ 先处理根，交换左右子树会影响后续访问子树的操作，容易出错
中序	左 → 根 → 右	✗ 不统一，中间处理根可能造成混乱
后序	左 → 右 → 根	✓ 先处理完子树，最后处理根，安全可靠
层次遍历	按层从上到下、从左到右	✗ 控制复杂，不便于递归实现交换

✓ 四、举个例子验证

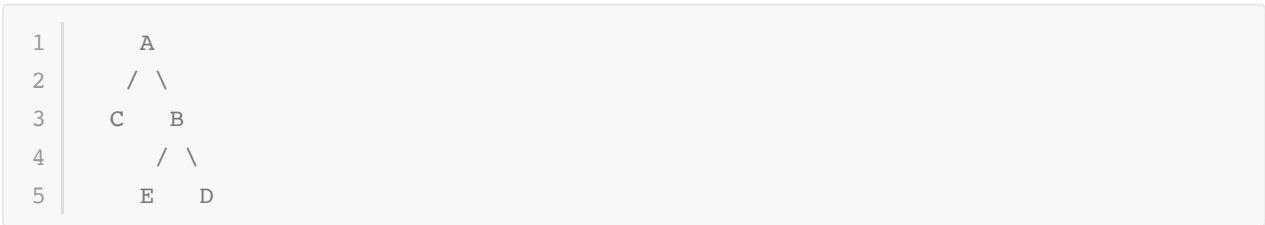
构造一棵简单二叉树：



使用后序遍历交换左右子树：

1. 处理 D（叶子，不做交换）
2. 处理 E（叶子，不做交换）
3. 处理 B，交换 D 和 E 的位置
4. 处理 C（叶子，不做交换）
5. 处理 A，交换 B 和 C 的位置

最终结果：



✓ 完全正确！

✓ 正确答案：C. 后序

📌 总结记忆技巧：

场景	推荐遍历方式
要先处理子树再处理父节点	后序遍历
交换左右子树	后序天然适合
操作父子依赖关系强的问题	后序最稳妥

最终答案：C. 后序

(8) 在下列存储形式中，（D）不是树的存储形式？

A. 双亲（parent）表示法 B. 孩子链表表示法 C. 孩子兄弟表示法 D. 顺序存储表示法

总结记忆技巧：

存储方式	是否是树的专用存储形式	说明
双亲表示法		用数组记录每个节点的父节点
孩子链表表示法		每个节点记录其所有孩子
孩子兄弟表示法		用“左孩子 + 右兄弟”模拟多叉树
顺序存储表示法		是一种通用的存储方式，并非树专用

(9) 一棵非空的二叉树的先序遍历序列与后序遍历序列正好相反，则该二叉树一定满足（C）。

A. 所有的结点均无左孩子 B. 所有的结点均无右孩子
C. 只有一个叶子结点 D. 是任意一棵二叉树

#C非空树的先序和后序相反，即“根左右”与“左右根”顺序相反，因此，树只有根结点，或者根结点只有左子树或右子树，依此类推，其子树有同样的性质。因此，树中所有非叶结点的度均为1，即二叉树仅有一个叶结点。充分必要条件是C，AB属于C。

(10) 设F是一个森林，B是由F变换得的二叉树。若F中有n个非终端结点，则B中右指针域为空的结点有（C）个。

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

解释及答案：

【数据结构】设F是一个森林，B是由F变换得的二叉树。若F中有n个非终端结点，则B中右指针域为空的结点有多少个

<https://blog.csdn.net/laue545/article/details/120107604>

在森林的每一颗子树中，转化二叉树后右指针域为空代表着这个节点是兄弟组的最后一个，再往右没有兄弟了。所以有多少个兄弟组就有多少右指针域为空。每个兄弟组都有自己的父亲，父亲就是非终端结点（分支结点）。

所以每棵子树转化后右指针域为空的结点树等于非终端结点（分支结点）。

又因为所有根结点转化后，最右边的根结点右指针域也为空。

所以最终转化后右指针域为空的结点总数为：

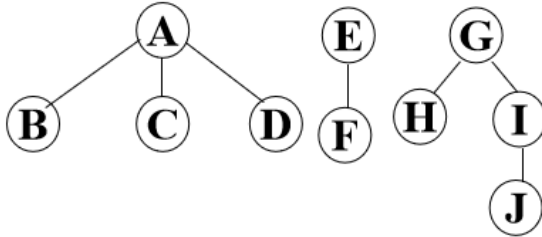
非终端结点数+1。

【专题】森林和二叉树的转换， https://blog.csdn.net/qq_42340716/article/details/134944165

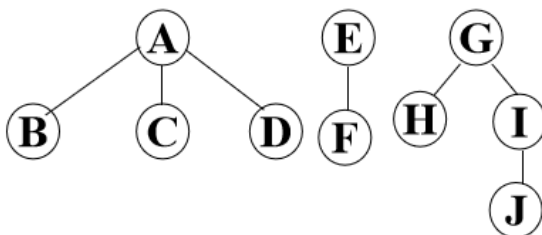
一、森林转换成二叉树^Q

- 步骤1：转换—将各棵树分别转换成二叉树；
- 步骤2：加线—将每棵树的根结点用线相连；
- 步骤3：旋转—以第一棵树根结点为二叉树的根，再以根结点为轴心，顺时针旋转，构成二叉树；

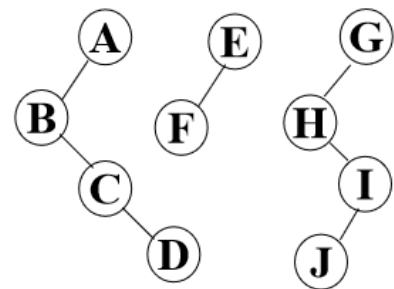
原森林



步骤一：转换——将各棵树分别转换成二叉树；

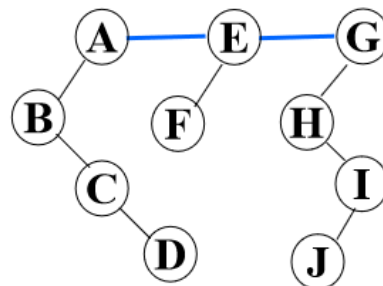
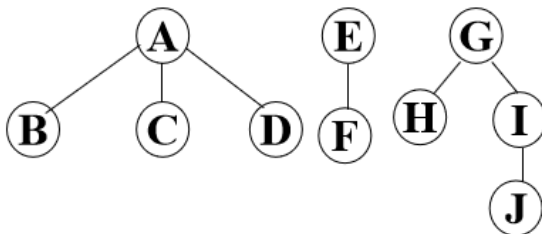


森林F

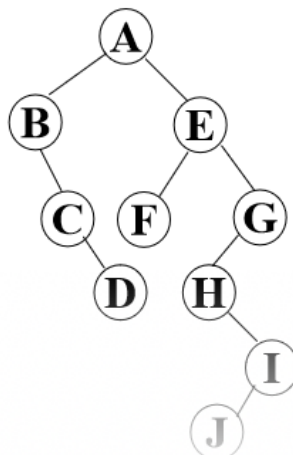
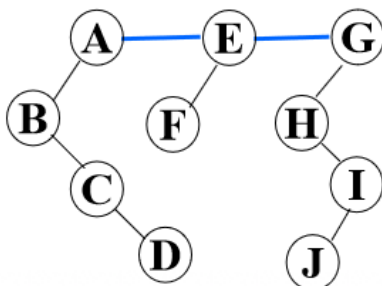


F中每棵树对应的二叉树

步骤二：加线——将每棵树的根结点用线相连；



步骤三：旋转——以第一棵树根结点为二叉树的根，再以根结点为轴心，顺时针旋转；



1.6 图 (15个题)

(1) 在一个图中，所有顶点的度数之和等于图的边数的 (C) 倍。

A. 1/2 B. 1 C. 2 D. 4

(2) 在一个有向图中，所有顶点的入度之和等于所有顶点的出度之和的 (B) 倍。

A. 1/2 B. 1 C. 2 D. 4

(3) 具有 n 个顶点的有向图最多有 (B) 条边。

A. n B. $n(n-1)$ C. $n(n+1)$ D. n^2

(4) n 个顶点的连通图用邻接矩阵表示时，该矩阵至少有 (B) 个非零元素。

A. n B. $2(n-1)$ C. $n/2$ D. n^2

这道题考查的是图的邻接矩阵表示与连通图的基本性质之间的关系。

✅ 题干：

n 个顶点的连通图用邻接矩阵表示时，该矩阵至少有 (B) 个非零元素。

选项如下：

- A. n
- B. $2(n-1)$ ✅
- C. $n/2$
- D. n^2



一、理解问题核心

我们的问题是：

对于一个有 n 个顶点的连通图，如果使用邻接矩阵来表示，那么这个矩阵中最少有多少个非零元素？



二、基本概念回顾

1. 邻接矩阵定义（无向图）

对于一个 无向图 来说：

- 邻接矩阵是一个 $n \times n$ 的对称矩阵
- 如果顶点 i 和 j 之间有一条边，则矩阵中的 $A[i][j] = A[j][i] = 1$ （或其他权重）
- 否则为 0

2. 连通图的最小边数

- 一个 n 个顶点的图要是连通图，至少需要形成一棵生成树
- 一棵生成树有且仅有 $n-1$ 条边

🎯 三、最少非零元素是多少？

每一条边在邻接矩阵中会表现为两个位置的非零值（因为是无向图）：

- 边 $(i, j) \rightarrow A[i][j]$ 和 $A[j][i]$ 都是非零

所以：

- 最少边数： $n-1$
- 对应非零元素数： $2 \times (n-1)$

✅ 正确答案：B. $2(n-1)$

✅ 总结记忆技巧：

内容	公式或结论
n 个顶点的连通图最少边数	$n - 1$ （生成树）
邻接矩阵中每条边占位	2 个位置（对称）
非零元素最少个数	$2 \times (n - 1)$ ✅

✅ 最终答案：B. $2(n-1)$

(5) G 是一个非连通无向图，共有 28 条边，则该图至少有（C）个顶点。

A. 7 B. 8 C. 9 D. 10

(6) 若从无向图的任意一个顶点出发进行一次深度优先搜索可以访问图中所有的顶点，则该图一定是（B）图。

A. 非连通 B. 连通 C. 强连通 D. 有向

(7) 下面（A）算法适合构造一个稠密图 G 的最小生成树。

A. Prim B. Kruskal C. Floyd D. Dijkstra

(8) 用邻接表表示图进行广度优先遍历时，通常借助（B）来实现算法。

A. 栈 B. 队列 C. 树 D. 图

(9) 用邻接表表示图进行深度优先遍历时，通常借助（A）来实现算法。

A. 栈 B. 队列 C. 树 D. 图

(10) 深度优先遍历类似于二叉树的（A）。

A. 先序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历

(11) 广度优先遍历类似于二叉树的（D）。

A. 先序遍历 B. 中序遍历 C. 后序遍历 D. 层次遍历

(12) 图的BFS生成树的树高比DFS生成树的树高（C）。

A. 小 B. 相等 C. 小或相等 D. 大或相等

生成树 (Spanning Tree) 是图论中的一个重要概念，指的是一个无向图的子图，它包含了原图的所有顶点，并且是一个树结构（即没有环，任意两点之间有且仅有一条路径）。生成树可以通过多种算法从一个连通无向图中生成，比如通过广度优先搜索（BFS）、深度优先搜索（DFS）等方法。

对于一个连通无向图($G=(V,E)$)，其中(V)是一组顶点，(E)是一组边，生成树($T=(V',E')$)满足以下条件：

1. ($V'=V$)，即生成树包含原图中的所有顶点。
2. (E')是(E)的一个子集，且不形成任何环。
3. 生成树是连通的，即对于(V')中的任意两个顶点，至少存在一条由(E')中的边组成的路径连接它们。

生成树在实际应用中许多用途，例如在网络设计、电路布线等领域。不同的生成树算法可能会产生不同的生成树结构，比如：

- **BFS生成树**：使用广度优先搜索算法遍历图时形成的生成树。BFS倾向于“水平”扩展，因此生成的树通常具有最小的高度，适用于寻找最短路径问题。
- **DFS生成树**：使用深度优先搜索算法遍历图时形成的生成树。DFS更倾向于深入探索，因此其生成树可能比BFS生成树更高，但在某些情况下能更快地遍历整个图。

此外，还有像Kruskal算法或Prim算法可以用来找到权重和最小的生成树，即最小生成树（Minimum Spanning Tree, MST），这对于需要考虑边权值的应用场景非常重要。

(13) 已知图的邻接矩阵如图所示邻接矩阵，则从顶点0出发按深度优先遍历的结果是（C）。

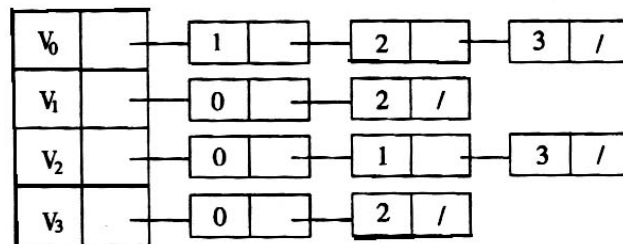
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(16)

A. 0243156 B. 0136542 C. 0134256 D. 0361542

(14) 已知图的邻接表如图所示，则从顶点0出发按广度优先遍历的结果是（D），按深度优先遍历的结果是（D）。

A. 0132 B. 0231 C. 0321 D. 0123



(15) 下面（B）方法可以判断出一个有向图是否有环。

A. 深度优先遍历 B. 拓扑排序 C. 求最短路径 D. 求关键路径

1.7 查找（13个题）

(1) 对n个元素的表做顺序查找时，若查找每个元素的概率相同，则平均查找长度为（C）。

A. (n-1)/2 B. n/2 C. (n+1)/2 D. n

#对于顺序查找，如果每个元素被查找的概率相同，并且元素在表中均匀分布，那么平均查找长度（Average Search Length, ASL）可以通过元素在表中的位置的期望值来计算。

考虑在顺序查找中，每个元素被查找时，它可能在表中的位置分别为1到n。因此，元素在表中的位置的期望值为：

$$\text{期望位置} = \frac{1+2+3+\dots+n}{n} = \frac{n(n+1)}{2n} = \frac{n+1}{2}$$

因此，平均查找长度为 $\frac{n+1}{2}$ 。所以答案是选项 C。

(2) 适用于折半查找的表的存储方式及元素排列要求为（D）。

A. 链接方式存储，元素无序 B. 链接方式存储，元素有序
C. 顺序方式存储，元素无序 D. 顺序方式存储，元素有序

(3) 当在一个有序的顺序表上查找一个数据时，既可用折半查找，也可用顺序查找，但前者比后者的查找速度（C）。

- A. 必定快 B. 不一定
C. 在大部分情况下要快 D. 取决于表递增还是递减

(4) 折半查找有序表（4, 6, 10, 12, 20, 30, 50, 70, 88, 100）。若查找表中元素58，则它将依次与表中（A）比较大小，查找结果是失败。

- A. 20, 70, 30, 50 B. 30, 88, 70, 50
C. 20, 50 D. 30, 88, 50

折半查找（也称为二分查找）是一种高效的查找算法，但前提是查找的列表必须是有序的。该算法每次将查找范围减半，从而快速定位目标值或者确定目标值不存在。

对于给定的有序表（4, 6, 10, 12, 20, 30, 50, 70, 88, 100），如果我们尝试查找元素58，过程如下：

1. 首先找到中间元素。对于有10个元素的列表，中间位置是第5个元素（索引从1开始算），即20。58大于20。
2. 因为58大于20，我们接下来在20右边的子列表中继续查找（20右边的子列表为30, 50, 70, 88, 100）。新子列表的中间元素是70。58小于70。
3. 由于58小于70，我们接下来在70左边且20右边的子列表中查找（这个新的子列表为30, 50）。新子列表的中间元素是50。58大于50。
4. 接下来我们比较58和30（因为58大于50，而在50与70之间的只有30）。显然，58不等于30，并且没有更多的元素可以比较了。

所以，查找58时，它依次与表中的元素20, 70, 30, 50进行比较，最终查找失败。因此正确答案是：

- A. 20, 70, 30, 50

其他选项不符合二分查找的过程。

(5) 对22个记录的有序表作折半查找，当查找失败时，至少需要比较（B）次关键字。


- A. 3 B. 4 C. 5 D. 6

这道题考查的是折半查找（二分查找）在查找失败时的比较次数问题。

题干：

对 22 个记录的有序表作折半查找，当查找失败时，至少需要比较（B）次关键字。

选项如下：

- A. 3
B. 4 
C. 5
D. 6

一、理解折半查找的原理

折半查找是一种高效的查找算法，适用于有序表，其基本思想是：

- 每次将中间位置的关键字与目标值比较
- 若相等则查找成功
- 若目标较小，则在左半部分继续查找
- 若目标较大，则在右半部分继续查找
- 重复这个过程，直到找到或查找范围为空（查找失败）

二、查找失败时的比较次数分析

虽然查找失败没有找到元素，但为了确认“确实不存在”，仍然要进行若干次比较。

我们要找的是：最坏情况下查找失败时，至少需要比较多少次关键字。

这个问题可以转化为：最多需要经过几次比较才能确定目标不在表中。

三、计算方法：使用判定树模型

折半查找的过程可以用一棵二叉判定树来表示：

- 每个节点代表一次比较
- 查找失败对应的是叶子节点的空孩子位置
- 所以查找失败时，比较次数 = 从根到该空节点所经历的边数（即比较次数）

我们可以用以下公式估算最大比较次数：

折半查找中，查找失败最多需要比较的次数为：

$$\lceil \log_2(n + 1) \rceil \quad (17)$$

其中 n 是记录数。

四、代入计算

对于 $n = 22$ ：

$$\lceil \log_2(22 + 1) \rceil = \lceil \log_2(23) \rceil \approx \lceil 4.53 \rceil = 5 \quad (18)$$

