NOI 入门级完整学习手册

基于 NOI 大纲（2025年修订版） | 含全网调研AI重评分 | 详细知识点讲解与代码示例

评分说明：

黄色背景 表示经过全网深度调研和AI模型重新评估后，评分发生了变化的知识点。

蓝色背景 表示评分与原始大纲一致的知识点。

评分维度：考试重要性（考频）、学习难度、区分度（拉开差距能力）、实用性（竞赛应用广度），综合1-10分。

目 录

第一章 基础知识与编程环境

1.1 计算机的基本构成

原始评分【1】→ AI重评【2】(基础重要，竞赛环境必备)

计算机系统由**硬件**和**软件**两大部分组成。硬件是计算机的物理组成部分，主要包括以下核心组件：

组件	功能说明	竞赛关注点
CPU（中央处理器）	执行指令和运算的核心部件，包含运算器和控制器	理解时间复杂度的物理基础
内存（RAM）	临时存储正在运行的程序和数据，断电后数据丢失	理解空间复杂度、数组大小限制
外存（硬盘/SSD）	永久存储数据和程序	文件读写操作的基础
输入设备	键盘、鼠标等，用于向计算机输入数据	标准输入(stdin)
输出设备	显示器、打印机等，用于输出处理结果	标准输出(stdout)

> **竞赛要点：**在NOI竞赛中，程序的运行时间和内存使用都有严格限制（通常时间1-2秒，内存256MB）。理解CPU执行速度（约10^8-10^9次基本运算/秒）和内存容量对于估算算法可行性至关重要。

1.2 操作系统基本概念

原始评分【1】→ AI重评【2】(环境操作基础，实用性提升)

操作系统（Operating System, OS）是管理计算机硬件和软件资源的系统软件。NOI竞赛环境主要使用**Linux**操作系统（NOI Linux 2.0，基于Ubuntu 20.04）。选手需要了解操作系统的基本功能：进程管理、内存管理、文件系统管理和设备管理。

1.3 计算机网络和Internet基本概念

难度评级【1】

了解计算机网络的基本概念，包括局域网（LAN）、广域网（WAN）、Internet的基本架构、IP地址、域名系统（DNS）等。此部分在CSP-J初赛中偶有考察。

1.4 计算机的历史和常见用途

难度评级【1】

了解计算机发展的重要里程碑：从ENIAC（1946年）到现代计算机。了解图灵（Alan Turing）、冯·诺依曼（John von Neumann）等计算机科学先驱的贡献。冯·诺依曼体系结构（存储程序概念）是现代计算机的基础架构。

1.5 NOI相关活动的历史与规则

难度评级【1】

NOI（全国青少年信息学奥林匹克竞赛）由中国计算机学会（CCF）主办。竞赛体系包括：CSP-J/S（非专业级软件能力认证）→ NOIP（联赛）→ NOI（国赛）→ IOI（国际赛）。CSP-J为入门级认证，面向初中及以下学生。

1.6 位、字节与字

原始评分【1】→ AI重评【3】(位运算基础，竞赛常用)

计算机中数据的最小单位是**位（bit）**，只能存储0或1。8个位组成一个**字节（Byte）**。**字（Word）**的大小取决于CPU架构（32位或64位）。

单位	大小	常见应用
1 bit	0 或 1	bool类型的逻辑值
1 Byte = 8 bits	0~255（无符号）	char类型
4 Bytes = 32 bits	约±21亿	int类型
8 Bytes = 64 bits	约$\pm 9.2 \times 10^{18}$	long long类型

> **注意：**CSP-J近年真题中多次出现需要使用`long long`类型的题目。当数据范围超过2×10^9时，必须使用`long long`。

1.7 程序设计语言及编译运行基本概念

原始评分【1】→ AI重评【2】(理解编译流程重要)

程序设计语言分为**低级语言**（机器语言、汇编语言）和**高级语言**（C、C++、Python等）。NOI系列竞赛使用**C++**语言。C++程序需要经过**编译**（将源代码转换为机器代码）才能运行，编译器将`.cpp`源文件编译为可执行文件。

1.8 文件/目录的图形界面操作

难度评级【1】

掌握在Windows或Linux图形界面中进行文件和目录的基本操作：创建、复制、移动、删除、重命名文件和文件夹。

1.9 集成开发环境（IDE）使用

原始评分【1】→ AI重评【2】(开发环境熟悉度重要)

常用的C++ IDE包括：Windows下的**Dev-C++**（轻量级，适合入门）、**Code::Blocks**（跨平台）、**Visual Studio Code**（功能强大）。Linux下可使用Code::Blocks或命令行编辑器（如vim、nano）配合g++编译器。

1.10 编译命令g++的基本使用

原始评分【1】→ AI重评【3】(编译命令基础，实用性强)

g++是GNU C++编译器，是NOI竞赛环境中的标准编译器。基本使用方法：

```
# 基本编译
g++ -o program source.cpp

# 带优化的编译（竞赛常用）
g++ -O2 -o program source.cpp

# 编译并启用C++14标准
g++ -std=c++14 -O2 -o program source.cpp

# 运行程序
./program
```

第二章 C++程序设计

2.1 程序基本概念

2.1.1 标识符、关键字、常量、变量、表达式

原始评分【1】→ AI重评【3】(编程基础，必备知识)

标识符是程序中用来命名变量、函数、类型等的名称，由字母、数字和下划线组成，不能以数字开头。**关键字**是C++语言保留的具有特殊含义的标识符（如`int`、`if`、`for`、`return`等），不能用作变量名。

```
// 合法标识符
int count = 0;
double totalScore = 95.5;
int _value = 10;

// 非法标识符
// int 2nd = 5;     // 不能以数字开头
// int class = 1;   // class是关键字
```

2.1.2 常量与变量的命名、定义及作用

原始评分【1】→ AI重评【3】(基础语法，重要性提升)

常量是程序运行期间值不会改变的量，使用`const`关键字定义。**变量**是可以在程序运行过程中改变其值的量。

```
const int MAXN = 100005;  // 常量，竞赛中常用于定义数组大小
const double PI = 3.14159265358979;
int n, m;                 // 变量
```

