欧拉回路与欧拉路

一笔画判定定理、Hierholzer 套圈法与当前弧优化

HGNU-ACM 实验室算法组

前置：邻接表存图与 DFS 遍历（第 1 章）｜配套：《骑马修栅栏》《欧拉回路》精讲





本章目录

一笔画引入与欧拉图判定定理

Hierholzer 套圈法与当前弧优化

第 1 节搞懂“一笔画为什么只能有 0 个或 2 个奇度点”，以及有向图、连通性怎么判；第 2 节掌握 Hierholzer 算法“回溯倒序入栈”与当前弧优化防 TLE。

本章目录（续）

两大欧拉路与欧拉回路真题精讲

常见错误、检查清单与练习安排

第 3 节逐页手推《骑马修栅栏》与《欧拉回路（1527）》两道真题；第 4 节总结常见 WA/TLE 排查清单与课后练习安排。

生活引入：扫雪车怎样不走冤枉路？

从手算小例子开始

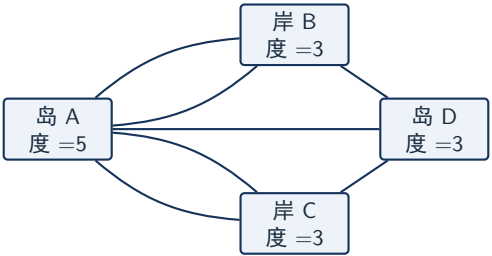
假设小区里有 4 个路口和 5 条街道。扫雪车想每条街道恰好走一次、不重复也不遗漏，能办到吗？

- 在第 1 章我们学过用邻接表存图、用 DFS 访问所有点；今天的问题换成了：把每条边恰好走一次！
- 写程序前先回答三个问题：能不能一笔画完？该从哪个点出发？遇到岔路怎么走才不会漏边？
- 大白话定义：如果一条路线能不重复、不遗漏地走完所有边，它叫欧拉路（欧拉通路）；如果最后还能回到起点，它叫欧拉回路。

本页要点

先判断“存不存在”再动手找路径；把每条边走一遍的路径叫欧拉路，能回到起点叫欧拉回路。

图解经典：为什么哥尼斯堡七桥走不完？



- 大白话术语：度数（**Degree**）就是一个点连着几条边；连接奇数条边的点叫奇度点，偶数条边叫偶度点。
- 七桥问题里 4 个点全都是奇度点（$5, 3, 3, 3$），无论你怎么试，总有一座桥过不去。

本页要点

1736 年欧拉证明：能否一笔画，只看图里有多少个“奇度点”以及边是不是连成一片！

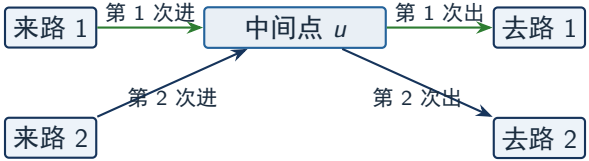
如果不用定理，暴力搜索为什么会 TLE？

- 暴力做法：用 DFS 枚举所有边的排列顺序，遇到走不通就退回换另一条边，直到走完 m 条边。
- 操作次数估算：当每个点平均连出 d 条边时，每步都有多种分支，最坏需要尝试 $O(m!)$ 或指数级状态。
- 若题目给 $m = 1000$ 条边（如《骑马修栅栏》），$1000!$ 远大于计算机每秒 10^8 次运算的极限，必然 **TLE**！
- 如果先判存在性：数清每个点的度数、检查一次连通性，只需 $O(n + m)$ 步就能 1 毫秒判出“行不行”。

本页要点

盲目暴搜尝试边的顺序是指数级 TLE；先用度数定理 $O(n + m)$ 判定，再线性构造路径才是正解！

核心原理：为什么中间点必须是“偶度点”？



- 只要点 u 既不是起点也不是终点，每次从一条边“进”来，就必须立刻从另一条边“出”去。
- 一进一出成对出现，消耗 2 条边，因此中间点的度数一定是 2 的倍数（偶度点）！
- 只有起点可以多出 1 条“只出不进”的边，终点可以多出 1 条“只进不出”的边。