但这求的是最多比较次数。

而题目问的是：“至少需要比较几次关键字”？

我们换一种思路：在查找失败时，最短路径下也至少需要走多少步？

五、构造法验证：查找失败的最少比较次数

我们来看最小可能的深度是多少层可以容纳 22 个结点。

层数 h	最多可容纳节点数（完全二叉树）
1	1
2	3
3	7
4	15
5	31

说明：

- 若要容纳 22 个结点，判定树至少要有 **5 层**
- 而查找失败最多会走到第 5 层（也就是比较 5 次）
- 但注意！题目问的是：**查找失败时至少需要比较多少次？**

⚠ 这不是指“所有失败情况都只比较这么多次”，而是说：

在所有的查找失败路径中，**最短的一条路径上需要比较多少次关键字**

也就是说，我们关心的是查找失败路径上的**比较次数的最小值**，而不是最大值。

我们考虑一个具体例子：

比如当前表长为 22，查找某个关键字比所有元素都小或都大，那么它会在查找过程中提前终止。

但不管怎样，在判断树结构中，查找失败至少要经历 **4 次比较**才能确定查找失败。

✅ **正确答案：B. 4**

📖 **小贴士：总结常见结论**

内容	公式/结论
查找失败的最大比较次数	$\lceil \log_2(n + 1) \rceil$
查找失败的最小比较次数	至少需要 4 次（对 $n = 22$ ）
判定树高度	5 层可覆盖 22 个元素

✅ **最终答案：B. 4**

(6) 折半搜索与二叉排序树的时间性能（ C ）。

- A. 相同 B. 完全不同

C. 有时不相同 D. 数量级都是 $O(\log_2 n)$

#对于二叉搜索树（BST），如果树是平衡的，那么在平均情况下，查找、插入和删除的时间复杂度也是 $O(\log n)$ 。但是，如果树不平衡，例如退化为链表，那么时间复杂度可能会变为 $O(n)$ 。

（7）分别以下列序列构造二叉排序树，与用其它三个序列所构造的结果不同的是（C）。

A. (100, 80, 90, 60, 120, 110, 130)

B. (100, 120, 110, 130, 80, 60, 90)

C. (100, 60, 80, 90, 120, 110, 130)

D. (100, 80, 60, 90, 120, 130, 110)

（8）在平衡二叉树中插入一个结点后造成了不平衡，设最低的不平衡结点为A，并已知A的左孩子的平衡因子为0右孩子的平衡因子为1，则应作（C）型调整以使其平衡。

A. LL B. LR C. RL D. RR

（9）下列关于m阶B-树的说法错误的是（D）。

A. 根结点至多有m棵子树

B. 所有叶子都在同一层次上

C. 非叶结点至少有 $m/2$ (m为偶数)或 $m/2+1$ (m为奇数) 棵子树

D. 根结点中的数据是有序的

（10）下面关于哈希查找的说法，正确的是（C）。

A. 哈希函数构造的越复杂越好，因为这样随机性好，冲突小

B. 除留余数法是所有哈希函数中最好的

C. 不存在特别好与坏的哈希函数，要视情况而定

D. 哈希表的平均查找长度有时也和记录总数有关

（11）下面关于哈希查找的说法，不正确的是（A）。

A. 采用链地址法处理冲突时，查找一个元素的时间是相同的

B. 采用链地址法处理冲突时，若插入规定总是在链首，则插入任一个元素的时间是相同的

C. 用链地址法处理冲突，不会引起二次聚集现象

D. 用链地址法处理冲突，适合表长不确定的情况

（12）设哈希表长为14，哈希函数是 $H(\text{key})=\text{key}\%11$ ，表中已有数据的关键字为15, 38, 61, 84共四个，现要将关键字为49的元素加到表中，用二次探测法解决冲突，则放入的位置是（D）。

A. 8 B. 3 C. 5 D. 9

（13）采用线性探测法处理冲突，可能要探测多个位置，在查找成功的情况下，所探测的这些位置上的关键字（A）。

- A. 不一定都是同义词 B. 一定都是同义词
C. 一定都不是同义词 D. 都相同

1.8 排序（15个题）

(1) 从未排序序列中依次取出元素与已排序序列中的元素进行比较，将其放入已排序序列的正确位置上的方法，这种排序方法称为（ C ）。

- A. 归并排序 B. 冒泡排序 C. 插入排序 D. 选择排序

(2) 从未排序序列中挑选元素，并将其依次放入已排序序列（初始时为空）的一端的方法，称为（ D ）。

- A. 归并排序 B. 冒泡排序 C. 插入排序 D. 选择排序

(3) 对n个不同的关键字由小到大进行冒泡排序，在下列（ B ）情况下比较的次数最多。

- A. 从小到大排列好的 B. 从大到小排列好的
C. 元素无序 D. 元素基本有序

(4) 对n个不同的排序码进行冒泡排序，在元素无序的情况下比较的次数最多为（ D ）。

- A. $n+1$ B. n C. $n-1$ D. $n(n-1)/2$

(5) 快速排序在下列（ C ）情况下最易发挥其长处。

- A. 被排序的数据中含有多个相同排序码
B. 被排序的数据已基本有序
C. 被排序的数据完全无序
D. 被排序的数据中的最大值和最小值相差悬殊

(6) 对n个关键字作快速排序，在最坏情况下，算法的时间复杂度是（ B ）。

- A. $O(n)$ B. $O(n^2)$ C. $O(n\log_2 n)$ D. $O(n^3)$

(7) 若一组记录的排序码为（46, 79, 56, 38, 40, 84），则利用快速排序的方法，以第一个记录为基准得到的一次划分结果为（ C ）。

- A. 38, 40, 46, 56, 79, 84 B. 40, 38, 46, 79, 56, 84
C. 40, 38, 46, 56, 79, 84 D. 40, 38, 46, 84, 56, 79

解释：A是霍尔法，C是挖坑法。考试时候如果出现歧义题目，都算对。

参考：【排序算法】深入解析快速排序，https://blog.csdn.net/a_hong_sen/article/details/136967292

(8) 下列关键字序列中，（ D ）是堆。

- A. 16, 72, 31, 23, 94, 53 B. 94, 23, 31, 72, 16, 53

C. 16, 53, 23, 94, 31, 72 D. 16, 23, 53, 31, 94, 72

#要判断一个序列是否是堆，我们需要查看序列是否满足堆的性质。

在最大堆中，对于任意节点 i ，其父节点 $\text{parent}(i)$ 的值大于等于 i 节点的值。在最小堆中，对于任意节点 i ，其父节点 $\text{parent}(i)$ 的值小于等于 i 节点的值。

(9) 堆是一种 (B) 排序。

A. 插入 B. 选择 C. 交换 D. 归并

#在堆排序中，我们不断地从堆顶取出元素，并且对剩余的元素进行调整，使得堆的性质得到维护。这种不断地选择最大（或最小）元素的过程称为选择排序。因此，堆排序是一种选择排序算法。

(10) 堆的形状是一棵 (C) 。

A. 二叉排序树 B. 满二叉树 C. 完全二叉树 D. 平衡二叉树

(11) 若一组记录的排序码为 (46, 79, 56, 38, 40, 84)，则利用堆排序的方法建立的初始堆为 (B) 。

A. 79, 46, 56, 38, 40, 84 B. 84, 79, 56, 38, 40, 46

C. 84, 79, 56, 46, 40, 38 D. 84, 56, 79, 40, 46, 38

解释：从下往上非叶节点下沉调整

(12) 下述几种排序方法中，要求内存最大的是 (C) 。

A. 希尔排序 B. 快速排序 C. 归并排序 D. 堆排序

(13) 下述几种排序方法中，(C) 是稳定的排序方法。

A. 希尔排序 B. 快速排序 C. 归并排序 D. 堆排序

(14) 数据表中有10000个元素，如果仅要求求出其中最大的10个元素，则采用(D)算法最节省时间。

A. 冒泡排序 B. 快速排序 C. 简单选择排序 D. 堆排序

(15) 下列排序算法中，(A) 不能保证每趟排序至少能将一个元素放到其最终的位置上。

A. 希尔排序 B. 快速排序 C. 冒泡排序 D. 堆排序

20240618笔试 (Python)

一. 选择题 (30 分, 每小题 2 分)

1. 设栈的输入序列是1 2 3 4 5, 则 (B) 不可能是其出栈序列。
A. 23415 B. 54132 C. 23145 D. 15432
2. 对线性表进行二分法查找, 其前提条件是 (C)。
A. 线性表以链接方式存储, 并且按关键码值排好序
B. 线性表以链接方式存储, 并且按关键码值的检索频率排好序
C. 线性表以顺序方式存储, 并且按关键码值排好序
D. 线性表以顺序方式存储, 并且按关键码值的检索频率排好序
3. 对于二叉树中的一个结点N, 如果其左子树的高度为h1, 右子树的高度为h2, 则以N为根结点子树高度为 (C)。
A. $\max(h1, h2)$ B. $\min(h1, h2)$ C. $\max(h1, h2) + 1$ D. $\min(h1, h2) + 1$
4. 在插入排序算法中, 如果待排序的序列已经是有序的, 则插入排序算法的时间复杂度为 (C)。
A. $O(n^2)$ B. $O(n \log n)$ C. $O(n)$ D. $O(1)$
5. 下列哪个概念属于存储结构? (B)。
A. 线性表 B. 链表 C. 栈 D. 队列
6. 在二分查找算法中, 每次比较后都将搜索范围缩小一半, 若要在长度为n的数组中查找一个元素, 最坏情况下需要比较多少次 (D)。
A. n B. $\lfloor n/2 \rfloor$ C. $\lfloor \log n \rfloor$ D. $\lfloor \log n \rfloor + 1$
7. 在一个具有n个结点的有序单链表中插入一个新结点并仍保持其有序, 其平均时间复杂度为 (C)。
A. $O(n \log n)$ B. $O(1)$ C. $O(n)$ D. $O(n^2)$
8. 假设有4棵结点关键码为整数的二叉树, 若它们的中序遍历序列分别如下, 请问其中可能是二叉搜索树 (二叉排序树) 的是 (B)。
A. 5, 3, 1, 2, 4 B. 1, 2, 3, 4, 5 C. 5, 3, 4, 2, 1 D. 1, 3, 5, 4, 2
9. 假设线性表有 $2n$ ($n > 100$) 个元素, 以下操作 (D) 在单链表上实现比在顺序表上实现效率更高。
A. 在表中最后一个元素的后面插入一个新元素
B. 顺序输出表中的前i个元素
C. 交换表中第i个元素和第 $n+i$ 个元素的值 ($i=0, \dots, n-1$)
D. 删除表中第1个元素
10. 用数组 $Q[n]$ 实现循环队列, 设队头的前一个元素的下标为f, 队尾的下标为 r, 则该队列中元素总数是 (D)。
A. $r - f$ B. $(n + f - r) \% n$ C. $n + r - f$ D. $(n + r - f) \% n$
11. 对于一个拥有21条边的非连通无向图, 其至少包括 (D) 个顶点。
A. 5 B. 6 C. 7 D. 8
12. 若使用分治算法解决某一问题, 每次把规模为n的问题分解为2个规模为 $n/2$ 的子问题, 将两个子问题的结果合并的时间开销为 $O(n)$, 那么这个算法的总时间复杂度为 (B)。
A. $O(n)$ B. $O(n \log n)$ C. $O(n^2)$ D. $O(n^2 \log n)$

13. 有 n 个顶点 m 条边的无向图，下列说法中错误的是（D）。
- A 采用邻接表储存，储存该图的空间复杂度为 $O(m+n)$ 。
- B 若为稠密图，更倾向于采用邻接矩阵存储。
- C 采用邻接表储存，删除一个顶点的最坏情况的时间复杂度为 $O(n+m)$ 。
- D 若为稀疏图，深度优先周游的时间复杂度低于广度优先周游的时间复杂度。
14. 利用栈将表达式 $3*2^{(4+2*2-6*3)}-5$ （其中 $^$ 为乘幂）转换为后缀表达式过程中，当扫描到6 时，运算符栈为（D）。
- A. $*^{(+*-}$ B. $*^{*-}$ C. $*^{(+$ D. $*^{(-$
15. 定义一棵没有1度结点的二叉树为满二叉树。对于一棵包含 k 个结点的满二叉树，其叶子结点的个数为（C）。
- A. $\lfloor k/2 \rfloor$ B. $\lfloor k/2 \rfloor - 1$ C. $\lfloor k/2 \rfloor + 1$ D. 以上三个都有可能

二. 判断

(10分，每小题1分；对填写“Y”，错填写“N”)

- （Y）分治法和动态规划法都运用了将问题分解为规模较小的子问题的思想。
- （N）在链表中查找和删除一个元素的时间复杂度都是 $O(1)$ 。
- （N）相比于顺序存储，完全二叉树更适合用链式表示存储。
- （Y）通常不能通过一棵二叉树的前序遍历结点序列和后序遍历结点序列来确定这颗二叉树的完整结构。
- （N）二叉搜索树一定是满二叉树。
- （N）归并排序算法一定比简单插入排序算法的执行效率高。
- （Y）在待排序元素为正序情况下，直接插入排序可能比快速排序的时间复杂度更小。
- （Y）一个有 n 个结点的无向图，最少有一个连通分量。
- （N）图的遍历算法（如深度优先搜索DFS和广度优先搜索BFS）都只能用于无向图。
- （Y）若连通无向图的边的权值互不相同，其最小生成树只有一个。

三. 填空（20分，每题2分）

- 按照简单且高效的原则，如果经常需要在线性表头部进行插入和删除操作，最合适的存储结构是_____；如果需要随机存取，最合适的存储结构是_____。链表、顺序表
- 假设顺序表中包含6个数据元素{a, b, c, d, e, f}, 他们的查找概率分别为{0.12, 0.25, 0.23, 0.2, 0.05, 0.15}, 顺序查找时为了使查找成功的平均比较次数最少，则表中的数据元素的存放顺序应该是_____。b,c,d,f,a,e
- 已知某棵完全二叉树中有120个结点，则该二叉树的结点一共有_____层，有_____个叶子结点。7、60
- 已知一棵树的中序遍历序列为DBGEACF，后序遍历序列为DGEBFCA，则这棵树的前序遍历结果为_____。ABDEGCF

5. 某段电文中只有a,b,c,d四种字符，各种字符出现的次数为：a出现1000次，b出现2000次，c出现6000次，d出现1000次，采用哈夫曼编码该电文的长度为_____个比特。**2000*8**
6. 一组记录的关键字为45，80，55，40，42，85，利用堆排序的方法，从最后一个非叶子结点开始调整，建立的初始最大堆为_____。**85 80 55 40 42 45**
7. 设一棵m叉树中有 N_1 个度数为1的结点（度数表示子结点个数）， N_2 个度数为2的结点，……， N_m 个度数为m的结点，则该m叉树中共有_____个终端结点（即叶结点）。

$$1 + \sum_{i=2}^m (i - 1) N_i$$
8. 设散列表的表长m=15，散列函数H(key)=key%11，表中已有4个结点：addr(16)=5，addr(37)=4，addr(50)=6，addr(70)=7，如果用线性探查处理冲突，关键字为38的结点的地址是_____。**8**
9. 设森林F中有4棵树，第1、2、3、4棵树的结点个数分别为10、9、11、7，当把森林F转换成一棵二叉树后，其根结点的右子树中有_____个结点。**27**
10. 一个无向图，如果边的数量m_____结点个数n，那么该图一定存在回路。**>=**

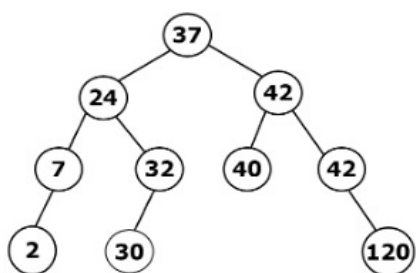
四. 简答（3题，共14分）

1. 对序列{H, E, B, L, G, A, F, J, I, C, D, K}中的关键码按字母序的升序重新排列，则：
 - a) 冒泡排序第一趟交换结束后的结果是？（1分）
 - b) 二路归并排序第一趟归并后的结果是？（1分）
 - c) 初始步长为4的希尔（shell）排序第一趟后的结果是？（2分）

答：前两问送分，第三问需要熟悉shell sort思想。

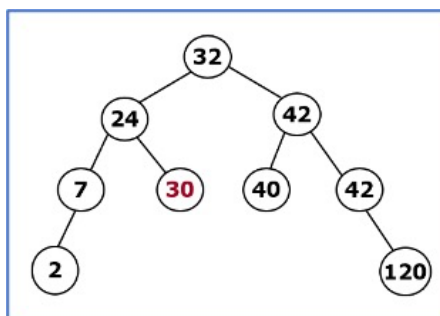
- 1、两个给出一个就对。从左往右冒泡排序的结果是 {E, B, H, G, A, F, J, I, C, D, K, L}；从右往左冒泡排序的结果是 {A, H, E, B, L, G, C, F, J, I, D, K}。；
- 2、二路归并排序一趟扫描的结果是 (E H B L A G F J C I D K)；
- 3、初始步长为4的希尔（shell）排序一趟的结果是 (G A B J H C D K I E F L)；

