CCF CSP Handbook

# CCF CSP 考场手册

题型决策树、C++ 模板与真题索引体系

## 面向考场检索的开源备考资料

本手册围绕“题目识别、数据范围判断、算法模板选择、易错点检查、真题反查”组织内容，目标是在复习和临场翻阅时都能快速定位可执行方案。

**判断** 从题面信号和数据范围定位题型。

**模板** 汇总 C++、STL、算法和大模拟骨架。

**检查** 覆盖下标、溢出、取整、格式和部分分策略。

**校准** 用历年真题反查判断树和模板入口。

开源备考资料 · 可打印 · 可持续修订

GitHub 发布版

# 使用前说明

这本手册的目标不是覆盖所有算法竞赛知识，而是在 CCF CSP 考场中帮助你快速完成四件事：

1. 根据题面关键词和数据范围判断题型。
2. 找到可用算法、C++ 语法、STL 容器和代码模板。
3. 避免输出格式、下标、溢出、初始化等低级错误。
4. 遇到难题时先写暴力或特殊情况，尽量拿部分分。

**重要提醒：**正式打印和考试前，必须重新核对 CCF CSP 官网的最新规则、语言标准、考场环境和允许携带材料说明。

# 目录

**使用前说明** i

**第一部分　导航与判断** 1

**第一章　全书总览与使用路径** 2
1.1 本手册的工作方式 . . . 2
1.2 全书结构路线图 . . . 2
1.3 官方规则速查 . . . 3
1.4 考场总流程 . . . 3
1.4.1 开场 10 分钟 . . . 3
1.4.2 每题固定流程 . . . 4
1.4.3 建议时间分配 . . . 4
1.5 数据范围与复杂度 . . . 5
1.5.1 复杂度速查表 . . . 5
1.5.2 复杂度校验规则 . . . 5
1.6 A/B/C/D/E 总体策略 . . . 5
1.6.1 A 题 . . . 5
1.6.2 B 题 . . . 6
1.6.3 C 题 . . . 6
1.6.4 D/E 题 . . . 6
1.7 总决策树 . . . 6
1.7.1 第一层：看题面信号 . . . 6
1.7.2 第二层：用数据范围排除错误方向 . . . 7
1.7.3 第三层：选择满分或部分分 . . . 8
1.8 关键词到章节索引 . . . 8
1.9 低级错误总检查 . . . 8
1.10 部分分总策略 . . . 9
1.11 本章完成后的后续连接 . . . 9

**第二章　考场 1 分钟极速索引** 10
2.1 本章定位 . . . 10
2.2 1 分钟总流程 . . . 10
2.3 题号策略速查 . . . 10
2.4 题面主体到模板 . . . 11

2.5 数据范围秒判 . . . 11
2.6 B 题优化入口 . . . 11
2.7 C 题建模速查 . . . 12
2.8 D/E 部分分入口 . . . 12
2.9 模板编号速查 . . . 12
2.10 C++ 语法救急 . . . 13
2.11 最后提交前 30 秒 . . . 13

**第三章 题型详细判断树 14**
3.1 本章定位 . . . 14
3.2 考场三层分流法 . . . 14
3.3 总判断流程 . . . 14
3.4 决策树叶子节点标准格式 . . . 15
3.5 从根节点开始的总决策树 . . . 15
3.6 详细叶子索引总表 . . . 16
3.7 题面结构第一判断 . . . 16
3.8 数据范围判断树 . . . 17
3.8.1 单变量规模 . . . 17
3.8.2 多变量规模 . . . 17
3.9 关键词到模板总索引 . . . 18
3.10 操作模式判断树 . . . 19
3.10.1 只有查询，没有修改 . . . 19
3.10.2 有修改，有查询 . . . 19
3.10.3 多轮模拟 . . . 19
3.11 A 题快速判断树 . . . 20
3.11.1 A 题详细叶子表 . . . 20
3.12 B 题快速判断树 . . . 21
3.12.1 B 题详细叶子表 . . . 21
3.13 C 题快速判断树 . . . 22
3.13.1 C 题读题分解 . . . 22
3.13.2 C 题模板选择 . . . 22
3.13.3 C 题详细叶子表 . . . 23
3.14 逐句读题分支树 . . . 23
3.14.1 从句子到模板 . . . 24
3.15 B 题专项分支树 . . . 24
3.15.1 B 题从暴力到优化 . . . 24
3.15.2 B 题二分答案确认表 . . . 25
3.16 C 题专项分支树 . . . 25
3.16.1 C 题对象建模表 . . . 25
3.16.2 C 题规则优先级表 . . . 25
3.16.3 C 题字符串解析分支 . . . 26
3.17 D/E 题快速判断树 . . . 26

3.18 D/E 部分分决策树 . . . 26
3.18.1 D/E 题详细叶子表 . . . 27
3.19 容易误判的分支 . . . 27
3.20 无法判断时的保底流程 . . . 28
3.21 判断树反向测试表 . . . 28
3.22 最后 5 分钟判断 . . . 28

**第二部分 基础与题型 30**

**第四章 C++ 语法与 STL 速查 31**
4.1 本章使用方式 . . . 31
4.2 最小程序骨架 . . . 31
4.3 输入输出 . . . 32
4.3.1 普通输入 . . . 32
4.3.2 读入数组 . . . 32
4.3.3 多组数据 . . . 32
4.3.4 整行输入 . . . 33
4.3.5 输出 . . . 33
4.3.6 小数输出 . . . 33
4.4 数据类型 . . . 33
4.4.1 整数类型 . . . 33
4.4.2 浮点类型 . . . 34
4.4.3 字符和布尔 . . . 34
4.5 表达式、整除和取整 . . . 35
4.5.1 基本运算符 . . . 35
4.5.2 整数除法和类型转换 . . . 35
4.5.3 正整数向上取整 . . . 35
4.5.4 有符号整数向下和向上取整 . . . 36
4.5.5 负数取模 . . . 36
4.5.6 运算优先级和括号 . . . 36
4.5.7 常量和初始化 . . . 37
4.6 控制流 . . . 37
4.6.1 if . . . 37
4.6.2 for . . . 37
4.6.3 while . . . 38
4.6.4 break 和 continue . . . 38
4.7 auto、引用和范围 for . . . 38
4.7.1 auto . . . 38
4.7.2 引用 . . . 39
4.7.3 范围 for . . . 39
4.7.4 size 的类型 . . . 39
4.8 函数 . . . 40

4.9 数组 . . . 40
4.9.1 普通数组 . . . 40
4.9.2 二维数组 . . . 41
4.9.3 初始化 . . . 41
4.10 vector . . . 41
4.10.1 创建 . . . 41
4.10.2 常用操作 . . . 41
4.10.3 遍历 . . . 42
4.10.4 删除元素 . . . 42
4.10.5 二维 vector . . . 42
4.11 string . . . 43
4.11.1 基本操作 . . . 43
4.11.2 子串 . . . 43
4.11.3 查找 . . . 43
4.11.4 字符串转数字 . . . 44
4.11.5 按空格分割 . . . 44
4.12 pair 和 tuple . . . 44
4.12.1 pair . . . 44
4.12.2 tuple . . . 45
4.13 结构体 struct . . . 45
4.14 排序 sort . . . 45
4.14.1 普通排序 . . . 45
4.14.2 自定义排序：从大到小 . . . 46
4.14.3 结构体排序 . . . 46
4.14.4 按 vector 的某一列排序 . . . 46
4.14.5 stable_sort . . . 46
4.14.6 比较器高危错误 . . . 47
4.15 二分相关函数 . . . 47
4.15.1 binary_search . . . 47
4.15.2 lower_bound 和 upper_bound . . . 47
4.16 map 和 unordered_map . . . 47
4.16.1 map . . . 47
4.16.2 unordered_map . . . 48
4.17 unordered_set . . . 48
4.18 set 和 multiset . . . 49
4.18.1 set . . . 49
4.18.2 multiset . . . 49
4.19 queue、stack、deque . . . 50
4.19.1 queue . . . 50
4.19.2 stack . . . 50
4.19.3 deque . . . 50
4.20 priority_queue . . . 50

4.20.1 大根堆 . . . 50
4.20.2 小根堆 . . . 51
4.20.3 pair 小根堆 . . . 51
4.21 常用算法函数 . . . 51
4.21.1 min、max、swap . . . 51
4.21.2 reverse . . . 51
4.21.3 unique 去重 . . . 51
4.21.4 accumulate . . . 51
4.21.5 gcd . . . 51
4.21.6 常用数学函数 . . . 52
4.22 lambda 入门 . . . 52
4.23 常见编译错误和原因 . . . 53
4.24 C 和 C++ 混用提醒 . . . 53
4.25 考场 C++ 快速检查 . . . 54

**第五章 A 题决策树与高频模板 55**
5.1 A 题定位 . . . 55
5.2 A 题开题流程 . . . 55
5.3 A 题读题标注法 . . . 55
5.3.1 读题时必须圈出的信息 . . . 55
5.3.2 草稿纸模板 . . . 56
5.3.3 样例手算方法 . . . 56
5.4 A 题总决策树 . . . 56
5.5 A 题关键词到动作 . . . 57
5.6 A 题数据范围判断 . . . 58
5.7 A 题常见输入形态 . . . 58
5.7.1 输入一个数 . . . 58
5.7.2 输入两个或多个数 . . . 58
5.7.3 输入 n 个数 . . . 58
5.7.4 输入 n 行记录 . . . 59
5.7.5 输入 n 行字符串 . . . 59
5.7.6 读到文件结束 . . . 59
5.8 简单统计 . . . 59
5.8.1 识别信号 . . . 59
5.8.2 一遍扫描模板 . . . 60
5.8.3 注意事项 . . . 60
5.8.4 统计满足条件的个数 . . . 60
5.8.5 同时统计多个类别 . . . 60
5.8.6 找最大值的位置 . . . 61
5.8.7 计数数组 . . . 61
5.9 存在、全部、任意判断 . . . 62
5.9.1 判断是否存在 . . . 62

5.9.2 判断是否全部满足 . . . 62
5.9.3 提前结束 . . . 62
5.10 公式题 . . . 63
5.10.1 识别信号 . . . 63
5.10.2 公式题流程 . . . 63
5.10.3 整数除法和取整 . . . 63
5.10.4 分段计算模板 . . . 64
5.10.5 分类讨论模板 . . . 64
5.10.6 绝对值和距离 . . . 65
5.10.7 取模 . . . 65
5.10.8 小数和百分比 . . . 65
5.11 排序和排名 . . . 66
5.11.1 识别信号 . . . 66
5.11.2 单个数组排序 . . . 66
5.11.3 结构体排序 . . . 67
5.11.4 排名常见规则 . . . 67
5.11.5 按字符串字典序排序 . . . 68
5.11.6 先去重再排序 . . . 68
5.11.7 找第 k 小和第 k 大 . . . 68
5.12 字符串处理 . . . 68
5.12.1 识别信号 . . . 68
5.12.2 逐字符扫描 . . . 68
5.12.3 大小写转换 . . . 69
5.12.4 查找子串 . . . 69
5.12.5 按空格分割一整行 . . . 69
5.12.6 按指定字符分割 . . . 70
5.12.7 判断前缀和后缀 . . . 70
5.12.8 统计每个小写字母 . . . 70
5.12.9 反转字符串 . . . 71
5.12.10 回文判断 . . . 71
5.13 进制处理 . . . 71
5.13.1 识别信号 . . . 71
5.13.2 任意进制转十进制 . . . 71
5.13.3 十进制转任意进制 . . . 72
5.13.4 位运算入门 . . . 72
5.13.5 进制题检查 . . . 72
5.14 日期时间 . . . 73
5.14.1 识别信号 . . . 73
5.14.2 闰年 . . . 73
5.14.3 每月天数 . . . 73
5.14.4 下一天 . . . 73
5.14.5 时间转秒 . . . 73

5.15 简单模拟 . . . 74
5.15.1 识别信号 . . . 74
5.15.2 方向数组 . . . 74
5.15.3 按命令移动 . . . 74
5.15.4 模拟题写法建议 . . . 75
5.16 坐标和距离入门 . . . 75
5.16.1 二维点 . . . 75
5.16.2 曼哈顿距离 . . . 75
5.16.3 欧几里得距离 . . . 75
5.16.4 坐标题检查 . . . 75
5.17 二维表格和矩阵入门 . . . 76
5.17.1 读入 n 行 m 列 . . . 76
5.17.2 每行求和 . . . 76
5.17.3 每列求和 . . . 76
5.17.4 找矩阵最大值 . . . 76
5.18 频率统计 . . . 77
5.18.1 值域小：用数组 . . . 77
5.18.2 值域大或 key 是字符串：用 map . . . 77
5.18.3 只需要快速统计：unordered_map . . . 77
5.19 区间入门 . . . 77
5.19.1 数据小：直接循环 . . . 78
5.19.2 多次区间和：前缀和 . . . 78
5.20 A 题输出格式专项 . . . 78
5.21 A 题新手错误对照表 . . . 79
5.22 A 题推荐代码组织 . . . 79
5.23 A 题自造边界样例 . . . 80
5.24 A 题本地调试流程 . . . 80
5.24.1 第一步：确认读入 . . . 81
5.24.2 第二步：确认中间结果 . . . 81
5.24.3 第三步：用最小样例 . . . 81
5.24.4 第四步：检查边界 . . . 81
5.25 A 题常见边界库 . . . 81
5.26 A 题手写模板清单 . . . 82
5.27 A 题临场优先级 . . . 82
5.28 A 题提交前总检查 . . . 83
5.29 A 题卡住时怎么办 . . . 83

**第六章 B 题决策树与高频算法入口 84**
6.1 B 题定位 . . . 84
6.2 B 题开题流程 . . . 84
6.3 B 题总决策树 . . . 84
6.4 B 题数据范围判断 . . . 85

6.5 B 题关键词到动作 . . . 85
6.6 从暴力到优化的转换表 . . . 86
6.7 前缀和 . . . 86
6.7.1 识别信号 . . . 86
6.7.2 一维前缀和模板 . . . 87
6.7.3 0 下标前缀和 . . . 87
6.7.4 二维前缀和 . . . 87
6.7.5 前缀和易错点 . . . 88
6.8 前后缀预处理 . . . 88
6.8.1 识别信号 . . . 88
6.8.2 前缀最大和后缀最大 . . . 88
6.8.3 前后缀易错点 . . . 89
6.9 差分 . . . 89
6.9.1 识别信号 . . . 89
6.9.2 一维差分模板 . . . 89
6.9.3 差分易错点 . . . 90
6.9.4 二维差分入口 . . . 90
6.10 哈希统计 . . . 91
6.10.1 识别信号 . . . 91
6.10.2 值域小：计数数组 . . . 91
6.10.3 值域大：map . . . 91
6.10.4 字符串统计 . . . 91
6.10.5 找出现次数最多 . . . 92
6.11 余数和同余统计 . . . 92
6.11.1 识别信号 . . . 92
6.11.2 统计每个余数 . . . 92
6.11.3 两数和能被 k 整除 . . . 93
6.11.4 前缀和模 k . . . 93
6.12 排序加扫描 . . . 93
6.12.1 识别信号 . . . 93
6.12.2 排序后统计相同元素段 . . . 94
6.12.3 合并区间 . . . 94
6.12.4 事件排序 . . . 94
6.12.5 坐标离散化入口 . . . 95
6.13 堆和动态维护 . . . 95
6.13.1 priority_queue 识别信号 . . . 95
6.13.2 小根堆 . . . 96
6.13.3 保留最大的 k 个数 . . . 96
6.13.4 set 动态维护 . . . 96
6.13.5 map 动态维护 . . . 97
6.14 基础贪心 . . . 97
6.14.1 识别信号 . . . 97

6.14.2 最多不重叠区间 . . . 97
6.14.3 贪心易错点 . . . 98
6.15 单调栈和单调队列入口 . . . 98
6.15.1 单调栈识别信号 . . . 98
6.15.2 右边第一个更大元素 . . . 98
6.15.3 单调队列识别信号 . . . 98
6.16 双指针入门 . . . 99
6.16.1 识别信号 . . . 99
6.16.2 排序后找两数和 . . . 99
6.16.3 正数数组的连续区间 . . . 99
6.17 二分与二分答案 . . . 100
6.17.1 普通二分查找 . . . 100
6.17.2 二分答案识别 . . . 100
6.17.3 二分答案模板 . . . 100
6.18 BFS 入门 . . . 101
6.18.1 识别信号 . . . 101
6.18.2 网格 BFS 模板 . . . 101
6.18.3 BFS 易错点 . . . 102
6.18.4 多源 BFS . . . 102
6.19 DFS 和连通块 . . . 103
6.19.1 识别信号 . . . 103
6.19.2 网格 DFS . . . 103
6.20 Floyd 入门 . . . 104
6.20.1 识别信号 . . . 104
6.20.2 Floyd 模板 . . . 104
6.20.3 Floyd 易错点 . . . 105
6.21 Dijkstra 入门 . . . 105
6.21.1 识别信号 . . . 105
6.21.2 Dijkstra 模板 . . . 105
6.21.3 Dijkstra 易错点 . . . 106
6.22 并查集入门 . . . 106
6.22.1 识别信号 . . . 106
6.22.2 并查集模板 . . . 107
6.23 拓扑排序入口 . . . 107
6.23.1 识别信号 . . . 107
6.23.2 拓扑排序模板 . . . 107
6.23.3 拓扑排序易错点 . . . 108
6.24 简单 DP 入口 . . . 108
6.24.1 识别信号 . . . 108
6.24.2 线性 DP 示例 . . . 109
6.24.3 01 背包入口 . . . 109
6.24.4 完全背包入口 . . . 109

6.24.5 DP 易错点 . . . 110
6.25 基础数学入口 . . . 110
6.25.1 gcd 和 lcm . . . 110
6.25.2 快速幂 . . . 110
6.25.3 判断素数 . . . 110
6.25.4 枚举约数 . . . 111
6.26 B 题提交前检查 . . . 111
6.27 B 题冲分判断表 . . . 112
6.28 B 题常见边界库 . . . 112
6.29 B 题本地调试流程 . . . 113
6.29.1 先用暴力对拍思路 . . . 113
6.29.2 复杂度复查 . . . 113
6.29.3 状态清空 . . . 113
6.30 B 题卡住时的部分分策略 . . . 114
6.31 B 题手写模板清单 . . . 114

**第七章 C 题决策树与大模拟方法 115**
7.1 C 题定位 . . . 115
7.2 C 题开题流程 . . . 115
7.3 C 题读题三遍法 . . . 115
7.3.1 第一遍：只判断题型 . . . 115
7.3.2 第二遍：圈对象、状态、操作 . . . 116
7.3.3 第三遍：找坑点 . . . 116
7.4 从题面到代码的转换 . . . 116
7.4.1 对象变成 struct . . . 116
7.4.2 编号查找用 map 或 vector . . . 117
7.4.3 命令变成函数 . . . 117
7.4.4 规则变成 if . . . 117
7.5 状态维护和容器选择 . . . 117
7.5.1 容器选择表 . . . 117
7.5.2 操作复杂度估算 . . . 118
7.5.3 C 题容器选择错误信号 . . . 118
7.6 C 题总决策树 . . . 118
7.7 C 题考法地图 . . . 119
7.8 C 题细分决策树 . . . 119
7.8.1 第一问：这是大模拟还是算法题 . . . 119
7.8.2 大模拟细分树 . . . 120
7.8.3 字符串解析细分树 . . . 120
7.8.4 图网格细分树 . . . 120
7.9 大模拟总方法 . . . 121
7.9.1 大模拟识别信号 . . . 121
7.9.2 大模拟代码骨架 . . . 121

7.9.3 为什么要拆函数 . . . 122
7.10 长题面拆解表 . . . 122
7.11 命令解析 . . . 123
7.11.1 固定格式命令 . . . 123
7.11.2 整行命令 . . . 123
7.11.3 手写 split . . . 123
7.11.4 命令分发模板 . . . 124
7.11.5 命令解析易错点 . . . 124
7.12 状态设计 . . . 125
7.12.1 用结构体保存对象 . . . 125
7.12.2 状态编号 . . . 125
7.12.3 状态转移表 . . . 125
7.13 状态机模拟 . . . 125
7.13.1 识别信号 . . . 125
7.13.2 状态机模板 . . . 126
7.13.3 状态机检查 . . . 126
7.14 事件模拟 . . . 127
7.14.1 识别信号 . . . 127
7.14.2 事件结构体 . . . 127
7.14.3 事件排序 . . . 127
7.14.4 事件处理模板 . . . 127
7.14.5 同一时间事件顺序 . . . 127
7.15 优先级模拟 . . . 128
7.15.1 识别信号 . . . 128
7.15.2 priority_queue 模板 . . . 128
7.16 复杂排序和排名 . . . 129
7.16.1 识别信号 . . . 129
7.16.2 复杂排序模板 . . . 129
7.16.3 并列排名 . . . 129
7.16.4 复杂排序检查 . . . 129
7.17 表格和矩阵模拟 . . . 130
7.17.1 识别信号 . . . 130
7.17.2 矩阵读入和输出 . . . 130
7.17.3 交换两行 . . . 130
7.17.4 交换两列 . . . 130
7.17.5 顺时针旋转矩阵 . . . 130
7.17.6 表格模拟易错点 . . . 131
7.18 目录树和路径模拟 . . . 131
7.18.1 识别信号 . . . 131
7.18.2 路径规范化 . . . 131
7.18.3 路径转字符串 . . . 132
7.18.4 目录树节点 . . . 132

7.18.5 路径题易错点 . . . 132
7.19 URL 路由和参数解析 . . . 132
7.19.1 识别信号 . . . 132
7.19.2 拆分 URL . . . 133
7.19.3 简单路由匹配 . . . 133
7.19.4 URL 题易错点 . . . 134
7.20 配置和 JSON 类解析入口 . . . 134
7.20.1 识别信号 . . . 134
7.20.2 简单键值配置 . . . 134
7.20.3 读取带空格的配置值 . . . 135
7.20.4 嵌套结构处理思路 . . . 135
7.21 字符串解析 . . . 135
7.21.1 识别信号 . . . 135
7.21.2 逐字符扫描模板 . . . 135
7.21.3 解析 key=value . . . 136
7.21.4 路径解析 . . . 136
7.21.5 括号匹配 . . . 136
7.21.6 字符串解析易错点 . . . 137
7.22 表达式解析入口 . . . 137
7.22.1 识别信号 . . . 137
7.22.2 只含加减的表达式 . . . 137
7.22.3 递归解析思想 . . . 138
7.23 缓存和队列模拟 . . . 138
7.23.1 识别信号 . . . 138
7.23.2 FIFO 缓存 . . . 139
7.23.3 LRU 缓存入口 . . . 139
7.23.4 队列模拟易错点 . . . 139
7.24 懒删除优先队列 . . . 140
7.24.1 识别信号 . . . 140
7.24.2 懒删除思想 . . . 140
7.25 时间和日程模拟 . . . 141
7.25.1 识别信号 . . . 141
7.25.2 时间转分钟 . . . 141
7.25.3 判断区间冲突 . . . 141
7.25.4 扫描事件统计同时进行数量 . . . 141
7.25.5 时间题易错点 . . . 142
7.26 树形结构模拟 . . . 142
7.26.1 识别信号 . . . 142
7.26.2 父子关系存储 . . . 142
7.26.3 DFS 遍历树 . . . 142
7.26.4 子树大小 . . . 143
7.26.5 树题易错点 . . . 143

7.27 权限和规则匹配 . . . 143
7.27.1 识别信号 . . . 143
7.27.2 规则结构体 . . . 143
7.27.3 规则匹配函数 . . . 144
7.27.4 按顺序匹配 . . . 144
7.27.5 规则题易错点 . . . 144
7.28 网格和图混合题 . . . 144
7.28.1 识别信号 . . . 144
7.28.2 普通网格 BFS . . . 145
7.28.3 带状态 BFS 入口 . . . 145
7.28.4 图模拟策略 . . . 145
7.29 大模拟易错点 . . . 146
7.30 C 题分阶段实现 . . . 146
7.30.1 不要一次写完 . . . 146
7.30.2 命令题实现顺序 . . . 146
7.30.3 模拟题实现顺序 . . . 147
7.31 函数化检查清单 . . . 147
7.31.1 建议函数类型 . . . 147
7.31.2 函数设计原则 . . . 147
7.31.3 推荐代码顺序 . . . 147
7.32 分层测试方法 . . . 148
7.32.1 第一层：解析测试 . . . 148
7.32.2 第二层：单操作测试 . . . 148
7.32.3 第三层：组合测试 . . . 148
7.32.4 第四层：边界测试 . . . 148
7.33 C 题常见失分原因 . . . 149
7.34 C 题本地调试 . . . 149
7.34.1 调试输出建议 . . . 149
7.34.2 缩小样例 . . . 149
7.34.3 定位 bug 的方法 . . . 150
7.35 C 题常见边界库 . . . 150
7.36 C 题部分分策略 . . . 150
7.37 C 题分层拿分路线 . . . 151
7.37.1 30 分路线 . . . 151
7.37.2 60 分路线 . . . 151
7.37.3 80 到 100 分路线 . . . 151
7.38 失败降级策略 . . . 151
7.38.1 写不完大模拟 . . . 151
7.38.2 字符串解析写不出来 . . . 152
7.38.3 状态 BFS 写不出来 . . . 152
7.38.4 复杂度可能超时 . . . 152
7.39 C 题临场拿分路线 . . . 152

7.39.1 如果是大模拟 . . . 152
7.39.2 如果是字符串解析 . . . 152
7.39.3 如果是图或网格 . . . 153
7.39.4 如果是规则匹配 . . . 153
7.40 C 题模板索引 . . . 153
7.41 C 题考场速查页 . . . 153
7.41.1 读题 5 分钟内要确定 . . . 153
7.41.2 写代码前必须设计 . . . 154
7.41.3 最常用代码形状 . . . 154
7.41.4 最后 10 分钟检查 . . . 154
7.42 C 题提交前检查 . . . 155
7.43 C 题手写模板清单 . . . 155

**第八章 D/E 题部分分策略与高级算法入口 156**
8.1 D/E 题定位 . . . 156
8.2 D/E 题读题流程 . . . 156
8.3 部分分决策树 . . . 156
8.4 暴力模板 . . . 157
8.4.1 枚举所有点对 . . . 157
8.4.2 枚举所有区间 . . . 157
8.4.3 DFS 枚举选择 . . . 157
8.5 小数据常用算法 . . . 158
8.5.1 Floyd . . . 158
8.5.2 全排列 . . . 158
8.5.3 状压枚举子集 . . . 158
8.6 特殊性质策略 . . . 158
8.6.1 图是链 . . . 158
8.6.2 图是树 . . . 159
8.6.3 边权相同 . . . 159
8.6.4 无修改 . . . 159
8.7 按题型拿部分分 . . . 159
8.7.1 图论题 . . . 159
8.7.2 区间数据结构题 . . . 159
8.7.3 DP 题 . . . 160
8.7.4 字符串题 . . . 160
8.8 树状数组入口 . . . 160
8.8.1 识别信号 . . . 160
8.8.2 树状数组模板 . . . 160
8.9 线段树入口 . . . 161
8.9.1 识别信号 . . . 161
8.9.2 区间和线段树框架 . . . 161
8.9.3 不会线段树时怎么办 . . . 162

8.10 Kruskal 最小生成树入口 . . . 162
8.10.1 识别信号 . . . 162
8.10.2 Kruskal 模板 . . . 163
8.11 树上问题入口 . . . 163
8.11.1 识别信号 . . . 163
8.11.2 树上暴力拿分 . . . 163
8.11.3 LCA 倍增入口 . . . 164
8.11.4 树上差分入口 . . . 165
8.12 状态压缩 DP 入口 . . . 165
8.12.1 识别信号 . . . 165
8.12.2 集合 DP 基本形状 . . . 165
8.12.3 状压易错点 . . . 166
8.13 KMP 字符串匹配入口 . . . 166
8.13.1 识别信号 . . . 166
8.13.2 KMP 模板 . . . 166
8.13.3 不会 KMP 时 . . . 167
8.14 Trie 字典树入口 . . . 167
8.14.1 识别信号 . . . 167
8.14.2 小写字母 Trie . . . 167
8.14.3 不会 Trie 时 . . . 168
8.15 D/E 考场时间策略 . . . 168
8.16 常见高级算法入口 . . . 168
8.17 高级算法识别速查 . . . 168
8.18 D/E 模板练习优先级 . . . 169
8.19 什么时候放弃满分 . . . 169
8.20 D/E 题提交前检查 . . . 170

**第三部分 模板与检查 171**

**第九章 高频算法模板库 172**
9.1 模板库使用方法 . . . 172
9.2 模板速查表 . . . 172
9.3 T101 一维前缀和 . . . 173
9.3.1 适用场景 . . . 173
9.3.2 模板 . . . 173
9.3.3 易错点 . . . 174
9.4 T102 二维前缀和 . . . 174
9.4.1 适用场景 . . . 174
9.4.2 易错点 . . . 175
9.5 T103 一维差分 . . . 175
9.5.1 适用场景 . . . 175
9.5.2 易错点 . . . 175

9.6 T104 二维差分 . . . 175
9.6.1 适用场景 . . . 175
9.7 T105 排序自定义比较器 . . . 176
9.7.1 适用场景 . . . 176
9.7.2 易错点 . . . 176
9.8 T106 哈希统计 . . . 177
9.8.1 值域小 . . . 177
9.8.2 值域大或字符串 . . . 177
9.8.3 易错点 . . . 177
9.9 T107 二分答案 . . . 177
9.9.1 适用场景 . . . 177
9.9.2 最小可行值 . . . 177
9.9.3 最大可行值 . . . 178
9.10 T108 双指针 . . . 178
9.10.1 排序后找两数和 . . . 178
9.10.2 非负数组滑动窗口 . . . 178
9.11 T109 坐标离散化 . . . 179
9.11.1 适用场景 . . . 179
9.12 T110 事件排序 . . . 179
9.12.1 适用场景 . . . 179
9.12.2 区间同时进行数量 . . . 179
9.13 T201 网格 BFS . . . 180
9.13.1 适用场景 . . . 180
9.13.2 易错点 . . . 181
9.14 T202 DFS 连通块 . . . 181
9.14.1 网格 DFS . . . 181
9.14.2 易错点 . . . 181
9.15 T203 并查集 . . . 182
9.15.1 适用场景 . . . 182
9.16 T204 Dijkstra . . . 182
9.16.1 易错点 . . . 183
9.17 T205 拓扑排序 . . . 183
9.17.1 适用场景 . . . 183
9.18 T206 Floyd . . . 184
9.18.1 适用场景 . . . 184
9.19 T207 Kruskal . . . 185
9.19.1 适用场景 . . . 185
9.20 T208 LCA 倍增 . . . 186
9.20.1 适用场景 . . . 186
9.21 T301 01 背包 . . . 186
9.22 T302 完全背包 . . . 187
9.23 T401 树状数组 . . . 187

9.23.1 适用场景 . . . 188
9.24 T402 线段树 . . . 188
9.24.1 适用场景 . . . 188
9.24.2 易错点 . . . 189
9.25 T403 单调栈 . . . 189
9.25.1 适用场景 . . . 189
9.25.2 易错点 . . . 190
9.26 T404 单调队列 . . . 190
9.26.1 适用场景 . . . 190
9.26.2 易错点 . . . 190
9.27 T501 split 字符串分割 . . . 190
9.27.1 按指定字符分割 . . . 190
9.27.2 按空格分割 . . . 191
9.28 T502 KMP . . . 191
9.29 T503 Trie . . . 191
9.30 T510 整除取整和标准取模 . . . 192
9.30.1 适用场景 . . . 192
9.30.2 正整数向上取整 . . . 193
9.30.3 有符号通用版本 . . . 193
9.30.4 标准非负余数 . . . 193
9.30.5 易错点 . . . 193
9.31 T511 快速幂 . . . 194
9.31.1 适用场景 . . . 194
9.32 T512 素数和约数 . . . 194
9.32.1 gcd 和 lcm . . . 194
9.32.2 判断素数 . . . 194
9.32.3 枚举约数 . . . 194
9.33 T513 进制转换 . . . 195
9.33.1 任意进制转十进制 . . . 195
9.33.2 十进制转任意进制 . . . 195
9.34 T514 日期处理 . . . 195
9.34.1 闰年 . . . 195
9.34.2 每月天数 . . . 195
9.34.3 下一天 . . . 196
9.34.4 时间转秒 . . . 196
9.35 T601 大模拟骨架 . . . 196
9.35.1 适用场景 . . . 196
9.36 T602 命令分发 . . . 197
9.36.1 易错点 . . . 198
9.37 T603 状态机 . . . 198
9.37.1 检查点 . . . 198
9.38 T604 懒删除优先队列 . . . 198

9.38.1 适用场景 . . . 198
9.38.2 易错点 . . . 199
9.39 T605 LRU 缓存 . . . 199
9.39.1 适用场景 . . . 199
9.39.2 易错点 . . . 200
9.40 T606 路径规范化 . . . 200
9.40.1 适用场景 . . . 200
9.41 T607 URL 参数解析 . . . 201
9.41.1 适用场景 . . . 201
9.41.2 易错点 . . . 201
9.42 T608 括号匹配 . . . 202
9.42.1 适用场景 . . . 202
9.43 T609 简单表达式求值 . . . 202
9.43.1 只含加减和非负整数 . . . 202
9.43.2 易错点 . . . 203

**第十章 易错点总表 204**
10.1 本章使用方法 . . . 204
10.2 输入输出易错点 . . . 204
10.3 数据类型和溢出 . . . 204
10.4 整除、取整和取模 . . . 205
10.5 下标和边界 . . . 205
10.6 初始化和清空 . . . 205
10.7 排序和比较器 . . . 205
10.8 STL 容器易错点 . . . 206
10.9 字符串易错点 . . . 206
10.10 图论易错点 . . . 206
10.11 动态规划易错点 . . . 207
10.12 数据结构易错点 . . . 207
10.13 提交前总检查 . . . 207
10.14 最后十分钟清单 . . . 208

**第四部分 真题校准 209**

**第十一章 历年真题索引与模板映射 210**
11.1 本章使用方法 . . . 210
11.2 索引可信度 . . . 210
11.3 索引字段说明 . . . 211
11.4 近年整场校准索引 . . . 211
11.5 A 题索引 . . . 212
11.6 B 题索引 . . . 213
11.7 C 题索引：大模拟与中档算法入口 . . . 215

11.8 D/E 题索引：只为部分分服务 . . . . . 216
11.9 题目反查模板索引 . . . . . 217
11.10 按题面信号反查真题 . . . . . 218
11.11 真题校准清单 . . . . . 219
11.12 近年题型覆盖缺口 . . . . . 219
11.13 A/B 题型归纳 . . . . . 219
11.13.1 A 题高频题型 . . . . . 219
11.13.2 B 题高频题型 . . . . . 220
11.14 索引维护流程 . . . . . 220
11.15 资料来源记录 . . . . . 220

**版本记录** **222**

# 第一部分

# 导航与判断

# 第一章　全书总览与使用路径

## 1.1　本手册的工作方式

这本手册按“题目识别”而不是按“算法教材目录”组织。考场上你通常不是先想起某个算法，再去找题目匹配它；更常见的是先看到题面中的若干信号，例如区间、多次询问、最短、连通、排序、字符串、日期、进制、命令序列，然后再判断应该翻到哪个章节。

因此，本手册的每个决策入口都尽量回答下面七个问题：

1. 题面中出现了什么信号？
2. 数据范围允许什么复杂度？
3. 它更像 A、B、C、D、E 中哪类题？
4. 先尝试哪个模板？
5. 需要使用哪些 C++ 语法或 STL 容器？
6. 最容易错在哪里？
7. 如果满分做法不会，能否先写部分分？

**使用原则：**不要在考场上临时学习一个完全没练过的模板。模板页是用来提醒和防错的，不是用来从零学习的。

## 1.2　全书结构路线图

本书按考场使用顺序重排为四个分部。先用索引定位，再用题型章节展开，最后用模板和真题反向校准。

| 分部 | 章节 | 用法 |
| --- | --- | --- |
| 导航与判断 | 第 1–3 章 | 先看全书用法，再用 1 分钟极速索引定位，必要时进入详细判断树。 |
| 基础与题型 | 第 4–8 章 | 先补 C++ 和 STL 基础，再按 A/B/C/D/E 题型查看策略和高频入口。 |
| 模板与检查 | 第 9–10 章 | 找可复制模板，并在提交前扫易错点。 |
| 真题校准 | 第 11 章 | 用历年题反查模板和判断树，发现误判后回到第 3 章修正。 |

推荐阅读路线：

1. **考前通读：**第 1 章、第 4 章、第 5–8 章、第 10 章。
2. **刷题复盘：**第 3 章判断路径、第 9 章模板、第 11 章真题索引。
3. **考场翻查：**第 2 章极速索引；定位不准时翻第 3 章；写代码时翻第 9 章；提交前翻第 10 章。

## 1.3 官方规则速查

本节只作为考前检查清单。正式打印前必须重新访问 CCF CSP 官网核对最新版本。

| 项目 | 当前应记录的信息 |
| --- | --- |
| 官方网站 | https://www.cspro.org/ |
| 认证形式 | 上机编程，黑盒测试，按测试点得分。 |
| 题目数量 | 通常为 5 道编程题。每题 100 分，总分 500 分。 |
| 考试时间 | 通常为 240 分钟。 |
| 输入输出 | 使用标准输入、标准输出。不要输出提示语、调试信息或多余空格。 |
| 提交方式 | 每题提交一个源文件；同一题多次提交时，一般以最后一次提交为准。 |
| 语言环境 | 以官网报名系统和当次考试公告为准。C++ 版本、编译器和操作系统可能变化。 |
| 纸质资料 | 根据 CCF 官网答疑和考生须知，应考时可携带纸质书籍、打印资料、手写资料；不得使用通讯工具、存储设备。考前必须再次核对。 |
| 程序长度 | 官网注意事项中提到程序长度限制，准备模板时不要复制大量无关代码。 |

建议在打印版首页手写确认：

- 我已经在考前一周核对了官网规则。
- 我已经确认了本次考试允许携带的纸质资料范围。
- 我已经确认了本次考试 C++ 编译标准。
- 我知道不能携带电子资料、存储设备或联网设备。

## 1.4 考场总流程

### 1.4.1 开场 10 分钟

开场不要立刻写代码。先快速浏览 5 道题，目标是建立时间分配和得分路线。

1. 看每道题的题面长度。题面短不一定简单，题面长往往意味着模拟或细节多。
2. 看每道题的数据范围。数据范围比题面故事更重要。
3. 判断 A、B 是否可以快速完成。

4. 判断 C 是算法题还是大模拟。
5. 判断 D、E 是否有子任务、小数据、特殊性质或熟悉模板。
6. 在草稿纸上写下计划：先做哪题、卡住多久换题、最后回来看哪题。

### 1.4.2 每题固定流程

每道题都按同一套流程处理：

1. 读题面，圈出关键词。
2. 抄下所有数据范围。
3. 估算可接受复杂度。
4. 查本章总决策树，定位章节。
5. 写出暴力思路，用来理解题意和造样例。
6. 如果满分模板明确，再写正式代码。
7. 先过样例，再造边界样例。
8. 删除调试输出后提交。

**不要跳过复杂度估算。**很多新手失分不是因为不会算法，而是把 $O(n^2)$ 写到了 $n = 10^5$ 的题里。

### 1.4.3 建议时间分配

| 题号 | 建议时间 | 目标 | 卡住时的处理 |
|---|---|---|---|
| A | 20–35 分钟 | 稳定满分 | 15 分钟没思路就重新读题；30 分钟没过样例先跳。 |
| B | 40–70 分钟 | 尽量满分 | 优先查前缀和、差分、排序、哈希、BFS。 |
| C | 60–100 分钟 | 高分或满分 | 如果是大模拟，先拆结构体和函数，不要全写在主函数。 |
| D | 40–80 分钟 | 部分分到满分 | 先找小数据和熟悉模板。不会满分也要写暴力。 |
| E | 剩余时间 | 尽量拿分 | 重点看子任务、特殊性质、样例规律。 |

时间分配不是固定规则。若 A、B 很顺，可以把更多时间留给 C、D；若 A、B 卡住，不要让前两题吞掉整场考试。

## 1.5 数据范围与复杂度

### 1.5.1 复杂度速查表

| 输入规模 | 通常可考虑 | 常见模板 |
| --- | --- | --- |
| $n \le 20$ | 指数搜索、状态压缩、全排列、DFS 暴力 | DFS、状压 DP、枚举子集 |
| $n \le 100$ | $O(n^3)$ 通常可试 | Floyd、区间 DP、三重枚举 |
| $n \le 500$ | $O(n^3)$ 可能可行，需看时限 | Floyd、部分 DP |
| $n \le 1000$ | $O(n^2)$ 通常可行 | 双重循环、二维 DP、朴素图算法 |
| $n \le 5000$ | $O(n^2)$ 需谨慎 | 优化前的 DP、枚举配合剪枝 |
| $n \le 10^5$ | $O(n \log n)$ 或 $O(n)$ | 排序、二分、前缀和、差分、堆、树状数组 |
| $n \le 10^6$ | $O(n)$ 或较轻的 $O(n \log n)$ | 线性扫描、计数、前缀和 |
| $n \ge 10^7$ | 通常只能线性或公式 | 数学、简单扫描、避免大对象数组 |

### 1.5.2 复杂度校验规则

- 看到双重循环，先计算最坏次数。
- 有多组数据时，总复杂度要乘以组数，或者看所有 $n$ 的总和限制。
- 排序一般是 $O(n \log n)$，$n = 10^5$ 常见可行。
- Floyd 是 $O(n^3)$，只适合点数较小的图。
- Dijkstra 配合堆是 $O((n+m)\log n)$，适合正权稀疏图。
- BFS 适合边权全为 1 或每步代价相同的最短步数。
- 前缀和适合不修改的区间查询。
- 差分适合批量区间修改后统一查询。
- 树状数组和线段树适合修改与查询交织出现。

