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

C++ 七级

2026 年 09 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	D	C	B	A	B	A	D	C	B	A	D	D	B	C

第 1 题 下列 C++ 代码的输出结果是（ ）。

```cpp
#include <iostream>
using namespace std;
int main() {
    int a = 5, b = 3;
    cout << (a & b) + (a | b) << endl;
    return 0;
}
```

- A. 6
- B. 7
- C. 8
- D. 9

第 2 题 使用 `cmath` 或 `math.h` 中的数学库函数，下列说法中正确的是（ ）。

- A. `pow(2, 3)` 的返回值类型为 `int`
- B. `sin(30)` 的参数 `30` 表示 30 度
- C. `sqrt(4)` 的返回值类型为 `int`
- D. `log(1)` 的返回值为 `0.0`，且类型为 `double`

第 3 题 有 4 个字符，出现次数分别为 1、2、3、4。构造哈夫曼树后，出现次数为 1 的字符的哈夫曼编码长度为（ ）。

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

第 4 题 从 4×5 个点连接成的网格的左上角走到右下角，每次只能向右或向下移动，不同的路径共有（ ）条。

- A. 20
- B. 35
- C. 70

- [ ] **D.** 126

第 5 题 在含有 n 个结点的二叉排序树中查找一个元素，平均时间复杂度和最坏时间复杂度分别为（　　）。

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

第 6 题 有 4 堆石子，数量分别为 1、2、3、4。每次可以合并相邻两堆，合并代价为两堆石子数之和。将所有石子合并成一堆的最小总代价为（　　）。

- [ ] **A.** 17
- [ ] **B.** 19
- [ ] **C.** 20
- [ ] **D.** 23

第 7 题 在无权图中，使用 BFS 从起点开始遍历，并在访问由结点 `u` 扩展的相邻结点 `v` 时记录 `dist[v] = dist[u] + 1`，且起点的 `dist` 为 `0`，则最终 `dist[v]` 表示的是（　　）。

- [ ] **A.** 起点到结点 `v` 的最少边数
- [ ] **B.** 结点 `v` 的度数
- [ ] **C.** 从起点到结点 `v` 的路径上经过的最大边权
- [ ] **D.** 包含结点 `v` 的连通块大小

第 8 题 在二维网格上实现泛洪填充时，为了防止递归层数过深，最适合的非递归实现方式是（　　）。

- [ ] **A.** 使用哈希表记录每个格子被访问的次数
- [ ] **B.** 使用快速排序预处理网格
- [ ] **C.** 使用二分查找定位边界
- [ ] **D.** 使用队列实现 BFS 或使用显式栈模拟 DFS

第 9 题 关于哈希表，下列说法正确的是（　　）。

- [ ] **A.** 只要哈希函数选择合适，就可以完全避免冲突
- [ ] **B.** 在链地址法中，查找一个元素的时间复杂度一定为 $O(1)$
- [ ] **C.** 开放定址法发生冲突后，会在表内寻找下一个可用位置
- [ ] **D.** 哈希表的查找速度与表中元素个数无关

第 10 题 下列 C++ 代码的输出结果是（　　）。

```cpp
#include <iostream>
using namespace std;

void inc(int &x) {
    x++;
}
int main() {
    int a = 3;
    inc(a);
    cout << a;
    return 0;
}
```

- [ ] **A.** `3`
- [ ] **B.** `4`
- [ ] **C.** `5`
- [ ] **D.** 编译错误

第 11 题 用动态规划求两个序列 s_1 和 s_2 的最长公共子序列长度，若 `dp[i][j]` 表示 s_1 前 `i` 个元素与 s_2 前 `j` 个元素的 LCS 长度。当 $s_1[i-1] = s_2[j-1]$ 时，正确的状态转移是（　　）。

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

第 12 题 下列代码是一维数组优化 0/1 背包的核心片段，执行后 `dp[8]` 的输出结果是（　　）。

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
int main() {
    int w = 3, v = 5, W = 8;
    int dp[9] = {0};
    for (int c = W; c >= w; c--)
        dp[c] = max(dp[c], dp[c - w] + v);
    cout << dp[8] << endl;
    return 0;
}
```

- [ ] **A.** `0`
- [ ] **B.** `1`
- [ ] **C.** `3`
- [ ] **D.** `5`

第 13 题 若要求排序后相等元素的相对顺序保持不变，下列排序算法中最不适宜使用的是（　　）。

- [ ] **A.** 冒泡排序
- [ ] **B.** 插入排序
- [ ] **C.** 归并排序
- [ ] **D.** 快速排序

第 14 题 下列代码片段的时间复杂度为（　　）。

