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

Python 七级

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	B	C	A	B	B	A	A	B	C	A	B	B	A	B	A

第 1 题 下列哪个选项是python中的关键字?

- **A.** function
- **B.** class
- **C.** method
- **D.** object

第 2 题 下面程序的时间复杂度是（）

```
def func(n):
    for i in range(n):
        for j in range(i, n):
            print(i, j)
```

- **A.** n
- **B.** n*log(n)
- **C.** n的平方
- **D.** n的立方

第 3 题 以下代码输出的是什么（）

```
class A:
    def __init__(self):
        self.x = 1

class B(A):
    def __init__(self):
        super().__init__()
        self.y = 2

b = B()
print(b.x, b.y)
```

- **A.** 1 2

- [ ] **B.** 报错
- [ ] **C.** None 2
- [ ] **D.** 1 None

第 4 题 pow(10, log10(100))的值是

- [ ] **A.** 10
- [ ] **B.** 100
- [ ] **C.** 1000
- [ ] **D.** 10000

第 5 题 exp(log(2))的值是（）

- [ ] **A.** 1
- [ ] **B.** 2
- [ ] **C.** 3
- [ ] **D.** 10

第 6 题 给定一个无向图，图的节点编号从 0 到 n-1，图的边以邻接表的形式给出。编写的一个python程序，使用深度优先搜索（DFS）遍历该图，并输出遍历的节点顺序。
下面程序中横线处应该填写的是（）

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")

    for neighbor in graph[start]:
        if neighbor not in visited:
            ________________

graph = {
    0: [1, 2],
    1: [0, 3, 4],
    2: [0, 5],
    3: [1],
    4: [1, 5],
    5: [2, 4]
}

print("DFS 遍历顺序:")
dfs(graph, 0)
```

- [ ] **A.** dfs(graph, neighbor, visited)
- [ ] **B.** dfs(graph+1, neighbor, visited)
- [ ] **C.** dfs(graph, neighbor)

- [ ] **D.** dfs(graph+1, visited)

第 7 题 [10, 9, 2, 5, 3, 7, 101, 18]，最长的严格上升子序列是（）

- [ ] **A.** [2, 3, 7, 101]，长度为 4
- [ ] **B.** [2, 5, 7, 101]，长度为 5
- [ ] **C.** [2, 5, 7, 101]，长度为 3
- [ ] **D.** [2, 5, 7, 18]，长度为 6

第 8 题 给定一个整数数组 nums，找到其中最长的严格上升子序列的长度。
子序列 是指从原数组中删除一些元素（或不删除）后，剩余元素保持原有顺序的序列。
要求：
子序列必须是严格上升的（即每个元素都比前一个元素大）。
返回最长严格上升子序列的长度。
横线处应该填写的是（）

```
def length_of_lis(nums):
    if not nums:
        return 0

    dp = [1] * len(nums)
    for i in range(1, len(nums)):
        for j in range(i):
            if nums[j] < nums[i]:
                ________________
    return max(dp)
```

- [ ] **A.** dp[i] = max(dp[i], dp[j])
- [ ] **B.** dp[i] = max(dp[i], dp[j] + 1)
- [ ] **C.** dp[i] = max(dp[i]+1, dp[j] + 1)
- [ ] **D.** dp[i] = max(dp[i]+1, dp[j])

第 9 题 以下代码的时间复杂度是多少?

```
def fib(n):
   if n <= 1:
       return n
   return fib(n - 1) + fib(n - 2)
```

- [ ] **A.** n
- [ ] **B.** n的平方
- [ ] **C.** 2的n次幂
- [ ] **D.** log(n)

第 10 题 以下代码的时间复杂度是多少?

```python
def fib(n, memo={}):
    if n <= 1:
        return n
    if n not in memo:
        memo[n] = fib(n - 1, memo) + fib(n - 2, memo)
    return memo[n]
```

- [ ] **A.** n
- [ ] **B.** n的平方
- [ ] **C.** 2的n次幂
- [ ] **D.** log(n)

第 11 题 以下代码的功能是什么?

```python
def max_subarray(nums):
    dp = [0] * len(nums)
    dp[0] = nums[0]
    for i in range(1, len(nums)):
        dp[i] = max(nums[i], dp[i - 1] + nums[i])
    return max(dp)
```

- [ ] **A.** 计算数组的最大值
- [ ] **B.** 计算数组的最大子数组和
- [ ] **C.** 计算数组的最小值
- [ ] **D.** 计算数组的最小子数组和

第 12 题 以下代码的功能是什么?

```python
def coin_change(coins, amount):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0
    for coin in coins:
        for i in range(coin, amount + 1):
            dp[i] = min(dp[i], dp[i - coin] + 1)
    return dp[amount] if dp[amount] != float('inf') else -1
```

- [ ] **A.** 计算硬币的组合数
- [ ] **B.** 计算硬币的最小数量，使得总金额等于目标金额
- [ ] **C.** 计算硬币的最大数量，使得总金额等于目标金额
- [ ] **D.** 计算硬币的总金额

第 13 题 以下代码的功能是什么?

```python
def fuction1(text1, text2):
    m, n = len(text1), len(text2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if text1[i - 1] == text2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
    return dp[m][n]
```

- [ ] **A.** 计算两个字符串的最长公共子序列
- [ ] **B.** 计算两个字符串的最长公共子串
- [ ] **C.** 计算两个字符串的最长公共前缀
- [ ] **D.** 计算两个字符串的最长公共后缀

第 14 题 以下代码的功能是什么?