**溢出提醒：**只要出现求和、乘法、路径长度、方案数、时间累计，默认先考虑 `long long`。两个 `int` 相乘时写成 `1LL * a * b`。

## 1.6 A/B/C/D/E 总体策略

### 1.6.1 A 题

A 题通常考察基本实现能力。它可能是简单模拟、统计、字符串、日期、进制、排序或公式题。

- 目标：尽量一次 AC。
- 优先检查：输入格式、输出格式、整数溢出、小数精度、边界值。

- 常用章节：C++ 基础、数组统计、字符串、进制、日期、排序。
- 不建议：为了追求代码短而写不熟悉的技巧。

### 1.6.2 B 题

B 题常常是第一个真正需要算法识别的题，但算法通常不会太深。

- 高频方向：排序、前缀和、差分、哈希、二分、BFS、简单图、简单 DP。
- 读题重点：是否有多次询问，是否有区间，是否需要快速统计。
- 常见错误：用暴力过样例但过不了大数据。

### 1.6.3 C 题

C 题经常是分水岭。它可能是大模拟，也可能是中等算法题。

- 如果题面很长，先判断是否是规则模拟。
- 如果输入是命令、事件、状态变化，优先设计结构体和函数。
- 如果出现图、网格、路径、依赖关系，再进入对应算法分支。
- 大模拟题不要把所有逻辑塞进 `main`。

### 1.6.4 D/E 题

D、E 题要同时考虑满分算法和部分分。

- 先看是否给出子任务或特殊数据范围。
- 先写能过小数据的暴力，避免空题。
- 若满分算法明显且练过，再冲满分。
- 若 20 分钟没有形成可写方案，转为部分分策略。

## 1.7 总决策树

### 1.7.1 第一层：看题面信号

| 题面信号 | 优先怀疑 | 先翻章节 |
| --- | --- | --- |
| 字符、单词、路径、命令、表达式、括号 | 字符串处理或解析模拟 | 字符串专题、栈、大模拟 |
| 二进制、十六进制、位、补码、编码 | 进制和位运算 | 进制专题、位运算 |
| 日期、时间、星期、天数、日历 | 日期时间处理 | 日期专题 |
| 区间和、多次查询、子矩阵和 | 前缀和 | 前缀和模板 |

| | | |
|---|---|---|
| 区间加、范围修改、最后输出 | 差分 | 差分模板 |
| 修改和查询交替出现 | 树状数组或线段树 | 数据结构模板 |
| 最大值、最小值动态维护 | 堆、集合、单调结构 | 优先队列、set、单调队列 |
| 排序、排名、优先级、相同条件按某字段 | 自定义排序 | sort 与 lambda |
| 去重、出现次数、频率统计 | 哈希或 map | map、unordered map、计数数组 |
| 最少步数、迷宫、网格移动 | BFS | BFS、网格 BFS |
| 正权最短路、道路费用、距离最小 | Dijkstra | 最短路模板 |
| 任意两点最短路且点数小 | Floyd | Floyd 模板 |
| 合并集合、是否连通、同一类 | 并查集 | 并查集模板 |
| 依赖、先后顺序、任务调度 | 拓扑排序 | 拓扑排序模板 |
| 最大化最小值、最小化最大值、答案可行性 | 二分答案 | 二分答案模板 |
| 容量、价值、选择若干个、最多或恰好 | 背包 DP | 动态规划模板 |
| 规则很多、对象很多、按时间变化 | 大模拟 | 大模拟专题 |

### 1.7.2 第二层：用数据范围排除错误方向

| 如果你想写 | 但数据范围是 | 应改查 |
|---|---|---|
| 枚举所有点对 $O(n^2)$ | $n = 10^5$ | 排序、哈希、前缀和、树状数组、双指针 |
| 枚举所有区间 $O(n^2)$ | $n = 10^5$ | 前缀和、双指针、单调队列、DP 优化 |
| Floyd $O(n^3)$ | $n > 500$ 或边很多 | Dijkstra、BFS、图的特殊性质 |
| 每次查询都扫描数组 | 查询数 $q = 10^5$ | 前缀和、树状数组、线段树、离线处理 |
| 递归搜索所有方案 | $n > 25$ 且无剪枝 | DP、贪心、数学、二分答案 |
| 字符串反复插入删除 | 字符串长度和操作数很大 | 分块、链表思想、栈、模拟优化 |

### 1.7.3 第三层：选择满分或部分分

1. 如果模板明确、数据范围匹配、自己练过，写满分做法。
2. 如果模板大致明确但细节多，先写能跑的小版本，再逐步补全。
3. 如果满分算法不明确，找子任务和小数据，写暴力或特殊情况。
4. 如果题目很难但 A/B/C 还没稳，不要长时间停留在 D/E。

## 1.8 关键词到章节索引

这张表用于快速翻书。后续章节完成后，需要把“章节编号”和“模板编号”补齐。

| 关键词 | 翻到哪里 | 核心提醒 |
|---|---|---|
| 字符串、字符、大小写 | 字符串专题 | 注意 `cin` 和 `getline` 混用。 |
| 空格分割、命令行 | 字符串分割 | 用 `stringstream` 或手写分割。 |
| 括号、嵌套、表达式 | 栈、表达式解析 | 注意优先级和匹配失败。 |
| 进制转换 | 进制专题 | 判断是否超过 `long long`。 |
| 二进制位 | 位运算 | 大位数用 `1LL << k`。 |
| 日期、星期 | 日期专题 | 先确认闰年和月份天数。 |
| 区间查询 | 前缀和 | 不修改时优先前缀和。 |
| 区间修改 | 差分 | 修改后统一查询时优先差分。 |
| 在线修改查询 | 树状数组、线段树 | 看是单点还是区间。 |
| 排序、排名 | sort 与 lambda | 相同条件要写完整。 |
| 频率、出现次数 | map、unordered map、计数数组 | 值域小用数组更快。 |
| 最短步数 | BFS | 边权相同才用 BFS。 |
| 正权最短路 | Dijkstra | 距离用 `long long`。 |
| 连通、合并 | 并查集 | 路径压缩和按大小合并。 |
| 依赖、先后 | 拓扑排序 | 入度为 0 入队。 |
| 最优选择 | DP 或贪心 | 先看是否有状态转移。 |
| 答案范围 | 二分答案 | 必须能写 `check`。 |
| 大量规则 | 大模拟 | 结构体、函数、状态表。 |

## 1.9 低级错误总检查

提交前至少检查下面内容：

- 是否删除所有调试输出。
- 是否使用标准输入输出。
- 输出末尾是否多了不该有的内容。
- 是否把答案变量开成 `long long`。
- 数组是否开到最大范围。

- 下标是从 0 还是从 1 开始。
- 循环边界是否包含末尾。
- 多组数据时是否清空数组、图、队列、map。
- 比较器是否满足严格弱序。
- `lower_bound` 或 `find` 的返回值是否检查。
- `priority_queue` 是否需要小根堆。
- BFS 是否在入队时标记访问。
- Dijkstra 是否跳过过期状态。
- 递归深度是否可能爆栈。

## 1.10 部分分总策略

当满分做法暂时不会时，按这个顺序找分：

1. **小数据暴力：**如果 $n \le 1000$ 的测试点存在，写 $O(n^2)$；如果 $n \le 100$，考虑 $O(n^3)$。
2. **特殊结构：**图是树、链、完全图、DAG、无权图时，可能有更简单做法。
3. **特殊参数：**如果某个参数为 0、1、很小，单独写分支。
4. **无修改版本：**有修改和查询时，若部分数据没有修改，先用前缀和或静态算法。
5. **少量查询或少量修改：**查询少就每次暴力；修改少就重算或离线处理。
6. **样例和明显边界：**最后时间不足时，也要保证样例和简单边界正确。

**部分分代码也要干净。**不要写一堆互相冲突的特判。特判越多，越容易把能拿的分弄丢。

## 1.11 本章完成后的后续连接

后续章节需要逐步补齐以下链接：

- 将“关键词到章节索引”中的章节编号补全。
- 将每个决策树叶子节点连接到具体模板编号。
- 将真题索引中的每道题反向连接到本章的某个判断路径。
- 将个人错题中的错误补入“低级错误总检查”。

# 第二章　考场 1 分钟极速索引

## 2.1　本章定位

本章是考试时最先翻的压缩索引。完整判断树在第 3 章，完整模板在第 9 章，完整真题索引在第 11 章。本章只解决一个问题：**看到题面后，最快翻到该用的模板和注意点**。

**使用方法：**每道题先用本章 1 到 2 分钟定位。如果定位不准，再回第 3 章看详细判断树。

## 2.2　1 分钟总流程

| 时间 | 做什么 | 目的 |
| --- | --- | --- |
| 前 10 秒 | 看题号：A/B/C/D/E | 决定稳分、冲分、拆题、部分分。 |
| 第 10-25 秒 | 圈出输入主体：数组、矩阵、字符串、图、树、对象、事件 | 确定大类。 |
| 第 25-40 秒 | 圈出操作：查询、修改、排序、最优、路径、状态变化 | 确定模板方向。 |
| 第 40-55 秒 | 看数据范围：$n, m, q$、值域、字符串总长 | 排除不可能复杂度。 |
| 最后 5 秒 | 翻模板并写骨架 | 不在脑子里硬背。 |

## 2.3　题号策略速查

| 题号 | 目标 | 常见题型 | 第一动作 |
| --- | --- | --- | --- |
| A | 稳拿满分 | 统计、公式取整、排序、字符串、日期、二维数组 | 先写暴力扫描，检查输出格式。 |
| B | 尽量满分 | 前缀、差分、哈希、排序扫描、二分、简单 DP | 先写暴力，再找重复计算。 |
| C | 尽量拿高分 | 大模拟、字符串解析、状态机、事件、树/目录 | 先建对象和状态，不急着写代码。 |
| D | 能拿部分分就拿 | 图、树、DP、数据结构、数学 | 先看子任务和特殊性质。 |
| E | 保底部分分 | 高级算法或综合题 | 写小数据暴力，不耗尽时间。 |

## 2.4 题面主体到模板

| 主体 | 题面信号 | 模板/章节 |
|---|---|---|
| 数组 | 求和、计数、相邻、分段、排名、分组 | A 题、T101、T105、T106、T510 |
| 矩阵 | 图像、矩形、邻域、旋转、重塑 | T102、T104、矩阵扫描 |
| 字符串 | 分隔符、路径、URL、括号、表达式 | T501、T606、T607、T608、T609 |
| 坐标点 | 点集、距离、矩形、覆盖、邻居 | T106、T109、坐标公式 |
| 图 | 最短路、连通、依赖、最小代价 | T201、T203、T204、T205、T207 |
| 树 | 子树、祖先、路径、目录、合并 | DFS、T208、T402 |
| 多命令 | create、delete、query、request、release | T601、T602、T603 |
| 时间事件 | 到期、归还、租约、按时刻发生 | T110、T514、T603 |
| 选择物品 | 容量、费用、价值、至少达到 | T301、T302 |
| 最优答案 | 最小时间、最大阈值、最小最大 | T107 或 DP |

## 2.5 数据范围秒判

| 范围 | 可以想 | 不要想 |
|---|---|---|
| $n \le 20$ | 枚举、DFS、状压 | 复杂数据结构优先 |
| $n \le 100$ | $O(n^3)$、Floyd | 指数暴力 |
| $n \le 1000$ | $O(n^2)$ | $O(n^3)$ 大概率危险 |
| $n \le 10^5$ | $O(n \log n)$、$O(n)$ | $O(n^2)$ |
| $q \le 10^5$ | 每次 $O(\log n)$ 或 $O(1)$ | 每次扫描全体 |
| 值域小 | 计数数组、桶 | map 不一定必要 |
| 值域大但数量少 | map、hash、离散化 | 直接开值域数组 |

## 2.6 B 题优化入口

| 暴力写法 | 应想到 | 模板 |
|---|---|---|
| 每次区间求和都循环 | 前缀和 | T101 |
| 每次矩形求和都双重循环 | 二维前缀和 | T102 |
| 每次区间加都逐个改 | 差分 | T103 / T104 |
| 每次查是否出现都循环 | set / unordered_set | T106 |
| 排序后每次找位置 | 二分 / lower_bound | C++ 语法章 |
| 两个有序表逐个配对 | 双指针 | T108 |

| | | |
|---|---|---|
| 最小时间、最大阈值 | 二分答案 | T107 |
| 选若干个凑金额 | 背包 | T301 |
| 前置任务、依赖关系 | 拓扑排序 | T205 |

## 2.7 C 题建模速查

C 题先填这张表，再写代码。

| 要素 | 题面中找什么 | 代码怎么落地 |
|---|---|---|
| 对象 | 用户、文件、任务、节点、请求、词条 | `struct Obj` |
| 编号 | 数字编号、字符串名、路径 | vector / map / unordered_map |
| 状态 | 有效、过期、占用、空闲、允许、拒绝 | enum/int 状态机 |
| 命令 | 每行第一个单词或操作类型 | T602 命令分发 |
| 时间 | 当前时刻、持续时间、到期时间 | T110 / T514 |
| 优先级 | “优先”“若相同”“同时发生” | sort 比较器或 priority_queue |
| 失败规则 | 不存在、非法、超额、冲突 | 先判断，后修改，必要时回滚 |

## 2.8 D/E 部分分入口

| 看到子任务 | 写什么 | 可能分数来源 |
|---|---|---|
| $n \le 20$ | 子集枚举、DFS、状压 | 小数据分 |
| $n \le 100$ | Floyd、$O(n^3)$、暴力 DP | 中小数据分 |
| 链 | 当数组处理，前缀/差分 | 特殊性质分 |
| 树 | 每次 DFS，或 DFS 序 | 树特殊性质分 |
| 无修改 | 预处理后回答查询 | 静态分 |
| 查询少 | 每次暴力 | 查询少子任务 |
| 边权 1 | BFS | 图特殊性质分 |
| 值域小 | 计数数组 | 小值域分 |

## 2.9 模板编号速查

| 编号 | 模板 | 编号 | 模板 | 编号 | 模板 |
|---|---|---|---|---|---|
| T101 | 一维前缀和 | T102 | 二维前缀和 | T103 | 一维差分 |
| T104 | 二维差分 | T105 | 排序比较器 | T106 | 哈希统计 |
| T107 | 二分答案 | T108 | 双指针 | T109 | 离散化 |
| T110 | 事件排序 | T201 | 网格 BFS | T203 | 并查集 |
| T204 | Dijkstra | T205 | 拓扑排序 | T206 | Floyd |
| T207 | Kruskal | T208 | LCA | T301 | 01 背包 |

| | | | | | |
|---|---|---|---|---|---|
| T401 | 树状数组 | T402 | 线段树 | T501 | split |
| T510 | 整除取整 | T511 | 快速幂 | T512 | 素数约数 |
| T513 | 进制转换 | T514 | 日期处理 | T601 | 大模拟骨架 |
| T602 | 命令分发 | T603 | 状态机 | T604 | 懒删除堆 |
| T606 | 路径规范化 | T607 | URL 解析 | T608 | 括号匹配 |
| T609 | 表达式入门 | – | – | – | – |

## 2.10 C++ 语法救急

| 需求 | 写法 |
|---|---|
| 快速输入输出 | `ios::sync_with_stdio(false);` `cin.tie(nullptr);` |
| long long | 所有和、乘法、坐标平方、答案上界优先用 `long long`。 |
| 正整数向上取整 | $a \ge 0, b > 0$ 时用 `(a+b-1)/b`；有负数时查 T510。 |
| 标准非负余数 | `r=x%m; if (r<0) r+=m;`，用于负数取模下标。 |
| vector 排序 | `sort(a.begin(), a.end());` |
| 按结构体字段排序 | 用 `sort` 加 lambda；完整写法见第 4 章排序小节。 |
| map 计数 | `mp[x]++;` |
| set 判断存在 | `if (st.count(x)) ...` |
| lower_bound | 先排序；返回迭代器后检查是否等于 `end()`。 |
| 小根堆 | `priority_queue<int, vector<int>, greater<int>>`。 |
| 保留小数 | `cout << fixed << setprecision(k) << ans;` |

## 2.11 最后提交前 30 秒

1. 数组是否开够，是否有 $n+1$、$m+1$、边界哨兵。
2. 所有乘法、前缀和、答案是否用了 `long long`。
3. 多组数据是否清空 vector、map、队列、数组。
4. 排序比较器是否满足严格弱序，相等时是否按题目要求。
5. `lower_bound`、`find`、map 查询是否检查不存在。
6. 输出空格、换行、小数位是否和样例完全一致。
7. C 题非法操作是否真的没有改状态。
8. D/E 暴力是否至少能过样例和小数据。

# 第三章　题型详细判断树

## 3.1　本章定位

本章是整本手册的导航核心。考场上不要从算法目录开始翻，而应先用本章判断：

1. 这题属于哪一类？
2. 数据范围允许什么复杂度？
3. 题面信号对应哪个模板？
4. 如果满分不会，有什么部分分做法？

**判断树不是保证一眼看出满分算法的魔法。**它的作用是减少迷路：让你从题面信号稳定定位到最可能的算法和模板。

## 3.2　考场三层分流法

读任何题都按三层走。不要直接猜算法名。

| 层级 | 问什么 | 作用 |
| --- | --- | --- |
| 第一层：题号和任务 | A/B/C/D/E？要求输出一个值、一组答案，还是处理很多命令？ | 决定目标：稳拿、冲分、拆模拟、拿部分分。 |
| 第二层：题面结构 | 输入是数组、矩阵、图、字符串、对象、事件、树，还是混合规则？ | 决定先翻哪个章节。 |
| 第三层：操作和范围 | 是否多次查询/修改？$n, m, q$ 多大？值域多大？ | 决定具体模板和复杂度。 |

## 3.3　总判断流程

| 步骤 | 问题 | 动作 |
| --- | --- | --- |
| 1 | 这是第几题？ | A/B 先求稳，C 先拆题，D/E 先找部分分。 |
| 2 | 输出形式是什么？ | 单个答案偏统计/公式；多行答案偏查询/命令；复杂文本偏解析。 |
| 3 | 数据范围多大？ | 用复杂度表排除暴力或高级算法。 |

| | | |
|---|---|---|
| 4 | 有多次查询或修改吗? | 查前缀和、差分、树状数组、线段树、离线处理。 |
| 5 | 有图、网格、路径吗? | 查 BFS、DFS、Dijkstra、并查集、拓扑排序。 |
| 6 | 有字符串格式吗? | 查 split、路径、URL、括号、表达式、KMP。 |
| 7 | 有状态、命令、事件吗? | 查大模拟、状态机、事件排序。 |
| 8 | 有最优值、选择、容量吗? | 查 DP、贪心、二分答案。 |
| 9 | 满分算法不明确吗? | 立即找小数据和特殊性质分。 |

## 3.4 决策树叶子节点标准格式

本章后面的详细决策树尽量使用同一种叶子格式。考场上每次走到叶子，都要确认 5 件事：

| 项目 | 必须回答的问题 |
|---|---|
| 题面信号 | 是什么词、输入结构或输出要求让我走到这里? |
| 数据范围 | 暴力能不能过? 目标复杂度应是多少? |
| 模板入口 | 应翻第几章、哪个模板编号? |
| 易错点 | 下标、边界、溢出、排序规则、输出格式哪里最容易错? |
| 保底做法 | 满分不会时，能不能先写暴力、小数据或特殊性质版本? |

**如果某个叶子节点无法回答“数据范围”和“模板入口”，不要立刻写代码，先回到数据范围判断树重新分流。**

## 3.5 从根节点开始的总决策树

下面这棵树按考场读题顺序走。先用它确定大方向，再进入 A/B/C/D/E 专项树。

| 节点 | 问题 | 是：走向 | 否：走向 |
|---|---|---|---|
| R0 | 这是 A 题吗? | 进入 A 题详细树，先求稳。 | 看 R1。 |
| R1 | 这是 B 题吗? | 进入 B 题详细树，先找重复计算。 | 看 R2。 |
| R2 | 这是 C 题吗? | 进入 C 题详细树，先建模。 | 看 R3。 |
| R3 | 这是 D/E 题吗? | 先看子任务，进入部分分树。 | 回题面确认题号。 |
| R4 | 输出只有一个数或少量值吗? | 优先统计、公式、排序、DP。 | 看是否多次查询或命令。 |
| R5 | 每次输入操作后都要输出吗? | 命令分发、数据结构、状态机。 | 可能是最后统一输出的模拟/预处理。 |
| R6 | 有 $q$ 次查询/修改且 $q$ 大吗? | 前缀、差分、树状数组、线段树、离线。 | 暴力或简单模拟可能可行。 |
| R7 | 有“最小/最大/阈值/最少资源”吗? | 检查单调性，可能二分答案或 DP。 | 看图、字符串、状态。 |
| R8 | 有“路径/连通/最短/依赖”吗? | 图论或拓扑。 | 看字符串和大模拟。 |

| | | |
|---|---|---|
| R9 | 有复杂格式、路径、括号、字符串解析、栈、递归解析。 | 普通数组/矩阵/对象模拟。 |
| | 表达式吗? | |

## 3.6 详细叶子索引总表

这张表是全局的“最终落点”索引。专项树走不清楚时，直接查这一张。

| 叶子 | 题面信号 | 数据范围判断 | 模板入口 | 保底做法 |
|---|---|---|---|---|
| L01 统计扫描 | 求和、计数、最大最小 | $O(n)$ 可过 | A 题、T106 | 一遍循环。 |
| L02 排序排名 | 排名、中位数、多关键字 | $O(n\log n)$ 可过 | T105 | 先按单关键字过样例。 |
| L03 一维区间 | 多次区间和/数量 | 静态查询多 | T101 | 每次暴力扫小数据。 |
| L04 二维矩阵 | 子矩阵、邻域、图像 | $nm$ 可扫，多次查询用预处理 | T102/T104 | 小矩阵暴力。 |
| L05 哈希查重 | 是否出现、频率、点集 | 值域大或字符串键 | T106 | map 替代 unordered_map。 |
| L06 坐标离散 | 坐标大、点少 | 不能直接开数组 | T109 | map 坐标。 |
| L07 二分答案 | 最小时间、最大阈值 | check 单调且快 | T107 | 枚举小范围答案。 |
| L08 背包选择 | 选若干、容量、金额 | 总容量可 DP | T301/T302 | DFS 小数据。 |
| L09 BFS | 最短步数、边权 1 | 状态数可控 | T201 | 小图暴力搜索。 |
| L10 Dijkstra | 正权最短路 | $n, m$ 大，边权非 1 | T204 | 小图 Floyd。 |
| L11 拓扑 | 前置、依赖、逻辑门 | DAG 或需判环 | T205 | 小图反复找入度 0。 |
| L12 并查集 | 合并集合、连通性 | 只合并查连通 | T203 | BFS/DFS 小数据。 |
| L13 事件模拟 | 按时间发生、到期 | 事件数可排序 | T110 | 按输入顺序模拟小数据。 |
| L14 状态机 | 对象状态变化 | 状态少且规则明确 | T603 | if/else 先过样例。 |
| L15 路径/URL | 路径、参数、配置 | 总长度可线性 | T606/T607 | 手写 split。 |
| L16 表达式 | 括号、优先级、化学式 | 嵌套深度可递归 | T608/T609 | 暴力解析小表达式。 |
| L17 树上路径 | 祖先、子树、路径 | 查询多需预处理 | T208/T402 | 每次 DFS。 |
| L18 动态区间 | 修改和查询交替 | $q$ 大 | T401/T402 | 修改少就重算。 |
| L19 大模拟 | 长题面、多对象规则 | 规则多于算法 | T601/T602 | 分命令逐步拿分。 |
| L20 D/E 小数据 | 子任务 $n$ 小或特殊性质 | 暴力能过一档 | 第 8 章 | 先交可运行版本。 |

## 3.7 题面结构第一判断

| 输入主体 | 题面信号 | 第一反应 | 翻到哪里 |
|---|---|---|---|
| 一串数字 / 数组 | 求和、最大、最小、相邻、分段、前缀 | 扫描、排序、前缀和 | A/B、T101、T105 |
| 二维矩阵 | 图像、网格、矩形、邻域、旋转、重塑 | 下标转换、二维前缀、矩阵模拟 | A/B、T102、T104 |
| 很多字符串行 | 路径、URL、JSON、Markdown、命令、表达式 | 字符串解析、大模拟 | C、T501、T606、T607 |
| 点和边 | 连通、最短路、依赖、拓扑、最小代价 | 图论模板 | T201–T207 |
| 树 | 子树、祖先、路径、删点、合并 | DFS、LCA、树形 DP、DFS 序 | C/D/E、T208 |
| 多条操作 | 修改、查询、插入、删除、回滚 | 数据结构或离线处理 | T103、T401、T402 |
| 时间顺序事件 | 某时刻发生、到期、优先级、调度 | 事件排序、状态机、堆 | C、T110、T603、T604 |
| 选择若干物品 | 容量、费用、价值、至少/至多 | 背包 DP 或贪心 | T301、T302 |
| 答案最优值 | 最小时间、最大阈值、最小最大 | 二分答案或 DP | T107、D/E |

## 3.8 数据范围判断树

### 3.8.1 单变量规模

| 范围 | 可考虑 | 不宜考虑 |
|---|---|---|
| $n \le 10$ | 全排列、爆搜、手写模拟所有情况 | 不必急着写复杂优化 |
| $n \le 20$ | 子集枚举、状压 DP、折半搜索 | $O(n!)$ 仍危险 |
| $n \le 100$ | $O(n^3)$、Floyd、区间 DP | 指数算法 |
| $n \le 1000$ | $O(n^2)$、简单 DP、稠密图 | $O(n^3)$ 需谨慎 |
| $n \le 10^5$ | $O(n \log n)$、$O(n)$ | $O(n^2)$ |
| $n \le 10^6$ | $O(n)$、计数数组、简单筛法 | 多层 map 或重型结构 |
| $n \ge 10^7$ | 只考虑线性或数学公式 | 大数组套多维结构 |

### 3.8.2 多变量规模

| 范围组合 | 判断 | 常见模板 |
|---|---|---|
| $n, m \le 500$ 且矩阵 | $O(nm)$ 或 $O(nm \log nm)$ 通常可行 | 二维扫描、T102、T104 |
| $n, m \le 2000$ 且矩阵 | 尽量 $O(nm)$，少做多次全图扫描 | 二维前缀、BFS |

| | | |
|---|---|---|
| $q \le 10^5$ | 每次操作必须快 | 前缀、差分、树状数组、线段树 |
| 值域 $\le 10^6$ | 可以开计数数组 | 计数、桶、前缀计数 |
| 值域很大但数量少 | 不可直接开值域数组 | map、unordered_map、离散化 |
| 边数 $m \approx n$ | 稀疏图 | 邻接表、BFS、Dijkstra 堆优化 |
| 边数 $m \approx n^2$ 且 $n$ 小 | 稠密图 | Floyd、邻接矩阵 |
| 字符串总长 $\le 10^6$ | 按总长度线性处理 | split、KMP、Trie |

**数据范围优先级高于题面关键词。**例如题面写“最短路”，但 $n \le 100$ 且多源查询，可以用 Floyd；若 $n, m \le 10^5$，通常要 Dijkstra 或 BFS。

## 3.9 关键词到模板总索引

| 题面关键词 | 判断 | 模板 | 章节 |
|---|---|---|---|
| 多次区间和 | 静态区间查询 | T101 一维前缀和 | B / 模板库 |
| 子矩阵和、邻域均值 | 二维静态查询 | T102 二维前缀和 | B / 模板库 |
| 区间加，最后查询 | 离线修改 | T103 一维差分 | B / 模板库 |
| 矩形加，最后输出 | 二维离线修改 | T104 二维差分 | B / 模板库 |
| 排名、优先级、多关键字 | 比较器排序 | T105 排序比较器 | A/B/C |
| 出现次数、频率、查重 | 统计 | T106 哈希统计 | A/B |
| 最小化最大值、最大化最小值 | 二分答案 | T107 二分答案 | B/D |
| 排序后找两数、合并稀疏向量 | 双指针 | T108 双指针 | B |
| 坐标很大、点不多 | 离散化 | T109 坐标离散化 | B/D |
| 按时间发生、到期、归还 | 事件模拟 | T110 事件排序 | B/C |
| 最短步数、迷宫、扩散 | BFS | T201 网格 BFS | B/C |
| 连通块、区域大小 | DFS/BFS | T202 DFS 连通块 | B/C |
| 合并集合、同组、连通性 | 并查集 | T203 并查集 | B/D |
| 正权最短路 | Dijkstra | T204 Dijkstra | B/D |
| 依赖、先后顺序、任务计划 | 拓扑排序 | T205 拓扑排序 | B/D |
| 任意两点最短路，小图 | Floyd | T206 Floyd | B/D |
| 最小连接代价 | Kruskal | T207 Kruskal | D/E |
| 树上祖先、路径 | LCA | T208 LCA | D/E |
| 选或不选、容量 | 01 背包 | T301 01 背包 | B/D |
| 可重复选择 | 完全背包 | T302 完全背包 | B/D |
| 单点修改区间和 | 树状数组 | T401 树状数组 | D/E |
| 区间修改查询 | 线段树 | T402 线段树 | D/E |
| 最近更大/更小 | 单调栈 | T403 单调栈 | B/D |
| 滑动窗口极值 | 单调队列 | T404 单调队列 | B/D |
| 路径、配置分割 | split | T501 split | C |

| | | | |
|---|---|---|---|
| 长串匹配 | KMP | T502 KMP | D/E |
| 前缀查询 | Trie | T503 Trie | D/E |
| 分组、分页、负数取模 | 整除取整 | T510 整除取整和标准取模 | A/B |
| 快速幂、取模幂 | 数学 | T511 快速幂 | B/D |
| 素数、约数、因子 | 数学 | T512 素数约数 | B/D |
| 进制转换、编码 | 数字处理 | T513 进制转换 | A/B/C |
| 日期、时间、周期 | 日期处理 | T514 日期处理 | A/C |
| 长题面、多命令 | 大模拟 | T601 大模拟骨架 | C |
| 每行操作名 | 命令分发 | T602 命令分发 | C |
| 多状态对象 | 状态机 | T603 状态机 | C |
| 动态删除仍取最优 | 懒删除堆 | T604 懒删除优先队列 | C/D |
| 最近最少使用 | LRU | T605 LRU 缓存 | C |
| 目录路径 | 路径规范化 | T606 路径规范化 | C |
| URL 参数 | URL 解析 | T607 URL 参数解析 | C |
| 括号合法性 | 栈 | T608 括号匹配 | C |
| 简单表达式 | 扫描求值 | T609 表达式入门 | C |

## 3.10 操作模式判断树

### 3.10.1 只有查询，没有修改

| 查询内容 | 优先算法 | 模板 |
|---|---|---|
| 区间和、区间数量 | 前缀和 / 前缀计数 | T101 |
| 子矩阵和、邻域平均 | 二维前缀和 | T102 |
| 最大最小，静态 | 排序 / 预处理 / 单调结构 | T105 / T403 |
| 图上单源距离 | BFS / Dijkstra | T201 / T204 |
| 任意两点距离，小图 | Floyd | T206 |
| 字符串是否出现 | find / KMP | T502 |
| 子树信息，静态 | DFS 序 + 前缀 / 树状数组 | T208 / T401 |

### 3.10.2 有修改，有查询

| 操作形态 | 优先算法 | 模板 |
|---|---|---|
| 单点修改，区间和查询 | 树状数组 | T401 |
| 区间修改，单点查询 | 差分思想 / 树状数组 | T103 / T401 |
| 区间修改，区间查询 | 线段树 | T402 |
| 动态插入删除，查最大最小 | set / 堆 / 懒删除 | T604 |
| 修改很少 | 每次重算 | 部分分策略 |
| 查询很少 | 每次暴力 | 部分分策略 |
| 操作可以离线 | 排序事件、差分、并查集倒序 | T110 / T103 / T203 |

### 3.10.3 多轮模拟

| 模拟形态 | 判断重点 | 模板 |
|---|---|---|
| 每轮所有对象同时变化 | 先存变化量，再统一更新 | T601 / T603 |
| 事件按时间发生 | 先排序，时间相同按题目优先级 | T110 |
| 状态会过期 | 每次操作前处理过期，或懒处理 | T603 / T604 |
| 每次选最优对象 | priority_queue / set | T604 |
| 命令种类多 | 每种命令写独立函数 | T602 |

## 3.11 A 题快速判断树

A 题目标是稳。判断时按下面顺序，不要一开始就想复杂算法。

1. 是否只要求一个数？优先一遍扫描、公式、计数。
2. 是否有排序、排名、最大最小？用 `sort` 后处理。
3. 是否有字符或字符串？按字符扫描，先别写复杂解析。
4. 是否有二维数组？先画 $2 \times 3$ 小例子确认下标。
5. 是否有日期、时间、进制、取整、小数？翻对应模板，不临场硬想语法。

| 题面信号 | 做法 | 模板/章节 |
|---|---|---|
| 求和、计数、最大最小 | 一遍扫描 | A 题统计 |
| 出现次数、直方图 | 计数数组 / map | T106 |
| 小数、百分比、平均、方差 | 公式 + `setprecision` | A 题公式 |
| 分组、分页、向上取整、负数取模 | 整数公式，不用浮点硬凑 | T510 |
| 排序、排名、中位数 | sort | T105 |
| 字符串、字符、校验码 | string 扫描 | T501 |
| 进制、数位、编码 | 进制转换或取模拆位 | T513 |
| 日期、星期、红绿灯周期 | 闰年、月份、取模 | T514 |
| 坐标距离、矩形面积 | 公式，注意边界 | A 题坐标 |
| 二维矩阵变换 | 写原坐标到新坐标公式 | A 题二维表格 |

### 3.11.1 A 题详细叶子表

| 叶子 | 识别信号 | 先写什么 | 易错点 | 翻到哪里 |
|---|---|---|---|---|
| A1 纯统计 | 求总和、数量、最大最小 | 一遍扫描 | 初值、long long | 第 5 章统计 |
| A2 频率统计 | 出现次数、直方图 | 数组计数或 map | 值域是否能开数组 | T106 |
| A3 排序输出 | 第几大、排名、中位数 | sort 后取值 | 下标从 0 还是 1 | T105 |
| A4 公式计算 | 平均、方差、加权、现值、分组 | 按公式翻译 | 整数除法、取整、小数位 | 第 5 章公式 / T510 |

| | | | |
|---|---|---|---|
| A5 字符处理 | 校验码、字符类别、string 扫描字符串计数 | getline 和 cin 混用 | T501 |
| A6 数位/进制 | 数位和、编码、进制分解 | 取模除法 | 0、负数、long long T513 |
| A7 日期时间 | 星期、天数、周期灯 | 月份表 + 闰年 | 闭区间、取模 T514 |
| A8 二维表格 | 图像、矩阵、邻居平均 | 画小矩阵找下标 | 行列反、边界 第 5 章二维 |
| A9 坐标几何 | 距离、矩形面积、点位置 | 套公式 | 交集为负取 0 第 5 章坐标 |
| A10 简单状态 | 连续得分、开关、当前状态 | 变量记录状态 | 状态重置时机 第 5 章模拟 |

## 3.12 B 题快速判断树

B 题目标是尽量拿满。它常常不是“高级算法”，而是把暴力换成预处理。

1. 暴力做法是什么？每次是否重复算同一段、同一块、同一类？
2. 重复算区间：前缀和。重复改区间：差分。
3. 重复查存在：set/map/hash。
4. 排序后规律明显：排序扫描、双指针、二分。
5. 有最小时间/最大阈值：先问答案是否单调。

| 题面信号 | 做法 | 模板 |
|---|---|---|
| 多次区间/矩阵查询 | 前缀和 | T101 / T102 |
| 区间修改最后输出 | 差分 | T103 / T104 |
| 频率统计、查重、点集匹配 | 哈希 / map / set | T106 |
| 排序后扫描、阈值统计 | sort + 扫描 | T105 |
| 两个有序序列匹配 | 双指针 | T108 |
| 答案具有单调性 | 二分答案 | T107 |
| 最短步数、扩散 | BFS | T201 |
| 连通关系、合并集合 | 并查集 | T203 |
| 简单依赖、任务计划 | 拓扑排序 | T205 |
| 背包信号、至少达到某值 | DP | T301 / T302 |
| 质因数、约数、幂 | 数学模板 | T511 / T512 |

### 3.12.1 B 题详细叶子表

| 叶子 | 暴力形态 | 优化判断 | 模板 | 保底 |
|---|---|---|---|---|
| B1 一维前缀 | 每次扫区间 | 只查不改，多次区间和 | T101 | 暴力扫小数据 |
| B2 二维前缀 | 每次扫矩形 | 矩阵静态，多次子矩阵 | T102 | 暴力双循环 |

| | | | | |
|---|---|---|---|---|
| B3 一维差分 | 每次区间逐个加 | 区间修改，最后输出/查询少 | T103 | 直接修改小数据 |
| B4 二维差分 | 每次矩形逐格加 | 矩形覆盖，最后统计 | T104 | 二维暴力 |
| B5 哈希统计 | 每次线性找存在 | 查重、频率、坐标点存在 | T106 | map 先写稳 |
| B6 排序扫描 | 枚举每个阈值再统计 | 排序后相邻段贡献可更新 | T105/T101 | 枚举阈值小数据 |
| B7 双指针 | 双重循环匹配 | 两序列有序或可排序 | T108 | map 匹配 |
| B8 二分答案 | 枚举答案 | check 单调,答案范围大 | T107 | 枚举小范围 |
| B9 简单 BFS | 枚举路径 | 无权最短步数/扩散 | T201 | DFS 小图 |
| B10 并查集 | 每次 DFS 判连通 | 只有合并和查连通 | T203 | 邻接表 DFS |
| B11 拓扑 | 反复扫描找可做任务 | 前置依赖固定 | T205 | 小 n 反复扫描 |
| B12 背包 | 枚举所有选择 | 选或不选，容量不太大 | T301 | DFS 子集 |
| B13 基础数学 | 枚举因子到 n | 质因数、幂、约数 | T511/T512 | 小范围试除 |
| B14 矩阵优化 | 按题意多重循环超时 | 调整乘法/扫描顺序 | 第 6 章矩阵 | 暴力部分分 |
| B15 时间区间 | 每分钟逐个标记 | 时间范围不大或可差分 | T103/T110 | 数组标记小范围 |

## 3.13 C 题快速判断树

C 题最怕一边读一边写。正确流程是先拆对象、状态、命令、规则。

### 3.13.1 C 题读题分解

| 要找什么 | 在题面中怎么标记 | 代码落点 |
|---|---|---|
| 对象 | 用户、文件、节点、任务、请求、设备、词条 | struct |
| 状态 | 空闲/占用、有效/过期、允许/拒绝、在线/离线 | enum 或 int |
| 命令 | create、delete、query、request、release | T602 命令分发 |
| 时间 | 到期、当前时刻、持续时间、周期 | T110、T514 |
| 规则优先级 | 先判断 A，再判断 B；相同时间谁优先 | 独立函数，写注释 |
| 输出 | 每次命令输出、最后统一输出、格式化文本 | 先写输出样例函数 |

### 3.13.2 C 题模板选择

| 题面信号 | 做法 | 模板 |
|---|---|---|
| 长题面、多对象、多规则 | 大模拟拆解 | T601 |

| | | |
|---|---|---|
| 每行一个命令 | 命令分发 | T602 |
| 对象状态变化 | 状态机 | T603 |
| 按时间发生 | 事件排序 | T110 |
| 每次取最优任务 | 优先队列 / 懒删除 | T604 |
| 路径、URL、配置 | 字符串解析 | T501 / T606 / T607 |
| 括号、表达式、方程式 | 栈 / 递归解析 / 扫描 | T608 / T609 |
| 缓存淘汰 | list + map 或队列 | T605 |
| 地图带状态 | 状态 BFS | C 题章节 |
| 树或目录结构 | DFS、父子 map、路径查找 | T606 / T208 |

### 3.13.3 C 题详细叶子表

| 叶子 | 题面信号 | 代码结构 | 易错点 | 模板 |
|---|---|---|---|---|
| C1 命令分发 | 每行操作名不同 | 主循环读 op, 分函数处理 | 参数数量不同 | T602 |
| C2 状态机 | 对象有状态变化 | 状态字段 + 转移函数 | 同一操作多次转移 | T603 |
| C3 事件排序 | 时间、到期、归还、同时发生 | Event 结构体排序 | 同时事件优先级 | T110 |
| C4 优先队列 | 每次取最优任务/对象 | heap 或 set | 失效元素删除 | T604 |
| C5 路径目录 | 文件、目录、父子、配额 | 树节点 + path map | 创建失败回滚 | T606 |
| C6 URL/参数 | 路径、查询参数、匹配规则 | split + 参数 map | 空参数、类型检查 | T607 |
| C7 JSON/配置 | 嵌套键值、查询路径 | 栈/递归建结构 | 转义和层级 | T501 |
| C8 Markdown/文本 | 多行文本、块、行内格式 | 先分块再处理行 | 空行分块 | T601 |
| C9 表达式/括号 | 运算、括号、化学式 | 栈或递归解析 | 优先级、嵌套倍数 | T608/T609 |
| C10 编码解码 | 十六进制、压缩、校验 | 指针扫描字符串 | 长度字段越界 | T513 |
| C11 矩阵图像 | 固定矩阵、扫描顺序、变换 | 二维数组 + 顺序表 | 行列、取整 | T102 |
| C12 权限规则 | 用户、角色、资源、权限 | set/map 规则判断 | 通配符、最大等级 | T603 |
| C13 带状态 BFS | 地图、钥匙、方向、时间 | visited 包含状态 | 状态维度漏掉 | T201 |
| C14 树/目录模拟 | 子树、删除、查询、合并 | DFS + 父子关系 | 删除后引用失效 | T208 |
| C15 分层拿分 | 规则太多写不完 | 先实现核心命令 | 输出格式别乱 | T601 |

## 3.14 逐句读题分支树

考场读题时，建议拿笔在题面旁边标记下面六类词。标完后再决定模板。

| 标记 | 看到什么就圈出来 | 后续判断 |
| --- | --- | --- |
| 对象 | 学生、用户、文件、节点、仓库、服务器、任务、词条 | 是否需要 struct？是否有编号到对象的 map？ |
| 属性 | 权重、时间、状态、坐标、容量、权限、父节点 | 是否要排序、比较器、状态机？ |
| 动作 | 查询、修改、创建、删除、移动、合并、授权、到期 | 是否是命令分发或数据结构？ |
| 范围 | $n$、$m$、$q$、值域、字符串总长、矩阵大小 | 直接排除不可能复杂度。 |
| 关系 | 相邻、包含、依赖、祖先、连通、路径、前缀 | 图、树、拓扑、字符串、区间？ |
| 目标 | 最大、最小、数量、是否存在、方案数、输出过程 | 统计、二分、DP、搜索还是模拟？ |

