# C++ 八级

2025 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 | A | C | B | C | D | B | A | B | C | A | D | A | D | D |

**第 1 题** 小杨想点一杯奶茶外卖，但还差5元起送。于是，小杨决定点一些小料。可选的小料包括：珍珠1元、椰果2元、奶冻3元、奶盖4元。每种小料最多点1份。请问共有多少种满足起送条件的点小料方案？（ ）。

- [ ] **A.** 16
- [ ] **B.** 10
- [ ] **C.** 9
- [ ] **D.** 7

**第 2 题** 小杨和小刘是好朋友，她们在逛商场时发现新设置的大头贴自拍机，于是决定一起拍一组照片。一组照片包括4张，这4张照片没有顺序区分。拍每张照片时，可以选择有相框或无相框、两人可以分别选择有头饰或无头饰、还可以从2种位置（小杨在左，或小刘在左）中选出一种。她们不希望一组照片中出现完全相同的相框、头饰、位置的组合。请问一组照片共有多少种不同的方案？（ ）。

- [ ] **A.** 1820
- [ ] **B.** 70
- [ ] **C.** 24
- [ ] **D.** 16

**第 3 题** 下列关于C++类的说法，错误的是（ ）。

- [ ] **A.** 派生类对象占用的内存总是不小于基类对象。
- [ ] **B.** 派生类可以不实现基类的虚函数。
- [ ] **C.** 如果一个类包含纯虚函数，则它不能包含成员变量。
- [ ] **D.** 如果一个类包含纯虚函数，则不能用它定义对象。

**第 4 题** 下列关于树和图的说法，错误的是（ ）。

- [ ] **A.** 每个连通图都存在生成树。
- [ ] **B.** 每个存在生成树的有向图，都一定是强连通的。
- [ ] **C.** 保留树的所有节点，并把树的每个节点指向其父节点，则可以将树转换为一个有向弱连通图。
- [ ] **D.** 保留树的所有节点，并把树的每个节点指向其子节点，则可以将树转换为一个有向无环图。

**第 5 题** 一对夫妻生男生女的概率相同。这对夫妻希望儿女双全。请问这对夫妻生下三个孩子时，实现儿女双全的概率是多少？（ ）。

- [ ] **A.** $\frac{1}{4}$
- [ ] **B.** $\frac{1}{2}$
- [ ] **C.** $\frac{3}{4}$
- [ ] **D.** $\frac{7}{8}$

**第 6 题** 二项式$(x+y)^6$的展开式中$x^2y^4$项的系数是（ ）。

- [ ] **A.** 720
- [ ] **B.** 120
- [ ] **C.** 20
- [ ] **D.** 15

**第 7 题** 对一个包含$V$个顶点、$E$条边的图，执行广度优先搜索，其最优时间复杂度是（ ）。

- [ ] **A.** $O(V)$
- [ ] **B.** $O(V+E)$
- [ ] **C.** $O(V^2)$
- [ ] **D.** $O(E)$

**第 8 题** 以下关于贪心法和动态规划的说法中，错误的是（ ）。

- [ ] **A.** 动态规划能解决大部分多阶段决策问题。
- [ ] **B.** 对特定的问题，贪心法不一定适用。
- [ ] **C.** 当特定的问题适用贪心法时，通常比动态规划的时间复杂度更低。
- [ ] **D.** 对很多问题，递推实现和递归实现动态规划方法的时间复杂度相当。

**第 9 题** 下面程序的输出为（ ）。

```cpp
#include <iostream>
using namespace std;
int main() {
    int N = 15, cnt = 0;
    for (int x = 1; x + x + x <= N; x++)
        for (int y = x; x + y + y <= N; y++)
            for (int z = y; x + y + z <= N; z++)
                cnt++;
    cout << cnt << endl;
    return 0;
}
```

- [ ] **A.** 45
- [ ] **B.** 102
- [ ] **C.** 174
- [ ] **D.** 3375

**第 10 题** 下面程序的时间复杂度为（ ）。

```
int primes[MAXP], num = 0;
bool isPrime[MAXN] = {false};
void sieve() {
    for (int n = 2; n <= MAXN; n++) {
        if (!isPrime[n])
            primes[num++] = n;
        for (int i = 0; i < num && n * primes[i] <= MAXN; i++) {
            isPrime[n * primes[i]] = true;
            if (n % primes[i] == 0)
                break;
        }
    }
}
```

