测一测

- 1. 从逻辑上可以把数据结构分成（）几大类
  A. 动态结构、静态结构
  B. 顺序结构、连式结构
  C. 线性结构、非线性结构
  D. 初等结构、构造型结构
- 2. 以下数据结构中，（）是非线性数据结构
  A. 树
  B. 队
  C. 栈

测一测

- 3. 计算机算法时指:
  - A. 计算方法
  - B. 排序方法
  - C. 解决问题的步骤序列
  - D. 调度方法
- 4. 关于算法的说法正确的是（）
  - A. 算法必须由计算机程序实现
  - B. 为解决某问题的算法与该问题编写的程序含义是相同的
  - C. 算法的可行性是指指令不能有二义性
  - D. 以上都不对

测一测

- 5. 算法的计算量的大小称为算法的（）
  A. 效率
  B. 复杂度
  C. 现实性
  D. 难度
- 6. 算法的时间复杂度取决于（）
  A. 问题的规模
  B. 待处理的数据初态
  C. A和B

测一测

- 7. 有下列运行时间：　（1）$f_1(n) = 1000$; (2) $f_2(n) = n^2 + 1000n$; (3) $f_3(n) = n^3 + 100n^2 + n + 1$;
  A. O(1)
  B. O(n)
  C. O(2n)
  D. $O(n^2)$
  E. $O(n^3)$

测一测

- 8. 下面的程序段中，对x的复制语句的频度为（）

```
for (i = 0; i< n; ++i)
    for (j = 0; j <= i; ++j)
        ++x;
```

A. O(1)
B. O(n)
C. O(2n)
D. $O(n^2)$
E. $O(n^3)$

线性表的定义

A

- 9. **线性表是（ ）**
  - A. **一个有限序列，可以为空**
  - B. **一个有限序列，不能为空**
  - C. **一个无限序列，可以为空**
  - D. **一个无限序列，不能为空**
- 10. **线性表是具有n个（ ）的有限序列**
  - A. **表元素**
  - B. **字符**
  - C. **数据元素**
  - D. **数据项**

线性表的顺序存储结构（定义）

- 11. 一个向量（一种顺序表），第一个元素的存储地址是100，每个元素的长度是2，则第五个元素的地址是多少呢？

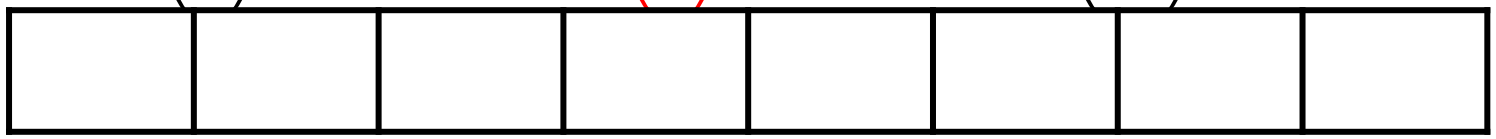
  A. 110　　B. 108　　C. 100　　D.120

Loc(a[k])= Loc (a[0]) + k*c = 100 + (5-1)*2 = 108

顺序存储的线性表的基本操作

- 12. 在长度为n的顺序表的表尾插入一个新的元素的时间复杂度为（）

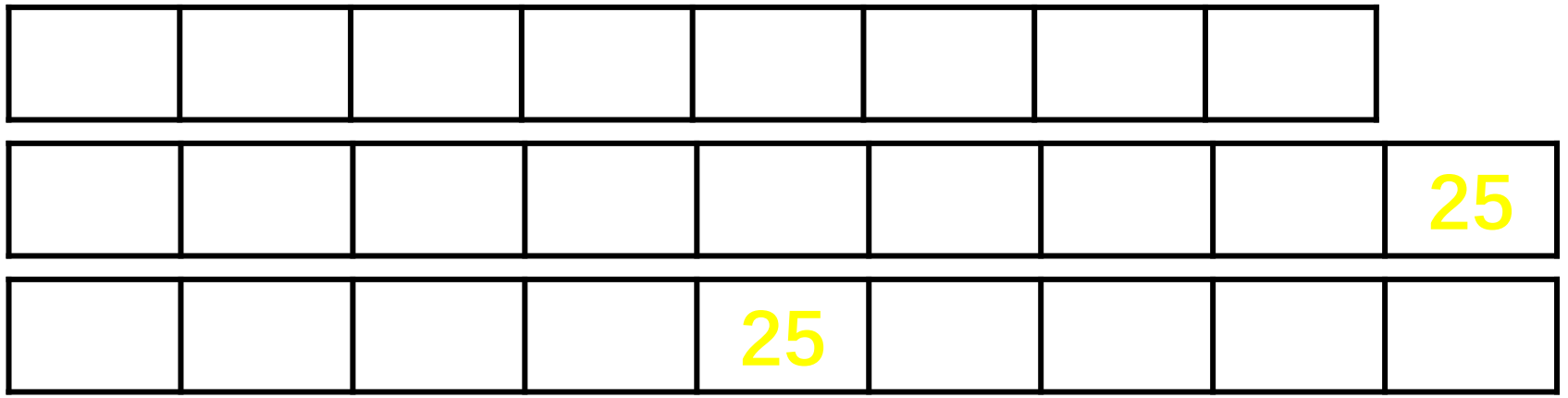
A. O(n)　　B. O(1)　　C. $O(n^2)$　　D. $O(\log_2 n)$



顺序存储的线性表的基本操作

- 13.在长度为n的顺序表的插入一个新的元素的时间复杂度为（）

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



时间复杂度的期望：

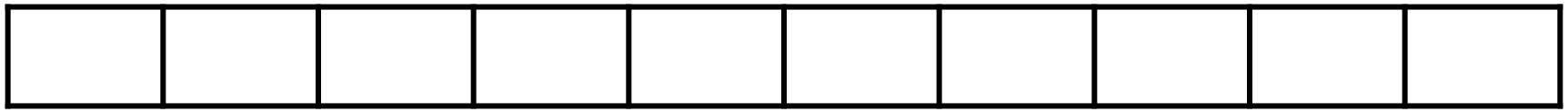
E(n)=(n+(n-1)+(n-2)+…… +1+0) /(n+1) + 1 = n/2 + 1

顺序存储的线性表的基本操作

- 14. 若设一个顺序表的长度为n。
- 那么在表中顺序查找一个值为x的元素时，在等概率的情况下，查找成功的数据平均比较次数为（C）。
- 向表中的第i（$1 \leq i \leq n+1$）个元素位置插入一个新元素时，为保持插入后表中原有元素的相对次序不变，需要从后向前依次后移（F）个元素。
- 在删除表中第i（$1 \leq i \leq n+1$）个元素时，为保持删除后表中原有元素的相对次序不变，需要从前向后依次前移（E）个元素。

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

F. n-i+1　　G. n-i-1　　J. i



随堂测试

- 15. 线性表采用链表存储时，其结点的地址（）
  - A. 必须是连续的
  - B. 一定是不连续的
  - C. 部分地址必须是连续的
  - D. 连续是否均可以
- 16. 不带表头节点的单链表head为空的判定条件是（）；带表头的单链表head为空的判定条件是()
  - A. head == NULL
  - B. head->next == NULL
  - C. head->next == first
  - D. head != NULL

随堂练习

- 17. 已知单链表中的节点p不是链尾结点，若在p之后插入节点s，则应执行（）操作。
  - A. s->next = p; p->next = s;
  - B. p->next = s; s->next = p;
  - C. s->next = p->next; p = s;
  - D. s->next = p->next; p->next = s;

随堂练习

- 18. 已知L是带表头结点的单链表（L是哨位结点），则删除头结点的语句是（）。
  - A. L = L->next;
  - B. L->next = L->next->next;
  - C. L = L->next->next;
  - D. L->next = L;

随堂练习

- 19. 已知单链表A的长度是m，单链表B的长度为n，若将B链接在A的末尾，在没有链尾指针的情况先，算法的时间复杂度应为（）。
  - A. O(1);
  - B. O(m);
  - C. O(n);
  - D. O(m + n);