### 3.14.1 从句子到模板

| 题面句子模式 | 立即追问 | 模板方向 |
| --- | --- | --- |
| “给定 $q$ 次询问” | 每次询问是否只读不改？ | 前缀和 / 数据结构 |
| “每次修改后回答” | 修改是点、区间还是对象状态？ | 树状数组 / 线段树 / 状态机 |
| “求满足条件的数量” | 条件能否拆成前缀、排序或哈希？ | 计数 / 排序扫描 / map |
| “求最小的 $x$ 使得……” | $x$ 越大是否越容易满足？ | 二分答案 |
| “选择若干个” | 每个物品选一次还是多次？ | 01 背包 / 完全背包 |
| “从 A 到 B 的最短……” | 边权是否全为 1？ | BFS / Dijkstra |
| “任务必须在前置任务后” | 是否有环？ | 拓扑排序 |
| “字符串由若干规则组成” | 是否有括号或嵌套？ | split / 栈 / 递归解析 |
| “路径包含 . 和 ..” | 是否需要规范化？ | T606 |
| “状态在某时刻过期” | 过期是在操作前还是操作后处理？ | 状态机 / 事件模拟 |

## 3.15 B 题专项分支树

B 题常见陷阱是“暴力能写但会超时”。先写出暴力，再问它重复算了什么。

### 3.15.1 B 题从暴力到优化

| 暴力形态 | 重复在哪里 | 优化方向 |
| --- | --- | --- |
| 每次询问都扫一段数组 | 区间和/数量重复计算 | 一维前缀和 T101 |

| | | |
|---|---|---|
| 每次询问都扫一个矩形 | 子矩阵重复计算 | 二维前缀和 T102 |
| 每次区间加都逐个改 | 区间修改重复 | 差分 T103/T104 |
| 每次判断点是否存在都扫所有点 | 存在性查询重复 | set / unordered_set |
| 每次找匹配点都双重循环 | 坐标或编号匹配重复 | map / 双指针 / 排序 |
| 每个候选答案都完整模拟 | 答案范围大 | 二分答案 T107 |
| 每次重新排序 | 只需维护极值或有序集合 | priority_queue / set |
| 每次从头判断依赖任务 | 依赖结构固定 | 拓扑排序 T205 |

### 3.15.2 B 题二分答案确认表

| 问题 | 是 | 否 |
|---|---|---|
| 答案是一个整数或实数吗? | 继续问单调性 | 多半不是二分答案。 |
| 给定答案 $x$，能快速判断可不可行吗? | 写 `check(x)` | 不适合二分。 |
| $x$ 变大后，可行性只朝一个方向变化吗? | 可以二分 | 不能二分。 |
| 题目问最小时间/最大阈值/最少资源吗? | 优先尝试二分 | 可能是 DP/贪心。 |

## 3.16 C 题专项分支树

C 题不是先想算法，而是先建模。建模错误比模板不会更致命。

### 3.16.1 C 题对象建模表

| 题面对象 | 常见字段 | 容器选择 |
|---|---|---|
| 用户 / 角色 / 权限 | 名称、权限集合、等级、资源类型 | `map<string, Obj>`, `set<string>` |
| 文件 / 目录 | 名称、父节点、子节点、大小、配额 | map 路径到节点，或树节点数组 |
| 服务器 / 节点 | 编号、算力、任务数、可用状态 | vector + set/priority_queue |
| 任务 / 请求 | 时间、优先级、资源需求、状态 | struct + sort/heap |
| 词条 / token | 字符串、权重、出现位置 | map/unordered_map + heap |
| 地图状态 | 坐标、方向、钥匙、时间、能量 | queue + visited 多维数组/map |

### 3.16.2 C 题规则优先级表

| 题面出现 | 必须单独确认 | 常见错误 |
|---|---|---|

| | | |
|---|---|---|
| “若同时发生” | 同时事件的排序规则 | 先后顺序写反。 |
| “如果不存在则……” | 缺省值或失败输出 | map 自动创建误判存在。 |
| “到期” | 到期时刻是否还有效 | $<$ 和 $\leq$ 写错。 |
| “优先选择” | 比较器所有关键字 | 忘记编号升序等最后规则。 |
| “忽略非法操作” | 是否改变状态、是否输出 | 非法时仍修改了数据。 |
| “回滚/失败” | 前面临时改动要撤销 | 配额、路径创建类常见。 |

### 3.16.3 C 题字符串解析分支

| 字符串形态 | 判断 | 模板 |
|---|---|---|
| 固定分隔符，无嵌套 | split 即可 | T501 |
| 路径，含 `.`、`..`、多个斜杠 | 栈式规范化 | T606 |
| URL，含路径和查询参数 | 先按 `?`，再按 `&` | T607 |
| 括号嵌套 | 栈或递归 | T608 |
| 有运算优先级 | 两栈或递归下降 | T609 |
| 化学式/表达式含倍数 | 递归解析返回 map | C 题表达式扩展 |
| 多行文本块 | 先分块，再行内解析 | T601 |

## 3.17 D/E 题快速判断树

D/E 题不要先问“我会不会满分”，先问“我能不能拿部分分”。

1. 找子任务数据范围：$n \leq 20$、链、树、无修改、边权 1、查询少。
2. 写最朴素暴力，估算它能过哪档。
3. 如果暴力只有 0 分，再找一个最接近的模板：BFS、Dijkstra、Kruskal、树状数组、线段树、DP。
4. 优先实现你能调通的版本，不把时间全部压在复杂满分做法上。

| 题面信号 | 满分方向 | 部分分方向 |
|---|---|---|
| $n \leq 20$ 子任务 | 状压 / DFS | 枚举子集 |
| 树上路径 | LCA / 树上差分 | 每次 DFS 找路径 |
| 区间修改查询 | 线段树 | 暴力 / 树状数组 / 差分 |
| 最小生成代价 | Kruskal | 小图枚举或直接 Kruskal |
| 字符串匹配 | KMP / Trie | find / 暴力匹配 |
| 正权最短路 | Dijkstra | 小图 Floyd / 无权 BFS |
| 最大化最小值 | 二分答案 | 枚举答案小范围 |
| 复杂 DP | 优化 DP | DFS / 小状态 DP |
| 动态集合取最值 | 平衡树 / 堆懒删除 | 每次排序或线性扫描 |

## 3.18 D/E 部分分决策树

下面这棵树只服务一件事：不会满分时，尽快产出能提交的代码。

1. 先看子任务表，有没有 $n \le 20$、$n \le 100$、链、树、无修改、边权 1、查询少。
2. 如果有小 $n$：优先 DFS、枚举、Floyd、$O(n^2)$ DP。
3. 如果有特殊图：边权 1 用 BFS，树用 DFS，DAG 用拓扑。
4. 如果有特殊操作：无修改用前缀，有修改少就每次重算，查询少就每次暴力。
5. 如果只有大数据：选择你最熟的相近模板，写 30 到 50 分版本。

| 子任务/特殊性质 | 可写版本 | 常用模板 |
|---|---|---|
| $n \le 20$ | 枚举子集、DFS、状压 | 状压 DP、搜索 |
| $n \le 100$ | Floyd、$O(n^3)$ DP、暴力查询 | T206 |
| 树退化成链 | 数组/前缀处理路径 | T101 / T103 |
| 没有修改 | 预处理后回答查询 | 前缀、DFS 序 |
| 查询次数很少 | 每次 DFS/BFS/扫描 | 暴力部分分 |
| 边权全为 1 | BFS 替代 Dijkstra | T201 |
| 图是 DAG | 拓扑 DP | T205 |
| 值域小 | 计数数组/桶 | T106 |
| 参数 $k$ 很小 | 状态压缩或按 $k$ 枚举 | 状压入口 |

### 3.18.1 D/E 题详细叶子表

| 叶子 | 识别信号 | 满分方向 | 部分分写法 | 翻到哪里 |
|---|---|---|---|---|
| DE1 小 n | $n \le 20$ 或参数小 | 状压/搜索优化 | 枚举子集、DFS | 第 8 章状压 |
| DE2 小图 | $n \le 100$ | Floyd/DP | Floyd 或暴力建图 | T206 |
| DE3 树路径 | 树上多次路径 | LCA/树剖/树差分 | 每次 DFS 找路径 | T208 |
| DE4 子树查询 | 子树加、子树查 | DFS 序 + 树状数组/线段树 | 每次遍历子树 | T401/T402 |
| DE5 动态区间 | 区间改查交替 | 线段树 | 小范围数组模拟 | T402 |
| DE6 动态集合 | 插入删除查极值 | set/堆懒删除 | 每次排序/扫描 | T604 |
| DE7 最短路 | 正权图路径 | Dijkstra/建图优化 | Floyd 小图、BFS 边权 1 | T204 |
| DE8 最小生成 | 连通最小代价 | Kruskal/Prim | 直接 Kruskal | T207 |
| DE9 字符串匹配 | 长串、多模式、前缀 | KMP/Trie/AC 自动机 | find 暴力 | T502/T503 |
| DE10 复杂 DP | 最优方案数、限制多 | 状态设计/优化 | 记忆化 DFS 小数据 | 第 8 章 DP |
| DE11 数学计数 | 组合、取模、约数 | 预处理/数论 | 枚举小范围 | T511/T512 |
| DE12 离线处理 | 操作可重排、查询很多 | 排序/分治/离线 DSU | 按输入暴力 | T110/T203 |
| DE13 几何 | 点线面、距离、交叉 | 计算几何 | 小数据枚举 | 后续补强 |
| DE14 网络流/匹配 | 分配、匹配、容量流 | 网络流/二分图 | 贪心或小图 DFS | 放弃满分 |

## 3.19 容易误判的分支

| 看起来像 | 但如果题面出现 | 应改判为 |
|---|---|---|
| 简单模拟 | $q$ 很大，每次都查区间 | 前缀和 / 差分 / 数据结构 |
| 排序题 | 要动态插入删除后仍排名 | set、树状数组、线段树 |
| 字符串题 | 有括号嵌套、表达式优先级 | 栈或递归下降 |
| 图搜索题 | 边权不是 1 | Dijkstra，不是普通 BFS |
| BFS 题 | 状态包含钥匙、方向、时间、能量 | 状态 BFS |
| 贪心题 | 选或不选、容量限制明显 | 背包 DP |
| 二分题 | check 不单调 | 不能二分，改 DP/贪心/扫描 |
| 树题 | 只有父子目录和路径字符串 | 可能是大模拟，不一定是 LCA |
| 数学题 | 有多次询问和大范围 | 预处理、筛法、快速幂 |

## 3.20 无法判断时的保底流程

如果读完题仍不知道算法：

1. 写出最暴力做法。
2. 估算暴力复杂度。
3. 看数据范围中有没有小数据能过。
4. 找特殊性质：无修改、链、树、边权 1、参数小。
5. 先实现能过样例和小数据的版本。
6. 再尝试从本章关键词表找优化方向。

## 3.21 判断树反向测试表

刷完一道题后，用这张表检查第 3 章是否真的帮你判断。如果某行填不上，就说明手册还要继续补。

| 问题 | 例子 | 应能定位到 |
|---|---|---|
| 题面主体是什么? | 数组、矩阵、图、字符串、对象、事件 | 题面结构第一判断 |
| 是否有多次操作? | $q$ 次查询、每次修改、每次输出 | 操作模式判断树 |
| 暴力复杂度是多少? | $O(nq)$、$O(n^2)$、$O(nm)$ | 数据范围判断树 |
| 关键词是否误导? | 看起来像模拟但 $q$ 很大 | 容易误判分支 |
| 满分不会时能拿什么? | 小数据、特殊性质、暴力 | D/E 部分分决策树 |
| 对应模板编号是什么? | T101、T107、T603 等 | 关键词到模板总索引 |

## 3.22 最后 5 分钟判断

| 情况 | 处理 |
| --- | --- |
| A/B 还没过样例 | 放下后面题，先修 A/B。 |
| C 写了一半但样例不过 | 注释掉高风险优化，保留最基础模拟。 |
| D/E 没思路 | 找最小子任务，写暴力或特殊性质。 |
| 数组越界风险高 | 全部下标检查一遍，必要时数组开大 5 到 10。 |
| 输出格式不确定 | 对照样例逐字符检查空格、换行、小数位。 |

# 第二部分

# 基础与题型

# 第四章　C++ 语法与 STL 速查

## 4.1　本章使用方式

本章不是系统 C++ 教材，而是 CSP 考场写代码时最常用的语法清单。目标是让你在不熟练时也能快速确认：

- 一个程序应该怎么开头。
- 输入输出怎么写。
- 什么时候必须用 `long long`。
- 整数除法、取模、向上取整、向下取整怎么写。
- 数组、`vector`、`string`、结构体怎么用。
- `sort`、`map`、`set`、`queue`、`priority_queue` 怎么写。
- lambda 比较器怎么写，常见错误是什么。

**考场原则：**优先使用自己练过的写法。会写清楚比写得短更重要。

## 4.2　最小程序骨架

大部分 CSP 题目都可以从这个骨架开始：

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

using ll = long long;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    // write code here

    return 0;
}
```

说明：

- `#include <bits/stdc++.h>` 在 g++ 中常用，包含大多数标准库。正式考试前确认环境支持。
- `using namespace std;` 可以少写 `std::`。

- `using ll = long long;` 以后可以用 `ll` 表示 `long long`。
- `ios::sync_with_stdio(false); cin.tie(nullptr);` 用来加速 C++ 输入输出。
- 使用上面两行加速后，不要混用 `scanf/printf` 和 `cin/cout`。

## 4.3 输入输出

### 4.3.1 普通输入

```cpp
int n;
cin >> n;

int a, b;
cin >> a >> b;

string s;
cin >> s; // reads until whitespace
```

### 4.3.2 读入数组

```cpp
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++) {
    cin >> a[i];
}
```

如果题目下标从 1 开始，考场上可以直接开成 `n + 1`：

```cpp
int n;
cin >> n;
vector<int> a(n + 1);
for (int i = 1; i <= n; i++) {
    cin >> a[i];
}
```

### 4.3.3 多组数据

题目明确给出 $T$：

```cpp
int T;
cin >> T;
while (T--) {
    solve();
}
```

题目没有给 $T$，读到文件结束：

```cpp
int a, b;
while (cin >> a >> b) {
    cout << a + b << '\n';
}
```

**多组数据常见坑：**每组开始前要清空数组、图、队列、map、答案变量。

### 4.3.4 整行输入

`cin >> s` 只能读到空格前。如果要读整行，用 `getline`：

```cpp
string line;
getline(cin, line);
```

如果前面刚用了 `cin >> n`，再用 `getline`，要先吃掉换行：

```cpp
int n;
cin >> n;
cin.ignore();

string line;
getline(cin, line);
```

如果担心行尾有多个字符，可以写：

```cpp
cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

### 4.3.5 输出

```cpp
cout << ans << '\n';
cout << a << ' ' << b << '\n';
```

**不要输出提示语。**例如不要写 `cout << "answer=" << ans;`，除非题目明确要求。

### 4.3.6 小数输出

```cpp
double x = 3.1415926;
cout << fixed << setprecision(2) << x << '\n'; // 3.14
```

说明：

- `fixed` 表示按小数位输出。
- `setprecision(2)` 表示保留 2 位小数。
- 需要包含 `<iomanip>`，但 `bits/stdc++.h` 已包含。

## 4.4 数据类型

### 4.4.1 整数类型

| 类型 | 大约范围 | 使用场景 |
| --- | --- | --- |
| `int` | $-2.1\times10^9$ 到 $2.1\times10^9$ | 下标、普通计数、小范围整数。 |
| `long long` | $-9.2\times10^{18}$ 到 $9.2\times10^{18}$ | 求和、乘法、答案、距离、方案数。 |
| `unsigned long long` | 0 到约 $1.8\times10^{19}$ | 很少必须用，新手慎用。 |

必须用 `long long` 的常见情况：

- $n$ 个数求和，每个数可达 $10^9$。
- 两个最大可达 $10^9$ 的数相乘。
- 最短路边权和可能很大。
- 方案数、组合数、累计时间、累计代价。

乘法防溢出：

```
int a = 1000000000;
int b = 1000000000;
long long x = 1LL * a * b;
```

**错误写法：**

```
long long x = a * b; // a*b already overflowed as int
```

### 4.4.2 浮点类型

- `double` 是常用浮点数。
- 判断浮点相等时不要直接用 `==`，通常用误差。

```
const double EPS = 1e-9;
if (abs(a - b) < EPS) {
    // a and b are almost equal
}
```

### 4.4.3 字符和布尔

```
char c = 'A';
bool ok = true;
```

字符判断：

```
isdigit(c); // is digit
isalpha(c); // is letter
islower(c);
isupper(c);
tolower(c);
toupper(c);
```

**注意：** `isdigit(c)` 返回非零表示真，不一定返回 1。

## 4.5 表达式、整除和取整

### 4.5.1 基本运算符

C++ 中常用算术运算符如下：

| 运算符 | 含义 | 注意点 |
| --- | --- | --- |
| `+ - *` | 加、减、乘 | 两个 `int` 相乘仍先按 `int` 计算。 |
| `/` | 除法 | 两边都是整数时做整数除法，结果向 0 截断。 |
| `%` | 取余 | 只能用于整数；负数取余的结果可能为负。 |
| `+= -= *= /= %=` | 复合赋值 | `x += y` 等价于 `x = x + y`。 |
| `++ --` | 自增、自减 | 简单循环中优先写 `i++` 或 `++i`，不要嵌进复杂表达式。 |

**整数除法不是数学里的向下取整。**在 C++ 中，`-3 / 2` 的结果是 `-1`，因为整数除法向 0 截断；数学意义上的 $\lfloor -3/2 \rfloor$ 是 $-2$。

### 4.5.2 整数除法和类型转换

如果两边都是整数，`/` 会丢掉小数部分：

```
int a = 5, b = 2;
cout << a / b << '\n'; // 2
```

如果要小数结果，至少把一个操作数转成浮点数：

```
cout << 1.0 * a / b << '\n'; // 2.5
cout << (double)a / b << '\n'; // 2.5
```

如果题目要求整数答案，不要为了方便乱转 `double`。大整数转浮点后可能丢精度。

### 4.5.3 正整数向上取整

最常见场景：$a$ 个物品，每组最多放 $b$ 个，问需要几组。若 $a \geq 0, b > 0$：

```
long long ceil_div_positive(long long a, long long b) {
    return (a + b - 1) / b;
}
```

常见特例：

```
long long half_up = (n + 1) / 2; // ceil(n / 2), n >= 0
long long blocks = (n + block - 1) / block;
```

**这个简式只适合 $a \geq 0, b > 0$。**如果 $a$ 可能为负，或者 $b$ 可能为负，不能直接套 `(a+b-1)/b`。

### 4.5.4 有符号整数向下和向上取整

如果题目涉及负坐标、负偏移、时间差、数学分段，建议使用下面的通用函数。它们要求 `b != 0`，并且遵循数学意义上的 floor 和 ceil。

```
long long floor_div(long long a, long long b) {
    long long q = a / b;
    long long r = a % b;
    if (r != 0 && ((r > 0) != (b > 0))) q--;
    return q;
}

long long ceil_div(long long a, long long b) {
    long long q = a / b;
    long long r = a % b;
    if (r != 0 && ((r > 0) == (b > 0))) q++;
    return q;
}
```

快速自测：

| 表达式 | `floor_div` | `ceil_div` |
|---|---|---|
| $3/2$ | 1 | 2 |
| $-3/2$ | -2 | -1 |
| $3/-2$ | -2 | -1 |
| $-3/-2$ | 1 | 2 |

### 4.5.5 负数取模

C++ 中 `x % m` 的符号跟 `x` 相同。例如 `-1 % 5` 是 `-1`，不是 `4`。如果题目需要 0 到 $m-1$ 的标准余数，写：

```
long long norm_mod(long long x, long long m) {
    long long r = x % m;
    if (r < 0) r += m;
    return r;
}
```

适用：`m > 0`。若 `x` 可能小于 `-m` 很多，上面仍然正确，因为 C++ 的一次取余后 `r` 已经在 `(-m,m)` 内。

### 4.5.6 运算优先级和括号

考场上不建议背复杂优先级。以下规则足够覆盖大多数题：

- 乘、除、取模优先于加减。
- 比较运算优先级低于加减乘除。
- `&&` 优先于 `||`。

- 位运算和比较混在一起时必须加括号。

```
if ((mask & (1 << k)) != 0) {
    // bit k is 1
}
```

**有疑问就加括号。**括号不会让程序变慢，却能显著减少读错表达式的风险。

### 4.5.7 常量和初始化

常用写法：

```
const int N = 100000 + 5;
const long long INF = (long long)4e18;
const int MOD = 1000000007;

int cnt = 0;
long long sum = 0;
bool ok = true;
```

提醒：

- 局部普通变量不会自动清零，定义后要赋初值。
- 全局数组会默认清零，但多组数据不会自动恢复初始状态。
- const 常量不能被修改，适合写数组上限、模数、无穷大。

## 4.6 控制流

### 4.6.1 if

```
if (x > 0) {
    cout << "positive\n";
} else if (x == 0) {
    cout << "zero\n";
} else {
    cout << "negative\n";
}
```

### 4.6.2 for

```
for (int i = 0; i < n; i++) {
    // 0, 1, ..., n-1
}

for (int i = 1; i <= n; i++) {
    // 1, 2, ..., n
}
```

倒序：

```
for (int i = n - 1; i >= 0; i--) {
    // be careful when i is unsigned
}
```

### 4.6.3 while

```
while (condition) {
    // loop
}
```

读到结束：

```
int x;
while (cin >> x) {
    // process x
}
```

### 4.6.4 break 和 continue

```
for (int i = 0; i < n; i++) {
    if (a[i] < 0) continue; // skip this round
    if (a[i] == target) break; // exit loop
}
```

## 4.7 auto、引用和范围 for

### 4.7.1 auto

auto 让编译器根据右侧表达式推断类型，常用于迭代器、pair、复杂容器遍历。

```
auto it = lower_bound(a.begin(), a.end(), x);
auto p = make_pair(3, 5);
```

遍历 map：

```
map<string, int> cnt;
for (auto [key, value] : cnt) {
    cout << key << ' ' << value << '\n';
}
```

**注意复制。**auto [key, value] 会复制键和值；如果元素很大，写 const auto& 更合适。

```
for (const auto& item : records) {
    // read item without copying
}
```

### 4.7.2 引用

引用相当于变量的别名。函数传大对象时常用引用避免复制。

```
void printVector(const vector<int>& a) {
    for (int x : a) cout << x << ' ';
}

void addOne(vector<int>& a) {
    for (int& x : a) x++;
}
```

区别：

- `const vector<int>& a`：不复制，也不允许函数内修改。
- `vector<int>& a`：不复制，允许函数内修改原数组。
- `vector<int> a`：会复制整个数组，数据大时慢。

### 4.7.3 范围 for

只读遍历：

```
for (int x : a) {
    cout << x << '\n';
}
```

修改原元素：

```
for (int& x : a) {
    x++;
}
```

**如果忘了写引用，修改不会回到原数组。**

```
for (int x : a) {
    x++; // only modifies the copy
}
```

### 4.7.4 size 的类型

`a.size()` 的返回类型是无符号整数。考场上为了减少类型麻烦，常写成：

```
int n = (int)a.size();
for (int i = 0; i < n; i++) {
    // use a[i]
}
```

倒序遍历不要写下面这种：

```
for (int i = a.size() - 1; i >= 0; i--) {
    // dangerous when a is empty
}
```

更稳的写法：

```
for (int i = (int)a.size() - 1; i >= 0; i--) {
    // ok even when a is empty: i starts at -1
}
```

## 4.8 函数

把重复逻辑写成函数，能减少大模拟和算法题的错误。

```
int add(int a, int b) {
    return a + b;
}

bool isEven(int x) {
    return x % 2 == 0;
}
```

传 vector：

```
int sumVector(const vector<int>& a) {
    int sum = 0;
    for (int x : a) sum += x;
    return sum;
}
```

说明：

- `const vector<int>& a` 表示不复制整个数组，并且函数内部不修改它。
- 如果函数需要修改数组，去掉 `const`。

```
void addOne(vector<int>& a) {
    for (int& x : a) x++;
}
```

## 4.9 数组

### 4.9.1 普通数组

全局数组适合大数组：

```
const int N = 100000 + 5;
int a[N];
long long pre[N];
```

说明：

- 全局数组默认初始化为 0。
- 局部大数组可能爆栈，尽量放全局或用 `vector`。

### 4.9.2 二维数组

```
const int N = 1005;
int grid[N][N];
```

如果规模不确定：

```
int n, m;
cin >> n >> m;
vector<vector<int>> grid(n, vector<int>(m, 0));
```

**内存估算：** int a[10000][10000] 大约 400MB，通常危险。

### 4.9.3 初始化

全局普通数组默认是 0。如果要重置数组：

```
int a[1005];
memset(a, 0, sizeof a);
memset(a, -1, sizeof a);
```

**注意：** memset 适合把数组设为 0 或 -1，不适合把 int 数组设成 1、2、无穷大。

设置成大数：

```
const int INF = 0x3f3f3f3f;
int dist[1005];
memset(dist, 0x3f, sizeof dist);
```

vector 初始化：

```
vector<int> a(n, 0);
vector<long long> dist(n + 1, (long long)4e18);
vector<vector<int>> grid(n, vector<int>(m, 0));
```

## 4.10 vector

### 4.10.1 创建

```
vector<int> a; // empty
vector<int> b(10); // 10 zeros
vector<int> c(10, -1); // 10 values, all -1
```

### 4.10.2 常用操作

```
a.push_back(5);
a.pop_back();
a.clear();
int n = (int)a.size();
bool empty = a.empty();
```

访问：

```
cout << a[0] << '\n';
cout << a.back() << '\n';
```

**访问前检查非空。**a.back()、a[0] 在空数组上会出错。

### 4.10.3 遍历

```
for (int i = 0; i < (int)a.size(); i++) {
    cout << a[i] << '\n';
}

for (int x : a) {
    cout << x << '\n';
}

for (int& x : a) {
    x++; // modify original element
}
```

### 4.10.4 删除元素

删除最后一个：

```
a.pop_back();
```

删除某个位置：

```
a.erase(a.begin() + pos);
```

**注意：**erase 是 $O(n)$，大量删除时可能超时。

删除所有等于 x 的元素：

```
a.erase(remove(a.begin(), a.end(), x), a.end());
```

### 4.10.5 二维 vector

二维网格：

```
int n, m;
cin >> n >> m;
vector<vector<int>> a(n, vector<int>(m));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        cin >> a[i][j];
    }
}
```

二维字符图：

```cpp
int n, m;
cin >> n >> m;
vector<string> g(n);
for (int i = 0; i < n; i++) {
    cin >> g[i];
}
cout << g[0][0] << '\n';
```

图的邻接表：

```cpp
int n, m;
cin >> n >> m;
vector<vector<int>> g(n + 1);
for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
}
```

## 4.11 string

### 4.11.1 基本操作

```cpp
string s = "abc";
int n = (int)s.size();
char c = s[0];
s.push_back('d');
s += "ef";
```

### 4.11.2 子串

```cpp
string s = "abcdef";
string t = s.substr(2, 3); // "cde"
```

`substr(pos, len)` 从 `pos` 开始取 `len` 个字符，下标从 0 开始。

### 4.11.3 查找

```cpp
string s = "abcdef";
if (s.find("cd") != string::npos) {
    cout << "found\n";
}
```

**不要写：**

```cpp
if (s.find("cd")) {
    // wrong when position is 0
}
```

### 4.11.4 字符串转数字

```cpp
int x = stoi("123");
long long y = stoll("1234567890123");
double z = stod("3.14");
```

数字转字符串：

```cpp
string s = to_string(123);
```

**如果数字很长，**`stoi/stoll` 会溢出或异常，要用字符串模拟。

### 4.11.5 按空格分割

```cpp
string line;
getline(cin, line);
stringstream ss(line);

string word;
while (ss >> word) {
    cout << word << '\n';
}
```

## 4.12 pair 和 tuple

### 4.12.1 pair

```cpp
pair<int, int> p = {3, 5};
cout << p.first << ' ' << p.second << '\n';
```

常用于坐标、边、距离和编号：

```cpp
vector<pair<int, int>> points;
points.push_back({x, y});
```

默认排序：

```cpp
sort(points.begin(), points.end());
```

默认先按 `first` 升序，再按 `second` 升序。

### 4.12.2 tuple

字段超过两个时可以用 tuple，但新手更推荐结构体。

```
tuple<int, int, int> t = {1, 2, 3};
auto [a, b, c] = t;
```

## 4.13 结构体 struct

结构体适合表示学生、任务、边、事件、状态。

```
struct Student {
    int id;
    string name;
    int score;
};

Student s;
s.id = 1;
s.name = "Alice";
s.score = 90;
```

数组：

```
vector<Student> students;
students.push_back({1, "Alice", 90});
```

图的边：

```
struct Edge {
    int to;
    int w;
};

vector<vector<Edge>> g(n + 1);
g[u].push_back({v, w});
```

## 4.14 排序 sort

### 4.14.1 普通排序

```
sort(a.begin(), a.end()); // ascending
sort(a.rbegin(), a.rend()); // descending
```

数组排序：

```
sort(a, a + n);
```

### 4.14.2 自定义排序：从大到小

```
sort(a.begin(), a.end(), [](int x, int y) {
    return x > y;
});
```

### 4.14.3 结构体排序

成绩高的在前；成绩相同，编号小的在前：

```
struct Student {
    int id;
    int score;
};

vector<Student> v;

sort(v.begin(), v.end(), [](const Student& a, const Student& b) {
    if (a.score != b.score) return a.score > b.score;
    return a.id < b.id;
});
```

### 4.14.4 按 vector 的某一列排序

```
vector<vector<int>> a;

sort(a.begin(), a.end(), [](const vector<int>& x, const vector<int>& y) {
    return x[0] < y[0]; // sort by column 0 ascending
});
```

先按第 0 列升序，再按第 1 列降序：

```
sort(a.begin(), a.end(), [](const vector<int>& x, const vector<int>& y) {
    if (x[0] != y[0]) return x[0] < y[0];
    return x[1] > y[1];
});
```

### 4.14.5 stable_sort

如果题目要求相同关键字保持原输入顺序：

```
stable_sort(v.begin(), v.end(), [](const Student& a, const Student& b) {
    return a.score > b.score;
});
```

### 4.14.6 比较器高危错误

比较器必须表达“谁应该排在前面”。

**正确：**

```
sort(v.begin(), v.end(), [](const Student& a, const Student& b) {
    if (a.score != b.score) return a.score > b.score;
    return a.id < b.id;
});
```

**错误：**

```
sort(v.begin(), v.end(), [](const Student& a, const Student& b) {
    return a.score >= b.score; // wrong: equality should return false
});
```

相等时返回 `true` 可能导致排序行为错误。

## 4.15 二分相关函数

### 4.15.1 binary_search

```
sort(a.begin(), a.end());
bool ok = binary_search(a.begin(), a.end(), x);
```

### 4.15.2 lower_bound 和 upper_bound

```
sort(a.begin(), a.end());

auto it1 = lower_bound(a.begin(), a.end(), x); // first >= x
auto it2 = upper_bound(a.begin(), a.end(), x); // first > x
```

下标：

```
int pos = lower_bound(a.begin(), a.end(), x) - a.begin();
```

**解引用前检查：**

```
auto it = lower_bound(a.begin(), a.end(), x);
if (it != a.end()) {
    cout << *it << '\n';
}
```

## 4.16 map 和 unordered_map

### 4.16.1 map

map 按 key 自动升序排列，常用于字符串计数、编号映射、需要有序遍历的场景。

```cpp
map<string, int> cnt;
cnt["apple"]++;
cnt["banana"] += 2;

cout << cnt["apple"] << '\n';
```

判断是否存在：

```cpp
if (cnt.count("apple")) {
    cout << "exists\n";
}

auto it = cnt.find("apple");
if (it != cnt.end()) {
    cout << it->second << '\n';
}
```

**注意：** `cnt[key]` 会在 key 不存在时自动创建一个默认值。

遍历：

```cpp
for (auto [key, value] : cnt) {
    cout << key << ' ' << value << '\n';
}
```

### 4.16.2 unordered_map

`unordered_map` 平均更快，但没有顺序。

```cpp
unordered_map<string, int> cnt;
cnt["apple"]++;
```

考场建议：

- 需要按 key 排序输出，用 `map`。
- 只做计数且数据量大，用 `unordered_map`。
- 自定义 key 时新手优先考虑转成字符串或 `long long` 编码，不要现场写复杂 hash。

## 4.17 unordered_set

`unordered_set` 用来快速判断一个值是否出现过，不关心顺序。

```cpp
unordered_set<int> seen;
seen.insert(5);

if (seen.count(5)) {
    cout << "yes\n";
}
```

字符串集合：

```cpp
unordered_set<string> words;
words.insert("abc");
if (words.count("abc")) {
    cout << "exists\n";
}
```

如果需要按升序遍历，用 set，不要用 unordered_set。

## 4.18 set 和 multiset

### 4.18.1 set

set 自动排序且不重复。

```cpp
set<int> s;
s.insert(3);
s.insert(1);
s.erase(3);

if (s.count(1)) {
    cout << "exists\n";
}
```

查找第一个大于等于 x 的元素：

```cpp
auto it = s.lower_bound(x);
if (it != s.end()) {
    cout << *it << '\n';
}
```

### 4.18.2 multiset

multiset 允许重复元素。

```cpp
multiset<int> s;
s.insert(5);
s.insert(5);
cout << s.count(5) << '\n'; // 2
```

删除一个 5：

```cpp
auto it = s.find(5);
if (it != s.end()) s.erase(it);
```

删除所有 5：

```cpp
s.erase(5);
```

## 4.19 queue、stack、deque

### 4.19.1 queue

BFS 常用队列。

```
queue<int> q;
q.push(1);
int x = q.front();
q.pop();
bool empty = q.empty();
```

### 4.19.2 stack

括号匹配、表达式、DFS 模拟可用栈。

```
stack<char> st;
st.push('(');
char c = st.top();
st.pop();
```

**调用 front/top 前必须检查非空。**

### 4.19.3 deque

双端队列，两边都能进出。

```
deque<int> dq;
dq.push_back(1);
dq.push_front(2);
dq.pop_back();
dq.pop_front();
```

单调队列会用到 deque。

## 4.20 priority_queue

### 4.20.1 大根堆

默认是大根堆，最大值在顶部。

```
priority_queue<int> pq;
pq.push(3);
pq.push(10);
cout << pq.top() << '\n'; // 10
pq.pop();
```

### 4.20.2 小根堆

```cpp
priority_queue<int, vector<int>, greater<int>> pq;
pq.push(3);
pq.push(10);
cout << pq.top() << '\n'; // 3
```

### 4.20.3 pair 小根堆

Dijkstra 常用：

```cpp
using P = pair<long long, int>; // distance, node
priority_queue<P, vector<P>, greater<P>> pq;
pq.push({0, 1});
```

pair 的默认比较先看 `first`，再看 `second`。

## 4.21 常用算法函数

### 4.21.1 min、max、swap

```cpp
int x = min(a, b);
int y = max(a, b);
swap(a, b);
```

### 4.21.2 reverse

```cpp
reverse(a.begin(), a.end());
reverse(s.begin(), s.end());
```

### 4.21.3 unique 去重

去重前必须排序：

```cpp
sort(a.begin(), a.end());
a.erase(unique(a.begin(), a.end()), a.end());
```

### 4.21.4 accumulate

```cpp
long long sum = accumulate(a.begin(), a.end(), 0LL);
```

**第三个参数写 `0LL`，否则可能按 `int` 求和。**

### 4.21.5 gcd

```cpp
long long g = gcd(a, b);
long long l = a / g * b;
```

**最小公倍数先除后乘。**这样比 `a * b / gcd(a,b)` 更不容易溢出。

### 4.21.6 常用数学函数

```cpp
double r = sqrt(x); // square root
double y = pow(a, b); // a^b, floating point
double f = floor(v); // round down as floating point
double c = ceil(v); // round up as floating point
long long z = llround(v); // nearest integer
```

提醒：

- sqrt、pow 返回浮点数，不适合直接处理大整数精确幂。
- 整数平方尽量写 1LL * x * x，不要写 pow(x, 2)。
- 判断一个整数是否为完全平方数时，先取 sqrt，再用整数乘法校验。
- 整数向上取整不要用 ceil(1.0*a/b) 代替整数公式。

```cpp
bool isSquare(long long x) {
    if (x < 0) return false;
    long long r = sqrt((long double)x);
    while (r > 0 && r > x / r) r--;
    while ((r + 1) <= x / (r + 1)) r++;
    return r * r == x;
}
```

## 4.22 lambda 入门

lambda 是临时函数，排序时非常常用。

最基本写法：

```cpp
[](int a, int b) {
    return a < b;
}
```

用于排序：

```cpp
sort(a.begin(), a.end(), [](int x, int y) {
    return x > y;
});
```

使用外部变量：

```cpp
int mod = 10;
sort(a.begin(), a.end(), [mod](int x, int y) {
    return x % mod < y % mod;
});
```

引用捕获：

```cpp
int cnt = 0;
auto add = [&](int x) {
```

```
    cnt += x;
};
add(5);
```

考场建议：

- 排序比较器中优先使用 `const Type&`，避免复制。
- 不熟悉捕获时，尽量不要写复杂 lambda。
- 复杂逻辑可以改成普通函数或先计算辅助字段。

## 4.23 常见编译错误和原因

| 报错或现象 | 常见原因 | 处理方法 |
|---|---|---|
| `expected ';'` | 少分号 | 看上一行结构体、变量定义、语句末尾。 |
| `was not declared` | 变量未定义或作用域不对 | 检查拼写和定义位置。 |
| `no matching function` | 函数参数类型不匹配 | 检查传参类型、引用、const。 |
| `segmentation fault` | 越界、空容器取值、递归爆栈 | 查数组下标、`front/top/back` 前是否非空。 |
| 样例正确但大数据错 | 溢出或复杂度过高 | 检查 `long long` 和复杂度。 |
| 输出格式错误 | 多输出、少换行、空格不对 | 严格对照题目输出格式。 |

## 4.24 C 和 C++ 混用提醒

如果你使用 C++，建议全程使用 `cin/cout` 或全程使用 `scanf/printf`，不要混用。

如果确实使用 C 风格输入输出：

```
int x;
scanf("%d", &x);

long long y;
scanf("%lld", &y);

printf("%d\n", x);
printf("%lld\n", y);
```

字符串：

```
char s[1005];
scanf("%s", s);
```

但对新手来说，C++ 的 `string`、`vector`、`map` 会明显减少代码量和错误率。

## 4.25 考场 C++ 快速检查

提交前快速看一遍：

- 是否包含了头文件和 `using namespace std;`。
- 是否开了快速输入输出。
- 是否混用了 `cin/cout` 和 `scanf/printf`。
- 求和、乘法、答案是否用了 `long long`。
- vector 是否按正确大小初始化。
- 访问 `front/top/back` 前是否判断非空。
- `lower_bound` 返回 `end` 时是否处理。
- 排序比较器相等时是否返回 `false` 或继续比较。
- 多组数据是否清空容器。
- 输出是否完全符合题意。

# 第五章　A 题决策树与高频模板

## 5.1　A 题定位

A 题通常不是为了考很深的算法，而是考你能否准确读题、正确实现、处理边界、按格式输出。对新手来说，A 题最重要的不是写得快，而是写得稳。

- 建议目标时间：20–35 分钟。
- 最大容忍时间：45 分钟。超过后仍无法通过样例，应先转下一题。
- 目标分数：100 分。
- 核心原则：先正确，再简洁；先样例，再边界。

**A 题最大敌人不是算法，而是低级错误。**常见低级错误包括输出格式错、下标错、整数溢出、没读完整输入、把题意中的“包含/不包含”看反。

## 5.2　A 题开题流程

每道 A 题按下面顺序处理，不要直接看完第一段就开始写。

1. 看输入格式：有几个数、几行、是否有字符串、是否有多组。
2. 看输出格式：输出一个数、一行多个数、字符串、小数、是否有特殊提示。
3. 看数据范围：是否需要 `long long`，数组要开多大。
4. 判断题型：统计、公式、排序、字符串、日期、进制、简单模拟。
5. 在草稿纸上写 1 到 2 个极端样例。
6. 写代码。
7. 过样例后，测试边界。

## 5.3　A 题读题标注法

新手做 A 题时，最容易出现“感觉看懂了，但代码写偏了”。建议在草稿纸上用固定格式标注。

### 5.3.1　读题时必须圈出的信息

| 要圈的信息 | 为什么重要 | 例子 |
| --- | --- | --- |

| | | |
|---|---|---|
| 输入数量 | 决定循环次数 | 读 $n$ 个数、读 $n$ 行、读到 EOF。 |
| 数据范围 | 决定数组大小和复杂度 | $n \le 10^5$ 时不要双重循环。 |
| 数值范围 | 决定是否用 `long long` | 每个数 $10^9$，总和可能溢出。 |
| 下标规则 | 决定 0 下标或 1 下标 | 第 1 个、第 $i$ 个、编号从 0 开始。 |
| 区间端点 | 决定是否包含端点 | $[l,r]$、$(l,r)$、从 $l$ 到 $r$。 |
| 输出格式 | 决定空格、换行、小数位 | 输出 YES，保留 2 位小数。 |
| 特殊情况 | 决定是否特判 | 无解、相等、为空、越界、最后一天。 |

### 5.3.2 草稿纸模板

做题前写 6 行：

```
Input:
Output:
n range:
value range:
Need long long? yes/no
Type: statistic / formula / sort / string / date / base / simulate
```

示例：

```
Input: n, then n integers
Output: sum and max
n range: 1e5
value range: 1e9
Need long long? yes
Type: statistic
```

