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

C++ 八级

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

第 1 题 一间的机房要安排6名同学进行上机考试，座位共2行3列。考虑到在座位上很容易看到同一行的左右两侧的屏幕，安排中间一列的同学做A卷，左右两列的同学做B卷。请问共有多少种排座位的方案？（ ）。

- [ ] **A.** 720
- [ ] **B.** 90
- [ ] **C.** 48
- [ ] **D.** 15

第 2 题 又到了毕业季，学长学姐们都在开心地拍毕业照。现在有3位学长、3位学姐希望排成一排拍照，要求男生不相邻、女生不相邻。请问共有多少种拍照方案？（ ）。

- [ ] **A.** 720
- [ ] **B.** 72
- [ ] **C.** 36
- [ ] **D.** 2

第 3 题 下列关于C++类和对象的说法，错误的是（ ）。

- [ ] **A.** 通过语句 `const int x = 5;` 定义了一个对象 `x` 。
- [ ] **B.** 通过语句 `std::string t = "12345";` 定义了一个对象 `t` 。
- [ ] **C.** 通过语句 `void (*fp)() = NULL;` 定义了一个对象 `fp` 。
- [ ] **D.** 通过语句 `class MyClass;` 定义了一个类 `MyClass` 。

第 4 题 关于生成树的说法，错误的是（ ）。

- [ ] **A.** 一个无向连通图，一定有生成树。
- [ ] **B.** n个顶点的无向图，其生成树要么不存在，要么一定包含$n-1$条边。
- [ ] **C.** n个顶点、$n-1$条边的无向图，不可能有多颗生成树。
- [ ] **D.** n个顶点、$n-1$条边的无向图，它本身就是自己的生成树。

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。请问这对夫妻生下两个孩子时，实现儿女双全的概率是多少？（ ）。

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

第 6 题 已定义变量 `double a, b;` ，下列哪个表达式可以用来判断一元二次方程$x^2 + ax + b = 0$是否有实根？（ ）。

- [ ] **A.** `4 * b - a * a < 0`
- [ ] **B.** `4 * b <= a * a`
- [ ] **C.** `a * a - 4 * b`
- [ ] **D.** `b * 4 - a * a`

第 7 题 n个结点的二叉树，执行广度优先搜索的平均时间复杂度是（ ）。

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

第 8 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 9 题 下面的 `sum_digit` 函数试图求出从 `1` 到 `n` （包含 `1` 和 `n` ）的数中，包含数字 `d` 的个数。该函数的时间复杂度为（ ）。

```cpp
#include <string>
int count_digit(int n, char d) {
    int cnt = 0;
    std::string s = std::to_string(n);
    for (int i = 0; i < s.length(); i++)
        if (s[i] == d)
            cnt++;
    return cnt;
}
int sum_digit(int n, char d) {
    int sum = 0;
    for (int i = 1; i <= n; i++)
        sum += count_digit(i, d);
    return sum;
}
```

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

第 10 题 下面程序的输出为（ ）。

```
#include <iostream>
const int N = 10;
int ch[N][N][N];
int main() {
    for (int x = 0; x < N; x++)
        for (int y = 0; y < N; y++)
            for (int z = 0; z < N; z++)
                if (x == 0 && y == 0 && z == 0)
                    ch[x][y][z] = 1;
                else {
                    if (x > 0)
                        ch[x][y][z] += ch[x - 1][y][z];
                    if (y > 0)
                        ch[x][y][z] += ch[x][y - 1][z];
                    if (z > 0)
                        ch[x][y][z] += ch[x][y][z - 1];
                }
    std::cout << ch[1][2][3] << std::endl;
    return 0;
}
```

- [ ] **A.** 60
- [ ] **B.** 20
- [ ] **C.** 15
- [ ] **D.** 10

第 11 题 下面 `count_triple` 函数的时间复杂度为( )。

```
int gcd(int a, int b) {
    if (a == 0)
        return b;
    return gcd(b % a, a);
}
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.** $O(n)$
- [ ] **B.** $O(n^2)$

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

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

第 12 题 下面 `quick_sort` 函数试图实现快速排序算法，两处横线处分别应该填入的是（ ）。