随堂练习

- 20. 循环链表的主要优点是（）
  - A. 不再需要头指针了
  - B. 已知某个结点的位置后，能够容易找到它的直接前驱
  - C. 在进行插入、删除运算时，能更好地保证链表不断开
  - D. 从表中的任意节点出发都能扫描到整个链表
- 21. 非空的循环单链表head的链尾结点P满足（）
  - A. P->next = NULL
  - B. P = NULL
  - C. P->next = head
  - D. P = head

随堂练习

- 22. 若线性表中最常用得到操作是在最后一个元素之后插入一个元素和删除第一个元素，则采用（）存储方式最节省运算时间。
  - A. 单链表
  - B. 仅有头指针的单循环链表
  - C. 双链表
  - D. 仅有尾指针的单循环链表
- 23. 设双向循环链表中结点的结构为（data, prior, next），且不带表头节点。若想在结点p之后插入结点s，则应执行（）操作。
  - A. p->next = s; s->prior = p; p->next->prior = s; s->next = p->next;
  - B. p->next = s; p->next->prior = s; s->prior = p; s->next = p->next;
  - C. s->prior = p; s->next = p->next; p->next = s; p->next->prior = s;
  - D. s->prior = p; s->next = p->next; p->next->prior = s; p->next = s;

随堂测试

- 24. 顺序表和链表是线性表的两种存储结构，下列操作中，（）在顺序存储结构上的实现要比在链表存储结构上实现的效率高。

  I. 输出表中任意一个元素的值

  II. 交换表中第i 个和第j 个元素的值（i≠j，i和j均小于表长）

  III. 顺序输出表中所有元素的值

  A. I

  B. I、II

  C. I、III

  D. II、III

随堂练习

- 25. 栈操作数据的原则是（）
  - A. 先进先出
  - B. 后进先出
  - C. 后进后出
  - D. 不分顺序
- 26. 某栈的输入序列是1,2,3,4，则不可能得到的输出序列是
  - A. 1, 2, 3, 4
  - B. 4, 1, 2, 3
  - C. 4, 3, 2, 1
  - D. 1, 3, 4, 2

随堂练习

- 在实现顺序栈操作时
  - 27-1. 在进栈之前应先判断栈是否（）
  - 27-2. 在出站之前应先判断栈是否（）

  A. 空　B. 满　C. 上溢　D. 下溢
- 28.以下有关顺序栈的操作中，正确的是（）

  A. n个元素进入一个栈后，它们的出栈顺序一定与进栈顺序相反（一次性进栈完毕再出栈）

  B. 若一个栈的存储空间为S[n]，则对栈的进栈和出栈操作最多只能执行n次

  C. 栈是一种对进栈和出栈操作的次序做了限制的线性表

  D. 空栈没有栈顶指针

随堂练习

- 29. 设链式栈中结点的结构为（data，next），top是指向栈顶的指针。若想在链式栈中入栈一个由指针s所指的结点，则应执行的操作是（）
  - A. top->next = s
  - B. s->next = top->next; top->next = s;
  - C. s->next = top; top = s;
  - D. s->next = top; top = top->next
- 30. 如果以链表作为栈的存储结构，则元素出栈时（）
  - A. 必须判断链栈是否为满
  - B. 必须判断链栈元素的类型
  - C. 必须判断链栈是否为空
  - D. 无须做任何判断

随堂练习

- 31. 为了增加内存空间的利用率和减少移除的可能，在两个栈共享一篇连续的存储空间时，应将两个栈的栈顶（初始栈底和栈顶重合；元素进栈时，两栈顶相向运动）分设在这篇存储空间的两端，当（）时才产生上溢。

  A. 两个栈的栈顶同时到达栈空间的中心点

  B. 其中一个栈的栈顶到达栈空间的中心点

  C. 两个栈的栈顶相加超过了栈空间的最大容量

  D. 两个栈的栈顶相加超过了栈空间的最大容量

随堂练习

- 32. 为解决计算机主机与打印机之间速度不匹配的问题，通常设置一个打印数据缓冲区。主机将要输出的数据一次写入该缓冲区，而打印机则依次从该缓冲区取出数据。该缓冲区的逻辑结构应该是（）

  A. 栈　　B. 队列　　C. 树　　D. 图

- 33.一个队列的入队序列为1，2，3，4，则可能的出队序列是（）

  A. 4,3,2,1　　B. 1,2,3,4

  C. 1,4,3,2　　D. 3,2,4,1

随堂练习

- 34. 设循环队列的小标范围是0~n-1，其头尾指针分别为f和r，则其元素个数为（）

  A. r – f　　B. r-f-1　　C. (r-f)%n + 1　　D. (r-f+n)%n

- 35. 若用一个大小为6的数组来实现循环队列，并且当rear和font的值分别为0和3，当从队列中删除一个元素，再加入两个元素后，rear和front的值分别为（）

  A. 1和5　B. 2和4　C. 4和2　D. 5和1

随堂练习

- 36. 对于链式队列，在执行入队操作时（）
  - 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
- 37. 最适合用作链队的链表是（）
  - A. 只带队首指针的循环单链表
  - B. 只带队尾指针的循环单链表
  - C. 只带对手指针的非循环单链表
  - D. 只带队尾指针的非循环单链表

随堂练习

- 38. 数组A中，每个元素的长度为3B，行下标i从0-7，列下标j从0-9，从首地址开始连续存放在存储器内，则存放该数组至少需要的单元数是（）

  A. 80　　B. 100　　C. 240　　D. 270

- 39. 设二维数组A[6][10]，每个数组元素占4个存储单元，若按行优先顺序表存放的数组元素A[3][5]的存储地址是1000，求A[0][0]的存储地址是__860

数组（存储和寻址）

- 40. 假设三维数组D[3][3][4]的首地址是d, 则D[1][2][2]的首地址是

  Loc(D[1][2][2])

  = d+ i*n*p*C+j*p*C+k*C

  = d+(1*3*4+2*4+2)*4

  = d+88

随堂练习

- 在5维数组A[10][10][10][10][10]中，每个元素占4个字节。设该数组的首元素地址为13。
- 41-1: A[1][2][3][4][5]的存储地址是：
- 41-2: 存储首地址是397的元素是索引是：A[][][][][]

随堂练习

- 42. 设矩阵A和B分别如下所示，我们采用行优先存储一维数组C的结果存储矩阵A，B 相乘的结果，则C[5]的值是（54）

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \qquad B = \begin{matrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{matrix}$$

$$C = \begin{matrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{matrix}$$

随堂练习

- 43. 有对角矩阵M[10][10], 其中有元素M[5][5] = 1, M[2][1] = (0)
- 44.有对角矩阵M[10][10]，其非零元素的值都不一样。在M中有元素M[5][5] = 1. 将M存储在一个一维数组C中，则C的第（C）个元素的值是1

  A. 1　　B. 20　　C. 5　　D. 8

随堂练习

- 45. 设矩阵A是一个对称矩阵，为了节省存储空间，将其下三角部分按行优先存放在一维数组B中。对下三角部分的任意元素a_{ij}(i≥j，且i和j从1开始取值)，一维数组B中下标k的值是:

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

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

随堂练习

- 46. 稀疏矩阵的一般的存储方法有（）

  A. 二维数组和三维数组　　B. 三元组和散列

  C. 三元组和十字链表　　D. 散列和十字链表

随堂练习

- 47. 有稀疏矩阵T，将其转置存储在一维数组中，其转置为：

$$T=\begin{matrix}0&1&0&0\\9&0&0&0\\0&2&0&5\\0&0&0&0\end{matrix}$$

$$\begin{matrix}\mathbf{0}&\mathbf{9}&\mathbf{0}&\mathbf{0}\\1&0&2&0\\0&0&0&0\\0&0&5&0\end{matrix}$$

