信息学
模拟与搜索

主讲：李宁远

在写代码之前：配环境

众所周知，NOI 系列比赛所用的系统是 NOI Linux 2.0 (这是链接），它是基于 Ubuntu 20.04.1 魔改而来，和 Ubuntu 20 差别不大。所以建议写代码时直接使用该系统，或是使用很类似的 Ubuntu 20+。
我个人用的是 Ubuntu 20，可以从清华 tuna 下载。
我们得到的是.iso 文件，不能直接跑成一个系统。想要使用，需要装到一个计算机里，或者使用虚拟机软件。我推荐第二种。

如何装虚拟机

我是很久之前用的 Vmware 16，然后不知道为啥也能在试用期过了之后继续用。现在这个版本应该下不到了，不过我有安装包 ()
接下来，可以参考我写的教程。
这一步建议大家自行查找资料，相互交流，因为我并不能了解所有问题，所以大概率问我没用 ()

代码编辑器

要写代码，就得有一个写代码的地方。
可以使用 IDE，即写代码和编译、运行代码都在一个软件里，例如 Dev-C++、Code::Blocks 等。
也可以使用文本编辑器，例如 VS Code、Gedit、Sublime Text 等，然后在终端里编译运行代码。
我个人用的是 Gedit，因为超级方便，而且可以简单调整得到好看的配色。不过其在打开大文件时会显示出 bug，所以如果要打开大的输入/输出文件，需要配合其它编辑器查看。

编译运行代码

当然可以使用 IDE 自带的编译运行功能，不过我个人习惯是在终端里编译运行代码。

OI 里用的编译器是 g++。以下是一个最简单的编译命令：g++ a.cpp。运行后，会得到可执行文件 a.out。运行该文件可以使用 ./a.out，然后就是正常的输入输出了。

编译选项

- -o out_file：指定输出文件名。例如 g++ a.cpp -o 1 会将编译后的可执行文件命名为 1，此时运行时需要使用 ./1。
- -O2、-O3、-Ofast。
- -std=c++11、-std=c++14、-std=c++17：调整 C++ 标准。惨痛案例：2024 省选：季风的 abs(int128)。
- -Wall：开启警告，虽然说是 all 但并没有开启所有警告。
- -Wextra：开启额外警告，但这个也不是所有警告。
- -Wconversion：开启类型转换警告，比如 long long 转 int 会有警告，因为可能会溢出。

编译选项

- -fsanitize=undefined,address：开启 undefined behavior 和 address sanitizer。这个选项会让编译器检查代码中可能存在的未定义行为和内存错误，能帮助你发现一些潜在的 bug。但是其在 Windows 上默认不支持，所以建议平常都用 ubuntu ()

终端快捷键

- 方向键上下：查看历史命令，可以上 + enter 直接执行上一条命令。
- Ctrl + C：终止当前运行的程序。
- Ctrl + Shift + C：复制选中的文本。
- Ctrl + Shift + V：粘贴剪贴板中的文本。
- Ctrl + D：停止当前输入、或是退出终端。

终端常用命令

- ls：列出当前目录下的文件和文件夹。
- cd：切换目录。
- mkdir：创建目录。
- rm：删除文件或目录。
- cp：复制文件或目录。
- mv：移动或重命名文件或目录。
- cat：查看文件内容。文件名支持通配符（即，其中的 * 可以匹配任意字符串，会一起输出所有匹配的文件内容）。
- head：查看文件的前几行，默认是前 10 行，也支持通配符。

终端常用命令

- diff a b：比较文件 a 和 b 的差异，输出不同的行。加入选项 -Z 可以忽略行末空格、回车等。
- ulimit：限制该终端内的最大栈空间、总空间、运行时间等。-s number 可以修改栈空间限制。
- time ./a.out：测量 a.out 的运行时间。
- /usr/bin/time -v test：测试程序 test 的具体时间、峰值空间。注意这是不算未使用的静态空间的。
- gedit：快速打开文本编辑器，后面加入文件名即可打开该文件。
- python3：运行 python3，可以当作计算器使用。
- factor a：快速分解 a 的质因数。

终端使用技巧

- 想要连续运行多条命令，可以使用 ; 分隔。不过不建议这样，因为当前面的命令执行失败时（如编译错误），后面的命令仍然会执行。
- 想要在一条命令执行失败时不执行后面的命令，可以使用 && 分隔。多条命令就直接串起来。这个很好用。
- 父目录是 ..，当前目录是 .。所以可以使用 cd .. 回到父目录。
- 可以使用 < 将文件内容作为输入。例如 ./a < 1.in 会将 1.in 的内容作为输入传给 a；可以使用 > 将输出重定向到文件。
- 可以使用 | 将一个命令的输出作为另一个命令的输入。

写代码

以下的内容绝大多数都是我个人的建议。对于写代码，每个人的习惯都不同。不过，如果觉得写代码、调代码很难受、很慢或是根本写不出来，不妨试试接下来的建议。

代码风格

在 OI 里，最重要的是自己能看懂，看的舒服。毕竟是单人竞赛，写成啥样都无所谓，但是为了好看懂一些，我还是建议保持一个自己的缩进风格和命名风格。
对于长的代码，很容易记不清哪个变量是表示什么，从而写和调都无法进行。建议在定义变量时写个注释。变量名长短倒是无所谓，长了不一定能记住，短了也不一定记不住。

