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

C++ 七级

2026 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	D	B	B	D	D	B	A	A	B	B	C	D	D	C

第 1 题 假设一个算法时间复杂度的递推式是 $T(n) = 2T(n-1) + 1$（n 为正整数），且 $T(0) = 1$，那么这个算法的时间复杂度是（ ）。

- A. $O(n)$
- B. $O(n \log n)$
- C. $O(n^2)$
- D. $O(2^n)$

第 2 题 下面关于“唯一分解定理”和“素数筛法”的说法中，错误的是（ ）。

- A. 如果预处理出 n 以内每个数的最小质因子，那么可以在 $O(\log n)$ 时间内完成任意一个不超过 n 的整数的质因数分解。
- B. 线性筛（欧拉筛）能够保证每个合数只被其最小质因子筛掉一次，这一性质依赖于唯一分解定理。
- C. 唯一分解定理保证：若一个数未被任何不超过其平方根的质数筛去，则它一定是质数。
- D. 唯一分解定理是埃氏筛时间复杂度为 $O(n \log \log n)$ 的根本原因。

第 3 题 若字符串A与字符串B的最长公共子序列（LCS）长度为 5，则（ ）。

- A. 它们的编辑距离为 5
- B. 它们至少有 5 个公共字符
- C. 它们最长公共子串长度为 5
- D. 它们一定长度相等

第 4 题 对于一棵包含 n 个顶点（$n \geq 2$）的树，其所有顶点的度数之和必定等于（ ）。

- A. $n - 1$
- B. $2n - 2$
- C. $2n$
- D. n^2

第 5 题 关于哈希表（Hash Table）在**不考虑扩容且采用简单均匀哈希函数的前提下**，下列说法中错误的是（ ）。

- A. 装载因子越大，发生冲突的概率通常越高
- B. 开放定址法在删除元素时实现相对复杂

- [ ] **C.** 链地址法在最坏情况下查找时间复杂度为 $O(n)$
- [ ] **D.** 查找哈希表的时间复杂度总是 $O(1)$

第 6 题 深度优先搜索（DFS）在遍历图时，每当访问到某个顶点后，选择一个相邻的未访问顶点继续搜索，直到某个顶点的所有相邻顶点均已被访问，则退回到前一顶点继续搜索。该算法主要运用了（ ）。

- [ ] **A.** 分治
- [ ] **B.** 贪心
- [ ] **C.** 动态规划
- [ ] **D.** 回溯

第 7 题 下面程序的运行结果为（ ）。

```cpp
#include <iostream>
#include <algorithm>

bool check(int n, int a[], int k, int dist) {
    int cnt = 1;
    int last = a[0];

    for (int i = 1; i < n; i++) {
        if (a[i] - last >= dist) {
            cnt++;
            last = a[i];
        }
    }

    return cnt >= k;
}

int solve(int n, int a[], int k) {
    std::sort(a, a + n);

    int l = 0;
    int r = a[n - 1] - a[0];

    while (l < r) {
        int mid = (l + r + 1) / 2;

        if (check(n, a, k, mid))
            l = mid;
        else
            r = mid - 1;
    }

    return l;
}

int main() {
    int a[] = {1, 2, 8, 4, 9};
    int n = 5;
    int k = 3;

    std::cout << solve(n, a, k) << std::endl;

    return 0;
}
```

- [ ] **A.** 2

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

第 8 题 下面程序的时间复杂度是（ ），假设数组 a 的值域范围是 D。

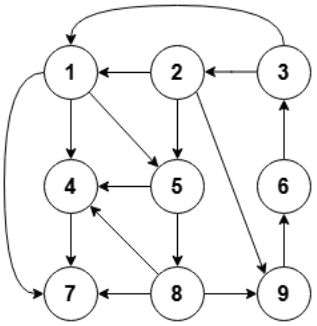
```cpp
#include <iostream>
#include <algorithm>

bool check(int n, int a[], int k, int dist) {
    int cnt = 1;
    int last = a[0];

    for (int i = 1; i < n; i++) {
        if (a[i] - last >= dist) {
            cnt++;
            last = a[i];
        }
    }

    return cnt >= k;
}

int solve(int n, int a[], int k) {
    std::sort(a, a + n);

    int l = 0;
    int r = a[n - 1] - a[0];

    while (l < r) {
        int mid = (l + r + 1) / 2;

        if (check(n, a, k, mid))
            l = mid;
        else
            r = mid - 1;
    }

    return l;
}

int main() {
    int a[] = {1, 2, 8, 4, 9};
    int n = 5;
    int k = 3;

    std::cout << solve(n, a, k) << std::endl;

    return 0;
}
```

