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

Python 六级

2025 年 06 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	A	B	B	B	B	D	D	A	A	A	D	A	B	C

第 1 题 下列哪一项不是面向对象编程（OOP）的基本特征？

- [ ] **A.** 继承 (Inheritance)
- [ ] **B.** 封装 (Encapsulation)
- [ ] **C.** 多态 (Polymorphism)
- [ ] **D.** 链接 (Linking)

第 2 题 为了让 `Dog` 类的构造函数能正确地调用其父类 `Animal` 的构造方法，横线线处应填入（ ）。

```
class Animal:
    def __init__(self, name: str):

        self.name = name
        print("Animal created")

    def speak(self) -> None:
           print("Animal speaks")

class Dog(Animal):
    ____________________________
        print("Dog created")

    def speak(self) -> None:
        print("Dog barks")

if __name__ == "__main__":
    animal: Animal = Dog("Rex", "Labrador")
    animal.speak()
```

- [ ] **A.**

```
def __init__(self, name: str, breed: str):
        super().__init__(name)
        self.breed = breed
```

- [ ] **B.**

```
def __init__(self, name: str, breed: str):
    self.breed = breed
```

- [ ] **C.**

```
def __init__(self, name: str, breed: str):
```

- [ ] **D.**

```
    self.breed = breed
```

第 3 题 代码同上一题，代码animal.speak()执行后输出结果是（ ）。

- [ ] **A.** 输出 Animal speaks
- [ ] **B.** 输出 Dog barks
- [ ] **C.** 编译错误
- [ ] **D.** 程序崩溃

第 4 题 以下Python代码执行后其输出是（ ）。

```
from collections import deque
stack = []
queue = deque()

# 元素入栈/入队 (1, 2, 3)
for i in range(1, 4):
    stack.append(i)
    queue.append(i)
print(f"{stack[-1]} {queue[0]}")
```

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

第 5 题 在一个使用列表实现的循环队列中，front 表示队头元素的位置（索引），rear 表示队尾元素的下一个插入位置（索引），队列的最大容量为 maxSize。那么判断队列已满的条件是（ ）

- [ ] **A.** `rear == front`
- [ ] **B.** `(rear + 1) % maxSize == front`
- [ ] **C.** `(rear - 1 + maxSize) % maxSize == front`
- [ ] **D.** `(rear - 1) == front`

第 6 题 在二叉树中，只有最底层的节点未被填满，且最底层节点尽量靠左填充的是（ ）。

- [ ] **A.** 完美二叉树
- [ ] **B.** 完全二叉树
- [ ] **C.** 完满二叉树
- [ ] **D.** 平衡二叉树

第 7 题 在使用数组表示完全二叉树时，如果一个节点的索引为i（从0开始计数），那么其左子节点的索引通常是（ ）。

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

第 8 题 已知一棵二叉树的前序遍历序列为 `GDAFEMHZ`，中序遍历序列为 `ADFGEHMZ`，则其后序遍历序列为（ ）。

- ☐ **A.** `ADFGEHMZ`
- ☐ **B.** `ADFGHMEZ`
- ☐ **C.** `AFDGEMZH`
- ☐ **D.** `AFDHZMEG`

第 9 题 设有字符集 {a, b, c, d, e}，其出现频率分别为 {5, 8, 12, 15, 20}，得到的哈夫曼编码为（ ）。

- ☐ **A.**

```
a: 010
b: 011
c: 00
d: 10
e: 11
```

- ☐ **B.**

```
a: 00
b: 10
c: 011
d: 100
e: 111
```

- ☐ **C.**

```
a: 10
b: 01
c: 011
d: 100
e: 111
```

- ☐ **D.**

```
a: 100
b: 01
c: 011
d: 100
e: 00
```

第 10 题 3位格雷编码中，编码 101 之后的下一个编码是（ ）。

- ☐ **A.** 100

☐ **C.** 110

☐ **D.** 001

第 11 题 请将下列 Python 实现的深度优先搜索（DFS）代码补充完整，横线处应填入（ ）。

```
from typing import List, Optional

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

def dfs_preorder(root: Optional[TreeNode], result: List[int]) -> None:
    if root is None:
        return

    _____
```

☐ **A.**

```
    result.append(root.val)
    dfs_preorder(root.left, result)
    dfs_preorder(root.right, result)
```

☐ **B.**

```
    result.append(root.val)
    dfs_preorder(root.left, result)
```

☐ **C.**

```
    result.append(root.val)
    dfs_preorder(root.right, result)
```

☐ **D.**

```
    dfs_preorder(root.left, result)
    dfs_preorder(root.right, result)
```

