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

Python 七级

2025 年 12 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	C	A	D	C	B	D	B	B	B	B	C	C	B	B	C

第 1 题 下面关于Python中形参、实参和作用域的说法中，错误的一项是（ ）。

- ☐ **A.** 形参是函数定义时声明的参数，仅在函数内部的作用域中有效
- ☐ **B.** 实参是调用函数时传递给函数的实际值，会按对应关系绑定到形参
- ☐ **C.** 函数内部修改形参的值，一定会影响外部实参的取值（无论实参类型）
- ☐ **D.** 若函数内要修改全局作用域的变量，需通过 `global` 关键字声明

第 2 题 已知三个序列：`s1 = {3, 1, 8, 2, 5, 6, 7, 4}`，`s2 = {1, 5, 1, 8, 6, 4, 7, 5, 6}`，`s3 = {1, 8, 3, 5, 7, 6, 2, 4}`。以下哪个序列是它们的最长公共子序列（ ）。

- ☐ **A.** `{1, 8, 5, 6}`
- ☐ **B.** `{1, 5, 6, 7}`
- ☐ **C.** `{1, 8, 6}`
- ☐ **D.** `{1, 5, 7, 4}`

第 3 题 现有一个地址区间为 $0 \sim 10$ 的哈希表，当出现冲突情况，会往后找第一个空的地址存储（到 10 冲突了就从 0 开始往后），现在要依次存储 $(1, 3, 5, 7, 9)$，哈希函数为 $h(x) = (x^2 + x) \bmod 11$。其中 9 存储在哈希表哪个地址中（ ）。

- ☐ **A.** 1
- ☐ **B.** 2
- ☐ **C.** 3
- ☐ **D.** 4

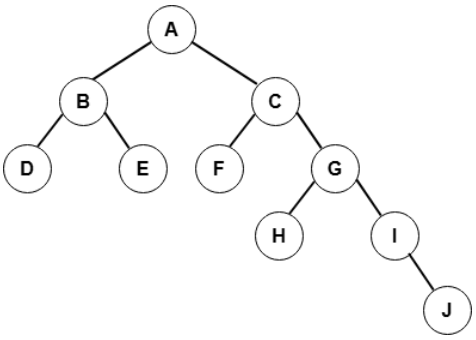
第 4 题 在0/1背包问题中，给定一组物品，每个物品有一个重量和价值，背包的容量有限。假设背包的最大容量为 W，物品的数量为n，其中第i个物品的重量为$w[i]$，价值为$v[i]$。以下关于0/1背包问题的描述，正确的是（ ）。

- ☐ **A.** 在解决0/1背包问题时，使用贪心算法可以保证找到最优解，因为物品只能放入一次。
- ☐ **B.** 0/1背包是P问题（多项式时间可解问题），它可以在 $O(nW)$ 的时间复杂度内解决。
- ☐ **C.** 0/1背包问题中，动态规划解法的空间复杂度为 $O(nW)$，但可以通过滚动数组技巧将空间复杂度优化到 $O(W)$。
- ☐ **D.** 0/1背包问题中，每个物品只能选择一次，并且子问题之间是独立的，无法重用计算结果。

第 5 题 一棵深度为 6（根节点深度为1）的完全二叉树，节点总数最少有（ ）。

- [ ] **A.** 31
- [ ] **B.** 32
- [ ] **C.** 63
- [ ] **D.** 64

第 6 题 对于如下二叉树，下面关于访问的顺序说法错误的是（ ）。



- [ ] **A.** `D E B F H J I G C A` 是它的后序遍历序列。
- [ ] **B.** `A B C D E F G H I J` 是它的广度优先遍历序列。
- [ ] **C.** `A B D E C F G H I J` 是它的先序遍历序列。
- [ ] **D.** `D B E A F C H G J I` 是它的中序遍历序列。

第 7 题 下面程序的运行结果为（ ）。

```python
def query(n, a, x):
    left = 0
    right = n
    while left < right:
        mid = left + (right - left) // 2
        if a[mid] >= x:
            right = mid
        else:
            left = mid + left
    if left == n:
        return -1
    return left

if __name__ == "__main__":
    n = 10
    x = 3
    num = [1, 2, 2, 3, 3, 4, 5, 5, 6, 7]
    result = query(n, num, x)
    print(result)
```