### 5.3.3 样例手算方法

样例不是只拿来跑程序的。写代码前先手算一遍：

1. 把输入拆成变量。
2. 按题目规则一步一步算。
3. 写出中间结果。
4. 对比样例输出。

如果你手算不出样例，就不要急着写代码。A 题手算不清，代码大概率也会写偏。

## 5.4 A 题总决策树

| 题面信号 | 优先判断为 | 去看 |
|---|---|---|
| 求和、平均、最大、最小、计数 | 简单统计 | 本章“统计模板” |

| | | |
|---|---|---|
| 根据公式算一个值 | 公式题 | 本章“公式题检查” |
| 要求排序、排名、第几名 | 排序题 | 第二章 sort；本章“排序排名” |
| 出现字符、字符串、单词、大小写 | 字符串题 | 第二章 string；本章“字符串处理” |
| 出现二进制、十六进制、进制转换 | 进制题 | 本章“进制处理” |
| 出现日期、时间、星期、天数 | 日期时间题 | 本章“日期时间” |
| 出现坐标、网格、方向移动 | 简单模拟或网格模拟 | 本章“简单模拟” |
| 出现百分比、保留小数、四舍五入 | 小数输出题 | 第二章小数输出 |
| 出现区间但数据很小 | 暴力枚举或前缀和 | 本章“区间入门” |
| 出现重复统计、出现次数 | 计数数组或 map | 第二章 map；本章“频率统计” |

## 5.5 A 题关键词到动作

这张表用于读题时快速决定下一步。每一行都应该能从题面直接判断。

| 你看到 | 立刻做什么 | 小心什么 |
|---|---|---|
| “共 $n$ 个” | 开数组或 `vector`，循环 $n$ 次 | 循环是 `i < n` 还是 `i <= n`。 |
| “编号从 1 到 n” | 优先开 `n+1`，按 1 下标写 | 不要访问 `a[0]`。 |
| “第 k 个” | 注意 $k$ 是 1 下标 | `a[k-1]` 或排序后 `a[k-1]`。 |
| “最大/最小” | 一遍扫描或排序 | 全负数时最大值不能初始化 0。 |
| “总和/累计” | 答案用 `long long` | `int` 乘法先溢出。 |
| “平均” | 判断输出整数还是小数 | 整数除法会截断。 |
| “每 k 个一组”“分成若干页” | 正整数向上取整 | 只在 $a \ge 0, b > 0$ 时用 `(a+b-1)/b`。 |
| “保留 x 位” | 用 `fixed << setprecision(x)` | 是否四舍五入由输出库自动处理。 |
| “升序/降序” | 使用 `sort` | 降序可用 `greater<int>()`。 |
| “相同则按...” | 写完整比较器 | 相等时不能返回 `true`。 |
| “字符串可能含空格” | 使用 `getline` | 前面用过 `cin` 要 `ignore`。 |
| “统计出现次数” | 值域小用数组，值域大用 `map` | 是否要求有序输出。 |
| “判断是否存在” | 用 `set/count` 或排序后二分 | 找不到时不要解引用迭代器。 |
| “日期” | 写闰年、每月天数 | 2 月和跨年。 |
| “进制” | 先判断是否超过 `long long` | 大数不能 `stoll`。 |

| | | |
|---|---|---|
| “区间 [l,r]” | 看有多少次查询 | 多次查询考虑前缀和。 |

## 5.6 A 题数据范围判断

| 看到的数据范围 | 含义 | 建议 |
|---|---|---|
| $n \le 100$ | 很小 | 直接模拟或双重循环通常可行。 |
| $n \le 1000$ | 小到中等 | $O(n^2)$ 通常可试。 |
| $n \le 10^5$ | 常见大数组 | 需要 $O(n)$ 或 $O(n \log n)$，不要双重循环。 |
| 数值 $\le 10^9$ | 单个数可放 int | 但加法和乘法可能要 `long long`。 |
| 乘积、总和、答案可能很大 | 有溢出风险 | 默认用 `long long`。 |
| 字符串长度很大 | 不能频繁插入删除 | 用线性扫描。 |

## 5.7 A 题常见输入形态

### 5.7.1 输入一个数

```cpp
int x;
cin >> x;
cout << x << '\n';
```

### 5.7.2 输入两个或多个数

```cpp
int a, b, c;
cin >> a >> b >> c;
cout << a + b + c << '\n';
```

### 5.7.3 输入 n 个数

```cpp
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++) {
    cin >> a[i];
}
```

### 5.7.4 输入 n 行记录

例如每行一个学生的编号和成绩：

```cpp
struct Student {
    int id;
    int score;
};

int n;
cin >> n;
vector<Student> a(n);
for (int i = 0; i < n; i++) {
    cin >> a[i].id >> a[i].score;
}
```

### 5.7.5 输入 n 行字符串

```cpp
int n;
cin >> n;
vector<string> s(n);
for (int i = 0; i < n; i++) {
    cin >> s[i];
}
```

### 5.7.6 读到文件结束

```cpp
int x;
while (cin >> x) {
    // process x
}
```

**题目没有说有 T 组数据时，不要自己加 T。**

## 5.8 简单统计

### 5.8.1 识别信号

题面出现：

- 求总和、平均数。
- 统计满足条件的个数。
- 找最大值、最小值。
- 统计正数、负数、奇数、偶数。
- 统计超过某个阈值的元素。

### 5.8.2 一遍扫描模板

```
int n;
cin >> n;

long long sum = 0;
int cnt = 0;
int mx = -1000000000;
int mn = 1000000000;

for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    sum += x;
    if (x > mx) mx = x;
    if (x < mn) mn = x;
    if (x % 2 == 0) cnt++;
}

cout << sum << ' ' << mx << ' ' << mn << ' ' << cnt << '\n';
```

### 5.8.3 注意事项

- 求和变量用 long long。
- 最大值、最小值初始值要足够小或足够大。
- 如果数据可能全是负数，最大值不能初始化为 0。
- 平均数如果要小数，使用 double 或输出时转换。

### 5.8.4 统计满足条件的个数

```
int n;
cin >> n;
int cnt = 0;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (x >= 60) {
        cnt++;
    }
}
cout << cnt << '\n';
```

### 5.8.5 同时统计多个类别

例如统计正数、负数、零：

```cpp
int n;
cin >> n;
int pos = 0, neg = 0, zero = 0;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (x > 0) pos++;
    else if (x < 0) neg++;
    else zero++;
}
cout << pos << ' ' << neg << ' ' << zero << '\n';
```

### 5.8.6 找最大值的位置

```cpp
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];

int bestPos = 0;
for (int i = 1; i < n; i++) {
    if (a[i] > a[bestPos]) {
        bestPos = i;
    }
}
cout << bestPos << ' ' << a[bestPos] << '\n';
```

如果题目说“第几个”，通常要输出 `bestPos + 1`。

### 5.8.7 计数数组

当值域很小，例如分数在 0 到 100：

```cpp
int n;
cin >> n;
vector<int> cnt(101, 0);
for (int i = 0; i < n; i++) {
    int score;
    cin >> score;
    cnt[score]++;
}

for (int s = 0; s <= 100; s++) {
    if (cnt[s] > 0) {
        cout << s << ' ' << cnt[s] << '\n';
    }
}
```

**计数数组前提：**数值不能是很大的 $10^9$，也不能是负数，除非你做了偏移。

## 5.9 存在、全部、任意判断

A 题常见问法：

- 是否存在一个数满足条件。
- 是否所有数都满足条件。
- 是否至少有一个字符串包含某内容。
- 是否没有任何违规情况。

### 5.9.1 判断是否存在

```
int n;
cin >> n;
bool found = false;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (x == 0) {
        found = true;
    }
}

cout << (found ? "YES" : "NO") << '\n';
```

### 5.9.2 判断是否全部满足

```
int n;
cin >> n;
bool ok = true;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (x < 60) {
        ok = false;
    }
}

cout << (ok ? "YES" : "NO") << '\n';
```

### 5.9.3 提前结束

如果后面的输入还必须读完，不要随便 `break`。除非你确定后续数据不需要再读，或者已经读完本组数据。

```cpp
bool found = false;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    if (x == target) {
        found = true;
    }
}
```

**新手建议：** A 题中先不要为了省时间提前 `break`，完整读入更稳。

## 5.10 公式题

### 5.10.1 识别信号

题面给出明确计算规则，例如：

- 根据若干输入计算费用、得分、距离、面积。
- 分段收费。
- 按比例、百分比、折扣计算。
- 对结果取整、向上取整、向下取整。

### 5.10.2 公式题流程

1. 先在草稿纸上写出数学式。
2. 判断每个中间变量是否可能溢出。
3. 判断除法是整数除法还是小数除法。
4. 判断是否需要四舍五入、向上取整、向下取整。
5. 用样例手算一遍。

### 5.10.3 整数除法和取整

公式题中最容易错的是“除法到底按什么规则取整”。先分清三种情况：

| 题目要求 | 常用写法 | 适用条件 |
| --- | --- | --- |
| 普通整数除法，向 0 截断 | `a / b` | C++ 默认规则，负数时不是向下取整。 |
| 正整数向上取整 | `(a + b - 1) / b` | 只适合 $a \ge 0, b > 0$。 |
| 数学意义向下/向上取整 | 使用 `floor_div` / `ceil_div` | 适合可能出现负数的公式。 |

正整数向上取整最常见，例如 $a$ 个物品每组放 $b$ 个：

```cpp
long long ceil_div_positive(long long a, long long b) {
    return (a + b - 1) / b;
}
```

若 $a$ 可能为负，或题目明确要求数学意义的向下/向上取整，使用通用写法：

```cpp
long long floor_div(long long a, long long b) {
    long long q = a / b;
    long long r = a % b;
    if (r != 0 && ((r > 0) != (b > 0))) q--;
    return q;
}

long long ceil_div(long long a, long long b) {
    long long q = a / b;
    long long r = a % b;
    if (r != 0 && ((r > 0) == (b > 0))) q++;
    return q;
}
```

**不要随便用浮点数做整数取整题**。如果题目全是整数，通常可以用整数公式避免误差；特别是数值接近 $10^{18}$ 时，`double` 可能无法精确表示每一个整数。

### 5.10.4 分段计算模板

```cpp
int x;
cin >> x;

int ans = 0;
if (x <= 10) {
    ans = x * 2;
} else if (x <= 20) {
    ans = 10 * 2 + (x - 10) * 3;
} else {
    ans = 10 * 2 + 10 * 3 + (x - 20) * 5;
}

cout << ans << '\n';
```

注意：

- 边界是 `<=` 还是 `<`。
- 每一段是否包含端点。
- 前面几段的费用是否已经累计。

### 5.10.5 分类讨论模板

A 题经常给出若干规则，按条件输出不同结果。

```cpp
int x;
cin >> x;

if (x < 0) {
    cout << "negative\n";
} else if (x == 0) {
    cout << "zero\n";
} else {
    cout << "positive\n";
}
```

分类讨论检查：

- 条件是否覆盖所有情况。
- 条件顺序是否正确。
- 边界值属于哪一类。
- 是否存在重叠条件。

### 5.10.6 绝对值和距离

```cpp
int a, b;
cin >> a >> b;
cout << abs(a - b) << '\n';
```

如果 $a, b$ 很大：

```cpp
long long a, b;
cin >> a >> b;
cout << llabs(a - b) << '\n';
```

### 5.10.7 取模

```cpp
long long ans = 0;
const long long MOD = 1000000007;
ans = (ans + x) % MOD;
ans = ans * y % MOD;
```

如果可能出现负数：

```cpp
long long r = x % MOD;
if (r < 0) r += MOD;
```

### 5.10.8 小数和百分比

保留小数：

```cpp
double x;
cin >> x;
cout << fixed << setprecision(2) << x << '\n';
```

整数转百分比：

```cpp
int good, total;
cin >> good >> total;
double rate = 100.0 * good / total;
cout << fixed << setprecision(2) << rate << "%\n";
```

平均数：

```cpp
long long sum = 0;
int n;
cin >> n;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    sum += x;
}
double avg = 1.0 * sum / n;
cout << fixed << setprecision(1) << avg << '\n';
```

**注意：** `sum / n` 如果两边都是整数，会做整数除法。写成 `1.0 * sum / n`。

## 5.11 排序和排名

### 5.11.1 识别信号

题面出现：

- 按成绩排序、按编号排序、按时间排序。
- 输出第 $k$ 个。
- 排名、名次、优先级。
- 相同分数时按另一个条件排序。

### 5.11.2 单个数组排序

```cpp
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];

sort(a.begin(), a.end());

for (int x : a) {
    cout << x << ' ';
```

```
}
cout << '\n';
```

降序：

```
sort(a.begin(), a.end(), greater<int>());
```

### 5.11.3 结构体排序

```
struct Student {
    int id;
    int score;
};

int n;
cin >> n;
vector<Student> a(n);
for (int i = 0; i < n; i++) {
    cin >> a[i].id >> a[i].score;
}

sort(a.begin(), a.end(), [](const Student& x, const Student& y) {
    if (x.score != y.score) return x.score > y.score;
    return x.id < y.id;
});

for (const auto& s : a) {
    cout << s.id << ' ' << s.score << '\n';
}
```

### 5.11.4 排名常见规则

排名题一定要看清楚“并列名次”。

普通排序后第 $k$ 个：

```
sort(a.begin(), a.end(), greater<int>());
cout << a[k - 1] << '\n';
```

并列排名示例：

```
vector<int> score = {100, 90, 90, 80};
int rank = 1;
for (int i = 0; i < (int)score.size(); i++) {
    if (i > 0 && score[i] != score[i - 1]) {
        rank = i + 1;
    }
    cout << score[i] << ' ' << rank << '\n';
}
```

**排序题高危点：**比较器里相等时不能返回 `true`，需要继续比较或返回 `false`。

### 5.11.5 按字符串字典序排序

```
int n;
cin >> n;
vector<string> s(n);
for (int i = 0; i < n; i++) cin >> s[i];

sort(s.begin(), s.end());

for (string x : s) {
    cout << x << '\n';
}
```

### 5.11.6 先去重再排序

```
sort(a.begin(), a.end());
a.erase(unique(a.begin(), a.end()), a.end());
```

### 5.11.7 找第 k 小和第 k 大

```
sort(a.begin(), a.end());
cout << a[k - 1] << '\n'; // kth smallest

sort(a.begin(), a.end(), greater<int>());
cout << a[k - 1] << '\n'; // kth largest
```

**第 k 个通常是 1 下标，所以用 k - 1。**

## 5.12 字符串处理

### 5.12.1 识别信号

题面出现：

- 字符、单词、大小写。
- 判断前缀、后缀、是否包含。
- 统计某字符出现次数。
- 按空格、逗号、冒号分割。
- 文件路径、命令、表达式。

### 5.12.2 逐字符扫描

```
string s;
cin >> s;

```

```
int digits = 0;
int letters = 0;
for (char c : s) {
    if (isdigit(c)) digits++;
    if (isalpha(c)) letters++;
}

cout << digits << ' ' << letters << '\n';
```

### 5.12.3 大小写转换

```
string s;
cin >> s;
for (char& c : s) {
    c = tolower(c);
}
cout << s << '\n';
```

### 5.12.4 查找子串

```
string s, t;
cin >> s >> t;
if (s.find(t) != string::npos) {
    cout << "YES\n";
} else {
    cout << "NO\n";
}
```

### 5.12.5 按空格分割一整行

```
string line;
getline(cin, line);

stringstream ss(line);
string word;
while (ss >> word) {
    cout << word << '\n';
}
```

如果前面读过整数：

```
int n;
cin >> n;
cin.ignore();

string line;
getline(cin, line);
```

### 5.12.6 按指定字符分割

```cpp
vector<string> split(const string& s, char sep) {
    vector<string> res;
    string cur;
    for (char c : s) {
        if (c == sep) {
            res.push_back(cur);
            cur.clear();
        } else {
            cur.push_back(c);
        }
    }
    res.push_back(cur);
    return res;
}
```

**字符串题常见坑：**空字符串、连续分隔符、行尾换行、下标从 0 开始。

### 5.12.7 判断前缀和后缀

前缀：

```cpp
bool startsWith(const string& s, const string& p) {
    if (p.size() > s.size()) return false;
    for (int i = 0; i < (int)p.size(); i++) {
        if (s[i] != p[i]) return false;
    }
    return true;
}
```

后缀：

```cpp
bool endsWith(const string& s, const string& p) {
    if (p.size() > s.size()) return false;
    int offset = (int)s.size() - (int)p.size();
    for (int i = 0; i < (int)p.size(); i++) {
        if (s[offset + i] != p[i]) return false;
    }
    return true;
}
```

### 5.12.8 统计每个小写字母

```cpp
string s;
cin >> s;
vector<int> cnt(26, 0);
for (char c : s) {
    if ('a' <= c && c <= 'z') {
```

```
        cnt[c - 'a']++;
    }
}
```

### 5.12.9 反转字符串

```
string s;
cin >> s;
reverse(s.begin(), s.end());
cout << s << '\n';
```

### 5.12.10 回文判断

```
bool isPalindrome(const string& s) {
    int l = 0, r = (int)s.size() - 1;
    while (l < r) {
        if (s[l] != s[r]) return false;
        l++;
        r--;
    }
    return true;
}
```

## 5.13 进制处理

### 5.13.1 识别信号

题面出现：

- 二进制、八进制、十六进制。
- 把某进制转为十进制。
- 把十进制转为某进制。
- 位、编码、补码、掩码。

### 5.13.2 任意进制转十进制

```
long long toDecimal(const string& s, int base) {
    long long ans = 0;
    for (char c : s) {
        int v;
        if ('0' <= c && c <= '9') v = c - '0';
        else if ('A' <= c && c <= 'F') v = c - 'A' + 10;
        else v = c - 'a' + 10;
        ans = ans * base + v;
```

```
    }
    return ans;
}
```

### 5.13.3 十进制转任意进制

```
string fromDecimal(long long x, int base) {
    if (x == 0) return "0";
    string digits = "0123456789ABCDEF";
    string res;
    while (x > 0) {
        res.push_back(digits[x % base]);
        x /= base;
    }
    reverse(res.begin(), res.end());
    return res;
}
```

### 5.13.4 位运算入门

```
if (x & 1) {
    // x is odd
}

if (x & (1 << k)) {
    // bit k is 1
}

x |= (1 << k); // set bit k to 1
x &= ~(1 << k); // set bit k to 0
```

**如果 $k \geq 31$，写 `1LL << k`，不要写 `1 << k`。**

### 5.13.5 进制题检查

- 输入数字是否可能很长。
- 转成十进制是否超过 `long long`。
- 十六进制字母是大写还是小写。
- 输出是否要求补前导零。
- 二进制位编号从 0 还是 1 开始。

## 5.14 日期时间

### 5.14.1 识别信号

题面出现：

- 年、月、日、星期。
- 给定日期求过几天。
- 两个日期之间的天数。
- 时间格式如 HH:MM:SS。

### 5.14.2 闰年

```
bool leap(int y) {
    return (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);
}
```

### 5.14.3 每月天数

```
int mdays(int y, int m) {
    int d[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
    if (m == 2 && leap(y)) return 29;
    return d[m];
}
```

### 5.14.4 下一天

```
void nextDay(int& y, int& m, int& d) {
    d++;
    if (d > mdays(y, m)) {
        d = 1;
        m++;
        if (m > 12) {
            m = 1;
            y++;
        }
    }
}
```

### 5.14.5 时间转秒

```
int toSeconds(int h, int m, int s) {
    return h * 3600 + m * 60 + s;
}
```

**日期题先看范围。**如果只跨几年，可以逐日模拟；如果年份很大，要转为天数公式或按周期处理。

## 5.15 简单模拟

### 5.15.1 识别信号

题面出现：

- 按规则一步一步执行。
- 有当前位置、方向、状态。
- 有若干操作命令。
- 有网格移动，但不要求最短路。
- 有游戏规则、计分规则。

### 5.15.2 方向数组

四方向：

```
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};
```

八方向：

```
int dx[8] = {-1,-1,-1,0,0,1,1,1};
int dy[8] = {-1,0,1,-1,1,-1,0,1};
```

判断是否在网格内：

```
bool inside(int x, int y, int n, int m) {
    return 0 <= x && x < n && 0 <= y && y < m;
}
```

### 5.15.3 按命令移动

```
int x = 0, y = 0;
string ops;
cin >> ops;

for (char op : ops) {
    if (op == 'U') x--;
    else if (op == 'D') x++;
    else if (op == 'L') y--;
    else if (op == 'R') y++;
}

cout << x << ' ' << y << '\n';
```

### 5.15.4 模拟题写法建议

- 状态变量先列出来，例如位置、方向、分数、时间。
- 每个操作只改变少量状态。
- 复杂规则拆成函数。
- 样例过后自己造最小边界和最大边界。

## 5.16 坐标和距离入门

A 题中若出现坐标，通常是简单计算，不一定是图搜索。

### 5.16.1 二维点

```
struct Point {
    long long x;
    long long y;
};

Point a, b;
cin >> a.x >> a.y >> b.x >> b.y;
```

### 5.16.2 曼哈顿距离

只能上下左右走时，常见距离：

```
long long dist = llabs(a.x - b.x) + llabs(a.y - b.y);
cout << dist << '\n';
```

### 5.16.3 欧几里得距离

直线距离：

```
double dx = a.x - b.x;
double dy = a.y - b.y;
double dist = sqrt(dx * dx + dy * dy);
cout << fixed << setprecision(6) << dist << '\n';
```

### 5.16.4 坐标题检查

- 坐标可能为负数。
- 坐标差的平方可能溢出，必要时用 `long long` 或 `double`。
- 题目要求的是曼哈顿距离还是直线距离。
- 输出是否需要小数。

## 5.17 二维表格和矩阵入门

A 题可能出现很简单的二维表格，例如统计每行和、每列和、找最大值。

### 5.17.1 读入 n 行 m 列

```
int n, m;
cin >> n >> m;
vector<vector<int>> a(n, vector<int>(m));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        cin >> a[i][j];
    }
}
```

### 5.17.2 每行求和

```
for (int i = 0; i < n; i++) {
    long long sum = 0;
    for (int j = 0; j < m; j++) {
        sum += a[i][j];
    }
    cout << sum << '\n';
}
```

### 5.17.3 每列求和

```
for (int j = 0; j < m; j++) {
    long long sum = 0;
    for (int i = 0; i < n; i++) {
        sum += a[i][j];
    }
    cout << sum << '\n';
}
```

### 5.17.4 找矩阵最大值

```
int bx = 0, by = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        if (a[i][j] > a[bx][by]) {
            bx = i;
            by = j;
        }
    }
}
```

```
cout << bx << ' ' << by << ' ' << a[bx][by] << '\n';
```

如果题目要求第几行第几列，通常输出 `bx + 1` 和 `by + 1`。

## 5.18 频率统计

### 5.18.1 值域小：用数组

如果数值范围是 0 到 1000：

```
int cnt[1001] = {};
int n;
cin >> n;
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    cnt[x]++;
}
```

### 5.18.2 值域大或 key 是字符串：用 map

```
int n;
cin >> n;
map<string, int> cnt;
for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
    cnt[s]++;
}

for (auto [s, c] : cnt) {
    cout << s << ' ' << c << '\n';
}
```

### 5.18.3 只需要快速统计：unordered_map

```
unordered_map<int, int> cnt;
for (int x : a) {
    cnt[x]++;
}
```

**如果题目要求按字典序输出，不能用 `unordered_map` 直接遍历。**

## 5.19 区间入门

A 题中也可能出现简单区间。先看数据范围。

### 5.19.1 数据小：直接循环

```
int l, r;
cin >> l >> r;
long long sum = 0;
for (int i = l; i <= r; i++) {
    sum += a[i];
}
```

### 5.19.2 多次区间和：前缀和

```
int n, q;
cin >> n >> q;
vector<long long> pre(n + 1, 0);
for (int i = 1; i <= n; i++) {
    long long x;
    cin >> x;
    pre[i] = pre[i - 1] + x;
}

while (q--) {
    int l, r;
    cin >> l >> r;
    cout << pre[r] - pre[l - 1] << '\n';
}
```

注意：

- 上面模板假设下标从 1 开始。
- 如果题目给的是 0 下标，要改公式。
- 前缀和数组用 long long。

## 5.20 A 题输出格式专项

提交前逐项对照：

- 题目要求输出 YES/NO 还是 Yes/No。
- 是否每个答案一行。
- 多个数之间是空格还是换行。
- 最后一项后面是否允许空格。一般允许，但不依赖。
- 小数保留几位。
- 是否要求百分号、冒号、逗号等符号。

输出数组推荐写法：

```cpp
for (int i = 0; i < n; i++) {
    if (i) cout << ' ';
    cout << a[i];
}
cout << '\n';
```

## 5.21 A 题新手错误对照表

| 现象 | 可能原因 | 快速检查 |
| --- | --- | --- |
| 样例输出差一个数 | 循环次数错 | 检查 `i < n` 和 `i <= n`。 |
| 样例数字对但格式错 | 空格、换行、大小写错误 | 对照输出格式逐字符看。 |
| 小数据对，大数据错 | `int` 溢出 | 把和、乘积、答案改为 `long long`。 |
| 排序后顺序怪 | 比较器错误 | 相等时返回 `false` 或继续比较。 |
| 读入字符串为空 | `cin` 后直接 `getline` | 加 `cin.ignore()`。 |
| 程序崩溃 | 数组越界或空容器取值 | 查 `a[i]`、`front/top/back`。 |
| 答案少算最后一项 | 区间端点没包含 | 检查 `<= r` 还是 `< r`。 |
| 答案多算一项 | 下标起点错 | 检查 0 下标和 1 下标是否混用。 |
| 找子串时明明在开头却判断失败 | 错用 `if (s.find(t))` | 改为和 `string::npos` 比较。 |
| 多组数据第二组错 | 没清空状态 | 每组重置数组、map、答案。 |

## 5.22 A 题推荐代码组织

简单题可以直接写在 `main`，但如果题目有多组数据，推荐写 `solve()`。

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

void solve() {
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    ll sum = 0;
    for (int x : a) {
        sum += x;
```

```
    }
    cout << sum << '\n';
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    solve();
    return 0;
}
```

如果题目有 $T$ 组：

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;
    while (T--) {
        solve();
    }
    return 0;
}
```

**如果题目没有多组数据，不要自己读一个 T。**

## 5.23 A 题自造边界样例

样例通过后，至少测试：

- 最小输入：$n = 0$ 或 $n = 1$，如果题目允许。
- 最大输入附近：最大数值、最大长度。
- 全相同数据。
- 全为 0。
- 全为负数，如果题目允许。
- 排序中有相同关键字。
- 字符串为空或只有一个字符，如果题目允许。
- 日期为 2 月 28 日、2 月 29 日、12 月 31 日。

## 5.24 A 题本地调试流程

如果样例不过，不要盲目改代码。按下面顺序查。

### 5.24.1 第一步：确认读入

临时输出读到的变量：

```
cerr << "n=" << n << '\n';
for (int i = 0; i < n; i++) {
    cerr << "a[" << i << "]=" << a[i] << '\n';
}
```

**提交前必须删除或注释调试输出**。虽然 cerr 通常不进入标准输出，但考场上不要依赖这一点，最终代码保持干净。

### 5.24.2 第二步：确认中间结果

例如统计题：

```
cerr << "sum=" << sum << " cnt=" << cnt << '\n';
```

排序题：

```
for (int x : a) cerr << x << ' ';
cerr << '\n';
```

### 5.24.3 第三步：用最小样例

把输入改成最小情况：

```
1
5
```

如果最小样例都错，通常是读入、初始化或输出格式错。

### 5.24.4 第四步：检查边界

- 循环是否少跑一次。
- 数组是否越界。
- 是否把 $n$ 当成最大下标。
- 是否忘记处理最后一个元素。
- 是否忘记初始化答案变量。

## 5.25 A 题常见边界库

做完样例后，根据题型选 2 到 3 个边界测。

| 题型 | 建议边界 | 目的 |
| --- | --- | --- |
| 统计 | $n = 1$，全 0，全负数，全相同 | 检查初始化和条件判断。 |

| | | |
|---|---|---|
| 最大最小 | 全负数、最大值在第一项、最大值在最后一项 | 检查初始值和循环。 |
| 排序 | 已有序、逆序、全相等、有重复关键字 | 检查比较器和稳定性。 |
| 字符串 | 长度 1、全数字、全字母、含空格、子串在开头 | 检查读取和 `find`。 |
| 日期 | 2 月 28 日、闰年 2 月 29 日、12 月 31 日 | 检查闰年和跨年。 |
| 进制 | 0、最大位、含 A-F、小写字母 | 检查特殊值和字符转换。 |
| 区间 | $l = r$，$l = 1$，$r = n$ | 检查前缀和端点。 |
| 二维表 | $1 \times 1$，单行，单列 | 检查双重循环。 |

## 5.26 A 题手写模板清单

这些模板建议单独练到不用看也能写。

| 模板 | 必须熟练程度 | 关联章节 |
|---|---|---|
| 读入 $n$ 个数 | 必须会默写 | 输入形态、vector |
| 一遍求和、最大、最小 | 必须会默写 | 简单统计 |
| 统计满足条件的个数 | 必须会默写 | 简单统计 |
| 结构体排序 | 必须照着能写 | 排序和排名 |
| 字符串逐字符扫描 | 必须会默写 | 字符串处理 |
| `getline` + `stringstream` | 必须照着能写 | 字符串处理 |
| 进制转十进制 | 照着能写 | 进制处理 |
| 闰年和每月天数 | 照着能写 | 日期时间 |
| 方向数组 | 照着能写 | 简单模拟 |
| 二维矩阵读入 | 必须会默写 | 二维表格 |
| 前缀和查询 | 照着能写 | 区间入门 |

## 5.27 A 题临场优先级

如果你在 A 题上紧张，按这个优先级保证分数：

1. 先保证读入正确。
2. 再保证样例能手算清楚。
3. 再写最直接的做法，不要优化。
4. 再处理边界。
5. 最后才考虑代码好不好看。

如果有两个写法：

- 一个写法短但你不熟。
- 一个写法长但你确定。

  A 题选第二个。

## 5.28 A 题提交前总检查

1. 输入是否完整读入。
2. 输出是否完全符合题目。
3. 是否有调试输出。
4. 是否需要 `long long`。
5. 数组是否越界。
6. 下标从 0 还是 1 开始是否统一。
7. 排序比较器是否正确。
8. 字符串读取是否受空格影响。
9. 小数精度是否正确。
10. 边界样例是否测试过。

## 5.29 A 题卡住时怎么办

- 读不懂题：重看输入输出样例，手算样例。
- 样例不过：打印中间变量在本地查错，提交前删掉。
- 怀疑溢出：把答案和中间量改成 `long long`。
- 怀疑下标：画出长度为 3 的小数组，手动走一遍。
- 超过 45 分钟仍未解决：保存代码，先做 B 或 C，之后回来。

# 第六章　B 题决策树与高频算法入口

## 6.1　B 题定位

B 题通常是从“会写代码”进入“会选算法”的第一道题。它一般不会要求特别深的算法证明，但常常要求你识别出比暴力更快的做法。

- 建议目标时间：40–70 分钟。
- 目标分数：尽量 100 分。
- 高频主题：排序、哈希统计、前缀和、差分、二分、简单 BFS、简单图、简单 DP。
- 核心风险：样例能过的暴力在大数据超时。

**B 题第一原则：**读完题后先看数据范围，再决定能不能暴力。不要因为 A 题刚做完很顺，就直接写双重循环。

## 6.2　B 题开题流程

1. 圈出数据范围：$n, m, q$ 分别多大。
2. 圈出操作次数：是否有很多查询、很多修改。
3. 判断暴力复杂度：如果每次操作都扫描一遍，总次数是多少。
4. 找题面信号：区间、排序、统计、最短、连通、依赖、答案范围。
5. 查 B 题决策树，定位高频模板。
6. 如果满分做法明确，直接写；如果不明确，先写可拿部分分的暴力。

## 6.3　B 题总决策树

| 题面信号 | 优先怀疑 | 首选模板 |
| --- | --- | --- |
| 多次询问区间和、区间最大统计 | 前缀和或预处理 | 一维/二维前缀和 |
| 多次区间加，最后查询结果 | 差分 | 一维/二维差分 |
| 区间修改和区间查询交替出现 | 树状数组或线段树 | 数据结构章节，B 题先看是否可简化 |

| | | |
|---|---|---|
| 统计出现次数、找重复、频率 | 哈希统计 | 数组计数、map、unordered_map |
| 排序后选择、排名、贪心安排 | 排序与扫描 | sort + 自定义比较器 |
| 前后两边信息、删除一个元素后的最优值 | 前后缀预处理 | 前缀最大/最小、后缀最大/最小 |
| 余数、整除、模 $k$ 相同 | 余数统计 | cnt[mod]、map 统计 |
| 第一个满足条件、答案具有单调性 | 二分或二分答案 | lower_bound / check |
| 最少步数、网格移动、迷宫 | BFS | 队列、距离数组 |
| 合并集合、判断是否同组 | 并查集 | DSU |
| 依赖关系、先后顺序 | 拓扑排序 | 入度 + 队列 |
| 小规模任意两点距离 | Floyd | 三重循环 |
| 正权图最短路 | Dijkstra | 堆优化 Dijkstra |
| 选择若干、容量限制、最优值 | 简单 DP | 背包或线性 DP |

## 6.4 B 题数据范围判断

| 数据范围 | 暴力是否可行 | 常见替代 |
|---|---|---|
| $n \le 1000$ | $O(n^2)$ 常常可行 | 双重循环、简单 DP |
| $n \le 10^5$ | $O(n^2)$ 通常不可行 | 排序、哈希、前缀和、二分 |
| $q \le 10^5$ 且每次查区间 | 每次扫描不可行 | 前缀和、树状数组、线段树 |
| 二维 $n, m \le 1000$ | $O(nm)$ 可行，$O(n^2m^2)$ 不可行 | 二维前缀和 |
| 图点数 $\le 500$ | Floyd 可能可行 | Floyd |
| 图点数、边数 $\le 10^5$ | Floyd 不可行 | BFS、Dijkstra、并查集 |
| 值域 $\le 10^6$ | 可用计数数组 | 数组计数 |
| 值域很大或字符串 key | 不能开值域数组 | map / unordered_map |

## 6.5 B 题关键词到动作

| 看到 | 立刻想 | 小心 |
|---|---|---|
| “多次询问” | 预处理，不能每次从头算 | 查询次数 $q$ 是否很大。 |
| “区间 [l,r] 的和” | 前缀和 | 下标和 `pre[r]-pre[l-1]`。 |
| “子矩阵和” | 二维前缀和 | 公式四项别写错。 |
| “区间加” | 差分 | 右端点后一位是否存在。 |
| “出现次数” | 计数数组或 map | 值域是否能开数组。 |

| | | |
|---|---|---|
| “去重后数量” | set 或 sort+unique | 是否需要保持顺序。 |
| “第一个不小于” | lower_bound | 容器必须有序。 |
| “最小的最大值” | 二分答案 | 必须能写 check。 |
| “删除一个后”“左边右边” | 前后缀预处理 | 注意边界位置。 |
| “除以 k 的余数”“能否整除” | 余数统计 | 负数取模要修正。 |
| “尽量多选”“最早结束”“最少次数覆盖” | 贪心 | 必须按正确关键字排序。 |
| “最短步数” | BFS | BFS 适合每步代价相同。 |
| “连通、合并” | 并查集 | 初始化每个点的父亲。 |
| “前置条件、依赖” | 拓扑排序 | 入度更新。 |

## 6.6 从暴力到优化的转换表

B 题经常不是完全不会，而是暴力会写但会超时。下面这张表用于把暴力做法改成常见优化。

| 暴力想法 | 为什么危险 | 常见优化 |
|---|---|---|
| 每次查询都遍历 $[l,r]$ | $O(nq)$，$q$ 大时超时 | 前缀和、树状数组 |
| 每次区间加都逐个元素修改 | $O(nq)$ | 差分、线段树 |
| 双重循环找两数关系 | $O(n^2)$ | 排序、哈希、双指针 |
| 每次判断是否出现都扫描数组 | $O(nq)$ | set、unordered_set、排序后二分 |
| 每次找最大值都扫描 | $O(nq)$ | priority_queue、set、预处理 |
| 枚举所有子矩阵 | $O(n^2m^2)$ 或更高 | 二维前缀和 |
| 图上每对点都 BFS | 点数大时超时 | 看是否只需单源；正权用 Dijkstra |
| 字符串反复拼接或插入 | 长字符串会很慢 | 线性扫描、vector 存片段 |

## 6.7 前缀和

### 6.7.1 识别信号

题面出现：

- 多次查询区间和。
- 查询前 $k$ 项和。
- 子数组和、某段累计值。
- 数组不修改，或者所有查询都在修改之前/之后。

### 6.7.2 一维前缀和模板

使用 1 下标最稳：

```
int n, q;
cin >> n >> q;
vector<long long> pre(n + 1, 0);

for (int i = 1; i <= n; i++) {
    long long x;
    cin >> x;
    pre[i] = pre[i - 1] + x;
}

while (q--) {
    int l, r;
    cin >> l >> r;
    cout << pre[r] - pre[l - 1] << '\n';
}
```

### 6.7.3 0 下标前缀和

如果原数组是 0 下标，也可以让 pre[i+1] 表示前 $i+1$ 个数之和：

```
vector<int> a(n);
vector<long long> pre(n + 1, 0);
for (int i = 0; i < n; i++) {
    cin >> a[i];
    pre[i + 1] = pre[i] + a[i];
}

// sum of a[l..r], 0-indexed
long long ans = pre[r + 1] - pre[l];
```

### 6.7.4 二维前缀和

适合多次查询子矩阵和。

```
int n, m, q;
cin >> n >> m >> q;
vector<vector<long long>> pre(n + 1, vector<long long>(m + 1, 0));

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        long long x;
        cin >> x;
        pre[i][j] = pre[i - 1][j] + pre[i][j - 1]
                  - pre[i - 1][j - 1] + x;
    }
```

```
}

while (q--) {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    long long ans = pre[x2][y2] - pre[x1 - 1][y2]
                - pre[x2][y1 - 1] + pre[x1 - 1][y1 - 1];
    cout << ans << '\n';
}
```

### 6.7.5 前缀和易错点

- 查询端点是否从 1 开始。
- pre[0] 必须是 0。
- 区间公式是一维两项、二维四项。
- 和可能溢出，使用 long long。
- 如果中途有修改，普通前缀和通常不适用。

## 6.8 前后缀预处理

### 6.8.1 识别信号

题面出现：

- 删除一个元素后，问剩余部分的最大、最小、和。
- 对每个位置，问它左边和右边的信息。
- 需要快速知道前 $i$ 个的最大值、后 $i$ 个的最大值。

### 6.8.2 前缀最大和后缀最大

```
int n;
cin >> n;
vector<int> a(n + 1);
for (int i = 1; i <= n; i++) cin >> a[i];

vector<int> preMax(n + 2, -1000000000);
vector<int> sufMax(n + 2, -1000000000);

for (int i = 1; i <= n; i++) {
    preMax[i] = max(preMax[i - 1], a[i]);
}
for (int i = n; i >= 1; i--) {
    sufMax[i] = max(sufMax[i + 1], a[i]);
}
```

```
15
16 // max value except position i
17 for (int i = 1; i <= n; i++) {
18     int best = max(preMax[i - 1], sufMax[i + 1]);
19     cout << best << '\n';
20 }
```

### 6.8.3 前后缀易错点

- 数组多开两位，方便处理 i-1 和 i+1。
- 初始值要根据题意设置，最大值用很小值，最小值用很大值。
- 如果是求和，使用 long long。
- 删除第一个或最后一个元素时，一边为空，要提前定义空边的值。

## 6.9 差分

### 6.9.1 识别信号

题面出现：

- 很多次区间加。
- 最后统一输出每个位置的值。
- 修改多，查询少或只在最后查询。

### 6.9.2 一维差分模板

区间 $[l, r]$ 加 $v$：

```
1  int n, q;
2  cin >> n >> q;
3  vector<long long> diff(n + 2, 0);
4
5  while (q--) {
6      int l, r;
7      long long v;
8      cin >> l >> r >> v;
9      diff[l] += v;
10     diff[r + 1] -= v;
11 }
12
13 long long cur = 0;
14 for (int i = 1; i <= n; i++) {
15     cur += diff[i];
16     cout << cur << '\n';
17 }
```

如果原数组已有初值：

```
vector<long long> a(n + 1), diff(n + 2, 0);
for (int i = 1; i <= n; i++) cin >> a[i];
for (int i = 1; i <= n; i++) {
    diff[i] += a[i] - a[i - 1];
}
```

### 6.9.3　差分易错点

- 数组要开到 n+2，因为会访问 r+1。
- 差分适合“多次修改后统一求结果”。
- 如果每次修改后立刻查询区间和，普通差分不够。
- 修改值可能很大，用 long long。

### 6.9.4　二维差分入口

适合很多次对子矩阵加值，最后输出整个矩阵。

对矩形 $(x1, y1)$ 到 $(x2, y2)$ 加 $v$：

```
int n, m, q;
cin >> n >> m >> q;
vector<vector<long long>> diff(n + 2, vector<long long>(m + 2, 0));

while (q--) {
    int x1, y1, x2, y2;
    long long v;
    cin >> x1 >> y1 >> x2 >> y2 >> v;
    diff[x1][y1] += v;
    diff[x2 + 1][y1] -= v;
    diff[x1][y2 + 1] -= v;
    diff[x2 + 1][y2 + 1] += v;
}

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        diff[i][j] += diff[i - 1][j] + diff[i][j - 1]
                    - diff[i - 1][j - 1];
        cout << diff[i][j] << (j == m ? '\n' : ' ');
    }
}
```

**二维差分只适合修改后统一求结果。**如果修改和查询交替出现，要考虑二维树状数组或其他方法，B 题中通常会有更简单性质。

## 6.10 哈希统计

### 6.10.1 识别信号

题面出现：

- 统计每个数或字符串出现次数。
- 判断重复。
- 找出现最多的元素。
- 判断两个集合是否有交集。

### 6.10.2 值域小：计数数组

```cpp
int n;
cin >> n;
vector<int> cnt(100001, 0);
for (int i = 0; i < n; i++) {
    int x;
    cin >> x;
    cnt[x]++;
}
```

### 6.10.3 值域大：map

```cpp
int n;
cin >> n;
map<long long, int> cnt;
for (int i = 0; i < n; i++) {
    long long x;
    cin >> x;
    cnt[x]++;
}

for (auto [x, c] : cnt) {
    cout << x << ' ' << c << '\n';
}
```

### 6.10.4 字符串统计

```cpp
int n;
cin >> n;
unordered_map<string, int> cnt;
for (int i = 0; i < n; i++) {
    string s;
    cin >> s;
```

```
    cnt[s]++;
}
```

### 6.10.5 找出现次数最多

```
map<string, int> cnt;
// fill cnt

string best;
int bestCnt = -1;
for (auto [s, c] : cnt) {
    if (c > bestCnt) {
        bestCnt = c;
        best = s;
    }
}
cout << best << ' ' << bestCnt << '\n';
```