2. 在下列二叉排序树中删除结点“37”的基本过程是：首先寻找该结点左子树上的最大者r，并利用r辅助删除该结点。请画出删除该结点后的二叉排序树结构。（3分）



答案及解释：

基本答案如下图所示，但也可以有第二种方式



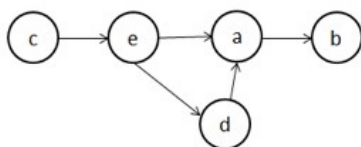
3. Dr. Stranger 的电脑感染了一种病毒。该病毒会将文档中的每种字母固定替换为另一种字母，且互不相同，可以看作文档所涉及字母集合上的双射函数。例如，文档 $D = \{ab, ac, bc\}$ ，涉及的字母集合为 $X = \{a, b, c\}$ ，病毒函数 $f: X \rightarrow X$ ，假设其中 $f(a) = b, f(b) = c, f(c) = a$ ，那么感染病毒后的文档 $D' = \{bc, ba, ca\}$ 。现在 Dr. Stranger 有一个重要文档 D，内容为 6 个按字典序排列的单词，涉及的字母集合为 $X = \{a, b, c, d, e\}$ 。该文档感染了病毒后，内容变为 $D' = \{cebdbac, cac, ecd, dca, aba, bac\}$ 。请你破解出该病毒规则 f ，并还原出原文档 D 的内容。给出思路（2 分）、具体步骤（3 分）、最终答案（2 分）。

答案及解释：

主要思路：就是推断病毒发作后文档D中的字母顺序，与字典序的字母顺序进行一一对应即可。每相邻的两个字符串进行比较；遇到不同的字母就建立边。

具体方法就是从相邻的两个字符串之间推断字母之间的顺序，如题目中单词cac在单词ecd之前，所以可以推断被替换后的字母顺序中字母c在字母e之前，将字母之间的顺序关系看成有向边，画出字母关系图，然后对图进行拓扑排序。得出的序列就是病毒发作后文档D中的字母顺序。

该题中可求得字母的顺序关系图如下：



通过拓扑排序，得到的病毒发作后文档D中的字母顺序{c, e, d, a, b}，而字典序为{a, b, c, d, e}，所以可以判断病毒将字母a替换为字母c，将字母b替换为字母e，将字母c替换为字母d，将字母d替换为字母a，将字母e替换为字母b。

最终文档D的原内容如下：{abeceda, ada, bac, cad, ded, eda}。

五. 算法填空（4题，共26分）

请注意：每个空最多填一条语句，不得使用分号连接多个表达式的写法。

1. (6分) 链表操作。

下面的程序描述的是一个只带表尾指针的循环单链表的操作(表尾指针指向链表最后一个结点, 最后一个结点的next指针指向链表的第一个结点)。先是将n个整数依次插入链表头部, 然后删掉链表尾部m个元素, 再往链表前部插入k个整数。请填空。

输入:

第1行是整数n($0 < n \leq 100$)

第2行是n个非负整数

第3行是整数m($0 < m \leq 200$)

第4行是整数k($0 < k \leq 100$)

第5行是k个非负整数

输出:

第1行: 将链表中的元素依次输出 (结果会是输入第2行的倒序)

第2行: 将链表删除尾部m个元素后的结果输出。如果链表为空, 输出 NULL

第3行: 依次输出链表前部又插入输入中的第5行的k个整数后的结果

样例输入:

4
1 2 3 4
3
2
100 200

样例输出:

4 3 2 1
4
200 100 4

```
1 class Node:
2     def __init__(self, data, next=None):
3         self.data, self.next = data, next
4
5 class LinkedList:
6     def __init__(self):
7         self.tail = None # 表尾指针, 指向链表最后一个结点
8     def pushFront(self, data): # 在链表头部插入元素
9         nd = Node(data)
10        if self.tail is None:
11            self.tail = nd
12            nd.next = nd
13        else:
14            _____(1)_____ # 2分
15            self.tail.next = nd
16    def popBack(self): # 在链表尾部删除元素
17        if self.tail is not None:
18            if _____(2)_____: # 1分
19                self.tail = None
20        else:
```

```

21         ptr = self.tail.next
22         while _____(3)_____: # 1分
23             ptr = ptr.next
24             ptr.next = ptr.next.next
25             _____(4)_____ # 1分
26     def print(self):
27         if self.tail is not None:
28             ptr = self.tail.next
29             print(ptr.data, end=" ")
30             ptr = ptr.next
31             while _____(5)_____: # 1分
32                 print(ptr.data, end=" ")
33                 ptr = ptr.next
34         else:
35             print("NULL")
36         print()
37
38     n = input()
39     a = list(map(int, input().split()))
40     Lst = LinkedList()
41     for x in a:
42         Lst.pushFront(x)
43     Lst.print()
44     n = int(input())
45     for i in range(n):
46         Lst.popBack()
47     Lst.print()
48     n = input()
49     b = list(map(int, input().split()))
50     for x in b:
51         Lst.pushFront(x)
52     Lst.print()
53

```

答:

- (1) nd.next = self.tail.next # 2分
- (2) self.tail.next is self.tail: # 1分
- (3) ptr.next != self.tail: # 1分
- (4) self.tail = ptr # 1分
- (5) ptr != self.tail.next: # 1分

2. (6分) 求二叉树的宽度。

给定一棵二叉树，求该二叉树的宽度。二叉树宽度定义：结点最多的那一层的结点数目。

输入：第一行是一个整数n，表示二叉树的结点个数。二叉树结点编号从0到n-1， $n \leq 100$ 。接下来有n行，依次对应二叉树的编号为0,1,2,...n-1的结点。

每行有两个整数，分别表示该结点的左儿子和右儿子的编号。如果第一个（第二个）数为-1则表示没有左（右）儿子

输出：输出1个整数，表示二叉树的宽度

样例输入

```
3
-1 -1
0 2
-1 -1
```

样例输出

```
2
```

```
1 class BinaryTree:
2     def __init__(self, data, left=None, right=None):
3         self.data, self.left, self.right = data, left, right
4     def countWidth(self):
5         width = [0 for i in range(200)] # width[i]记录第i层宽度
6         def traversal(root, level):
7             if root is None:
8                 return
9             width[level] += 1
10            _____(1)_____ # 1分
11            _____(2)_____ # 1分
12            traversal(self, 0)
13            return max(width)
14 def buildTree():
15     n = int(input())
16     nodes = [BinaryTree(None) for i in range(n)]
17     for nd in range(n):
18         L, R = map(int, input().split())
19         if L != -1:
20             nodes[nd].left = nodes[L]
21             _____(3)_____ # 1分
22         if R != -1:
23             nodes[nd].right = nodes[R]
24             _____(4)_____ # 1分
25     for i in range(n):
26         if _____(5)_____ : # 2分
27             return nodes[i]
28     return None
29 tree = buildTree()
30 print(tree.countWidth())
```

答:

- (1) traversal(root.left, level + 1) # 1分
- (2) traversal(root.right, level + 1) # 1分
- (3) nodes[L].data = L # 1分
- (4) nodes[R].data = R # 1分
- (5) nodes[i].data is None: # 2分

3. (7分) 用Prim算法求无向图最小生成树。

输入：第一行两个整数n,m (n< 100)，表示图有n个结点， m条边。结点编号从0开始算。接下来有m行，每行三个数s e w, 表示边(s,e)的权值是w(w<1000)。

输出：输出最小生成树的所有边

输入样例：

```
4 4
0 1 8.5
1 2 4
2 3 9
3 1 7
```

输出样例：

```
(1,0)(2,1)(3,1)
```

```
1 def prim(G): # Prim算法求图G最小生成树，返回最小生成树的边的列表
2     INF = 1 << 30 # 无穷大
3     # G是邻接矩阵,矩阵中None表示没有边，顶点编号从0开始
4     n = len(G) # n是顶点数目，顶点编号0 - (n-1)
5     dist = [INF for i in range(n)] # 各顶点到已经建好的那部分树的距离
6     used = [False for i in range(n)] # 标记顶点是否已经被加入最小生成树
7     prev = [None for i in range(n)]
8     # 顶点i是通过边(i,prev[i])被加入最小生成树的
9     doneNum = 0 # 已经被加入最小生成树的顶点数目
10    edges = [] # 最小生成树的边的列表
11    while _____(1)_____: # 1分
12        if doneNum == 0:
13            x = minDist = 0 # 顶点0最先被加入最小生成树
14        else:
15            x, minDist = None, INF
16            for i in range(n):
17                if not used[i] and _____(2)_____: # 1分
18                    x, minDist = i, dist[i]
19            _____(3)_____ # 1分
```

```

20         doneNum += 1
21         if doneNum > 1:
22             edges.append(_____(4)_____) # 1分
23         for v in range(n):
24             if not used[v] and _____(5)_____: #2分
25                 dist[v] = G[x][v]
26                 _____(6)_____ # v是通过x连接到最小生成树的 #1分
27         return edges
28
29
30 n, m = map(int, input().split())
31 G = [[None for i in range(n)] for j in range(n)] # 邻接矩阵
32 for i in range(m):
33     tmp = input().split()
34     s, e, w = int(tmp[0]), int(tmp[1]), float(tmp[2])
35     G[s][e] = G[e][s] = w
36 edges = prim(G) # edges的元素是一个元组(u,v), 表示一条边
37 for e in edges:
38     print(f"({e[0]}, {e[1]})", end=" ")
39
40

```

答:

- (1) doneNum < n: # 1分
- (2) dist[i] < minDist: # 1分
- (3) used[x] = True # 1分
- (4) (x, prev[x]) # 1分
- (5) `G[x][v] is not None and G[x][v] < dist[v]`: #2分
- (6) prev[v] = x # v是通过x连接到最小生成树的 #1分

4. (7分) 完成下列算法，计算一个无向图中所有连通分量的结点个数。

输入：图的结点数、边数和边的列表。

输出：每个连通分量的结点个数。

输入样例：

5

[(0, 1), (1, 2), (3, 4)]

输出样例：

[3, 2]

```

1 def dfs(node, visited, graph, n):
2     count = 1

```

```

3     visited[node] = True
4     for i in range(n):
5         if i in graph[node] and ____ (1) ____: # 1分
6             count += ____ (2)
7             ____ # 1分
8     return count
9
10
11 def count_components(n, edges):
12     graph = {i: [] for i in range(n)}
13     for u, v in edges:
14         ____ (3) ____ # 1分
15         ____ (4) ____ # 1分
16
17     visited = [False] * n
18     components = []
19     for i in range(n):
20         if ____ (5) ____: # 1分
21             components.append(____ (6) ____ ) # 2分
22
23     return components
24
25     n = int(input())
26     edges = eval(input())
27     print(count_components(n, edges))
28

```

答:

- (1) not visited[i]: # 1分
- (2) dfs(i, visited, graph, n) # 2分
- (3) graph[u].append(v) # 1分
- (4) graph[v].append(u) # 1分
- (5) not visited[i]: # 1分
- (6) dfs(i, visited, graph, n) # 2分

20230620笔试 (Python)

一. 选择题 (30 分, 每小题 2 分)

1. 下列叙述中正确的是 (D)。
A: 散列是一种基于索引的逻辑结构
B: 基于顺序表实现的逻辑结构属于线性结构
C: 数据结构设计影响算法效率, 逻辑结构起到了决定作用
D: 一个逻辑结构可以有多种类型的存储结构, 且不同类型的存储结构会直接影响到数据处理效率
2. 在广度优先搜索算法中, 一般使用什么辅助数据结构? (A)。
A: 队列 B: 栈
C: 树 D: 散列
3. 若某线性表采取链式存储, 那么该线性表中结点的存储地址 (B)。
A: 一定不连续 B: 既可连续亦可不连续 C: 一定连续 D: 与头结点存储地址保持连续
4. 若某线性表常用操作是在表尾插入或删除元素, 则时间开销最小的存储方式是 (C)。
A: 单链表 B: 仅有头指针的单循环链表
C: 顺序表 D: 仅有尾指针的单循环链表
5. 给定后缀表达式 (逆波兰式) $ab+-c*d$ 对应的中缀表达式是 (B)。
A: $a-b-c*d$ B: $-(a+b)*c-d$
C: $a+b*c-d$ D: $(a+b)*(-c-d)$
6. 今有一空栈 S, 基于待进栈的数据元素序列 a, b, c, d, e, f 依次进行进栈、进栈、出栈、进栈、进栈、出栈的操作, 上述操作完成以后, 栈 S 的栈顶元素为 (B)。
A: f B: c C: a D: d
7. 对于单链表, 表头节点为 head, 判定空表的条件是 (D)。
A: `head.next == None`
B: `head != None`
C: `head.next == head`
D: `head == None`
8. 以下典型排序算法中, 具有稳定排序特性的是 (A)。
A: 冒泡排序 (Bubble Sort) B: 直接选择排序 (Selection Sort)
C: 快速排序 (Quick Sort) D: 希尔排序 (Shell Sort)
9. 以下关键字列表中, 可以有效构成一个大根堆 (即最大值二叉堆, 最大值在堆顶) 的序列是 (D)。
A: 5 8 1 3 9 6 2 7 B: 9 8 1 7 5 6 2 33
C: 9 8 6 3 5 1 2 7 D: 9 8 6 7 5 1 2 3
10. 以下典型排序算法中, 内存开销最大的是 (C)。
A: 冒泡排序 B: 快速排序
C: 归并排序 D: 堆排序
11. 排序算法依赖于对元素序列的多趟比较/移动操作 (即执行多轮循环), 第一趟结束后, 任一元素都无法确定其最终排序位置的算法是 (D)。
A: 选择排序 B: 快速排序 C: 冒泡排序 D: 插入排序

12. 考察以下基于单链表的操作，相较于顺序表实现，带来更高时间复杂度的操作是（D）。
- A: 合并两个有序线性表，并保持合成后的线性表依然有序
 - B: 交换第一个元素与第二个元素的值
 - C: 查找某一元素值是否在线性表中出现
 - D: 输出第 i 个 ($0 \leq i < n$, n 为元素个数) 元素
13. 已知一个整型数组序列，序列元素值依次为 (19, 20, 50, 61, 73, 85, 11, 39)，采用某种排序算法，在多趟比较/移动操作（即执行多轮循环）后，依次得到以下中间结果（每一行对应一趟）如下：
- (1) 19 20 11 39 73 85 50 61
 - (2) 11 20 19 39 50 61 73 85
 - (3) 11 19 20 39 50 61 73 85
- 请问，上述过程使用的排序算法是（C）。
- A: 冒泡排序
 - B: 插入排序
 - C: 希尔排序
 - D: 归并排序
14. 今有一非连通无向图，共有 36 条边，该图至少有（C）个顶点。
- A: 8 B: 9 C: 10 D: 11
15. 令 $G=(V, E)$ 是一个无向图，若 G 中任何两个顶点之间均存在唯一的简单路径相连，则下面说法中错误的是（A）。
- A: 图 G 中添加任何一条边，不一定造成图包含一个环
 - B: 图 G 中移除任意一条边得到的图均不连通
 - C: 图 G 的逻辑结构实际上退化为树结构
 - D: 图 G 中边的数目一定等于顶点数目减 1

二. 判断

(10 分，每小题 1 分；对填写“Y”，错填写“N”)

1. (Y) 按照前序、中序、后序方式周游一棵二叉树，分别得到不同的结点周游序列，然而三种不同的周游序列中，叶子结点都将以相同的顺序出现。
2. (N) 构建一个含 N 个结点的（二叉）最小值堆，时间效率最优情况下的时间复杂度大 O 表示为 $O(N \log N)$ 。
3. (N) 对任意一个连通的无向图，如果存在一个环，且这个环中的一条边的权值不小于该环中任意一个其它的边的权值，那么这条边一定不会是该无向图的最小生成树中的边。
4. (Y) 通过树的周游可以求得树的高度，若采取深度优先遍历方式设计求解树高度问题的算法，算法空间复杂度大 O 表示为 $O(\text{树的高度})$ 。
5. (Y) 树可以等价转化二叉树，树的先序遍历序列与其相应的二叉树的前序遍历序列相同。
6. (Y) 如果一个连通无向图 G 中所有边的权值均不同，则 G 具有唯一的最小生成树。
7. (Y) 求解最小生成树问题的 Prim 算法是一种贪心算法。
8. (Y) 使用线性探测法处理散列表碰撞问题，若表中仍有空槽（空单元），插入操作一定成功。

9. (N) 从链表中删除某个指定值的结点, 其时间复杂度是 $O(1)$ 。
10. (Y) Dijkstra 算法的局限性是无法正确求解带有负权值边的图的最短路径。

三. 填空 (20 分, 每题 2 分)