- [ ] **A.** `2`
- [ ] **B.** `3`
- [ ] **C.** `4`
- [ ] **D.** `5`

第 8 题 下面程序中，函数 `query` 的时间复杂度是（ ）。

```python
def query(n, a, x):

    left = 0
    right = n
    while left < right:
        mid = left + (right - left) // 2
        if a[mid] >= x:
            right = mid
        else:
            left = mid + 1
    if left == n:
        return -1
    return left

if __name__ == "__main__":
    n = 10
    x = 3
    num = [1, 2, 2, 3, 3, 4, 5, 5, 6, 7]
    print(query(n, num, x))
```

- [ ] **A.** $O(1)$
- [ ] **B.** $O(\log n)$
- [ ] **C.** $O(n)$
- [ ] **D.** $O(n \log n)$

第 9 题 有5个字符，它们出现的次数分别为2次、2次、3次、3次、5次。现在要用哈夫曼编码的方式来为这些字符进行编码，最小加权路径长度WPL（每个字符的出现次数×它的编码长度，再把每个字符结果加起来）的值为（ ）。

- [ ] **A.** 30
- [ ] **B.** 34
- [ ] **C.** 43
- [ ] **D.** 47

第 10 题 下面程序的运行结果为（ ）。

```python
def f(n):

    if n <= 2:
        return n * 2
    return f(n - 1) + f(n - 2)

if __name__ == "__main__":
       print(f(5))
```

- [ ] **A.** `10`
- [ ] **B.** `16`
- [ ] **C.** `26`
- [ ] **D.** `30`

第 11 题 一个简单无向图 G 有36条边，且每个顶点的度数都为4，则图 G 的顶点个数为（ ）。

- [ ] **A.** 9
- [ ] **B.** 12
- [ ] **C.** 18

- [ ] **D.** 36

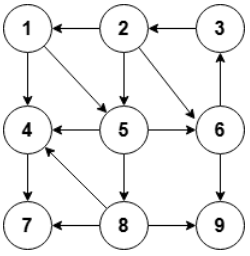
第 12 题 下面关于二叉树的说法正确的是（ ）。

- [ ] **A.** 任意二叉树的中序遍历与后序遍历必定不相同。
- [ ] **B.** 对任意二叉树，若已知先序遍历与后序遍历，则该二叉树唯一确定。
- [ ] **C.** 若二叉树有 n 个结点，根节点高度为 1，则其高度满足：$\lceil \log_2(n+1) \rceil \leq h \leq n$。
- [ ] **D.** 在二叉树的先序遍历中，根后紧跟的结点一定是根的左孩子。

第 13 题 假设一个算法时间复杂度的递推式是 $T(n) = 8T(\frac{n}{4}) + n\sqrt{n}$（$n$ 为正整数），和 $T(0) = 1$，那么这个算法的时间复杂度是（ ）。

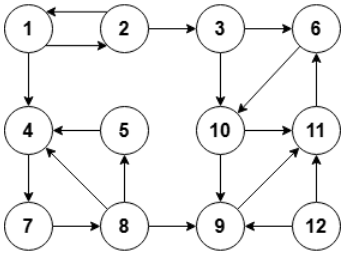
- [ ] **A.** $O(n\sqrt{n})$
- [ ] **B.** $O(n\sqrt{n}\log n)$
- [ ] **C.** $O(n^2)$
- [ ] **D.** $O(n^2 \log n)$

第 14 题 下面哪一个可能是下图的深度优先遍历序列（ ）。



- [ ] **A.** 1, 5, 6, 3, 2, 8, 9, 4, 7
- [ ] **B.** 1, 5, 8, 9, 7, 4, 6, 3, 2
- [ ] **C.** 3, 2, 1, 4, 7, 6, 9, 5, 8
- [ ] **D.** 2, 5, 6, 3, 8, 7, 9, 4, 1

第 15 题 下面这个有向图的强连通分量的个数是（ ）。



- [ ] **A.** 3
- [ ] **B.** 4
- [ ] **C.** 5
- [ ] **D.** 6

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 Python语言中，表达式 `3 ^ 2` 的结果类型为 `int` ，值为 `1` 。