```python
def knapsack(weights, values, capacity):
    n = len(weights)
    dp = [[0] * (capacity + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, capacity + 1):
            if weights[i - 1] <= j:
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1])
            else:
                dp[i][j] = dp[i - 1][j]
    return dp[n][capacity]
```

- [ ] **A.** 计算背包问题的最小价值
- [ ] **B.** 计算背包问题的最大价值
- [ ] **C.** 计算背包问题的最小重量
- [ ] **D.** 计算背包问题的最大重量

第 15 题 以下代码的功能是什么?

```python
def unique_paths(m, n):
    dp = [[1] * n for _ in range(m)]
    for i in range(1, m):
        for j in range(1, n):
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
    return dp[m - 1][n - 1]
```

- [ ] **A.** 计算从左上角到右下角的唯一路径数
- [ ] **B.** 计算从左上角到右下角的最短路径
- [ ] **C.** 计算从左上角到右下角的最长路径

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 表达式 `1e6` 、 `1000000` 和 `10^6` 的值是相同的。

第 2 题 在python语言中，函数调用前必须有函数声明或定义。

第 3 题 快速排序一般是不稳定的。

第 4 题 `int` 类型能表达的数都能使用 `float` 类型精确表达。

第 5 题 使用了math模块中的表达式 `cos(60)` 的结果类型为 `float` 、值约为 `0.5` 。

第 6 题 一颗N层的满二叉树，一定有2^N-1个结点。

第 7 题 邻接表和邻接矩阵都是图的存储形式。为了操作时间复杂度考虑，同一个图可以同时维护两种存储形式。

第 8 题 子类对象包含父类的所有成员（包括私有成员）。从父类继承的私有成员也是子类的成员，因此子类可以直接访问。

第 9 题 动态规划算法通常有递归实现和递推实现。但由于递归调用在运行时会由于层数过多导致程序崩溃，因此有些动态规划算法只能用递推实现。

第 10 题 按照下面的规则生成一棵二叉树：以一个人为根节点，其父亲为左子节点，母亲为右子节点。对其父亲、母亲分别用同样规则生成左子树和右子树。以此类推，记录30代的直系家谱，则这是一棵满二叉树。

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　图上移动

3.1.2　题目描述

小 A 有一张包含 n 个结点与 m 条边的无向图，结点以 $1,2,\ldots,n$ 标号。小 A 会从图上选择一个结点作为起点，每一步移动到某个与当前小 A 所在结点相邻的结点。对于每个结点 i（$1\le i\le n$），小 A 想知道从结点 i 出发恰好移动 $1,2,\ldots,k$ 步之后，小 A 可能位于哪些结点。由于满足条件的结点可能有很多，你只需要求出这些结点的数量。

3.1.3　输入格式

第一行，三个正整数 n,m,k，分别表示无向图的结点数与边数，最多移动的步数。

接下来 m 行，每行两个正整数 u_i,v_i，表示图中的一条连接结点 u_i 与 v_i 的无向边。

3.1.4　输出格式

共 n 行，第 i 行（$1\le i\le n$）包含 k 个整数，第 j 个整数（$1\le j\le k$）表示从结点 i 出发恰好移动 j 步之后可能位于的结点数量。

3.1.5 样例

3.1.5.1 输入样例 1

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

3.1.5.2 输出样例 1

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

3.1.6 数据范围

对于 20% 的测试点，保证 $k = 1$。

对于另外 20% 的测试点，保证 $1 \leq n \leq 50$，$1 \leq m \leq 50$。

对于所有测试点，保证 $1 \leq n \leq 500$，$1 \leq m \leq 500$，$1 \leq k \leq 20$，$1 \leq u_i, v_i \leq n$。

3.1.7 参考程序

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

f = [[[False] * (n + 2) for _ in range(n + 2)] for __ in range(k + 2)]

for i in range(1, n + 1):
    f[0][i][i] = True

for t in range(1, k + 1):
    for x in range(1, n + 1):
        for y in range(1, n + 1):
            if f[t - 1][x][y]:
                for z in to[y]:
                    f[t][x][z] = True

for i in range(1, n + 1):
    Ans = []
    for t in range(1, k + 1):
        ans = 0
        for j in range(1, n + 1):
            ans += (f[t][i][j] == True)
        Ans.append(str(ans))
    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 等价消除

3.2.9 题目描述

小 A 有一个仅包含小写英文字母的字符串 S。

对于一个字符串，如果能通过每次删去其中两个相同字符的方式，将这个字符串变为空串，那么称这个字符串是可以被等价消除的。

小 A 想知道 S 有多少子串是可以被等价消除的。

一个字符串 S' 是 S 的子串，当且仅当删去 S 的某个可以为空的前缀和某个可以为空的后缀之后，可以得到 S'。

3.2.10 输入格式

第一行，一个正整数 $|S|$，表示字符串 S 的长度。

第二行，一个仅包含小写英文字母的字符串 S。

3.2.11 输出格式

一行，一个整数，表示答案。

3.2.12 样例

3.2.12.3 输入样例 1

```
7
aaaaabb
```

3.2.12.4 输出样例 1

```
9
```

3.2.12.5 输入样例 2

```
9
babacabab
```

3.2.12.6 输出样例 2

```
2
```

3.2.13 数据范围

对于 20% 的测试点，保证 S 中仅包含 `a` 和 `b` 两种字符。

对于另外 20% 的测试点，保证 $1 \leq |S| \leq 2000$。

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

3.2.14 参考程序

```python
n = int(input())
s = input()
state, cnt, ans = 0, {}, 0
cnt[0] = 1
for i in range(n):
    c = ord(s[i]) - ord('a')
    state ^= (1 << c)
    if state not in cnt:
        cnt[state] = 0
    ans += cnt[state]
    cnt[state] += 1
print(ans)
```