CCF 编程能力等级认证

Grade Examination of Software Programming

C++ 七级

2025 年 06 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	C	B	D	A	D	A	B	D	B	A	D	C	A	B

第 1 题 已知小写字母 `b` 的ASCII码为98，下列C++代码的输出结果是（ ）。

```
#include <iostream>
using namespace std;
int main() {
    char a = 'b' ^ 4;
    cout << a;
    return 0;
}
```

- [ ] **A.** `b`
- [ ] **B.** `bbbb`
- [ ] **C.** `f`
- [ ] **D.** `102`

第 2 题 已知 `a` 为 `int` 类型变量， `p` 为 `int *` 类型变量，下列赋值语句不符合语法的是（ ）。

- [ ] **A.** `*(p + a) = *p;`
- [ ] **B.** `*(p - a) = a;`
- [ ] **C.** `p + a = p;`
- [ ] **D.** `p = p + a;`

第 3 题 下列关于C++类的说法，错误的是( )。

- [ ] **A.** 如需要使用基类的指针释放派生类对象，基类的析构函数应声明为虚析构函数。
- [ ] **B.** 构造派生类对象时，只调用派生类的构造函数，不会调用基类的构造函数。
- [ ] **C.** 基类和派生类分别实现了同一个虚函数，派生类对象仍能够调用基类的该方法。
- [ ] **D.** 如果函数形参为基类指针，调用时可以传入派生类指针作为实参。

第 4 题 下列C++代码的输出是（ ）。

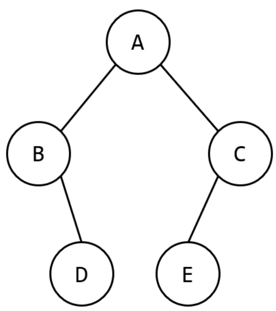
```cpp
#include <iostream>
using namespace std;
int main() {
    int arr[5] = {2, 4, 6, 8, 10};
    int * p = arr + 2;
    cout << p[3] << endl;
    return 0;
}
```

- [ ] **A.** `6`
- [ ] **B.** `8`
- [ ] **C.** 编译出错，无法运行。
- [ ] **D.** 不确定，可能发生运行时异常。

第 5 题 假定只有一个根节点的树的深度为1，则一棵有N个节点的完全二叉树，则树的深度为( )。

- [ ] **A.** $\lfloor \log_2(N) \rfloor + 1$
- [ ] **B.** $\lfloor \log_2(N) \rfloor$
- [ ] **C.** $\lceil \log_2(N) \rceil$。
- [ ] **D.** 不能确定。

第 6 题 对于如下图的二叉树，说法正确的是（ ）。



- [ ] **A.** 先序遍历是 `ABDEC` 。
- [ ] **B.** 中序遍历是 `BDACE` 。
- [ ] **C.** 后序遍历是 `DBCEA` 。
- [ ] **D.** 广度优先遍历是 `ABCDE` 。

第 7 题 图的存储和遍历算法，下面说法错误的是（ ）。

- [ ] **A.** 图的深度优先遍历须要借助队列来完成。
- [ ] **B.** 图的深度优先遍历和广度优先遍历对有向图和无向图都适用。
- [ ] **C.** 使用邻接矩阵存储一个包含v个顶点的有向图，统计其边数的时间复杂度为$O(v^2)$。
- [ ] **D.** 同一个图分别使用出边邻接表和入边邻接表存储，其边结点个数相同。

第 8 题 一个连通的简单有向图，共有28条边，则该图至少有( )个顶点。

- [ ] **A.** 5

- [ ] **B.** 6

- [ ] **C.** 7

- [ ] **D.** 8

第 9 题 以下哪个方案不能合理解决或缓解哈希表冲突（ ）。

- [ ] **A.** 在每个哈希表项处，使用不同的哈希函数再建立一个哈希表，管理该表项的冲突元素。

- [ ] **B.** 在每个哈希表项处，建立二叉排序树，管理该表项的冲突元素。

- [ ] **C.** 使用不同的哈希函数建立额外的哈希表，用来管理所有发生冲突的元素。

- [ ] **D.** 覆盖发生冲突的旧元素。

第 10 题 以下关于动态规划的说法中，错误的是（ ）。

- [ ] **A.** 动态规划方法通常能够列出递推公式。

- [ ] **B.** 动态规划方法的时间复杂度通常为状态的个数。

- [ ] **C.** 动态规划方法有递推和递归两种实现形式。

- [ ] **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 11 题 下面程序的输出为（ ）。

```cpp
#include <iostream>
using namespace std;
int rec_fib[100];
int fib(int n) {
    if (n <= 1)
        return n;
    if (rec_fib[n] == 0)
        rec_fib[n] = fib(n - 1) + fib(n - 2);
    return rec_fib[n];
}
int main() {
    cout << fib(6) << endl;
    return 0;
}
```