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

**第 11 题** 下列Dijkstra算法，假设图 $graph$ 中顶点数 $v$、边数 $e$，则程序的时间复杂度为（ ）。

```
typedef struct Edge {
    int in, out;     // 从下标in顶点到下标out顶点的边
    int len;         // 边长度
    struct Edge * next;
} Edge;
// v: 顶点个数, graph: 出边邻接表, start: 起点下标, dis: 输出每个顶点的最短距离
void dijkstra(int v, Edge * graph[], int start, int * dis) {
    const int MAX_DIS = 0x7fffff;
    for (int i = 0; i < v; i++)
        dis[i] = MAX_DIS;
    dis[start] = 0;
    int * visited = new int[v];
    for (int i = 0; i < v; i++)
        visited[i] = 0;
    visited[start] = 1;
    for (int t = 0; ; t++) {
        int min = MAX_DIS, minv = -1;
        for (int i = 0; i < v; i++) {
            if (visited[i] == 0 && min > dis[i]) {
                min = dis[i];
                minv = i;
            }
        }
        if (minv < 0)
            break;
        visited[minv] = 1;
        for (Edge * e = graph[minv]; e != NULL; e = e->next)
            if (dis[e->out] > e->len)
                dis[e->out] = e->len;
    }
    delete[] visited;
}
```

- [ ] **A.** $O(v^2)$
- [ ] **B.** $O(v \log v + e)$
- [ ] **C.** $O((v + e) \log v)$
- [ ] **D.** $O(v + e)$

**第 12 题** 下面 `count_triple` 函数的时间复杂度为( )。

```cpp
int gcd(int m, int n) {
    if (m == 0) return n;
    return gcd(n % m, m);
}
int count_triple(int n) {
    int cnt = 0;
    for (int v = 1; v * v * 4 <= n; v++)
        for (int u = v + 1; u * (u + v) * 2 <= n; u += 2)
            if (gcd(u, v) == 1) {
                int a = u * u - v * v;
                int b = u * v * 2;
                int c = u * u + v * v;
                cnt += n / (a + b + c);
            }
    return cnt;
}
```

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

**第 13 题** 下面 `merge_sort` 函数试图实现归并排序算法，横线处应该填入的是（ ）。

```cpp
#include <vector>
using namespace std;
void merge_sort(vector<int> & arr, int left, int right) {
    if (right - left <= 1)
        return;

    int mid = (left + right) / 2;
    merge_sort(________); // 在此处填入选项
    merge_sort(________); // 在此处填入选项

    vector<int> temp(right - left);
    int i = left, j = mid, k = 0;
    while (i < mid && j < right)
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
            temp[k++] = arr[j++];
    while (i < mid)
        temp[k++] = arr[i++];
    while (j < right)
        temp[k++] = arr[j++];
    for (i = left, k = 0; i < right; ++i, ++k)
        arr[i] = temp[k];
}
```

- [ ] **A.**

```cpp
arr, left, mid
arr, mid, right
```

- [ ] **B.**

```cpp
arr, left, mid + 1
arr, mid + 1, right
```

- [ ] **C.**

```
arr, left, mid
arr, mid + 1, right
```

- [ ] **D.**

```
arr, left, mid + 1
arr, mid + 1, right + 1
```

**第 14 题** 下面Prim算法程序中，横线处应该填入的是（ ）。

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int prim(vector<vector<int>> & graph, int n) {
    vector<int> key(n, INT_MAX);
    vector<int> parent(n, -1);
    key[0] = 0;
    for (int i = 0; i < n; i++) {
        int u = min_element(key.begin(), key.end()) - key.begin();
        if (key[u] == INT_MAX)
            break;
        for (int v = 0; v < n; v++) {
            if (___________) { // 在此处填入选项
                key[v] = graph[u][v];
                parent[v] = u;
            }
        }
    }
    int sum = 0;
    for (int i = 0; i < n; i++) {
        if (parent[i] != -1) {
            cout << "Edge: " << parent[i] << " - " << i << " Weight: " << key[i] << endl;
            sum += key[i];
        }
    }
    return sum;
}
int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> graph(n, vector<int>(n, 0));
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u][v] = w;
        graph[v][u] = w;
    }
    int result = prim(graph, n);
    cout << "Total weight of the minimum spanning tree: " << result << endl;
    return 0;
}
```

- [ ] **A.**

```
graph[u][v] >= 0 && key[v] > graph[u][v]
```

- [ ] **B.**

```
graph[u][v] <= 0 && key[v] > graph[u][v]
```

- [ ] **C.**

```
graph[u][v] == 0 && key[v] > graph[u][v]
```

- [ ] **D.**

```
graph[u][v] != 0 && key[v] > graph[u][v]
```

**第 15 题** 下面的程序使用出边邻接表表达的带权无向图，则从顶点0到顶点3的最短距离为（ ）。