第 12 题 给定一个二叉树，返回每一层中最大的节点值，结果以数组形式返回，横线处应填入（ ）。

```
from collections import deque
import math
from typing import List, Optional

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

def largestValues(root: Optional[TreeNode]) -> List[int]:

    result = []
    if not root:
        return result

    queue = deque([root])

```

```
    while queue:
        level_size = len(queue)
        max_val = -math.inf

        for _ in range(level_size):
            _____________________

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        result.append(max_val)

    return result
```

- [ ] **A.**

```
node = queue.popright()
max_val = max(max_val, node.val)
```

- [ ] **B.**

```
node = queue.popleft()
```

- [ ] **C.**

```
max_val = max(max_val, node.val)
```

- [ ] **D.**

```
node = queue.popleft()
max_val = max(max_val, node.val)
```

第 13 题 下面代码实现一个二叉排序树的插入函数（没有相同的数值），横线处应填入（ ）。

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

def insert(root, key):

    if root is None:
        return TreeNode(key)

    ___________________________

    return root
```

- [ ] **A.**

```
  if key < root.val:
       root.left = insert(root.left, key)
   elif key > root.val:
       root.right = insert(root.right, key)
```

- [ ] **B.**

```
    if key > root.val:
        root.left = insert(root.left, key)
    elif key > root.val:
        root.right = insert(root.right, key)
```

- [ ] **C.**

```
    if key < root.val:
        root.left = insert(root.left, key)
    elif key >= root.val:
        root.right = insert(root.left, key)
```

- [ ] **D.**

```
    if key < root.val:
        root.left = insert(root.right, key)
    elif key > root.val:
        root.right = insert(root.left, key)
```

第 14 题 以下关于动态规划算法特性的描述，正确的是（ ）。

- [ ] **A.** 子问题相互独立，不重叠
- [ ] **B.** 问题包含重叠子问题和最优子结构
- [ ] **C.** 只能从底至顶迭代求解
- [ ] **D.** 必须使用递归实现，不能使用迭代

第 15 题 给定n个物品和一个最大承重为W的背包，每个物品有一个重量$wt[i]$和价值$val[i]$，每个物品只能选择放或不放。目标是选择若干个物品放入背包，使得总价值最大，且总重量不超过W。关于下面代码，说法正确的是（ ）。

```
def knapsack1D(W: int, wt: list[int], val: list[int], n: int) -> int:
    dp = [0] * (W + 1)

    for i in range(n):
        for w in range(W, wt[i] - 1, -1):
            dp[w] = max(dp[w], dp[w - wt[i]] + val[i])

    return dp[W]
```

- [ ] **A.** 该算法不能处理背包容量为 `0` 的情况
- [ ] **B.** 外层循环 `i` 遍历背包容量，内层遍历物品
- [ ] **C.** 从大到小遍历 `w` 是为了避免重复使用同一物品
- [ ] **D.** 这段代码计算的是最小重量而非最大价值

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 构造函数只能自动不可以被手动调用。

第 2 题 给定一组字符及其出现的频率，构造出的哈夫曼树是唯一的。

第 3 题 为了实现一个队列，使其出队操作（`pop`）的时间复杂度为$O(1)$并且避免数组删除首元素的$O(n)$问题，一种常见且有效的方法是使用环形数组，通过调整队首和队尾指针来实现。

第 4 题 对一棵从小到大的二叉排序树进行中序遍历，可以得到一个递增的有序序列。

第 5 题 如果二叉搜索树在连续的插入和删除操作后，所有节点都偏向一侧，导致其退化为类似于链表的结构，这时其查找、插入、删除操作的时间复杂度会从理想情况下的$O(logn)$退化到$O(nlogn)$。

第 6 题 执行下列代码，`my_dog.name` 的最终值是 `Charlie`。

```python
class Dog:
    def __init__(self, name):
        self.name = name

if __name__ == "__main__":
    my_dog = Dog("Buddy")
    my_dog.name = "Charlie"
```

第 7 题 下列 python 代码可以成功执行，并且子类 `Child` 的实例能通过其成员函数访问父类 `Parent` 的属性 `value`。

```python
class Parent:
    def __init__(self):
        self._value = 100

class Child(Parent):
    def get_protected_val(self):
        return self._value
```

第 8 题 下列代码中的 `tree` 列表，表示的是一棵完全二叉树 (`-1` 代表空节点)按照层序遍历的结果。

```python
tree = [1, 2, 3, 4, -1, 6, 7]
```

第 9 题 在树的深度优先搜索（DFS）中，可以使用栈作为辅助数据结构以实现“先进后出”的访问顺序。

第 10 题 下面代码采用动态规划求解零钱兑换问题：给定n种硬币，第 i 种硬币的面值为 $coins[i - 1]$，目标金额为 amt，每种硬币可以重复选取，求能够凑出目标金额的最少硬币数量；如果不能凑出目标金额，返回 `-1`。