A. $\begin{matrix}\mathbf{0}&\mathbf{1}&\mathbf{1}\\1&0&9\\2&1&2\\2&3&5\end{matrix}$

B. $\begin{matrix}\mathbf{0}&\mathbf{1}&\mathbf{9}\\1&0&1\\1&2&2\\3&2&5\end{matrix}$

C. $\begin{matrix}\mathbf{1}&\mathbf{1}&\mathbf{0}\\1&0&9\\2&2&1\\2&3&5\end{matrix}$

D. $\begin{matrix}\mathbf{0}&\mathbf{1}&\mathbf{9}\\1&0&1\\2&2&2\\3&2&5\end{matrix}$

随堂练习

- 48. 空格串与空串是相同的，这种说法是（）

  A. 正确　　B. 错误

- 49. 两个串相等的充分必要条件是（）

  A. 两个串的长度相等

  B. 两串所包含的字符集合相等

  C. 两串长度相等且对应位置的字符相等

  D. 两串长度相等且字符集合相等

随堂练习

- 50. 字符串的两种存储方式是（）

  A. 顺序存储和链式存储的方式　　B. 顺序存储和堆存储的方式

  C. 堆存储和链式存储的方式　　D. 堆存储和数组存储的方式

- 51. 设字符串str1是”Demon”， str2是”_”, str3是“Hunter”, 则str1, str2, str3拼接后的结果是（）

  A. DemonHunter　　B. _DemonHunter

  C. DemonHunter_　　D. Demon_Hunter

随堂练习

- 52. 若字符串S = “software”, 其字串的数目是（） （备注：其中空串和字符串自身也可看作是字符串的字串）

  A. 8　　B. 37　　C. 39　　D. 9

- 53. 若字符串是S = “Data structures are fascinating.”， 字串“_are_fa”的位置是（） （备注: 下标从0开始，这里的”_”表示空格）。

  A. 12　　B. 15　　C. 14　　D. 16

随堂练习

- 54. 在数据结构中，树结构的定义是什么？

A. 一个有序的线性数据结构，每个元素都有两个指向子元素的指针

B. 一个非线性的数据结构，由节点组成，每个节点可以有零个或多个子节点。

C. 一个线性的数据结构，每个元素都有一个指向父元素的指针。

D. 一个包含所有数据元素的列表，没有节点和子节点的概念。

- 55. 树最适合用来表示

A. 有序元素　　B. 无序元素

C. 元素之间具有分支层次关系的数据

D. 元素之间无联系的数据

随堂练习

- 56. 一棵有n个结点的树的所有结点的度数之和为（）

  A. n – 1　　B. n　　C. n + 1　　D. 2n

随堂测试

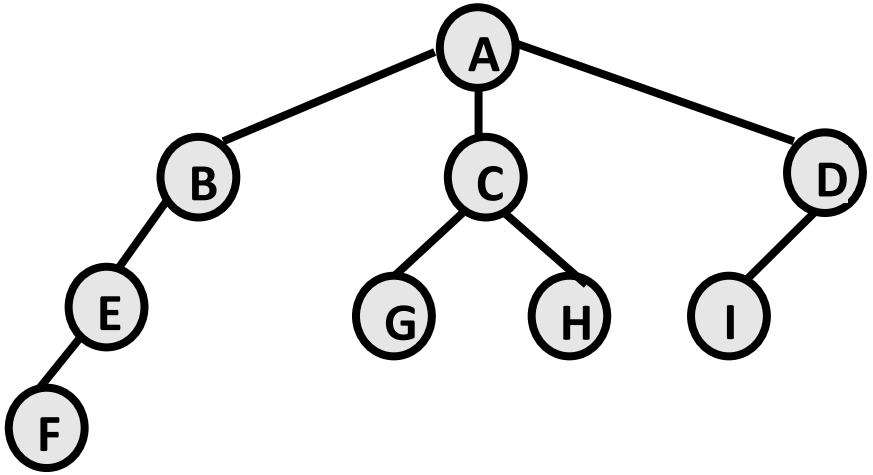
- 57. 有树如图右所示，以下不是根节点到叶子节点的路径的是：

  A. ABEF

  B. ACGH

  C. ACG

  D. ADI



随堂练习

- 58.给定一个普通树，该树的结点数为N，请选择下列哪个选项可以正确计算该树的高度？

  A. 遍历根节点的所有子节点，并递归计算每个子节点的高度，然后取最大值作为树的高度。

  B. 遍历根节点的所有子节点，并递归计算每个子节点的高度，然后取平均值作为树的高度。

  C. 从根节点开始，按层级遍历树中的所有节点，直到遍历完所有节点，所经过的层级即为树的高度。

  D. 统计树中所有节点的个数，减去1，得到树的高度。

随堂练习

- 59. 按照二叉树的定义，具有3个结点的二叉树有（）种

  A. 3　　B. 4　　C. 5　　D. 6

随堂练习

- 60. 某二叉树的第5层至多有（）个结点。（设根节点的层数是1）

  A. 8　　B. 16　　C. 32　　D. 64

- 61. 某二叉树树只有度为0和度为2的结点，且树高为8，则树的结点个数为（）(设根节点的层数是1)

  A. 127　　B. 261　　C. 68　　D. 不确定

随堂练习

- 62. 一棵高为4的二叉树，至少有（）个结点。设根节点的高度为0

  A . 5　　B. 6　　C. 7　　D. 4

- 63. 一棵高为5的二叉树，至多有（）个结点 。设根节点的高度为1

  A . 31　　B. 32　　C. 10　　D. 16

随堂练习

- 64.有10个叶子的二叉树中有（）个度为2的结点

  A. 8　　B. 9　　C. 10　　D. 11

- 65. 如果一棵二叉树有10个度为2的结点，5个度为1的结点，则度为0的结点的个数为（）

  A. 9　　B. 11　　C. 15　　D. 不确定

随堂练习

- 66. 一棵有124个叶结点的完全二叉树最多有（）个结点

  A. 247　　B. 248　　C. 249　　D. 250

- 67. 具有1000个结点的完全二叉树的次底层叶结点的个数为（）

  A. 11　　B. 12　　C. 24　　D. 36

随堂练习

- 68. 以下是一棵顺序存储的树，则编号为10的节点在树上的层数是（）。（根节点在的层数是0）

  A. 3　　B. 4　　C. 2　　D. 以上都不对

- 69. 以下是一棵顺序存储的完全二叉树，则编号为9的节点的双亲结点是编号为（）的结点。

  A. 5　　B. 4　　D. 6　　D. 2

结点值	*31*	*23*	*12*	*66*	*94*	*17*	*5*	*70*	*62*	*49*
结点编号	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*

二叉树链式存储

- 70. 按照链式存储的二叉树中，结点P有孩子的条件是（）。

A. P!=NULL　　B. P->right != NULL

C. P->right == 0　　D. p->right == 1

随堂测试

- 71 一棵二叉树的先序遍历序列为A、B、C、D、E、F，中序遍历序列为C、B、A、E、D、F，后序遍历顺序为（）

  A. C、B、E、F、D、A　　B. F、E、D、C、B、A

  D. C、B、E、D、F、A　　D. 不确定

- 72 一棵二叉树的后序遍历顺序为D、A、B、E、C，中序遍历序列为D、E、B、A、C。先序遍历顺序为（）

  A. A、C、B、E、D　　B. D、E、C、B、A

  C. D、E、A、B、C　　D. C、E、D、B、A

随堂练习

- 73. 以下是一段树的中根遍历的C语言代码，空格处应该填的代码是：

A. inorderTraversal(root->left);
printf("%d ", root->data);
inorderTraversal(root->right);

B. printf("%d ", root->data);
inorderTraversal(root->right);
inorderTraversal(root->left);

C. printf("%d ", root->data);

D. 以上都不对

```
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {

        ____________________________

    }
}
```

随堂练习

- 74. 以下是一段树的先根遍历的C语言代码，空格处应该填的代码是：

A.
```
inorderTraversal(root->left);
printf("%d ", root->data);
inorderTraversal(root->right);
```

B.
```
printf("%d ", root->data);
inorderTraversal(root->left);
inorderTraversal(root->right);
```

C.
```
inorderTraversal(root->right);
inorderTraversal(root->left);
printf("%d ", root->data);
```

D. 以上都不对

```
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        _____
    }
}
```

随堂练习

- 75. 以下是一段树的后根遍历的C语言代码，空格处应该填的代码是：

A.
```
inorderTraversal(root->right);
inorderTraversal(root->left);
printf("%d ", root->data);
```

B.
```
printf("%d ", root->data);
inorderTraversal(root->right);
inorderTraversal(root->left);
```

C.
```
inorderTraversal(root->left);
printf("%d ", root->data);
inorderTraversal(root->right);
```

D. 以上都不对

```
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
void inorderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        ____________________________
    }
}
```

随堂练习

- 76. 一棵二叉树的先序遍历序列和后序遍历顺序正好相反，则该二叉树一定是:

  A. 空树或者只有一个结点　　B. 完全二叉树

  C. 二叉排序树　　D. 高度等于其结点数

- 77. 在一棵非空二叉树的中序遍历序列中，根结点的右边（）

  A. 只有右子树上的所有结点　　B. 只有右子树上的部分结点

  C. 只有左子树上的部分结点　　D.只有左子树上的所有结点

随堂练习

- 78. 以下是一段非递归的中序遍历代码，空格部分应填入：

A. curr = stack[top++];
printf("%d ", curr->data);
curr = curr->right;

B. curr = stack[top++];
printf("%d ", curr->data);
curr = curr->left;

C. curr = stack[top--];
curr = curr->right;
printf("%d ", curr->data);

D. curr = stack[top--];
printf("%d ", curr->data);
curr = curr->right;

```
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
void inorderTraversal(struct TreeNode*
root) {
    struct TreeNode* stack[100];
    int top = -1;
    struct TreeNode* curr = root;
    while (curr != NULL || top != -1) {
        while (curr != NULL) {
            stack[++top] = curr;
            curr = curr->left;
        }
        ____________________________
    }
}
```

随堂练习

- 79.已知一算术表达式的中缀（中序）形式为A+BxC-D/E，后缀（后序）形式为ABCx+DE/-，其前缀形（）

  A. -A+BxC/DE

  B. -A+BxCD/E

  C. -+xABC/DE

  D. -+AxBC/DE

(返回71, 72题)

随堂练习

- 80 . 以下是二叉树按层遍历的代码，空格处的代码段是（）

A. x = queue[rear--]

B. printf("%d ", current->data);

C. x = queue[rear++]

D. x = queue[front++]

```
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};
struct Node* queue[100];
void levelOrderTraversal(struct Node* root) {
    if (root == NULL) return;
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        struct Node* current = queue[front++];

        ____________________________________

        if (current->left != NULL)
            queue[rear++] = current->left;
        if (current->right != NULL)
            queue[rear++] = current->right;
    }
}
```

复习

- 复习1 给定二叉树如图所示，N表示根，L表示根节点的左子树，R表示根节点的右子树，若遍历序列是3,1,7,5,6,2,4, 则遍历方式是（）

  A . LRN　　B. NRL　　C. RLN　　D. RNL

- 复习题2 已知一棵完全二叉树的第6层（根为第一层）有8个叶结点，则完全二叉树结点的个数有（）个。

  A. 39　B. 52　C. 111　D. 119

复习

- 复习3 设一棵二叉树的前序遍历为abdecf, 后序为dbfeac，则中序为

  A. adbecf　　B. dfecab　　C. dbeacf　　D. abcdef

- 复习4 设一棵二叉树的中序遍历为badce, 后序为bdeca，则前序序为

  A. adbec　　B. decab　　C. debac　　D. abcde

- 复习5 设一棵二叉树的前序遍历为abdefc, 后序为dbfeac，则前序序为

  A. dbfeac　　B. dfebca　　C. bdfeca　　D. bdefac

随堂练习

- 81. 现有数组[1, 2, 3, 4, 5]和一段创建二叉树的代码，若要把二叉树建成下图所示的树，空白处应该填写的代码是：

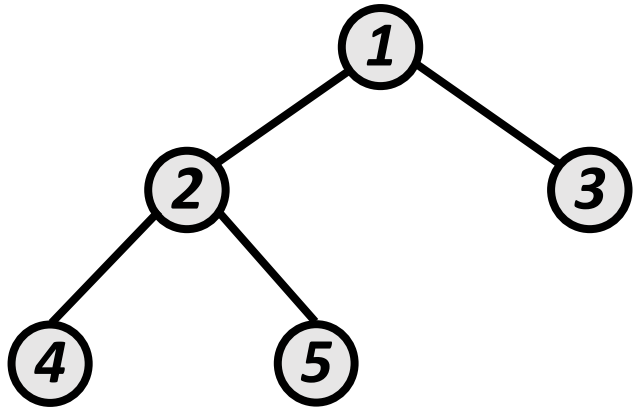
```
struct Node* createBinaryTree() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    ____________________________
    return root;
}
```

A. root->left->left = createNode(4);
   root->right->right = createNode(5);

B. root->right->left = createNode(4);
   root->right->right = createNode(5);

C. root->left->left = createNode(4);
   root->left->right = createNode(5);

D. root->left->right = createNode(4);
   root->right->left= createNode(5);



随堂练习

- 82. 以下是一段按照后根遍历复制二叉树的代码，空白处应该填写的代码是：

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

```
struct TreeNode* copyTree(struct TreeNode* root) {
    if (root == NULL) {
        return NULL;
    }


    return new_root;
}
```

① struct **TreeNode*** new_root =
createNode(root->val);
② new_root->right = new_right;
③ new_root->left = new_left;
④ struct **TreeNode*** new_right =
copyTree(root->right);
⑤ struct **TreeNode*** new_left =
copyTree(root->left);

随堂练习

- 83. 引入二叉线索数的目的是（）
  A. 加快查找结点的前驱或后继的速度
  B. 为了能在二叉树中方便地进行插入和删除
  C. 为了能方便地找到双亲
  D. 使二叉树的遍历结果唯一

X D

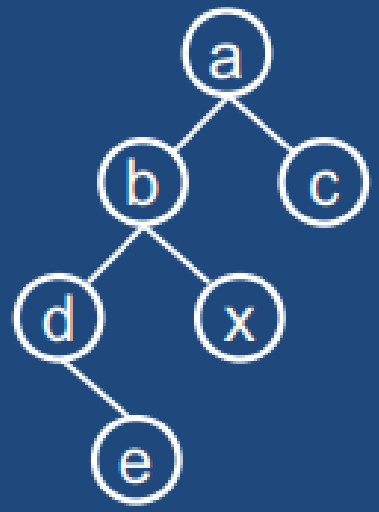
随堂练习

- 84. 判断线索二叉树中p结点有孩子结点的条件是（）

  A. p!=NULL　　B. P->rchild != NULL　　C. p->rtag == 0　D. p-rtag == 1

- 85. 若对图所示的二叉树进行中序线索化，则结点x的左右线索指向的结点分别是（）

  A. c,c　　B. c,a　　C. d,c　　D. b,a



C

随堂练习

- 86.以下是一段查找线索二叉树中中根序的第一个结点的代码，空白处应该填写的是（）

A. for(root->left != NULL) {
root = root->left;}

B. print(root);

C. while (root->left != NULL) {
root = root->left;}

D. 以上都不正确

```
struct TreeNode* createNode(int val) {
struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct
TreeNode));
newNode->val = val;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
struct TreeNode* findFirstNode(struct
TreeNode* root) {
if (root == NULL) {
return NULL;
}
 ____________________________
return root;
}
```

A

随堂练习

- 87. 以下是一段查找线索二叉树中中根序的最后一个结点的代码，空白处应该填写的是（）

A. while (root->right != NULL) {
root = root->right;}

B. print(root);

C. while (root->left != NULL) {
root = root->left;}

D. 以上都不正确

```
struct TreeNode* createNode(int val) {
struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct
TreeNode));
newNode->val = val;
newNode->left = NULL;
newNode->right = NULL;
return newNode;
}
struct TreeNode* findFirstNode(struct
TreeNode* root) {
if (root == NULL) {
return NULL;
}
```

```
return root;
}
```

C

随堂练习

- 88. 以下是中序二叉线索树中查找某节点前驱的部分代码，空白处应该填入的代码段是（）

A.
```
while (pre->rtag == 0)
  pre = pre->left;
```
B.
```
print(node)
```
C.
```
while (pre->rtag == 0)
  pre = pre->right;
```
D.
```
while (pre->ltag == 0)
  pre = pre->left;
```

```
BiThrNode *inBiSearchPre(BiThrNode *node)
{
    BiThrNode *pre;
    pre = node->left;
    if (node->ltag != 1)
    {
        ____________________________
    }
    return pre;
}
```