如果有并列规则，例如字典序最小，由 map 遍历天然按字典序升序。

## 6.11 余数和同余统计

### 6.11.1 识别信号

题面出现：

- 能否被 $k$ 整除。
- 两个数相加后是 $k$ 的倍数。
- 前缀和模 $k$ 相同。
- 按余数分类统计。

### 6.11.2 统计每个余数

```
int n, k;
cin >> n >> k;
vector<long long> cnt(k, 0);

for (int i = 0; i < n; i++) {
    long long x;
    cin >> x;
    int r = (int)(x % k);
    if (r < 0) r += k;
    cnt[r]++;
}
```

### 6.11.3 两数和能被 k 整除

```cpp
long long ans = 0;
for (int r = 0; r < k; r++) {
    int need = (k - r) % k;
    if (r < need) {
        ans += cnt[r] * cnt[need];
    } else if (r == need) {
        ans += cnt[r] * (cnt[r] - 1) / 2;
    }
}
cout << ans << '\n';
```

### 6.11.4 前缀和模 k

如果两个前缀和模 $k$ 相同，它们之间的区间和能被 $k$ 整除。

```cpp
int n, k;
cin >> n >> k;
vector<long long> cnt(k, 0);
cnt[0] = 1;

long long sum = 0;
long long ans = 0;
for (int i = 0; i < n; i++) {
    long long x;
    cin >> x;
    sum += x;
    int r = (int)(sum % k);
    if (r < 0) r += k;
    ans += cnt[r];
    cnt[r]++;
}
cout << ans << '\n';
```

**余数题注意：** 如果 $k$ 很大，不能开 `vector cnt(k)`，改用 `map` 或 `unordered_map`。

## 6.12 排序加扫描

### 6.12.1 识别信号

题面出现：

- 排序后相邻元素有关系。
- 需要合并区间。
- 需要按时间顺序处理事件。
- 需要按优先级选择。

### 6.12.2 排序后统计相同元素段

```cpp
sort(a.begin(), a.end());
int n = (int)a.size();
for (int i = 0; i < n; ) {
    int j = i;
    while (j < n && a[j] == a[i]) {
        j++;
    }
    int count = j - i;
    cout << a[i] << ' ' << count << '\n';
    i = j;
}
```

### 6.12.3 合并区间

输入若干区间，合并重叠部分：

```cpp
vector<pair<int, int>> segs;
sort(segs.begin(), segs.end());

vector<pair<int, int>> merged;
for (auto [l, r] : segs) {
    if (merged.empty() || l > merged.back().second) {
        merged.push_back({l, r});
    } else {
        merged.back().second = max(merged.back().second, r);
    }
}
```

如果题目认为相邻区间也可合并，例如 $[1,2]$ 和 $[3,4]$，条件要改成 `l > merged.back().second + 1`。

### 6.12.4 事件排序

当题目有“按时间发生”的操作，先把事件读入结构体，再排序处理。

```cpp
struct Event {
    int time;
    int type;
    int id;
};

vector<Event> events;

sort(events.begin(), events.end(), [](const Event& a, const Event& b) {
    if (a.time != b.time) return a.time < b.time;
    return a.type < b.type; // tie rule depends on statement
});
```

```
13
14 for (const Event& e : events) {
15     // process event
16 }
```

事件排序检查：

- 同一时间多个事件的处理顺序。
- 开始事件和结束事件谁先处理。
- 是否需要稳定排序。
- 时间和答案是否要 long long。

### 6.12.5 坐标离散化入口

当坐标很大，但只关心出现过的坐标，可以离散化。

识别信号：

- 坐标可达 $10^9$，不能直接开数组。
- 点的数量只有 $10^5$。
- 只需要比较大小、排序、相对位置。

模板：

```
1  vector<int> xs;
2  for (int x : original) {
3      xs.push_back(x);
4  }
5
6  sort(xs.begin(), xs.end());
7  xs.erase(unique(xs.begin(), xs.end()), xs.end());
8
9  auto getId = [&](int x) {
10     return (int)(lower_bound(xs.begin(), xs.end(), x) - xs.begin()) + 1;
11 };
12
13 int id = getId(original[0]);
```

**区间离散化要小心。**如果题目处理连续区间长度，不能只保存端点编号，还要考虑相邻坐标之间的真实长度。

## 6.13 堆和动态维护

### 6.13.1 priority_queue 识别信号

题面出现：

- 每次取当前最大或最小。

- 动态加入元素。
- 模拟优先级队列。
- 求前 $k$ 大、前 $k$ 小。

### 6.13.2 小根堆

```cpp
priority_queue<int, vector<int>, greater<int>> pq;
pq.push(5);
pq.push(2);
cout << pq.top() << '\n'; // 2
pq.pop();
```

### 6.13.3 保留最大的 k 个数

用小根堆保存当前最大的 $k$ 个数。

```cpp
int k;
cin >> k;
priority_queue<int, vector<int>, greater<int>> pq;

for (int x : a) {
    pq.push(x);
    if ((int)pq.size() > k) {
        pq.pop();
    }
}

cout << pq.top() << '\n'; // kth largest
```

### 6.13.4 set 动态维护

set 适合动态插入、删除、找前驱后继。

```cpp
set<int> s;
s.insert(10);
s.insert(20);
s.insert(30);

int x = 25;
auto it = s.lower_bound(x); // first >= x
if (it != s.end()) {
    cout << "next=" << *it << '\n';
}
if (it != s.begin()) {
    --it;
    cout << "prev=" << *it << '\n';
}
```

**迭代器小心：**移动迭代器前先判断是不是 `begin()` 或 `end()`。

### 6.13.5 map 动态维护

map 适合动态统计并按 key 有序遍历。

```
map<int, int> cnt;
cnt[x]++;

cnt[x]--;
if (cnt[x] == 0) {
    cnt.erase(x);
}
```

取最小 key 和最大 key：

```
int mn = cnt.begin()->first;
int mx = cnt.rbegin()->first;
```

**取 begin/rbegin 前确认 map 非空。**

## 6.14 基础贪心

### 6.14.1 识别信号

B 题中的贪心通常比较直接，常见信号：

- 尽量多选一些区间、任务、物品。
- 每次选择当前最早结束、最小代价、最大收益。
- 排序后从前往后扫描。
- 题目问最少次数覆盖、最多不冲突任务。

### 6.14.2 最多不重叠区间

按右端点从小到大排序，每次选最早结束的区间。

```
vector<pair<int, int>> segs; // l, r
sort(segs.begin(), segs.end(), [](auto a, auto b) {
    if (a.second != b.second) return a.second < b.second;
    return a.first < b.first;
});

int ans = 0;
int lastEnd = -1000000000;
for (auto [l, r] : segs) {
    if (l > lastEnd) { // use l >= lastEnd if endpoints can touch
        ans++;
        lastEnd = r;
    }
```

```
}
cout << ans << '\n';
```

### 6.14.3 贪心易错点

- 先确认排序关键字，是按左端点、右端点、代价还是收益。
- 区间相邻是否算冲突：`l > lastEnd` 还是 `l >= lastEnd`。
- 贪心不是看起来合理就一定对。B 题若想不出证明，至少用小样例反驳自己。

## 6.15 单调栈和单调队列入口

### 6.15.1 单调栈识别信号

题面出现：

- 找每个元素左边或右边第一个更大/更小的元素。
- 柱状图、温度、最近更高。
- 暴力向左/向右扫描会超时。

### 6.15.2 右边第一个更大元素

```
int n;
cin >> n;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];

vector<int> ans(n, -1);
stack<int> st; // indices, a[st] decreasing

for (int i = 0; i < n; i++) {
    while (!st.empty() && a[i] > a[st.top()]) {
        ans[st.top()] = i;
        st.pop();
    }
    st.push(i);
}
```

### 6.15.3 单调队列识别信号

题面出现固定长度滑动窗口最大值/最小值。

```
int n, k;
cin >> n >> k;
vector<int> a(n);
for (int i = 0; i < n; i++) cin >> a[i];

```

```
deque<int> dq; // indices, values decreasing
for (int i = 0; i < n; i++) {
    while (!dq.empty() && dq.front() <= i - k) dq.pop_front();
    while (!dq.empty() && a[dq.back()] <= a[i]) dq.pop_back();
    dq.push_back(i);

    if (i >= k - 1) {
        cout << a[dq.front()] << '\n'; // max of window
    }
}
```

**单调结构新手建议：**如果没练过，不要在考场第一次使用。可以作为 B/C/D 题冲分模板，但必须提前练。

## 6.16 双指针入门

### 6.16.1 识别信号

题面出现：

- 数组已排序，找一对数。
- 连续区间满足某个条件。
- 左右端点移动，区间和或区间长度变化。

### 6.16.2 排序后找两数和

```
sort(a.begin(), a.end());
int l = 0, r = n - 1;
bool ok = false;

while (l < r) {
    long long sum = 1LL * a[l] + a[r];
    if (sum == target) {
        ok = true;
        break;
    } else if (sum < target) {
        l++;
    } else {
        r--;
    }
}

cout << (ok ? "YES" : "NO") << '\n';
```

### 6.16.3 正数数组的连续区间

如果数组元素都是非负数，可以用滑动窗口。

```cpp
int l = 0;
long long sum = 0;
int best = 0;

for (int r = 0; r < n; r++) {
    sum += a[r];
    while (sum > limit && l <= r) {
        sum -= a[l];
        l++;
    }
    best = max(best, r - l + 1);
}
```

**滑动窗口前提：**通常要求元素非负或条件具有单调性。若有负数，窗口移动可能失效。

## 6.17 二分与二分答案

### 6.17.1 普通二分查找

数组必须有序。

```cpp
sort(a.begin(), a.end());
auto it = lower_bound(a.begin(), a.end(), x);
if (it != a.end()) {
    cout << *it << '\n';
}
```

### 6.17.2 二分答案识别

题面出现：

- 最大化最小值。
- 最小化最大值。
- 求最小的可行时间、容量、代价。
- 给定一个答案 $mid$，容易判断是否可行。

### 6.17.3 二分答案模板

找最小可行值：

```cpp
bool check(long long mid) {
    // return true if mid is feasible
    return true;
}

long long l = 0, r = 1000000000000000000LL;
while (l < r) {
```

```
    long long mid = l + (r - l) / 2;
    if (check(mid)) {
        r = mid;
    } else {
        l = mid + 1;
    }
}
cout << l << '\n';
```

找最大可行值：

```
long long l = 0, r = 10000000000000000000LL;
while (l < r) {
    long long mid = l + (r - l + 1) / 2;
    if (check(mid)) {
        l = mid;
    } else {
        r = mid - 1;
    }
}
cout << l << '\n';
```

**二分答案关键：**不是所有题都能二分。必须满足可行性具有单调性。

## 6.18 BFS 入门

### 6.18.1 识别信号

题面出现：

- 最少步数。
- 迷宫、网格移动。
- 每次操作代价相同。
- 从起点扩展到终点。

### 6.18.2 网格 BFS 模板

```
int n, m;
cin >> n >> m;
vector<string> g(n);
for (int i = 0; i < n; i++) cin >> g[i];

vector<vector<int>> dist(n, vector<int>(m, -1));
queue<pair<int, int>> q;

int sx, sy, tx, ty;
cin >> sx >> sy >> tx >> ty;
```

```
11
12 dist[sx][sy] = 0;
13 q.push({sx, sy});
14
15 int dx[4] = {-1, 0, 1, 0};
16 int dy[4] = {0, 1, 0, -1};
17
18 while (!q.empty()) {
19     auto [x, y] = q.front();
20     q.pop();
21
22     for (int d = 0; d < 4; d++) {
23         int nx = x + dx[d];
24         int ny = y + dy[d];
25         if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
26         if (g[nx][ny] == '#') continue;
27         if (dist[nx][ny] != -1) continue;
28
29         dist[nx][ny] = dist[x][y] + 1;
30         q.push({nx, ny});
31     }
32 }
33
34 cout << dist[tx][ty] << '\n';
```

### 6.18.3 BFS 易错点

- 坐标输入是 0 下标还是 1 下标。
- 障碍字符是什么。
- 起点是否可能等于终点。
- 应在入队时标记访问，避免重复入队。
- BFS 适合每步代价相同；边权不同要考虑 Dijkstra。

### 6.18.4 多源 BFS

如果有多个起点，且要求每个点到最近起点的距离，把所有起点同时入队。

```
1 vector<vector<int>> dist(n, vector<int>(m, -1));
2 queue<pair<int, int>> q;
3
4 for (auto [x, y] : starts) {
5     dist[x][y] = 0;
6     q.push({x, y});
7 }
8
9 while (!q.empty()) {
```

```
    auto [x, y] = q.front();
    q.pop();
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d];
        int ny = y + dy[d];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (dist[nx][ny] != -1) continue;
        dist[nx][ny] = dist[x][y] + 1;
        q.push({nx, ny});
    }
}
```

识别信号：

- 多个火源、多个出口、多个起点。
- 求离最近特殊点的距离。
- 每一步代价相同。

## 6.19 DFS 和连通块

### 6.19.1 识别信号

题面出现：

- 统计连通块数量。
- 找一片区域大小。
- 网格中相邻的同类格子。
- 图中从某点能到哪些点。

### 6.19.2 网格 DFS

```
int n, m;
vector<string> g;
vector<vector<int>> vis;
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};

int dfs(int x, int y) {
    vis[x][y] = 1;
    int area = 1;
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d];
        int ny = y + dy[d];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (vis[nx][ny]) continue;
        if (g[nx][ny] == '#') continue;
```

```
        area += dfs(nx, ny);
    }
    return area;
}
```

统计连通块：

```
int components = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        if (!vis[i][j] && g[i][j] != '#') {
            components++;
            int area = dfs(i, j);
        }
    }
}
```

**递归深度风险：**如果网格很大，递归 DFS 可能爆栈。B 题中可改用 BFS 更稳。

## 6.20 Floyd 入门

### 6.20.1 识别信号

- 问任意两点之间最短路。
- 点数较小，例如 $n \le 300$ 或 $n \le 500$。
- 图可能是稠密图。

### 6.20.2 Floyd 模板

```
const long long INF = (long long)4e18;

int n, m;
cin >> n >> m;
vector<vector<long long>> dist(n + 1, vector<long long>(n + 1, INF));

for (int i = 1; i <= n; i++) {
    dist[i][i] = 0;
}

for (int i = 0; i < m; i++) {
    int u, v;
    long long w;
    cin >> u >> v >> w;
    dist[u][v] = min(dist[u][v], w);
    dist[v][u] = min(dist[v][u], w); // remove if directed
}

```

```
for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dist[i][k] == INF || dist[k][j] == INF) continue;
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

### 6.20.3 Floyd 易错点

- 三层循环顺序必须是 k,i,j。
- 无向图要双向赋值，有向图不要。
- 多条边取最小边权。
- 点数大时不能用 Floyd。

## 6.21 Dijkstra 入门

### 6.21.1 识别信号

- 正权图最短路。
- 边权不是全 1。
- 点和边较多，不能 Floyd。
- 单源最短路：从一个起点到其他点。

### 6.21.2 Dijkstra 模板

```
struct Edge {
    int to;
    long long w;
};

const long long INF = (long long)4e18;

int n, m, s;
cin >> n >> m >> s;
vector<vector<Edge>> g(n + 1);

for (int i = 0; i < m; i++) {
    int u, v;
    long long w;
    cin >> u >> v >> w;
    g[u].push_back({v, w});
    g[v].push_back({u, w}); // remove if directed
```

```
}

vector<long long> dist(n + 1, INF);
priority_queue<pair<long long, int>,
               vector<pair<long long, int>>,
               greater<pair<long long, int>>> pq;

dist[s] = 0;
pq.push({0, s});

while (!pq.empty()) {
    auto [d, u] = pq.top();
    pq.pop();
    if (d != dist[u]) continue;

    for (auto e : g[u]) {
        if (dist[e.to] > dist[u] + e.w) {
            dist[e.to] = dist[u] + e.w;
            pq.push({dist[e.to], e.to});
        }
    }
}
```

### 6.21.3 Dijkstra 易错点

- 只适用于非负边权。存在负边权时不能直接用。
- 距离用 long long。
- 无向图双向加边，有向图单向加边。
- 堆中可能有过期状态，要跳过。
- 如果所有边权都是 1，用 BFS 更简单。

## 6.22 并查集入门

### 6.22.1 识别信号

题面出现：

- 合并两个集合。
- 判断两个元素是否属于同一组。
- 连通块数量。
- 朋友关系、同类关系、网络连通。

### 6.22.2 并查集模板

```cpp
struct DSU {
    vector<int> parent, sz;

    DSU(int n) {
        parent.resize(n + 1);
        sz.assign(n + 1, 1);
        for (int i = 1; i <= n; i++) parent[i] = i;
    }

    int find(int x) {
        if (parent[x] == x) return x;
        return parent[x] = find(parent[x]);
    }

    bool unite(int a, int b) {
        int ra = find(a);
        int rb = find(b);
        if (ra == rb) return false;
        if (sz[ra] < sz[rb]) swap(ra, rb);
        parent[rb] = ra;
        sz[ra] += sz[rb];
        return true;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }
};
```

## 6.23 拓扑排序入口

### 6.23.1 识别信号

题面出现：

- 任务必须按先后顺序完成。
- A 必须在 B 前面。
- 课程依赖、工程依赖。
- 判断是否存在合法顺序。

### 6.23.2 拓扑排序模板

```cpp
int n, m;
cin >> n >> m;
```

```
vector<vector<int>> g(n + 1);
vector<int> indeg(n + 1, 0);

for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v; // u before v
    g[u].push_back(v);
    indeg[v]++;
}

queue<int> q;
for (int i = 1; i <= n; i++) {
    if (indeg[i] == 0) q.push(i);
}

vector<int> order;
while (!q.empty()) {
    int u = q.front();
    q.pop();
    order.push_back(u);

    for (int v : g[u]) {
        indeg[v]--;
        if (indeg[v] == 0) q.push(v);
    }
}

if ((int)order.size() == n) {
    cout << "YES\n";
} else {
    cout << "NO\n"; // has cycle
}
```

### 6.23.3 拓扑排序易错点

- 边的方向：如果 A 在 B 前，应连 A 到 B。
- 入度为 0 的点一开始全部入队。
- 输出顺序是否要求字典序最小。若要求，队列换成小根堆。
- 若排序结果少于 $n$ 个点，说明有环。

## 6.24 简单 DP 入口

### 6.24.1 识别信号

题面出现：

- 求最优值：最大、最小、方案数。
- 当前结果依赖前一个或前几个位置。
- 选择或不选择某个物品。
- 容量、价值、花费、恰好装满。

### 6.24.2 线性 DP 示例

例如每一步只能从前一步或前两步转移：

```cpp
int n;
cin >> n;
vector<long long> dp(n + 1, 0);
dp[0] = 1;
dp[1] = 1;
for (int i = 2; i <= n; i++) {
    dp[i] = dp[i - 1] + dp[i - 2];
}
cout << dp[n] << '\n';
```

### 6.24.3 01 背包入口

每个物品只能选一次：

```cpp
int n, V;
cin >> n >> V;
vector<int> w(n + 1), val(n + 1);
for (int i = 1; i <= n; i++) {
    cin >> w[i] >> val[i];
}

vector<int> dp(V + 1, 0);
for (int i = 1; i <= n; i++) {
    for (int j = V; j >= w[i]; j--) {
        dp[j] = max(dp[j], dp[j - w[i]] + val[i]);
    }
}
cout << dp[V] << '\n';
```

### 6.24.4 完全背包入口

每个物品可以选无限次，容量正序。

```cpp
int n, V;
cin >> n >> V;
vector<int> w(n + 1), val(n + 1);
for (int i = 1; i <= n; i++) {
    cin >> w[i] >> val[i];
```

```
}

vector<int> dp(V + 1, 0);
for (int i = 1; i <= n; i++) {
    for (int j = w[i]; j <= V; j++) {
        dp[j] = max(dp[j], dp[j - w[i]] + val[i]);
    }
}
cout << dp[V] << '\n';
```

### 6.24.5 DP 易错点

- 先定义 dp[i] 表示什么。
- 写出初始状态。
- 写出转移方程。
- 01 背包容量倒序，完全背包容量正序。
- 方案数可能很大，要看是否取模。

## 6.25 基础数学入口

### 6.25.1 gcd 和 lcm

```
long long g = gcd(a, b);
long long l = a / g * b;
```

**先除后乘，降低溢出风险。**

### 6.25.2 快速幂

如果题目要求 $a^b \bmod mod$：

```
long long modPow(long long a, long long b, long long mod) {
    long long res = 1 % mod;
    a %= mod;
    while (b > 0) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
```

### 6.25.3 判断素数

单个数判断：

```cpp
bool isPrime(long long x) {
    if (x < 2) return false;
    for (long long d = 2; d * d <= x; d++) {
        if (x % d == 0) return false;
    }
    return true;
}
```

### 6.25.4 枚举约数

```cpp
vector<long long> divisors;
for (long long d = 1; d * d <= x; d++) {
    if (x % d == 0) {
        divisors.push_back(d);
        if (d != x / d) divisors.push_back(x / d);
    }
}
sort(divisors.begin(), divisors.end());
```

**数学题先看范围。**如果要对很多个数判断素数，可能需要筛法；如果只是少量数，试除通常够用。使用：

```cpp
int n, m;
cin >> n >> m;
DSU dsu(n);

while (m--) {
    int op, a, b;
    cin >> op >> a >> b;
    if (op == 1) dsu.unite(a, b);
    else cout << (dsu.same(a, b) ? "YES" : "NO") << '\n';
}
```

## 6.26 B 题提交前检查

- 是否估算过复杂度。
- 是否把暴力写到了大数据上。
- 前缀和、差分的下标是否统一。
- 求和、答案、距离是否用 `long long`。
- 排序比较器是否正确。
- `lower_bound` 前是否排序。
- BFS 是否在入队时标记。

- 并查集是否初始化所有点。
- 多组数据是否清空容器。

## 6.27 B 题冲分判断表

当你已经有一个暴力思路，但担心超时时，用这张表判断下一步。

| 当前情况 | 优先尝试 | 原因 |
| --- | --- | --- |
| 暴力是两层循环找一对数 | 排序、哈希、双指针 | 二元关系常能降到 $O(n\log n)$ 或 $O(n)$。 |
| 暴力是每次求区间和 | 前缀和 | 静态区间查询标准做法。 |
| 暴力是每次区间加 | 差分 | 最后统一查询时最简单。 |
| 暴力是每次找最大/最小 | 堆或 set | 动态维护极值。 |
| 坐标太大但点数不多 | 离散化 | 把大坐标压缩成小编号。 |
| 规则按时间发生 | 事件排序 | 统一按时间顺序处理。 |
| 左边右边都要快速知道信息 | 前后缀预处理 | 删除一个元素或分割数组常见。 |
| 涉及整除、余数相同 | 余数统计 | 同余关系常能转成计数。 |
| 最多不冲突任务或区间 | 贪心排序 | 常见按右端点排序。 |
| 每个点找旁边第一个更大/更小 | 单调栈 | 避免每次向旁边扫。 |
| 滑动窗口最大/最小 | 单调队列 | 固定窗口极值。 |
| 网格最短步数 | BFS | 每步代价相同。 |
| 网格连通区域 | DFS/BFS | 统计区域或连通块。 |
| 图边权为 1 | BFS | 比 Dijkstra 简单。 |
| 图边权为正 | Dijkstra | 单源正权最短路。 |
| 点数小且任意两点 | Floyd | 写法短，适合小图。 |
| 合并和查询同组 | 并查集 | 连通性标准做法。 |
| 依赖顺序 | 拓扑排序 | 入度为 0 的点先处理。 |

## 6.28 B 题常见边界库

| 题型 | 边界样例 | 检查目标 |
| --- | --- | --- |
| 前缀和 | $l=1$，$r=n$，$l=r$ | 端点公式。 |
| 二维前缀和 | 单行、单列、整个矩阵、单个格子 | 四项公式。 |
| 差分 | 修改到 $r=n$，只改一个点 | `r+1` 和数组大小。 |
| 排序 | 全相等、逆序、相同主关键字 | 比较器。 |

| | | |
|---|---|---|
| 哈希统计 | 没有重复、全部重复、key 很大 | 容器选择。 |
| 双指针 | 无解、两个数刚好在两端、有重复 | 指针移动。 |
| 二分答案 | 最小值可行、最大值不可行、边界答案 | `check` 单调性。 |
| BFS | 起点等于终点、无路、全障碍附近 | 距离初始化。 |
| DFS 连通块 | 全障碍、全可走、单个格子 | 访问标记。 |
| 并查集 | 自己和自己合并、重复合并 | find/unite。 |
| Dijkstra | 不连通、多条边、起点到自己 | INF 和取最小边。 |
| 余数统计 | 负数、$k = 1$、所有余数相同 | 取模修正和计数公式。 |
| 贪心 | 区间端点相同、相邻是否冲突 | 排序关键字和边界条件。 |
| 单调栈 | 全递增、全递减、全相等 | 比较符号是 `<` 还是 `<=`。 |

## 6.29 B 题本地调试流程

### 6.29.1 先用暴力对拍思路

如果你写了优化算法，但不确定是否正确，可以在本地用小数据和暴力对比。考场时间紧时不一定完整写对拍，但可以用这个思想手工造样例。

- $n$ 取 3 到 6。
- 手算所有操作结果。
- 用暴力程序或草稿验证优化结果。
- 专门造边界：重复、空、单点、全相等。

### 6.29.2 复杂度复查

提交前问：

- 有没有隐藏的双重循环。
- `map` 操作是否在循环里被调用很多次，是否仍可接受。
- 排序是否放在循环内部。若是，通常危险。
- BFS/Dijkstra 是否会重复跑很多次。
- 字符串拼接是否在大循环里反复发生。

### 6.29.3 状态清空

多组数据时，尤其注意：

```
g.assign(n + 1, vector<int>());
dist.assign(n + 1, INF);
vis.assign(n + 1, 0);
```

```
cnt.clear();
while (!q.empty()) q.pop();
```

**队列、优先队列没有 clear。**最简单的方法是在每组数据内部重新定义它。

## 6.30 B 题卡住时的部分分策略

1. 如果不会优化，先写 $O(n^2)$ 暴力，确认题意。
2. 如果有多档数据，先保证小数据正确。
3. 如果是区间题，先写单次查询或无修改版本。
4. 如果是图题，先写 BFS/DFS 处理无权或小图。
5. 如果是复杂排序规则，先保证主关键字正确，再处理并列规则。

## 6.31 B 题手写模板清单

这些模板建议在考前单独练习到“看着能快速写对”。

| 模板 | 熟练要求 | 主要用途 |
| --- | --- | --- |
| 一维前缀和 | 必须会默写 | 静态区间和。 |
| 二维前缀和 | 照着能写 | 子矩阵和。 |
| 一维差分 | 必须会默写 | 区间加，最后查询。 |
| 二维差分 | 照着能写 | 矩形加，最后输出。 |
| map/unordered_map 统计 | 必须会默写 | 频率统计。 |
| 排序加结构体比较器 | 必须会默写 | 排名、事件、贪心。 |
| 双指针 | 照着能写 | 排序后找对、滑动窗口。 |
| 二分答案 | 照着能写 | 最大化最小值、最小化最大值。 |
| 前后缀预处理 | 照着能写 | 删除一个元素、左右信息。 |
| 余数统计 | 照着能写 | 整除、同余、区间和取模。 |
| 基础贪心 | 照着能写 | 区间选择、任务安排。 |
| 单调栈/队列 | 了解并练过再用 | 最近更大、滑动窗口极值。 |
| 网格 BFS | 必须照着能写 | 最短步数。 |
| 并查集 | 照着能写 | 连通性。 |
| 拓扑排序 | 照着能写 | 依赖顺序。 |
| Dijkstra | 照着能写 | 正权最短路。 |

# 第七章　C 题决策树与大模拟方法

## 7.1　C 题定位

C 题通常是 CSP 的分水岭。它不一定算法很难，但题面往往更长、规则更多、状态更多，代码量明显增加。

- 建议目标时间：60–100 分钟。
- 目标：能满分则满分；若细节太多，至少拿到主要规则分。
- 高频主题：大模拟、字符串解析、状态维护、网格搜索、图论基础、数据结构基础、简单 DP。
- 最大风险：读题不完整、状态设计混乱、把所有逻辑堆进 `main`。

**C 题第一原则：**先拆题，再写代码。题面越长，越不能急着敲。

## 7.2　C 题开题流程

1. 快速浏览题面，判断是大模拟还是算法题。
2. 圈出所有对象：人、文件、任务、节点、格子、命令、事件。
3. 圈出所有状态：位置、时间、分数、开关、权限、颜色、方向。
4. 圈出所有操作：新增、删除、查询、移动、更新、比较。
5. 圈出优先级和并列规则。
6. 设计结构体和函数。
7. 先实现最基础规则，通过样例的一部分。
8. 再逐条补特殊规则。

## 7.3　C 题读题三遍法

C 题不要试图一遍读懂所有细节。建议固定读三遍。

### 7.3.1　第一遍：只判断题型

第一遍快速读，不抠细节，只回答：

- 是大模拟，还是算法题？
- 输入是一次性数据，还是很多命令？

- 输出是每次查询输出，还是最后统一输出?
- 数据范围是否明显要求优化?

第一遍结束后，如果你还不知道题目在讲什么，先看样例解释。

### 7.3.2 第二遍：圈对象、状态、操作

第二遍要动笔。把题目中的名词和动词分开。

| 类别 | 怎么找 | 例子 |
| --- | --- | --- |
| 对象 | 被操作的名词 | 用户、文件、进程、任务、车辆、节点、格子。 |
| 属性 | 对象身上的数据 | 编号、名字、位置、时间、状态、得分。 |
| 操作 | 题目中的动词或命令 | 新增、删除、移动、查询、合并、修改。 |
| 规则 | 操作发生时的限制 | 如果不存在则忽略;如果冲突按优先级。 |
| 输出 | 什么时候打印 | 每次 query、最后统计、发生错误时。 |

### 7.3.3 第三遍：找坑点

第三遍只找容易错的细节:

- 相同时间、相同分数、相同优先级怎么处理。
- 编号从 0 还是 1 开始。
- 找不到对象时怎么处理。
- 重复添加时怎么处理。
- 删除后是否还能查询。
- 最后一条命令或最后一个事件是否要特殊处理。
- 输出顺序按输入顺序、编号顺序、字典序还是时间顺序。

## 7.4 从题面到代码的转换

### 7.4.1 对象变成 struct

题目中的每类对象，优先考虑一个结构体。

```
struct Task {
    int id;
    int startTime;
    int endTime;
    int priority;
    int status;
```

```
};
```

### 7.4.2 编号查找用 map 或 vector

如果编号是 1 到 $n$：

```
vector<Task> tasks(n + 1);
```

如果编号很大或是字符串：

```
map<string, Task> tasks;
unordered_map<int, Task> tasksById;
```

### 7.4.3 命令变成函数

每个命令单独写函数：

```
void addTask(int id, int priority) {
    // add or update
}

void deleteTask(int id) {
    // delete if exists
}

void queryTask(int id) {
    // print answer
}
```

### 7.4.4 规则变成 if

题目里的每条规则，都应该在代码中有对应的 `if`。

```
if (!tasks.count(id)) {
    cout << "Not Found\n";
    return;
}

if (tasks[id].status == FINISHED) {
    cout << "Finished\n";
    return;
}
```

**不要把多个规则揉成一行复杂表达式。**C 题的代码要好查错，宁可多写几行。

## 7.5 状态维护和容器选择

### 7.5.1 容器选择表

| 需求 | 推荐容器 | 原因 |
|---|---|---|
| 编号是 1 到 $n$ | `vector<Obj>` | 直接下标访问，最快。 |
| 编号很大但唯一 | `unordered_map<int,Obj>` | 平均快速查找。 |
| 编号是字符串 | `map<string,Obj>` 或 `unordered_map` | 字符串到对象映射。 |
| 需要按 key 有序输出 | `map` | 自动按 key 排序。 |
| 需要判断是否存在 | `set` / `unordered_set` | 查重。 |
| 需要动态最小/最大 | `set` / `priority_queue` | 维护极值。 |
| 需要前驱后继 | `set` | `lower_bound`。 |
| 需要按进入顺序处理 | `queue` | FIFO。 |
| 需要最近使用顺序 | `list + map` | LRU。 |
| 需要父子层级 | `vector<Node>` + child | 树结构。 |

### 7.5.2 操作复杂度估算

| 容器 | 常见复杂度 | 注意 |
|---|---|---|
| `vector` 下标访问 | $O(1)$ | 中间插入删除慢。 |
| `map` 查找插入 | $O(\log n)$ | 有序但常数较大。 |
| `unordered_map` 查找插入 | 平均 $O(1)$ | 无序，最坏情况不稳定。 |
| `set` 查找插入删除 | $O(\log n)$ | 可找前驱后继。 |
| `priority_queue` 取最优 | $O(\log n)$ 插入删除堆顶 | 不能直接删除中间元素。 |
| `queue` 进出队 | $O(1)$ | 只能访问队首。 |
| `list` 插入删除 | 已有迭代器时 $O(1)$ | 不能随机访问。 |

### 7.5.3 C 题容器选择错误信号

- 每次命令都全表扫描，且命令数很大：应该用 map/set/hash。
- 每次取最大值都排序：应该用 set 或堆。
- 每次删除 vector 中间元素：可能需要标记删除、list 或 set。
- 需要有序输出却用了 unordered_map：输出顺序会错。
- priority_queue 里元素更新了但旧值还在：需要懒删除。

## 7.6 C 题总决策树

| 题面信号 | 优先怀疑 | 首选方法 |
|---|---|---|
| 题面很长，规则很多 | 大模拟 | 结构体 + 函数拆解 |

| | | |
|---|---|---|
| 输入是一串命令 | 命令解析模拟 | stringstream / split / 状态表 |
| 按时间发生事件 | 事件模拟 | 事件排序 / 优先队列 |
| 对象有多种状态 | 状态机模拟 | enum/int 表示状态 |
| 文件路径、配置、表达式 | 字符串解析 | 逐字符扫描 / 栈 |
| 网格、地图、移动 | 网格模拟或 BFS/DFS | 方向数组、vis、dist |
| 图、连通、最短 | 图论 | BFS / Dijkstra / 并查集 |
| 区间修改查询 | 数据结构 | 前缀和 / 差分 / 树状数组 / 线段树 |
| 求最优值且有阶段 | DP | 先定义状态 |

## 7.7 C 题考法地图

这张表用于快速判断 C 题该从哪个方向切入。

| 题目长相 | 本质 | 先翻哪里 |
|---|---|---|
| 很多命令，每条命令改变系统状态 | 命令模拟 | 命令解析、状态设计、函数化 |
| 按时间发生、开始结束、排队 | 事件模拟 | 事件排序、优先级、队列模拟 |
| 字符串里有路径、参数、配置 | 字符串解析 | split、key=value、路径规范化 |
| 输入像 URL、路由、参数 | 路由匹配 | URL 路由和参数解析 |
| 输入像 JSON、配置、嵌套括号 | 结构解析 | 递归解析、栈、逐字符扫描 |
| 输入像 Markdown、HTML、轻量标记 | 标记语言模拟 | 行扫描、状态切换 |
| 有用户、角色、权限、规则 | 规则匹配 | 权限和规则匹配 |
| 有表格、矩阵、坐标变换 | 表格模拟 | 矩阵操作、行列映射 |
| 有缓存、队列、窗口、淘汰 | 数据结构模拟 | 队列、set、map、懒删除 |
| 有目录、树、层级关系 | 树结构模拟 | 目录树、父子关系、DFS |
| 有地图并带钥匙/状态 | 状态搜索 | 带状态 BFS |

## 7.8 C 题细分决策树

### 7.8.1 第一问：这是大模拟还是算法题

| 题面特征 | 判断 | 下一步 |
|---|---|---|

| | | |
|---|---|---|
| 规则很多、命令很多、对象很多 | 大模拟 | 拆对象、状态、命令 |
| 题面不长但数据范围大 | 算法题 | 回到 B/D 算法决策树 |
| 有地图和最短步数 | 搜索题 | BFS / 状态 BFS |
| 有依赖关系 | 图论题 | 拓扑 / 最短路 / 并查集 |
| 有复杂字符串格式 | 解析题 | 字符串解析决策树 |
| 有大量修改查询 | 数据结构题 | 前缀/差分/树状数组/线段树 |

### 7.8.2 大模拟细分树

| 题面信号 | 类型 | 首选结构 |
|---|---|---|
| 每行一个操作名 | 命令系统 | `handleCommand` 分发 |
| 对象有生命周期 | 增删改查模拟 | `map<int,Obj>` 或 `vector<Obj>` |
| 对象有多个状态 | 状态机 | 状态常量 + 转移函数 |
| 按时间发生 | 事件模拟 | `Event` + 排序 |
| 每次取最优对象 | 优先级模拟 | 堆 / set / 懒删除堆 |
| 行列、表格、棋盘 | 表格模拟 | 二维数组 / 行列映射 |
| 路径、目录、父子层级 | 树结构模拟 | 节点数组 + child map |
| 规则匹配、权限判断 | 规则系统 | `Rule` + `match` |

### 7.8.3 字符串解析细分树

| 字符串形态 | 优先方法 | 小心 |
|---|---|---|
| 按空格分隔 | `stringstream` | 前面是否要 `ignore` |
| 按固定字符分隔 | `split` | 连续分隔符 |
| `key=value` | `find + substr` | 值是否含空格 |
| 路径 `/a/b/c` | 路径 split / normalize | 根目录、`..` |
| 括号匹配 | 栈 | 空栈和最后剩余 |
| 表达式求值 | 扫描 / 递归解析 | 优先级、括号 |
| URL 参数 | 先按 `?`，再按 `&` | 参数为空 |
| 嵌套 JSON 类 | 逐字符扫描 / 栈 | 先利用题目限制 |

### 7.8.4 图网格细分树

| 题面信号 | 类型 | 首选模板 |
|---|---|---|
| 最少步数、每步代价相同 | 无权最短路 | BFS |
| 多个起点最近距离 | 多源 BFS | 所有起点入队 |

| | | |
|---|---|---|
| 有钥匙、方向、模式等额外状态 | 状态 BFS | `dist[x][y][state]` |
| 统计区域大小 | 连通块 | DFS / BFS |
| 正权边最短路 | 正权图 | Dijkstra |
| 只问是否连通 | 连通性 | 并查集 / DFS |
| 任务依赖顺序 | DAG | 拓扑排序 |

## 7.9 大模拟总方法

### 7.9.1 大模拟识别信号

- 题面长度明显长。
- 有很多“如果……否则……”规则。
- 输入包含多条命令。
- 有多个对象，每个对象有状态。
- 输出要求根据最终状态或每次查询得到。

### 7.9.2 大模拟代码骨架

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

struct Item {
    int id;
    string name;
    int state;
};

int n, m;
vector<Item> items;

void readInput() {
    cin >> n >> m;
    items.resize(n);
    for (int i = 0; i < n; i++) {
        cin >> items[i].id >> items[i].name;
        items[i].state = 0;
    }
}

void handleCommand(const string& cmd) {
    if (cmd == "ADD") {
        // handle add
```

```
    } else if (cmd == "DEL") {
        // handle delete
    } else if (cmd == "QUERY") {
        // handle query
    }
}

void solve() {
    for (int i = 0; i < m; i++) {
        string cmd;
        cin >> cmd;
        handleCommand(cmd);
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    readInput();
    solve();
    return 0;
}
```

### 7.9.3 为什么要拆函数

- 读入、处理、输出分开，查错更容易。
- 每个命令一个函数，避免主函数过长。
- 特殊规则可以单独补，不容易破坏已有逻辑。
- 样例错时能定位是哪条规则错。

## 7.10 长题面拆解表

读 C 题时，在草稿纸上画这个表：

| 项目 | 要记录什么 | 示例 |
| --- | --- | --- |
| 对象 | 题目中被操作的东西 | 用户、文件、任务、格子、节点。 |
| 状态 | 对象身上的变量 | 位置、分数、权限、颜色、是否激活。 |
| 命令 | 输入中每种操作 | add、delete、query、move。 |
| 规则 | 每条命令如何改变状态 | 如果存在则更新，否则新增。 |
| 优先级 | 相同条件时谁先 | 时间小优先、编号小优先。 |
| 边界 | 特殊或非法情况 | 找不到、重复、越界、空集合。 |
| 输出 | 什么时候输出什么 | 每次 query 输出，最后统一输出。 |

## 7.11 命令解析

### 7.11.1 固定格式命令

如果每条命令格式固定：

```
string op;
cin >> op;
if (op == "add") {
    int id, x;
    cin >> id >> x;
} else if (op == "query") {
    int id;
    cin >> id;
}
```

### 7.11.2 整行命令

如果命令中可能有空格，先读整行。

```
int q;
cin >> q;
cin.ignore();

while (q--) {
    string line;
    getline(cin, line);
    stringstream ss(line);
    string op;
    ss >> op;

    if (op == "add") {
        string name;
        int x;
        ss >> name >> x;
    }
}
```

### 7.11.3 手写 split

按指定字符分割路径、日期、配置项：

```
vector<string> split(const string& s, char sep) {
    vector<string> res;
    string cur;
    for (char c : s) {
        if (c == sep) {
            res.push_back(cur);
            cur.clear();
```

```
        } else {
            cur.push_back(c);
        }
    }
    res.push_back(cur);
    return res;
}
```

### 7.11.4 命令分发模板

命令很多时，不要把所有代码写在一个很长的 `if` 里。可以先分发到函数。

```
void handleAdd() {
    int id, x;
    cin >> id >> x;
    // add logic
}

void handleDelete() {
    int id;
    cin >> id;
    // delete logic
}

void handleQuery() {
    int id;
    cin >> id;
    // query logic
}

void handleCommand() {
    string op;
    cin >> op;
    if (op == "add") handleAdd();
    else if (op == "delete") handleDelete();
    else if (op == "query") handleQuery();
}
```

### 7.11.5 命令解析易错点

- 命令参数个数是否固定。
- 参数中是否可能包含空格。
- 命令大小写是否敏感。
- 非法命令是否会出现。
- 每条命令是否都可能输出。

- 查询不存在对象时输出什么。

## 7.12 状态设计

### 7.12.1 用结构体保存对象

```
struct User {
    int id;
    string name;
    int score;
    bool active;
};

map<int, User> users;
```