第 2 题 使用 `math` 模块中的正弦函数，表达式 `math.sin(90)` 的结果类型为 `double` ，值约为 `1` 。

第 3 题 使用Python的字符串比较方式对比 `"10"` 和 `"9"` ，表达式 `"10" > "9"` 的结果为 `True`

第 4 题 选择排序是一种不稳定的排序算法，而冒泡排序是一种稳定的排序算法。

第 5 题 求两个长度为 n 序列的最长公共子序列（LCS）长度时，可以使用滚动数组将空间复杂度从 $O(n^2)$ 优化到 $O(n)$。

第 6 题 在无向图中，所有顶点的度数之和等于边数的两倍。

第 7 题 使用邻接矩阵存储一个有 V 个顶点、E 条边的图，对该图进行一次完整的BFS遍历，时间复杂度为 $O(V+E)$。

第 8 题 在图像处理或游戏开发中，泛洪（flood fill）算法既可以用BFS实现，也可以用DFS实现。

第 9 题 使用链地址法处理冲突的哈希表，当所有元素都映射到同一个槽位时，查找操作的最坏时间复杂度为 $O(n)$ ，其中 n 为元素个数。

第 10 题 一个包含 V 个顶点的连通无向图，其任何一棵生成树都恰好包含 $V-1$ 条边。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：城市规划
- **时间限制**：3.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

A 国有 n 座城市，城市之间由 m 条双向道路连接，任意一座城市均可经过若干条双向道路到达另一座城市。城市依次以 $1,2,\ldots,n$ 编号。第 i（$1 \le i \le m$）条双向道路连接城市 u_i 与城市 v_i。

对于城市 u 和城市 v 而言，它们之间的连通度 $d(u,v)$ 定义为从城市 u 出发到达城市 v 所需经过的双向道路的最少条数。由于道路是双向的，可以知道连通度满足 $d(u,v)=d(v,u)$，特殊地有 $d(u,u)=0$。

现在 A 国正在规划城市建设方案。城市 u 的建设难度为它到其它城市的最大连通度。请你求出建设难度最小的城市，如果有多个满足条件的城市，则选取其中编号最小的城市。形式化地，你需要求出使得 $\max_{1\le i\le n} d(u,i)$ 最小的 u，若存在多个可能的 u 则选取其中最小的。

3.1.2 输入格式

第一行，两个正整数 n,m，表示 A 国的城市数量与双向道路数量。

接下来 m 行，每行两个整数 u_i,v_i，表示一条连接城市 u_i 与城市 v_i 的双向道路。

3.1.3 输出格式

输出一行，一个整数，表示建设难度最小的城市编号。如果有多个满足条件的城市，则选取其中编号最小的城市。

3.1.4 样例

3.1.4.1 输入样例 1

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

3.1.4.2 输出样例 1

```
1
```

3.1.4.3 输入样例 2

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

3.1.4.4 输出样例 2

```
2
```

3.1.5 数据范围

对于 40 的测试点，保证 $1 \le n \le 300$。

对于所有测试点，保证 $1 \le n \le 2000$，$1 \le m \le 2000$，$1 \le u_i, v_i \le n$。

3.1.6 参考程序

```python
# tokens里面缓存了输入
tokens = []

# 这个函数将输入的内容放到tokens里面，随后再读取tokens里的第一条内容作为输入

def get_val():
    while not tokens:
        tokens.extend(input().split())
    return int(tokens.pop(0))

# 得到城市和边的数量
n = get_val()
m = get_val()
# 邻接表，每一个节点后面的列表表示这个节点到列表里面的节点有边
e = [[] for _ in range(n + 1)]

for _ in range(m):
    u = get_val()
    v = get_val()
    e[u].append(v)
    e[v].append(u)

# 假设当前在遍历节点i，则d[j]表示节点i目前发现的到j的最短距离。如果这个值为n+1则表明我们还没遍历到j，否则表示我们已经遍历到了j
d = [0] * (n + 1)
# 这个列表其实是队列，用在以i为中心的广度优先搜索中，记录即将被bfs遍历到的节点
q = [0] * (n + 2)
# 这个数组缓存了各个城市的建设成本
mx = [0] * (n + 1)

def bfs(u):
    # 初始化d数组，表明所有节点都还没被遍历到
    for i in range(1, n + 1):
        d[i] = n + 1
    # 初始节点已经被遍历到了
    d[u] = 0
    # 将初始节点入队
    q[1] = u
    # 广度优先的队列左右边界，初始化为1，表明还没开始遍历。这个队列的长度是n+2，而一共只有n个节点，所以这个队列肯定不会被装满，一个节点只会入队不超过1次
    ql, qr = 1, 1
    # 只要队列还不空就继续遍历
    while ql <= qr:
        # 将队首的元素出栈，将队首元素称为x
        x = q[ql]
        ql += 1
        # 遍历从x出发能够直接到达的节点
        for y in e[x]:
            # 如果当前找到的最短路径不如从初始节点到x再多走一步，那么就将y节点入队
            if d[x] + 1 < d[y]:
                d[y] = d[x] + 1
                # 理论上来讲，使用数组一样的数据结构存储队列元素，这里需要取模。但是这道题里面，每一个节点只会入队一次，所以qr最大就到n+1，因此不需要取模
                qr += 1
                q[qr] = y
    # 队列中最后一个弹出的元素一定是最后入队的，这个元素一定离初始节点最远，所以他的距离就是建设难度
    mx[u] = d[q[qr]]

ans = 1
for i in range(1, n + 1):
    bfs(i)
    # 判断当前的建设难度是否小于之前找到的最小的，如果是则更新答案
```

```
    if mx[i] < mx[ans]:
        ans = i

print(ans)
```