2.1.3 头文件与名字空间

原始评分【2】→ AI重评【3】(代码组织关键)

头文件包含了函数声明和宏定义。竞赛中常用`#include <bits/stdc++.h>`万能头文件（包含所有标准库）。**名字空间**用于避免命名冲突，竞赛中常用`using namespace std;`。

```
#include <bits/stdc++.h>  // 万能头文件（竞赛专用）
using namespace std;       // 使用标准命名空间

int main() {
    // 程序代码
    return 0;
}
```

2.1.4 编辑、编译、解释、调试概念

原始评分【2】→ AI重评【3】(调试能力重要)

编辑是编写源代码的过程。**编译**是将源代码翻译成机器代码的过程（C++使用编译方式）。**解释**是逐行翻译并执行源代码（如Python）。**调试**是查找和修复程序错误的过程。

2.2 基本数据类型

原始评分【1】→ AI重评【3】(类型理解基础)

类型	大小	范围	用途
`int`	4字节	-2^{31} ~ $2^{31}-1$ (约±2.1×10^9)	常规整数
`long long`	8字节	-2^{63} ~ $2^{63}-1$ (约±9.2×10^{18})	大整数
`float`	4字节	约7位有效数字	单精度浮点（少用）
`double`	8字节	约15位有效数字	双精度浮点
`char`	1字节	-128 ~ 127 或 0 ~ 255	字符
`bool`	1字节	true(1) 或 false(0)	逻辑值

> **竞赛常见错误：**当题目数据范围超过2×10^9时忘记使用`long long`，导致整数溢出。建议养成习惯：看到大数据范围立即使用`long long`。

2.3 程序基本语句

2.3.1 输入输出语句

原始评分【2】→ AI重评【3】(IO基础，频繁使用)

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n;
    // C++风格输入输出
    cin >> n;
    cout << "n = " << n << endl;

    // C风格输入输出（通常更快）
    scanf("%d", &n);
    printf("n = %d\n", n);

    return 0;
}
```

> **性能提示：**在大数据量输入时，`scanf/printf` 比 `cin/cout` 更快。如果使用 `cin/cout`，可以添加 `ios::sync_with_stdio(false); cin.tie(0);` 来加速。

2.3.2 条件语句

原始评分【2】→ AI重评【3】(控制流基础)

```
// if-else语句
if (score >= 90) {
    cout << "优秀" << endl;
} else if (score >= 60) {
    cout << "及格" << endl;
} else {
    cout << "不及格" << endl;
}

// switch语句
switch (grade) {
    case 'A': cout << "优秀"; break;
    case 'B': cout << "良好"; break;
    default:  cout << "其他"; break;
}
```

2.3.3 循环语句

原始评分【2】→ AI重评【3】(循环结构核心)

```
// for循环 - 最常用
for (int i = 0; i < n; i++) {
    // 循环体
}

// while循环
while (条件) {
    // 循环体
}

// do-while循环（至少执行一次）
do {
    // 循环体
} while (条件);
```

2.3.4 多层循环语句

原始评分【3】→ AI重评【4】(复杂度提升，常见考点)

多层循环（嵌套循环）在竞赛中非常常见，用于处理二维数组、枚举多个变量等场景。需要注意**时间复杂度**：两层循环为$O(n^2)$，三层为$O(n^3)$。

```
// 打印九九乘法表
for (int i = 1; i <= 9; i++) {
    for (int j = 1; j <= i; j++) {
        printf("%d×%d=%-3d", j, i, i * j);
    }
    printf("\n");
}
```

2.4 基本运算

2.4.1 算术/关系/逻辑运算

原始评分【1】→ AI重评【3】(运算基础，频繁使用)

运算类型	运算符	示例	说明
算术运算	`+ - *`	`a + b`	加减乘
	`/`	`7 / 2 = 3`	整数除法取整
	`%`	`7 % 2 = 1`	取余（模运算）
	`++ --`	`i++`	自增自减
	`?:`	`a>b ? a : b`	三目运算
关系运算	`> >= < <= == !=`	`a == b`	返回bool值
逻辑运算	`&& || !`	`a>0 && b>0`	与、或、非

2.4.2 位运算

原始评分【2】→ AI重评【5】(高频考点，难度较大)

位运算直接操作二进制位，在竞赛中应用广泛（状态压缩、快速判断奇偶等）。

运算符	名称	示例	结果	常见用途
`&`	按位与	`5 & 3` (101 & 011)	1 (001)	判断奇偶：`n & 1`
`|`	按位或	`5 | 3` (101 | 011)	7 (111)	设置某一位
`^`	按位异或	`5 ^ 3` (101 ^ 011)	6 (110)	交换两数、加密
`~`	按位取反	`~5`	-6	补码运算
`<<`	左移	`1 << 3`	8	乘以2的幂
`>>`	右移	`8 >> 2`	2	除以2的幂

```
// 位运算常见技巧
int n = 10;
if (n & 1) cout << "奇数"; else cout << "偶数";  // 判断奇偶
int x = 1 << 10;  // x = 1024, 即2^10
// 交换两个数（不用临时变量）
a ^= b; b ^= a; a ^= b;
```

2.5 数学库常用函数

原始评分【3】→ AI重评【4】(数学思维增强)

函数	功能	示例
`abs(x)`	绝对值	`abs(-5) = 5`
`sqrt(x)`	平方根	`sqrt(16) = 4.0`
`ceil(x)`	上取整	`ceil(3.2) = 4`
`floor(x)`	下取整	`floor(3.8) = 3`
`round(x)`	四舍五入	`round(3.5) = 4`
`pow(x,y)`	x的y次方	`pow(2,10) = 1024`
`log(x)`	自然对数	`log(e) = 1.0`
`log2(x)`	以2为底的对数	`log2(8) = 3.0`

2.6 结构化程序设计

原始评分【1】→ AI重评【3】(基础结构，重要性提升)

原始评分【2】→ AI重评【3】(代码组织与设计)

程序的三种基本结构：**顺序结构**（按顺序执行）、**分支结构**（条件判断）、**循环结构**（重复执行）。模块化程序设计将复杂问题分解为若干子问题，每个子问题用一个函数实现。

2.7 数组

2.7.1 一维数组

原始评分【1】→ AI重评【4】(基础数据结构，应用广)

数组是存储相同类型元素的连续内存空间。数组下标从0开始。

