CCF 编程能力等级认证
Grade Examination of Software Programming

Python 六级

2025 年 03 月

1 单选题（每题 2 分，共 30 分）

题号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
答案	D	B	A	B	B	B	B	A	A	A	B	A	C	D	D

第 1 题 在面向对象编程中，类是一种重要的概念。下面关于类的描述中，不正确的是（ ）。

- ☐ **A.** 类是一个抽象的概念，用于描述具有相同属性和行为的对象集合。
- ☐ **B.** 类可以包含属性和方法，属性用于描述对象的状态，方法用于描述对象的行为。
- ☐ **C.** 类可以被实例化，生成具体的对象。
- ☐ **D.** 类一旦定义后，其属性和方法不能被修改或扩展。

第 2 题 哈夫曼编码是一种用于数据压缩的算法。以下关于哈夫曼编码的描述中，不正确的是（ ）。

- ☐ **A.** 哈夫曼编码是一种变长编码，频率高的字符使用较短的编码，频率低的字符使用较长的编码。
- ☐ **B.** 在构造哈夫曼树时，频率越低的字符离根节点越近，频率越高的字符离根节点越远。
- ☐ **C.** 哈夫曼编码的生成过程基于贪心算法，每次选择频率最低的两个节点进行合并。
- ☐ **D.** 哈夫曼编码是一种前缀编码，任何一个字符的编码都不会是另一个字符编码的前缀，因此可以实现唯一解码。

第 3 题 以下代码实现了树的哪种遍历方式?

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def traverse(root):
    if root is None:
        return
    print(root.val, end=" ")
    traverse(root.left)
    traverse(root.right)

```

- ☐ **A.** 前序遍历
- ☐ **B.** 中序遍历

- [ ] **C.** 后序遍历

- [ ] **D.** 层次遍历

第 4 题 以下关于完全二叉树的代码描述，正确的是（ ）。

```python
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def is_complete_tree(root):
    if root is None:
        return True

    q = deque()
    q.append(root)
    has_null = False

    while q:
        node = q.popleft()

        if node is None:
            has_null = True
        else:
            if has_null:
                return False
            q.append(node.left)
            q.append(node.right)

    return True

```

- [ ] **A.** 该代码用于判断一棵树是否为满二叉树

- [ ] **B.** 该代码用于判断一棵树是否为完全二叉树

- [ ] **C.** 该代码用于判断一棵树是否为二叉搜索树

- [ ] **D.** 该代码用于计算树的高度

第 5 题 以下代码实现了二叉排序树的哪种操作?

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def op(root, val):
    if root is None:
        return TreeNode(val)
    if val < root.val:
```

```
        root.left = op(root.left, val)
    else:
        root.right = op(root.right, val)
    return root
```

- [ ] **A.** 查找
- [ ] **B.** 插入
- [ ] **C.** 删除
- [ ] **D.** 遍历

第 6 题 给定字符集 `{A, B, C, D}` 的出现频率分别为 `{5, 1, 6, 2}`，则正确的哈夫曼编码是（ ）。

- [ ] **A.** `A: 0, B: 100, C: 11, D: 101`
- [ ] **B.** `A: 11, B: 100, C: 0, D: 101`
- [ ] **C.** `A: 0, B: 101, C: 11, D: 100`
- [ ] **D.** `A: 10, B: 101, C: 0, D: 100`

第 7 题 关于动态规划的描述，正确的是（ ）。

- [ ] **A.** 动态规划算法的时间复杂度总是低于贪心算法。
- [ ] **B.** 动态规划要求问题必须具有最优子结构和重叠子问题两个性质。
- [ ] **C.** 动态规划通过递归实现时不需要存储中间结果。
- [ ] **D.** 动态规划的核心思想是将问题分解为互不重叠的子问题。

第 8 题 以下代码中，类的构造函数被调用了（ ）次。

```
import copy
class MyClass:
    def __init__(self):
        print("Constructor called!")
if __name__ == "__main__":
    obj1 = MyClass()
    obj2 = copy.deepcopy(obj1)
```

- [ ] **A.** 1
- [ ] **B.** 2
- [ ] **C.** 3
- [ ] **D.** 0

第 9 题 以下代码中function1()实现了循环队列的哪种操作?

```
class CircularQueue:
    def __init__(self, k):
        self.size = k
        self.arr = [0] * k
        self.front = -1
```

```
        self.rear = -1

    def is_full(self):

        return (self.rear + 1) % self.size == self.front

    def is_empty(self):

        return self.front == -1

    def fuction1(self, value):
        if self.is_full():
            return False
        if self.is_empty():
            self.front = 0
        self.rear = (self.rear + 1) % self.size
        self.arr[self.rear] = value
        return True
```