### 7.12.2 状态编号

状态不多时可以用整数常量：

```
const int WAITING = 0;
const int RUNNING = 1;
const int FINISHED = 2;
const int FAILED = 3;
```

也可以用 enum：

```
enum Status {
    WAITING,
    RUNNING,
    FINISHED,
    FAILED
};
```

### 7.12.3 状态转移表

复杂状态题建议先画表：

| 当前状态 | 操作 | 新状态 | 输出/副作用 |
|---|---|---|---|
| WAITING | start | RUNNING | 记录开始时间 |
| RUNNING | finish | FINISHED | 计算得分 |
| RUNNING | fail | FAILED | 输出错误信息 |

## 7.13 状态机模拟

### 7.13.1 识别信号

题面出现：

- 一个对象有多个状态。
- 不同状态下，同一个命令效果不同。
- 状态按规则切换。
- 非法状态转换需要忽略或报错。

### 7.13.2 状态机模板

```
const int WAITING = 0;
const int RUNNING = 1;
const int FINISHED = 2;

struct Job {
    int id;
    int status;
    int score;
};

void start(Job& job) {
    if (job.status != WAITING) {
        return;
    }
    job.status = RUNNING;
}

void finish(Job& job) {
    if (job.status != RUNNING) {
        return;
    }
    job.status = FINISHED;
    job.score += 10;
}
```

### 7.13.3 状态机检查

- 初始状态是什么。
- 每个操作允许在哪些状态下发生。
- 不允许的操作是忽略、报错，还是改变成特殊状态。
- 终止状态是否还能被修改。
- 状态改变时是否还要更新其他字段。

## 7.14 事件模拟

### 7.14.1 识别信号

题面出现：

- 按时间顺序发生事件。
- 事件有开始、结束、到达、离开。
- 同一时间有多个事件。
- 需要维护当前正在发生的对象集合。

### 7.14.2 事件结构体

```
struct Event {
    int time;
    int type;
    int id;
};

vector<Event> events;
```

### 7.14.3 事件排序

```
sort(events.begin(), events.end(), [](const Event& a, const Event& b) {
    if (a.time != b.time) return a.time < b.time;
    return a.type < b.type; // depends on statement
});
```

### 7.14.4 事件处理模板

```
for (const Event& e : events) {
    if (e.type == 0) {
        // start event
    } else if (e.type == 1) {
        // end event
    } else {
        // query event
    }
}
```

### 7.14.5 同一时间事件顺序

这是事件模拟最容易错的地方。一定要从题面确定：

- 同一时间，结束事件先处理还是开始事件先处理。

- 查询是在事件前还是事件后。
- 如果没有说明，要从样例推断，但不要过度猜。
- 排序比较器中必须体现这个顺序。

## 7.15 优先级模拟

### 7.15.1 识别信号

题面出现：

- 每次选择优先级最高的任务。
- 优先级相同按时间、编号、字典序。
- 动态加入任务。
- 当前最大、当前最小。

### 7.15.2 priority_queue 模板

```
struct Task {
    int priority;
    int time;
    int id;
};

struct Cmp {
    bool operator()(const Task& a, const Task& b) const {
        if (a.priority != b.priority) return a.priority < b.priority;
        if (a.time != b.time) return a.time > b.time;
        return a.id > b.id;
    }
};

priority_queue<Task, vector<Task>, Cmp> pq;
```

这个比较器表示：

- 优先级大的先出。
- 优先级相同，时间小的先出。
- 时间相同，编号小的先出。

**priority_queue 的比较器和 sort 相反，容易写错。**不确定时，用几个任务手动推一下 `top()` 是谁。

## 7.16 复杂排序和排名

### 7.16.1 识别信号

题面出现：

- 多个排序关键字。
- 排名、积分榜、成绩表。
- 相同分数按编号、时间、字典序排序。
- 并列排名或去重排名。

### 7.16.2 复杂排序模板

```
struct Player {
    int id;
    int score;
    int penalty;
    string name;
};

sort(a.begin(), a.end(), [](const Player& x, const Player& y) {
    if (x.score != y.score) return x.score > y.score;
    if (x.penalty != y.penalty) return x.penalty < y.penalty;
    if (x.name != y.name) return x.name < y.name;
    return x.id < y.id;
});
```

### 7.16.3 并列排名

```
int rank = 1;
for (int i = 0; i < (int)a.size(); i++) {
    if (i > 0 && a[i].score != a[i - 1].score) {
        rank = i + 1;
    }
    cout << a[i].id << ' ' << rank << '\n';
}
```

### 7.16.4 复杂排序检查

- 每个关键字是升序还是降序。
- 相同关键字时是否继续比较。
- 是否要求稳定排序。
- 字符串按字典序还是输入顺序。
- 比较器相等时不能返回 `true`。

## 7.17 表格和矩阵模拟

### 7.17.1 识别信号

题面出现：

- 表格、座位、棋盘、网格。
- 行列操作。
- 旋转、翻转、交换行列。
- 查询某个单元格。

### 7.17.2 矩阵读入和输出

```
int n, m;
cin >> n >> m;
vector<vector<int>> a(n, vector<int>(m));
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        cin >> a[i][j];
    }
}
```

### 7.17.3 交换两行

```
int r1, r2;
cin >> r1 >> r2;
--r1; --r2;
swap(a[r1], a[r2]);
```

### 7.17.4 交换两列

```
int c1, c2;
cin >> c1 >> c2;
--c1; --c2;
for (int i = 0; i < n; i++) {
    swap(a[i][c1], a[i][c2]);
}
```

### 7.17.5 顺时针旋转矩阵

```
vector<vector<int>> rotateClockwise(const vector<vector<int>>& a) {
    int n = (int)a.size();
    int m = (int)a[0].size();
    vector<vector<int>> b(m, vector<int>(n));
    for (int i = 0; i < n; i++) {
```

```
        for (int j = 0; j < m; j++) {
            b[j][n - 1 - i] = a[i][j];
        }
    }
    return b;
}
```

### 7.17.6 表格模拟易错点

- 题目行列通常从 1 开始，代码常从 0 开始。
- 旋转后行数和列数会交换。
- 如果操作很多，真实移动整个矩阵可能超时，要考虑只维护行列映射。
- 输出每行空格格式。

## 7.18 目录树和路径模拟

### 7.18.1 识别信号

题面出现：

- 文件、目录、路径。
- /a/b/c、相对路径、绝对路径。
- 创建、删除、查询文件。
- 父目录、子目录。

### 7.18.2 路径规范化

处理 . 和 ..：

```
vector<string> normalizePath(const string& path) {
    vector<string> st;
    vector<string> parts = split(path, '/');
    for (string part : parts) {
        if (part.empty() || part == ".") {
            continue;
        } else if (part == "..") {
            if (!st.empty()) st.pop_back();
        } else {
            st.push_back(part);
        }
    }
    return st;
}
```

### 7.18.3 路径转字符串

```cpp
string joinPath(const vector<string>& parts) {
    if (parts.empty()) return "/";
    string res;
    for (const string& p : parts) {
        res += "/";
        res += p;
    }
    return res;
}
```

### 7.18.4 目录树节点

```cpp
struct Node {
    string name;
    map<string, int> child;
    bool isFile = false;
};

vector<Node> tree;
```

新建节点：

```cpp
int newNode(const string& name, bool isFile) {
    Node node;
    node.name = name;
    node.isFile = isFile;
    tree.push_back(node);
    return (int)tree.size() - 1;
}
```

### 7.18.5 路径题易错点

- 根目录怎么表示。
- 连续斜杠是否可能出现。
- 路径末尾斜杠是否影响结果。
- 查询不存在路径时输出什么。
- 文件和目录是否允许同名。

## 7.19 URL 路由和参数解析

### 7.19.1 识别信号

题面出现：

- URL、路径、路由。
- `/user/123/profile`。
- 查询参数，例如 `?a=1&b=2`。
- 通配符、变量匹配。

### 7.19.2 拆分 URL

```cpp
struct Url {
    string path;
    map<string, string> query;
};

Url parseUrl(const string& s) {
    Url res;
    int qpos = (int)s.find('?');
    if (qpos == (int)string::npos) {
        res.path = s;
        return res;
    }

    res.path = s.substr(0, qpos);
    string qs = s.substr(qpos + 1);
    vector<string> pairs = split(qs, '&');
    for (string p : pairs) {
        int eq = (int)p.find('=');
        if (eq == (int)string::npos) {
            res.query[p] = "";
        } else {
            string key = p.substr(0, eq);
            string val = p.substr(eq + 1);
            res.query[key] = val;
        }
    }
    return res;
}
```

### 7.19.3 简单路由匹配

模式 `/user/<id>` 匹配 `/user/123`：

```cpp
bool matchRoute(const string& pattern, const string& path,
                map<string, string>& params) {
    vector<string> a = parsePath(pattern);
    vector<string> b = parsePath(path);
    if (a.size() != b.size()) return false;

```

```
    for (int i = 0; i < (int)a.size(); i++) {
        if (!a[i].empty() && a[i][0] == '<' && a[i].back() == '>') {
            string name = a[i].substr(1, (int)a[i].size() - 2);
            params[name] = b[i];
        } else if (a[i] != b[i]) {
            return false;
        }
    }
    return true;
}
```

### 7.19.4 URL 题易错点

- 有没有查询参数。
- 参数值是否可能为空。
- 路径末尾斜杠是否等价。
- 多个路由都匹配时，第一条生效还是更具体的生效。
- URL 是否需要解码，题目不要求就不要自己加。

## 7.20 配置和 JSON 类解析入口

### 7.20.1 识别信号

题面出现：

- 键值对。
- 花括号、方括号、冒号、逗号。
- 嵌套对象。
- 查询某个字段。

### 7.20.2 简单键值配置

每行形如 `key=value`：

```
int n;
cin >> n;
map<string, string> conf;
for (int i = 0; i < n; i++) {
    string line;
    cin >> line;
    int pos = (int)line.find('=');
    string key = line.substr(0, pos);
    string value = line.substr(pos + 1);
    conf[key] = value;
}
```

### 7.20.3 读取带空格的配置值

```
string line;
getline(cin, line);
int pos = (int)line.find('=');
string key = line.substr(0, pos);
string value = line.substr(pos + 1);
```

### 7.20.4 嵌套结构处理思路

新手处理 JSON 类题时，不要一开始写完整通用解析器。先看题目限制：

- 是否只有字符串和对象，没有数组。
- 是否保证格式合法。
- 查询是否只按点号路径。
- 是否只需要保存完整路径到值的映射。

如果只需要查询路径，可以把嵌套字段拍平成 key：

```
// Example flattened keys:
// user.name -> Alice
// user.age -> 18
map<string, string> value;
```

**配置/JSON 题先利用限制。**CSP 题通常不会要求你写工业级 JSON 解析器，重点是按题目给定范围实现。

## 7.21 字符串解析

### 7.21.1 识别信号

C 题中的字符串通常比 A/B 题更复杂。题面出现下面内容时，要进入字符串解析思路：

- 路径，例如 `/a/b/c`。
- 配置项，例如 `key=value`。
- 表达式，例如括号、加减乘除。
- 带引号的字符串。
- 嵌套结构。
- 命令参数中可能包含空格。

### 7.21.2 逐字符扫描模板

```
string s;
cin >> s;

```

```
for (int i = 0; i < (int)s.size(); i++) {
    char c = s[i];
    if (isdigit(c)) {
        int j = i;
        while (j < (int)s.size() && isdigit(s[j])) j++;
        string number = s.substr(i, j - i);
        i = j - 1;
    } else if (isalpha(c)) {
        int j = i;
        while (j < (int)s.size() && isalpha(s[j])) j++;
        string word = s.substr(i, j - i);
        i = j - 1;
    } else {
        // symbol
    }
}
```

### 7.21.3 解析 key=value

```
string s;
cin >> s;
int pos = (int)s.find('=');
string key = s.substr(0, pos);
string value = s.substr(pos + 1);
```

### 7.21.4 路径解析

```
vector<string> parsePath(const string& path) {
    vector<string> parts;
    string cur;
    for (char c : path) {
        if (c == '/') {
            if (!cur.empty()) {
                parts.push_back(cur);
                cur.clear();
            }
        } else {
            cur.push_back(c);
        }
    }
    if (!cur.empty()) parts.push_back(cur);
    return parts;
}
```

### 7.21.5 括号匹配

```cpp
bool checkBrackets(const string& s) {
    stack<char> st;
    for (char c : s) {
        if (c == '(' || c == '[' || c == '{') {
            st.push(c);
        } else if (c == ')' || c == ']' || c == '}') {
            if (st.empty()) return false;
            char t = st.top();
            st.pop();
            if (c == ')' && t != '(') return false;
            if (c == ']' && t != '[') return false;
            if (c == '}' && t != '{') return false;
        }
    }
    return st.empty();
}
```

### 7.21.6 字符串解析易错点

- `cin >> s` 不能读取空格。
- `getline` 前可能需要 `cin.ignore()`。
- `find` 找不到时返回 `string::npos`。
- `substr(pos, len)` 的第二个参数是长度，不是结束位置。
- 连续分隔符和开头/结尾分隔符要单独考虑。
- 字符串数字可能超过 `long long`。

## 7.22 表达式解析入口

### 7.22.1 识别信号

题面出现：

- 表达式、括号、运算符。
- 加减乘除优先级。
- 逻辑表达式。
- 嵌套结构。

### 7.22.2 只含加减的表达式

```cpp
long long evalPlusMinus(const string& s) {
    long long ans = 0;
    long long num = 0;
```

```
    int sign = 1;

    for (int i = 0; i <= (int)s.size(); i++) {
        if (i < (int)s.size() && isdigit(s[i])) {
            num = num * 10 + (s[i] - '0');
        } else {
            ans += sign * num;
            num = 0;
            if (i < (int)s.size()) {
                if (s[i] == '+') sign = 1;
                else if (s[i] == '-') sign = -1;
            }
        }
    }
    return ans;
}
```

### 7.22.3 递归解析思想

复杂表达式通常需要递归下降。新手先掌握思想：

```
string s;
int pos = 0;

long long parseNumber() {
    long long x = 0;
    while (pos < (int)s.size() && isdigit(s[pos])) {
        x = x * 10 + (s[pos] - '0');
        pos++;
    }
    return x;
}
```

**表达式题如果没练过，不要强行写复杂递归下降。**先拿没有括号、没有优先级的部分分。

## 7.23 缓存和队列模拟

### 7.23.1 识别信号

题面出现：

- 缓存、页面置换、最近使用。
- 队列、排队、服务窗口。
- 先进先出、最近最少使用。
- 容量限制，满了要删除一个。

### 7.23.2 FIFO 缓存

```
int cap;
cin >> cap;
queue<int> q;
set<int> inCache;

for (int x : requests) {
    if (inCache.count(x)) {
        continue;
    }
    if ((int)q.size() == cap) {
        int old = q.front();
        q.pop();
        inCache.erase(old);
    }
    q.push(x);
    inCache.insert(x);
}
```

### 7.23.3 LRU 缓存入口

最近最少使用可以用 `list + unordered_map`，但新手要谨慎。

```
int cap;
list<int> order; // front is most recent
unordered_map<int, list<int>::iterator> pos;

void access(int x) {
    if (pos.count(x)) {
        order.erase(pos[x]);
    } else if ((int)order.size() == cap) {
        int old = order.back();
        order.pop_back();
        pos.erase(old);
    }
    order.push_front(x);
    pos[x] = order.begin();
}
```

### 7.23.4 队列模拟易错点

- 队列为空时不能取 `front`。
- 缓存容量可能为 0，要看题目是否允许。
- 命中缓存时是否需要更新时间。
- 删除元素时，辅助集合或 map 也要同步删除。

## 7.24 懒删除优先队列

### 7.24.1 识别信号

题面出现：

- 需要动态删除任务。
- 还要每次取最大/最小。
- `priority_queue` 中间删除不方便。

### 7.24.2 懒删除思想

不直接从堆里删除旧元素，而是在取堆顶时检查它是否还有效。

```
struct Task {
    int priority;
    int id;
};

struct Cmp {
    bool operator()(const Task& a, const Task& b) const {
        if (a.priority != b.priority) return a.priority < b.priority;
        return a.id > b.id;
    }
};

priority_queue<Task, vector<Task>, Cmp> pq;
map<int, int> currentPriority;

void addOrUpdate(int id, int priority) {
    currentPriority[id] = priority;
    pq.push({priority, id});
}

void removeTask(int id) {
    currentPriority.erase(id);
}

Task getTop() {
    while (!pq.empty()) {
        Task t = pq.top();
        if (currentPriority.count(t.id) &&
            currentPriority[t.id] == t.priority) {
            return t;
        }
        pq.pop();
    }
    return {-1, -1};
```

```
}
```

**懒删除会让堆里有废数据。**每次取 top 前必须清理过期元素。

## 7.25 时间和日程模拟

### 7.25.1 识别信号

题面出现：

- 时间段、会议、日程。
- 判断冲突。
- 开始时间、结束时间。
- 按时间排序处理。

### 7.25.2 时间转分钟

```
int toMinute(const string& s) {
    int h = stoi(s.substr(0, 2));
    int m = stoi(s.substr(3, 2));
    return h * 60 + m;
}
```

### 7.25.3 判断区间冲突

两个半开区间 $[l1, r1)$ 和 $[l2, r2)$ 冲突：

```
bool overlap(int l1, int r1, int l2, int r2) {
    return max(l1, l2) < min(r1, r2);
}
```

如果题目是闭区间，条件要改。

### 7.25.4 扫描事件统计同时进行数量

```
vector<pair<int, int>> events;
for (auto [l, r] : intervals) {
    events.push_back({l, 1});
    events.push_back({r, -1});
}

sort(events.begin(), events.end(), [](auto a, auto b) {
    if (a.first != b.first) return a.first < b.first;
    return a.second < b.second; // end before start for [l,r)
});

int cur = 0, best = 0;
```

```
for (auto [time, delta] : events) {
    cur += delta;
    best = max(best, cur);
}
```

### 7.25.5 时间题易错点

- 时间段是闭区间还是半开区间。
- 结束时间和开始时间相同是否冲突。
- 跨天如何处理。
- 同一时间开始和结束谁先处理。

## 7.26 树形结构模拟

### 7.26.1 识别信号

题面出现：

- 层级、父子、目录、组织结构。
- 每个节点有父节点。
- 查询祖先、子树、路径。

### 7.26.2 父子关系存储

```
int n;
cin >> n;
vector<vector<int>> children(n + 1);
vector<int> parent(n + 1, 0);

for (int i = 2; i <= n; i++) {
    int p;
    cin >> p;
    parent[i] = p;
    children[p].push_back(i);
}
```

### 7.26.3 DFS 遍历树

```
vector<int> depth(n + 1, 0);

void dfsTree(int u) {
    for (int v : children[u]) {
        depth[v] = depth[u] + 1;
        dfsTree(v);
```

```
    }
}
```

### 7.26.4 子树大小

```
vector<int> subtree(n + 1, 1);

void calcSubtree(int u) {
    subtree[u] = 1;
    for (int v : children[u]) {
        calcSubtree(v);
        subtree[u] += subtree[v];
    }
}
```

### 7.26.5 树题易错点

- 根节点是谁。
- 输入给的是父节点，还是边。
- 节点编号从 0 还是 1 开始。
- 树很深时递归可能爆栈。
- 如果需要频繁祖先查询，简单爬父亲可能超时。

## 7.27 权限和规则匹配

### 7.27.1 识别信号

题面出现：

- 用户、角色、权限。
- 规则匹配。
- 黑名单、白名单。
- 多条规则按顺序匹配，第一条生效。
- 默认允许或默认拒绝。

### 7.27.2 规则结构体

```
struct Rule {
    string user;
    string action;
    string resource;
    bool allow;
};
```

```
7
8 vector<Rule> rules;
```

### 7.27.3 规则匹配函数

```
1 bool match(const Rule& r, const string& user,
2          const string& action, const string& resource) {
3     if (r.user != "*" && r.user != user) return false;
4     if (r.action != "*" && r.action != action) return false;
5     if (r.resource != "*" && r.resource != resource) return false;
6     return true;
7 }
```

### 7.27.4 按顺序匹配

```
1 bool checkPermission(const string& user,
2                   const string& action,
3                   const string& resource) {
4     for (const Rule& r : rules) {
5         if (match(r, user, action, resource)) {
6             return r.allow;
7         }
8     }
9     return false; // default deny
10 }
```

### 7.27.5 规则题易错点

- 多条规则是第一条生效，还是最后一条生效。
- 默认值是允许还是拒绝。
- 通配符如何匹配。
- 大小写是否敏感。
- 资源路径是否需要前缀匹配。

## 7.28 网格和图混合题

### 7.28.1 识别信号

题面出现：

- 地图、障碍、传送门、钥匙、门。
- 不只是简单最短路，还有状态。
- 每个格子可能有属性。

- 移动过程中会改变状态。

### 7.28.2 普通网格 BFS

如果只有障碍和空地，用 B 题 BFS 模板即可。

### 7.28.3 带状态 BFS 入口

如果移动时还带着状态，例如是否拿到钥匙，可以把状态加入距离数组。

```
struct Node {
    int x;
    int y;
    int state;
};

vector<vector<vector<int>>> dist(n,
    vector<vector<int>>(m, vector<int>(S, -1)));

queue<Node> q;
dist[sx][sy][0] = 0;
q.push({sx, sy, 0});

while (!q.empty()) {
    Node cur = q.front();
    q.pop();

    for (int d = 0; d < 4; d++) {
        int nx = cur.x + dx[d];
        int ny = cur.y + dy[d];
        int ns = cur.state;
        // update ns according to grid[nx][ny]

        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (dist[nx][ny][ns] != -1) continue;
        dist[nx][ny][ns] = dist[cur.x][cur.y][cur.state] + 1;
        q.push({nx, ny, ns});
    }
}
```

**状态数不能太大。**如果状态有 $2^k$，先看 $k$ 是否很小。$k \le 10$ 常见可做，$k = 30$ 通常不行。

### 7.28.4 图模拟策略

如果题目是图但规则很多：

- 先把图建对：有向/无向、权值、编号。
- 再处理规则：哪些边能走，什么时候能走。
- 如果边权全为 1，用 BFS。

- 如果边权为正，用 Dijkstra。
- 如果只是连通关系，用并查集或 DFS。

## 7.29 大模拟易错点

- 同一时间多个事件的顺序。
- 相同优先级的并列规则。
- 删除元素后迭代器失效。
- 查询不存在对象。
- 重复添加同一对象。
- 空集合时取最大、最小。
- 最后一轮事件是否处理。
- 题目中的编号是 0 开始还是 1 开始。
- 输出顺序是否要求按编号、时间或字典序。

## 7.30 C 题分阶段实现

### 7.30.1 不要一次写完

C 题最怕写 150 行后才第一次运行。建议按阶段实现：

1. 只写读入，打印读到的数据检查。
2. 写数据结构，不写复杂规则。
3. 实现第一种命令或最基础操作。
4. 跑样例，看是否能得到部分中间结果。
5. 再补第二种命令。
6. 最后处理特殊规则、并列规则、异常情况。

### 7.30.2 命令题实现顺序

| 阶段 | 做什么 | 目标 |
| --- | --- | --- |
| 1 | 读入所有命令，只识别命令名 | 确认输入没错。 |
| 2 | 实现新增/初始化类命令 | 建立基本状态。 |
| 3 | 实现查询类命令 | 能输出最简单答案。 |
| 4 | 实现修改/删除类命令 | 状态开始完整。 |
| 5 | 实现特殊规则 | 冲满分。 |

### 7.30.3 模拟题实现顺序

1. 先实现没有特殊情况的主流程。
2. 再实现边界检查。
3. 再实现冲突和优先级。
4. 最后实现异常输入或题目特别说明的细节。

## 7.31 函数化检查清单

C 题代码一长，就必须主动控制结构。

### 7.31.1 建议函数类型

| 函数类型 | 示例 | 作用 |
| --- | --- | --- |
| 读入函数 | `readInput()` | 避免读入和处理混在一起。 |
| 解析函数 | `parsePath(s)` | 字符串处理单独测试。 |
| 查询函数 | `query(id)` | 查询逻辑独立。 |
| 修改函数 | `update(id,x)` | 状态改变集中处理。 |
| 检查函数 | `canMove(x,y)` | 规则判断集中处理。 |
| 输出函数 | `printAnswer()` | 控制格式。 |

### 7.31.2 函数设计原则

- 一个函数只做一类事情。
- 函数名直接反映题目操作。
- 不要让一个函数超过 50 行，除非只是模板。
- 修改状态的函数要特别小心副作用。
- 查询函数尽量不要修改状态，除非题目要求。

### 7.31.3 推荐代码顺序

```
// constants and types
struct ...

// global variables
int n, m;
vector<...> ...

// helper functions
bool inside(...);
vector<string> split(...);

```

```
// operation functions
void handleAdd(...);
void handleQuery(...);

// main solve
void solve() { ... }
```

## 7.32 分层测试方法

### 7.32.1 第一层：解析测试

只测试输入解析，不测算法。

- 路径是否被正确分割。
- 命令参数是否读对。
- 数字字符串是否转换正确。
- 空格和换行是否处理正确。

### 7.32.2 第二层：单操作测试

每种操作单独测：

- 只 add 一次。
- 只 query 一次。
- add 后 query。
- delete 后 query。

### 7.32.3 第三层：组合测试

测试多操作交替：

- add、delete、add 同一个对象。
- 多个对象按优先级排序。
- 同一时间多个事件。
- 状态反复切换。

### 7.32.4 第四层：边界测试

测试异常和边界：

- 空集合查询。
- 不存在对象。
- 重复对象。

- 最大数据附近。
- 最后一条命令。

## 7.33 C 题常见失分原因

| 失分原因 | 表现 | 预防 |
|---|---|---|
| 读题漏规则 | 样例某一步对不上 | 三遍读题，列规则表。 |
| 状态没初始化 | 第一次查询就错 | 结构体构造或统一 init。 |
| 命令读错 | 后续全错 | 先调试输出 op 和参数。 |
| 比较器错 | 排名/优先级错 | 手写 3 个对象测试 top/order。 |
| 删除不同步 | 查到已删除对象 | 容器、set、map 同步维护。 |
| 最后事件漏处理 | 前面都对，最后错 | 检查循环结束后的收尾。 |
| 输出顺序错 | 数值对但顺序错 | 看题目要求按什么顺序。 |
| 复杂度过高 | 小样例过，大数据超时 | 估算每条命令复杂度。 |

## 7.34 C 题本地调试

### 7.34.1 调试输出建议

调试时输出状态：

```
void debugTask(const Task& t) {
    cerr << "id=" << t.id
         << " status=" << t.status
         << " priority=" << t.priority << '\n';
}
```

处理命令时：

```
cerr << "op=" << op << '\n';
```

**提交前删除所有调试输出。**C 题代码长，尤其容易漏掉 `cerr` 或临时 `cout`。

### 7.34.2 缩小样例

如果官方样例很长，自己造小样例：

- 1 个对象，1 条命令。
- 1 个对象，2 条命令。
- 2 个对象，测试排序或优先级。
- 不存在对象的查询。
- 重复添加和删除。

### 7.34.3 定位 bug 的方法

1. 先确认读入正确。
2. 再确认每条命令被识别正确。
3. 再确认命令前状态正确。
4. 再确认命令后状态正确。
5. 最后确认输出格式。

## 7.35 C 题常见边界库

| 题型 | 边界 | 检查目标 |
| --- | --- | --- |
| 命令模拟 | 只有 1 条命令、没有查询、连续查询 | 初始化和输出。 |
| 新增删除 | 重复新增、删除不存在、删除后查询 | 对象生命周期。 |
| 状态机 | 非法状态转换、终止状态继续操作 | 状态规则。 |
| 事件模拟 | 同一时间多个事件、开始和结束同一时刻 | 事件排序。 |
| 优先级 | 优先级相同、时间相同、编号相同附近 | 比较器。 |
| 字符串解析 | 空字段、连续分隔符、路径以分隔符结尾 | split 和 substr。 |
| 网格 | 起点等于终点、无路、只有一个格子 | BFS 初始化。 |
| 带状态 BFS | 拿不到状态、重复状态、状态数最大 | dist 维度。 |
| 图 | 自环、重边、不连通、有向/无向 | 建图。 |

## 7.36 C 题部分分策略

1. 先实现最基础命令。
2. 暂时忽略最复杂的特殊规则，先让程序能跑。
3. 如果有查询，先保证简单查询正确。
4. 如果有多种对象，先支持一种对象。
5. 如果有多档数据，先满足小数据。
6. 留出最后 10 分钟清理输出和调试信息。

## 7.37 C 题分层拿分路线

### 7.37.1 30 分路线

目标是程序能跑，能处理最简单情况。

- 正确读入。
- 正确识别命令。
- 实现最基本的新增、查询或主流程。
- 不处理复杂并列规则。
- 不处理极端异常情况。

### 7.37.2 60 分路线

目标是覆盖大部分普通规则。

- 所有命令都有实现。
- 常见异常处理正确，例如不存在、重复、越界。
- 基本排序或优先级正确。
- 简单边界样例正确。
- 复杂度对中等数据可接受。

### 7.37.3 80 到 100 分路线

目标是处理所有细节。

- 所有并列规则正确。
- 同一时间事件顺序正确。
- 所有边界和异常符合题意。
- 大数据复杂度可过。
- 输出顺序和格式完全正确。

## 7.38 失败降级策略

### 7.38.1 写不完大模拟

- 保留已完成命令，不要为了补复杂规则改崩基础逻辑。
- 查询类命令优先，因为它们直接产生输出。
- 特殊规则用清晰的 `if` 补，不要重构大段代码。

### 7.38.2 字符串解析写不出来

- 先处理不含空格、不含嵌套的情况。
- 先用 `split` 处理简单分隔符。
- 括号和嵌套拿不到满分时，至少保证普通 token 正确。

### 7.38.3 状态 BFS 写不出来

- 先写普通 BFS。
- 如果状态只有 2 种，尝试 `dist[x][y][2]`。
- 如果状态很多，先拿没有特殊状态的数据。

### 7.38.4 复杂度可能超时

- 先估每条命令复杂度。
- 能用 map 查找就不要全表扫描。
- 能预处理就不要每次重复计算。
- 时间不够时，保留暴力拿小数据分。

## 7.39 C 题临场拿分路线

### 7.39.1 如果是大模拟

1. 先读懂输入输出。
2. 建结构体保存对象。
3. 实现最简单命令。
4. 实现查询输出。
5. 再补删除、修改、异常情况。
6. 最后补优先级、并列规则。

### 7.39.2 如果是字符串解析

1. 先确认用 `cin` 还是 `getline`。
2. 先把字符串拆成 token。
3. 能用 `split` 就不要写复杂解析。
4. 如果有括号，先写括号匹配。
5. 如果有嵌套，先拿无嵌套部分分。

### 7.39.3 如果是图或网格

1. 先把图或网格读入正确。
2. 判断是最短路、连通块还是模拟移动。
3. 无权最短路用 BFS。
4. 带状态时先估算状态数量。
5. 不会状态优化时，先写普通 BFS 拿部分分。

### 7.39.4 如果是规则匹配

1. 把每条规则保存为结构体。
2. 单独写 `match` 函数。
3. 确认第一条生效还是最后一条生效。
4. 确认默认值。
5. 用 2 到 3 条规则手测。

## 7.40 C 题模板索引

| 题面关键词 | 使用模板 | 本章位置 |
| --- | --- | --- |
| 多命令 | 命令分发 | 命令解析 |
| 对象状态 | struct + 状态机 | 状态设计、状态机模拟 |
| 按时间发生 | Event + sort | 事件模拟 |
| 优先级 | priority_queue | 优先级模拟 |
| 排名 | 多关键字 sort | 复杂排序和排名 |
| 表格、矩阵 | vector<vector<int» | 表格和矩阵模拟 |
| 路径、目录 | split + normalizePath | 目录树和路径模拟 |
| URL、参数 | parseUrl | URL 路由和参数解析 |
| 配置、键值 | key=value | 配置和 JSON 类解析 |
| 括号、表达式 | stack / 递归解析 | 字符串解析、表达式解析 |
| 缓存 | queue / list + map | 缓存和队列模拟 |
| 动态删除最大值 | 懒删除堆 | 懒删除优先队列 |
| 时间段冲突 | overlap / events | 时间和日程模拟 |
| 父子层级 | children + DFS | 树形结构模拟 |
| 地图状态 | state BFS | 网格和图混合题 |

## 7.41 C 题考场速查页

### 7.41.1 读题 5 分钟内要确定

- 是大模拟、字符串解析、图网格、数据结构，还是 DP。

- 有哪些对象。
- 每个对象有哪些状态。
- 有哪些命令或事件。
- 输出发生在每次查询还是最后。
- 数据范围是否允许全表扫描。

### 7.41.2 写代码前必须设计

- struct 里有哪些字段。
- 用 vector、map、set 还是堆。
- 每个命令对应哪个函数。
- 比较器的排序规则。
- 不存在、重复、越界怎么处理。

### 7.41.3 最常用代码形状

```cpp
struct Obj {
    int id;
    int status;
};

map<int, Obj> obj;

void handleAdd() {}
void handleDelete() {}
void handleQuery() {}

void solve() {
    int q;
    cin >> q;
    while (q--) {
        string op;
        cin >> op;
        if (op == "add") handleAdd();
        else if (op == "delete") handleDelete();
        else if (op == "query") handleQuery();
    }
}
```

### 7.41.4 最后 10 分钟检查

- 删除调试输出。
- 检查所有命令是否都处理。

- 检查输出大小写、空格、换行。
- 检查比较器并列规则。
- 检查空集合、找不到对象。
- 检查最后一个事件。
- 提交一个当前最稳版本。

## 7.42 C 题提交前检查

- 是否删除所有调试输出。
- 每种命令是否都处理了。
- 每种状态是否都有初始值。
- 不存在对象时是否按题意处理。
- 重复对象是否按题意处理。
- 排序和优先级比较器是否正确。
- 字符串读取是否会漏掉空格。
- 多组数据是否清空所有容器。
- 最后一个事件或最后一段数据是否处理。
- 输出顺序是否符合题目。

## 7.43 C 题手写模板清单

| 模板 | 熟练要求 | 用途 |
| --- | --- | --- |
| 结构体 + map 保存对象 | 必须会写 | 对象状态模拟。 |
| 命令分发 | 必须会写 | 多命令题。 |
| getline + stringstream | 必须会写 | 整行命令。 |
| split | 照着能写 | 路径、配置解析。 |
| 状态机 | 照着能写 | 多状态对象。 |
| 事件排序 | 照着能写 | 时间顺序模拟。 |
| priority_queue 自定义比较 | 练过再用 | 优先级模拟。 |
| 括号匹配 | 照着能写 | 表达式/嵌套结构。 |
| 带状态 BFS | 理解后再用 | 网格状态题。 |

# 第八章　D/E 题部分分策略与高级算法入口

## 8.1　D/E 题定位

D/E 题通常是高分段的分水岭。对新手来说，目标不是每题都满分，而是避免空题，尽量拿到小数据、特殊性质和熟悉模板的分数。

- D 题目标：优先找 20–60 分的可写做法，熟悉算法时再冲满分。
- E 题目标：不空题，找特殊性质、小数据、暴力分。
- 核心原则：先写能得分的版本，再考虑优化。
- 最大风险：想满分算法想 60 分钟，最后一行代码都没有。

**D/E 题第一原则：**如果 20 分钟内没有形成清晰满分方案，立刻转为部分分路线。

## 8.2　D/E 题读题流程

1. 先看有没有子任务或部分分说明。
2. 抄下所有数据范围，尤其是小数据档。
3. 找特殊性质：链、树、DAG、无修改、无查询、边权相同、参数很小。
4. 写出最暴力做法。
5. 估算暴力能拿哪些数据点。
6. 如果会优化，再逐步优化。

## 8.3　部分分决策树

| 题面或数据特征 | 可写做法 | 目标 |
|---|---|---|
| $n \le 20$ | DFS、枚举子集、状态压缩 | 小数据分 |
| $n \le 100$ | $O(n^3)$、Floyd、区间 DP | 小中数据 |
| $n \le 1000$ | $O(n^2)$ | 中小数据 |
| 图是树 | DFS、树形 DP、LCA 入门 | 特殊性质分 |
| 图是链 | 转成数组问题 | 特殊性质分 |
| 边权全为 1 | BFS | 替代 Dijkstra |
| 没有修改 | 前缀和、静态预处理 | 静态数据分 |

| | | |
|---|---|---|
| 查询很少 | 每次暴力 | 小 $q$ 分 |
| 修改很少 | 修改后重算 | 小修改分 |
| 参数 $k$ 很小 | 状压、枚举 $k$ | 参数小分 |
| 只要求判断可行 | 二分答案 / 贪心 check | 冲高分 |

## 8.4　暴力模板

### 8.4.1　枚举所有点对

```cpp
long long ans = 0;
for (int i = 0; i < n; i++) {
    for (int j = i + 1; j < n; j++) {
        // check pair (i, j)
    }
}
```

适用：$n \le 5000$ 需谨慎，$n \le 1000$ 通常可试。

### 8.4.2　枚举所有区间

```cpp
for (int l = 0; l < n; l++) {
    long long sum = 0;
    for (int r = l; r < n; r++) {
        sum += a[r];
        // process interval [l, r]
    }
}
```

适用：$n \le 5000$ 以内看时限；如果 $n = 10^5$，只能拿小数据分。

### 8.4.3　DFS 枚举选择

```cpp
int n;
vector<int> choose;

void dfs(int idx) {
    if (idx == n) {
        // evaluate choose
        return;
    }

    // not choose idx
    dfs(idx + 1);

    // choose idx
    choose.push_back(idx);
    dfs(idx + 1);
```

```
    choose.pop_back();
}
```

适用：$n \le 20$ 左右。

## 8.5 小数据常用算法

### 8.5.1 Floyd

点数小且问任意两点最短路：

```
for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

### 8.5.2 全排列

```
vector<int> p(n);
iota(p.begin(), p.end(), 0);
do {
    // evaluate permutation
} while (next_permutation(p.begin(), p.end()));
```

适用：$n \le 9$ 或 10 左右。

### 8.5.3 状压枚举子集

```
for (int mask = 0; mask < (1 << n); mask++) {
    for (int i = 0; i < n; i++) {
        if (mask & (1 << i)) {
            // i is selected
        }
    }
}
```

适用：$n \le 20$ 左右。若 $n \ge 31$，注意 `1LL << n` 和复杂度。

## 8.6 特殊性质策略

### 8.6.1 图是链

如果图是一条链，很多图问题可以转成数组问题：

- 距离变成下标差或前缀和。

- 子树变成连续区间。
- 路径查询变成区间查询。

### 8.6.2 图是树

树比一般图简单：

- 没有环。
- 两点之间路径唯一。
- 可用 DFS 统计子树。
- 可用树形 DP。

### 8.6.3 边权相同

如果所有边权都是 1：

- 最短路用 BFS。
- 不需要 Dijkstra。

### 8.6.4 无修改

如果没有修改，只有查询：

- 优先预处理。
- 区间问题考虑前缀和。
- 图问题考虑一次 BFS/Dijkstra 后回答多次查询。

## 8.7 按题型拿部分分

### 8.7.1 图论题

| 不会满分时 | 可写做法 | 可能覆盖 |
| --- | --- | --- |
| 不会复杂最短路 | 小图 Floyd | $n$ 小的数据 |
| 不会带权最短路 | 无权 BFS | 边权全 1 的数据 |
| 不会动态图 | 每次修改后重跑 BFS/DFS | 操作数小的数据 |
| 不会强连通 | 普通 DFS 连通性 | 无向图或特殊数据 |
| 不会树上高级算法 | 从每个查询端点暴力爬/DFS | $n, q$ 小的数据 |

### 8.7.2 区间数据结构题

| 不会满分时 | 可写做法 | 可能覆盖 |
| --- | --- | --- |
| 不会线段树 | 每次暴力扫描区间 | 小 $nq$ 数据 |

| | | |
|---|---|---|
| 只有查询无修改 | 前缀和 | 静态数据 |
| 只有区间修改最后输出 | 差分 | 离线数据 |
| 单点修改区间查询 | 树状数组入口 | 中等数据 |
| 区间修改单点查询 | 差分思想 / 树状数组 | 中等数据 |

### 8.7.3 DP 题

| 不会满分时 | 可写做法 | 可能覆盖 |
|---|---|---|
| 状态想不清 | DFS 暴力枚举 | $n$ 小的数据 |
| 背包优化不会 | 二维 DP | 小容量数据 |
| 状态压缩不会 | 回溯搜索 | $n$ 小的数据 |
| 转移太复杂 | 只处理部分状态或特殊参数 | 特殊性质数据 |

### 8.7.4 字符串题

| 不会满分时 | 可写做法 | 可能覆盖 |
|---|---|---|
| 不会 KMP | `find` 或暴力匹配 | 短字符串数据 |
| 不会 Trie | map 存完整字符串 | 字符串数量较少 |
| 不会复杂解析 | split 简单格式 | 无嵌套数据 |
| 不会哈希 | 直接比较字符串 | 小数据 |

## 8.8 树状数组入口

### 8.8.1 识别信号

- 单点修改，前缀和或区间和查询。
- 需要动态维护数组和。
- 数据范围 $n, q \le 10^5$。

### 8.8.2 树状数组模板

```
struct BIT {
    int n;
    vector<long long> tree;

    BIT(int n) : n(n), tree(n + 1, 0) {}

    void add(int idx, long long val) {
        for (; idx <= n; idx += idx & -idx) {
            tree[idx] += val;
        }
    }
```

```
12
13      long long sumPrefix(int idx) {
14          long long res = 0;
15          for (; idx > 0; idx -= idx & -idx) {
16              res += tree[idx];
17          }
18          return res;
19      }
20
21      long long rangeSum(int l, int r) {
22          return sumPrefix(r) - sumPrefix(l - 1);
23      }
24  };
```

**树状数组一般用 1 下标。**

## 8.9 线段树入口

### 8.9.1 识别信号

- 区间查询和区间修改交替出现。
- 需要维护区间和、区间最大值、区间最小值。
- $n, q \le 10^5$，暴力扫描会超时。

### 8.9.2 区间和线段树框架