D

随堂练习

- 89. 以下是中序二叉线索树中查找某节点后继的部分代码，空白处应该填入的代码段是（）

A.
```
while (post->rtag == 0)
  post = post->left;
```
B. print(node)

C.
```
while (post->rtag == 0)
  post = post->right;
```
D.
```
while (post->ltag == 0)
  post = post->left;
```

```
BiThrNode *inBiSearchPost(BiThrNode *node)
{
    BiThrNode *post;
    post = node->right;
    if (node->rtag != 1 {

    ____________________________

    }
    return post;
}
```

A

随堂练习

- 90. 以下是后序线索二叉树中求解某结点p的后序首结点的部分代码，空白处应填入（）

A. return p->right;
else return p->left;
B. return p->right;
C. return p->right->left;
else return p->left->left;
D. 以上都不对

```
struct Node {
    int data;
    struct Node* left; struct Node* right;
    int LThread; int RThread;
};
struct Node* findPredecessor(struct Node* p)
{
    if (IsFirst(p)) return NULL;
    else{
        if (p->LThread == 1) return p->left;
        else{
            if (RThread == 0)

        }
    }
}
```

A C

随堂练习

- 91. 在后序线索二叉树中，结点P的位置如图1所示，则P的后继是（）

  A. P的父结点

  B. 父结点的右子树后序遍历的第一个结点

  C.父结点的左子树后序遍历的第一个结点

  D. 无后继

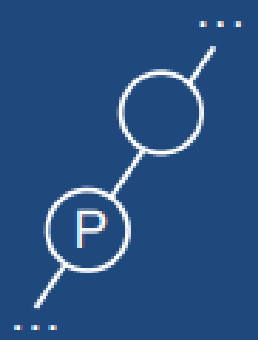


图1

- 92. 在先序线索二叉树中，结点P的位置如图2所示，则P的前驱是（）

  A. P的父结点

  B.父结点的右子树先序遍历的第一个结点

  C.父结点的左子树先序遍历的最后一个结点

  D. 无后继

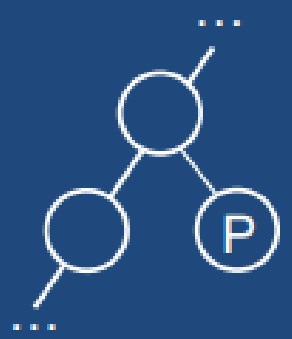


图2

C

随堂练习

- 93. 对中序线索二叉树进行遍历，其遍历的思路是（）

A. 先确定先序首结点，再依次查找其先序后继结点

B. 先确定中序首结点，再依次查找其后序后继结点

C. 先确定中序首结点，再依次查找其中序后继结点

D.先确定中序首结点，再依次查找其先序后继结点

A

随堂练习

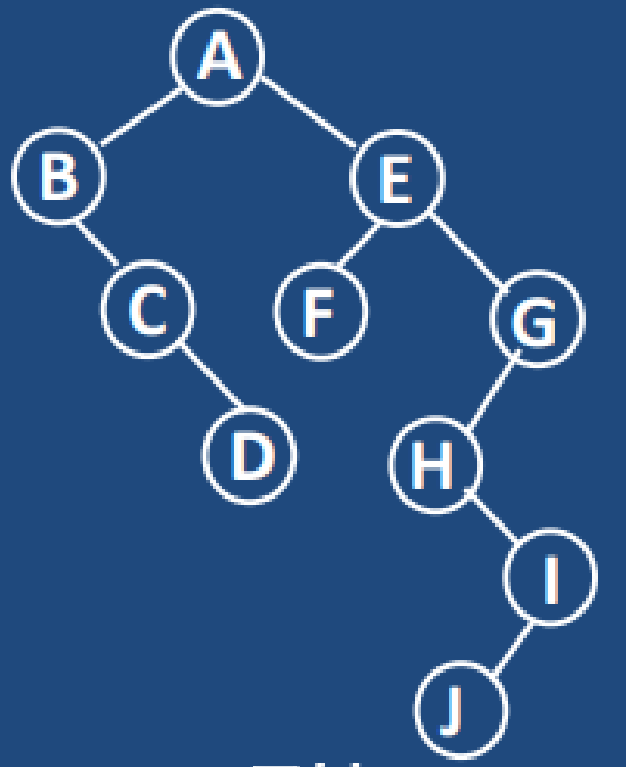
- 94. 某二叉树如图所示，在其对应的中序线索二叉树中将结点P作为结点E的左孩子，并让结点E的左孩子作为P的左孩子，则P的对应的左指针和右指针分别指向()

