

比赛赛前注意事项

rainboy

rainboylvx@qq.com · idx.roj.ac.cn

摘要 本文是 2024 版赛前注意事项的 2026 修订版，面向 CSP-J/S 第二轮与 NOIP 的选手。文中删去了全部虚拟机 (VMWare) 相关内容，比赛环境统一按 CCF 官方发布的 **NOI Linux 2.0 (noilinux 2.0)** 改写；内容包括环境速查、赛前必练清单、常用代码模板、对拍与调试命令，以及历年最容易丢分的细节。

关键字 CSP-J/S; NOIP; NOI Linux 2.0; 代码模板; 对拍; 考试细节

本版改动

- 删除原第 1 章全部虚拟机 (VMWare) 内容：不再讲打开虚拟机、传文件的三种方法；
- 比赛环境统一改写为 **NOI Linux 2.0 (noilinux 2.0)**，新增环境速查表、终端与编辑器选择、五个易踩的坑；
- 代码模板全部更新：统一为 C++14 + freopen 写法，补上二进制枚举、next_permutation、链式前向星；
- 新增「赛前必须练熟的五件事」「代码里的常见错误」「时间分配建议」与附录检查清单。

2026 关键信息速览

比赛日期 2026 年 10 月 31 日 (周六)

比赛时间 入门级 08:30 – 12:00 提高级 14:30 – 18:30

比赛系统 NOI Linux 2.0 (noilinux 2.0, Ubuntu 20.04.1 定制, 64 位)

编译器 g++ 9.3.0, 默认 C++14

提交内容 只提交 .cpp 源文件，放在形如 AH-J00001 的考试文件夹中

关键动作 写对 freopen，写对文件名，删掉所有调试输出

目录

1	赛前须知	3
1.1	第二轮的时间与流程	3
1.2	NOI Linux 2.0 环境速查	3
1.3	赛前必须练熟的五件事	4
2	noilinux 2.0 使用要点	4
2.1	登录与桌面	4
2.2	打开终端	4
2.3	选一个顺手的编辑器	4
2.4	五个最容易踩的坑	5
3	简单代码模板	5
3.1	整体模板	5
3.2	文件读写	6
3.3	01 序列 (子集 / 组合枚举)	6
3.4	全排列	7
3.4.1	方案一: DFS 手写 (可轻松改成部分排列)	7
3.4.2	方案二: next_permutation (更快更短)	7
3.5	DFS 通用框架	8
3.6	随机数据生成	8
3.7	对拍脚本	9
3.8	图论: 存图	9
3.9	常用命令	10
3.9.1	编译执行	10
3.9.2	给自己配几个快捷键 (可选)	10
3.9.3	比较与限时	10
4	考试注意的细节	11
4.1	提交前一定再看一遍	11
4.2	代码里的常见错误	11
4.3	数据规模与内存	11
4.4	常数与卡常	11
4.5	时间分配建议	11
4.6	参考资料	12
附录 A	赛前检查清单	12
A.1	知识	12
A.2	操作	12
A.3	考前一天	12

1 赛前须知

1.1 第二轮的时间与流程

CSP-J/S 2026 第二轮认证于 **2026 年 10 月 31 日（星期六）** 在全国各省同步举行：

表 1 CSP-J/S 2026 第二轮认证时间

日期	08:30 – 12:00	14:30 – 18:30
2026-10-31（周六）	CSP-J2（入门级）	CSP-S2（提高级）

表 2 第二轮认证的入场、离场与缴费规则

开考前入场	一般要求提前 40–45 分钟到达考点，凭准考证与有效证件入场
迟到限制	认证开始 30 分钟后不得进入考场，按缺考处理
提前离场	非特殊原因不得提前离场，须等认证正式结束后统一离场
报名缴费	第二轮由省负责人统一报名，选手只需在缴费期内完成缴费，逾期视为放弃
考试时长	每组 3.5 小时，4 道题，通常每道题 100 分