```
1   struct SegTree {
2       int n;
3       vector<long long> tree, lazy;
4
5       SegTree(int n) : n(n), tree(4 * n + 4), lazy(4 * n + 4) {}
6
7       void apply(int node, int l, int r, long long val) {
8           tree[node] += val * (r - l + 1);
9           lazy[node] += val;
10      }
11
12      void push(int node, int l, int r) {
13          if (lazy[node] == 0 || l == r) return;
14          int mid = (l + r) / 2;
15          apply(node * 2, l, mid, lazy[node]);
16          apply(node * 2 + 1, mid + 1, r, lazy[node]);
17          lazy[node] = 0;
18      }
19
20      void rangeAdd(int node, int l, int r, int ql, int qr, long long val) {
21          if (ql <= l && r <= qr) {
```

```
            apply(node, l, r, val);
            return;
        }
        push(node, l, r);
        int mid = (l + r) / 2;
        if (ql <= mid) rangeAdd(node * 2, l, mid, ql, qr, val);
        if (qr > mid) rangeAdd(node * 2 + 1, mid + 1, r, ql, qr, val);
        tree[node] = tree[node * 2] + tree[node * 2 + 1];
    }

    long long query(int node, int l, int r, int ql, int qr) {
        if (ql <= l && r <= qr) return tree[node];
        push(node, l, r);
        int mid = (l + r) / 2;
        long long res = 0;
        if (ql <= mid) res += query(node * 2, l, mid, ql, qr);
        if (qr > mid) res += query(node * 2 + 1, mid + 1, r, ql, qr);
        return res;
    }
};
```

调用：

```
SegTree seg(n);
seg.rangeAdd(1, 1, n, l, r, v);
long long ans = seg.query(1, 1, n, l, r);
```

### 8.9.3 不会线段树时怎么办

- 如果没有修改，用前缀和。
- 如果只有区间修改、最后查询，用差分。
- 如果是单点修改、区间查询，用树状数组。
- 如果数据小，直接暴力扫描。

## 8.10 Kruskal 最小生成树入口

### 8.10.1 识别信号

- 连接所有点。
- 总代价最小。
- 无向图。
- 选择若干边让图连通。

### 8.10.2 Kruskal 模板

```cpp
struct Edge {
    int u, v;
    long long w;
};

sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {
    return a.w < b.w;
});

DSU dsu(n);
long long ans = 0;
int cnt = 0;
for (auto e : edges) {
    if (dsu.unite(e.u, e.v)) {
        ans += e.w;
        cnt++;
    }
}

if (cnt == n - 1) cout << ans << '\n';
else cout << "Impossible\n";
```

## 8.11 树上问题入口

### 8.11.1 识别信号

- 给定一棵树。
- 查询两点路径。
- 查询祖先、最近公共祖先。
- 对树上路径加值或统计经过次数。

### 8.11.2 树上暴力拿分

如果不会 LCA，且 $n, q$ 小，可以每次 DFS 找路径。

```cpp
bool dfsPath(int u, int parent, int target, vector<int>& path) {
    if (u == target) {
        path.push_back(u);
        return true;
    }
    for (int v : g[u]) {
        if (v == parent) continue;
        if (dfsPath(v, u, target, path)) {
            path.push_back(u);
```

```
            return true;
        }
    }
    return false;
}
```

### 8.11.3 LCA 倍增入口

```
const int LOG = 20;
vector<vector<int>> up;
vector<int> depth;
vector<vector<int>> g;

void dfsLca(int u, int p) {
    up[0][u] = p;
    for (int k = 1; k < LOG; k++) {
        up[k][u] = up[k - 1][up[k - 1][u]];
    }
    for (int v : g[u]) {
        if (v == p) continue;
        depth[v] = depth[u] + 1;
        dfsLca(v, u);
    }
}

int lca(int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    int diff = depth[a] - depth[b];
    for (int k = 0; k < LOG; k++) {
        if (diff & (1 << k)) a = up[k][a];
    }
    if (a == b) return a;
    for (int k = LOG - 1; k >= 0; k--) {
        if (up[k][a] != up[k][b]) {
            a = up[k][a];
            b = up[k][b];
        }
    }
    return up[0][a];
}
```

初始化：

```
up.assign(LOG, vector<int>(n + 1));
depth.assign(n + 1, 0);
dfsLca(1, 1);
```

### 8.11.4 树上差分入口

多次给路径 $(u, v)$ 加 1，最后统计每个点或边被经过次数。

点差分思路：

```
vector<long long> diff(n + 1, 0);

void addPath(int u, int v) {
    int w = lca(u, v);
    diff[u]++;
    diff[v]++;
    diff[w]--;
    diff[up[0][w]]--;
}

void collect(int u, int p) {
    for (int v : g[u]) {
        if (v == p) continue;
        collect(v, u);
        diff[u] += diff[v];
    }
}
```

**树上差分分点权和边权两种，公式不同。不会时先不要硬套，至少用暴力路径拿小数据分。**

## 8.12 状态压缩 DP 入口

### 8.12.1 识别信号

- $n$ 很小，例如 $n \le 20$。
- 每个元素只有选或不选。
- 状态可以用二进制集合表示。
- 旅行、分组、覆盖、集合最优值。

### 8.12.2 集合 DP 基本形状

```
int N = 1 << n;
vector<int> dp(N, INF);
dp[0] = 0;

for (int mask = 0; mask < N; mask++) {
    for (int i = 0; i < n; i++) {
        if ((mask & (1 << i)) == 0) {
            int nmask = mask | (1 << i);
            dp[nmask] = min(dp[nmask], dp[mask] + cost[i]);
        }
    }
```

```
}
```

### 8.12.3 状压易错点

- $1 << n$ 当 $n \geq 31$ 会溢出，要用 `1LL << n`。
- 状态数是 $2^n$，$n = 25$ 已经很大。
- 数组大小和内存要提前估算。

## 8.13 KMP 字符串匹配入口

### 8.13.1 识别信号

- 长文本中多次查找模式串。
- 字符串长度很大，暴力匹配会超时。
- 需要统计模式串出现次数。

### 8.13.2 KMP 模板

```cpp
vector<int> prefixFunction(const string& p) {
    int n = (int)p.size();
    vector<int> pi(n, 0);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && p[i] != p[j]) j = pi[j - 1];
        if (p[i] == p[j]) j++;
        pi[i] = j;
    }
    return pi;
}

int countMatch(const string& text, const string& p) {
    vector<int> pi = prefixFunction(p);
    int j = 0;
    int cnt = 0;
    for (char c : text) {
        while (j > 0 && c != p[j]) j = pi[j - 1];
        if (c == p[j]) j++;
        if (j == (int)p.size()) {
            cnt++;
            j = pi[j - 1];
        }
    }
    return cnt;
}
```

### 8.13.3 不会 KMP 时

- 字符串短时用暴力匹配。
- C++ 可用 `s.find(t)` 拿部分数据。
- 多模式串匹配可能要 Trie 或 AC 自动机，但新手优先拿小数据。

## 8.14 Trie 字典树入口

### 8.14.1 识别信号

- 很多字符串插入和查询。
- 前缀查询。
- 字典、单词集合。

### 8.14.2 小写字母 Trie

```cpp
struct Trie {
    struct Node {
        int child[26];
        bool end = false;
        Node() {
            memset(child, -1, sizeof child);
        }
    };

    vector<Node> tr;

    Trie() {
        tr.push_back(Node());
    }

    void insert(const string& s) {
        int u = 0;
        for (char c : s) {
            int x = c - 'a';
            if (tr[u].child[x] == -1) {
                tr[u].child[x] = (int)tr.size();
                tr.push_back(Node());
            }
            u = tr[u].child[x];
        }
        tr[u].end = true;
    }

    bool find(const string& s) {
        int u = 0;
```

```
        for (char c : s) {
            int x = c - 'a';
            if (tr[u].child[x] == -1) return false;
            u = tr[u].child[x];
        }
        return tr[u].end;
    }
};
```

### 8.14.3 不会 Trie 时

- 只查完整字符串，用 `set<string>` 或 `unordered_set<string>`。
- 数据小，用逐个字符串比较。
- 前缀查询可以用排序后 `lower_bound` 尝试部分分。

## 8.15 D/E 考场时间策略

| 剩余时间 | 建议动作 | 目标 |
|---|---|---|
| 60 分钟以上 | 先思考满分算法，再写部分分保底 | 冲高分 |
| 30–60 分钟 | 先写暴力或特殊性质，再看能否优化 | 稳定拿分 |
| 15–30 分钟 | 只写最确定的小数据做法 | 不空题 |
| 15 分钟以内 | 写样例能过的最小实现，清理输出 | 抢分 |

## 8.16 常见高级算法入口

| 题面信号 | 可能算法 | 先翻哪里 |
|---|---|---|
| 最大化最小值、最小化最大值 | 二分答案 | B 题二分答案 |
| 区间修改查询交替 | 线段树 / 树状数组 | 后续数据结构章 |
| 树上路径、祖先 | LCA / 树上差分 | 树专题 |
| 强连通、缩点 | Tarjan | 图论专题 |
| 最小连接代价 | 最小生成树 | Kruskal / Prim |
| 网络最短距离 | Dijkstra / Floyd | B 题图论入口 |
| 容量和价值 | 背包 DP | B 题 DP 入口 |
| 集合状态很小 | 状态压缩 DP | 状压入口 |
| 字符串匹配很多次 | KMP / Trie | 字符串专题 |

## 8.17 高级算法识别速查

| 关键词 | 优先想到 | 不会时的替代 |
|---|---|---|
| 区间加、区间查询 | 线段树 | 差分、树状数组、暴力 |
| 单点修改、区间和 | 树状数组 | 前缀和静态版、暴力 |
| 树上路径查询 | LCA | 每次 DFS 找路径 |
| 多次树上路径加 | 树上差分 | 暴力路径加 |
| 连接所有点最小代价 | Kruskal | 枚举小图、Prim 小数据 |
| 集合状态很小 | 状压 DP | DFS 枚举 |
| 长串匹配短串 | KMP | `find` / 暴力 |
| 前缀查询很多 | Trie | set / 排序后二分 |
| 最大化最小值 | 二分答案 | 枚举答案小范围 |
| 任务依赖 | 拓扑排序 | 小图 DFS 检查 |

## 8.18 D/E 模板练习优先级

| 模板 | 优先级 | 原因 |
|---|---|---|
| 二分答案 | 高 | B/D 都常见，代码短，提分明显。 |
| 树状数组 | 高 | 区间题常见，比线段树简单。 |
| Dijkstra | 高 | 图论常见，B/D 都可能用。 |
| Kruskal | 中 | 最小生成树题专用，模板短。 |
| 线段树 | 中高 | 强但代码长，建议练熟基础版。 |
| LCA | 中 | 树上题常见，但新手可先会暴力。 |
| 状态压缩 DP | 中 | 看到 $n \le 20$ 很有用。 |
| KMP | 中低 | 字符串题有用，但 CSP 中不一定高频。 |
| Trie | 中低 | 前缀题有用，完整字符串可用 set 替代。 |
| 树上差分 | 低到中 | 需要 LCA 基础，适合冲高分。 |

## 8.19 什么时候放弃满分

出现以下情况，应转为部分分：

- 20 分钟想不出核心算法。
- 会算法名字，但模板完全不会写。
- 写到一半发现状态定义不清。
- 样例都无法解释。
- A/B/C 还没稳。

## 8.20 D/E 题提交前检查

- 是否至少能过样例。
- 是否能过自己造的小数据。
- 是否处理了最小输入。
- 如果是暴力，是否明确知道能拿哪些数据。
- 是否删除调试输出。
- 是否没有因为追求优化改坏暴力基础逻辑。

# 第三部分

# 模板与检查

# 第九章　高频算法模板库

## 9.1　模板库使用方法

本章把前面章节中反复出现的算法模板统一编号。考场上建议先通过决策树确定题型，再来本章找对应模板。

- `T001--T099`：基础实现和 STL。
- `T100--T199`：前缀和、差分、排序、哈希、二分。
- `T200--T299`：搜索和图论。
- `T300--T399`：动态规划。
- `T400--T499`：数据结构。
- `T500--T599`：字符串、数学、日期。

## 9.2　模板速查表

| 编号 | 名称 | 识别信号 | 复杂度 |
|---|---|---|---|
| T101 | 一维前缀和 | 多次静态区间和查询 | 预处理 $O(n)$，查询 $O(1)$ |
| T102 | 二维前缀和 | 多次子矩阵和查询 | 预处理 $O(nm)$，查询 $O(1)$ |
| T103 | 一维差分 | 多次区间加，最后查询 | $O(n+q)$ |
| T104 | 二维差分 | 多次矩形加，最后输出 | $O(nm+q)$ |
| T105 | 排序自定义比较器 | 多关键字排序、排名 | $O(n\log n)$ |
| T106 | 哈希统计 | 出现次数、查重、频率 | 平均 $O(n)$ |
| T107 | 二分答案 | 最大化最小值、最小化最大值 | $O(\log V\cdot check)$ |
| T108 | 双指针 | 排序后找对、滑动窗口 | $O(n)$ 或 $O(n\log n)$ |
| T109 | 坐标离散化 | 坐标大但点数少 | $O(n\log n)$ |
| T110 | 事件排序 | 按时间发生、扫描线 | $O(n\log n)$ |
| T201 | 网格 BFS | 最短步数、迷宫 | $O(nm)$ |
| T202 | DFS 连通块 | 区域大小、连通块数量 | $O(n+m)$ 或 $O(nm)$ |
| T203 | 并查集 | 合并集合、判断同组 | 近似 $O(1)$ |

| | | | |
|---|---|---|---|
| T204 | Dijkstra | 正权最短路 | $O((n+m)\log n)$ |
| T205 | 拓扑排序 | 依赖关系、先后顺序 | $O(n+m)$ |
| T206 | Floyd | 小图任意两点最短路 | $O(n^3)$ |
| T207 | Kruskal | 最小生成树 | $O(m\log m)$ |
| T208 | LCA 倍增 | 树上最近公共祖先 | $O(n\log n)$ 预处理，$O(\log n)$ 查询 |
| T301 | 01 背包 | 每个物品选或不选 | $O(nV)$ |
| T302 | 完全背包 | 每个物品可选多次 | $O(nV)$ |
| T401 | 树状数组 | 单点修改、区间和 | $O(\log n)$ |
| T402 | 线段树 | 区间修改、区间查询 | $O(\log n)$ |
| T403 | 单调栈 | 最近更大/更小元素 | $O(n)$ |
| T404 | 单调队列 | 滑动窗口最大/最小 | $O(n)$ |
| T501 | split 字符串分割 | 路径、配置、命令解析 | $O(\|s\|)$ |
| T502 | KMP | 长文本匹配模式串 | $O(n+m)$ |
| T503 | Trie | 字符串集合、前缀查询 | $O(\sum \|s\|)$ |
| T510 | 整除取整和标准取模 | 分组数、负数取模、数学 floor/ceil | $O(1)$ |
| T511 | 快速幂 | $a^b \bmod m$ | $O(\log b)$ |
| T512 | 素数和约数 | 判断素数、枚举约数 | $O(\sqrt{n})$ |
| T513 | 进制转换 | 任意进制与十进制 | $O(\text{位数})$ |
| T514 | 日期处理 | 闰年、每月天数、日期推进 | 按实现而定 |
| T601 | 大模拟骨架 | 多对象、多命令、长题面 | 按题目而定 |
| T602 | 命令分发 | 每行一个操作名 | 按命令复杂度 |
| T603 | 状态机 | 对象多状态切换 | 按操作复杂度 |
| T604 | 懒删除优先队列 | 动态删除/更新后仍取最优 | $O(\log n)$ 均摊 |
| T605 | LRU 缓存 | 最近最少使用淘汰 | $O(1)$ 均摊 |
| T606 | 路径规范化 | `/a/../b`、目录路径 | $O(\|s\|)$ |
| T607 | URL 参数解析 | `path?a=1&b=2` | $O(\|s\|)$ |
| T608 | 括号匹配 | 括号合法性、嵌套结构 | $O(n)$ |
| T609 | 简单表达式求值 | 只含加减的表达式 | $O(n)$ |

## 9.3 T101 一维前缀和

### 9.3.1 适用场景

- 数组不修改。
- 多次查询 $[l,r]$ 的和。
- 需要快速得到一段累计值。

### 9.3.2 模板

```
int n, q;
```

```
cin >> n >> q;
vector<long long> pre(n + 1, 0);

for (int i = 1; i <= n; i++) {
    long long x;
    cin >> x;
    pre[i] = pre[i - 1] + x;
}

while (q--) {
    int l, r;
    cin >> l >> r;
    cout << pre[r] - pre[l - 1] << '\n';
}
```

### 9.3.3 易错点

- 模板默认下标从 1 开始。
- pre[0] 是 0。
- 求和用 long long。
- 如果有修改，普通前缀和不适用。

## 9.4 T102 二维前缀和

### 9.4.1 适用场景

多次查询子矩阵和。

```
int n, m, q;
cin >> n >> m >> q;
vector<vector<long long>> pre(n + 1, vector<long long>(m + 1, 0));

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        long long x;
        cin >> x;
        pre[i][j] = pre[i - 1][j] + pre[i][j - 1]
                - pre[i - 1][j - 1] + x;
    }
}

while (q--) {
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;
    long long ans = pre[x2][y2] - pre[x1 - 1][y2]
                - pre[x2][y1 - 1] + pre[x1 - 1][y1 - 1];
```

```
    cout << ans << '\n';
}
```

### 9.4.2 易错点

- 四项公式不要写反。
- 默认坐标从 1 开始。
- 子矩阵端点是否保证 $x1 \le x2, y1 \le y2$。

## 9.5 T103 一维差分

### 9.5.1 适用场景

多次区间加，最后统一输出每个位置的值。

```
int n, q;
cin >> n >> q;
vector<long long> diff(n + 2, 0);

while (q--) {
    int l, r;
    long long v;
    cin >> l >> r >> v;
    diff[l] += v;
    diff[r + 1] -= v;
}

long long cur = 0;
for (int i = 1; i <= n; i++) {
    cur += diff[i];
    cout << cur << '\n';
}
```

### 9.5.2 易错点

- 数组开到 n+2。
- 如果修改后立刻查询区间和，普通差分不够。
- 修改值用 long long。

## 9.6 T104 二维差分

### 9.6.1 适用场景

多次对子矩形加值，最后输出整个矩阵。

```
int n, m, q;
cin >> n >> m >> q;
vector<vector<long long>> diff(n + 2, vector<long long>(m + 2, 0));

while (q--) {
    int x1, y1, x2, y2;
    long long v;
    cin >> x1 >> y1 >> x2 >> y2 >> v;
    diff[x1][y1] += v;
    diff[x2 + 1][y1] -= v;
    diff[x1][y2 + 1] -= v;
    diff[x2 + 1][y2 + 1] += v;
}

for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        diff[i][j] += diff[i - 1][j] + diff[i][j - 1]
                    - diff[i - 1][j - 1];
    }
}
```

## 9.7 T105 排序自定义比较器

### 9.7.1 适用场景

多关键字排序、排名、优先级排序。

```
struct Node {
    int id;
    int score;
    int time;
};

sort(a.begin(), a.end(), [](const Node& x, const Node& y) {
    if (x.score != y.score) return x.score > y.score;
    if (x.time != y.time) return x.time < y.time;
    return x.id < y.id;
});
```

### 9.7.2 易错点

- 相等时不能返回 `true`。
- 每个关键字升序还是降序要写清楚。
- 需要保持原顺序时用 `stable_sort`。

## 9.8 T106 哈希统计

### 9.8.1 值域小

```
vector<int> cnt(100001, 0);
for (int x : a) {
    cnt[x]++;
}
```

### 9.8.2 值域大或字符串

```
unordered_map<string, int> cnt;
for (string s : words) {
    cnt[s]++;
}
```

### 9.8.3 易错点

- 需要有序输出时用 map。
- key 很大不能开计数数组。
- cnt[x] 会自动创建 key。

## 9.9 T107 二分答案

### 9.9.1 适用场景

- 求最小可行值。
- 求最大可行值。
- 给定答案 $mid$ 能判断是否可行。
- 可行性具有单调性。

### 9.9.2 最小可行值

```
bool check(long long mid) {
    // return true if mid is feasible
}

long long l = 0, r = 1000000000000000000LL;
while (l < r) {
    long long mid = l + (r - l) / 2;
    if (check(mid)) r = mid;
    else l = mid + 1;
}
cout << l << '\n';
```

### 9.9.3 最大可行值

```cpp
long long l = 0, r = 10000000000000000000LL;
while (l < r) {
    long long mid = l + (r - l + 1) / 2;
    if (check(mid)) l = mid;
    else r = mid - 1;
}
cout << l << '\n';
```

**二分答案必须有单调性。**

## 9.10 T108 双指针

### 9.10.1 排序后找两数和

```cpp
sort(a.begin(), a.end());
int l = 0, r = n - 1;
bool ok = false;

while (l < r) {
    long long sum = 1LL * a[l] + a[r];
    if (sum == target) {
        ok = true;
        break;
    } else if (sum < target) {
        l++;
    } else {
        r--;
    }
}
```

### 9.10.2 非负数组滑动窗口

```cpp
int l = 0;
long long sum = 0;
int best = 0;

for (int r = 0; r < n; r++) {
    sum += a[r];
    while (sum > limit && l <= r) {
        sum -= a[l];
        l++;
    }
    best = max(best, r - l + 1);
}
```

**滑动窗口通常要求元素非负或条件具有单调性。**

## 9.11 T109 坐标离散化

### 9.11.1 适用场景

坐标值很大，但出现的坐标数量不多。

```
vector<int> xs;
for (int x : original) {
    xs.push_back(x);
}

sort(xs.begin(), xs.end());
xs.erase(unique(xs.begin(), xs.end()), xs.end());

auto getId = [&](int x) {
    return (int)(lower_bound(xs.begin(), xs.end(), x) - xs.begin()) + 1;
};
```

**如果题目涉及区间长度，不能只看离散编号，还要保留真实坐标差。**

## 9.12 T110 事件排序

### 9.12.1 适用场景

按时间顺序处理事件，或扫描区间开始/结束。

```
struct Event {
    int time;
    int type;
    int id;
};

sort(events.begin(), events.end(), [](const Event& a, const Event& b) {
    if (a.time != b.time) return a.time < b.time;
    return a.type < b.type;
});

for (const Event& e : events) {
    if (e.type == 0) {
        // start
    } else {
        // end
    }
}
```

### 9.12.2 区间同时进行数量

```
vector<pair<int, int>> events;
```

```
for (auto [l, r] : intervals) {
    events.push_back({l, 1});
    events.push_back({r, -1});
}

sort(events.begin(), events.end(), [](auto a, auto b) {
    if (a.first != b.first) return a.first < b.first;
    return a.second < b.second; // end before start for [l,r)
});

int cur = 0, best = 0;
for (auto [t, delta] : events) {
    cur += delta;
    best = max(best, cur);
}
```

**同一时间开始和结束谁先处理，要按题意决定。**

## 9.13 T201 网格 BFS

### 9.13.1 适用场景

迷宫、最短步数、每一步代价相同。

```
int n, m;
cin >> n >> m;
vector<string> g(n);
for (int i = 0; i < n; i++) cin >> g[i];

int sx, sy, tx, ty;
cin >> sx >> sy >> tx >> ty;

vector<vector<int>> dist(n, vector<int>(m, -1));
queue<pair<int, int>> q;

dist[sx][sy] = 0;
q.push({sx, sy});

int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};

while (!q.empty()) {
    auto [x, y] = q.front();
    q.pop();

    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d];
        int ny = y + dy[d];
```

```
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (g[nx][ny] == '#') continue;
        if (dist[nx][ny] != -1) continue;
        dist[nx][ny] = dist[x][y] + 1;
        q.push({nx, ny});
    }
}

cout << dist[tx][ty] << '\n';
```

### 9.13.2 易错点

- 坐标是 0 下标还是 1 下标。
- 入队时标记访问。
- 边权不全相等时不要用普通 BFS。

## 9.14 T202 DFS 连通块

### 9.14.1 网格 DFS

```
int n, m;
vector<string> g;
vector<vector<int>> vis;
int dx[4] = {-1, 0, 1, 0};
int dy[4] = {0, 1, 0, -1};

int dfs(int x, int y) {
    vis[x][y] = 1;
    int area = 1;
    for (int d = 0; d < 4; d++) {
        int nx = x + dx[d];
        int ny = y + dy[d];
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
        if (vis[nx][ny]) continue;
        if (g[nx][ny] == '#') continue;
        area += dfs(nx, ny);
    }
    return area;
}
```

### 9.14.2 易错点

- 大网格递归可能爆栈，可改 BFS。
- 障碍字符要按题目改。

## 9.15 T203 并查集

```cpp
struct DSU {
    vector<int> parent, sz;

    DSU(int n) {
        parent.resize(n + 1);
        sz.assign(n + 1, 1);
        for (int i = 1; i <= n; i++) parent[i] = i;
    }

    int find(int x) {
        if (parent[x] == x) return x;
        return parent[x] = find(parent[x]);
    }

    bool unite(int a, int b) {
        int ra = find(a);
        int rb = find(b);
        if (ra == rb) return false;
        if (sz[ra] < sz[rb]) swap(ra, rb);
        parent[rb] = ra;
        sz[ra] += sz[rb];
        return true;
    }

    bool same(int a, int b) {
        return find(a) == find(b);
    }
};
```

### 9.15.1 适用场景

合并集合、判断是否同组、连通块。

## 9.16 T204 Dijkstra

```cpp
struct Edge {
    int to;
    long long w;
};

const long long INF = (long long)4e18;

int n, m, s;
cin >> n >> m >> s;
```

```
vector<vector<Edge>> g(n + 1);

for (int i = 0; i < m; i++) {
    int u, v;
    long long w;
    cin >> u >> v >> w;
    g[u].push_back({v, w});
    g[v].push_back({u, w}); // remove if directed
}

vector<long long> dist(n + 1, INF);
priority_queue<pair<long long, int>,
               vector<pair<long long, int>>,
               greater<pair<long long, int>>> pq;

dist[s] = 0;
pq.push({0, s});

while (!pq.empty()) {
    auto [d, u] = pq.top();
    pq.pop();
    if (d != dist[u]) continue;

    for (auto e : g[u]) {
        if (dist[e.to] > dist[u] + e.w) {
            dist[e.to] = dist[u] + e.w;
            pq.push({dist[e.to], e.to});
        }
    }
}
```

### 9.16.1 易错点

- 不能处理负边权。
- 无向图双向加边，有向图单向加边。
- 距离用 long long。

## 9.17 T205 拓扑排序

### 9.17.1 适用场景

任务依赖、课程先修、判断有向图是否有环。

```
int n, m;
cin >> n >> m;
vector<vector<int>> g(n + 1);
vector<int> indeg(n + 1, 0);
```

```cpp

for (int i = 0; i < m; i++) {
    int u, v;
    cin >> u >> v; // u before v
    g[u].push_back(v);
    indeg[v]++;
}

queue<int> q;
for (int i = 1; i <= n; i++) {
    if (indeg[i] == 0) q.push(i);
}

vector<int> order;
while (!q.empty()) {
    int u = q.front();
    q.pop();
    order.push_back(u);
    for (int v : g[u]) {
        indeg[v]--;
        if (indeg[v] == 0) q.push(v);
    }
}

if ((int)order.size() == n) {
    // has topological order
} else {
    // has cycle
}
```

## 9.18 T206 Floyd

### 9.18.1 适用场景

点数小，任意两点最短路。

```cpp
const long long INF = (long long)4e18;

int n, m;
cin >> n >> m;
vector<vector<long long>> dist(n + 1, vector<long long>(n + 1, INF));

for (int i = 1; i <= n; i++) dist[i][i] = 0;

for (int i = 0; i < m; i++) {
    int u, v;
    long long w;
```

```
    cin >> u >> v >> w;
    dist[u][v] = min(dist[u][v], w);
    dist[v][u] = min(dist[v][u], w); // remove if directed
}

for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dist[i][k] == INF || dist[k][j] == INF) continue;
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
        }
    }
}
```

**三层循环顺序必须是 k,i,j。**

## 9.19 T207 Kruskal

### 9.19.1 适用场景

无向图，连接所有点的最小总代价。

```
struct Edge {
    int u, v;
    long long w;
};

vector<Edge> edges;
sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {
    return a.w < b.w;
});

DSU dsu(n);
long long ans = 0;
int cnt = 0;

for (auto e : edges) {
    if (dsu.unite(e.u, e.v)) {
        ans += e.w;
        cnt++;
    }
}

if (cnt == n - 1) cout << ans << '\n';
else cout << "Impossible\n";
```

## 9.20 T208 LCA 倍增

### 9.20.1 适用场景

树上最近公共祖先、树上路径查询。

```
const int LOG = 20;
vector<vector<int>> up;
vector<int> depth;
vector<vector<int>> g;

void dfsLca(int u, int p) {
    up[0][u] = p;
    for (int k = 1; k < LOG; k++) {
        up[k][u] = up[k - 1][up[k - 1][u]];
    }
    for (int v : g[u]) {
        if (v == p) continue;
        depth[v] = depth[u] + 1;
        dfsLca(v, u);
    }
}

int lca(int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    int diff = depth[a] - depth[b];
    for (int k = 0; k < LOG; k++) {
        if (diff & (1 << k)) a = up[k][a];
    }
    if (a == b) return a;
    for (int k = LOG - 1; k >= 0; k--) {
        if (up[k][a] != up[k][b]) {
            a = up[k][a];
            b = up[k][b];
        }
    }
    return up[0][a];
}
```

初始化：

```
up.assign(LOG, vector<int>(n + 1));
depth.assign(n + 1, 0);
dfsLca(1, 1);
```

## 9.21 T301 01 背包

```
int n, V;
```

```
cin >> n >> V;
vector<int> w(n + 1), val(n + 1);
for (int i = 1; i <= n; i++) {
    cin >> w[i] >> val[i];
}

vector<int> dp(V + 1, 0);
for (int i = 1; i <= n; i++) {
    for (int j = V; j >= w[i]; j--) {
        dp[j] = max(dp[j], dp[j - w[i]] + val[i]);
    }
}
cout << dp[V] << '\n';
```

**01 背包容量倒序。**

## 9.22 T302 完全背包

```
int n, V;
cin >> n >> V;
vector<int> w(n + 1), val(n + 1);
for (int i = 1; i <= n; i++) {
    cin >> w[i] >> val[i];
}

vector<int> dp(V + 1, 0);
for (int i = 1; i <= n; i++) {
    for (int j = w[i]; j <= V; j++) {
        dp[j] = max(dp[j], dp[j - w[i]] + val[i]);
    }
}
cout << dp[V] << '\n';
```

**完全背包容量正序。**

## 9.23 T401 树状数组

```
struct BIT {
    int n;
    vector<long long> tree;

    BIT(int n) : n(n), tree(n + 1, 0) {}

    void add(int idx, long long val) {
        for (; idx <= n; idx += idx & -idx) {
            tree[idx] += val;
        }
```

```
    }

    long long sumPrefix(int idx) {
        long long res = 0;
        for (; idx > 0; idx -= idx & -idx) {
            res += tree[idx];
        }
        return res;
    }

    long long rangeSum(int l, int r) {
        return sumPrefix(r) - sumPrefix(l - 1);
    }
};
```

### 9.23.1 适用场景

单点修改、区间和查询。

## 9.24 T402 线段树

### 9.24.1 适用场景

区间加、区间和查询。模板较长，只有练过才建议考场使用。

```
struct SegTree {
    int n;
    vector<long long> tree, lazy;

    SegTree(int n) : n(n), tree(4 * n + 4), lazy(4 * n + 4) {}

    void apply(int node, int l, int r, long long val) {
        tree[node] += val * (r - l + 1);
        lazy[node] += val;
    }

    void push(int node, int l, int r) {
        if (lazy[node] == 0 || l == r) return;
        int mid = (l + r) / 2;
        apply(node * 2, l, mid, lazy[node]);
        apply(node * 2 + 1, mid + 1, r, lazy[node]);
        lazy[node] = 0;
    }

    void rangeAdd(int node, int l, int r, int ql, int qr, long long val) {
        if (ql <= l && r <= qr) {
            apply(node, l, r, val);
            return;
```

```
        }
        push(node, l, r);
        int mid = (l + r) / 2;
        if (ql <= mid) rangeAdd(node * 2, l, mid, ql, qr, val);
        if (qr > mid) rangeAdd(node * 2 + 1, mid + 1, r, ql, qr, val);
        tree[node] = tree[node * 2] + tree[node * 2 + 1];
    }

    long long query(int node, int l, int r, int ql, int qr) {
        if (ql <= l && r <= qr) return tree[node];
        push(node, l, r);
        int mid = (l + r) / 2;
        long long res = 0;
        if (ql <= mid) res += query(node * 2, l, mid, ql, qr);
        if (qr > mid) res += query(node * 2 + 1, mid + 1, r, ql, qr);
        return res;
    }
};
```

调用：

```
SegTree seg(n);
seg.rangeAdd(1, 1, n, l, r, v);
long long ans = seg.query(1, 1, n, l, r);
```

### 9.24.2 易错点

- 默认区间是 1 到 $n$。
- 懒标记下传不要漏。
- 求和用 `long long`。

## 9.25 T403 单调栈

### 9.25.1 适用场景

找每个元素右边第一个更大元素、左边第一个更小元素等。

```
vector<int> ans(n, -1);
stack<int> st; // indices

for (int i = 0; i < n; i++) {
    while (!st.empty() && a[i] > a[st.top()]) {
        ans[st.top()] = i;
        st.pop();
    }
    st.push(i);
}
```

### 9.25.2 易错点

- 比较符号决定严格更大还是大于等于。
- 栈里通常存下标，不是值。
- 全相等数据要测试。

## 9.26 T404 单调队列

### 9.26.1 适用场景

固定长度滑动窗口最大值或最小值。

```
deque<int> dq; // indices, values decreasing
for (int i = 0; i < n; i++) {
    while (!dq.empty() && dq.front() <= i - k) dq.pop_front();
    while (!dq.empty() && a[dq.back()] <= a[i]) dq.pop_back();
    dq.push_back(i);

    if (i >= k - 1) {
        cout << a[dq.front()] << '\n';
    }
}
```

### 9.26.2 易错点

- 先删除过期下标。
- 最大值队列保持递减，最小值队列保持递增。
- 输出窗口从 `i >= k - 1` 开始。

## 9.27 T501 split 字符串分割

### 9.27.1 按指定字符分割

```
vector<string> split(const string& s, char sep) {
    vector<string> res;
    string cur;
    for (char c : s) {
        if (c == sep) {
            res.push_back(cur);
            cur.clear();
        } else {
            cur.push_back(c);
        }
    }
    res.push_back(cur);
```

```
    return res;
}
```

### 9.27.2 按空格分割

```
stringstream ss(line);
string word;
while (ss >> word) {
    // use word
}
```

## 9.28 T502 KMP

```
vector<int> prefixFunction(const string& p) {
    int n = (int)p.size();
    vector<int> pi(n, 0);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && p[i] != p[j]) j = pi[j - 1];
        if (p[i] == p[j]) j++;
        pi[i] = j;
    }
    return pi;
}

int countMatch(const string& text, const string& p) {
    vector<int> pi = prefixFunction(p);
    int j = 0, cnt = 0;
    for (char c : text) {
        while (j > 0 && c != p[j]) j = pi[j - 1];
        if (c == p[j]) j++;
        if (j == (int)p.size()) {
            cnt++;
            j = pi[j - 1];
        }
    }
    return cnt;
}
```

## 9.29 T503 Trie

```
struct Trie {
    struct Node {
        int child[26];
```

```
    bool end = false;
    Node() {
        memset(child, -1, sizeof child);
    }
};

vector<Node> tr;

Trie() {
    tr.push_back(Node());
}

void insert(const string& s) {
    int u = 0;
    for (char c : s) {
        int x = c - 'a';
        if (tr[u].child[x] == -1) {
            tr[u].child[x] = (int)tr.size();
            tr.push_back(Node());
        }
        u = tr[u].child[x];
    }
    tr[u].end = true;
}

bool find(const string& s) {
    int u = 0;
    for (char c : s) {
        int x = c - 'a';
        if (tr[u].child[x] == -1) return false;
        u = tr[u].child[x];
    }
    return tr[u].end;
}
};
```

**这个 Trie 模板只适合小写英文字母。**

## 9.30 T510 整除取整和标准取模

### 9.30.1 适用场景

- 分组、分页、分块，要求正整数向上取整。
- 坐标、时间差、偏移量可能为负，要求数学意义的向上或向下取整。
- 同余、周期、哈希下标中需要把负数余数转成 0 到 $m-1$。

### 9.30.2 正整数向上取整

若 $a \geq 0, b > 0$:

```
long long ceil_div_positive(long long a, long long b) {
    return (a + b - 1) / b;
}
```

示例:

```
long long pages = ceil_div_positive(n, pageSize);
long long blocks = ceil_div_positive(n, blockSize);
```

### 9.30.3 有符号通用版本

```
long long floor_div(long long a, long long b) {
    long long q = a / b;
    long long r = a % b;
    if (r != 0 && ((r > 0) != (b > 0))) q--;
    return q;
}

long long ceil_div(long long a, long long b) {
    long long q = a / b;
    long long r = a % b;
    if (r != 0 && ((r > 0) == (b > 0))) q++;
    return q;
}
```

### 9.30.4 标准非负余数

```
long long norm_mod(long long x, long long m) {
    long long r = x % m;
    if (r < 0) r += m;
    return r;
}
```

### 9.30.5 易错点

- `(a+b-1)/b` 只适合 $a \geq 0, b > 0$，不是通用向上取整。
- C++ 整数除法向 0 截断，`-3 / 2` 是 `-1`。
- C++ 负数取模可能得到负数，`-1 % 5` 是 `-1`。
- 除数 `b` 或模数 `m` 不能为 0。
- 若 `a + b - 1` 可能溢出，需要改写或先确认范围。

# 9.31 T511 快速幂

## 9.31.1 适用场景

计算 $a^b \bmod mod$。

```
long long modPow(long long a, long long b, long long mod) {
    long long res = 1 % mod;
    a %= mod;
    while (b > 0) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}
```

# 9.32 T512 素数和约数

## 9.32.1 gcd 和 lcm

```
long long g = gcd(a, b);
long long l = a / g * b;
```

## 9.32.2 判断素数

```
bool isPrime(long long x) {
    if (x < 2) return false;
    for (long long d = 2; d * d <= x; d++) {
        if (x % d == 0) return false;
    }
    return true;
}
```

## 9.32.3 枚举约数

```
vector<long long> divisors;
for (long long d = 1; d * d <= x; d++) {
    if (x % d == 0) {
        divisors.push_back(d);
        if (d != x / d) divisors.push_back(x / d);
    }
}
sort(divisors.begin(), divisors.end());
```

## 9.33 T513 进制转换

### 9.33.1 任意进制转十进制

```cpp
long long toDecimal(const string& s, int base) {
    long long ans = 0;
    for (char c : s) {
        int v;
        if ('0' <= c && c <= '9') v = c - '0';
        else if ('A' <= c && c <= 'F') v = c - 'A' + 10;
        else v = c - 'a' + 10;
        ans = ans * base + v;
    }
    return ans;
}
```

### 9.33.2 十进制转任意进制

```cpp
string fromDecimal(long long x, int base) {
    if (x == 0) return "0";
    string digits = "0123456789ABCDEF";
    string res;
    while (x > 0) {
        res.push_back(digits[x % base]);
        x /= base;
    }
    reverse(res.begin(), res.end());
    return res;
}
```

**数字很长时不能转成 long long，要用字符串模拟。**

## 9.34 T514 日期处理

### 9.34.1 闰年

```cpp
bool leap(int y) {
    return (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);
}
```

### 9.34.2 每月天数

```cpp
int mdays(int y, int m) {
    int d[] = {0,31,28,31,30,31,30,31,31,30,31,30,31};
    if (m == 2 && leap(y)) return 29;
    return d[m];
```

```
}
```

### 9.34.3 下一天

```
void nextDay(int& y, int& m, int& d) {
    d++;
    if (d > mdays(y, m)) {
        d = 1;
        m++;
        if (m > 12) {
            m = 1;
            y++;
        }
    }
}
```

### 9.34.4 时间转秒

```
int toSeconds(int h, int m, int s) {
    return h * 3600 + m * 60 + s;
}
```

## 9.35 T601 大模拟骨架

### 9.35.1 适用场景

C 题长题面、多对象、多命令、多状态。

```
struct Item {
    int id;
    string name;
    int state;
};

int n, q;
vector<Item> items;

void readInput() {
    cin >> n >> q;
    items.resize(n);
    for (int i = 0; i < n; i++) {
        cin >> items[i].id >> items[i].name;
        items[i].state = 0;
    }
}

```

```
void solve() {
    while (q--) {
        string op;
        cin >> op;
        // dispatch command
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    readInput();
    solve();
    return 0;
}
```

## 9.36 T602 命令分发

```
void handleAdd() {
    int id, x;
    cin >> id >> x;
    // add logic
}

void handleDelete() {
    int id;
    cin >> id;
    // delete logic
}

void handleQuery() {
    int id;
    cin >> id;
    // query logic
}

void handleCommand() {
    string op;
    cin >> op;
    if (op == "add") handleAdd();
    else if (op == "delete") handleDelete();
    else if (op == "query") handleQuery();
}
```

### 9.36.1 易错点

- 每种命令的参数个数不同。
- 命令参数可能包含空格时要用 `getline`。
- 查询命令通常直接输出，修改命令通常不输出。

## 9.37 T603 状态机

```
const int WAITING = 0;
const int RUNNING = 1;
const int FINISHED = 2;

struct Job {
    int id;
    int status;
    int score;
};

void start(Job& job) {
    if (job.status != WAITING) return;
    job.status = RUNNING;
}

void finish(Job& job) {
    if (job.status != RUNNING) return;
    job.status = FINISHED;
    job.score += 10;
}
```

### 9.37.1 检查点

- 初始状态是什么。
- 每个操作允许在哪些状态发生。
- 非法状态转换是忽略还是输出错误。
- 状态变化时是否要同步修改其他字段。

## 9.38 T604 懒删除优先队列

### 9.38.1 适用场景

动态更新或删除元素，同时需要不断取当前最优元素。