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

第 9 题 某二叉树共有 10 个结点，记为 A~J，已知它的先序遍历序列为：**A B D H I E C F J G**，中序遍历序列为：**H D I B E A F J C G**，则该二叉树的后序遍历序列是（ ）。

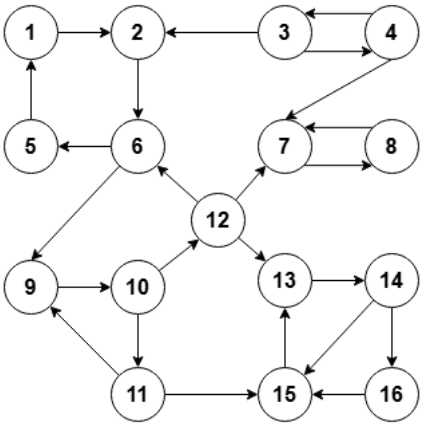
- [ ] **A.** H I D E B J F G C A
- [ ] **B.** H I D B E J F G C A
- [ ] **C.** I H D E B J F G C A
- [ ] **D.** H I D E B F J G C A

第 10 题 下面哪一个可能是下图的深度优先遍历序列（ ）。



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

第 11 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

第 12 题 关于泛洪算法（Flood Fill）的说法，正确的是（ ）。

- [ ] **A.** 泛洪算法只适用于二维网格中的四连通或八连通问题。
- [ ] **B.** 泛洪算法必须使用递归方式实现。
- [ ] **C.** 泛洪算法本质上是对图进行一次从起点出发的搜索。
- [ ] **D.** 泛洪算法只能用于统计连通块个数，不能用于计算面积或周长。

第 13 题 有 `6` 个字符，它们出现的次数分别为：`{2, 3, 3, 4, 6, 8}`，现在用哈夫曼编码为这些字符编码，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.** 58
- [ ] **B.** 60
- [ ] **C.** 62
- [ ] **D.** 64

第 14 题 关于单链表、双链表和循环链表，下列说法正确的是（ ）。

- [ ] **A.** 在单链表中，若已知某结点的指针，则可以在 $O(1)$ 时间内删除该结点。
- [ ] **B.** 循环链表中一定不存在空指针。
- [ ] **C.** 在循环双链表中，尾结点的 next 指针一定为 NULL。
- [ ] **D.** 在带头结点的循环单链表中，判定链表是否为空只需判断头结点的 next 是否指向自身。

第 15 题 下列关于树的遍历的说法中，正确的一项是（ ）。

- [ ] **A.** 对任意一棵树进行深度优先遍历，所得序列一定唯一。
- [ ] **B.** 已知一棵二叉树的先序遍历和后序遍历序列，可以唯一确定这棵二叉树。
- [ ] **C.** 已知一棵二叉树的先序遍历和中序遍历序列，可以唯一确定这棵二叉树。
- [ ] **D.** 一棵二叉树的中序遍历序列是单调递增的，则该二叉树一定是二叉平衡树。

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 C++ 语言中，表达式 `4 ^ 2` 的结果类型为 `int`，值为 `6`。

第 2 题 C++ 中引用可以重新绑定。

第 3 题 在 C++ 中，若函数形参为引用类型，则在函数内部对该形参的修改会影响对应的实参。

第 4 题 如果一个最值问题可以用动态规划在多项式时间内求解，那么也一定存在一种贪心策略，可以在多项式时间内求得最优解。

第 5 题 使用归并排序对 n 个元素进行排序时，无论最好、最坏还是平均情况，时间复杂度均为 $O(n \log n)$。

第 6 题 在无向连通图中删除一条边，该图就一定变成非连通图。

第 7 题 在一个无向图中，每个顶点有不同的编号，在执行深度优先遍历过程中选择下一个顶点时总是优先选择编号更小的相邻顶点，则从指定顶点开始的遍历序列是唯一的。

第 8 题 若所有字符出现频率相同，则哈夫曼编码一定会得到完全二叉树。

第 9 题 使用 `math.h` 或 `cmath` 头文件中的函数，表达式 `sin(90)` 的结果为 `1`。

第 10 题 在一个无向连通图中，从任意顶点开始进行深度优先遍历，最终得到的 DFS 生成树一定包含图中的所有顶点。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