```
int a[100005];  // 定义数组，竞赛中通常开大一些
int n;
cin >> n;
for (int i = 0; i < n; i++) {
    cin >> a[i];  // 读入数组
}
// 求数组元素之和
long long sum = 0;
for (int i = 0; i < n; i++) {
    sum += a[i];
}
```

2.7.2 二维数组与多维数组

原始评分【3】→ AI重评【4】(常见数据结构)

```
int grid[105][105];  // 二维数组，常用于矩阵、地图
int n, m;
cin >> n >> m;
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        cin >> grid[i][j];
```

2.8 字符串处理

原始评分【2】→ AI重评【4】(字符串处理基础)

C++中处理字符串有两种方式：**字符数组**（C风格）和**string类**（C++风格，推荐）。

```
// string类常用操作
string s = "hello";
int len = s.length();         // 长度: 5
s += " world";                // 拼接: "hello world"
string sub = s.substr(0, 5); // 子串: "hello"
int pos = s.find("world");    // 查找: 6
char c = s[0];                // 访问字符: 'h'

// 字符数组
char str[105];
scanf("%s", str);
int len2 = strlen(str);
```

2.9 函数与递归

2.9.1 函数定义与调用

原始评分【2】→ AI重评【5】(递归难点，重要考点)

```
// 函数定义
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);  // 递归调用
}

// 函数调用
int result = gcd(12, 8);  // result = 4
```

2.9.2 传值与传引用参数

原始评分【3】→ AI重评【4】(参数传递细节重要)

```
// 传值：函数内修改不影响原变量
void addOne(int x) { x++; }

// 传引用：函数内修改会影响原变量
void addOne(int &x) { x++; }

int a = 5;
addOne(a);  // 传引用后 a = 6
```

2.10 结构体与联合体

原始评分【3】→ AI重评【4】(数据组织关键)

```
// 结构体：将不同类型的数据组合在一起
struct Student {
    string name;
    int score;
    bool operator < (const Student &other) const {
        return score > other.score;  // 按分数降序排序
    }
};

Student stu[105];
sort(stu, stu + n);  // 使用自定义排序
```

2.11 指针与引用

原始评分【4】→ AI重评【5】(核心难点，区分度高)

指针存储变量的内存地址，引用是变量的别名。在竞赛中，指针主要用于链表、树等数据结构的实现。

```
int x = 10;
int *p = &x;   // p指向x的地址
cout << *p;     // 解引用，输出10
*p = 20;        // 通过指针修改x的值

int &ref = x;   // ref是x的引用（别名）
ref = 30;       // 等价于 x = 30
```

2.12 文件读写

原始评分【2】→ AI重评【3】(竞赛中偶尔应用)

```
// 文件重定向（竞赛常用）
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);

// C++文件流
ifstream fin("input.txt");
ofstream fout("output.txt");
int n;
fin >> n;
fout << n << endl;
```

2.13 STL模板

2.13.1 常用函数

原始评分【3】→ AI重评【5】(高频实用，效率提升)

```
#include <bits/stdc++.h>
using namespace std;

int a[] = {3, 1, 4, 1, 5, 9, 2, 6};
int n = 8;

sort(a, a + n);                    // 排序: 1 1 2 3 4 5 6 9
sort(a, a + n, greater<int>());    // 降序排序
int mx = *max_element(a, a + n);   // 最大值
int mn = *min_element(a, a + n);   // 最小值
swap(a[0], a[1]);                  // 交换两个元素
reverse(a, a + n);                 // 反转数组
```

2.13.2 STL容器

原始评分【4】→ AI重评【6】(竞赛必备，应用广泛)

容器	特点	常用操作	竞赛应用
`vector`	动态数组	push_back, size, []	邻接表、动态存储
`stack`	后进先出	push, pop, top	表达式求值、括号匹配
`queue`	先进先出	push, pop, front	BFS
`list`	双向链表	push_back, insert	频繁插入删除

```
// vector示例
vector<int> v;
v.push_back(10);
v.push_back(20);
for (int i = 0; i < v.size(); i++) cout << v[i] << " ";

// stack示例
stack<int> st;
st.push(1); st.push(2); st.push(3);
while (!st.empty()) {
    cout << st.top() << " ";  // 输出 3 2 1
    st.pop();
}

// queue示例（BFS常用）
queue<int> q;
q.push(1); q.push(2);
while (!q.empty()) {
    int front = q.front(); q.pop();
    cout << front << " ";  // 输出 1 2
}
```

第三章 数据结构

3.1 线性结构

3.1.1 链表

原始评分【3】→ AI重评【4】(数据结构基础)

链表是一种动态数据结构，每个节点包含数据域和指针域。与数组相比，链表支持O(1)的插入和删除，但不支持随机访问。

```
// 静态链表（竞赛常用，避免动态内存分配）
struct Node {
    int val, next;
} nodes[100005];
int head = -1, cnt = 0;

void insert(int val) {  // 头插法
    nodes[cnt] = {val, head};
    head = cnt++;
}
```

3.1.2 栈

原始评分【3】→ AI重评【5】(高频考点，应用广)

栈是一种**后进先出**（LIFO）的数据结构。在CSP-J中，栈常用于表达式求值、括号匹配、单调栈等场景。