```
struct Task {
    int priority;
    int id;
```

```
};

struct Cmp {
    bool operator()(const Task& a, const Task& b) const {
        if (a.priority != b.priority) return a.priority < b.priority;
        return a.id > b.id;
    }
};

priority_queue<Task, vector<Task>, Cmp> pq;
map<int, int> currentPriority;

void addOrUpdate(int id, int priority) {
    currentPriority[id] = priority;
    pq.push({priority, id});
}

void removeTask(int id) {
    currentPriority.erase(id);
}

Task getTop() {
    while (!pq.empty()) {
        Task t = pq.top();
        if (currentPriority.count(t.id) &&
            currentPriority[t.id] == t.priority) {
            return t;
        }
        pq.pop();
    }
    return {-1, -1};
}
```

### 9.38.2 易错点

- 每次取堆顶前都要清理过期元素。
- 更新元素时直接压入新值，不删除旧值。
- 判断有效性必须和当前状态表一致。

## 9.39 T605 LRU 缓存

### 9.39.1 适用场景

最近最少使用淘汰，访问后变成最新。

```
int cap;
list<int> order; // front is most recent
```

```
unordered_map<int, list<int>::iterator> pos;

void access(int x) {
    if (cap == 0) return;

    if (pos.count(x)) {
        order.erase(pos[x]);
    } else if ((int)order.size() == cap) {
        int old = order.back();
        order.pop_back();
        pos.erase(old);
    }

    order.push_front(x);
    pos[x] = order.begin();
}
```

### 9.39.2 易错点

- 容量为 0 要特判。
- 删除链表节点后，map 里的迭代器也要更新。
- front 是最新还是最旧要统一。

## 9.40 T606 路径规范化

### 9.40.1 适用场景

文件路径、目录路径，处理 .、..、多余斜杠。

```
vector<string> normalizePath(const string& path) {
    vector<string> st;
    vector<string> parts = split(path, '/');
    for (string part : parts) {
        if (part.empty() || part == ".") {
            continue;
        } else if (part == "..") {
            if (!st.empty()) st.pop_back();
        } else {
            st.push_back(part);
        }
    }
    return st;
}

string joinPath(const vector<string>& parts) {
    if (parts.empty()) return "/";
    string res;
```

```
    for (const string& p : parts) {
        res += "/";
        res += p;
    }
    return res;
}
```

**如果题目允许相对路径，是否能越过根目录要按题意处理。**

## 9.41 T607 URL 参数解析

### 9.41.1 适用场景

形如 /path/to?a=1&b=2 的字符串。

```
struct Url {
    string path;
    map<string, string> query;
};

Url parseUrl(const string& s) {
    Url res;
    int qpos = (int)s.find('?');
    if (qpos == (int)string::npos) {
        res.path = s;
        return res;
    }

    res.path = s.substr(0, qpos);
    string qs = s.substr(qpos + 1);
    vector<string> pairs = split(qs, '&');
    for (string p : pairs) {
        int eq = (int)p.find('=');
        if (eq == (int)string::npos) {
            res.query[p] = "";
        } else {
            string key = p.substr(0, eq);
            string val = p.substr(eq + 1);
            res.query[key] = val;
        }
    }
    return res;
}
```

### 9.41.2 易错点

- URL 可能没有 ?。

- 参数值可能为空。
- 重复参数时保留第一个还是最后一个要看题目。

## 9.42 T608 括号匹配

### 9.42.1 适用场景

判断括号是否合法，或处理简单嵌套结构。

```cpp
bool checkBrackets(const string& s) {
    stack<char> st;
    for (char c : s) {
        if (c == '(' || c == '[' || c == '{') {
            st.push(c);
        } else if (c == ')' || c == ']' || c == '}') {
            if (st.empty()) return false;
            char t = st.top();
            st.pop();
            if (c == ')' && t != '(') return false;
            if (c == ']' && t != '[') return false;
            if (c == '}' && t != '{') return false;
        }
    }
    return st.empty();
}
```

## 9.43 T609 简单表达式求值

### 9.43.1 只含加减和非负整数

```cpp
long long evalPlusMinus(const string& s) {
    long long ans = 0;
    long long num = 0;
    int sign = 1;

    for (int i = 0; i <= (int)s.size(); i++) {
        if (i < (int)s.size() && isdigit(s[i])) {
            num = num * 10 + (s[i] - '0');
        } else {
            ans += sign * num;
            num = 0;
            if (i < (int)s.size()) {
                if (s[i] == '+') sign = 1;
                else if (s[i] == '-') sign = -1;
            }
        }
```

```
    }
    return ans;
}
```

### 9.43.2 易错点

- 上面模板不处理乘除和括号。
- 如果有空格，先跳过空格。
- 如果有负数开头，检查是否符合模板逻辑。

# 第十章　易错点总表

## 10.1　本章使用方法

本章用于考前最后复习和提交前检查。很多失分不是因为算法不会，而是因为输入输出、下标、溢出、初始化、比较器等细节错误。

建议使用方式：

- 考前每天快速看一遍。
- 每次模拟赛后，把自己犯过的错补进本章。
- 考场提交前，用最后的总检查表快速扫一遍。

## 10.2　输入输出易错点

| 问题 | 表现 | 检查方法 |
| --- | --- | --- |
| 多输出提示语 | 答案格式错误 | 不要输出 `answer=`、提示文字。 |
| 少换行或多内容 | 样例看似接近但不完全一致 | 严格对照题目输出格式。 |
| 大小写错误 | `YES` 写成 `Yes` | 看题目要求。 |
| 小数位错误 | 精度或格式错误 | 用 `fixed << setprecision(k)`。 |
| `cin` 后直接 `getline` | 读到空行 | 加 `cin.ignore()`。 |
| 多组数据未清空 | 第一组对，后面错 | 每组重置数组、图、map、队列。 |

## 10.3　数据类型和溢出

| 场景 | 风险 | 正确做法 |
| --- | --- | --- |
| 求和 | `int` 溢出 | 答案用 `long long`。 |
| 两个大整数相乘 | 先按 `int` 溢出 | 写 `1LL * a * b`。 |
| 最短路距离 | 路径和很大 | `dist` 用 `long long`。 |
| 计数方案数 | 方案数巨大 | 看是否取模，用 `long long`。 |
| 位运算左移 | `1 << k` 溢出 | 大位数用 `1LL << k`。 |
| INF 太小 | 最短路被错误更新 | `long long INF = 4e18`。 |

## 10.4 整除、取整和取模

| 场景 | 常见错误 | 正确处理 |
| --- | --- | --- |
| 正整数分组数 | 少算最后不足一组 | $a \geq 0, b > 0$ 时用 `(a+b-1)/b`。 |
| 负数参与除法 | 误以为 `/` 是向下取整 | C++ 整数除法向 0 截断，必要时用 `floor_div/ceil_div`。 |
| 负数参与取模 | 余数下标为负导致越界 | 用 `r=x%m; if (r<0) r+=m;`。 |
| 大整数取整 | 用 `double` 后丢精度 | 尽量用整数公式；接近 $10^{18}$ 时避免浮点取整。 |
| 除数或模数 | 没判断 0 | 除法和取模前确认分母不为 0。 |

## 10.5 下标和边界

- 题目编号从 1 开始，代码数组是否开了 `n+1`。
- 0 下标数组的第 $k$ 个元素是 `a[k-1]`。
- 区间 $[l, r]$ 是否包含两端。
- 前缀和 1 下标公式是 `pre[r] - pre[l-1]`。
- 访问 `r+1` 时数组是否开到 `n+2`。
- 空数组不能访问 `a[0]` 或 `a.back()`。
- 循环是 `i < n` 还是 `i <= n`。

## 10.6 初始化和清空

| 对象 | 常见错误 | 正确处理 |
| --- | --- | --- |
| 全局数组 | 以为每组自动清空 | 多组数据手动清空。 |
| 局部数组 | 未初始化 | 定义时初始化或使用 `vector`。 |
| `vector` | 上组残留 | `assign` 或重新定义。 |
| `map/set` | 上组残留 | `clear()`。 |
| `queue` | 没有 `clear` | 每组内部重新定义。 |
| `priority_queue` | 没有 `clear` | 每组内部重新定义。 |
| 答案变量 | 没重置 | 每组开始设为 0 或初始值。 |

## 10.7 排序和比较器

- 比较器表示“谁排前面”。
- 相等时不能返回 `true`。

- 多关键字排序要逐项判断。
- 升序和降序不要写反。
- 需要保持原输入顺序时用 `stable_sort`。
- `priority_queue` 的比较器和 `sort` 直觉相反，必须手测。

错误示例：

```
return a.score >= b.score; // wrong
```

正确示例：

```
if (a.score != b.score) return a.score > b.score;
return a.id < b.id;
```

## 10.8 STL 容器易错点

| 容器/函数 | 易错点 | 提醒 |
|---|---|---|
| `vector` | 越界访问 | 确认大小和下标。 |
| `map[key]` | 不存在时会创建 key | 只判断存在用 `count/find`。 |
| `unordered_map` | 遍历无序 | 要有序输出用 `map`。 |
| `set.lower_bound` | 返回 `end` 后解引用 | 先判断 `it != end`。 |
| `queue.front` | 空队列访问 | 先判断 `empty()`。 |
| `stack.top` | 空栈访问 | 先判断 `empty()`。 |
| `priority_queue` | 不能删除中间元素 | 用懒删除或 set。 |
| `erase` | 迭代器失效 | 删除时小心循环写法。 |

## 10.9 字符串易错点

- `cin >> s` 不能读取空格。
- `getline` 可能读到上一行剩余换行。
- `s.find(t)` 找不到返回 `string::npos`。
- 不要写 `if (s.find(t))`。
- `substr(pos, len)` 第二个参数是长度。
- 字符串数字可能超过 `long long`。
- 连续分隔符、开头分隔符、结尾分隔符要测试。

## 10.10 图论易错点

- 无向图要双向加边，有向图不要。
- 点编号从 0 还是 1 开始。

- BFS 应在入队时标记访问。
- Dijkstra 不能处理负权边。
- Dijkstra 要跳过过期状态。
- Floyd 三层循环顺序是 `k,i,j`。
- 并查集初始化每个点的父亲。
- 拓扑排序边方向不要反。

## 10.11 动态规划易错点

- 先定义 `dp[i]` 的含义，再写转移。
- 初始状态不要漏。
- 01 背包容量倒序。
- 完全背包容量正序。
- 求最大值时初始值是否应该是负无穷。
- 方案数是否需要取模。
- 多组数据是否清空 DP 数组。

## 10.12 数据结构易错点

- 树状数组通常用 1 下标。
- 线段树递归区间边界要统一。
- 懒标记下传不要漏。
- 单调栈比较符号决定是否允许相等。
- 单调队列先删除过期元素。
- LCA 的 `LOG` 要足够大。
- 递归 DFS 在大数据上可能爆栈。

## 10.13 提交前总检查

1. 删除所有调试输出。
2. 确认没有输出提示语。
3. 确认输入格式和题目一致。
4. 确认多组数据清空状态。
5. 确认答案和中间乘法使用 `long long`。

6. 确认数组大小足够。
7. 确认下标从 0 还是 1 开始。
8. 确认排序比较器正确。
9. 确认特殊边界测试过。
10. 确认提交的是当前最新代码。

## 10.14 最后十分钟清单

如果考试快结束：

- 不要重构。
- 不要大改已经能过样例的代码。
- 删除调试输出。
- 提交当前最稳版本。
- 如果还有时间，再做小修小补。
- 对没把握的题，至少提交能过样例或小数据的版本。

# 第四部分

# 真题校准

# 第十一章　历年真题索引与模板映射

## 11.1　本章使用方法

本章用于把历年 CCF CSP 真题和前面章节的决策树、模板连接起来。它不是题解合集，而是考场手册的“反向索引”：看到某类题面时，能快速想到应该翻哪一章、套哪个模板、检查哪些坑。

每道题尽量记录：

- 年份月份和题号。
- 题名。
- 表面题型。
- 核心考点。
- 应翻章节或模板编号。
- 易错点或第一判断信号。
- 当前核对状态。

**重要：**题名索引用来训练和复盘，考场判断不能依赖“我见过这个题名”。真正可靠的是题面信号、数据范围、输入输出结构和操作类型。

## 11.2　索引可信度

| 状态 | 含义 | 使用方式 |
| --- | --- | --- |
| 已核对 | 题名和题号来自官方 OJ 或多个公开索引一致。 | 可作为打印版索引使用。 |
| 资料交叉核对 | 题名、题号、考点来自公开题解索引或多份资料，尚未逐题打开官方题面复核。 | 可用于判断树训练，打印前最好再核对数据范围。 |
| 待官方复核 | 题名或考点来自零散题解，可能存在简称、错别字或考点概括不准。 | 只当待补任务，不当最终结论。 |

**打印前原则：**真题索引中“题名”要尽量准；但判断树必须只依赖题面特征。这样即使遇到新题，也能按相同流程落到模板。

## 11.3 索引字段说明

| 字段 | 含义 |
| --- | --- |
| 编号 | 例如 `202312-1`。 |
| 题名 | 真题题目名称。 |
| 题型 | 从手册决策树角度归类。 |
| 核心考点 | 真正需要用到的算法或实现技巧。 |
| 翻到哪里 | 对应章节或模板。 |
| 状态 | 已核对、资料交叉核对、待官方复核。 |

## 11.4 近年整场校准索引

这一张表用来反向校准整本手册：A/B 看稳分算法，C 看大模拟和解析，D/E 看部分分和高级模板入口。

| 场次 | A/B 题 | C 题 | D/E 题信号 | 判断树重点 |
| --- | --- | --- | --- | --- |
| 202406 | 矩阵重塑（一/二） | 文本分词 | 货物调度、哥德尔机 | 矩阵下标、文本 token、优先队列、贪心/背包。 |
| 202403 | 词频统计、相似度计算 | 化学方程式配平 | 十滴水、文件夹合并 | 计数、集合交并、表达式/方程、set/堆、DFS 序/线段树。 |
| 202312 | 仓库规划、因子化简 | 树上搜索 | 宝藏、彩色路径 | 暴力枚举、质因数分解、树 DFS、前缀/分块、状压 DP。 |
| 202309 | 坐标变换（一/二） | 梯度求解 | 阴阳龙、阻击 | 坐标前缀合成、后缀表达式、set、树形 DP。 |
| 202305 | 重复局面、矩阵运算 | 解压缩 | 电力网络、闪耀巡航 | 字符串哈希、矩阵乘法顺序、位运算解析、最短路/状压。 |
| 202303 | 田地丈量、垦田计划 | LDAP | 星际网络 II、施肥 | 矩形相交、二分答案、递归表达式、线段树/离散化。 |
| 202212 | 现值计算、训练计划 | JPEG 解码 | 聚集方差、星际网络 | 公式、拓扑顺序、矩阵扫描、启发式合并、线段树建图。 |
| 202209 | 如此编码、何以包邮 | 防疫大数据 | 通信/调度类复杂题 | 进制分解、01 背包、时间窗口、集合维护。 |
| 202206 | 归一化处理、寻宝大冒险 | 角色授权 | 高级模拟/几何题 | 小数输出、坐标哈希、权限集合、规则匹配。 |
| 202203 | 未初始化警告、出行计划 | 计算资源调度器 | 图/数据结构题 | set 判断、差分区间、对象筛选、优先级排序。 |
| 202112 | 序列查询、新解 | 登机牌号码 | 磁盘文件操作、极差路径 | 分段贡献、数学优化、编码解析、线段树。 |
| 202109 | 数组推导、非零段划分 | 脉冲神经网络 | 收集卡牌、箱根山岳险天下 | 前缀最大反推、贡献扫描、大模拟、概率/图。 |
| 202104 | 灰度直方图、邻域均值 | DHCP 服务器 | 校门外的树、疫苗运输 | 计数数组、二维前缀、大模拟状态机、DP。 |
| 202012 | 安全指数、最佳阈值 | 带配额的文件系统 | 食材运输、星际旅行 | 公式、排序阈值扫描、目录树/配额、大模拟到树/图。 |

| | | | | |
|---|---|---|---|---|
| 202009 | 检测点查询、风险人群筛查 | 点亮数字人生 | 星际旅行、密信与计数 | 距离排序、连续区间判断、拓扑/逻辑门、图与计数。 |
| 202006 | 线性分类器、稀疏向量 | Markdown 渲染器 | 1246、乔乔和牛牛逛超市 | 坐标代入、双指针、文本渲染、DP/高级优化。 |
| 201912 | 报数、回收站选址 | 化学方程式 | 区块链、魔数 | 简单模拟、坐标哈希、表达式解析、图/事件模拟。 |

**用法：**做完一套真题后，把自己第一眼误判的题写在本表旁边。例如“看到最小时间没有想到二分答案”，就回第 3 章补二分判断信号。

## 11.5 A 题索引

| 编号 | 题名 | 题型 | 核心考点 | 翻到哪里 | 状态 |
|---|---|---|---|---|---|
| 202406-1 | 矩阵重塑（其一） | 下标 / 矩阵 | 一维二维下标互转 | A 题二维数组 | 资料交叉核对 |
| 202403-1 | 词频统计 | 统计 | 循环计数、频率 | A 题频率统计 | 资料交叉核对 |
| 202312-1 | 仓库规划 | 枚举 / 判断 | 双重循环、支配关系 | A 题简单模拟 | 资料交叉核对 |
| 202309-1 | 坐标变换（其一） | 坐标模拟 | 坐标平移 | A 题坐标入门 | 资料交叉核对 |
| 202305-1 | 重复局面 | 哈希统计 | 字符串 / 局面计数 | A 题频率统计、字符串 | 资料交叉核对 |
| 202303-1 | 田地丈量 | 坐标 / 几何 | 矩形相交面积 | A 题坐标距离、公式 | 资料交叉核对 |
| 202212-1 | 现值计算 | 公式 / 小数 | 幂、浮点计算 | A 题公式题 | 资料交叉核对 |
| 202209-1 | 如此编码 | 进制 / 模拟 | 数位分解 | A 题进制处理 | 资料交叉核对 |
| 202206-1 | 归一化处理 | 小数 / 统计 | 平均、方差、小数输出 | A 题小数输出 | 资料交叉核对 |
| 202203-1 | 未初始化警告 | 模拟 / set | 变量出现判断 | A 题存在判断、set | 资料交叉核对 |
| 202112-1 | 序列查询 | 模拟 / 查询 | 下标、区间含义 | A 题区间入门 | 资料交叉核对 |
| 202109-1 | 数组推导 | 模拟 / 统计 | 最小最大和 | A 题简单统计 | 资料交叉核对 |
| 202104-1 | 灰度直方图 | 统计 | 计数数组 | A 题计数数组 | 资料交叉核对 |
| 202012-1 | 期末预测之安全指数 | 统计 / 公式 | 加权求和、取 max | A 题简单统计 | 资料交叉核对 |
| 202009-1 | 称检测点查询 | 排序 / 距离 | 距离计算、排序 | A 题排序、坐标距离 | 资料交叉核对 |
| 202006-1 | 线性分类器 | 模拟 / 几何判断 | 代入直线、分类统计 | A 题公式题、坐标 | 资料交叉核对 |

| | | | | | |
|---|---|---|---|---|---|
| 201912-1 | 报数 | 模拟 | 跳过含 7 或 7 倍数 | A 题简单模拟 | 资料交叉核对 |
| 201909-1 | 小明种苹果 | 统计 | 总量、最大减少量 | A 题简单统计 | 资料交叉核对 |
| 201903-1 | 小中大 | 排序 / 输出格式 | 中位数、整数/小数输出 | A 题排序、格式 | 资料交叉核对 |
| 201812-1 | 小明上学 | 时间模拟 | 红绿灯时间累计 | A 题时间处理 | 资料交叉核对 |
| 201809-1 | 卖菜 | 数组模拟 | 邻域平均 | A 题数组边界 | 资料交叉核对 |
| 201803-1 | 跳一跳 | 状态模拟 | 连续奖励计分 | A 题状态扫描 | 资料交叉核对 |
| 201712-1 | 最小差值 | 排序 | 相邻差最小 | A 题排序 | 资料交叉核对 |
| 201709-1 | 打酱油 | 数学 / 贪心 | 优惠规则分解 | A 题公式题 | 资料交叉核对 |
| 201703-1 | 分蛋糕 | 贪心模拟 | 累加达到阈值分组 | A 题简单模拟 | 资料交叉核对 |
| 201612-1 | 中间数 | 排序 / 统计 | 比它大和小数量 | A 题排序/计数 | 资料交叉核对 |
| 201609-1 | 最大波动 | 扫描 | 相邻差绝对值最大 | A 题简单统计 | 资料交叉核对 |
| 201604-1 | 折点计数 | 扫描 | 左右邻居比较 | A 题简单扫描 | 资料交叉核对 |
| 201512-1 | 数位之和 | 数位 | 十进制拆位 | A 题数字处理 | 资料交叉核对 |
| 201509-1 | 数列分段 | 扫描 | 相邻变化计数 | A 题简单扫描 | 资料交叉核对 |
| 201503-1 | 图像旋转 | 矩阵 | 下标变换输出 | A 题二维表格 | 资料交叉核对 |
| 201412-1 | 门禁系统 | 统计 | 出现次数累计 | A 题频率统计 | 资料交叉核对 |
| 201409-1 | 相邻数对 | 排序 / 计数 | 排序后相邻比较 | A 题排序 | 资料交叉核对 |
| 201403-1 | 相反数 | 统计 / 查找 | 取相反数、集合判断 | A 题存在判断 | 资料交叉核对 |
| 201312-1 | 出现次数最多的数 | 统计 | 计数数组 / map | A 题频率统计 | 资料交叉核对 |

## 11.6 B 题索引

| 编号 | 题名 | 题型 | 核心考点 | 翻到哪里 | 状态 |
|---|---|---|---|---|---|
| 202406-2 | 矩阵重塑(其二) | 矩阵 / 模拟 | 下标映射、批量输出 | B 题矩阵、复杂度 | 资料交叉核对 |
| 202403-2 | 相似度计算 | 集合 | 交集、并集、去重 | B 题哈希统计 | 资料交叉核对 |

| | | | | | |
|---|---|---|---|---|---|
| 202312-2 | 因子化简 | 数学 / 因数分解 | 质因数分解、幂 | B 题基础数学 | 资料交叉核对 |
| 202309-2 | 坐标变换(其二) | 前缀 / 坐标变换 | 操作前缀合成 | B 题前缀预处理 | 资料交叉核对 |
| 202305-2 | 矩阵运算 | 矩阵 / 模拟优化 | 计算顺序、复杂度 | B 题表格矩阵、复杂度 | 资料交叉核对 |
| 202303-2 | 垦田计划 | 二分答案 / 贪心 | 最小时间、check | B 题二分答案 | 资料交叉核对 |
| 202212-2 | 训练计划 | 拓扑 / 模拟 | 最早最晚时间 | B 题拓扑排序入口 | 资料交叉核对 |
| 202209-2 | 何以包邮? | DP / 背包 | 01 背包 | B 题简单 DP | 资料交叉核对 |
| 202206-2 | 寻宝! 大冒险! | 坐标 / 哈希 | 点集匹配 | B 题哈希统计、坐标 | 资料交叉核对 |
| 202203-2 | 出行计划 | 差分 / 区间统计 | 时间区间覆盖 | B 题差分 | 资料交叉核对 |
| 202112-2 | 序列查询新解 | 前缀 / 数学 | 分段贡献 | B 题前缀和、排序扫描 | 资料交叉核对 |
| 202109-2 | 非零段划分 | 差分 / 扫描 | 贡献、扫描统计 | B 题差分、排序扫描 | 资料交叉核对 |
| 202104-2 | 邻域均值 | 二维前缀和 | 子矩阵平均 | B 题二维前缀和 | 资料交叉核对 |
| 202012-2 | 期末预测之最佳阈值 | 排序 / 前缀统计 | 阈值扫描、计数 | B 题排序加扫描 | 资料交叉核对 |
| 202009-2 | 风险人群筛查 | 模拟 / 坐标 | 连续停留判断 | B 题排序扫描、模拟 | 资料交叉核对 |
| 202006-2 | 稀疏向量 | 双指针 / map | 有序向量点积 | B 题双指针、哈希统计 | 资料交叉核对 |
| 201912-2 | 回收站选址 | 坐标 / 哈希 | 邻居存在、评分统计 | B 题哈希坐标 | 资料交叉核对 |
| 201909-2 | 小明种苹果(续) | 统计 / 标记 | 掉落判断、连续组 | B 题统计扫描 | 资料交叉核对 |
| 201903-2 | 二十四点 | 表达式 / 栈 | 运算优先级、求值 | C 题表达式入门 | 资料交叉核对 |
| 201812-2 | 小明放学 | 时间 / 模拟 | 周期取模、信号灯 | B 题时间处理 | 资料交叉核对 |
| 201809-2 | 买菜 | 区间重叠 | 时间段交集累计 | B 题区间扫描 | 资料交叉核对 |
| 201803-2 | 碰撞的小球 | 模拟 | 位置更新、碰撞反向 | B 题模拟 | 资料交叉核对 |
| 201712-2 | 游戏 | 队列模拟 | 报数淘汰 | B 题队列 | 资料交叉核对 |
| 201709-2 | 公共钥匙盒 | 事件排序 | 还钥匙优先、时间排序 | C 题事件模拟 | 资料交叉核对 |
| 201703-2 | 学生排队 | 序列模拟 | 移动元素 | B 题 vector 操作 | 资料交叉核对 |

| | | | | | |
|---|---|---|---|---|---|
| 201612-2 | 工资计算 | 反推 / 枚举 | 税率分段、枚举答案 | B 题分段函数 | 资料交叉核对 |
| 201609-2 | 火车购票 | 模拟 / 贪心 | 连续座位优先 | B 题模拟 | 资料交叉核对 |
| 201604-2 | 俄罗斯方块 | 矩阵模拟 | 碰撞检测、落点 | B 题矩阵模拟 | 资料交叉核对 |
| 201512-2 | 消除类游戏 | 二维扫描 | 标记后统一清除 | B 题矩阵模拟 | 资料交叉核对 |
| 201509-2 | 日期计算 | 日期 | 月份天数、闰年 | A/B 日期模板 | 资料交叉核对 |
| 201503-2 | 数字排序 | 排序 / 频率 | 频次降序、值升序 | B 题排序比较器 | 资料交叉核对 |
| 201412-2 | Z 字形扫描 | 矩阵遍历 | 对角线方向切换 | B 题矩阵扫描 | 资料交叉核对 |
| 201409-2 | 画图 | 二维标记 | 矩形覆盖面积 | B 题二维表格 | 资料交叉核对 |
| 201403-2 | 窗口 | 模拟 / 栈序 | 点击命中、置顶 | B 题模拟、列表 | 资料交叉核对 |
| 201312-2 | ISBN 号码 | 字符串 / 校验 | 加权求和、校验码 | A 题字符串、公式 | 资料交叉核对 |

## 11.7　C 题索引：大模拟与中档算法入口

C 题最需要索引，因为题面通常很长，且同一道题可能既像模拟又暗含数据结构。下面表格优先记录“考场第一判断”。

| 编号 | 题名 | 第一识别信号 | 模板入口 | 易错提醒 |
|---|---|---|---|---|
| 202406-3 | 文本分词 | 词表、token、合并规则 | T601、T604、字符串解析 | 输入文本和词表顺序不要混。 |
| 202403-3 | 化学方程式配平 | 方程、元素、系数未知 | 表达式解析、线性方程 | 元素名大小写和括号倍数。 |
| 202312-3 | 树上搜索 | 树、子树、搜索顺序 | DFS、树模拟 | 子树权值和、删除/保留状态。 |
| 202309-3 | 梯度求解 | 表达式、变量、后缀计算 | T608、T609、栈 | 操作数顺序、取模和导数规则。 |
| 202305-3 | 解压缩 | 编码串、块、长度 | 字符串解析、位运算 | 十六进制、长度字段、边界。 |
| 202303-3 | LDAP | 递归条件表达式 | 递归下降、集合/bitset | 与/或/非优先级和括号。 |
| 202212-3 | JPEG 解码 | 8x8 矩阵、Z 字形、变换 | 矩阵模拟、浮点/取整 | 下标顺序、四舍五入、截断范围。 |
| 202209-3 | 防疫大数据 | 时间窗口、风险地区、人员轨迹 | T601、set/map | 生效时间和过期时间。 |
| 202206-3 | 角色授权 | 用户、角色、权限、资源 | map/set、规则匹配 | 通配符、角色继承、权限判断顺序。 |
| 202203-3 | 计算资源调度器 | 节点选择、多级约束 | T601、排序、set | 优先级和约束过滤顺序。 |

| | | | | |
|---|---|---|---|---|
| 202112-3 | 登机牌号码 | 编码、校验、字符串转换 | 进制/编码、模拟 | 字符集映射和补位。 |
| 202109-3 | 脉冲神经网络 | 多轮状态更新 | 大模拟、滚动数组 | 同一轮更新要先累计后生效。 |
| 202104-3 | DHCP 服务器 | 请求/响应、租约状态 | T603 状态机、事件时间 | 到期时间和状态转移顺序。 |
| 202012-3 | 带配额的文件系统 | 目录树、文件大小、配额 | T606、树、map | 路径创建失败要回滚。 |
| 202009-3 | 点亮数字人生 | 逻辑门、依赖顺序 | T205 拓扑排序 | 环检测、多个输入输出。 |
| 202006-3 | Markdown 渲染器 | 文本块、列表、段落 | T601、字符串解析 | 空行分块和宽度换行。 |
| 201912-3 | 化学方程式 | 化学式、元素守恒 | 字符串解析、map | 多位系数、括号嵌套。 |
| 201909-3 | 字符画 | 像素块、RGB、转义输出 | 矩阵模拟、字符串输出 | 转义序列和连续颜色压缩。 |
| 201903-3 | 损坏的 RAID5 | 磁盘、条带、异或恢复 | 模拟、位运算 | 十六进制转换和缺盘数量。 |
| 201812-3 | CIDR 合并 | IP/前缀、排序、合并 | T513、排序扫描 | IP 转整数、包含关系。 |
| 201809-3 | 元素选择器 | CSS 选择器、层级匹配 | 字符串解析、树/栈 | 大小写、祖先关系。 |
| 201803-3 | URL 映射 | 路由规则、参数提取 | T607、字符串匹配 | 参数类型、匹配失败。 |
| 201712-3 | Crontab | 时间范围、周期触发 | T514、事件枚举 | 星期、月份、闭区间。 |
| 201709-3 | JSON 查询 | 键值、嵌套对象、查询 | 字符串解析、栈/map | 转义字符和层级路径。 |
| 201703-3 | Markdown | 标题、列表、强调 | 字符串解析 | 块级规则优先于行内规则。 |
| 201612-3 | 权限查询 | 用户/角色/权限等级 | map/set、规则判断 | 通配等级和最大权限。 |
| 201609-3 | 炉石传说 | 角色、随从、攻击 | T603 状态机 | 死亡结算顺序。 |
| 201604-3 | 路径解析 | 当前目录、相对路径 | T606 路径规范化 | 多个斜杠、.. 到根目录。 |
| 201509-3 | 模板生成系统 | 变量替换、文本模板 | T501 split、map | 引号内容、未定义变量为空。 |
| 201403-3 | 命令行选项 | 选项、参数、命令行 | T602 命令分发 | 有参/无参选项区别。 |

## 11.8 D/E 题索引：只为部分分服务

对新手来说，D/E 索引不要求背满分算法。它的价值是告诉你：如果看到类似信号，至少能写哪个暴力或小数据版本。

| 编号 | 题名 | 满分方向信号 | 新手部分分入口 |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 202403-4 | 十滴水 | set/map、优先队列、连锁反应 | 小数据暴力模拟每次爆裂。 |
| 202403-5 | 文件夹合并 | DFS 序、链表、线段树 | 只做小树 DFS 合并，或无修改子任务。 |
| 202312-4 | 宝藏 | 分块、前缀统计 | 暴力枚举查询，小范围预处理。 |
| 202312-5 | 彩色路径 | 状压 DP、折半搜索 | 颜色数小就状压，否则 DFS 小图。 |
| 202309-4 | 阴阳龙 | 动态 set / 平衡树 | 小数据每次排序或线性扫描。 |
| 202309-5 | 阻击 | 树形 DP、线段树、树剖 | 链/小树暴力 DFS。 |
| 202305-4 | 电力网络 | 图论、最短路/割点信号 | 小图建图后暴力枚举。 |
| 202305-5 | 闪耀巡航 | 最短路、状压 DP | 点数小用 Floyd + 状压。 |
| 202303-4 | 星际网络 II | 线段树、离散化 | 小范围直接数组模拟。 |
| 202303-5 | 施肥 | 分治、线段树/树状数组 | 差分可拿部分分。 |
| 202212-4 | 聚集方差 | 启发式合并 | 小数据暴力维护集合。 |
| 202212-5 | 星际网络 | 线段树建图、高精度 | 小图建边后跑最短路。 |
| 202112-4 | 磁盘文件操作 | 线段树、区间覆盖 | 小范围数组模拟。 |
| 202104-4 | 校门外的树 | DP | 先写 $O(n^2)$ 或按区间暴力。 |
| 202009-4 | 星际旅行 | 图/几何/矩阵类优化 | 小数据按定义模拟。 |
| 201812-4 | 数据中心 | 最小生成树 | T207 Kruskal。 |
| 201412-4 | 最优灌溉 | 最小生成树 | T207 Kruskal。 |

## 11.9 题目反查模板索引

如果你记得“题看起来像什么”，但想不起算法名，按这一表反查。

| 题面样子 | 优先翻看 | 典型真题 |
|---|---|---|
| 一堆对象，每个对象有多个属性，要求按规则选择 | C 题状态机、大模拟骨架、排序比较器 | 计算资源调度器、DHCP 服务器、文本分词 |
| 路径、目录、文件、配额、URL、JSON、Markdown | 字符串解析、路径规范化、递归结构 | 路径解析、带配额的文件系统、JSON 查询、URL 映射 |
| 矩阵、图像、8x8、Z 字形、旋转、重塑 | 二维数组、矩阵扫描、下标变换 | JPEG 解码、矩阵重塑、Z 字形扫描、图像旋转 |
| 多次询问，每次问区间/矩形/时间覆盖 | 前缀和、差分、线段树 | 邻域均值、出行计划、磁盘文件操作 |
| 要求最小时间、最大收益、阈值最优，且答案越大越容易/越难 | 二分答案 | 垦田计划、资源压缩题 |
| 依赖关系、先后顺序、逻辑门、任务计划 | 拓扑排序 | 训练计划、点亮数字人生 |
| 集合合并、连通块、最小连接代价 | 并查集、Kruskal | 数据中心、最优灌溉 |
| 字符串反复变换、替换函数、循环节 | 映射图、环、倍增/取模 | 字符串变换类题 |

## 11.10 按题面信号反查真题

这一节比“题号索引”更接近考场：先看题面出现了什么，再找类似真题和模板。每次刷题后都应把误判的题补到这里。

| 题面信号 | 第一判断 | 类似真题 | 模板入口 |
| --- | --- | --- | --- |
| 只要求统计数量、最大最小、频次 | A 题扫描/计数 | 出现次数最多的数、灰度直方图、词频统计 | A 题统计、T106 |
| 相邻元素、连续段、分段数量 | 一遍扫描或排序后扫描 | 数列分段、折点计数、非零段划分 | A 题扫描、B 题排序扫描 |
| 矩形、坐标点、覆盖面积 | 坐标公式或哈希点集 | 田地丈量、画图、回收站选址、寻宝大冒险 | 坐标、T106、T109 |
| 矩阵重塑、旋转、Z 字形、局部平均 | 二维下标或二维前缀 | 图像旋转、Z 字形扫描、邻域均值、矩阵重塑 | T102、矩阵扫描 |
| 时间段、红绿灯、租约、到期 | 时间模拟或事件排序 | 小明上学、小明放学、公共钥匙盒、DHCP 服务器 | T110、T514、T603 |
| 稀疏数据,两列编号和值 | map 或双指针 | 稀疏向量、坐标变换、回收站选址 | T106、T108 |
| 阈值、最优分界点、排序后评价 | 排序扫描或前缀统计 | 最佳阈值、非零段划分 | T105、T101 |
| 最小时间、最大可行值、资源压缩 | 二分答案 | 垦田计划、资源压缩类题 | T107 |
| 选择若干物品,金额达到目标 | 背包 DP | 何以包邮 | T301 |
| 依赖任务、前置关系、逻辑门 | 拓扑排序 | 训练计划、点亮数字人生 | T205 |
| 路径、目录、文件、配额 | 字符串解析 + 树形结构 | 路径解析、带配额的文件系统、文件夹合并 | T606、T601、T402 |
| JSON、URL、Markdown、模板替换 | 格式解析 | JSON 查询、URL 映射、Markdown、模板生成系统 | T501、T607、T601 |
| 化学式、表达式、括号、方程 | 栈/递归解析 | 化学方程式、化学方程式配平、梯度求解、二十四点 | T608、T609 |
| 缓存、最近最少使用、淘汰 | 链表/map 或队列模拟 | 缓存类模拟题 | T605 |
| 对象有状态,命令改变状态 | 状态机 | DHCP 服务器、炉石传说、角色授权、计算资源调度器 | T603、T602 |
| 每次取最优对象,可能删除失效对象 | 堆或 set, 必要时懒删除 | 文本分词、任务调度类题、十滴水 | T604 |
| 树上子树、祖先、路径、合并 | DFS 序、LCA、树形 DP | 树上搜索、文件夹合并、阻击 | T208、T402 |
| 动态区间修改查询 | 线段树/树状数组 | 磁盘文件操作、星际网络 II、施肥 | T401、T402 |
| 最小连接代价、网络建设 | 最小生成树 | 最优灌溉、数据中心 | T207 |
| 点少、状态多、颜色/集合限制 | 状压 DP | 彩色路径、闪耀巡航 | T301、状压入口 |

## 11.11 真题校准清单

用真题校准判断树时，不要只看自己是否 AC，而要记录“第一眼判断是否正确”。下面这张表可以打印后手写补充。

| 检查项 | 如果答不上来 | 应补到哪里 |
| --- | --- | --- |
| 这题第一眼信号是什么? | 只记得题名，不知道题面特征 | 本章“按题面信号反查真题”。 |
| 数据范围卡掉了哪些暴力? | 没估复杂度就写代码 | 第 3 章数据范围判断树。 |
| 满分模板是什么? | 只会题解，不会迁移 | 第 9 章模板库和第 3 章关键词索引。 |
| 有没有更简单部分分? | D/E 空着没写 | 第 8 章 D/E 部分分策略。 |
| 我误判成了什么? | 例如把二分题当贪心,把表达式当普通字符串 | 第 3 章容易误判分支。 |
| 这题有没有固定易错点? | 样例过了但提交错 | 第 10 章易错点总表。 |

## 11.12 近年题型覆盖缺口

当前手册已经覆盖 A/B/C 大多数常见形态，但仍有一些题型需要后续继续补强。

| 缺口 | 可能出现的位置 | 后续补强方向 |
| --- | --- | --- |
| 复杂表达式解析和方程求解 | C/D | 增加递归下降解析、线性方程消元入门。 |
| 高级树上数据结构 | D/E | 增加 DFS 序 + 线段树、树链剖分识别但不强求实现。 |
| 概率/期望 DP | D/E | 只保留识别和小数据暴力，后续再补模板。 |
| 网络流/匹配 | E | 先写识别信号和放弃策略，避免考场误耗时间。 |
| 复杂几何 | D/E | 先补向量、叉积、排序扫描基础。 |
| 交互式或工程化长模拟 | C | 继续扩充对象状态表、命令分发样例。 |

## 11.13 A/B 题型归纳

### 11.13.1 A 题高频题型

| 题型 | 常见信号 | 对应章节 |
| --- | --- | --- |
| 计数统计 | 出现次数、直方图、满足条件数量 | A 题简单统计、频率统计 |
| 公式计算 | 安全指数、现值、加权求和 | A 题公式题 |
| 坐标处理 | 检测点、坐标变换、矩形 | A 题坐标和距离 |
| 字符串/哈希 | 重复局面、字符串计数 | A 题字符串、频率统计 |
| 二维数组 | 图像旋转、矩阵重塑、卖菜 | A 题数组边界、二维表格 |
| 简单模拟 | 按题意判断、逐项检查 | A 题简单模拟 |

### 11.13.2 B 题高频题型

| 题型 | 常见信号 | 对应章节 |
| --- | --- | --- |
| 前缀和 / 二维前缀和 | 多次区间或矩形查询 | B 题前缀和 |
| 差分 | 时间区间覆盖、区间修改 | B 题差分 |
| 排序扫描 | 阈值、排名、分段统计 | B 题排序加扫描 |
| 哈希 / 集合 | 坐标点集、稀疏数据、相似度 | B 题哈希统计 |
| 二分答案 | 最小化最大时间、资源压缩 | B 题二分答案 |
| 背包 DP | 选择若干使总和达到要求 | B 题简单 DP |
| 拓扑 / 依赖 | 任务计划、前置关系 | B 题拓扑排序 |
| 矩阵模拟 | 重塑、扫描、局部统计 | B 题矩阵处理 |
| 数学分解 | 因子、约数、质因数 | B 题基础数学 |

## 11.14 索引维护流程

每补充一道真题时，按下面流程处理：

1. 从官网模拟系统或可信题库核对题名。
2. 记录数据范围，尤其是 $n$、$m$、$q$、值域和时间限制。
3. 标注题型和核心考点。
4. 指向 A/B/C/D/E 对应章节和模板编号。
5. 写 1 到 3 个易错点。
6. 如果现有章节覆盖不足，回到对应章节补模板。

## 11.15 资料来源记录

本章初版参考了公开题解目录和题目汇总页，用于建立索引框架。正式使用前应回到 CCF CSP 官方模拟系统核对。

- CCF CSP 官网：https://www.cspro.org/
- CCF 认证管理系统公开页面：https://csp.ccf.org.cn/

- CSP Project 公开题目索引：https://ccf-csp-project.github.io/CSP-Project-with-MkDocs/problem/
- 公开题解目录示例：博客园 J_StrawHat 的 CCF-CSP 认证 C++ 题解目录。
- 公开题目汇总示例：何公山等平台的 CCF-CSP 历年题目链接汇总。

# 版本记录

| 版本 | 日期 | 内容 |
| --- | --- | --- |
| v0.1 | 2026-05-27 | 建立 LaTeX 项目骨架，完成手册入口、C++ 语法、A/B/C/D/E 策略、真题索引、模板库、易错点、详细判断树和考场极速索引初版。 |
| v0.2 | 2026-06-04 | 重排全书结构为“导航与判断、基础与题型、模板与检查、真题校准”，统一章节文件编号；补强 C++ 基础、整数取整和标准取模。 |