```cpp
long long s = 0;
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j += i)
        s += i + j;
```

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

第 15 题 已知 `int a[6] = {1, 3, 5, 7, 9, 11}; int *p = a + 1;`，则表达式 `*(p + 3)` 的值是（　　）。

- [ ] **A.** `5`
- [ ] **B.** `7`
- [ ] **C.** `9`
- [ ] **D.** `11`

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 使用 `cmath` 或 `math.h` 中的函数，表达式 `exp(0)` 的结果值为 `1.0`，且类型为 `double`。

第 2 题 采用开放定址法处理冲突的哈希表中，删除一个元素后可以直接将该位置置空，不会影响后续查找。

第 3 题 在哈夫曼树中，出现次数更多的叶子结点，其深度总是更小。

第 4 题 在一个有向图中，所有顶点的入度之和总是等于所有顶点的出度之和。

第 5 题 广度优先搜索通常借助队列实现，深度优先搜索通常借助栈或递归实现。

第 6 题 快速排序的平均时间复杂度为 $O(n \log n)$，最坏时间复杂度也为 $O(n \log n)$。

第 7 题 为解决 0/1 背包问题，使用一维数组优化时，内层容量循环应从大到小枚举。

第 8 题 使用邻接表存储图时，遍历某个顶点的所有邻边所需时间与图中顶点数成正比。

第 9 题 在按层序从 1 开始对结点编号的完全二叉树中，编号为 i（$i > 1$）的结点的父结点编号为 $\lfloor i/2 \rfloor$。

第 10 题 在定义了数组 `int arr[10];` 后，表达式 `arr` 和表达式 `&arr[0]` 总是等价的。

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：必经之路
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

给定一张有 n 个结点 m 条边的有向图 G，G 中的结点依次以 $1, 2, \ldots, n$ 编号。第 i 条边（$1 \leq i \leq m$）从结点 u_i 指向结点 v_i。

G 中任一入度为 0 的结点可以作为合法起点，任一出度为 0 的结点可以作为合法终点。

如果 G 中所有可能的从合法起点到合法终点的路径都会经过结点 u，则称 u 是必经点。注意必经点可以为合法起点或合法终点。

请你求出 G 中所有必经点的编号。

例如，在下图中合法起点有点 1 与点 2，合法终点有点 7 与点 8。

```
(1)            (5)---->(7)
   \           ^  \    ^
    v         /    v  /
    (3)      /     (6)
    ^  \    /         \
   /    v  /           v
(2)---->(4)            (8)
```

所有合法起点到合法终点的路径为：

- $1 \to 3 \to 4 \to 5 \to 7$
- $1 \to 3 \to 4 \to 5 \to 6 \to 7$
- $1 \to 3 \to 4 \to 5 \to 6 \to 8$
- $2 \to 3 \to 4 \to 5 \to 7$
- $2 \to 3 \to 4 \to 5 \to 6 \to 7$
- $2 \to 3 \to 4 \to 5 \to 6 \to 8$
- $2 \to 4 \to 5 \to 7$
- $2 \to 4 \to 5 \to 6 \to 7$
- $2 \to 4 \to 5 \to 6 \to 8$

因此必经点有两个，编号分别为 $4, 5$。

3.1.2 输入格式

第一行，两个正整数 n, m，表示有向图 G 中的结点数与边数。

接下来 m 行，每行两个正整数 u_i, v_i，表示一条从结点 u_i 指向结点 v_i 的有向边。

保证 G 中至少有一个合法起点，至少有一个合法终点，且至少存在一条从一个合法起点到一个合法终点路径，同时不存在孤立点（即出度和入度都为 0 的点）。

3.1.3 输出格式

第一行，一个整数，表示必经点的数量 k。

如果存在必经点，则第二行从小到大输出 G 中所有必经点的编号。

3.1.4 样例

3.1.5 输入样例 1