```
void swap(int & a, int & b) {
    int temp = a; a = b; b = temp;
}
int partition(int a[], int l, int r) {
    int pivot = a[l], i = l + 1, j = r;
    while (i <= j) {
        while (i <= j && a[j] >= pivot)
            j--;
        while (i <= j && a[i] <= pivot)
            i++;
        if (i < j)
            swap(a[i], a[j]);
    }
    ________;         // 在此处填入选项
    return ________;  // 在此处填入选项
}
void quick_sort(int a[], int l, int r) {
    if (l < r) {
        int pivot = partition(a, l, r);
        quick_sort(a, l, pivot - 1);
        quick_sort(a, pivot + 1, r);
    }
}
```

- [ ] **A.**

```
swap(a[l], a[i])
i
```

- [ ] **B.**

```
swap(a[l], a[j])
i
```

- [ ] **C.**

```
swap(a[l], a[i])
j
```

- [ ] **D.**

```
swap(a[l], a[j])
j
```

第 13 题 下面 `LIS` 函数试图求出最长上升子序列的长度，横线处应该填入的是（ ）。

```
int max(int a, int b) {
    return (a > b) ? a : b;
}
int LIS(vector<int> & nums) {
    int n = nums.size();
    if (n == 0)
        return 0;
    vector<int> dp(n, 1);
    int maxLen = 1;
```

```
    for (int i = 1; i < n; i++) {
        for (int j = 0; j < i; j++)
            if (nums[j] < nums[i])
                ________;  // 在此处填入选项
        maxLen = max(maxLen, dp[i]);
    }
    return maxLen;
}
```

- [ ] **A.**

```
dp[j] = max(dp[j] + 1, dp[i])
```

- [ ] **B.**

```
dp[j] = max(dp[j], dp[i] + 1)
```

- [ ] **C.**

```
dp[i] = max(dp[i] + 1, dp[j])
```

- [ ] **D.**

```
dp[i] = max(dp[i], dp[j] + 1)
```

第 14 题 下面 `LIS` 函数试图求出最长上升子序列的长度，其时间复杂度为（ ）。

```
#define INT_MIN (-1000)
int LIS(vector<int> & nums) {
    int n = nums.size();
    vector<int> tail;
    tail.push_back(INT_MIN);
    for (int i = 0; i < n; i++) {
        int x = nums[i], l = 0, r = tail.size();
        while (l < r) {
            int mid = (l + r) / 2;
            if (tail[mid] < x)
                l = mid + 1;
            else
                r = mid;
        }
        if (r == tail.size())
            tail.push_back(x);
        else
            tail[r] = x;
    }
    return tail.size() - 1;
}
```

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

第 15 题 下面的程序使用邻接矩阵表达的带权无向图，则从顶点0到顶点3的最短距离为（ ）。