- **试题名称**：拆分
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

小 A 想将正整数 n 拆分成若干个正整数之和，并最大化拆分后的正整数之积。小 A 希望你帮他计算出拆分后正整数之积的最大值。由于答案可能很大，你只需要求出答案对 10^9 取模的结果。

形式化地，n 的拆分是满足 $a_1 + \cdots + a_k = n$ 的若干个正整数 $a_1, \ldots, a_k$，其中 $1 \le k \le n$。你需要求出 n 的所有拆分中 $a_1 \times \cdots \times a_n$ 的最大值对 10^9 取模的结果。

3.1.2 输入格式

第一行，一个正整数 t，表示数据组数。

对于每组数据：一行，一个整数 n，表示给定的正整数。

3.1.3 输出格式

对于每组数据：输出一行，一个整数，表示 n 拆分后正整数之积的最大值对 10^9 取模的结果。

3.1.4 样例

3.1.4.1 输入样例

```
3
5
8
100
```

3.1.4.2 输出样例

```
6
18
755407364
```

3.1.5 数据范围

对于 40% 的测试点，保证 $n \le 50$。

对于所有测试点，保证 $1 \le t \le 10^4$，$1 \le n \le 10^6$。

3.1.6 参考程序

```cpp
#include <cstdio>
#include <cmath>
#include <algorithm>

using namespace std;

const int N = 1e6 + 5;
const int mod = 1e9;
const double ln2 = log(2);
const double ln3 = log(3);

int f[N];
double lnf[N];

int main() {
    f[0] = f[1] = 1;
    for (int i = 0; i < N; i++) {
        if (i + 2 < N && lnf[i] + ln2 > lnf[i + 2]) {
            lnf[i + 2] = lnf[i] + ln2;
            f[i + 2] = 2ll * f[i] % mod;
        }
        if (i + 3 < N && lnf[i] + ln3 > lnf[i + 3]) {
            lnf[i + 3] = lnf[i] + ln3;
            f[i + 3] = 3ll * f[i] % mod;
        }
    }
    int t;
    scanf("%d", &t);
    while (t--) {
        int n;
        scanf("%d", &n);
        printf("%d\n", f[n]);
    }
    return 0;
}
```

3.2 编程题 2

- **试题名称**：物流网络
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

一个物流网络由 n 个城市和 m 条双向公路组成。每条公路都有两个属性：

- 运输费用 w_i
- 景观评分 b_i

当一辆运输车从城市 1 运送货物到城市 n 时，需要支付经过道路的运输费用之和。

为了推广旅游线路，物流公司推出了一项优惠政策：在运输路径上，可以**免除景观评分最高**的那条公路的运输费用。如果有多条公路的景观评分同为最大值，则只免除其中 **一条** 的费用。