1. 定义二叉树中一个结点的度数为其子结点的个数。现有一棵结点总数为 101 的二叉树, 其中度数为 1 的结点数有 30 个, 则度数为 0 结点有 ____ 个。 **36**
2. 定义完全二叉树的根结点所在层为第一层。如果一个完全二叉树的第六层有 23 个叶结点, 则它的总结点数可能为 ____ (请填写所有 3 个可能的结点数, 写对 1 个得 1 分, 2 个得 1.5 分, 写错 1 个不得分)。 **54,80,81**
3. 对于初始排序码序列 (51, 41, 31, 21, 61, 71, 81, 11, 91), 用双指针原地交换实现, 第 1 趟快速排序 (以第一个数字为中值) 的结果是: ____。 **11 41 31 21 51 71 81 61 91**
4. 如果输入序列是已经正序, 在(改进)冒泡排序、直接插入排序和直接选择排序算法中, ____ 算法最慢结束。 **直接选择排序**
5. 已知某二叉树的先根周游序列为 (A, B, D, E, C, F, G), 中根周游序列为 (D, B, E, A, C, G, F), 则该二叉树的后根次序周游序列(____)。 **D,E,B,G,F,C,A**
6. 使用栈计算后缀表达式 (操作数均为一位数) "1 2 3 + 4 * 5 + 3 + -", 当扫描到第二个 + 号但还未对该 + 号进行运算时, 栈的内容 (以栈底到栈顶从左往右的顺序书写) 为 ____。 **1,20,5**
7. 51 个顶点的连通图 G 有 50 条边, 其中权值为 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 的边各 5 条, 则连通图 G 的最小生成树各边的权值之和为 ____。 **275**
8. 包含 n 个顶点无向图的邻接表存储结构中, 所有顶点的边表中最多有 ____ 个结点。具有 n 个顶点的有向图, 顶点入度和出度之和最大值不超过 ____。 **n(n-1), 2(n-1)**
9. 给定一个长度为 7 的空散列表, 采用双散列法解决冲突, 两个散列函数分别为: $h_1(\text{key}) = \text{key} \% 7$, $h_2(\text{key}) = \text{key} \% 5 + 1$ 请向散列表依次插入关键字为 30, 58, 65 的集合元素, 插入完成后 65 在散列表中存储地址为 ____。 **3**
10. 阅读算法 ABC, 回答问题。

```
1 def ABC(n):
2     k, m = 2, int(n**0.5)
3     while (k <= m) and (n % k != 0):
4         k += 1
5     return k > m
```

- 1) 算法的功能是: ____。 **素数判断**
- 2) 算法的时间复杂度是 $O(\text{____})$ 。 **$n^{**0.5}$**

四. 简答 (3题, 共14分)

1. (4 分) 字符串匹配算法从长度为 n 的文本串 S 中查找长度为 m 的模式串 P 的首次出现。

a) 字符串匹配的朴素算法使用暴力搜索，大致过程如下：对于 P 在 S 中可能出现的 $n-m+1$ 个位置，比对此位置时 P 和 S 中对应子串是否相等。其时间复杂度 $O((n-m+1)m)$ 。请举例说明算法时间复杂度一种最坏情况（注：例子中请只出现 a 和 b 两种字符）。（1分）

b) 已知字符串 S 为“abaabaabaabcc”，模式串 t 为“abaabc”。采用朴素算法进行查找，请写出字符比对的总次数和查找结果。（2 分）

c) 朴素算法存在很大的改进空间，说明在上述(b)步骤中，第一次出现不匹配 ($s[i+j] \neq t[j]$) 时 ($i=0, j=5$)，为了避免冗余比对，则下次比对时， i 和 j 的值可以分别调整为多少？（1分）

字符串匹配的朴素算法：

```
1 def issubstring(s, t):
2     for i in range(len(s)):
3         for j in range(len(t)):
4             if s[i + j] != t[j]:
5                 break
6         else:
7             return True
8     return False
```

答：

问题 a) 如 $P = a^{m-1}b$ ， $S = a^{n-1}b$ ，执行朴素算法刚好需要 $(n-m+1)m$ 次比对。

问题 b) 需要 24 次比对得到 True 结果

问题 c) $i=3, j=2$

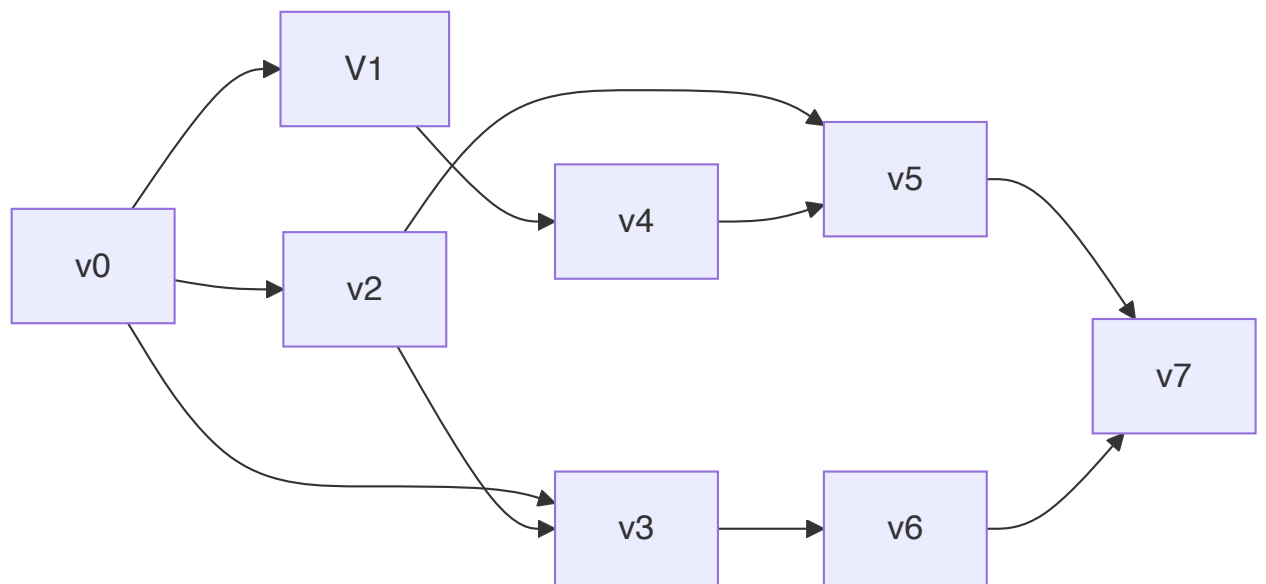
2. (5 分) 有八项活动，每项活动标记为 V +编号 $n(0 \leq n \leq 7)$ ，每项活动要求的前驱如下：

活动	V0	V1	V2	V3	V4	V5	V6	V7
前驱	无前驱	V0	V0	V0, V2	V1	V2, V4	V3	V5, V6

- (1) 画出相应的 AOV (Active On Vertex) 网络（即节点为活动，边为先后关系的有向图），
- (2) 并给出一个拓扑排序序列，如存在多种，则按照编号从小到大排序，输出最小的一种。

答：

(1) AOV网络



(2) 其中一个序列为：V0,V1,V2,V3,V4,V5,V6,V7

3. (5 分) 简要回答下列 BST 树以及 BST 树更新过程的相关问题。

(1) 请简述什么是二叉查找树 (BST) (1 分)

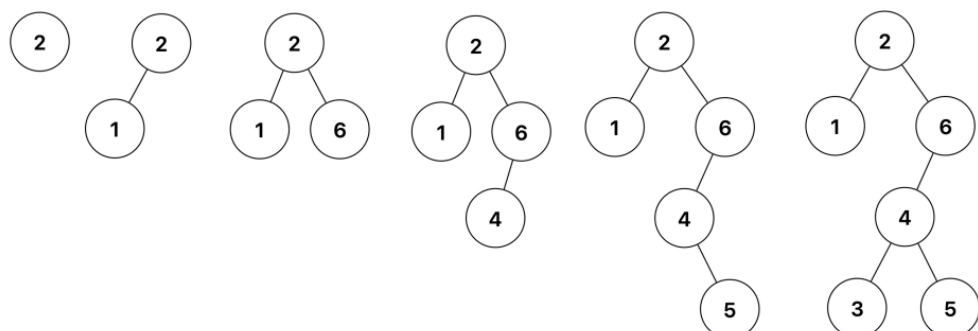
(2) 请图示 2,1,6,4,5,3 按顺序插入一棵 BST 树的中间过程和最终形态 (2 分)

(3) 请图示以上 BST 树，依次删除节点 4 和 2 的过程和树的形态 (2 分)

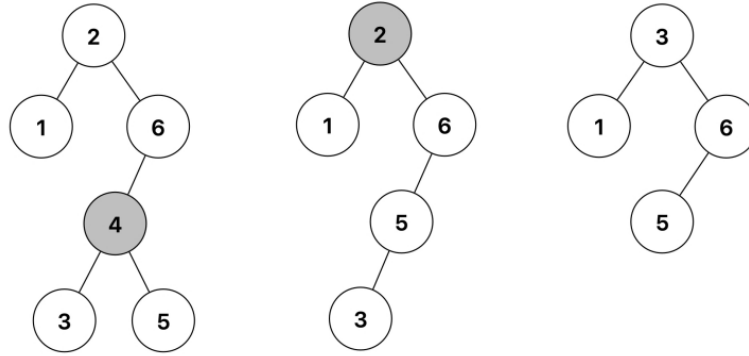
答：

(1) BST 树是二叉树，对于所有的子树而言，其左子树上所有关键值都小于根，右子树上的所有关键值都大于根

(2)



(3)



五. 算法填空（4题，共26分）

1. （6 分）拓扑排序：给定一个有向图，求拓扑排序序列。

输入：第一行是整数 n ，表示图有 n 顶点 ($1 \leq n \leq 100$)，编号 1 到 n 。接下来 n 行，第 i 行列了顶点 i 的所有邻点，以 0 结尾。没有邻点的顶点，对应行就是单独一个 0。

输出：一个图的拓扑排序序列。如果图中有环，则输出“Loop”。

样例输入 (#及其右边的文字是说明，不是输入的一部分)：

```

1 5          #5 个顶点
2 0          #1 号顶点无邻点
3 4 5 1 0    #2 号顶点有邻点 4 5 1
4 1 0
5 5 3 0
6 3 0

```

样例输出

```

1 2 4 5 3 1

```

请对下面的解题程序进行填空

```

1 class Edge: # 表示邻接表中的图的边,v 是终点
2     def __init__(self, v):
3         self.v = v
4
5
6 def topoSort(G):    # G 是邻接表, 顶点从 0 开始编号
7     # G[i][j]是 Edge 对象, 代表边 <i, G[i][j].v>
8     n = len(G)
9     import queue
10    inDegree = [0] * n # inDegree[i]是顶点 i 的入度
11    q = queue.Queue()
12    # q 是队列, q.put(x)可以将 x 加入队列, q.get()取走并返回对头元素
13    # q.empty()返回队列是否为空

```

```

14
15     for i in range(n):
16         for e in G[i]:
17             inDegree[e.v] += 1 # 【1 分】
18
19     for i in range(n):
20         if inDegree[i] == 0:
21             q.put(i) # 【1 分】
22
23     seq = []
24     while not q.empty():
25         k = q.get()
26         seq.append(k) # 【1 分】
27         for e in G[k]:
28             inDegree[e.v] -= 1 # 【1 分】
29             if inDegree[e.v] == 0:
30                 q.put(e.v) # 【1 分】
31
32     if len(seq) != n: # 【1 分】
33         return None
34     else:
35         return seq
36
37
38 n = int(input())
39 G = [[] for _ in range(n)] # 邻接表
40 for i in range(n):
41     lst = list(map(int, input().split()))
42     print(lst)
43     G[i] = [Edge(x - 1) for x in lst[:-1]]
44     print(G[i])
45
46 result = topoSort(G)
47 if result is not None:
48     for x in result:
49         print(x + 1, end=" ")
50 else:
51     print("Loop")
52

```

2. (7 分) 链表操作：读入一个从小到大排好序的整数序列到链表，然后在链表中删除重复的元素，使得重复的元素只保留 1 个，然后将整个链表内容输出。

输入样例：

```
1 | 1 2 2 2 3 3 4 4 6
```

输出样例：

```
1 | 1 2 3 4 6
```

请对程序填空:

```
1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5
6 a = list(map(int, input().split()))
7 head = Node(a[0])
8 p = head
9 for x in a[1:]:
10     p.next = Node(x)    # 【2 分】
11     p = p.next
12
13 p = head
14 while p:
15     while p.next and p.data == p.next.data: # 【2 分】
16         p.next = p.next.next    # 【1 分】
17     p = p.next
18
19 p = head
20 while p:
21     print(p.data, end=" ")
22     p = p.next    # 【2 分】
23
```

3. (7 分) 无向图判定: 给定一个无向图, 判断是否连通, 是否有回路。

输入: 第一行两个整数 n, m , 分别表示顶点数和边数。顶点编号从 0 到 $n-1$ 。($1 \leq n \leq 110, 1 \leq m \leq 10000$) 接下来 m 行, 每行两个整数 u 和 v , 表示顶点 u 和 v 之间有边。

输出:

如果图是连通的, 则在第一行输出“connected:yes”, 否则第一行输出“connected:no”。

如果图中有回路, 则在第二行输出“loop:yes”, 否则第二行输出“loop:no”。

样例输入

```
1 | 3 2
2 | 0 1
3 | 0 2
```

样例输出

```
1 | connected:yes
2 | loop:no
```

请进行程序填空：

```
1 def isConnected(G): # G 是邻接表,顶点编号从 0 开始,判断是否连通
2     n = len(G)
3     visited = [False for _ in range(n)]
4     total = 0
5
6     def dfs(v):
7         nonlocal total
8         visited[v] = True
9         total += 1
10        for u in G[v]:
11            if not visited[u]:
12                dfs(u)
13
14    dfs(0)
15    return total == n      # 【2 分】
16
17 def hasLoop(G): # G 是邻接表,顶点编号从 0 开始,判断有无回路
18     n = len(G)
19     visited = [False for _ in range(n)]
20
21     def dfs(v, x): # 返回值表示本次 dfs 是否找到回路,x 是深度优先搜索树上 v 的父结点
22         visited[v] = True
23         for u in G[v]:
24             if visited[u] == True:
25                 if u != x: # 【2 分】
26                     return True
27             else:
28                 if dfs(u, v): # 【2 分】
29                     return True
30         return False
31
32     for i in range(n):
33         if not visited[i]: # 【1 分】
34             if dfs(i, -1):
35                 return True
36     return False
37
38 n, m = map(int, input().split())
39 G = [[] for _ in range(n)]
40 for _ in range(m):
41     u, v = map(int, input().split())
42     G[u].append(v)
43     G[v].append(u)
44
45 if isConnected(G):
46     print("connected:yes")
47 else:
48     print("connected:no")
```

```

49
50 if hasLoop(G):
51     print("loop:yes")
52 else:
53     print("loop:no")
54

```

4. (6 分) 堆排序：输入若干个整数，下面的程序使用堆排序算法对这些整数从小到大排序，请填空。
程序中建立的堆是大顶堆（最大元素在堆顶）

输入样例：

```

1 | 1 3 43 8 7

```

输出样例：

```

1 | 1 3 7 8 43

```

请进行程序填空：

```

1 def heap_sort(arr):
2     heap_size = len(arr)
3
4     def goDown(i):
5         if i * 2 + 1 >= heap_size: # a[i]没有儿子
6             return
7         L, R = i * 2 + 1, i * 2 + 2
8
9         if R >= heap_size or arr[L] > arr[R]: # 【1 分】
10             s = L
11         else:
12             s = R
13
14         if arr[s] > arr[i]:
15             arr[s], arr[i] = arr[i], arr[s] # 【2 分】
16             goDown(s)
17
18     def heapify(): # 将列表 a 变成一个堆
19         for k in range(len(arr) // 2 - 1, -1, -1): # 【1 分】
20             goDown(k)
21
22     heapify()
23     for i in range(len(arr) - 1, -1, -1):
24         arr[0], arr[i] = arr[i], arr[0] # 【1 分】
25         heap_size -= 1

```



```

26         goDown(0)    # 【1 分】
27
28
29 a = list(map(int, input().split()))
30 heap_sort(a)
31 for x in a:
32     print(x, end=" ")
33

```

卷面写法怪异，正常写法应该是

```

1  def heapify(arr, n, i):
2      largest = i # 将当前节点标记为最大值
3      left = 2 * i + 1 # 左子节点的索引
4      right = 2 * i + 2 # 右子节点的索引
5
6      # 如果左子节点存在且大于根节点，则更新最大值索引
7      if left < n and arr[i] < arr[left]:
8          largest = left
9
10     # 如果右子节点存在且大于根节点或左子节点，则更新最大值索引
11     if right < n and arr[largest] < arr[right]:
12         largest = right
13
14     # 如果最大值索引发生了变化，则交换根节点和最大值，并递归地堆化受影响的子树
15     if largest != i:
16         arr[i], arr[largest] = arr[largest], arr[i]
17         heapify(arr, n, largest)
18
19
20 def buildMaxHeap(arr):
21     n = len(arr)
22
23     # 从最后一个非叶子节点开始进行堆化
24     for i in range(n // 2 - 1, -1, -1):
25         heapify(arr, n, i)
26
27
28 def heapSort(arr):
29     n = len(arr)
30
31     buildMaxHeap(arr) # 构建大顶堆
32
33     # 逐步取出堆顶元素（最大值），并进行堆化调整
34     for i in range(n - 1, 0, -1):
35         arr[i], arr[0] = arr[0], arr[i] # 交换堆顶元素和当前最后一个元素
36         heapify(arr, i, 0) # 对剩余的元素进行堆化
37
38     return arr
39

```

```
40 a = list(map(int, input().split()))
41 heapSort(a)
42 for x in a:
43     print(x, end=" ")
```

20220621笔试 (Python)

一. 选择题 (30 分, 每小题 2 分)

1. 双向链表中的每个结点有两个引用域, prev 和 next, 分别引用当前结点的前驱与后继, 设 p 引用链表中的一个结点, q 引用一待插入结点, 现要求在 p 前插入 q, 则正确的插入操作为 (D)。

A: p.prev=q; q.next=p; p.prev.next=q; q.prev=p.prev;
B: q.prev=p.prev; p.prev.next=q; q.next=p; p.prev=q.next;
C: q.next=p; p.next=q; p.prev.next=q; q.next=p;
D: p.prev.next=q; q.next=p; q.prev=p.prev; p.prev=q.