代码习惯

先想好代码大致的结构是什么再开写，细节可以边写边想。这提升了我的代码速度，但是也提升了写着写着发现细节挂了的概率（真人真事，幸好不是正赛）。对一些题而言细节可能会相当复杂，此时建议先彻底分析清楚再开始写。可以使用 lambda 函数。其对我的意义是在写代码时，减少上下翻代码的频率，从而让注意力更集中。就比如某个地方需要写一个子过程，那 lambda 可以写在局部，而要写成函数就只能在该函数上方新加一个，来回修改的话就会极大增加上下翻的时间。

调代码

当我们完全了解每个变量的含义后，至少我们有一个保底的调试方法：一步步的输出变量的值，并手动计算所有变量，找出不同的变量并以此找出错误。所以，在调代码前，明白所有变量的含义是极其重要的。

这里 是我写的一些调代码的注意事项和想题的方式。里面所有都是亲身经历.jpg

可能大家都会用 gdb 等调试器，但是我一直没用过，只是输出变量调代码也挺舒服的，所以就不说了 ()

调代码

当代码 RE 时，首选开 -fsanitize=undefined,address。undefined 会写出具体出问题的位置，address 则只会写出函数调用栈和内存位置（没啥用）。

如果找不到哪里挂了，可以在多个地方输出不同的调试信息，看看到哪里就 RE 了。当然这个也可以通调试器过打断点之类的方式实现。

可以善用 assert 来检查变量间应有的关系，或是一个变量应满足的性质。assert 会在条件不满足时直接 RE 并告知是哪个 assert 挂了，能快速定位到问题。

对拍

当找不到错误数据时，如果交上去挂了，或是大数据挂了不好调试，或是想要确认自己的代码是否正确时，可以使用对拍。

对拍其实就是写一个不容易出错的暴力代码，然后再写一个数据生成器，生成数据后对比自己代码和暴力代码的输出是否一致（如果是用 spj 则还需要自己写一个 spj）。不一致就可以得到错误数据。

对拍

组合上面提到的终端命令行写法即可得到快速对拍的脚本。
我个人推荐在数据生成器中写对拍脚本，这样可以保证时间相近时数据生成器也可以得到不同的随机种子，生成不同的数据。数据强度很难保证，但由于可以跑很久对拍，所有直接随机也可以有很好的效果。
对拍时可以把代码的全局数组开小以加快速度。

读代码

看懂变量的含义即可。看不懂的话，尝试跑一跑代码，看看那个量有啥用，就行了。

dfs 和 bfs 的剪枝

dfs 与 bfs 的做法大家应该已经熟知。于是来讲述一些剪枝。

- 可达性剪枝：粗略判断当前状态是否可转移至目标状态，不可行就不干。
- 重复状态剪枝：如果当前状态已被搜索过，或是与当前状态本质相同的状态已被搜索过，就不再进行搜索。

其中第二种剪枝，在“本质相同的状态”个数被砍到可以被计算的时候，其复杂度就有保证了，此时也可以叫它记忆化搜索。

meet in the middle

起点到终点的最短路可以分成两段。我们手动划分状态空间为两半，起点搜完起点的一半，终点搜完终点的一半，然后枚举起点侧，看看它能否连向一个终点侧的状态。可能用各种方式来维护所有路径的信息，可能是求最小值，或是求和之类的。

传统的搜索要搜索所有状态上的所有路径，而 meet in the middle 则只需要搜索一半的状态上的所有路径，然后枚举另一半的状态，于是减小了复杂度。

meet in the middle

例题：P2962. [USACO09NOV] Lights G
题意：有 n 盏灯，每盏灯与若干盏灯相连，每盏灯上都有一个开关，如果按下一盏灯上的开关，这盏灯以及与之相连的所有灯的开关状态都会改变。一开始所有灯都是关着的，你需要将所有灯打开，求最小的按开关次数。

迭代加深搜索

其适用于搜索一个目标状态的最短路。当从一个状态扩展出的状态很多，边的长度又很短，而且几乎没有重复时，迭代加深搜索的效果很好。

由于我们事先并不知道目标状态在 bfs 搜索树里的深度(即最短路)，所以如果我们想要一次性搜到答案的话，就需要保存所有搜到的状态，这样需要很大的空间。想要小空间，就只能 dfs，然而其第一次搜到某个状态时不一定就是最短路，于是可以用迭代加深。第 i 次搜索时只搜索深度不超过 i 的状态，多次搜索，直到找到目标状态为止。这个 OI 里啥用没有，听听就行。

A* 搜索

其适用于搜索一个目标状态的最短路。对每个状态，有一个起点到它的实际代价 g，和一个它到目标状态的估计代价 h。A* 搜索是 bfs 的变体，其会优先搜索 $g+h$ 最小的状态。
这个 OI 里啥用没有，听听就行。

IDA*

上面俩结合一下，也是 OI 里啥用没有。

Thank You