本页要点

除了起点和终点，路径中所有点都是“进一次、出一次”，因此奇度点只能是起点或终点！

无向图欧拉路与欧拉回路的判定表

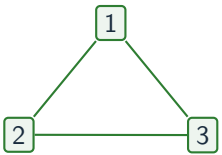
奇度点个数（连通时）	结论	起点与终点怎么选
0 个（全是偶度点）	存在欧拉回路	从任意有边点出发，最终必然回到出发点
恰 2 个奇度点	存在欧拉路	必须从一个奇度点出发，停在另一个奇度点
其他（1 个或 >2 个）	无解（不存在）	无向图里奇度点必定成对出现，>2 个摆不平起终点

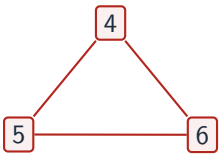
- 为什么无向图不会有“恰 1 个奇度点”？因为每条边贡献 2 个度数，全图总度数是偶数，奇度点个数必为偶数。
- 如果要找字典序最小的欧拉路，当奇度点有 2 个时，选编号较小的奇度点作起点。

本页要点

无向图连通时：0 个奇度点出回路（任意有边点起），2 个奇度点出欧拉路（奇度点起）。

致命前提：只数奇度点不查连通性会怎样？





7 (孤立点)

- 看左边两个分离的三角形：点 $1 \sim 6$ 度数全为 2，奇度点个数 $= 0$！但一笔能画完吗？不能！
- 还要注意右边孤立点 7：点 7 没有边（度为 0），它不影响三角形 $1-2-3$ 的一笔画！
- 因此连通性检查的准确写法是：所有度数 > 0 的点（有边的点）必须属于同一个连通块。

本页要点

只判度数不判连通性遇到“两个分离的环”会 WA；孤立点（度为 0）不算障碍！

有向图怎么判？承接第 1 章的“入度与出度”

目标路径类型	有向图度数充要条件（所有有边点弱连通）
有向欧拉回路	每个点都满足 $\text{in}[u] == \text{out}[u]$（进来几次就出去几次）
有向欧拉路	恰有 1 个起点满足 $\text{out}[s] - \text{in}[s] == 1$；恰有 1 个终点满足 $\text{in}[t] - \text{out}[t] == 1$；其余点均满足 $\text{in}[u] == \text{out}[u]$

- 回忆第 1 章拓扑排序：有向边 $u \to v$ 让 $\text{out}[u]$ 加 1、$\text{in}[v]$ 加 1。
- 有向图千万不能只统计总度数 $\text{in}[u] + \text{out}[u]$ 的奇偶性！例如一个点入度 2、出度 0，总度数是偶数，但它只能进不能出，根本走不通。

本页要点

有向图看“出度减入度”：回路要求全相等；通路要求起点出多 1、终点入多 1，其余全相等。

手算跟踪表：用充要条件检验 4 张小图

测试图描述	度数 / 出入度	奇点/异常点	连通？	判定结论
图 A：$1-2-3-1$ 加 $1-4$	$\deg = [3,2,2,1]$	奇点：$1,4$（2 个）	是	欧拉路
图 B：$1-2-3-1$ 与 $4-5-4$	$\deg = [2,2,2,2,2]$	奇点：0 个	否	无解
图 C：$1\to2, 2\to3, 3\to2$	$(0,1),(2,1),(1,1)$	1 出多 1，2 入多 1	是	欧拉路
图 D：$1\to2, 1\to3$	$(0,2),(1,0),(1,0)$	1 出多 2	是	无解