#假设链表是 $A \leftrightarrow B \leftrightarrow C$, 要在 B 前插入 Q, 那么会得到 $A \leftrightarrow Q \leftrightarrow B \leftrightarrow C$ 的链表。这是在结点 B 前插入 Q 结点的步骤:

1. `p.prev.next = q;` 这一步是把 p 的前一个结点的 next 指针指向 q。在例子中, 是把 A 的 next 指针指向 Q。
 2. `q.next = p;` 这一步是把 q 的 next 指针指向 p。在例子中, 是把 Q 的 next 指针指向 B。
 3. `q.prev = p.prev;` 这一步是把 q 的 prev 指针指向 p 的前一个结点。在例子中, 是把 Q 的 prev 指针指向 A。
 4. `p.prev = q;` 这一步是把 p 的 prev 指针指向 q。在例子中, 是把 B 的 prev 指针指向 Q。
2. 给定一个 N 个相异元素构成的有序数列, 设计一个递归算法实现数列的二分查找, 考察递归过程中栈的使用情况, 请问这样一个递归调用栈的最小容量应为 (D)。
- A: N B: N/2 C: $\lceil \log_2(N) \rceil$ **D: $\lceil \log_2(N + 1) \rceil$**

#当N=1时, 函数调用用到1次栈。

#当N=2时, 需要两个帧: 二分查找, 只能往一侧走, 所以是函数调用+递归1次, 等于2。

#当N=3是, 因为是二分查找, 中间节点已经比完不用考虑, 情况同N=2。

#当N=4时, 第一次递归二分查找, 剩下 1 2 4; 按照最坏情况考虑, 1 2 还需要2次。结果是3。

3. 数据结构有三个基本要素:逻辑结构、存储结构以及基于结构定义的行为(运算)。下列概念中(B)属于存储结构。

A:线性表 **B:链表** C:字符串 D:二叉树

#在这些选项中, 有些描述的是数据的逻辑结构, 而有些是存储结构。逻辑结构指的是数据对象中数据元素之间的相互关系, 而存储结构是指数据结构在计算机中的表示 (也就是内存中的存储形式)。

A: **线性表** - 这是一种逻辑结构, 它描述元素按线性顺序排列的规则。

B: **链表** - 这是一种存储结构, 它是线性表的链式存储方式, 通过节点的相互链接来实现。

正确答案是 B: 链表, 因为它指的是数据的物理存储方式, 即内存中的链式存储结构。

4. 为了实现一个循环队列（或称环形队列），采用数组 $Q[0..m-1]$ 作为存储结构，其中变量 $rear$ 表示这个循环队列中队尾元素的实际位置，添加结点时按 $rear=(rear+1) \% m$ 进行指针移动，变量 $length$ 表示当前队列中的元素个数，请问这个循环队列的队列首位元素的实际位置是（ B ）。
- A: $rear-length$ **B: $(1+rear+m-length) \% m$** C: $(rear-length+m) \% m$ D: $m-length$

$length = rear - head + 1$ ，再对环形队列的特点做调整，得到B。

5. 给定一个二叉树，若前序遍历序列与中序遍历序列相同，则二叉树是（ D ）。
- A: 根结点无左子树的二叉树
B: 根结点无右子树的二叉树
C: 只有根结点的二叉树或非叶子结点只有左子树的二叉树
D: 只有根结点的二叉树或非叶子结点只有右子树的二叉树

#因为在前序遍历中，根节点总是首先访问的，而在中序遍历中，根节点必然在中间。

6. 用 Huffman 算法构造一个最优二叉编码树，待编码的字符权值分别为{3, 4, 5, 6, 8, 9, 11, 12}，请问该最优二叉编码树的带权外部路径长度为（ B ）。（补充说明：树的带权外部路径长度定义为树中所有叶子结点的带权路径长度之和；其中，结点的带权路径长度定义为该结点到树根之间的路径长度与该结点权值的乘积）（ B ）
- A: 58 **B: 169** C: 72 D: 18

#为了构造哈夫曼树，我们遵循一个重复的选择过程，每次选择两个最小的权值创建一个新的节点，直到只剩下一个节点为止。我们可以按照以下步骤操作：

1. 将给定的权值排序：{3, 4, 5, 6, 8, 9, 11, 12}。
2. 选择两个最小的权值：3 和 4，将它们组合成一个新的权值为 7 的节点。
现在权值变为：{5, 6, 7, 8, 9, 11, 12}。
3. 再次选择两个最小的权值：5 和 6，将它们组合成一个新的权值为 11 的节点。
现在权值变为：{7, 8, 9, 11, 11, 12}。
4. 选择两个最小的权值：7 和 8，将它们组合成一个新的权值为 15 的节点。
现在权值变为：{9, 11, 11, 12, 15}。
5. 选择两个最小的权值：9 和 11，将它们合并成一个新的权值为 20 的节点。
现在权值变为：{11, 12, 15, 20}。
6. 选择两个最小的权值：11 和 12，合并成一个新的权值为 23 的节点。
现在权值变为：{15, 20, 23}。
7. 选择两个最小的权值：15 和 20，合并成一个新的权值为 35 的节点。
现在权值变为：{23, 35}。
8. 最后，合并这两个节点得到根节点，权值为 $23 + 35 = 58$ 。

现在我们可以计算哈夫曼树的带权外部路径长度（WPL）。



现在让我们计算每个叶子节点的带权路径长度：

- 权值 3 的节点路径长度为 4，WPL 部分为 $3 * 4 = 12$ 。
- 权值 4 的节点路径长度为 4，WPL 部分为 $4 * 4 = 16$ 。
- 权值 5 的节点路径长度为 4，WPL 部分为 $5 * 4 = 20$ 。
- 权值 6 的节点路径长度为 4，WPL 部分为 $6 * 4 = 24$ 。
- 权值 9 的节点路径长度为 3，WPL 部分为 $9 * 3 = 27$ 。
- 权值 8 的节点路径长度为 3，WPL 部分为 $8 * 3 = 24$ 。
- 权值 11 的节点路径长度为 2，WPL 部分为 $11 * 2 = 22$ 。
- 权值 12 的节点路径长度为 2，WPL 部分为 $12 * 2 = 24$ 。

将所有部分的 WPL 相加，我们得到整棵哈夫曼树的 WPL：

$$WPL = 12 + 16 + 20 + 24 + 27 + 24 + 22 + 24 = 169$$

7. 假设需要对存储开销 1GB (GigaBytes) 的数据进行排序，但主存储器（RAM）当前可用的存储空间只有 100MB (MegaBytes)。针对这种情况，（ B ）排序算法是最适合的。
 A: 堆排序 **B: 归并排序** C: 快速排序 D: 插入排序

#对于这种情况，最适合的排序算法是归并排序（B）。

归并排序是一种外部排序算法，它的主要思想是将数据分成较小的块，然后逐步合并这些块以获得有序的结果。由于主存储器的可用空间有限，归并排序非常适合这种情况，因为它可以在有限的主存中进行部分排序，并将排序好的部分写入外部存储（磁盘）中。然后再将不同部分进行合并，直到得到完全排序的结果。

堆排序（A）通常需要对整个数据集进行排序，因此不适合主存储器有限的情况。

快速排序（C）通常是一种原地排序算法，它需要频繁地交换数据，这可能导致频繁的磁盘访问，不适合主存储器有限的情况。

插入排序（D）的时间复杂度较低，但它需要频繁地移动数据，这可能导致频繁的磁盘访问，也不适合主存储器有限的情况。

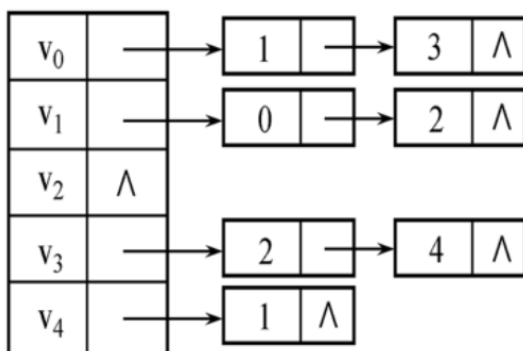
8. 已知一个无向图 G 含有 18 条边，其中度数为 4 的顶点个数为 3，度数为 3 的顶点个数为 4，其他顶点的度数均小于 3，请问图 G 所含的顶点个数至少是（ C ）。
- A: 10 B: 11 **C: 13** D: 15

#一条边贡献两个度数

9. 给定一个无向图 G ，从顶点 V_0 出发进行无向图 G 的深度优先遍历，访问的边集合为： $\{(V_0, V_1), (V_0, V_4), (V_1, V_2), (V_1, V_3), (V_4, V_5), (V_5, V_6)\}$ ，则下面哪条边（C）不能出现在 G 中？
A: (V_0, V_2) B: (V_4, V_6)
C: (V_4, V_3) D: (V_0, V_6)

#在生成树中，跨两颗子树的边会与DFS过程产生矛盾。

10. 已知一个有向图 G 的邻接入边表（或称逆邻接表）如下图所示，从顶点 v_0 出发对该图 G 进行深度优先周游，得到的深度优先周游结点序列为（A）。
A: $V_0V_1V_4V_3V_2$ B: $V_0V_1V_2V_3V_4$ C: $V_0V_1V_3V_2V_4$ D: $V_0V_2V_1V_3V_4$



11. 若按照排序的稳定性和不稳定性对排序算法进行分类，则（D）是不稳定排序。
A: 冒泡排序 B: 归并排序 C: 直接插入排序 **D: 希尔排序**

#根据排序算法的稳定性，如果需要选择一个不稳定排序算法，选项D：希尔排序是正确的选项。

稳定排序算法是指，当有两个相等的元素A和B，且在排序前A出现在B的前面，在排序后A仍然会出现在B的前面。而不稳定排序算法则无法保证这种相对顺序。

冒泡排序（A）和直接插入排序（C）都属于稳定排序算法，它们在比较和交换元素时会考虑相等元素的顺序关系。

归并排序（B）是一种稳定排序算法，它通过分治的思想将待排序的序列划分为较小的子序列，然后逐步合并这些子序列并保持相对顺序。

希尔排序（D）是一种不稳定排序算法，它使用间隔序列来对数据进行分组，然后对每个分组进行插入排序。在插入排序的过程中，相等元素的顺序可能会发生变化。

12. 以下（C）分组中的两个排序算法的最坏情况下时间复杂度的大O表示相同。
A: 快速排序和堆排序 B: 归并排序和插入排序 C: **快速排序和选择排序** D: 堆排序和冒泡排序

#选项C：快速排序和选择排序中的两个排序算法的最坏情况下时间复杂度的大O表示相同。

快速排序和选择排序都属于不同的排序算法，但它们的最坏情况下的时间复杂度都是 $O(n^2)$ 。

快速排序的最坏情况下时间复杂度发生在每次选择的基准元素都划分出了一个很小的子序列，使得递归的深度达到了 n ，导致时间复杂度为 $O(n^2)$ 。

选择排序的最坏情况下时间复杂度发生在每次选择最小（或最大）元素时，需要遍历未排序部分的所有元素，导致时间复杂度为 $O(n^2)$ 。

13. 设结点 X 和 Y 是二叉树中任意的两个结点，在该二叉树的先根周游遍历序列中 X 出现在 Y 之前，而在其后根周游序列中 X 出现在 Y 之后，则 X 和 Y 的关系是（ C ）。
- A: X 是 Y 的左兄弟 B: X 是 Y 的右兄弟
C: X 是 Y 的祖先 D: X 是 Y 的后裔

#如果X和Y分属不交的子树，那么他们在先根和后根周游中的前后关系是一致的。出现反转的话，他们只能是祖先/后裔关系。

14. 考虑一个森林 F，其中每个结点的子结点个数均不超过 2。如果森林 F 中叶子结点的总个数为 L，度数为 2 结点（子结点个数为 2）的总个数为 N，那么当前森林 F 中树的个数为（ C ）。
- A: $L-N-1$ B: 无法确定 **C: $L-N$** D: $N-L$

#一棵树中，叶节点比二度节点数量多一。

15. 回溯法是一类广泛使用的算法，以下叙述中不正确的是（ C ）。
- A: 回溯法可以系统地搜索一个问题的所有解或者任意解
B: 回溯法是一种既具备系统性又具备跳跃性的搜索算法
C: 回溯算法需要借助队列数据结构来保存从根结点到当前扩展结点的路径
D: 回溯算法在生成解空间的任一结点时，先判断当前结点是否可能包含问题的有效解，如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向祖先结点回溯

二. 判断

(10 分，每小题 1 分；对填写“Y”，错填写“N”)

- （ Y ）对任意一个连通的、无环的无向图，从图中移除任何一条边得到的图均不连通。
- （ Y ）给定一棵二叉树，前序周游序列和中序周游序列分别是 HGEDBFCA 和 EGBDHFAC 时，其后序周游序列必是 EBDGACFH。
- （ N ）假设一棵二叉搜索树的结点数在 1 到 1000 之间，现在查找数值为 363 的结点。以下三个序列皆有可能是查过的序列：A). 2, 252, 401, 398, 330, 344, 397, 363; B). 925, 202, 911, 240, 912, 245, 363; C). 935, 278, 347, 621, 299, 392, 358, 363。

#911子树左子节点240的右子节点912，不能比911大

- （ N ）构建一个含 N 个结点的（二叉）最小值堆，建堆的时间复杂度大 O 表示为 $O(N \log_2 N)$ 。
- （ Y ）队列是动态集合，其定义的出队列操作所移除的元素总是在集合中存在时间最长的元素。
- （ N ）任一有向图的拓扑序列既可以通过深度优先搜索求解，也可以通过宽度优先搜索求解。

#必须是DAG

- （ Y ）对任一连通无向图 G，其中 E 是唯一权值最小的边，那么 E 必然属于任何一个最小生成树。

8. (N) 对一个包含负权值边的图，迪杰斯特拉(Dijkstra)算法能够给出最短路径问题的正确答案。
9. (N) 分治算法通常将原问题分解为几个规模较小但类似于原问题的子问题，并要求算法实现写成某种递归形式，递归地求解这些子问题，然后再合并这些子问题的解来建立原问题的解。

#递归的求解子问题，最后子问题足够小可以直接求解，形式不一定是递归。

#分治和动态规划都是将问题分解为子问题，然后合并子问题的解得到原问题的解。但是不同的是，分治法分解出的子问题是不重叠的，因此分治法解决的问题不拥有重叠子问题，而动态规划解决的问题拥有重叠子问题。

10. (Y) 考察某个具体问题是否适合应用动态规划算法，必须判定它是否具有最优子结构性质。

#如果一个问题的最优解可以由其子问题的最优解有效地构造出来，那么称这个问题拥有**最优子结构 (Optimal Substructure)**。最优子结构保证了动态规划中原问题的最优解可以由子问题的最优解推导而来。因此，一个问题必须拥有最优子结构，才能使用动态规划去解决。例如数塔问题中，每一个位置的 dp 值都可以由它的两个子问题推导得到。

至此，重叠子问题和最优子结构的内容已介绍完毕。需要指出，一个问题必须拥有重叠子问题和最优子结构，才能使用动态规划去解决。

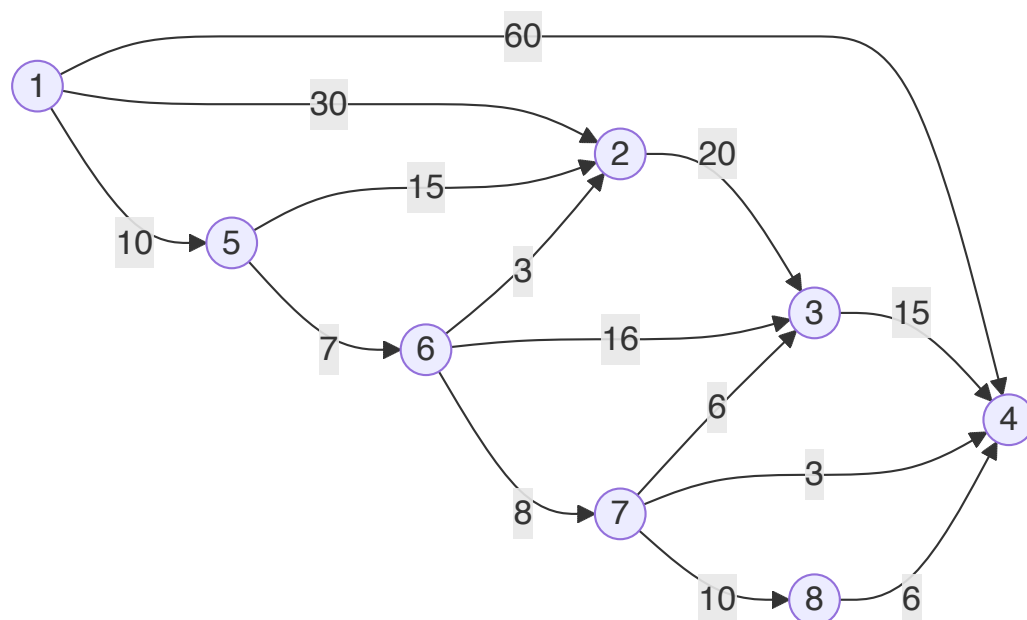
三. 填空 (20 分, 每题 2 分)