- [ ] **A.** 8

- [ ] **B.** 13

- [ ] **C.** 64

- [ ] **D.** 结果是随机的。

第 12 题 下面程序的时间复杂度为（ ）。

```cpp
int rec_fib[MAX_N];
int fib(int n) {
    if (n <= 1)
        return n;
    if (rec_fib[n] == 0)
        rec_fib[n] = fib(n - 1) + fib(n - 2);
    return rec_fib[n];
}
```

☐ **A.** $O(2^n)$

☐ **B.** $O(\phi^n), \phi = \frac{\sqrt{5}-1}{2}$

☐ **C.** $O(n^2)$

☐ **D.** $O(n)$

第 13 题 下面 `search` 函数的平均时间复杂度为( )。

```
int search(int n, int * p, int target) {
    int low = 0, high = n;
    while (low < high) {
        int middle = (low + high) / 2;
        if (target == p[middle]) {
            return middle;
        } else if (target > p[middle]) {
            low = middle + 1;
        } else {
            high = middle;
        }
    }
    return -1;
}
```

☐ **A.** $O(n\log(n))$

☐ **B.** $O(n)$

☐ **C.** $O(\log(n))$

☐ **D.** $O(1)$

第 14 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;
        }
    }
}
```

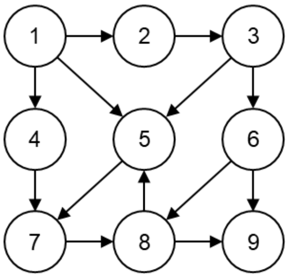
☐ **A.** $O(n)$

☐ **B.** $O(n \times \log n)$

☐ **C.** $O(n \times \log\log n)$

☐ **D.** $O(n^2)$

第 15 题 下列选项中，哪个不可能是下图的广度优先遍历序列（ ）。



- [ ] **A.** 1, 2, 4, 5, 3, 7, 6, 8, 9
- [ ] **B.** 1, 2, 5, 4, 3, 7, 8, 6, 9
- [ ] **C.** 1, 4, 5, 2, 7, 3, 8, 6, 9
- [ ] **D.** 1, 5, 4, 2, 7, 3, 8, 6, 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 & 12` 的结果类型为 `int` 、值为 `8` 。

第 2 题 C++语言中，指针变量指向的内存地址不一定都能够合法访问。

第 3 题 对n个元素的数组进行快速排序，最差情况的时间复杂度为$O(n\log n)$。

第 4 题 一般情况下， `long long` 类型占用的字节数比 `float` 类型多。

第 5 题 使用 `math.h` 或 `cmath` 头文件中的函数，表达式 `pow(10, 3)` 的结果的值为 `1000` 、类型为 `int` 。

第 6 题 二叉排序树的中序遍历序列一定是有序的。

第 7 题 无论哈希表采用何种方式解决冲突，只要管理的元素足够多，都无法避免冲突。

第 8 题 在C++语言中，类的构造函数和析构函数均可以声明为虚函数。

第 9 题 动态规划方法将原问题分解为一个或多个相似的子问题，因此必须使用递归实现。

第 10 题 如果将城市视作顶点，公路视作边，将城际公路网络抽象为简单图，可以满足城市间的车道级导航需求。

3 编程题（每题 25 分，共 50 分）

3.1 编程题 1

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

3.1.1 题目描述

给定由 n 个结点与 m 条边构成的简单无向图 G，结点依次以 $1,2,\dots,n$ 编号。简单无向图意味着 G 中不包含重边与自环。G 的**线图** $L(G)$ 通过以下方式构建：

- 初始时线图 $L(G)$ 为空。

- 对于无向图 G 中的一条边，在线图 $L(G)$ 中加入与之对应的一个结点。
- 对于无向图 G 中两条不同的边 $(u_1,v_1),(u_2,v_2)$，若存在 G 中的结点同时连接这两条边（即 u_1,v_1 之一与 u_2,v_2 之一相同），则在线图 $L(G)$ 中加入一条无向边，连接 $(u_1,v_1),(u_2,v_2)$ 在线图中对应的结点。

请你求出线图 $L(G)$ 中所包含的无向边的数量。

3.1.2 输入格式

第一行，两个正整数 n,m，分别表示无向图 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 G 中连接 u_i,v_i 的一条无向边。

3.1.3 输出格式

输出共一行，一个整数，表示线图 $L(G)$ 中所包含的无向边的数量。

3.1.4 样例

3.1.4.1 输入样例 1

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

3.1.4.2 输出样例 1