  A. F， E

  B. F, NULL

  C. E， F

  D. F， H



二叉树

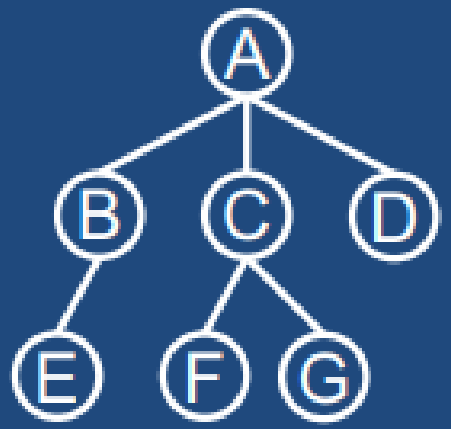
C A

随堂练习

- 95. 有如下结构的树，将其转化成二叉树后对应的先序遍历的结果是（）

  A. ABCDEFG　B. ABCEFDG

  C. ABECFGD　D. ACBDFGE



- 96. 如果T_2是由树T转化成的二叉树，那么T中结点的先根遍历序列对应T_2中结点地那的（）遍历序列。

  A. 前序　B. 中序　C. 后序　D. 层次序

B

随堂练习

- 97. 将森林转化成对应的二叉树，若在二叉树中，结点u是结点v的父结点的父结点，则在原来的森林中，u和v的关系可能是（）

I. 父子关系　II. 兄弟关系　III. u的父结点与v的父结点是兄弟关系

A. 只有II　B. I和II　C. I 和III　D. I、II和III

B B

随堂练习

- 98. 由如图1所示的二叉树，将其转化成树后，结点D的左孩子和右孩子分别是（）

  A. C、G　　B. G、NULL

  C. F、G　　D. NULL、G

- 99.由如图2所示的二叉树，将其转化成森林后，结点E的孩子分别是（）

  A. F、G　　B. 无孩子

  C. F　　D. G

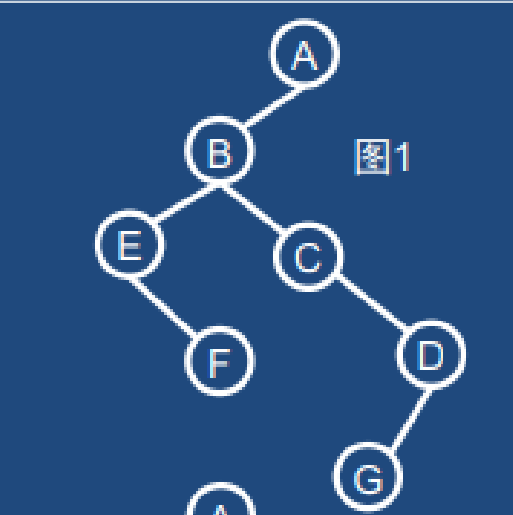


图1

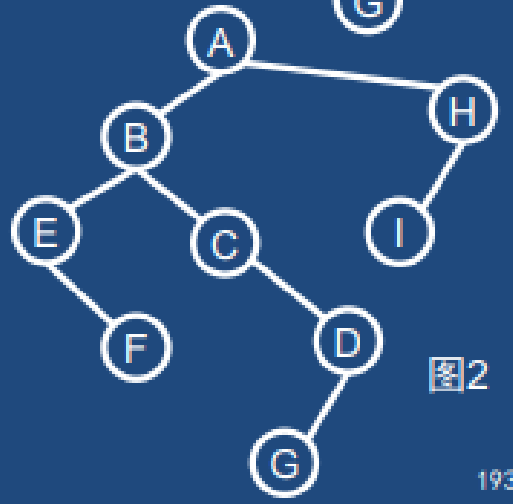


图2

2023/12/14　　193

B

随堂练习

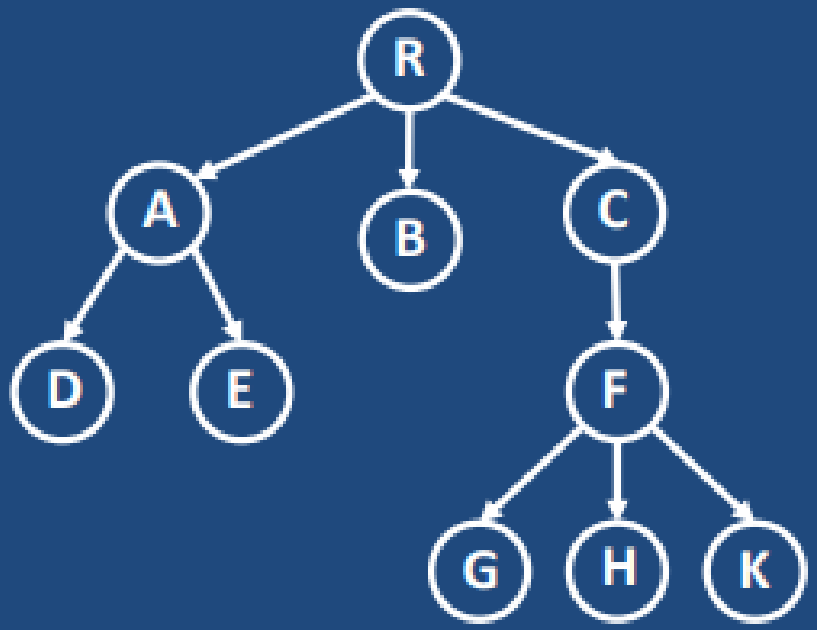
- 100. 如图所示的树的先根序对应的节点次数序列是（）

A. 0 0 2 0 0 0 0 3 1 0

B. 3 2 0 0 0 1 3 0 0 0

C. 3 2 1 0 0 0 0 3 0 0

D. 以上都不对



b

随堂练习

- 101. 关于树的存储，不正确的是（）

A. 用树上结点的层次顺序和子节点的下标对树进行顺序存储可能会造成很大的空间浪费

B. 根据先根遍历的顺序在不引入其他信息的基础上存储树没有歧义

C. 孩子-兄弟的链接存储结果和先把树转化成二叉树再进行链接存储的结果一样

D. 只存储父结点的结构不利于对整棵树进行遍历

a

随堂练习

- 102.如下是一段查找树中某结点的父节点的代码。空白处应该填写

A. `TreeNode* parent = find_parent(child, target);`
B. `TreeNode* parent = find_parent(child->right, target);`
C. `TreeNode* parent = find_parent(child, target ->right);`
D. `printf("%d", root);`

```
TreeNode* find_parent(TreeNode* root, TreeNode* target) {
    if (root == NULL) return NULL;
    for (int i = 0; i < root->count; i++) {
        TreeNode* child = root->child[i];
        if (child == target) {
            return root;
        } else {
            _____________________________________________
            if (parent != NULL) return parent;
        }
    }
    return NULL;
}
```

D

随堂练习

- 103.如下是一段用于从一棵树上找到值为value的节点的代码。空白处应该填写（）

A. return NULL;
B. TreeNode* parent = find_parent(child->right, target);
C. TreeNode* parent = find_parent(child, target ->right);
D. return root;

```
TreeNode* searchValue(TreeNode* root, int value)
{
    if (root == NULL) {
        return NULL;
    }
    if (root->value == value) {
        _____
    }
    for (int i = 0; i < root->count; i++) {
        TreeNode* result = searchValue(
                    root->child [i], value);
        if (result != NULL) {
            return result;
        }
    }
    return NULL;
}
```

C

- 104.如下是一段用于从一棵树上删除值为value的节点的代码，成功返回True，否则返回False，则空白处应该填写（）

```
bool deleteNode(Node* root, target) {
    Node* target = searchNode(root, id);
    if(target == NULL) return False;
    else if(target == root){
        root = NULL;
        return True;
    }
    else{
        for(i = 0; i < root->count; i++){
            if(target == root->child[i]->data){
                for(j = i+1; j< root->count; j++)
                ____________________________;
            }
        }
        for(i = 0; i < root->count; i++)
            deleteNode(root->child[i], target);

    }
}
```

A. return True;

B. root->child[i] = root->child[i+1];

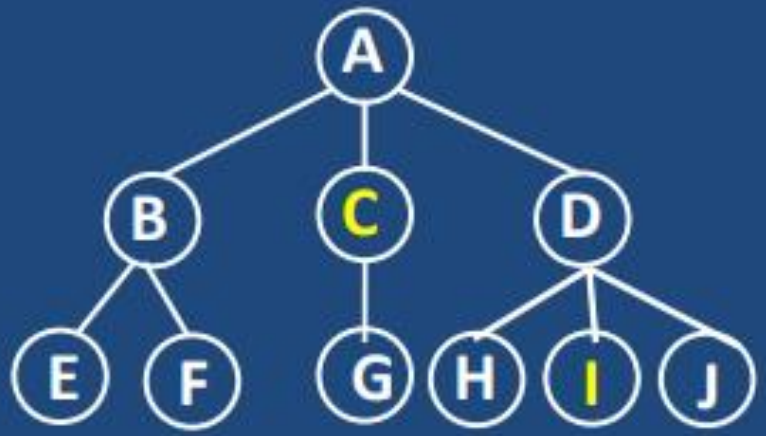
C. root->child[j-1] = root->child[j];
return True;

D. root->child[j] = root->child[j+1];
return True;

A BEF CG DHIJ

随堂练习

- 105. 有如下树，其先根遍历是（）
- 106. 有如下树，其后根遍历是（）

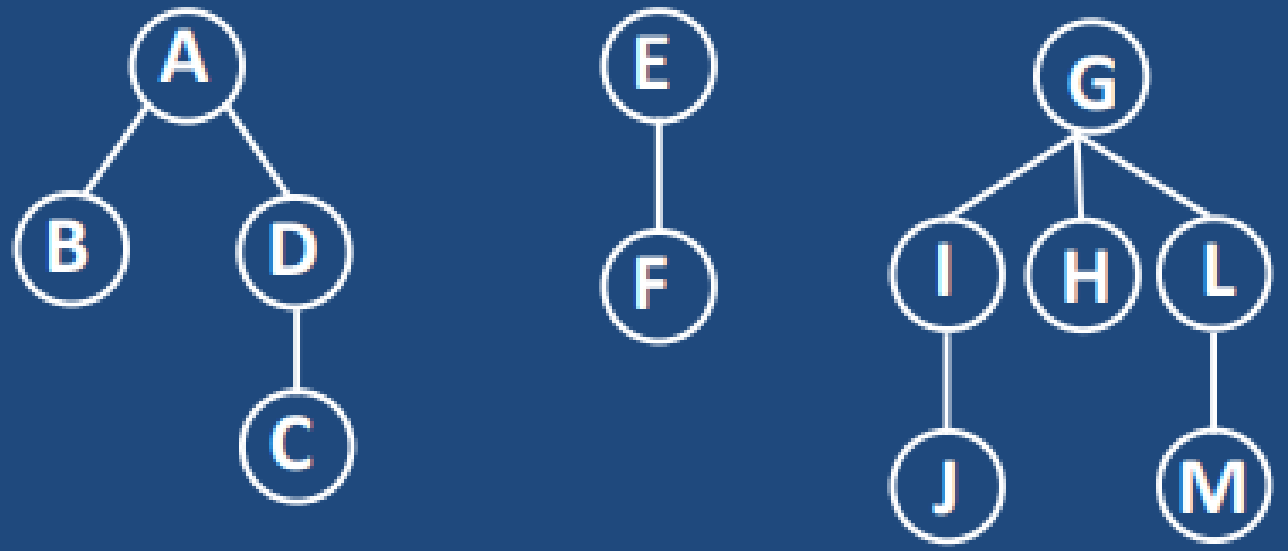


2023/12/14

234

随堂练习

- 107. 有如下森林，其先根遍历顺序是（）
- 108. 有如下森林，其对应的二叉树的中序遍历结果是（）



D

随堂练习

- 109. 如下是一段用于后根遍历树上所有结点的代码，空白处应该填写的是（）

A.
```
printf("%d ", t->data)
for(i = 0; i < t->count; i++)
    preorder(t);
```
B.
```
printf("%d ", t->data)
```
C.
```
for(i = 0; i < t->count; i++)
    printf("%d ", t->data)
    preorder(t);
```
D.
```
for(i = 0; i < t->count; i++)
    preorder(t);
printf("%d ", t->data)
```

```
void preorder(Node* t){
   if(t == NULL) return
   else{

   ____________________________

}
```