```
// 括号匹配示例
bool isValid(string s) {
    stack<char> st;
    for (char c : s) {
        if (c == '(' || c == '[' || c == '{') st.push(c);
        else {
            if (st.empty()) return false;
            char top = st.top(); st.pop();
            if (c == ')' && top != '(') return false;
            if (c == ']' && top != '[') return false;
            if (c == '}' && top != '{') return false;
        }
    }
    return st.empty();
}
```

3.1.3 队列

原始评分【3】→ AI重评【5】(高频考点，应用广)

队列是一种**先进先出（FIFO）**的数据结构。在CSP-J中，队列是BFS（广度优先搜索）的核心数据结构。

3.2 简单树

3.2.1 树的定义与基本概念

原始评分【3】→ AI重评【5】(基础数据结构)

树是一种非线性数据结构，由节点和边组成。重要概念包括：**根节点**、**叶子节点**（无子节点）、**父节点**、**子节点**、**深度**（根到该节点的路径长度）、**高度**（该节点到最深叶子的路径长度）。

3.2.2 二叉树

原始评分【3】→ AI重评【5】(基础树结构)

二叉树是每个节点最多有两个子节点（左子节点和右子节点）的树。基本性质：

性质	描述
第i层最多节点数	2^{i-1}
深度为k的二叉树最多节点数	$2^k - 1$
叶子节点数 = 度为2的节点数 + 1	$n_0 = n_2 + 1$

3.2.3 树/二叉树的表示与存储

原始评分【4】→ AI重评【6】(存储实现关键)

```
// 二叉树的数组存储（适用于完全二叉树）
int tree[100005];  // tree[1]为根，tree[2i]为左子，tree[2i+1]为右子

// 二叉树的链式存储
struct TreeNode {
    int val;
    int left, right;  // 左右子节点编号
} nodes[100005];
```

3.2.4 二叉树的遍历

原始评分【4】→ AI重评【6】(遍历是基础技能)

```
// 前序遍历：根 → 左 → 右
void preorder(int u) {
    if (u == -1) return;
    cout << nodes[u].val << " ";
    preorder(nodes[u].left);
    preorder(nodes[u].right);
}

// 中序遍历：左 → 根 → 右
void inorder(int u) {
    if (u == -1) return;
    inorder(nodes[u].left);
    cout << nodes[u].val << " ";
    inorder(nodes[u].right);
}

// 后序遍历：左 → 右 → 根
void postorder(int u) {
    if (u == -1) return;
    postorder(nodes[u].left);
    postorder(nodes[u].right);
    cout << nodes[u].val << " ";
}
```

3.3 特殊树

3.3.1 完全二叉树

原始评分【4】→ AI重评【6】(重要树结构)

完全二叉树是除最后一层外每层都满的二叉树，最后一层的节点从左到右连续排列。可以用数组高效存储：节点i的左子节点为2i，右子节点为2i+1，父节点为i/2。

3.3.2 哈夫曼树与哈夫曼编码

原始评分【4】→ AI重评【5】(经典算法，应用有限)

哈夫曼树是带权路径长度最短的二叉树。构造方法：每次取权值最小的两个节点合并。哈夫曼编码是一种最优前缀编码，用于数据压缩。

3.3.3 二叉搜索树（BST）

原始评分【4】→ AI重评【5】(重要树结构)

二叉搜索树满足：左子树所有节点值 < 根节点值 < 右子树所有节点值。中序遍历BST可以得到有序序列。

3.4 简单图

3.4.1 图的定义与相关概念

原始评分【3】→ AI重评【5】(图论基础)

图由**顶点（Vertex）**和**边（Edge）**组成。图分为**有向图**和**无向图**。重要概念：度（与顶点相连的边数）、路径、环、连通性。

3.4.2 图的存储

原始评分【4】→ AI重评【6】(图论基础实现)

```
// 邻接矩阵（适用于稠密图）
int adj[505][505];  // adj[i][j] = 1 表示i到j有边
adj[u][v] = 1;
adj[v][u] = 1;  // 无向图

// 邻接表（适用于稀疏图，竞赛常用）
vector<int> G[100005];
G[u].push_back(v);
G[v].push_back(u);  // 无向图

// 带权邻接表
vector<pair<int,int>> G[100005];  // {目标节点, 权值}
G[u].push_back({v, w});
```

第四章 算法

4.1 算法概念与描述

原始评分【1】→ AI重评【3】(基础理论重要) **原始评分【2】→ AI重评【3】**(表达能力提升)

算法是解决特定问题的一系列明确步骤。算法的五个基本特性：**有穷性、确定性、可行性、输入、输出**。评价算法的主要指标是**时间复杂度**和**空间复杂度**。

时间复杂度	名称	$n=10^6$时操作次数	可行性
O(1)	常数	1	极快
O(log n)	对数	20	极快
O(n)	线性	10^6	快
O(n log n)	线性对数	2×10^7	可行
$O(n^2)$	平方	10^{12}	不可行
$O(2^n)$	指数	极大	不可行

4.2 入门算法

4.2.1 枚举法

原始评分【1】→ AI重评【3】(基础解题方法)

枚举法（暴力法）是最基础的算法思想：遍历所有可能的解，逐一检验是否满足条件。

```
// 示例：找出1~n中所有质数
for (int i = 2; i <= n; i++) {
    bool isPrime = true;
    for (int j = 2; j * j <= i; j++) {
        if (i % j == 0) { isPrime = false; break; }
    }
    if (isPrime) cout << i << " ";
}
```

4.2.2 模拟法

原始评分【1】→ AI重评【6】(高频考点，实用性强)

模拟法是按照题目描述的过程，用代码逐步模拟实现。这是CSP-J中**出现频率最高**的算法类型，几乎每年T1和T2都会考察。

> **高频考点：**根据CSP-J历年真题统计，模拟法在T1中出现率超过80%。关键是准确理解题意，注意边界条件和特殊情况。

4.3 基础算法

4.3.1 贪心法

原始评分【3】→ AI重评【6】(高频且区分度高)

贪心算法在每一步都选择当前最优的方案，期望最终得到全局最优解。贪心算法不一定能得到最优解，但对于某些特定问题（如活动选择、哈夫曼编码）可以证明其正确性。