```
8 9
1 3
2 3
3 4
4 5
5 6
6 7
6 8
2 4
5 7
```

3.1.6 输出样例 1

```
2
4 5
```

3.1.7 输入样例 2

```
8 9
1 3
2 3
3 4
4 5
5 6
6 7
6 8
2 5
4 7
```

3.1.8 输出样例 2

```
0
```

3.1.9 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 100$，$1 \leq m \leq 200$。

对于所有测试点，保证 $1 \leq n \leq 1000$，$1 \leq m \leq 2000$。保证 G 中至少有一个合法起点，至少有一个合法终点，且至少存在一条从一个合法起点到一个合法终点路径，同时不存在孤立点（即出度和入度都为 0 的点）。

3.1.10 参考程序

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int N = 1005;
const int E = 2005;

int n, m;
int h[N], to[E], nx[E], et;
int id[N], od[N];
bool vis[N];
int q[N], ql, qr;
int ans[N], cnt;

bool chk(int ban) {
    ql = qr = 0;
    for (int i = 1; i <= n; i++) {
        vis[i] = (i == ban);
        if (!vis[i] && !id[i]) {
            q[++qr] = i;
            vis[i] = 1;
        }
    }
    while (ql < qr) {
        int u = q[++ql];
        for (int i = h[u]; i; i = nx[i]) {
            int v = to[i];
            if (vis[v])
                continue;
            if (!od[v])
                return 0;
            q[++qr] = v;
            vis[v] = 1;
        }
    }
    return 1;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        to[++et] = v;
        nx[et] = h[u];
        h[u] = et;
        od[u]++;
        id[v]++;
    }
    for (int i = 1; i <= n; i++)
        if (chk(i))
            ans[++cnt] = i;
    printf("%d\n", cnt);
    for (int i = 1; i <= cnt; i++)
        printf("%d%c", ans[i], " \n"[i == cnt]);
    return 0;
}
```

3.2 编程题 2

- **试题名称**：括号序列

- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

对于字符串 S 与 T，如果从 S 中删除任意多个字符可以得到 T，那么 T 是 S 的子序列。换言之，T 是选取 S 中的若干字符按下标顺序连接而成的。两个子序列不同当且仅当所选取的下标不同。

例如 `sun` 是 `sequence` 的子序列，因为从 `sequence` 中删除 `eq`、`e` 和 `ce` 可以得到 `sun`；`sequence` 有 2^8 个不同的子序列，其中有空字符串，也有三个不同的子序列 `e`，因为 `sequence` 的第 $2, 5, 8$ 个字符都为 `e`，分别保留这三个字符得到的子序列是不同的。

对于字符串 S，如果 S 满足以下条件那么 S 是合法括号序列：

- S 是空字符串，或者
- S 可由 `(`、合法括号序列、`)` 三者连接得到，或者
- S 可由两个合法括号序列连接得到。

例如 `()`、`()()`、`(())` 和 `(()())` 都是合法括号序列。但是 `(()`、`)(` 不是合法括号序列。

给定一个长度为 n 的仅包含 `(` 与 `)` 的字符串 S。请你求出 S 所有 2^n 个子序列中有多少个合法括号序列。由于答案可能很大，请你输出答案对 10^9 取模的结果。

例如，S 为 `))(()(` 时共有 3 个子序列是合法括号序列，分别为空字符串与两个不同的子序列 `()`。

3.2.2 输入格式

第一行，一个正整数 n，表示字符串 S 的长度。

第二行，长度为 n 的仅包含 `(` 与 `)` 的字符串 S。

3.2.3 输出格式

输出一行，一个整数，表示 S 的合法括号子序列的数量对 10^9 取模的结果。

3.2.4 样例

3.2.5 输入样例 1

```
6
))(()(
```

3.2.6 输出样例 1

```
3
```

3.2.7 输入样例 2

```
34
((((((((((((((((()))))))))))))))))
```

3.2.8 输出样例 2

```
333606220
```

3.2.9 数据范围

对于 40% 的测试点，保证 $1 \leq n \leq 400$。

对于所有测试点，保证 $1 \leq n \leq 2000$。

3.2.10 参考程序

```cpp
#include <cstdio>
#include <algorithm>
using namespace std;

const int N = 2005;
const int mod = 1e9;

int n;
char s[N];
int f[N][N];

int main() {
    scanf("%d", &n);
    scanf("%s", s + 1);
    f[0][0] = 1;
    for (int i = 1; i <= n; i++)
        for (int j = 0; j <= n; j++) {
            f[i][j] = f[i - 1][j];
            if (s[i] == '(' && j)
                f[i][j] = (f[i][j] + f[i - 1][j - 1]) % mod;
            if (s[i] == ')' && j < n)
                f[i][j] = (f[i][j] + f[i - 1][j + 1]) % mod;
        }
    printf("%d\n", f[n][0]);
    return 0;
}
```