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

C++ 六级

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

第 1 题 下列代码执行后的输出结果是（　　）。

```
class Animal {
    public:
    virtual void speak() {
        cout << "Animal ";
    }
    virtual ~Animal() = default;
};

class Cat : public Animal {
    public:
    void speak() override {
        cout << "Cat ";
    }
};

int main() {
    Animal *p = new Cat();
    p->speak();
    delete p;
    return 0;
}
```

- A. `Animal`
- B. `Cat`
- C. `Animal Cat`
- D. 编译错误

第 2 题 下列代码中，横线处应填写（　　），才能正确调用基类的带参数构造函数。

```
class Machine {
    protected:
    string id;

    public:
    Machine(string s) : id(s) {}
};

class Robot : public Machine {
    int level;

    public:
    Robot(string s, int n) : __________, level(n) {}
};
```

- [ ] **A.** `Machine(s)`
- [ ] **B.** `Machine::id(s)`
- [ ] **C.** `super(s)`
- [ ] **D.** `id(s)`

第 3 题 下列代码执行后的输出顺序是（　　）。

```
class Base {
    public:
    Base() {
        cout << "B ";
    }
    virtual ~Base() {
        cout << "~B ";
    }
};

class Derived : public Base {
    public:
    Derived() {
        cout << "D ";
    }
    ~Derived() {
        cout << "~D ";
    }
};

int main() {
    Base *p = new Derived();
    delete p;
    return 0;
}
```

- [ ] **A.** `B D ~B ~D`
- [ ] **B.** `D B ~D ~B`
- [ ] **C.** `B D ~D ~B`
- [ ] **D.** `B D ~B`

第 4 题 下列代码执行后的输出结果是（　　）。

```cpp
stack<int> s;
queue<int> q;
for (int i = 2; i <= 6; i += 2) {
    s.push(i);
    q.push(i);
}
s.pop();
q.pop();
cout << s.top() << " " << q.front();
```

- [ ] **A.** `2 4`
- [ ] **B.** `4 4`
- [ ] **C.** `4 6`
- [ ] **D.** `6 2`

第 5 题 下面循环队列采用“空出一个位置”的方式区分队空和队满。横线处应填写（　　）。

```cpp
const int MAXN = 8;
int data[MAXN];
int front = 0, rear = 0;

bool full() {
    return ____________________________;
}
```

- [ ] **A.** `rear == front`
- [ ] **B.** `(front + 1) % MAXN == rear`
- [ ] **C.** `(rear + 1) % MAXN == front`
- [ ] **D.** `rear == MAXN - 1`

第 6 题 下列函数实现了二叉树的哪种遍历方式（　　）。

```cpp
void visit(TreeNode *root) {
    if (root == nullptr)
        return;
    visit(root->left);
    cout << root->val << " ";
    visit(root->right);
}
```

- [ ] **A.** 前序遍历
- [ ] **B.** 中序遍历
- [ ] **C.** 后序遍历
- [ ] **D.** 层序遍历

第 7 题 已知一棵二叉树的先序遍历序列为 `A B D E C F`，中序遍历序列为 `D B E A C F`，则其后序遍历序列是（　　）。

- [ ] **A.** `D E B F C A`
- [ ] **B.** `D B E F C A`
- [ ] **C.** `E D B F C A`
- [ ] **D.** `D E B C F A`

第 8 题 下面函数用于计算二叉树的高度，横线处应填写（　　）。

```
int height(TreeNode *root) {
    if (root == nullptr)
        return 0;
    int leftH = height(root->left);
    int rightH = height(root->right);
    return ____________________________;
}
```

- [ ] **A.** `leftH + rightH`
- [ ] **B.** `min(leftH, rightH) + 1`
- [ ] **C.** `max(leftH, rightH)`
- [ ] **D.** `max(leftH, rightH) + 1`

第 9 题 以下代码实现二叉树左子树优先的深度优先搜索算法，则横线上应填写（　　）。

```
void dfs(TreeNode *root) {
    if (root == nullptr)
        return;

    stack<TreeNode *> s;
    s.push(root);
    while (!s.empty()) {
        TreeNode *node = s.top();
        s.pop();
        cout << node->value << " ";

        ____________________________ // 在此处填入代码
    }
}
```

- [ ] **A.**

```
s.push(node->right);
s.push(node->left);
```

- [ ] **B.**

```
s.push(node->left);
s.push(node->right);
```

- [ ] **C.**

```
if (node->right)
    s.push(node->right);
if (node->left)
    s.push(node->left);
```

- [ ] **D.**

```
if (node->left)
    s.push(node->left);
if (node->right)
    s.push(node->right);
```

第 10 题 下面函数在二叉搜索树中查找值 `x`。横线处应填写（　　）。

```
TreeNode *searchBST(TreeNode *root, int x) {
    if (root == nullptr || root->val == x)
        return root;
    if (x < root->val)
        return searchBST(root->left, x);
    return ____________________________;
}
```