- [ ] **A.** 入队
- [ ] **B.** 出队
- [ ] **C.** 查看队首元素
- [ ] **D.** 判断队列是否为空

第 10 题 以下代码实现了二叉树的深度优先搜索（DFS），并统计了叶子节点的数量。横线上应填写（ ）。

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def count_leaf_nodes(root):
    if root is None:
        return 0

    stack = []
    stack.append(root)
    count = 0

    while stack:
        node = stack.pop()
        if node.left is None and node.right is None:
            count += 1
        if node.right:
            stack.append(node.right)
        if node.left:
            ____________________

    return count
```

- [ ] **A.** `stack.append(node.left)`

☐ **B.** `stack.append(node.right)`

☐ **C.** `stack.delete(node.left)`

☐ **D.** `stack.delete(node.right)`

第 11 题 以下代码实现了二叉树的广度优先搜索（BFS），并查找特定值的节点。横线上应填写（ ）。

```
from collections import deque

class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def find_node(root, target):
    if root is None:
        return None

    q = deque()
    q.append(root)

    while q:
        current = q.popleft()

        if current.val == target:
            return current
        ____________________

    return None
```

☐ **A.**

```
    if current.left:
            q.append(current.right)
    if current.right:
            q.append(current.left)
```

☐ **B.**

```
    if current.left:
            q.append(current.left)
    if current.right:
            q.append(current.right)
```

☐ **C.**

```
    if current.left:
            q.append(current)
    if current.right:
            q.append(current.right)
```

☐ **D.**

```
    if current.left:
            q.append(current.left)
    if current.right:
            q.append(current)
```

第 12 题 以下代码用于生成 n 位格雷编码。横线上应填写（ ）。

```
def generate_gray_code(n):
    if n == 0:
        return ["0"]
    if n == 1:
        return ["0", "1"]

    prev = generate_gray_code(n - 1)

    result = ["0" + s for s in prev]
    _____________________________

    return result
```

- [ ] **A.** `result += ["1" + s for s in reversed(prev)]`
- [ ] **B.** `result += ["0" + s for s in reversed(prev)]`
- [ ] **C.** `result += ["1" + s for s in reversed(1)]`
- [ ] **D.** `result += ["1" + s for s in reversed(0)]`

第 13 题 以下代码实现了0/1背包问题的动态规划解法。假设物品重量为 `weights[]`，价值为 `values[]`，背包容量为 `W`，横线上应填写（ ）。

```
def knapsack(W, weights, values):
    n = len(weights)
    dp = [[0] * (W + 1) for _ in range(n + 1)]

    for i in range(1, n + 1):
        for j in range(1, W + 1):
            if weights[i - 1] > j:
                dp[i][j] = dp[i - 1][j]
            else:
                dp[i][j] = max(_____________________________________)

    return dp[n][W]
```

- [ ] **A.** `dp[i][j], dp[i - 1][j - weights[i - 1]] + values[i - 1]`
- [ ] **B.** `dp[i - 1][j], dp[i][j - weights[i - 1]] + values[i - 1]`
- [ ] **C.** `dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1]`
- [ ] **D.** `dp[i - 1][j], dp[i - 1][j - weights[i]] + values[i]`

第 14 题 以下代码用于检查字符串中的括号是否匹配，横线上应填写（ ）。

```
def is_balanced(s):
    stack = []
```

```
    for c in s:
        if c in '([{':
            stack.append(c)
        else:
            if not stack:
                return False
            top = stack.pop()

            if (c == ')' and top != '(') or \
               (c == ']' and top != '[') or \
               (c == '}' and top != '{'):
                return False
    ____________________
```

- [ ] **A.** `True`
- [ ] **B.** `False`
- [ ] **C.** `return stack`
- [ ] **D.** `return not stack`

第 15 题 给定一个二叉排序树（BST），其中节点的值均为正整数。以下关于BST的说法中，错误的是：

- [ ] **A.** 对BST进行中序遍历，得到的序列一定是有序的。
- [ ] **B.** 在BST中查找一个值为k的节点，最坏情况下需要遍历整棵树。
- [ ] **C.** 向BST中插入一个新节点，可能会破坏BST的平衡性。
- [ ] **D.** 删除BST中的一个节点后，树的高度一定不会增加。

2　判断题（每题 2 分，共 20 分）

题号	1	2	3	4	5	6	7	8	9	10
答案	√	×	×	√	×	×	√	√	√	√

第 1 题 哈夫曼树在构造过程中，每次合并权值最小的两个节点，最终生成的树带权路径和最小。

第 2 题 格雷编码的相邻两个编码之间必须有多位不同，以避免数据传输错误。

第 3 题 在树的深度优先搜索（DFS）中，使用队列作为辅助数据结构以实现“先进后出”的访问顺序。

第 4 题 以下代码实现的是二叉树的中序遍历：

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def traverse(root):
    if root is None:
        return
    traverse(root.left)
    print(root.val, end=" ")
    traverse(root.right)
```