- 图 C 中点 2 出度 1、入度 2，恰为终点；路径 $1\to2\to3\to2$ 走完所有有向边。

本页要点

判定分两步走：先扫度数排除不合法，再跑一次搜索看是否走完所有边（连通性）。

正确性论证：为什么“边数用尽”就能代替单独判连通？

论证阶段	核心不变量说明
初始化	先验证所有点的度数合法（无向奇点数 $\in \{0,2\}$，有向出入度差满足条件），并从第一个有边点（或唯一合法起点）出发。
保持	若所有有边点在同一个连通块内，任意一条尚未走过的边都能从当前连通块顺着图走过去。
终止	当算法从起点出发走完后，记录下的边数恰好等于 m，就说明没有任何边落在别的孤立连通块里！

- 小技巧：考场上不必专门再写一遍并查集或额外 BFS 来判连通！只要跑完 Hierholzer 算法后判断结果边数是否等于 m（即顶点序列长度是否等于 $m+1$），就能同时完成连通性检验！

本页要点

“度数合法 + 走完后的边数恰好等于 m”，等价于“所有有边点连通且找到完整欧拉路”！

模板代码：无向图欧拉路判定与选起点

```
int odd = 0, start = 1;
for (int u = 1; u <= n; ++u) {
    if (deg[u] > 0) start = u;              // 先找任意一个有边点当保底起点
    if (deg[u] & 1) ++odd;                  // 统计奇度点个数
}
if (odd != 0 && odd != 2) {                 // 奇度点不是 0 或 2，直接无解
    cout << "NO\n"; return 0;
}
for (int u = 1; u <= n; ++u) {
    if (deg[u] & 1) { start = u; break; }   // 若有奇度点，必须从奇度点出发！
}
```

本页要点

有奇度点必须选奇度点；全是偶度点则选任意度数 > 0 的点！

上机验证：3 组边界测试与反事实追问

测试类型	输入图（n 点 m 边）	预期判定	考查目的
普通合法图	$n=4, m=4$：$1-2, 2-3, 3-4, 4-1$	欧拉回路	奇点数 $=0$，起点 1 合法
孤立点干扰	$n=4, m=3$：$2-3, 3-4, 4-2$（1 无边）	欧拉回路	起点不能写死 1，应选 2
双环不连通	$n=4, m=4$：自环 $1-1$，$2-3-4-2$	无解 NO	奇点虽为 0，但边未全连通

反事实追问

1. 如果把“找最小奇度点”的 `break` 删掉，对存在性判定有影响吗？对“字典序最小”呢？
2. 如果输入中有自环 (u, u)，无向图里 `deg[u]` 应该加 1 还是加 2？（提示：自环给 u 贡献了“一进一出”两个端头，必须加 2！）

本页要点

测试要专挑“1 号点没边”“两个分离环”“自环”这三类边界，才能提前抓出漏洞！

易错对照：判定阶段的 3 个高频坑

错误类型	错误写法	正确做法与后果
WA 起点写死 1	直接 `dfs(1)` 不找有边点	若 1 号点度数为 0（孤立点），路径长度为 $1 < m+1$ 误判无解
WA 漏判连通性	只判断 `odd == 0 || odd == 2`	两个分离偶环也会被误判有解，必须检查最终边数 `== m`
WA 有向图套无向公式	有向图统计 `(in[u]+out[u])%2`	有向图必须分别核对 $\mathrm{out}[u]-\mathrm{in}[u]$ 是否全为 0 或仅一对 ± 1

本页要点

判定阶段只防三件事：1 号可能是孤立点、图可能不连通、有向图必须查出入度差！

概念辨析：欧拉图 vs 半欧拉图

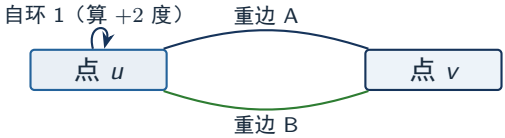
术语叫法	大白话含义	对应奇度点 / 出入度条件（连通时）
欧拉图	存在欧拉回路（能回到起点）	无向图奇点数 $= 0$；有向图全满足 in == out
半欧拉图	存在欧拉路但不能回起点	无向图奇点数 $= 2$；有向图恰一对差 ± 1
非欧拉图	无法一笔画完	其余所有情况（或不连通）