- [ ] **A.** `searchBST(root->left, x)`
- [ ] **B.** `searchBST(root->right, x)`
- [ ] **C.** `searchBST(root, x + 1)`
- [ ] **D.** `root->right`

第 11 题 有 5 个字符，其出现频率分别为 2、4、5、9、12。按哈夫曼算法构造编码树，其最小带权路径长度 WPL 为（　　）。

- [ ] **A.** 68
- [ ] **B.** 69
- [ ] **C.** 71
- [ ] **D.** 73

第 12 题 下面代码用反射法生成 `n` 位格雷编码，横线处应填写（　　）。

```
vector<string> gray(int n) {
    vector<string> ans = {"0", "1"};
    for (int bit = 2; bit <= n; ++bit) {
        int oldSize = ans.size();
        for (int i = oldSize - 1; i >= 0; --i)
            ans.push_back(____________________________);
        for (int i = 0; i < oldSize; ++i)
            ans[i] = "0" + ans[i];
    }
    return ans;
}
```

- [ ] **A.** `ans[i] + "1"`
- [ ] **B.** `"0" + ans[i]`
- [ ] **C.** `"1" + ans[i]`
- [ ] **D.** `ans[oldSize - i - 1]`

第 13 题 下面代码计算走到第 `n` 级台阶的方法数，每次可以走 1 级或 2 级。横线处应填写（　　）。

```
int ways(int n) {
    if (n <= 2)
        return n;
    vector<int> dp(n + 1);
    dp[1] = 1;
    dp[2] = 2;
    for (int i = 3; i <= n; ++i)
        dp[i] = ____________________________;
    return dp[n];
}
```

- [ ] **A.** `dp[i - 1] + 1`

- [ ] **B.** `dp[i - 1] + dp[i - 2]`
- [ ] **C.** `dp[i - 2] + 2`
- [ ] **D.** `2 * dp[i - 1]`

第 14 题 下面代码求从包含非负元素的数组中选择若干个互不相邻元素所能得到的最大和。横线处应填写（　　）。

```
int maxSum(vector<int> &a) {
    int n = a.size();
    if (n == 0)
        return 0;
    if (n == 1)
        return a[0];
    vector<int> dp(n);
    dp[0] = a[0];
    dp[1] = max(a[0], a[1]);
    for (int i = 2; i < n; ++i)
        dp[i] = ____________________________;
    return dp[n - 1];
}
```

- [ ] **A.** `dp[i - 1] + a[i]`
- [ ] **B.** `dp[i - 2] + a[i]`
- [ ] **C.** `max(dp[i - 1], dp[i - 2] + a[i])`
- [ ] **D.** `max(dp[i - 1], a[i])`

第 15 题 下面是一维数组实现的 0/1 背包。内层循环必须从大到小枚举容量，主要原因是（　　）。

```
for (int i = 0; i < n; ++i) {
    for (int w = W; w >= weight[i]; --w) {
        dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
    }
}
```

- [ ] **A.** 保证每件物品最多被选择一次
- [ ] **B.** 保证物品必须按照重量从大到小选择
- [ ] **C.** 降低时间复杂度到 $O(n)$
- [ ] **D.** 防止数组 `dp` 发生越界

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 下列代码可以正常编译，因为编译器会自动为 `Student` 类生成一个无参数构造函数。

```cpp
class Student {
    public:
    Student(int x) {
        age = x;
    }

    private:
    int age;
};

int main() {
    Student s;
}
```

第 2 题 下列代码合法，因为派生类可以直接访问基类的私有成员 `value` 。

```cpp
class Base {
    private:
    int value = 10;
};

class Child : public Base {
    public:
    int get() {
        return value;
    }
};
```

第 3 题 下列代码执行后，输出结果为 `30` 。

```cpp
queue<int> q;
q.push(10);
q.push(20);
q.push(30);
q.pop();
cout << q.front();
```

第 4 题 一棵完全二叉树按照从上到下、从左到右的顺序，将节点依次存储在数组 `tree[1]` 、 `tree[2]` 、…… 中。若节点 `tree[i]` 存在左孩子，则其左孩子存储在 `tree[2 * i]` 中。

第 5 题 对任意一棵二叉搜索树执行中序遍历，得到的关键字序列一定是非递减的。

```cpp
void inorder(TreeNode *root) {
    if (!root)
        return;
    inorder(root->left);
    cout << root->val << " ";
    inorder(root->right);
}
```