请你计算，从城市 1 到城市 n 的**最小运输费用**。

3.2.2 输入格式

第一行两个整数 n, m，分别表示城市数量和公路数量。

接下来 m 行，每行四个整数 u, v, w, b，表示存在一条连接城市 u 和城市 v 的双向公路，其中 w 为运输费用，b 为景观评分。

3.2.3 输出格式

输出一个整数，表示从城市 1 到城市 n 的最小费用。
如果无法到达，输出 `-1`。

3.2.4 样例

3.2.4.1 输入样例

```
3 3
1 2 10 5
2 3 20 6
1 3 100 1
```

3.2.4.2 输出样例

```
0
```

3.2.5 样例解释

路径 $1 \to 2 \to 3$：费用 $10 + 20$，最大美丽值 6 (边 $2 - 3$)。免除 20，总花费 10。

路径 $1 \to 3$：费用 100，最大美丽值 1 (边 $1 - 3$)。免除 100，总花费 0。

最小费用为 0。

3.2.6 数据范围

$1 \le n \le 5000$, $1 \le m \le 5000$，$1 \le w, b \le 10^9$。

3.2.7 参考程序 1

```cpp
#include <algorithm>
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

struct Highway {
    int u, v;
    int fee, b;
    bool operator<(const Highway &other) const {
        return b == other.b ? fee < other.fee : b < other.b;
    }
};

void solve() {

    int n, m;
    cin >> n >> m;

    vector<Highway> highways(m);
    for (int i = 0; i < m; i ++)
        cin >> highways[i].u >> highways[i].v >> highways[i].fee >> highways[i].b;

    sort(highways.begin(), highways.end());

    vector<vector<const Highway *> > network(n + 1);

    vector<long long> dp1(n + 1, (long long)1e18);
    vector<long long> dpn(n + 1, (long long)1e18);

    dp1[1] = 0, dpn[n] = 0;
    long long answer = (long long)1e18;

    for (int ri = 0; ri < m; ri ++) {
        const Highway &r = highways[ri];
        answer = min(
            answer,
            min(dp1[r.u] + dpn[r.v],
                    dp1[r.v] + dpn[r.u])
        );

        network[r.u].push_back(&r);
        network[r.v].push_back(&r);

        vector<int> cur;
        cur.push_back(r.u), cur.push_back(r.v);
        for (int t = 0; t < cur.size(); t ++) {
            int i = cur[t];
            for (int t2 = 0; t2 < network[i].size(); t2 ++) {
                const Highway* j = network[i][t2];
                int k = (i == j->u ? j->v : j->u);
                if (dp1[i] + j->fee < dp1[k]) {
                    dp1[k] = dp1[i] + j->fee;
                    cur.push_back(k);
                }
            }
        }
        cur.clear();


        cur.push_back(r.u), cur.push_back(r.v);
```

```
        for (int t = 0; t < cur.size(); t ++) {
            int i = cur[t];
            for (int t2 = 0; t2 < network[i].size(); t2 ++) {
                const Highway* j = network[i][t2];
                int k = (i == j->u ? j->v : j->u);
                if (dpn[i] + j->fee < dpn[k]) {
                    dpn[k] = dpn[i] + j->fee;
                    cur.push_back(k);
                }
            }
        }
        cur.clear();
    }

    cout << (answer == (long long)1e18 ? -1 : answer) << '\n';
}

int main() {
    solve();
    return 0;
}
```

3.2.8 参考程序 2

```cpp
#include <algorithm>
#include <cstring>
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

struct Road {
    int u, v, w, b;
    bool operator<(const Road &other) const {
        return b == other.b ? w < other.w : b < other.b;
    }
};

void solve() {
    int n, m;
    cin >> n >> m;
    vector<Road> roads(m);
    for (auto &road : roads) {
        cin >> road.u >> road.v >> road.w >> road.b;
    }
    sort(roads.begin(), roads.end());
    vector<vector<const Road *>> graph(n + 1);
    const auto relax = [&](vector<long long> &dist, queue<int> &q) {
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            for (auto road : graph[u]) {
                int v = u == road->u ? road->v : road->u;
                if (dist[u] + road->w < dist[v]) {
                    dist[v] = dist[u] + road->w;
                    q.push(v);
                }
            }
        }
    };
    vector<long long> dist1(n + 1, 1e18), distn(n + 1, 1e18);
    dist1[1] = 0, distn[n] = 0;
    queue<int> q1, qn;
    long long ans = 1e18;
    for (const auto &road : roads) {
        ans = min(ans, min(dist1[road.u] + distn[road.v], dist1[road.v] + distn[road.u]));
        graph[road.u].push_back(&road);
        graph[road.v].push_back(&road);
        q1.push(road.u);
        q1.push(road.v);
        relax(dist1, q1);
        qn.push(road.u);
        qn.push(road.v);
        relax(distn, qn);
    }
    cout << (ans == 1e18 ? -1 : ans) << "\n";
}

int main() {
    solve();
    return 0;
}
```