B

随堂练习

- 110. 如下是一段用于后根遍历树上所有结点的代码，空白处应该填写的是（）

A.
```
for(i = 0; i < t->count; i++)
    Visit_Le(t->child[i]);
```
B.
```
head = head ->next;
if (head != NULL)
    Visit_Le(head);
```
C.
```
Visit_Le(head);
```
D.
```
for(i = 0; i < t->count; i++)
    printf("%d ", t->data)
```

```
// tail 指向Q的队尾， head指向t
void Visit_Le(Node* t){
    if(t == NULL) return
    else{
        printf("%d ", t->data);
        for(i = 0; i < t->count; i++){
            tail->next = t->child[i];
            tail = tail->next;
        }

        ____________________________
    }
    }
}
```

C D

随堂练习

- 111. 由权值8, 4, 5, 7的4个叶结点构造的一棵哈夫曼树，该树的带权路径长度为（）

  A. 24　　B. 36　　C. 48　　D. 72

- 112. 假设哈夫曼二叉树中只有度为0或者2的结点，有n个叶子的哈夫曼树的结点总数为（）

  A. 不确定　　B. 2n　　C. 2n+1　　D. 2n - 1

D

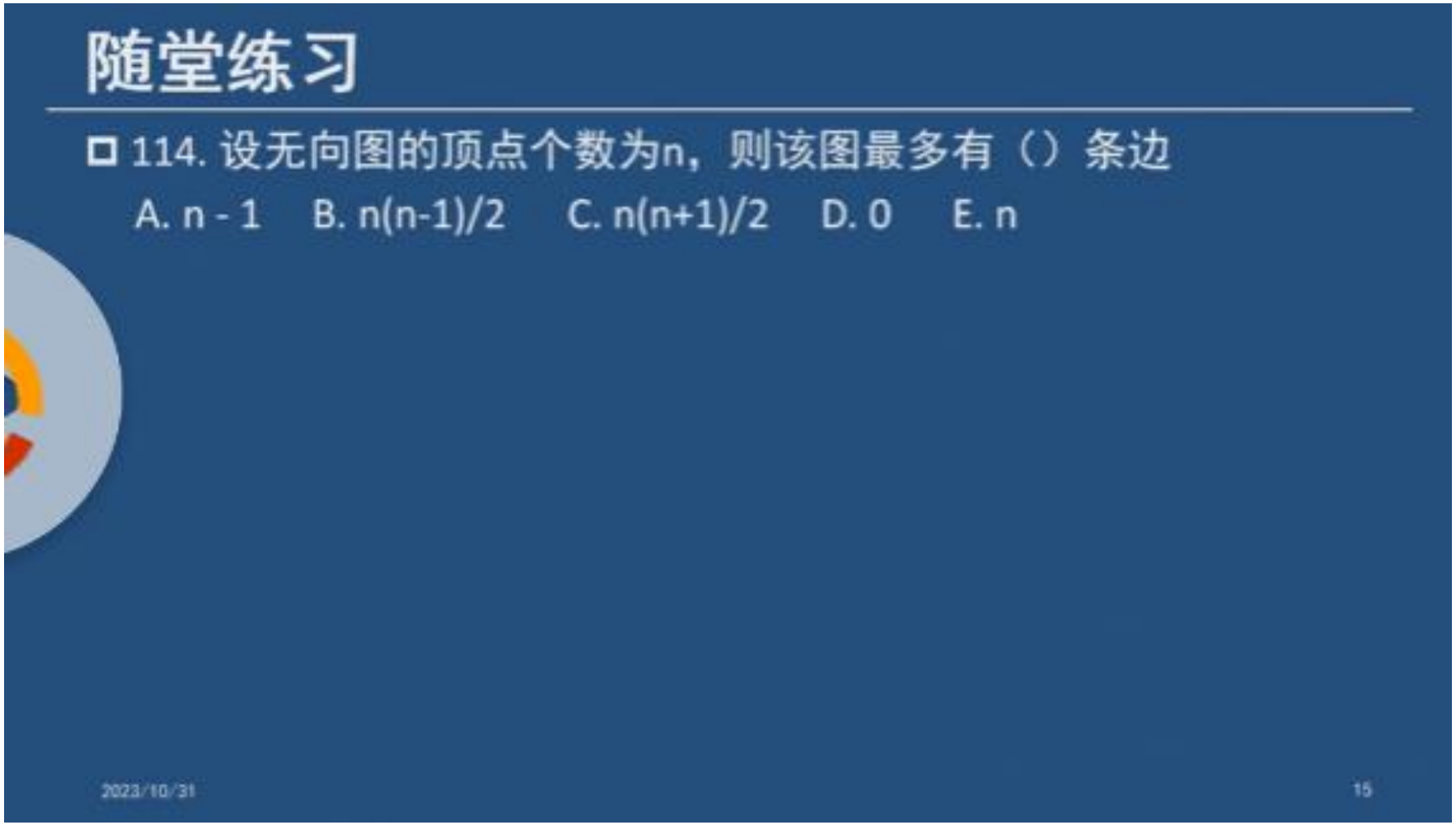
随堂练习

- 113. 假设哈夫曼二叉树中只有度为0或者2的结点，根据使用频率为5个字符设计的哈夫曼编码，不可能的是（）

A. 000, 001, 010, 011, 1　　B. 0000, 0001, 001, 01, 1

C. 000, 001, 01, 10, 11　　D. 00, 100, 101, 110, 111

B


随堂练习
114. 设无向图的顶点个数为n，则该图最多有（）条边
A. n - 1 B. n(n-1)/2 C. n(n+1)/2 D. 0 E. n
2023/10/31
15

B C

随堂练习

- 115. 在一个无向图中，所有顶点的度数之和等于所有边的（）倍。

  A. 1/2　　B. 2　　C. 1　　D. 4

- 116. 在一个有向图中，所有顶点的入度之和等于所有顶点出度之和的（）倍

  A. 1/2　　B. 2　　C. 1　　D. 4

2023/10/31　　20

A

随堂练习

- 117. 图中有关路径的定义的是（）
  - A. 由相邻点序偶所形成的序列
  - B. 由不同顶点所形成的序列
  - C. 由不同的边所形成的序列
  - D. 上述定义都不是

2023/10/31 25

A B

随堂练习

- 118. 含有n个顶点的联通无向图，其边的个数至少为（）。

  A. n－1　B. n　C. n＋1　D. $n\log_2 n$

- 119. 一个由n个结点的无向图，最少有（）个联通分量。

  A. 0　B. 1　C. n - 1　D. n

B b D

随堂练习

有邻接矩阵A = $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

- 120. 该图共有（）顶点

  A. 9　B. 3　C. 6　D. 1　E.以上均不对

- 121. 若是有向图，该图共有（）条边

  A. 5　B. 4　C. 3　D. 2　E.以上均不对

- 122. 若是无向图，则共有（）条边

  A. 5　B. 4　C. 3　D. 2　E.以上均不对

B

随堂练习

- 123.在下列有关图的存储结构的说法中错误的是( )。

A.用邻接矩阵存储一个图时所占用的存储空间大小与图中的顶点个数有关，而与图的边数无关

B.邻接表只能用于有向图的存储，邻接矩阵对于有向图和无向图的存储都适用

C.邻接矩阵适用于稠密图(边数接近于顶点数的二次方)，邻接表适用于稀疏图(边数远小于顶点数的二次方)

D.对同一个有向图来说，邻接表中的边结点数与逆邻接表中的边结点数相等

D

随堂练习

- 124. 已知无向图G用邻接矩阵存储，现在结点下标为u和v之间增加一条边。则相应的代码是（）

A.
```
void add_edge(int v, int u) {
    if (v < graph_size && u < graph_size)
        edge[u][v] = 1;
}
```

B.
```
void add_edge(int v, int u) {
    if (v < graph_size && u < graph_size)
        edge[v][u] = 1;
}
```

C.
```
void add_edge(int v, int u) {
    if (v < graph_size && u < graph_size)
        edge[u][v] = 0;
        edge[v][u] = 0;
}
```