第 6 题 若使用下列代码从节点 `start` 开始访问一棵树，则第一次到达某个节点时所经过的边数，一定是从 `start` 到该节点的最少边数。

```cpp
vector<int> tree[100];
bool visited[100];
int dist[100];

void search(int start) {
    queue<int> q;
    q.push(start);
    visited[start] = true;
    dist[start] = 0;

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v : tree[u]) {
            if (!visited[v]) {
                visited[v] = true;
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

第 7 题 哈夫曼编码的生成过程基于贪心算法，出现频率越高的字符，其编码长度一定不会比出现频率更低的字符更长。

第 8 题 在 n 位格雷码中，任意两个编码之间都只相差一个二进制位。

第 9 题 下列一维动态规划代码实现的是完全背包问题，因为在处理第 `i` 种物品时，同一种物品可能被重复选择。

```cpp
for (int i = 0; i < n; ++i) {
    for (int w = weight[i]; w <= W; ++w) {
        dp[w] = max(dp[w], dp[w - weight[i]] + value[i]);
    }
}
```

第 10 题 下列递归程序能得到正确的斐波那契数，其时间复杂度和空间复杂度都是 $O(n)$。

```cpp
int fib(int n) {
    if (n <= 1)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

3　编程题（每题 25 分，共 50 分）

3.1　编程题 1

- **试题名称**：数组划分
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1　题目描述

给定 n 个整数构成的数组 $A = [a_1, a_2, \ldots, a_n]$。

你需要将数组 A 划分为若干非空连续子段。对于划分得到的某个子段，它的偏差值定义为子段内整数和的平方。划分方案的偏差值定义为所有子段偏差值之和。

你需要最小化划分方案的偏差值。

形式化地，你可以将 A 划分为若干非空连续子段 $A_1, A_2, \ldots, A_k$，使得 $A = A_1 + A_2 + \ldots A_k$，这里的 $+$ 代表数组的连接。对于 $1 \leq i \leq k$，设数组 $A_i = [a_1^{(i)}, \ldots, a_{m_i}^{(i)}]$ 包含 m_i 个整数。你需要最小化 $\sum_{i=1}^{k} \left(\sum_{j=1}^{m_i} a_j^{(i)} \right)^2$。

3.1.2 输入格式

第一行，一个正整数 n，表示数组 A 的长度。

第二行，n 个整数 $a_1, a_2, \ldots, a_n$，表示数组 A。

3.1.3 输出格式

一行，一个整数，表示划分方案偏差值的最小值。

3.1.4 样例

3.1.5 输入样例 1

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

3.1.6 输出样例 1

```
6
```

3.1.7 输入样例 2

```
6
-1 -1 4 -5 -1 4
```

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 $0 \leq a_i \leq 50$。

对于所有测试点，保证 $1 \leq n \leq 2000$，$-100 \leq a_i \leq 100$。

3.1.10 参考程序

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

const int N = 2005;
const long long oo = 1e18;

int n;
int a[N], pre[N];
long long f[N];

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &a[i]);
        pre[i] = pre[i - 1] + a[i];
    }
    for (int i = 1; i <= n; i++) {
        f[i] = oo;
        for (int j = 0; j < i; j++)
            f[i] = min(f[i], f[j] + 1ll * (pre[i] - pre[j]) * (pre[i] - pre[j]));
    }
    printf("%lld\n", f[n]);
    return 0;
}
```

3.2 编程题 2

- **试题名称**：分树规划
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

老师有一棵有 n 个结点的树，结点依次以 $1, 2, \ldots, n$ 编号。

老师想将这棵树作为奖品分给两位同学。具体而言，老师会选择一条边并从树上删去它，从而将这棵树分为两个连通块。两位同学分别可以得到其中一个连通块。

如果有同学拿到的连通块结点数明显小于另一位同学，那么这位同学会不太高兴。为了避免这种情况出现，老师想知道两个连通块结点数之差的绝对值最小是多少。

3.2.2 输入格式

第一行，一个正整数 n，表示结点数量。

接下来 $n-1$ 行，每行两个正整数 u_i, v_i，表示一条连接结点 u_i 和结点 v_i 的边。

3.2.3 输出格式

输出一行，一个整数，表示答案。

3.2.4 样例

3.2.5 输入样例 1

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

3.2.6 输出样例 1

```
0
```

3.2.7 输入样例 2

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

3.2.8 输出样例 2

```
2
```

3.2.9 数据范围

对于 40% 的测试点，保证 $2 \leq n \leq 500$。

对于所有测试点，保证 $2 \leq n \leq 2 \times 10^4$。

3.2.10 参考程序

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

const int N = 2e4 + 5;
const int E = N << 1;

int n;
int h[N], to[E], nx[E], et;
int ans;

void ae(int u, int v) {
    et++;
    to[et] = v;
    nx[et] = h[u];
    h[u] = et;
}

int dfs(int u, int p=0) {
    int sz = 1;
    for (int i = h[u]; i; i = nx[i])
        if (to[i] != p)
            sz += dfs(to[i], u);
    ans = min(ans, abs(n - 2 * sz));
    return sz;
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        ae(u, v);
        ae(v, u);
    }
    ans = n;
    dfs(1);
    printf("%d\n", ans);
    return 0;
}
```