```
int weight[4][4] = {
    { 0,  5,  8, 10},
    { 5,  0,  1,  7},
    { 8,  1,  0,  3},
    {10,  7,  3,  0}};
```

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

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 `13` 。

第 2 题 C++语言中，访问数据发生下标越界时，总是会产生运行时错误，从而使程序异常退出。

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

第 4 题 5个相同的红球和4个相同的蓝球排成一排，要求每个蓝球的两侧都必须至少有一个红球，则一共有15种排列方案。

第 5 题 使用 `math.h` 或 `cmath` 头文件中的函数，表达式 `log(8)` 的结果类型为 `double` 、值约为 `3` 。

第 6 题 C++是一种面向对象编程语言，C则不是。继承是面向对象三大特性之一，因此，使用C语言无法实现继承。

第 7 题 n个顶点的无向完全图，有n^{n-2}棵生成树。

第 8 题 已知三个 `double` 类型的变量 `a` 、 `b` 和 `theta` 分别表示一个三角形的两条边长及二者的夹角（弧度），则三角形的周长可以通过表达式 `sqrt(a * a + b * b - 2 * a * b * cos(theta))` 求得。

第 9 题 有V个顶点、E条边的图的深度优先搜索遍历时间复杂度为$O(V+E)$。

第 10 题 从32名学生中选出4人分别担任班长、副班长、学习委员和组织委员，老师要求班级综合成绩排名最后的4名学生不得参选班长或学习委员（仍可以参选副班长和组织委员），则共有$P(30,4)$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**：树上旅行
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.1.1 题目描述

给定一棵有 n 个结点的**有根树**，结点依次以 $1, 2, \ldots, n$ 编号，其中根结点的编号为 1。

小 A 计划在这棵有根树上进行 q 次旅行。在第 i 次旅行中，小 A 首先会选定结点 s_i 作为起点，并移动若干次。移动分为以下两种：

1. 移动至当前结点的父结点。特殊地，如果当前位于根结点，则不进行移动。
2. 移动至当前结点的所有子结点中**编号最小**的结点。特殊地，如果当前位于叶子结点，则不进行移动。

由于移动次数可能很大，对于第 i 次旅行，旅行中的移动将以 k_i 个不为零的整数构成的序列 $a_{i,1}, a_{i,2}, \ldots, a_{i,k_i}$ 表示。对于 $a_{i,j}$，若 $a_{i,j} > 0$ 则代表进行 $a_{i,j}$ 次第一种移动；若 $a_{i,j} < 0$ 则代表进行 $-a_{i,j}$ 次第二种移动。根据给出的序列从左至右完成所有移动后，小 A 所在的结点即是旅行的终点。

给定每次旅行的起点与移动序列，请你求出旅行终点的结点编号。

3.1.2 输入格式

第一行，两个正整数 n, q，分别表示有根树的结点数量，以及旅行次数。

第二行，$n - 1$ 个整数 $p_2, p_3, \ldots, p_n$，其中 p_i 表示结点 i 的父结点编号。

接下来 $2q$ 行中的第 $2i - 1$ 行（$1 \leq i \leq q$）包含两个正整数 s_i, k_i，分别表示第 i 次旅行的起点编号，以及移动序列的长度。第 $2i$ 行包含 k_i 个整数 $a_{i,1}, a_{i,2}, \ldots, a_{i,k_i}$，表示移动序列。

3.1.3 输出格式

输出共 q 行，第 i 行包含一个整数，表示第 i 次旅行终点的结点编号。

3.1.4 样例

3.1.4.1 输入样例 1

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

3.1.4.2 输出样例 1

```
4
1
4
2
```

3.1.4.3 输入样例 2

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

3.1.4.4 输出样例 2

```
1
7
1
```

3.1.5 数据范围

子任务编号	测试点占比	n	q	$\sum k_i$	特殊性质
1	20%	≤ 100	≤ 100	≤ 1000	保证 $a_{i,j}$ 为 1 或 -1
2	20%	$\le 10^4$	$\le 10^4$	$\le 4\times 10^4$	仅包含第一种移动
3	20%	$\le 10^4$	$\le 10^4$	$\le 4\times 10^4$	仅包含第二种移动
4	40%	$\le 10^5$	$\le 2\times 10^4$	$\le 10^5$	-

对于所有测试点，保证 $1\le n\le 10^5$，$1\le q\le 2\times 10^4$，$1\le p_i\le n$，$1\le s_i\le n$，$k_i\ge 1$ 且 $\sum k_i\le 10^5$，$1\le |a_{i,j}|\le n$。

3.1.6 参考程序

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

const int N = 1e5 + 5;
const int L = 18;

int n, q;
int h[N], nx[N];
int par[L][N], son[L][N];

void dfs(int u) {
    for (int i = 1; i < L; i++)
        par[i][u] = par[i - 1][par[i - 1][u]];
    son[0][u] = u;
    for (int i = h[u]; i; i = nx[i]) {
        dfs(i);
        if (son[0][u] == u || i < son[0][u])
            son[0][u] = i;
    }
    for (int i = 1; i < L; i++)
        son[i][u] = son[i - 1][son[i - 1][u]];
}

int move(int go[L][N], int u, int step) {
    for (int i = 0; i < L; i++)
        if ((step >> i) & 1)
            u = go[i][u];
    return u;
}