1. 目标串长是 n ，模式串长是 m ，朴素模式匹配算法思想为：从目标串第一个字符开始，依次与模式串字符匹配；若匹配失败，则尝试匹配的目标串起始字符位置往后移一位，重新开始依次和模式串字符匹配；.....；直到匹配成功或遍历完整个目标串为止。则该算法中字符的最多比较次数是 _____（使用大 O 表示法）。 $O(mn)$
2. 在一棵含有 n 个结点的树中，只有度（树节点的度指子节点数量）为 k 的分支结点和度为 0 的终端（叶子）结点，则该树中含有的终端（叶子）结点的数目为：_____。 $n - (n-1)/k$
#设叶节点 x 个，分支节点 $n-x$ 个，树的性质有节点数等于所有节点度数加 1， $(n-x)*k + 1 = n$, $x = n - (n-1)/k$
3. 对一组记录进行非递减排序，其关键码为 [46, 70, 56, 38, 40, 80]，则利用快速排序的方法，以第一个记录为基准得到的第一次划分结果为 _____。
#[38, 40, 46, 56, 70, 80]。基本知识点，由于算法有多种实现，只需要 46 把数列分开了就行
4. 对长度为 3 的顺序表进行查找，若查找第一个元素的概率为 $1/2$ ，查找第二个元素的概率为 $1/4$ ，查找第三个元素的概率为 $1/8$ ，则执行任意查找需要比较元素的平均个数为 _____。
$1 * (1/2) + 2 * (1/4) + 3 * (1/8) + 3 * (1/8) = 1.75$ ，还有 $1/8$ 的失败查询概率。
5. 设有一组记录的关键字为 {19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79}，用链地址法（拉链法）构造散列表，散列函数为 $H(\text{key}) = \text{key} \text{ MOD } 13$ ，散列地址为 1 的链中有 _____ 个记录。4
6. 删除长度为 n 的顺序表的第 i 个数据元素需要移动表中的 _____ 个数据元素， $(1 \leq i \leq n)$ 。 $n - i$
7. 已知以数组表示的小根堆为 [8, 15, 10, 21, 34, 16, 12]，删除关键字 8 之后需要重新建堆，在此过程中，关键字的比较次数是 _____。3

8. 在广度优先遍历、拓扑排序、求最短路径三种算法中，可以判断出一个有向图是否有环（回路）的是 ____。拓扑排序
- #求最短路径和广度优先遍历中，发现重复访问的节点时，并不知道该节点是否当前节点的祖先拓扑排序有一种实现是用入度来判断，输出入度为0的节点。
9. 有 n ($n \geq 2$) 个顶点的有向强连通图最少有 ____ 条边。 n
10. 若栈 S1 中保存整数，栈 S2 中保存运算符，函数 F() 依次执行下述各步操作：
- ①从 S1 中依次弹出两个操作数 a 和 b (先弹出 a，再弹出 b)；
 - ②从 S2 中弹出一个运算符 op；
 - ③执行相应的运算 $b \text{ op } a$ ；
 - ④将运算结果压入 S1 中。
- 假定 S1 中的操作数依次是 5, 8, 3, 2 (2 在栈顶)，S2 中的运算符依次是 *, -, // (// 在栈顶)。调用三次 F() 后，S1 栈顶保存的值是 ____。35

四. 简答 (3题, 共20分)

1. (7 分) 试用 Dijkstra 算法求出下图中顶点 1 到其余各顶点的最短路径，写出算法执行过程中各步的状态，填入下表。



顶点1到其他顶点的最短路径长度

所选 顶点	U(已确定最短路径的 顶点集合)	T(未确定最短路径的 顶点集合)	2	3	4	5	6	7	8
初态	{1}	{2, 3, 4, 5, 6, 7, 8}	30	∞	60	10	∞	∞	∞
5	{1, 5}	{2, 3, 4, 6, 7, 8}	25	∞	60	10	17	∞	∞
6	{1, 5, 6}	{2, 3, 4, 7, 8}	20	33	60	10	17	25	∞
2	{1, 5, 6, 2}	{3, 4, 7, 8}	20	33	60	10	17	25	∞
7	{1, 5, 6, 2, 7}	{3, 4, 8}	20	31	28	10	17	25	35
4	{1, 5, 6, 7, 2, 4}	{3, 8}	20	31	28	10	17	25	35
3	{1, 5, 6, 7, 2, 4, 3}	{8}	20	31	28	10	17	25	35
8	{1, 5, 6, 7, 2, 4, 3, 8}	{}	20	31	28	10	17	25	35

2. (6 分) 给定一组记录的关键码集合为{18, 73, 5, 10, 68, 99, 27}, 回答下列 3 个问题:
- 画出按照记录顺序构建形成的二叉排序(搜索)树(2 分);
 - 画出删除关键码为 73 后的二叉排序树(2 分)。
 - 画出令原关键码集合(未删除 73 前)查询效率最高的最优二叉排序树(仅需考虑关键码查询成功时的效率, 且集合内每个关键码被查询概率相等)(2 分)。

#二叉排序树是一种特殊的二叉树, 它具有以下性质:

- 对于树中的每个节点, 其左子树中的所有节点的关键码值小于该节点的关键码值。
- 对于树中的每个节点, 其右子树中的所有节点的关键码值大于该节点的关键码值。
- 左右子树也是二叉排序树。

在构建二叉排序树时, 我们按照以下原理:

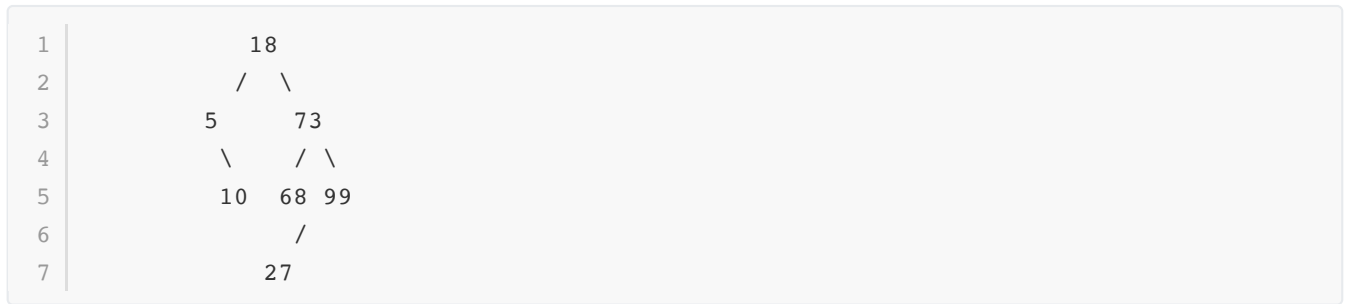
- 从根节点开始, 依次遍历关键码集合中的每个关键码。
- 对于每个关键码, 如果树为空, 则将关键码作为树的根节点。如果树不为空, 则按照二叉排序树的性质, 找到适当的位置插入关键码, 以保持二叉排序树的性质不变。

具体来说, 在插入关键码时, 我们从根节点开始, 逐级比较关键码值, 并根据比较结果决定向左子树或右子树移动, 直到找到合适的插入位置。如果待插入的关键码已经存在于树中, 则不执行任何操作(可以视为树中已经存在该关键码)。

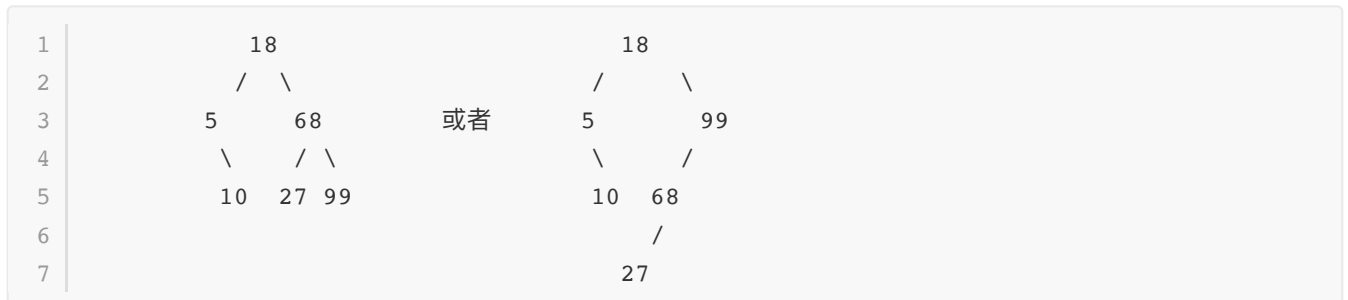
这样, 通过按照关键码集合中记录的顺序依次插入节点, 就可以构建出一个二叉排序树。

让我们逐个回答这些问题:

a) 二叉排序树构建过程如下:

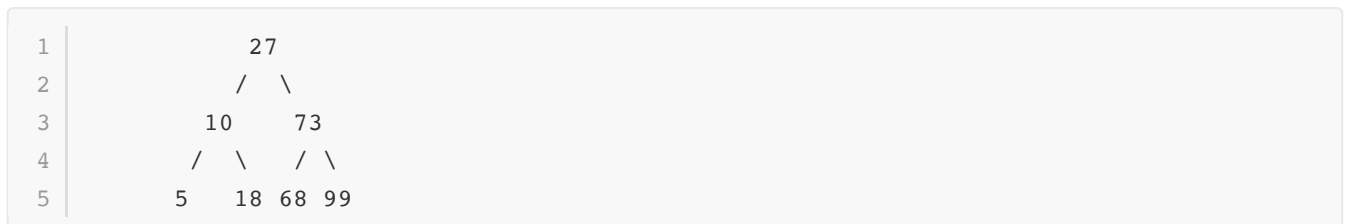


b) 删除关键码为 73 后的二叉排序树:



c) 查询效率最高的最优二叉排序树:

由于关键码集合内每个关键码被查询概率相等，最优二叉排序树应该是平衡树，即左右子树高度差不超过1。因此，我们可以构建一个高度平衡的二叉排序树。



这是一棵高度平衡的二叉排序树，查询效率最高，因为平衡树的查找时间复杂度是 $O(\log n)$ 。

3. (7 分) 奇偶交换排序如下所述：对于原始记录序列 $\{a_1, a_2, a_3, \dots, a_n\}$ ，第一趟对所有奇数 i ，将 a_i 和 a_{i+1} 进行比较，若 $a_i > a_{i+1}$ ，则将二者交换；第二趟对所有偶数 i ；第三趟对所有奇数 i ；第四趟对所有偶数 i ，...，依次类推直到整个记录序列有序为止。代码如下：

```

1  def ExSort(a, n): # a[1..n]为待排序记录, n为记录数目
2
3      change1 = change2 = True # 标志变量, bool型
4      if n <= 0:
5          return "Error"
6      while (change1 or change2):
7
8          change1 = False # 奇数,
9          for i in range(1, n, 2):
10             if a[i] > a[i+1]:
11                 a[i], a[i+1] = a[i+1], a[i]

```

```

12         change1 = True
13
14         if not change1 and not change2:
15             break
16
17         change2 = False # 偶数
18         for i in range(2, n, 2):
19             if a[i] > a[i+1]:
20                 a[i], a[i+1] = a[i+1], a[i]
21                 change2 = True

```

- a) 请写出序列 {18, 73, 5, 10, 68, 99, 27, 10} 在前 4 趟排序中每趟排序后的结果。 (2 分)
- b) 奇偶交换排序是否是稳定的排序? (1 分)
- c) 在序列为初始状态为“正序”和“逆序”两种情况下, 试给出序列长度为 n 的情况下, 排序过程所需进行的关键码比较次数和记录的交换次数? (4 分)

#c)

- o 正序情况下:

关键码比较次数: $n/2 + (n-1)/2$, 分奇偶情况化简下来就是 $n-1$ 。即每两趟需要 $n-1$ 次比较。

记录交换次数: 0。由于序列已经有序, 不需要进行交换,

- o 逆序情况下:

逆序尝试下来应该是在第 n 趟排序后实现正序, 然后再经过两趟确认。每两趟要 $n-1$ 次比较, 总计需要 $n/2$ 个两趟; 再两趟确认 $n-1$ 次。所以总的比较次数是前两个数相乘, 加上第三个数, 即 $n(n-1)/2 + (n-1)$ 。

关键码比较次数: $n(n-1)/2 + (n-1)$

记录交换次数: $n(n-1)/2$, 因为需要俩俩比较确定位置。

- a) 序列 {18, 73, 5, 10, 68, 99, 27, 10} 在前4趟排序中每趟排序后的结果如下:

```

1  def ExSort(a, n): # a[1..n]为待排序记录, n为记录数目
2      change1 = change2 = True # 标志变量, bool型
3      if n <= 0:
4          return "Error"
5      cnt = 0
6      while (change1 or change2):
7          change1 = False # 奇数,
8          for i in range(1, n, 2):
9              if a[i] > a[i+1]:
10                 a[i], a[i+1] = a[i+1], a[i]
11                 change1 = True
12
13             cnt += 1; print(f"pass {cnt}: {a[1:]}")
14             if not change1 and not change2:
15                 break
16
17             change2 = False # 偶数
18             for i in range(2, n, 2):

```

```

19         if a[i] > a[i+1]:
20             a[i], a[i+1] = a[i+1], a[i]
21             change2 = True
22
23         cnt += 1; print(f"pass {cnt}: {a[1:]}")
24         if cnt == 4:
25             break
26
27     # 题面是奇数第一趟，偶数是第二趟，这也没有都都比较，才一半，怎么算一趟？题面有问题吧
28     a = [0] + [18, 73, 5, 10, 68, 99, 27, 10]
29     ExSort(a, len(a)-1)
30     """
31     pass 1: [18, 73, 5, 10, 68, 99, 10, 27]
32     pass 2: [18, 5, 73, 10, 68, 10, 99, 27]
33     pass 3: [5, 18, 10, 73, 10, 68, 27, 99]
34     pass 4: [5, 10, 18, 10, 73, 27, 68, 99]
35     """

```

b) 奇偶交换排序是稳定的排序。稳定排序是指如果两个元素相等，在排序后它们的相对顺序仍然保持不变。奇偶交换排序在交换过程中只涉及相邻的两个元素，因此相等元素之间的相对顺序不会改变。

五. 算法填空（2题，共20分）

1. 填空完成下列程序：读入一个整数序列，用单链表存储之，然后将该单链表颠倒后输出该单链表内容。算法输入的一行是 n 个整数，即要存入单链表的整数序列。

样例输入

1 2 3 4 5

样例输出

5 4 3 2 1

uqrftvpwm

Open Poe faster with our new desktop app! Download

ion Code

isistant

python

```

class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def reverse(self):
        prev = None
        current = self.head
        
```

Start

Python数据结构与算...

ch04基本结构(线性表...

2022数算B-Python...

2021 数算B-python版...

五. 算法填空 (2题, 共 20 分)

1. 填空完成下列程序: 读入一个整数序列, 用单链表存储之, 然后将该单链表颠倒后输出该单链表内容。算法输入的第一行是整数 n, 第二行是 n 个整数, 即要存入单链表的整数序列。

样例输入

1 2 3 4 5

样例输出

5 4 3 2 1

```

class Node:
    def __init__(self, data, next=None):
        self.data, self.next = data, next

class LinkList:
    def __init__(self, lst):
        self.head = Node(lst[0])
        p = self.head
        for i in lst[1:]:
            node = Node(i)
            p.next = ① # (1分)
            p = ② # (2分)

    def reverse(self):
        
```

三个指针来反转链表的指针指向关系 如左侧

这是什么写法, 如何填写?

```

1 class Node:
2     def __init__(self, data, next = None):
3         self.data, self.next = data, next
4
5 class LinkedList:
6     def __init__(self, lst):
7         self.head = Node(lst[0])
8         p = self.head
9         for i in lst[1:]:
10             p.next = Node(i)    # 等号右侧填空 (1分)
11             p = p.next         # 等号右侧填空 (2分)
12
13     def reverse(self): # 把head当pre用, 天才 said by 胡睿诚
14         p = self.head.next
15         self.head.next = None    # 等号右侧填空 (2分)
16         while p is not None:
17             q = p
18             p = p.next            # 等号右侧填空 (1分)
19             q.next = self.head    # 等号右侧填空 (2分)
20             self.head = q        # 留空行, 此行代码需要填写 (2分)
21
22     def reverse_3p(self): # 常规思维: 三个指针来反转链表的指针指向关系

```

```

23     prev = None
24     current = self.head
25     while current:
26         next_node = current.next
27         current.next = prev
28         prev = current
29         current = next_node
30     self.head = prev
31
32     def print_list(self):
33         p = self.head
34         while p:
35             print(p.data, end=" ")
36             p = p.next
37         print()
38
39     #a = list(map(int, input().split()))
40     a = [1, 2, 3, 4, 5]
41     b = a.copy()
42     a = LinkedList(a)
43     b = LinkedList(b)
44     a.reverse()
45     b.reverse_3p()
46     a.print_list()
47     b.print_list()
48     """
49     5 4 3 2 1
50     5 4 3 2 1
51     """
52
53     """
54     5 4 3 2 1
55     """

```

`reverse_3p` 这段代码是用来反转一个链表的指针指向关系的。下面是对代码的逐行解释：

1. `prev = None`: 初始化一个变量 `prev`，用于保存当前节点的前一个节点，初始值为 `None`。
2. `current = self.head`: 初始化一个变量 `current`，指向链表的头节点。
3. `while current:`: 进入一个循环，只要 `current` 不为空（即还有节点未遍历）就继续执行。
4. `next_node = current.next`: 将 `current` 节点的下一个节点保存到 `next_node` 中，以备后续使用。
5. `current.next = prev`: 将 `current` 节点的指针指向前一个节点 `prev`，实现了反转指针的方向。
6. `prev = current`: 将 `prev` 更新为当前节点 `current`，为下一轮循环做准备。
7. `current = next_node`: 将 `current` 更新为下一个节点 `next_node`，继续遍历链表。
8. `self.head = prev`: 最后将链表的头节点更新为反转后的链表的头节点 `prev`，完成整个链表的反转。