```
// 经典贪心：活动选择问题
// 给定n个活动的开始和结束时间，选择最多不冲突的活动
struct Activity {
    int start, end;
};
bool cmp(Activity a, Activity b) {
    return a.end < b.end;  // 按结束时间排序
}
sort(act, act + n, cmp);
int count = 1, lastEnd = act[0].end;
for (int i = 1; i < n; i++) {
    if (act[i].start >= lastEnd) {
        count++;
        lastEnd = act[i].end;
    }
}
```

4.3.2 递推法

原始评分【3】→ AI重评【5】(动态规划基础)

递推法通过已知的初始值和递推关系，逐步计算后续结果。递推是动态规划的基础。

```
// 斐波那契数列
int fib[105];
fib[1] = 1; fib[2] = 1;
for (int i = 3; i <= n; i++)
    fib[i] = fib[i-1] + fib[i-2];
```

4.3.3 递归法

原始评分【4】→ AI重评【6】(重要思维方式)

递归是函数调用自身的编程技巧。递归需要满足两个条件：**基准情形**（终止条件）和**递归步骤**（问题规模缩小）。

```
// 汉诺塔问题
void hanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        cout << from << " -> " << to << endl;
        return;
    }
    hanoi(n - 1, from, aux, to);
    cout << from << " -> " << to << endl;
    hanoi(n - 1, aux, to, from);
}
```

4.3.4 二分法

原始评分【4】→ AI重评【7】(高频且效率关键)

二分法将搜索范围每次缩小一半，时间复杂度O(log n)。应用场景：有序数组查找、二分答案。

```
// 二分查找
int binarySearch(int a[], int n, int target) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (a[mid] == target) return mid;
        else if (a[mid] < target) left = mid + 1;
        else right = mid - 1;
    }
    return -1;  // 未找到
}

// 二分答案（竞赛高频模板）
int left = 0, right = 1e9;
while (left < right) {
    int mid = (left + right) / 2;
    if (check(mid)) right = mid;  // check函数判断mid是否可行
    else left = mid + 1;
}
// left就是答案
```

> **重要：**二分法是CSP-J中区分度最高的算法之一。近年T2和T4中多次出现二分答案的考法。

4.3.5 倍增法

原始评分【4】→ AI重评【7】(新增高效技巧)

倍增法是一种以2的幂次为步长进行跳跃的算法思想。常用于求解LCA（最近公共祖先）、稀疏表（ST表）等问题。核心思想：将任意整数分解为若干2的幂次之和。

4.4 算法策略

4.4.1 前缀和

原始评分【3】→ AI重评【5】(常用优化技巧)

前缀和是一种预处理技巧，可以在O(1)时间内求出数组任意区间的和。

```
// 一维前缀和
int a[100005], prefix[100005];
prefix[0] = 0;
for (int i = 1; i <= n; i++)
    prefix[i] = prefix[i-1] + a[i];

// 查询区间[l, r]的和
int sum = prefix[r] - prefix[l-1];

// 二维前缀和
int s[505][505];
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
        s[i][j] = s[i-1][j] + s[i][j-1] - s[i-1][j-1] + a[i][j];
```

4.4.2 差分

原始评分【4】→ AI重评【6】(新增且实用)

差分是前缀和的逆运算。差分数组可以在O(1)时间内对数组的一个区间进行加减操作。

```
// 差分数组
int diff[100005] = {0};

// 对区间[l, r]的所有元素加val
diff[l] += val;
diff[r + 1] -= val;

// 还原原数组
for (int i = 1; i <= n; i++)
    diff[i] += diff[i-1];  // diff[i]就是原数组a[i]的值
```

4.5 数值处理算法

4.5.1 高精度运算

原始评分【4】→ AI重评【5】(应用有限但难度较高)

当数值超过 `long long` 的范围时，需要使用高精度运算（用数组模拟大数运算）。

```
// 高精度加法
string add(string a, string b) {
    string result = "";
    int carry = 0;
    int i = a.size() - 1, j = b.size() - 1;
    while (i >= 0 || j >= 0 || carry) {
        int sum = carry;
        if (i >= 0) sum += a[i--] - '0';
        if (j >= 0) sum += b[j--] - '0';
        result = char(sum % 10 + '0') + result;
        carry = sum / 10;
    }
    return result;
}
```

4.6 排序算法

原始评分【3】→ AI重评【5】(基础且频繁)

算法	时间复杂度（平均）	时间复杂度（最坏）	空间复杂度	稳定性
冒泡排序	$O(n^2)$	$O(n^2)$	O(1)	稳定
选择排序	$O(n^2)$	$O(n^2)$	O(1)	不稳定
插入排序	$O(n^2)$	$O(n^2)$	O(1)	稳定
计数排序	O(n+k)	O(n+k)	O(k)	稳定
sort(STL)	O(n log n)	O(n log n)	O(log n)	不稳定

```
// 冒泡排序
for (int i = 0; i < n - 1; i++)
    for (int j = 0; j < n - 1 - i; j++)
        if (a[j] > a[j+1]) swap(a[j], a[j+1]);

// 选择排序
for (int i = 0; i < n - 1; i++) {
    int minIdx = i;
    for (int j = i + 1; j < n; j++)
        if (a[j] < a[minIdx]) minIdx = j;
    swap(a[i], a[minIdx]);
}

// 插入排序
for (int i = 1; i < n; i++) {
    int key = a[i], j = i - 1;
    while (j >= 0 && a[j] > key) {
        a[j+1] = a[j]; j--;
    }
    a[j+1] = key;
}

// 计数排序（适用于值域较小的情况）
int cnt[100005] = {0};
for (int i = 0; i < n; i++) cnt[a[i]]++;
int idx = 0;
for (int i = 0; i <= maxVal; i++)
    while (cnt[i]--) a[idx++] = i;
```

4.7 搜索算法

4.7.1 深度优先搜索（DFS）

原始评分【5】→ AI重评【8】(高频关键算法)

DFS沿着一条路径尽可能深入，直到无法继续时回溯。通常用**递归**或**栈**实现。