```
3
```

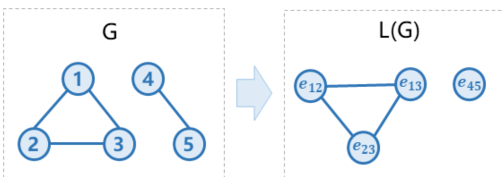
3.1.4.3 输入样例 2

```
5 10
1 2
1 3
1 4
1 5
2 3
2 4
2 5
3 4
3 5
4 5
```

3.1.4.4 输出样例 2

```
30
```

3.1.4.5 样例解释 1



3.1.5 数据范围

对于 60% 的测试点，保证 $1 \le n \le 500$，$1 \le m \le 500$。

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

3.1.6 参考程序

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

const int N = 1e5 + 5;

int n, m, deg[N];
long long ans;

int main() {
    scanf("%d%d", &n, &m);
    while (m--) {
        int u, v;
        scanf("%d%d", &u, &v);
        deg[u]++;
        deg[v]++;
    }
    for (int i = 1; i <= n; i++)
        ans += 1ll * deg[i] * (deg[i] - 1) / 2;
    printf("%lld\n", ans);
    return 0;
}
```

3.2 编程题 2

- **试题名称**：调味平衡
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

小 A 准备了 n 种食材用来制作料理，这些食材依次以 $1, 2, \ldots, n$ 编号，第 i 种食材的酸度为 a_i，甜度为 b_i。对于每种食材，小 A 可以选择将其放入料理，或者不放入料理。料理的酸度 A 为放入食材的酸度之和，甜度 B 为放入食材的甜度之和。如果料理的酸度与甜度相等，那么料理的调味是**平衡的**。

过于清淡的料理并不好吃，因此小 A 想在满足料理调味平衡的前提下，合理选择食材，最大化料理的酸度与甜度之和。你能帮他求出在调味平衡的前提下，料理酸度与甜度之和的最大值吗?

3.2.2 输入格式

第一行，一个正整数 n，表示食材种类数量。

接下来 n 行，每行两个正整数 a_i, b_i，表示食材的酸度与甜度。

3.2.3 输出格式

输出共一行，一个整数，表示在调味平衡的前提下，料理酸度与甜度之和的最大值。

3.2.4 样例

3.2.4.1 输入样例 1

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

3.2.4.2 输出样例 1

```
8
```

3.2.4.3 输入样例 2

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

3.2.4.4 输出样例 2

```
2
```

3.2.5 数据范围

对于 40% 的测试点，保证 $1 \le n \le 10$，$1 \le a_i, b_i \le 10$。

对于另外 20% 的测试点，保证 $1 \le n \le 50$，$1 \le a_i, b_i \le 10$。

对于所有测试点，保证 $1 \le n \le 100$，$1 \le a_i, b_i \le 500$。

3.2.6 参考程序

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

const int N = 105;
const int C = 505;
const int D = N * C * 2;

int n;
int f[D];

int main() {
    scanf("%d", &n);
    for (int i = 0; i < D; i++)
        f[i] = -1e9;
    f[N * C] = 0;
    while (n--) {
        int a, b;
        scanf("%d%d", &a, &b);
        int x = a + b, y = a - b;
        if (y <= 0) {
            for (int i = -y; i < D; i++)
                f[i + y] = max(f[i + y], f[i] + x);
        } else {
            for (int i = D - y - 1; i; i--)
                f[i + y] = max(f[i + y], f[i] + x);
```

```
        }
    }
    printf("%d\n", f[N * C]);
    return 0;
}
```