这段代码使用了三个指针：`prev`、`current`、`next_node`，通过不断更新它们的指向关系，实现了链表的反转。

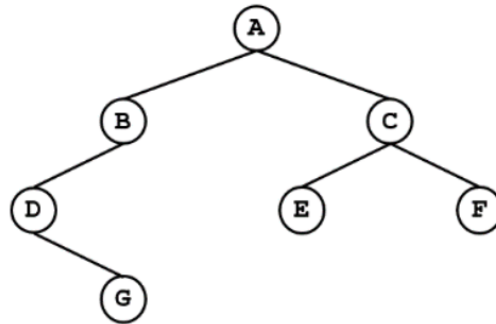
2. 填空完成下列程序：输入一棵二叉树的扩充二叉树的先根周游（前序遍历）序列，构建该二叉树，并输出它的中根周游（中序遍历）序列。这里定义一棵扩充二叉树是指将原二叉树中的所有空引用增加一个表示为@的虚拟叶结点。譬如下图所示的一棵二叉树，

输入样例：

ABD@G@@@CE@@F@@

输出样例：

DGBAECF



```
1 s = input()
2 ptr = 0
3
4 class BinaryTree:
5     def __init__(self, data, left=None, right=None):
6         self.data, self.left, self.right = data, left, right
7
8     def addLeft(self, tree):
9         self.left = tree
10
11    def addRight(self, tree):
12        self.right = tree
13
14    def inorderTraversal(self):
15        if self.left:
16            self.left.inorderTraversal()    # (1分)
17        print(self.data, end=" ")
18        if self.right:
19            self.right.inorderTraversal()    # (1分)
20
21    def buildTree():
22        global ptr
23        if s[ptr] == "@":
24            ptr += 1
25            return None    # (2分)
26        tree = BinaryTree(s[ptr])    # (1分)
27        ptr += 1
28        tree.addLeft(buildTree())    # (2分)
```



```

29     tree.addRight(buildTree()) # (2分)
30
31     return tree
32
33 tree = buildTree() # (1分)
34 tree.inorderTraversal()
35
36 """
37 sample input:
38 ABD@G@@@CE@@F@@
39
40 sample output:
41 DGBAECF
42 """

```

笔试中，对于程序阅读理解，要求还是挺高的。因为AC的代码通常有多种写法，如果考出来写的不规范代码，就有点难受。例如：上面程序，递归程序带着全局变量，难受。

较好的写法是：

```

1  class TreeNode:
2      def __init__(self, data):
3          self.data = data
4          self.left = None
5          self.right = None
6
7  def buildTree(preorder):
8      if not preorder:
9          return None
10
11     data = preorder.pop(0)
12     if data == "@":
13         return None
14
15     node = TreeNode(data)
16     node.left = buildTree(preorder)
17     node.right = buildTree(preorder)
18
19     return node
20
21 def inorderTraversal(node):
22     if node is None:
23         return []
24
25     result = []
26     result.extend(inorderTraversal(node.left))
27     result.append(node.data)
28     result.extend(inorderTraversal(node.right))
29
30     return result

```

```
31
32 preorder = input()
33 tree = buildTree(list(preorder))
34
35 inorder = inorderTraversal(tree)
36 print(''.join(inorder))
37
38 """
39 sample input:
40 ABD@G@@@CE@@F@@
41
42 sample output:
43 DGBAECF
44 """
```

20210620笔试 (Python)

一. 选择题 (30 分, 每小题 2 分)

1. 下列不影响算法时间复杂性的因素有 (C)。
A: 问题的规模 B: 输入值
C: 计算结果 D: 算法的策略
2. 链表不具有的特点是 (A)。
A: 可随机访问任意元素 B: 插入和删除不需要移动元素
C: 不必事先估计存储空间 D: 所需空间与线性表长度成正比
3. 设有三个元素X, Y, Z 顺序进栈 (进的过程中允许出栈), 下列得不到的出栈排列是 (C)。
A: XYZ B: YZX C: ZXY D: ZYX
4. 判定一个无序表 Q (链表实现) 为空的条件是 (A)。
A: Q.head == None B: Q == None
C: Q.head == 0 D: Q.head != None
5. 若定义二叉树中根结点的层数为零, 树的高度等于其结点的最大层数加一。则当某二叉树的前序序列和后序序列正好相反, 则该二叉树一定是 (B) 的二叉树。
A: 空或只有一个结点 B: 高度等于其节点数
C: 任一结点无左孩子 D: 任一结点无右孩子
6. 任意一棵二叉树中, 所有叶结点在前序、中序和后序周游序列中的相对次序 (B)。
A: 发生改变 B: 不发生改变
C: 不能确定 D: 以上都不对
7. 假设线性表中每个元素有两个数据项 key1 和 key2, 现对线性表按以下规则进行排序: 先根据数据项 key1 的值进行非递减排序; 在 key1 值相同的情况下, 再根据数据项 key2 的值进行非递减排序。满足这种要求的排序方法是 (D)。
A: 先按key1 值进行冒泡排序, 再按 key2 值进行直接选择排序
B: 先按key2 值进行冒泡排序, 再按 key1 值进行直接选择排序
C: 先按key1 值进行直接选择排序, 再按 key2 值进行冒泡排序
D: 先按key2 值进行直接选择排序, 再按 key1 值进行冒泡排序
8. 有 n^2 个整数, 找到其中最小整数需要比较次数至少为 (C) 次。
A: n B: $\log_2 n$ C: n^2-1 D: n-1
9. n 个顶点的无向完全图的边数为 (C)。
A: n (n-1) B: n (n+1)
C: n (n-1) / 2 D: n (n+1) / 2
10. 设无向图 $G = (V, E)$, 和 $G' = (V', E')$, 如果 G' 是 G 的生成树, 则下面说法中错误的是 (B)。
A: G' 是 G 的子图 B: G' 是 G 的连通分量
C: G' 是 G 的极小连通子图且 $V=V'$ D: G' 是 G 的一个无环子图

[解析] 选项B错误, 因为连通分量是无向图的极大连通子图, 其中极大的含义是将依附于连通分量中顶点的所有边都加上, 所以, 连通分量中可能存在回路。(可能不是树的结构)

连通分量和强连通分量是两个不同的概念。

1. **连通分量 (Connected Component)**：在无向图中，一个连通分量是指图中的一个极大连通子图，其中的任意两个节点都可以通过图中的边互相到达。连通分量可以理解为无向图的极大连通子图。
2. **强连通分量 (Strongly Connected Component)**：在有向图中，一个强连通分量是指图中的一个极大子图，其中的任意两个节点都可以通过有向路径互相到达。强连通分量可以理解为有向图的极大连通子图。

强连通分量是针对有向图而言的，而连通分量是针对无向图而言的。因此，它们的定义和性质有所不同。

11. 有一个散列表如下图所示，其散列函数为 $h(\text{key}) = \text{key} \bmod 13$ ，该散列表使用再散列函数 $H2(\text{Key}) = \text{Key} \bmod 3$ 解决碰撞，问从表中检索出关键码 38 需进行几次比较（B）。

0	1	2	3	4	5	6	7	8	9	10	11	12
26	38			17			33		48			25

A: 1 B: 2 C: 3 D: 4

解释：

查找关键码38的过程：

1. 计算初始位置： $h(38) = 38 \bmod 13 = 12$

2. 第1次比较：查找索引12，发现值为25，不是38，发生冲突。

3. 计算再散列偏移量： $H2(38) = 38 \bmod 3 = 2$

4. 第2次比较：计算新位置 $(12+2) \bmod 13 = 14 \bmod 13 = 1$ ，查找索引1，发现值为38，匹配成功。

因此，从表中检索出关键码38需要进行 2次 比较。

答案：B(2次比较)

12. 在一棵度为 3 的树中，度为 3 的节点个数为 2，度为 2 的节点个数为 1，则度为 0 的节点个数为（C）。
A: 4 B: 5 C: 6 D: 7
13. 由同一组关键字集合构造的各棵二叉排序树(B)。
A: 其形态不一定相同，但平均查找长度相同
B: 其形态不一定相同，平均查找长度也不一定相同
C: 其形态均相同，但平均查找长度不一定相同
D: 其形态均相同，平均查找长度也都相同
14. 允许表达式内多种括号混合嵌套，检查表达式中括号是否正确配对的算法，通常选用（A）。
A: 栈 B: 线性表 C: 队列 D: 二叉排序树
15. 在映射抽象数据类型（ADT Map）的不同实现方法中，适合对动态查找表进行高效率查找的组织结构是（C）。
A: 有序表 B: 堆排序 C: 二叉排序树 D: 快速排序

额外题目：

16. 下列叙述中正确的是（A）。

- A: 一个逻辑结构可以有多种类型的存储结构，且不同类型的存储结构会直接影响到数据处理效率
- B: 散列是一种基于索引的逻辑结构
- C: 基于顺序表实现的逻辑结构属于线性结构
- D: 数据结构设计影响算法效率，逻辑结构起到了决定作用

17. 在广度优先搜索算法中，一般使用什么辅助数据结构？（A）

- A: 队列 B: 栈 C: 树 D: 散列表

二. 判断

(10 分，每小题 1 分；对填写“Y”，错填写“N”)

- 1. （Y）考虑一个长度为 n 的顺序表中各个位置插入新元素的概率是相同的，则顺序表的插入算法平均时间复杂度为 $O(n)$ 。
- 2. （N）希尔排序算法的每一趟都要调用一次或多次直接插入排序算法，所以其效率比直接插入排序算法差。
- 3. （Y）直接插入排序、冒泡排序、希尔排序都是在数据正序的情况下比数据在逆序的情况下要快。
- 4. （N）由于碰撞的发生，基于散列表的检索仍然需要进行关键码对比，并且关键码的比较次数仅取决于选择的散列函数与处理碰撞的方法两个因素。
- 5. （N）若有一个叶子结点是二叉树中某个子树的前序遍历结果序列的最后一个结点，则它一定是该子树的中序遍历结果序列的最后一个结点。
- 6. （Y）若某非空且无重复元素二叉树的先序序列和后序序列正好相同，则该二叉树只有一个根结点。
- 7. （Y）有 n 个节点的二叉排序树有多种，其中树高最小的二叉排序树是搜索效率最好。
- 8. （N）强连通分量是有向图中的最小强连通子图。
- 9. （Y）用相邻接矩阵法存储一个图时，在不考虑压缩存储的情况下，所占用的存储空间大小只与图中结点数有关，而与图的边数无关。
- 10. （N）若定义一个有向图的根是指可以从这个结点可到达图中任意其他结点，则可知一个有向图中至少有一个根。

额外题目：

- 11. （Y）按照前序、中序、后序方式周游一棵二叉树，分别得到不同的结点周游序列，然而三种不同的周游序列中，叶子结点都将以相同的顺序出现。
- 12. （N）构建一个含 N 个结点的（二叉）最小值堆，时间效率最优情况下的时间复杂度大 O 表示为 $O(N \log N)$ 。

三. 填空 (20 分, 每题 2 分)

1. 线性表的顺序存储与链式存储是两种常见存储形式; 当表元素有序排序进行二分检索时, 应采用 ____ 存储形式。 (顺序)
2. 如果只想得到1000个元素的序列中最小的前5个元素, 在冒泡排序、快速排序、堆排序和归并排序中, 哪种算法最快? (堆排序)
3. 设环形队列的容量为20 (单元编号从0到19), 现经过一系列的入队和出队运算后, 队头变量 (第一个元素的位置) $front=18$, 队尾变量 (待插入元素的位置) $rear=11$, 在这种情况下, 环形队列中有 ____ 个元素。 (13)
4. 一棵含有101个结点的二叉树中有36个叶子结点, 度为2的结点个数是 ____ 和度为1的结点个数是 ____。 (35, 30)
5. 已知二叉树的前序遍历结果 (先根周游序列) 为 ADC, 这棵二叉树的树型有 ____ 种可能。 (3)
6. 已知二叉树的中序序列为 DGBAECF, 后序序列为 GDBEFCA, 该二叉树的前序序列是 ____。 (ABDGCEF)
7. 对于具有57个结点的完全二叉树, 如果按层次自顶向下, 同一层自左向右, 顺序从0开始对全部结点进行编号, 则有: 编号为18的结点的父结点的编号是 ____, 编号为19的结点的右子女结点的编号是 ____。 (8, 40)
8. 有n个数据对象的二路归并排序中, 每趟归并的时间复杂度为 ____。 ($O(n)$)
9. 对一组记录进行降序排序, 其关键码为 (46, 70, 56, 38, 40, 80, 60, 22), 采用初始步长为4的希尔 (shell) 排序, 第一趟扫描的结果是 (_____); 而采用归并排序第一轮归并结果是 (_____)。 (46,80,60,38,40,70,56,22), (70,46,56,38,80,40,60,22)
10. 如果一个图节点多而边少 (稀疏图), 适宜采用邻接矩阵和邻接表中的 ____ 方式进行存储。

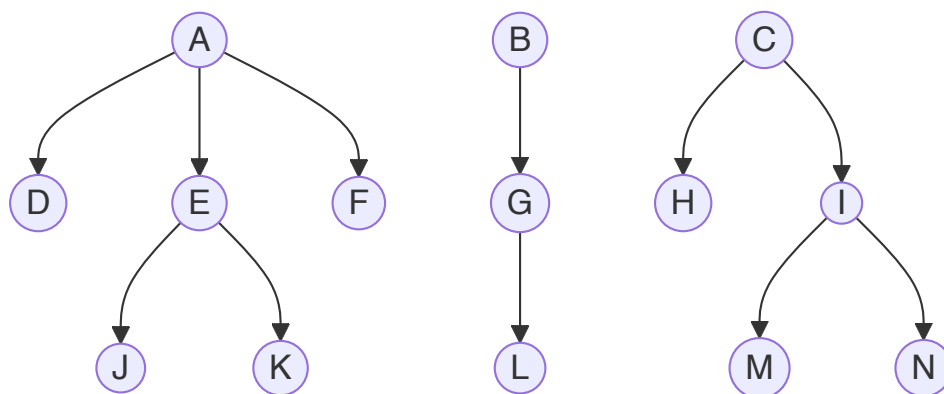
额外题目:

11. 定义完全二叉树的根结点所在层为第一层。如果一个完全二叉树的第六层有23个叶结点, 则它的总结点数可能为? (请填写所有3个可能的结点数, 写对1个得1分, 2个得1.5分, 写错1个不得分)。(54, 80, 81)

#可以是第六层是满的 但是叶节点只有23个 还有9个节点是非叶节点。这样第七层要么是17个节点要么是18个。(1+2+4+8+16+32) + 17 或 18 = 80, 或 81

四. 简答 (24 分, 每小题 6 分)

1. 树周游算法可以很好地应用到森林的周游上。查看下列森林结构, 请给出其深度优先周游序列和广度优先周游序列。



Answer:

深度优先：ADEJKFBGLCHIMN；广度优先：ABCDEFGHijklmn

森林的广度优先周游序列定义如下：

1. 从森林中的每棵树的根节点开始，将根节点加入到遍历队列中。
2. 从队列中取出一个节点，并将其加入到遍历结果中。
3. 将该节点的所有未被访问过的子节点加入到队列中。
4. 重复步骤2和步骤3，直到队列为空。

在广度优先周游序列中，每一层的节点按照从左到右的顺序加入到遍历结果中。这确保了每棵树的每一层都会被遍历到，而且同一层的节点会按照从左到右的顺序被遍历。

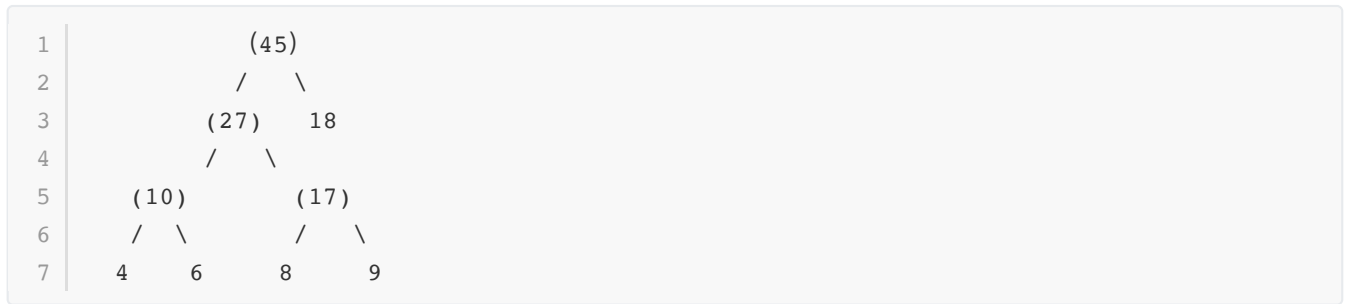
2、哈夫曼树是进行编码的一种有效方式。设给定五个字符，其相应的权值分别为 {4, 8, 6, 9, 18}，试画出相应的哈夫曼树，并计算它的带权外部路径长度 WPL 。

为了构建哈夫曼树，我们需要按照权值从小到大对字符进行排序，并不断地合并权值最小的两个节点，直到只剩下一个节点为止。这个节点就是哈夫曼树的根节点。每次合并两个节点时，新节点的权值为这两个节点的权值之和。

下面是构建哈夫曼树的过程：

1. 将给定的五个字符按照权值从小到大排序：
 1. {4, 6, 8, 9, 18}
2. 不断合并权值最小的两个节点，直到只剩下一个节点：
 1. 合并 4 和 6，得到节点 a，权值为 10
 2. 合并 8 和 9，得到节点 b，权值为 17
 3. 合并节点 a 和 b，得到节点c，权值为 27
 4. 合并节点c和 18，得到根节点d，权值为45