```
#include <vector>
using namespace std;
class Edge {
public:
    int dest;
    int weight;
    Edge(int d, int w) : dest(d), weight(w) {}
};
class Graph {
private:
    int num_vertex;
    vector<vector<Edge>> vve;
public:
    Graph(int v) : num_vertex(v), vve(v) {}
    void addEdge(int s, int d, int w) {
        vve[s].emplace_back(d, w);
        vve[d].emplace_back(s, w)
    }
};
int main() {
    Graph g(4);
    g.addEdge(0, 1, 8);
    g.addEdge(0, 2, 5);
    g.addEdge(1, 2, 1);
    g.addEdge(1, 3, 3);
    g.addEdge(2, 3, 7);
    return 0;
}
```

- [ ] **A.** 12
- [ ] **B.** 11
- [ ] **C.** 10
- [ ] **D.** 9

## 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 `'9' ^ 3` 的结果值为 `'999'` 。

**第 2 题** 下列C++语言代码，能够安全地输出 `arr[5]` 的值。

```
int n = 5;
int arr[n] = {1, 2, 3};
std::cout << arr[5];
```

**第 3 题** 对$n$个元素的数组进行排序，最差情况的时间复杂度为 $O(n^2)$。

**第 4 题** 有4个红球、3个蓝球和2个绿球排成一排（相同色球视为完全相同），则不同的排列方案数为1260种。

**第 5 题** 使用 `math.h` 或 `cmath` 头文件中的函数，对于 `int` 类型的变量 `x` ，表达式 `fabs(x)` 和 `sqrt(x * x)` 的结果总是近似相等的。

**第 6 题** 运算符重载是C++语言静态多态的一种典型体现，而使用C语言则无法实现运算符重载。

**第 7 题** 存在一个简单无向图满足：顶点数为6，边数为8，6个顶点的度数分别为3、3、3、3、2、2。

**第 8 题** 已知两个 `double` 类型的变量 `r` 和 `theta` 分别表示一个扇形的圆半径及圆心角（弧度），则扇形的周长可以通过表达式 `(2 + theta) * r` 求得。

**第 9 题** Dijkstra算法的时间复杂度为 $O(V^2)$，其中 $V$ 为图中顶点的数量。

**第 10 题** 从32名学生中选出2人分别担任男生班长和女生班长（男生班长必须是男生，女生班长必须是女生），则共有 $C(32,2)/2$ 种不同的选法。

# 3 编程题（每题 25 分，共 50 分）

## 3.1 编程题 1

- **试题名称**：最短距离
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

### 3.1.1 题目描述

给定正整数 $p,q$ 以及常数 $N=10^{18}$。现在构建一张包含 $N$ 个结点的带权无向图，结点依次以 $1,2,\ldots,N$ 编号。对于任意满足 $1\le u<v\le N$ 的 $u,v$，向图中加入一条连接结点 $u$ 与结点 $v$ 的无向边，边权取决于 $u,v$ 是否互质：

- 若 $u,v$ 互质（即 $u,v$ 的最大公因数为 1），则连接结点 $u$ 与结点 $v$ 的无向边长度为 $p$;
- 否则连接结点 $u$ 与结点 $v$ 的无向边长度为 $q$。

现在给定 $n$ 组询问，第 $i$（$1\le i\le n$）组询问给定两个正整数 $a_i,b_i$，你需要回答结点 $a_i$ 与结点 $b_i$ 之间的最短距离。

### 3.1.2 输入格式

第一行，三个正整数 $n,p,q$，分别表示询问数量，结点编号互质时的边权，以及结点编号不互质时的边权。

接下来 $n$ 行，每行两个正整数 $a_i,b_i$，表示一组询问。

### 3.1.3 输出格式

输出共 $n$ 行，每行一个整数，表示结点 $a_i$ 与结点 $b_i$ 之间的最短距离。

### 3.1.4 样例

#### 3.1.4.1 输入样例 1

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

#### 3.1.4.2 输出样例 1

```
4
4
3
4
```

#### 3.1.4.3 输入样例 2

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

#### 3.1.4.4 输出样例 2

```
2
2
4
2
0
```

### 3.1.5 数据范围

对于 30% 的测试点，保证 $1 \le n \le 10$，$1 \le a_i, b_i \le 50$。

对于另外 30% 的测试点，保证 $1 \le a_i, b_i \le 250$。

对于所有测试点，保证 $1 \le n \le 10^4$，$1 \le a_i, b_i \le 10^9$，$1 \le p, q \le 10^9$。

### 3.1.6 参考程序

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

using namespace std;

const int N = 1e5 + 5;

int n, p, q;
int a, b;
int ans;

int gcd(int a, int b) {
    if (!a || !b) return a + b;
    return gcd(b, a % b);
}

int main() {
    scanf("%d%d%d", &n, &p, &q);
    while (n--) {
        scanf("%d%d", &a, &b);
        if (a == b)
            ans = 0;
        else if (a == 1 || b == 1)
            ans = p;
        else {
            ans = min(p + p, q + q);
            if (gcd(a, b) == 1)
                ans = min(ans, p);
            else
                ans = min(ans, q);
        }
        printf("%d\n", ans);
    }
    return 0;
}
```