1.2 NOI Linux 2.0 环境速查

自 2021 年 9 月 1 日起，NOI Linux 2.0（基于 Ubuntu 20.04.1 定制的 64 位系统）就是 NOI 系列比赛与 CSP-J/S 的标准环境，2026 年继续沿用。它的软件清单在赛前最好背下来：

表 3 NOI Linux 2.0 环境简表（以 CCF 官方公布为准）

类别	软件 / 模块	版本
系统	Kernel（64 位）	5.4.0-42-generic
语言环境	GCC / G++	9.3.0
	FPC（Pascal）	3.0.4
	Python（非竞赛语言）	2.7 / 3.8
调试工具	GDB / DDD	9.1 / 3.3.12
集成开发环境	Code::Blocks	20.03
	Lazarus / Geany	2.0.6 / 1.36
文本编辑器	VS Code / Sublime Text	1.54.3 / 3.2.2
	Vim / Emacs / Gedit / nano / Joe	8.1 / 26.3 / 3.36.2 / 4.8 / 4.6
其他	Firefox / mc / XTerm	79.0 / 4.8.24 / 3.5.3

注意 g++ 是 **9.3.0**，默认标准为 C++14，完整支持 C++11/14，C++17 大部分可用。**不要在代码里使用 C++20 才有的语法**（如 ranges、format、三路比较运算符），考场上编译不过就是 0 分。

重要变化（相对更老的 NOI Linux）

1. 系统由 32 位变为 **64 位**，指针占 8 字节，sizeof(long) 与指针大小相关，不要想当然。
2. 不再提供 GUIDE，代码要在终端用 g++ 编译，或用 Code::Blocks / VS Code。
3. 桌面目录名是中文「桌面」，路径里带中文，**个别程序会因此报错**，一律在自己的家目录下工作。
4. 文件管理器右键默认**没有**「新建文件」，用 touch 或编辑器新建。

5. 系统默认没有 unrar，压缩包只认 .zip。

1.3 赛前必须练熟的五件事

- (1) 在自己的考试文件夹里用终端新建 xxx.cpp，并用 g++ 成功编译运行；
- (2) 正确写出 freopen，确认程序读的是 .in、写的是 .out；
- (3) 会看编译错误信息，会根据 RE / WA 的现象定位到具体代码行；
- (4) 会用 diff 做对拍，会用 timeout 限制运行时间；
- (5) 会清理文件夹、只提交 .cpp 源文件。

技巧 这五件事任何一件在考场上第一次做，都会消耗你 20 分钟以上的宝贵时间。赛前请在家用 noilinux 2.0 的实机或镜像完整走一遍流程。

2 noilinux 2.0 使用要点

2.1 登录与桌面

开机登录后是 Ubuntu 20.04 的 GNOME 桌面。桌面左上角「活动」可以搜索程序，左侧 Dock 上有文件管理器与终端图标。

路径里不要有中文 系统的「桌面」目录名是中文。把代码和题目数据放在 ~/（即 /home/<用户名>/）下，例如 ~/AH-J00001/，路径全英文，能避免绝大多数玄学问题。

2.2 打开终端

三种方式，任选：

- 快捷键 Ctrl + Alt + T（最稳、最快）；
- 顶部「活动」搜索 Terminal；
- 在文件管理器里进入目标文件夹，右键「在终端中打开」。

打开后先确认自己在哪儿、有哪些文件：

```
pwd          # 当前目录
ls -l        # 列出文件
cd ~/AH-J00001
```

2.3 选一个顺手的编辑器

表 4 noilinux 2.0 里可选的编辑器

编辑器	说明
Code::Blocks	开箱即用的 C/C++ IDE，自带编译按钮，无需插件，最保险
VS Code	界面现代，但需要装 C/C++ 插件才能补全与调试；若考场不能联网，就只当高级记事本用
Vim / nano	纯终端操作，速度最快，前提是你已经用熟了
Geany	轻量级 IDE，够用