下面是相应的哈夫曼树：



计算带权外部路径长度（WPL）：

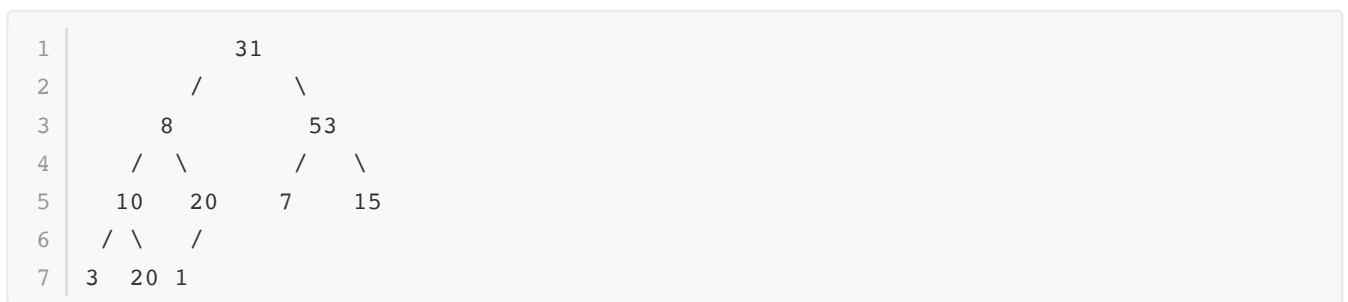
$$\begin{aligned}
 WPL &= 4 * 3 + 6 * 3 + 8 * 3 + 9 * 3 + 18 * 1 \\
 &= 12 + 18 + 24 + 27 + 18 \\
 &= 99
 \end{aligned}$$

所以，哈夫曼树的带权外部路径长度为 99。

3、下图是一棵完全二叉树：

- 1) 请根据初始建堆算法对该完全二叉树建堆，请画出构建的小根堆（2 分）；
- 2) 基于（1）中得到的堆，删除其中的最小元素，请用图给出堆的调整过程（2 分）；
- 3) 基于（1）中得到的堆，向其中插入元素 2，请给出堆的调整过程（2 分）。

注：每移动一个元素视为一个执行步骤，画出所有执行步骤



解答：

对完全二叉树建堆，给出初始建堆算法python

```

1 def heapify(arr):
2     n = len(arr)
3     # 从最后一个非叶子节点开始，依次向下调整每个节点
4     for i in range(n // 2 - 1, -1, -1):
5         heapify_down(arr, i, n)
6
7 def heapify_down(arr, i, n):
8     smallest = i # 假设当前节点最小
9     left_child = 2 * i + 1
10    right_child = 2 * i + 2
11
12    # 如果左子节点存在且小于当前节点，则更新最小值索引
  
```



```

13     if left_child < n and arr[left_child] < arr[smallest]:
14         smallest = left_child
15
16     # 如果右子节点存在且小于当前节点, 则更新最小值索引
17     if right_child < n and arr[right_child] < arr[smallest]:
18         smallest = right_child
19
20     # 如果最小值索引不等于当前节点索引, 则交换当前节点和最小值节点的值, 并继续向下调整
21     if smallest != i:
22         arr[i], arr[smallest] = arr[smallest], arr[i]
23         heapify_down(arr, smallest, n)
24
25 # 测试
26 arr = [31, 8, 53, 10, 20, 7, 15, 3, 20, 1]
27 heapify(arr)
28 print(arr)
29 # [1, 3, 7, 10, 8, 53, 15, 31, 20, 20]

```

1) 根据初始建堆算法, 我们从最后一个非叶子节点开始, 依次向上调整, 保证每个节点都满足小根堆的性质。

```

1      1
2     /  \
3    3    7
4   / \  / \
5  10 8 53 15
6  / \
7 31 20

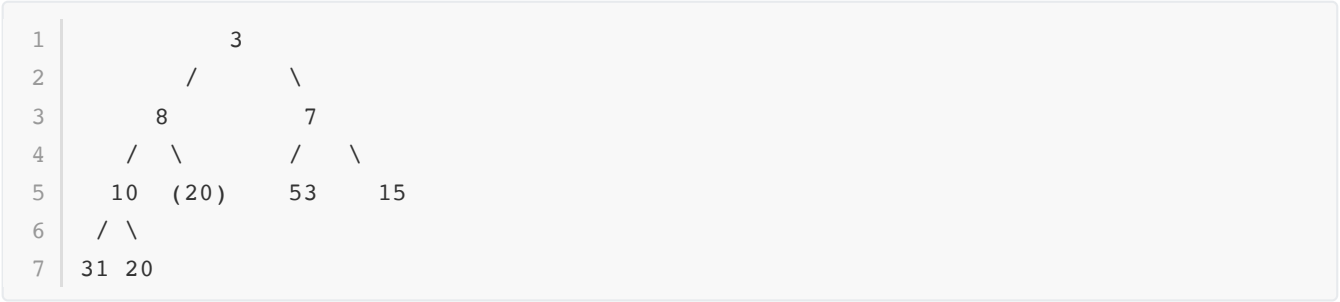
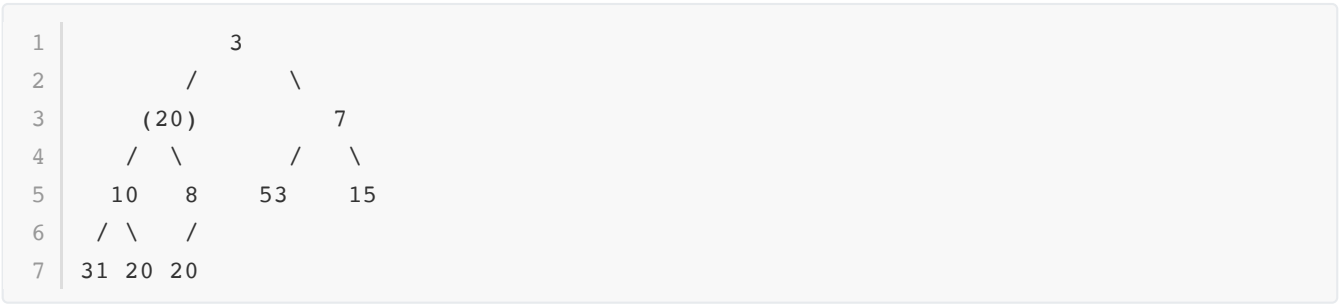
```

2) 删除最小元素后, 将最后一个元素放到根节点, 并进行堆调整:

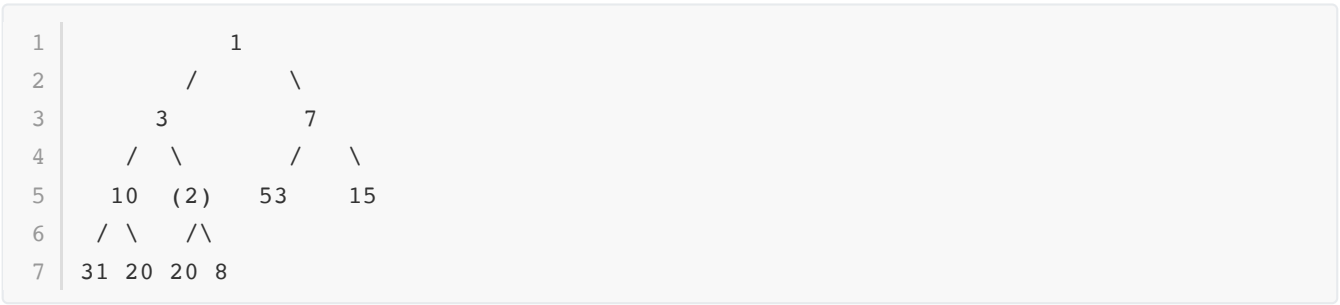
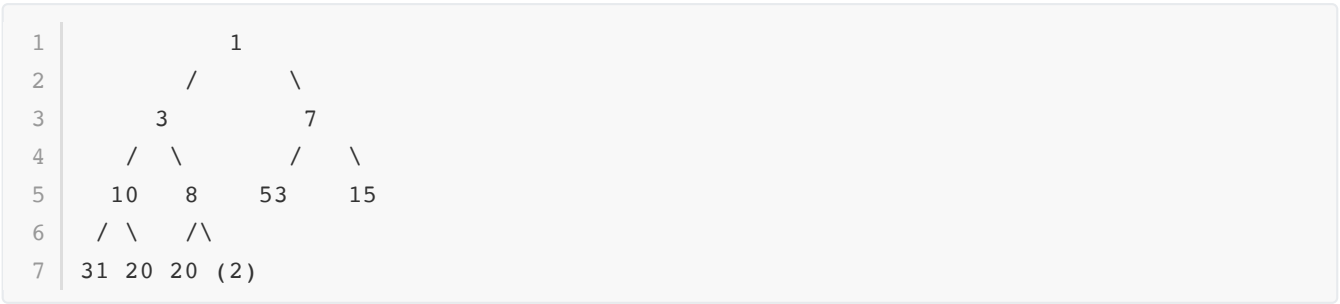
```

1      (20)
2     /  \
3    3    7
4   / \  / \
5  10 8 53 15
6  / \
7 31 20

```



3) 插入元素 2 后，将元素 2 放到堆的最后，然后进行堆调整：

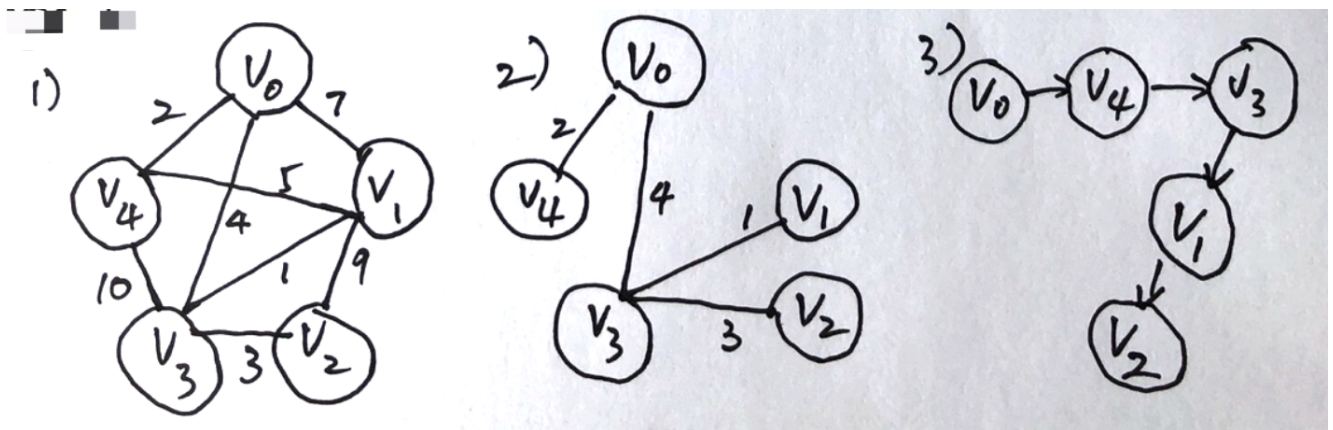


1				1			
2		/			\		
3		(2)			7		
4		/	\		/	\	
5		10	3		53	15	
6		/	\	/	\		
7		31	20	20	8		

4、已知图 G 的顶点集合 $V=\{V_0, V_1, V_2, V_3, V_4\}$ ，邻接矩阵如下图所示，可用 prim 算法求 G 的最小生成树。

$$\begin{bmatrix} 0 & 7 & \infty & 4 & 2 \\ 7 & 0 & 9 & 1 & 5 \\ \infty & 9 & 0 & 3 & \infty \\ 4 & 1 & 3 & 0 & 10 \\ 2 & 5 & \infty & 10 & 0 \end{bmatrix} \quad (19)$$

- 1) 根据邻接矩阵，画出图 G (2 分)；
- 2) 根据 prim 算法，求图 G 从顶点 V_0 出发的最小生成树 (2 分)；
- 3) 用图表示出最小生成树每一步的生成过程 (2 分)。



五. 算法 (16 分, 每小题 8 分)

1、已知下列 pre2post 函数的功能是根据一个满二叉树的前序遍历序列，求其后序遍历序列，请完成填空（假设序列长度不超过 32）。

```

1 # 返回先根序列 preorder[start:start+length]对应的后根序列
2 def pre2post(preorder, start, length):
3     if length == 1:
4         return preorder[start] # 1分
5     else:
6         length = length // 2 # 2分
7         left = pre2post(preorder, start + 1, length) # 1分
8         right = pre2post(preorder, start + 1 + length, length) # 2分
9         root = preorder[start] # 2分
10        return left + right + root
11
12 print(pre2post("ABC", 0, 3)) # 输出 BCA
13 print(pre2post("ABDECFG", 0, 7)) # 输出 DEBFGCA

```

2、阅读下列程序，完成图的深度优先周游算法实现的迷宫探索。已知图采用邻接表表示，Graph 类和 Vertex 类基本定义如下：

```

1 class Graph:
2     def __init__(self):
3         self.vertices = {}
4
5     def addVertex(self, key, label): #添加节点, id 为key, 附带数据 label
6         self.vertices[key] = Vertex(key, label)
7
8     def getVertex(self, key): # 返回 id 为 key 的节点
9         return self.vertices.get(key)
10
11    def __contains__(self, key): # 判断 key 节点是否在图中
12        return key in self.vertices
13
14    def addEdge(self, f, t, cost=0): # 添加从节点 id==f 到 id==t 的边
15        if f in self.vertices and t in self.vertices:
16            self.vertices[f].addNeighbor(t, cost)
17
18    def getVertices(self): # 返回所有的节点 key
19        return self.vertices.keys()
20
21    def __iter__(self): # 迭代每一个节点对象
22        return iter(self.vertices.values())
23
24
25 class Vertex:
26     def __init__(self, key, label=None): # 缺省颜色为"white"
27         self.id = key
28         self.label = label
29         self.color = "white"
30         self.connections = {}
31
32     def addNeighbor(self, nbr, weight=0): # 添加到节点 nbr 的边

```

```

33         self.connections[nbr] = weight
34
35     def setColor(self, color): # 设置节点颜色标记
36         self.color = color
37
38     def getColor(self): # 返回节点颜色标记
39         return self.color
40
41     def getConnections(self): # 返回节点的所有邻接节点列表
42         return self.connections.keys()
43
44     def getId(self): # 返回节点的 id
45         return self.id
46
47     def getLabel(self): # 返回节点的附带数据 label
48         return self.label
49
50 #https://github.com/Yuqiu-
Yang/problem_solving_with_algorithms_and_data_structures_using_python/blob/maste
r/ch7/ch4_maze2.txt
51 mazelist = [
52     "+++++",
53     "+  +  ++ ++  +",
54     "E    +  ++++++",
55     "+ +  ++  ++++ +++ ++",
56     "+ +  + + ++  +++ +",
57     "+          ++ ++ + +",
58     "+++++ + +  ++ + +",
59     "+++++ +++ + + ++ +",
60     "+          + + S+ + +",
61     "+++++ + + + + + +",
62     "+++++",
63 ]
64
65 def mazeGraph(mlist, rows, cols): # 从 mlist 创建图, 迷宫有 rows 行 cols 列
66     mGraph = Graph()
67     vstart = None
68     for row in range(rows):
69         for col in range(cols):
70             if mlist[row][col] != "+":
71                 mGraph.addVertex((row, col), mlist[row][col])
72                 if mlist[row][col] == "S":
73                     vstart = mGraph.getVertex((row, col)) # 等号右侧填空 (1分)
74
75     for v in mGraph:
76         row, col = v.getId()
77         for i in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
78             if 0 <= row + i[0] < rows and 0 <= col + i[1] < cols:
79                 if (row + i[0], col + i[1]) in mGraph:
80                     mGraph.addEdge((row, col), (row + i[0], col + i[1])) #括号中两
个参数填空 (1分)
81

```

```

82     return mGraph, vstart # 返回图对象, 和开始节点
83
84
85 def searchMaze(path, vcurrent, mGraph): # 从 vcurrent 节点开始 DFS 搜索迷宫, path 保
    存路径
86     path.append(vcurrent.getId())
87     vcurrent.setColor("gray")
88     if vcurrent.getLabel() != "E":
89         done = False
90         for nbr in vcurrent.getConnections(): # in 后面部分填空 (2分)
91             nbr_vertex = mGraph.getVertex(nbr)
92             if nbr_vertex.getColor() == "white":
93                 done = searchMaze(path, nbr_vertex, mGraph) # 参数填空 (2分)
94                 if done:
95                     break
96         if not done:
97             path.pop() # 这条语句空着, 填空 (2分)
98             vcurrent.setColor("white")
99     else:
100         done = True
101     return done # 返回是否成功找到通路
102
103
104 g, vstart = mazeGraph(mazelist, len(mazelist), len(mazelist[0]))
105 path = []
106 searchMaze(path, vstart, g)
107 print(path)
108
109 # [(8, 15), (7, 15), (7, 14), (6, 14), (5, 14), (4, 14), (4, 13), (5, 13), (6,
    13), (6, 12), (6, 11), (6, 10), (5, 10), (5, 9), (4, 9), (3, 9), (2, 9), (2, 8),
    (2, 7), (1, 7), (1, 6), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (5, 4), (4, 4),
    (3, 4), (2, 4), (2, 3), (1, 3), (1, 2), (2, 2), (2, 1), (2, 0)]

```