```
// DFS模板 - 全排列
int n, path[15];
bool used[15];

void dfs(int depth) {
    if (depth == n) {
        for (int i = 0; i < n; i++) cout << path[i] << " ";
        cout << endl;
        return;
    }
    for (int i = 1; i <= n; i++) {
        if (!used[i]) {
            used[i] = true;
            path[depth] = i;
            dfs(depth + 1);
            used[i] = false;  // 回溯
        }
    }
}
```

4.7.2 广度优先搜索（BFS）

原始评分【5】→ AI重评【8】(高频关键算法)

BFS逐层扩展搜索，使用**队列**实现。BFS可以找到**最短路径**（边权相同时）。

```
// BFS模板 - 迷宫最短路
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
int dist[505][505];
bool vis[505][505];

void bfs(int sx, int sy) {
    queue<pair<int,int>> q;
    q.push({sx, sy});
    vis[sx][sy] = true;
    dist[sx][sy] = 0;
    while (!q.empty()) {
        auto [x, y] = q.front(); q.pop();
        for (int d = 0; d < 4; d++) {
            int nx = x + dx[d], ny = y + dy[d];
            if (nx >= 0 && nx < n && ny >= 0 && ny < m
                && !vis[nx][ny] && grid[nx][ny] != '#') {
                vis[nx][ny] = true;
                dist[nx][ny] = dist[x][y] + 1;
                q.push({nx, ny});
            }
        }
    }
}
```

4.8 图论算法

4.8.1 图的遍历

原始评分【4】→ AI重评【7】(图论核心算法) **原始评分【4】→ AI重评【7】**(图论核心算法)

```
// 图的DFS遍历
bool vis[100005];
void dfs(int u) {
    vis[u] = true;
    for (int v : G[u]) {
        if (!vis[v]) dfs(v);
    }
}

// 图的BFS遍历
void bfs(int start) {
    queue<int> q;
    q.push(start);
    vis[start] = true;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int v : G[u]) {
            if (!vis[v]) {
                vis[v] = true;
                q.push(v);
            }
        }
    }
}
```

4.8.2 泛洪算法（Flood Fill）

原始评分【5】→ AI重评【7】(常见应用，区分度高)

泛洪算法用于填充连通区域，常用于统计连通块数量、图像填色等问题。

```
// Flood Fill - 统计连通块数量
int n, m, grid[505][505];
bool vis[505][505];
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};

void floodFill(int x, int y, int color) {
    vis[x][y] = true;
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d], ny = y + dy[d];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m
            && !vis[nx][ny] && grid[nx][ny] == color) {
            floodFill(nx, ny, color);
        }
    }
}

// 统计连通块数量
int count = 0;
for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        if (!vis[i][j]) {
            floodFill(i, j, grid[i][j]);
            count++;
        }
```

4.9 动态规划

4.9.1 动态规划的基本思路

原始评分【4】→ AI重评【9】(拉开差距关键)

动态规划（DP）是解决**最优化问题**的核心算法思想。DP的核心要素：

要素	说明
最优子结构	问题的最优解包含子问题的最优解
重叠子问题	不同的决策路径会产生相同的子问题
状态定义	用数组dp[i]表示到第i个阶段的最优解
状态转移方程	描述状态之间的递推关系
边界条件	初始状态的值

> **核心考点：**动态规划是CSP-J中**拉开差距的关键算法**。根据历年真题统计，T3和T4中DP出现频率最高。掌握DP是获得省一等奖的必要条件。

4.9.2 简单一维动态规划

原始评分【4】→ AI重评【9】(基础DP，频繁考)

```
// 经典问题：最长递增子序列（LIS）
// dp[i] = 以a[i]结尾的最长递增子序列长度
int dp[100005];
for (int i = 0; i < n; i++) {
    dp[i] = 1;
    for (int j = 0; j < i; j++) {
        if (a[j] < a[i])
            dp[i] = max(dp[i], dp[j] + 1);
    }
}
int ans = *max_element(dp, dp + n);

// 经典问题：爬楼梯
// dp[i] = 到达第i级台阶的方案数
dp[1] = 1; dp[2] = 2;
for (int i = 3; i <= n; i++)
    dp[i] = dp[i-1] + dp[i-2];
```

4.9.3 简单背包类型动态规划

原始评分【5】→ AI重评【9】(高频难点)

```
// 0-1背包问题
// n个物品，背包容量W，第i个物品重量w[i]，价值v[i]
// dp[j] = 容量为j时的最大价值
int dp[100005] = {0};
for (int i = 1; i <= n; i++)
    for (int j = W; j >= w[i]; j--)  // 逆序遍历!
        dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
cout << dp[W];

// 完全背包问题（每种物品可以选无限次）
for (int i = 1; i <= n; i++)
    for (int j = w[i]; j <= W; j++)  // 正序遍历!
        dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
```

4.9.4 简单区间类型动态规划

原始评分【5】→ AI重评【9】(高频难点)

```
// 区间DP模板：石子合并
// dp[i][j] = 合并第i堆到第j堆石子的最小代价
int dp[505][505], prefix[505];
memset(dp, 0x3f, sizeof(dp));
for (int i = 1; i <= n; i++) dp[i][i] = 0;

for (int len = 2; len <= n; len++) {       // 枚举区间长度
    for (int i = 1; i + len - 1 <= n; i++) { // 枚举左端点
        int j = i + len - 1;                  // 右端点
        for (int k = i; k < j; k++) {         // 枚举分割点
            dp[i][j] = min(dp[i][j],
                dp[i][k] + dp[k+1][j] + prefix[j] - prefix[i-1]);
        }
    }
}
```

第五章 数学与其他

5.1 数及其运算

原始评分【1】→ AI重评【2】(基础数学知识)

了解自然数、整数、有理数、实数的概念及其四则运算。在计算机中，整数和浮点数的表示方式不同，需要注意精度问题。

5.1.1 进制与进制转换

原始评分【1】→ AI重评【3】(竞赛常用基础)

```
// 十进制转其他进制
void toBase(int n, int base) {
    if (n == 0) return;
    toBase(n / base, base);
    cout << n % base;
}