技巧 比赛只看最后提交的 .cpp，与用什么编辑器无关。赛前就用你打算在考场用的那一个，不要在考场上第一次打开 VS Code。

2.4 五个最容易踩的坑

- (1) 右键没有「新建文件」：touch a.cpp，或者直接在编辑器里「另存为」。
- (2) 没有 unrar：考场数据包若是 .rar，解压不了。请老师提供 .zip，或现场协助解压。
- (3) 中文路径 / 中文文件名：一律用英文加下划线，绝不出现空格。
- (4) 编译通过但运行没输出：多半是忘了 freopen 或者忘了把注释去掉，也可能是读错文件名。
- (5) 自己机器快、评测机慢：本地测出来的时间要乘以 2 再和时限比。

3 简单代码模板

以下模板都是「先骗分、再优化」的通用骨架。考场上时间紧张时，抄骨架、填逻辑，比从零开始写要快得多，也稳得多。

3.1 整体模板

```
// 极简通用模板
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e6 + 5;
int n, m;
int a[maxn];

void init() {
    cin >> n;
    for (int i = 1; i ≤ n; i++) cin >> a[i];
}

void solve() {
    // 主体逻辑写在这里
}

int main() {
    // 【提交前必做】把下面两行注释去掉，并把 xxx 换成题面给定的文件名
    // freopen("xxx.in", "r", stdin);
    // freopen("xxx.out", "w", stdout);

    ios::sync_with_stdio(false);
    cin.tie(nullptr);

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

关于输入输出 用了 freopen 之后，**不要**再用 printf / puts 之外的方式往屏幕上输出调试信息，也不要再 cin 交互。调试信息请全部删掉再提交——往 .out 里输出的任何多余字符都是 WA。

3.2 文件读写

```
// 方式一：C 风格，最常用，也最不容易出错
freopen("fruit.in", "r", stdin);
freopen("fruit.out", "w", stdout);
```

要点：

- 文件名 **逐字** 照抄题面，注意大小写、下划线；（如 meeting.in 不是 Meeting.in）
- 只用相对路径，**不要写** /home/noi/AH-J00001/fruit.in 这种绝对路径；
- 程序要在考试文件夹内运行，./fruit 而不是 ~/AH-J00001/fruit；
- 不要对 stdin / stdout 做 fclose，return 0 时会自动刷新。

3.3 01 序列（子集 / 组合枚举）

凡是满足下列条件之一的问题，都可以用这段代码拿到 10 到 20 分：

- 数据范围小 ($n \leq 10 \sim 20$)；
- 每个元素「选」或「不选」；
- 求所有子集 / 组合的最优值。

```
// 枚举 n 个元素的全部  $2^n$  个子集
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e6 + 5;
int n;
int b[maxn]; // b[i] = 0/1, 表示第 i 个元素选不选

void dfs(int dep) {
    if (dep > n) {
        for (int i = 1; i <= n; i++) cout << b[i] << " ";
        cout << "\n";
        return;
    }
    for (int i = 0; i <= 1; i++) { // i: 0 → 1
        b[dep] = i;
        dfs(dep + 1);
    }
}

int main() {
    cin >> n;
    dfs(1);
    return 0;
}
```

如果 $n \leq 20$ ，用二进制枚举更快更短：

```
for (int S = 0; S < (1 << n); S++) {
    for (int i = 0; i < n; i++)
        if (S >> i & 1) {
            // 第 i 个元素被选中
        }
}
```

注意 二进制枚举时 $1 \ll n$ 在 $n \geq 31$ 时会溢出 int，此时必须写 $1LL \ll n$ ，且 S 要用 long long。

3.4 全排列

任何「枚举所有排列就能出答案」的题，都能用下面两种写法之一骗到分。

3.4.1 方案一：DFS 手写（可轻松改成部分排列）

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

const int maxn = 1e5 + 5;
int n;
int rcd[maxn]; // rcd[dep] 表示第 dep 个位置放哪个数
int vis[maxn]; // vis[i] 表示数 i 是否已被使用

void full_permutation(int dep) {
    if (dep > n) {
        for (int i = 1; i ≤ n; i++) cout << rcd[i] << " ";
        cout << "\n";
        return;
    }
    for (int i = 1; i ≤ n; i++) {
        if (vis[i]) continue; // 用过了就跳过
        rcd[dep] = i;
        vis[i] = 1; // 标记占用
        full_permutation(dep + 1); // 下一个位置
        vis[i] = 0; // 恢复现场
    }
}

int main() {
    n = 3;
    full_permutation(1);
    return 0;
}
```