D.
```
void add_edge(int v, int u) {
    if (v < graph_size && u < graph_size){
        edge[u][v] = 1;
        edge[v][u] = 1;
        }
}
```

A

随堂练习

- 125.已知有向图G用邻接表存储，现在删除下标为u的结点到下标为v的结点的边。则空白处的代码是（）

A. pre->next = pre->next->next;
return true;

B. pre->next = NULL

C. return true;

D. 以上都不对

```
bool deleteEdge(struct Node** head, int u,
        int v, int graph_size) {
    if (u < graph_size && v < graph_size){
        pre = head[u];
        while(pre->next != NULL &&
                pre->next->data != v)
            pre = pre->next;
        if (pre->next == NULL)
            return false;
        else{
            ____________________
        }
    }
    return false;
}
```

D D

随堂练习

- 126.无向图G=(V，E)，其中:V={a, b, c, d, e, f}，E={(a, b), (a, e), (a, c), (b, e), (c, f), (f, d), (e, d)}，以顶点a为源点，对该图进行深度优先遍历，得到的顶点序列正确的是()。

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

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

- 127.如图所示，在下面的5个序列中，符合深度优先遍历的序列有()。序列为:

a, e, b, d, f, c;　a, c, f, d, e, b;　a, e, d, f, c, b;

a, e, f, d, c, b;　a, e, f, d, b, c

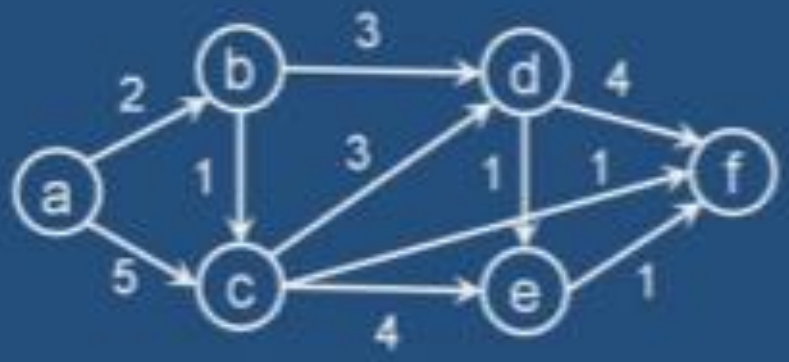
A.5个　　B.4个　　C.3个　　D.2个

A、

随堂练习

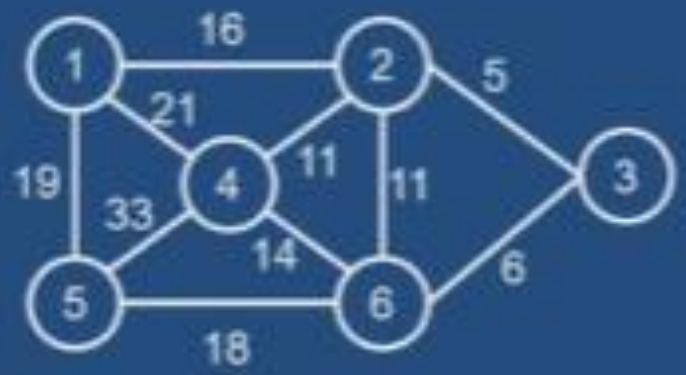
- 137. 对如图所示的有向带权图，若采用Dijkstra算法求源点a到其他各个顶点的最短路径，则得到的第一条最短路径的目标顶点是b，第二条最短路径的目标是c，后续得到的其余最短路径的目标顶点依次是（）

A. d, e, f　　B. e, d, f　　C. f, d, e　　D. f, e, d



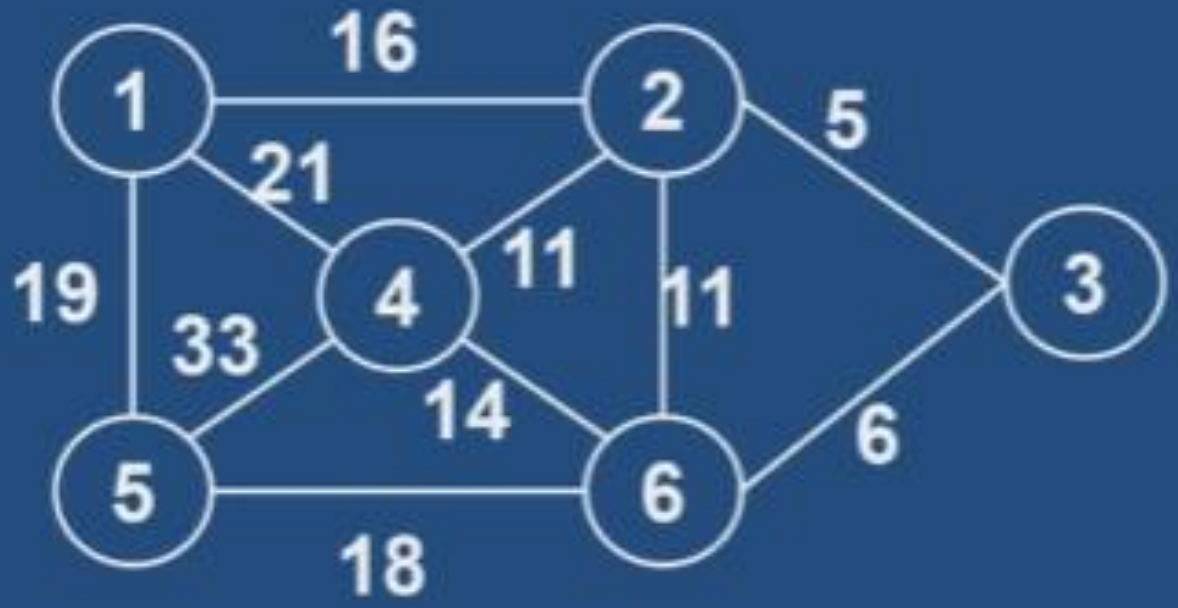
随堂练习

- 以下是一个地区的通信网，边表示城市之间的通信线路，边上的权表示假设线路花费的代价。如何选择能联通每个城市且总代价最小，试画出可能的线路



随堂练习

- 以下是一个地区的通信网，边表示城市之间的通信线路，边上的权表示假设线路花费的代价。如何选择能联通每个城市且总代价最小，试画出可能的线路



2023/11/20

218

随堂练习

- 141.对线性表进行二分查找时，要求线性表必须(　)。

  A.以顺序方式存储　　B.以链表方式存储

  C.以顺序方式存储，且关键字要有序

  D.以链表方式存储，且关键字要有序

- 142.有一个有序表为{1，3，9，12，32，41，45，62，75，77，82，95，99}，采用二分查找法，找到82共进行(　)比较。

  A.1次　　B.2次　　C.4次　　D.8次

随堂练习

- 143.在一棵二叉查找树上，查找关键字为35的结点，依次比较的关键字可能是（）

A. 28, 36, 18, 46, 35　　B. 18, 36, 28, 46, 35

C. 46, 28, 18, 36, 35　　D. 46, 36, 18, 28, 35

随堂练习

- 144.设有一组记录的关键字为{19，14，23，1, 68，20，84，27，55，11，10，79}，用链地址法构造Hash表，Hash函数为H(key)=key Mod 13，散列地址为1的链中有()个关键字。

  A.1　B.2　C.3　D. 4

随堂练习

- Hash表的地址区间为0~16，Hash函数为H(K)=K mod 17.采用线性探查法处理冲突，并将关键字序列{26，25，72，38，8，18，59}依次存储到Hash表中。

145.关键字59存放在Hash表中的地址是()。

A.8　　B.9　　C. 10　　D.11

146.存放关键字59需要探查的次数是()。

A.2　　B.3　　C.4　　D.5

随堂练习

- 147.设 Hash 表长为14,Hash函数是H(key)=key Mod11，表中已有数据的关键字为15、38、61、84，共4个，现要将关键字为49的结点添加到表中，用平方探查法解决冲突，则放入的位置是()。

  A.8　　B.3　　C. 5　　D.9

若存在冲突，按照$d+1^2$，$d-1^2$，$d+2^2$，$d-2^2$，...进行探测