第 5 题 python可以直接定义多个构造函数，但默认无参数的构造函数只能有一个。

第 6 题 二叉排序树（BST）中，若某节点的左子树为空，则该节点一定是树中的最小值节点。

第 7 题 在动态规划解决一维硬币找零问题时，若硬币面额为 `[1, 3, 4]`，目标金额为 `6`，则最少需要 2 枚硬币（3+3）。

第 8 题 面向对象编程中，封装是指将数据和行为绑定在一起，并对外隐藏实现细节。

第 9 题 以下代码创建的树是一棵完全二叉树：

```python
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
```

第 10 题 栈和队列均可以用双向链表实现，插入和删除操作的时间复杂度为 O(1) 。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.1.1 树上漫步

3.1.2 题目描述

小 A 有一棵 n 个结点的树，这些结点依次以 $1, 2, \ldots, n$ 标号。

小 A 想在这棵树上漫步。具体来说，小 A 会从树上的某个结点出发，每一步可以移动到与当前结点相邻的结点，并且小 A 只会在偶数步（可以是零步）后结束漫步。

现在小 A 想知道，对于树上的每个结点，从这个结点出发开始漫步，经过偶数步能结束漫步的结点有多少个（可以经过重复的节点）。

3.1.3 输入格式

第一行，一个正整数 n。

接下来 $n-1$ 行，每行两个整数 u_i, v_i，表示树上有一条连接结点 u_i 和结点 v_i 的边。

3.1.4 输出格式

一行，n 个整数，第 i 个整数表示从结点 i 出发开始漫步，能结束漫步的结点数量。

3.1.5 样例

3.1.5.1 输入样例 1

```
3
1 3
2 3
```

3.1.5.2 输出样例 1

```
2 2 1
```

3.1.5.3 输入样例 2

```
4
1 3
3 2
4 3
```

3.1.5.4 输出样例 2

```
3 3 1 3
```

3.1.6 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 10^3$。

对于所有测试点，保证 $1 \leq n \leq 2 \times 10^5$。

3.1.7 参考程序

```python
n = int(input())
to = [[] for i in range(n + 2)]
for i in range(n - 1):
    x, y = map(int, input().split())
    to[x].append(y)
    to[y].append(x)

cnt = [0, 0]
color = [0 for i in range(n + 2)]
def dfs(x, c, f):
    color[x] = c
    cnt[c] += 1
    for y in to[x]:
        if y != f:
            dfs(y, c ^ 1, x)

dfs(1, 0, 0)
ans = []
for i in range(1, n + 1):
    ans.append(str(cnt[color[i]]))

print(' '.join(ans))
```

3.2 编程题 2

- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.2.8 环线

3.2.9 题目描述

小 A 喜欢坐地铁。地铁环线有 n 个车站，依次以 $1, 2, \ldots, n$ 标号。车站 i（$1 \leq i < n$）的下一个车站是车站 $i+1$。特殊地，车站 n 的下一个车站是车站 1。

小 A 会从某个车站出发，乘坐地铁环线到某个车站结束行程，这意味着小 A 至少会经过一个车站。小 A 不会经过一个车站多次。当小 A 乘坐地铁环线经过车站 i 时，小 A 会获得 a_i 点快乐值。请你安排小 A 的行程，选择出发车站与结束车站，使得获得的快乐值总和最大。

3.2.10 输入格式

第一行，一个正整数 n，表示车站的数量。

第二行，n 个整数 $a_1, a_2, \ldots, a_n$，分别表示经过每个车站时获得的快乐值。

3.2.11 输出格式

一行，一个整数，表示小 A 能获得的最大快乐值。

3.2.12 样例

3.2.12.5 输入样例 1

```
4
-1 2 3 0
```

3.2.12.6 输出样例 1

```
5
```

3.2.12.7 输入样例 2

```
5
-3 4 -5 1 3
```

3.2.12.8 输出样例 2

```
5
```

3.2.13 数据范围

对于 20% 的测试点，保证 $1 \leq n \leq 200$。

对于 40% 的测试点，保证 $1 \leq n \leq 2000$。

对于所有测试点，保证 $1 \leq n \leq 2 \times 10^5$，$-10^9 \leq a_i \leq 10^9$。

3.2.14 参考程序

```python
n = int(input())
a = [0] + list(map(int, input().split()))
pre = [0 for i in range(2 * n + 5)]

for i in range(1, n + 1):
    a.append(a[i])

for i in range(1, 2 * n + 1):
    pre[i] = pre[i - 1] + a[i]

ql, qr, ans = 1, 1, -1e18
q = [0 for i in range(2 * n + 5)]

for i in range(1, 2 * n + 1):
    while ql <= qr and q[ql] < i - n:
        ql += 1
    ans = max(ans, pre[i] - pre[q[ql]])
    while ql <= qr and pre[i] < pre[q[qr]]:
        qr -= 1
    qr += 1
    q[qr] = i

print(ans)
```