3.4.2 方案二：next_permutation（更快更短）

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

int a[100];

int main() {
    int n = 5;
    for (int i = 1; i ≤ n; i++) a[i] = i; // 必须从字典序最小开始

    do {
        for (int i = 1; i ≤ n; i++) cout << a[i] << " ";
        cout << "\n";
    } while (next_permutation(a + 1, a + 1 + n));

    return 0;
}
```

想对字符串做全排列，就排下标、再按下标取字符：

```
char str[100] = "0hello"; // 下标 0 处占位, 方便 1 基
int a[100];
int n = 5;
for (int i = 1; i ≤ n; i++) a[i] = i;
do {
    for (int i = 1; i ≤ n; i++) cout << str[a[i]];
    cout << "\n";
} while (next_permutation(a + 1, a + 1 + n));
```

3.5 DFS 通用框架

搜索是最后 30 分钟的保命手段, 框架要能闭着眼写出来。

```
void dfs(int dep, /* 当前状态参数 */) {
    vis[/* 状态 */] = 1; // 标记访问
    if (dep > m) { // 边界
        // check answer
        vis[/* 状态 */] = 0;
        return;
    }
    for (/* 枚举所有转移 */) {
        // do something
        dfs(dep + 1, /* 新状态 */);
        vis[/* 状态 */] = 0; // 恢复现场
    }
}
```

技巧 写搜索时先把 **边界** 和 **恢复现场** 两处写好, 再填中间的循环。90% 的搜索 bug 都出在这两处。

3.6 随机数据生成

用来自己造数据、验证贪心和暴力是否一致。

```
// 随机数据生成
#include <bits/stdc++.h>
using namespace std;

const int maxn = 1e6 + 5;
std::random_device rd;
std::default_random_engine __rnd(rd());
std::mt19937 mtrnd(rd()); // 更高质量的生成器

// 生成 [l, r] 之间的随机整数
int rnd(int l, int r) {
    return __rnd() % (r - l + 1) + l;
}

int main() {
    int n = rnd(4, 7);
    cout << n << "\n";
    for (int i = 1; i ≤ n; i++) cout << rnd(1, 10) << " ";
    cout << "\n";
    return 0;
}
```

注意 题面里的「特殊性质」往往对应某一档部分分。造数据时要**专门造**这些特殊情形，例如「所有 a_i 相等」「图是一条链」「 $n = 1$ 」。

3.7 对拍脚本

对拍 = 同一份输入，让「暴力」和「待验证程序」跑一遍，比较输出。这是赛场上找 bug 最快的手段。

```
#!/bin/bash
# 文件: check.sh
# 1.cpp = 你的程序, 2.cpp = 暴力程序
for i in {1..1000}; do
    printf "\r$i"
    ./gen > in          # 生成数据
    ./1 < in > out1      # 你的程序
    ./2 < in > out2      # 暴力程序
    diff out1 out2 || break
done
```

使用前先编译好三个程序，然后运行脚本：

```
g++ -O2 -o gen gen.cpp      # 数据生成器
g++ -O2 -o 1 1.cpp          # 你的程序
g++ -O2 -o 2 2.cpp          # 暴力程序
chmod +x check.sh
./check.sh
```

注意 对拍比的是「去行末空格 + 去末尾回车之后」的内容。手写的输出格式如果有差异（比如多一个空格），会被判为不同，先确认输出格式再怀疑算法。

3.8 图论：存图

J 组、S 组的前两题，存图直接用邻接矩阵 + 暴力 DFS / BFS 就够骗分了。数据规模大一些，就用链式前向星或 vector 邻接表。