```python
def coinChangeDPComp(coins: list[int], amt: int) -> int:
    n = len(coins)
    MAX = amt + 1


    dp = [MAX] * (amt + 1)
    dp[0] = 0


    for i in range(1, n + 1):
        for a in range(1, amt + 1):
            if coins[i - 1] > a:
                dp[a] = dp[a]
            else:

                dp[a] = min(dp[a], dp[a - coins[i - 1]] + 1)

    return dp[amt] if dp[amt] != MAX else -1
```

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：学习小组
- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.1.1　题目描述

班主任计划将班级里的 n 名同学划分为若干个学习小组，每名同学都需要分入某一个学习小组中。观察发现，如果一个学习小组中恰好包含 k 名同学，则该学习小组的讨论积极度为 a_k。

给定讨论积极度 $a_1, a_2, \ldots, a_n$，请你计算将这 n 名同学划分为学习小组的所有可能方案中，讨论积极度之和的最大值。

3.1.2　输入格式

第一行，一个正整数 n，表示班级人数。

第二行，n 个非负整数 $a_1, a_2, \ldots, a_n$，表示不同人数学习小组的讨论积极度。

3.1.3　输出格式

输出共一行，一个整数，表示所有划分方案中，学习小组讨论积极度之和的最大值。

3.1.4　样例

3.1.4.1　输入样例 1

```
4
1 5 6 3
```

3.1.4.2　输出样例 1

```
10
```

3.1.4.3　输入样例 2

```
8
0 2 5 6 4 3 3 4
```

3.1.4.4　输出样例 2

```
12
```

3.1.5　数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 10$。

对于所有测试点，保证 $1 \leq n \leq 1000$，$0 \leq a_i \leq 10^4$。

3.1.6 参考程序

```python
# 首先记录同学总数
n = int(input())
# 记录一个小组有k个同学的情况下，讨论积极度
a = [0] + list(map(int, input().split()))

# 0个同学的时候，讨论度为0
dp = [0]
# 遍历同学总数k
for k in range(1, n + 1):
    # i,j表示将k个同学分解为i个同学和j个同学两部分
    dp.append(0)
    for i in range(0, k):
        j = k - i
        # 核心状态转移公式
        dp[k] = max([a[k], dp[i] + dp[j], d[k]])
print(dp[n])

```

3.2 编程题 2

- **试题名称**：最大因数
- **时间限制**：6.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

给定一棵有 10^9 个结点的有根树，这些结点依次以 $1,2,\ldots,10^9$ 编号，根结点的编号为 1。对于编号为 k（$2 \le k \le 10^9$）的结点，其父结点的编号为 k 的因数中除 k 以外最大的因数。

现在有 q 组询问，第 i（$1 \le i \le q$）组询问给定 x_i, y_i，请你求出编号分别为 x_i, y_i 的两个结点在这棵树上的距离。两个结点之间的距离是连接这两个结点的简单路径所包含的边数。

3.2.2 输入格式

第一行，一个正整数 q，表示询问组数。

接下来 q 行，每行两个正整数 x_i, y_i，表示询问结点的编号。

3.2.3 输出格式

输出共 q 行，每行一个整数，表示结点 x_i, y_i 之间的距离。

3.2.4 样例

3.2.4.1 输入样例 1

```
3
1 3
2 5
4 8
```

3.2.4.2 输出样例 1

```
1
2
1
```

3.2.4.3 输入样例 2

```
1
120 650
```

3.2.4.4 输出样例 2

```
9
```

3.2.5 数据范围

对于 60% 的测试点，保证 $1 \leq x_i, y_i \leq 1000$。

对于所有测试点，保证 $1 \leq q \leq 1000$，$1 \leq x_i, y_i \leq 10^9$。

3.2.6 参考程序

```python
# 读取询问的组数
n = int(input())

def factorize(x):
    """
    得到x以及其全部祖先节点的编号列表，从大到小排列，第一个是x，最后一个是根节点
    :param x: 输入节点的编号
    :return 一个列表，如上所述
    """
    f = [x]
    # 遍历能够整除x的最小的数，除掉这个数之后剩下的数就是x最大的因数了
    for i in range(2, int(x ** 0.5) + 1):
        # 不断除以这个数，直到不能除尽
        while x % i == 0:
            x //= i
            f.append(f[-1] // i)
    if f[-1] != 1:
        f.append(1)
    return f

for _ in range(n):
    # 接受这一组输入的x, y
    x, y = map(int, input().split())

    a = factorize(x)
    b = factorize(y)
    p1, p2 = 0, 0
    # 找到x和y到根节点的路径上最大的公共节点
    while a[p1] != b[p2]:
        if a[p1] > b[p2]:
            p1 += 1
        else:
            p2 += 1
    print(p1 + p2)
```