3.2 编程题 2

- **试题名称**：学习小组
- **时间限制**：10.0 s
- **内存限制**：512.0 MB

3.2.1 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。班级里的同学依次以 $1, 2, \ldots, n$ 编号，第 i 名同学有其发言积极度 c_i。

观察发现，如果一个学习小组中恰好包含编号为 $p_1, p_2, \ldots, p_k$ 的 k 名同学，则该学习小组的基础讨论积极度为 a_k，综合讨论积极度为 $a_k + \max\{c_{p_1}, c_{p_2}, \ldots, c_{p_k}\} - \min\{c_{p_1}, c_{p_2}, \ldots, c_{p_k}\}$，也即基础讨论积极度加上小组内同学的最大发言积极度与最小发言积极度之差。

给定基础讨论积极度 $a_1, a_2, \ldots, a_n$，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，综合讨论积极度之和的最大值。

3.2.2 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 $c_1, c_2, \ldots, c_n$，表示每位同学的发言积极度。

第三行，n 个非负整数 $a_1, a_2, \ldots, a_n$，表示不同人数学习小组的基础讨论积极度。

3.2.3 输出格式

输出一行，一个整数，表示所有划分方案中，学习小组综合讨论积极度之和的最大值。

3.2.4 样例

3.2.4.1 输入样例 1

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

3.2.4.2 输出样例 1

```
12
```

3.2.4.3 输入样例 2

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

3.2.4.4 输出样例 2

```
21
```

3.2.5 数据范围

对于 40 的测试点，保证 $c_i = 0$。

对于所有测试点，保证 $1 \le n \le 300$，$0 \le c_i \le 10^4$，$0 \le a_i \le 10^4$。

3.2.6 参考程序

```python
getint = lambda: map(int, input().split())
getints = lambda: list(getint())

# 同学数量
n = int(input())
# 按照讨论积极程度从小到大排序，方便后面计算转移方程
c = sorted(getints())
# 不同分组的基础讨论度
a = [0] + getints()
# 用来进行动态规划的数组
# f[j][k]表示在当前已经用掉k个学生，且包含了j个人数不为1的组的情况下的最大积极度
# 我们在这里使用了滚动数组优化，所以没有题解中的i维度
f = [[0] * (n + 1) for _ in range(n + 1)]

# 考虑人数最少的组不少于i个人的情况
for i in range(n, 0, -1):
    # j表示分成了多少个组
    # 一共就n个人，所以组的数量绝对不大于n//i+1
    for j in range(1, n//i+1):
        # 如果最后一个小组刚刚好有i个人，因为之前的组都有至少i个人，所以新加入的组应该选择第j大和第j小的元素作为极大极小值
        diff = c[n - j] - c[j - 1] if i > 1 else 0
        if diff < 0:
            break
        # 假设我们考虑用掉了k个学生，只需要考虑k>=i的情况即可
        for k in range(i, n + 1):
            # 判断最后一个小组如果刚刚好有i个人能否比
            f[j][k] = max(f[j][k], f[j - 1][k - i] + a[i] + diff)

#
print(max(f[i][n] for i in range(n + 1)))
```