```
const int maxn = 1e5 + 5;

// 1) 邻接矩阵: n ≤ 1000, 边权、重边都无所谓
int g[1005][1005];

// 2) 链式前向星 (邻接表): n, m 达到 1e5 ~ 1e6 时用
struct Edge { int to, w, nxt; } e[1000005];
int head[100005], ecnt;

void add(int u, int v, int w = 1) {
    e[++ecnt] = {v, w, head[u]};
    head[u] = ecnt;
}
```

```
// 3) vector 邻接表: 写起来最短, 把边存成结构体
// 名字用 g2, 避开上面邻接矩阵的 g; maxn 就是上面那个上界
struct edge {
    int v, w;
};

vector<edge> g2[maxn]; // g2[u]: 从 u 出发的全部边

void add_edge(int u, int v, int w) {
    g2[u].push_back({v, w});
    g2[v].push_back({u, w}); // 无向图
}
```

3.9 常用命令

3.9.1 编译执行

```
# 编译 + 立即用 in 文件跑一遍
g++ -g -o 1 1.cpp && ./1 < in

# 与评测机尽量一致: 开 O2、指定 C++14、打开全部警告
g++ -O2 -std=c++14 -Wall -o 1 1.cpp
```

以评测命令为准 评测机用的是它自己的固定编译命令。所以代码必须能在 **不开任何自定义宏、不加任何 额外链接库** 的情况下编译通过 —— 不要依赖你本地加的 `-DDEBUG` 之类的开关。

3.9.2 给自己配几个快捷键 (可选)

打开 `~/.bashrc`, 在末尾追加以下内容, 然后重开终端:

```
function b() { g++ -g -O2 -std=c++14 -o "$1" "$1.cpp"; }
function bb() { g++ -g -O2 -std=c++14 -o "$1" "$1.cpp" && "./$1" < in; }
function bbb() { g++ -g -O2 -std=c++14 -o "$1" "$1.cpp" && "./$1" < "$2"; }
```

用法:

- `b 1` 编译 `1.cpp`;
- `bb 1` 编译并以 `in` 为输入运行;
- `bbb 1 in2` 编译并以 `in2` 为输入运行。

注意 考场机器是重装的, `~/.bashrc` 的改动**不会保留**。这套快捷键只适合在家里练手, 赛前请确认自己用完整命令也不手生。

3.9.3 比较与限时