## 3.2 编程题 2

- **试题名称**：最小生成树
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

### 3.2.1 题目描述

给定一张包含 $n$ 个结点 $m$ 条边的**带权连通无向图**，结点依次以 $1, 2, \ldots, n$ 编号，第 $i$ 条边（$1 \le i \le m$）连接结点 $u_i$ 与结点 $v_i$，边权为 $w_i$。

对于每条边，请你求出从图中移除该条边后，图的最小生成树中所有边的边权和。特别地，若移除某条边后图的最小生成树不存在，则输出 $-1$。

### 3.2.2 输入格式

第一行，两个正整数 $n, m$，分别表示图的结点数与边数。

接下来 $m$ 行中的第 $i$ 行（$1 \le i \le m$）包含三个正整数 $u_i, v_i, w_i$，表示图中连接结点 $u_i$ 与结点 $v_i$ 的边，边权为 $w_i$。

### 3.2.3 输出格式

输出共 $m$ 行，第 $i$ 行（$1 \le i \le m$）包含一个整数，表示移除第 $i$ 条边后，图的最小生成树中所有边的边权和。若移除第 $i$ 条边后图的最小生成树不存在，则输出 $-1$。

### 3.2.4 样例

#### 3.2.4.1 输入样例 1

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

#### 3.2.4.2 输出样例 1

```
14
15
-1
-1
10
```

#### 3.2.4.3 输入样例 2

```
6 10
1 2 6
2 3 3
3 1 4
3 4 5
4 5 8
5 6 2
6 4 1
3 2 4
5 4 4
3 3 6
```

#### 3.2.4.4 输出样例 2

```
15
16
17
-1
15
17
18
15
15
15
```

### 3.2.5 数据范围

| 子任务编号 | 测试点占比 | $n$ | $m$ | 特殊性质 |
|---|---|---|---|---|
| 1 | 20% | $\le 50$ | $\le 100$ | - |
| 2 | 30% | $\le 10^5$ | $\le 10^5$ | $n=m$ |
| 3 | 30% | $\le 500$ | $\le 2\times 10^4$ | - |
| 4 | 20% | $\le 10^5$ | $\le 10^5$ | - |

对于所有测试点，保证 $1\le n\le 10^5$，$1\le m\le 10^5$，$1\le u_i,v_i\le n$，$1\le w_i\le 10^9$。

### 3.2.6 参考程序

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

using namespace std;

const int N = 1e5 + 5;
const int M = 2e5 + 5;
const long long oo = 1e18;

int n, m;
int u[M], v[M], w[M], p[M];
int h[N], id[M], nx[M], et;
int f[N], mark[M];
long long s, ans[M];
int dep[N], pid[N];

bool cmp(int x, int y) {
    return w[x] < w[y];
}

int getf(int u) {
    return f[u] ? f[u] = getf(f[u]) : u;
}

void link(int x, int p) {
    id[++et] = p;
    nx[et] = h[x];
    h[x] = et;
}

void dfs(int x, int f=0, int p=0) {
    dep[x] = dep[f] + 1;
    pid[x] = p;
    for (int i = h[x]; i; i = nx[i]) {
        int to = u[id[i]] ^ v[id[i]] ^ x;
        if (to != f)
            dfs(to, x, id[i]);
```

```
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= m; i++) {
        scanf("%d%d%d", &u[i], &v[i], &w[i]);
        p[i] = i;
    }
    sort(p + 1, p + m + 1, cmp);
    for (int i = 1; i <= m; i++) {
        int x = u[p[i]], y = v[p[i]];
        if (getf(x) == getf(y))
            continue;
        mark[p[i]] = 1;
        f[getf(x)] = y;
        s += w[p[i]];
        link(x, p[i]);
        link(y, p[i]);
    }
    for (int i = 1; i <= m; i++)
        ans[i] = mark[i] ? oo : s;
    dfs(1);
    for (int i = 1; i <= n; i++)
        f[i] = 0;
    for (int i = 1; i <= m; i++) {
        if (mark[p[i]])
            continue;
        int x = getf(u[p[i]]), y = getf(v[p[i]]);
        while (x != y) {
            if (dep[x] < dep[y])
                x ^= y ^= x ^= y;
            int to = u[pid[x]] ^ v[pid[x]] ^ x;
            ans[pid[x]] = s - w[pid[x]] + w[p[i]];
            f[x] = to;
            x = getf(x);
        }
    }
    for (int i = 1; i <= m; i++)
        printf("%lld\n", ans[i] < oo ? ans[i] : -1);
    return 0;
}
```