// 其他进制转十进制
int toDecimal(string s, int base) {
    int result = 0;
    for (char c : s) {
        result = result * base + (c - '0');
    }
    return result;
}
```

5.2 初等数学

原始评分【1】→ AI重评【2】(基础数学) **原始评分【1】→ AI重评【2】**(基础数学)

掌握初中阶段的代数知识（一元一次方程、一元二次方程、不等式等）和几何知识（三角形、四边形、圆的面积和周长等）。CSP-J 2023年T3就考察了一元二次方程的求解。

5.3 初等数论

5.3.1 整除、因数、倍数、质数、合数

原始评分【3】→ AI重评【5】(数学基础，频繁考)

```
// 判断质数
bool isPrime(int n) {
    if (n < 2) return false;
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) return false;
    return true;
}

// 分解质因数
void factorize(int n) {
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            cout << i << " ";
            n /= i;
        }
    }
    if (n > 1) cout << n;
}
```

5.3.2 取整

原始评分【3】→ AI重评【4】(数学技巧)

上取整：`ceil(a/b) = (a + b - 1) / b`（整数除法技巧）。下取整：C++中整数除法默认下取整。

5.3.3 模运算与取余

原始评分【3】→ AI重评【6】(高频数学技巧)

模运算的基本性质：`(a + b) % m = ((a % m) + (b % m)) % m`，乘法同理。竞赛中常见"答案对10^9+7取模"。

```
const int MOD = 1e9 + 7;
long long ans = 0;
for (int i = 0; i < n; i++) {
    ans = (ans + a[i]) % MOD;
}
```

5.3.4 整数唯一分解定理

原始评分【3】→ AI重评【5】(数学基础)

任何大于1的正整数都可以唯一地分解为若干质数的乘积：$n = p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_k^{a_k}$。

5.3.5 辗转相除法（欧几里得算法）

原始评分【3】→ AI重评【6】(高频数学算法)

```
// 最大公约数（GCD）
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

// 最小公倍数（LCM）
int lcm(int a, int b) {
    return a / gcd(a, b) * b;  // 先除后乘防溢出
}

// C++14起可直接使用 __gcd(a, b)
```

5.3.6 素数筛法

原始评分【4】→ AI重评【7】(高频数学算法)

```
// 埃氏筛法 O(n log log n)
bool notPrime[10000005];
void sieve(int n) {
    notPrime[0] = notPrime[1] = true;
    for (int i = 2; i * i <= n; i++) {
        if (!notPrime[i]) {
            for (int j = i * i; j <= n; j += i)
                notPrime[j] = true;
        }
    }
}

// 线性筛（欧拉筛）O(n)
int primes[1000005], cnt = 0;
bool notPrime2[10000005];
void linearSieve(int n) {
    for (int i = 2; i <= n; i++) {
        if (!notPrime2[i]) primes[cnt++] = i;
        for (int j = 0; j < cnt && i * primes[j] <= n; j++) {
            notPrime2[i * primes[j]] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

5.4 离散与组合数学

5.4.1 集合与计数原理

原始评分【2】→ AI重评【3】(基础数学概念) 原始评分【2】→ AI重评【3】(组合数学基础)

原始评分【2】→ AI重评【3】(组合数学基础)

加法原理：完成一件事有n类方法，第i类有a_i种，总方法数为 Σa_i。**乘法原理**：完成一件事需要n个步骤，第i步有a_i种方法，总方法数为 Πa_i。

5.4.2 排列与组合

原始评分【4】→ AI重评【5】(组合数学重要) 原始评分【4】→ AI重评【5】(组合数学重要)

排列：从n个元素中取r个排列，P(n,r) = n!/(n-r)!。**组合**：从n个元素中取r个组合，C(n,r) = n!/(r!(n-r)!)。

```
// 计算组合数 C(n, r)
long long C[1005][1005];
void initC(int maxn) {
    for (int i = 0; i <= maxn; i++) {
        C[i][0] = 1;
        for (int j = 1; j <= i; j++)
            C[i][j] = (C[i-1][j-1] + C[i-1][j]) % MOD;
    }
}
```

5.4.3 杨辉三角

难度评级【4】

杨辉三角的第n行第k个数就是组合数C(n,k)。杨辉三角的递推关系：C(n,k) = C(n-1,k-1) + C(n-1,k)。

5.4.4 ASCII码

原始评分【2】→ AI重评【3】(字符串处理基础)

字符	ASCII码	常用关系
'0'~'9'	48~57	数字字符转数字：`c - '0'`
'A'~'Z'	65~90	大写转小写：`c + 32` 或 `c + 'a' - 'A'`
'a'~'z'	97~122	小写转大写：`c - 32` 或 `c - 'a' + 'A'`

附录：评分变化汇总表

以下表格列出了所有经过AI模型重新评估后评分发生变化的知识点（黄色高亮标记）。评分基于全网深度调研数据，综合考虑考试重要性、学习难度、区分度和实用性四个维度。

编号	知识点	原始评分	AI重评分	变化	评分理由
1	计算机的基本构成	【1】	【2】	+1	基础重要，竞赛环境必备
2	操作系统基本概念及常见操作	【1】	【2】	+1	环境操作基础，实用性提升
3	计算机网络和Internet基本概念	【1】	【1】	-	竞赛中应用较少
4	计算机的历史和常见用途	【1】	【1】	-	理论性强，实用性低
5	NOI相关活动的历史与规则	【1】	【1】	-	非技术核心内容
6	位、字节与字	【1】	【3】	+2	位运算基础，竞赛常用
7	程序设计语言及编译运行基本概念	【1】	【2】	+1	理解编译流程重要
8	文件/目录的图形界面操作	【1】	【1】	-	竞赛中应用少
9	Windows/Linux集成开发环境使用	【1】	【2】	+1	开发环境熟悉度重要
10	编译命令g++的基本使用	【1】	【3】	+2	编译命令基础，实用性强
11	标识符、关键字、常量、变量、表达式	【1】	【3】	+2	编程基础，必备知识
12	常量与变量的命名、定义及作用	【1】	【3】	+2	基础语法，重要性提升
13	头文件与名字空间	【2】	【3】	+1	代码组织关键
14	编辑、编译、解释、调试概念	【2】	【3】	+1	调试能力重要
15	基本数据类型(int,long long,float,double,char,bool)	【1】	【3】	+2	类型理解基础