```
# 最基础的比较
diff out1 out2

# 忽略行末空格
diff -b out1 out2

# 并排显示差异, 改错时非常直观
vimdiff out1 out2

# 限制程序运行时间 (超时会被 kill, 用来验证是否会 TLE)
timeout 1.1s ./1 < in
```

4 考试注意的细节

4.1 提交前一定再看一遍

- (1) 不要忘记 `freopen`，并且文件名与题面完全一致。
- (2) 文件夹名里的短横线：形如 `AH-J00001`，中间那个是 -（减号），不是下划线 `_`。
- (3) 只提交 `.cpp` 源文件。不要提交 `.in`、`.out`、`.o`、可执行文件，也不要提交整个压缩包。
- (4) 每个题目的源文件名按题面要求，通常是英文小写（如 `fruit.cpp`），不要自己改名。
- (5) 代码里保留调试输出？**全部删掉**。写文件之外的一切输出都会被计入答案。
- (6) 交卷前，在你自己的文件夹里重新编译一遍，确认没有编译错误、没有警告级别的隐患。

4.2 代码里的常见错误

表 5 常见错误现象与最可能的原因

现象	最可能的原因
全部 WA / 0 分	文件名写错、忘了 <code>freopen</code> 、输出多一个字符
RE（运行时错误）	数组越界、开得太小、递归爆栈、除以 0、vector 下标超出
TLE（超时）	复杂度过高；或用了 <code>endl</code> 、 <code>cin</code> 未关同步导致常数过大
MLE（超内存）	数组过大，例如 <code>int a[1e8]</code> 就已经 400 MB
本地对、评测错	未定义行为（越界读写、未初始化变量），或依赖了本机才有的东西

4.3 数据规模与内存

- `int` 4 字节，`long long` 8 字节，指针在 64 位系统上 8 字节；
- 常用参照：`int a[1e6]` $\approx$ 4 MB，`int a[1e7]` $\approx$ 40 MB，`int a[1e8]` $\approx$ 400 MB；
- 题目一般给 256 MB 或 512 MB，**全局数组**放在静态区，不要开在 `main` 里；
- 累加、乘积、坐标运算超过 2×10^9 就换 `long long`；
- 用 `printf` 输出 `long long` 时格式是 `%lld`，不是 `%d`。

4.4 常数与卡常

- (1) 关闭 `cin / cout` 的同步：`ios::sync_with_stdio(false); cin.tie(nullptr);`
- (2) 大量输出用 `"\n"`，**不要用** `endl`（每次都刷新缓冲区）。
- (3) 能递推就不要递归，能数组就不要 `map`。
- (4) 本地测出的时间 **乘 2** 再和时限比较，留出安全余量。
- (5) 真的卡常时，先检查有没有在循环里做多余的操作，而不是急着换算法。

4.5 时间分配建议

表 6 3.5 小时的时间分配建议

时段	做什么
前 10 分钟	通读 4 道题，按难度排序，决定开题顺序
第 10 – 60 分钟	把最容易的一题写对、写稳，先拿到第一份分
第 60 – 150 分钟	主攻第二题，同时对其他题写暴力拿部分分
第 150 – 190 分钟	检查所有题的 <code>freopen</code> 、文件名、输出格式；能对拍就对拍
最后 20 分钟	停止写新代码，只做检查与提交；确保每个文件夹里都有能编译的 <code>.cpp</code>

最后 20 分钟不要做的事 不要重构代码、不要改一个「看起来不太对」的算法、不要为了省一秒钟去动已经 AC 的程序。这段时间唯一值得做的事是 **检查与提交**。

4.6 参考资料

- (1) CCF 关于 NOI Linux 2.0 的发布公告：
noi.cn/gynoi/jsgz/2021-07-16/732450.shtml
- (2) CSP-J/S 2026 第二轮报名通知：
noi.cn/xw/2026-09-29/933744.shtml
- (3) 各省第二轮认证者须知（务必看本省的）：
noi.cn/gs/index_3.shtml

附录 A 赛前检查清单

A.1 知识

- ☐ 会写 freopen, 知道 .in / .out 从题面哪里抄
- ☐ 会写搜索骨架, 能快速改成暴力拿部分分
- ☐ 会枚举子集、枚举排列、枚举组合
- ☐ 会写随机数据生成器, 会写对拍脚本
- ☐ 能默写常用的图论存图方式 (邻接矩阵 / 链式前向星)
- ☐ 熟悉 int / long long 溢出边界, 熟悉常见内存占用

A.2 操作

- ☐ 在家里用 noilinux 2.0 完整做过一次模拟赛
- ☐ 会用 Ctrl+Alt+T 开终端, 会用 pwd / ls / cd
- ☐ 会用 g++ -O2 -std=c++14 -Wall 编译, 会看编译错误
- ☐ 会用 diff 对拍, 会用 timeout 测超时
- ☐ 会用你打算在考场用的那个编辑器 (VSCode / Code::Blocks / Vim)
- ☐ 会清理文件夹, 只保留 .cpp

A.3 考前一天

- ☐ 确认准考证、身份证件、考点位置与交通方式
- ☐ 确认本省第二轮认证者须知里的入场时间与禁带物品
- ☐ 早点睡。赛前熬夜复习的收益远小于第二天脑子不清醒的损失

本文基于 2024 版赛前注意事项改写, 删去虚拟机相关内容, 比赛环境按 NOI Linux 2.0 整理。

文中软件版本与时间安排请以 CCF 官方当年公告为准。

祝各位选手 CSP-J/S 2026 一切顺利。