```

```
int main() {
    scanf("%d%d", &n, &q);
    for (int i = 2; i <= n; i++) {
        scanf("%d", &par[0][i]);
        nx[i] = h[par[0][i]];
        h[par[0][i]] = i;
    }
    par[0][1] = 1;
    dfs(1);
    while (q--) {
        int s, k;
        scanf("%d%d", &s, &k);
        while (k--) {
            int a;
            scanf("%d", &a);
            if (a < 0)
                s = move(son, s, -a);
            else
                s = move(par, s, a);
        }
        printf("%d\n", s);
    }
    return 0;
}
```

3.2 编程题 2

- **试题名称**：遍历计数
- **时间限制**：1.0 s
- **内存限制**：512.0 MB

3.2.1 题目描述

给定一棵有 n 个结点的树 T，结点依次以 $1, 2, \ldots, n$ 标号。树 T 的深度优先遍历序可由以下过程得到：

1. 选定深度优先遍历的起点 s（$1 \le s \le n$），当前所在结点即是起点。
2. 若当前结点存在未被遍历的相邻结点 u 则遍历 u，也即令当前所在结点为 u 并重复这一步；否则回溯。
3. 按照遍历结点的顺序依次写下结点编号，即可得到一组深度优先遍历序。

第一步中起点的选择是任意的，并且第二步中遍历相邻结点的顺序是任意的，因此对于同一棵树 T 可能有多组不同的深度优先遍历序。请你求出树 T 有多少组不同的深度优先遍历序。由于答案可能很大，你只需要求出答案对 10^9 取模之后的结果。

3.2.2 输入格式

第一行，一个整数 n，表示树 T 的结点数。

接下来 $n-1$ 行，每行两个正整数 u_i, v_i，表示树 T 中的一条连接结点 u_i, v_i 的边。

3.2.3 输出格式

输出一行，一个整数，表示树 T 的不同的深度优先遍历序数量对 10^9 取模的结果。

3.2.4 样例

3.2.4.1 输入样例 1

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

3.2.4.2 输出样例 1

```
6
```

3.2.4.3 输入样例 2

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

3.2.4.4 输出样例 2

```
112
```

3.2.5 数据范围

对于 40% 的测试点，保证 $1 \le n \le 8$。

对于另外 20% 的测试点，保证给定的树是一条链。

对于所有测试点，保证 $1 \le n \le 10^5$。

3.2.6 参考程序

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

const int N = 1e5 + 5;
const int mod = 1e9;

int n, deg[N], fac[N];
int pre[N], suf[N];
int ans;

int main() {
    scanf("%d", &n);
    fac[0] = 1;
    for (int i = 1; i <= n; i++)
        fac[i] = 1ll * fac[i - 1] * i % mod;
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        deg[u]++;
        deg[v]++;
    }
    pre[0] = 1;
    for (int i = 1; i <= n; i++)
```

```
        pre[i] = 1ll * pre[i - 1] * fac[deg[i] - 1] % mod;
    suf[n + 1] = 1;
    for (int i = n; i; i--)
        suf[i] = 1ll * suf[i + 1] * fac[deg[i] - 1] % mod;
    for (int i = 1; i <= n; i++)
        ans = (ans + 1ll * pre[i - 1] * fac[deg[i]] % mod * suf[i + 1]) % mod;
    printf("%d\n", ans);
    return 0;
}
```