16	输入输出语句(cin/cout/scanf/printf)	【2】	【3】	+1	IO基础，频繁使用
17	条件语句(if/switch)	【2】	【3】	+1	控制流基础
18	循环语句(for/while/do while)	【2】	【3】	+1	循环结构核心
19	多层循环语句	【3】	【4】	+1	复杂度提升，常见考点
20	算术/关系/逻辑运算	【1】	【3】	+2	运算基础，频繁使用
21	位运算	【2】	【5】	+3	高频考点，难度较大
22	数学库常用函数	【3】	【4】	+1	数学思维增强
23	顺序/分支/循环结构	【1】	【3】	+2	基础结构，重要性提升
24	模块化程序设计与流程图	【2】	【3】	+1	代码组织与设计
25	数组与数组下标	【1】	【4】	+3	基础数据结构，应用广
26	二维数组与多维数组	【3】	【4】	+1	常见数据结构
27	字符数组与string类	【2】	【4】	+2	字符串处理基础
28	函数定义与调用、递归函数	【2】	【5】	+3	递归难点，重要考点
29	传值与传引用参数	【3】	【4】	+1	参数传递细节重要
30	结构体与联合体	【3】	【4】	+1	数据组织关键
31	指针与引用	【4】	【5】	+1	核心难点，区分度高
32	文件读写	【2】	【3】	+1	竞赛中偶尔应用
33	STL常用函数(min/max/swap/sort)	【3】	【5】	+2	

					高频实用，效率提升
34	STL容器(stack/queue/list/vector)	【4】	【6】	+2	竞赛必备，应用广泛
35	链表(单/双向/循环)	【3】	【4】	+1	数据结构基础
36	栈	【3】	【5】	+2	高频考点，应用广
37	队列	【3】	【5】	+2	高频考点，应用广
38	树的定义与基本概念	【3】	【5】	+2	基础数据结构
39	二叉树定义与基本性质	【3】	【5】	+2	基础树结构
40	树/二叉树的表示与存储	【4】	【6】	+2	存储实现关键
41	二叉树的遍历(前序/中序/后序)	【4】	【6】	+2	遍历是基础技能
42	完全二叉树定义、性质与数组表示	【4】	【6】	+2	重要树结构
43	哈夫曼树与哈夫曼编码	【4】	【5】	+1	经典算法，应用有限
44	二叉搜索树	【4】	【5】	+1	重要树结构
45	图的定义与相关概念	【3】	【5】	+2	图论基础
46	图的存储(邻接矩阵/邻接表)	【4】	【6】	+2	图论基础实现
47	算法概念	【1】	【3】	+2	基础理论重要
48	算法描述(自然语言/流程图/伪代码)	【2】	【3】	+1	表达能力提升
49	枚举法	【1】	【3】	+2	基础解题方法
50	模拟法	【1】	【6】	+5	高频考点，实用性强
51	贪心法	【3】	【6】	+3	高频且区分度高
52	递推法	【3】	【5】	+2	动态规划基础
53	递归法	【4】	【6】	+2	重要思维方式

54	二分法	【4】	【7】	+3	高频且效率关键
55	倍增法	【4】	【7】	+3	新增高效技巧
56	前缀和	【3】	【5】	+2	常用优化技巧
57	差分	【4】	【6】	+2	新增且实用
58	高精度运算	【4】	【5】	+1	应用有限但难度较高
59	排序基本概念	【3】	【5】	+2	基础且频繁
60	冒泡排序	【3】	【3】	-	基础教学用
61	选择排序	【3】	【3】	-	基础教学用
62	插入排序	【3】	【3】	-	基础教学用
63	计数排序	【3】	【4】	+1	常用优化排序
64	深度优先搜索(DFS)	【5】	【8】	+3	高频关键算法
65	广度优先搜索(BFS)	【5】	【8】	+3	高频关键算法
66	图的深度优先遍历	【4】	【7】	+3	图论核心算法
67	图的广度优先遍历	【4】	【7】	+3	图论核心算法
68	泛洪算法(Flood Fill)	【5】	【7】	+2	常见应用，区分度高
69	动态规划基本思路	【4】	【9】	+5	拉开差距关键
70	简单一维动态规划	【4】	【9】	+5	基础DP，频繁考
71	简单背包类型动态规划	【5】	【9】	+4	高频难点
72	简单区间类型动态规划	【5】	【9】	+4	高频难点
73	自然数/整数/有理数/实数及运算	【1】	【2】	+1	基础数学知识
74	进制与进制转换	【1】	【3】	+2	竞赛常用基础
75	代数(初中部分)	【1】	【2】	+1	基础数学
76	几何(初中部分)	【1】	【2】	+1	基础数学

77	整除/因数/倍数/质数/合数	【3】	【5】	+2	数学基础，频繁考
78	取整	【3】	【4】	+1	数学技巧
79	模运算与取余	【3】	【6】	+3	高频数学技巧
80	整数唯一分解定理	【3】	【5】	+2	数学基础
81	辗转相除法(欧几里得算法)	【3】	【6】	+3	高频数学算法
82	素数筛法(埃氏筛/线性筛)	【4】	【7】	+3	高频数学算法
83	集合	【2】	【3】	+1	基础数学概念
84	加法原理	【2】	【3】	+1	组合数学基础
85	乘法原理	【2】	【3】	+1	组合数学基础
86	排列	【4】	【5】	+1	组合数学重要
87	组合	【4】	【5】	+1	组合数学重要
88	杨辉三角	【4】	【4】	-	基础数学工具
89	ASCII码	【2】	【3】	+1	字符串处理基础

数据来源与参考：

[1] NOI 大纲（2025年修订版）- https://www.noi.cn/xw/2025-04-18/841584.shtml

[2] CSP-J 历年复赛真题考察内容 (2010～2023) 考点分析

[3] CSP-J/S2025 入门级题目知识构成分析报告

[4] AI模型评分（gpt-4.1-mini），评分维度：考试重要性、学习难度、区分度、实用性