- 题面常说“判断是不是欧拉图”或“求欧拉通路”：看到回路就必须回到起点，看到通路/欧拉路则允许起终点不同。

本页要点

“欧拉图” $=$ 存在闭环一笔画；“半欧拉图” $=$ 存在不闭合的一笔画！

细节图解：自环与重边的度数怎么数？



- 自环（**Self-loop**）：一条边自己连回自己。从 u 出发又回到 u，占用 u 处“一出一进”共 2 个端头！
- 因此在无向图中，存图代码 `g[u].push_back({u, e}); g[u].push_back({u, e});`（或者只存一次但注意度数 $+2$）；用 `used[e]` 判重即可自然走一次跳过第二次！
- 图上点 u 的总度数是 $2+1+1=4$（偶度点）。

本页要点

无向图自环贡献 2 度，重边每条单独贡献 1 度，千万不能漏算！

有向图连通性细节：弱连通 vs 强连通

- 在第 1 章我们知道有向图有方向，那么判定有向欧拉回路时，怎么检查连通性？
- 大白话术语：弱连通就是把有向边暂时当成无向边看，所有有边点连成一片；强连通则是任意两点都能顺着箭头互达。
- 神奇结论：只要所有有边点满足 $\text{in}[u] == \text{out}[u]$ 且弱连通，它就已经自动强连通了，一定存在有向欧拉回路！
- 实操上依然不用单独写强连通分量，直接跑一次 Hierholzer 后检查走过的边数是不是等于 m 即可！

本页要点

有向图照样靠“检查度数 + 跑完边数是否等于 m”一步搞定连通性判断！

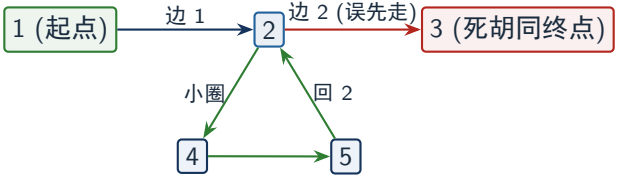
判定速查总结：无向图 vs 有向图一图看清

判断目标	无向图条件（所有有边点连通）	有向图条件（所有有边点弱连通）
欧拉回路	所有点度数为偶数（奇点数 $= 0$）	所有点都满足 $\mathrm{in}[u] == \mathrm{out}[u]$
欧拉路	恰有 2 个奇度点，从其一出发	恰 1 点 $\mathrm{out} - \mathrm{in} = 1$（起），$1$ 点 $\mathrm{in} - \mathrm{out} = 1$（终）
起点选择	有奇点选奇点，无奇点选有边点	有起点选唯一起点，否则选任意有边点

本页要点

无向数奇点（0 或 2），有向看入出差（全 0 或一对 ± 1），最后核对是否走完 m 条边！

新手大坑：顺着 DFS 边走边记点为什么会出错？



- 假设程序在点 2 先挑了通往死胡同 3 的边：若刚进 `dfs` 就记录，路径变成 `1 -> 2 -> 3`。
- 等退回点 2 再去走小圈 $2 \to 4 \to 5 \to 2$，如果接在后面就会变成 `1 -> 2 -> 3 -> 4 -> 5 -> 2`——从 3 根本跳不到 4！

本页要点

正序记点会把“死胡同终点”提前记死，导致半路上挂着的小圈无法插回正确位置！

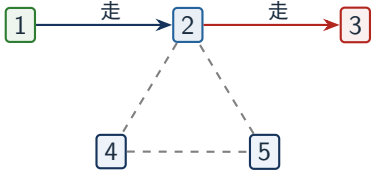
核心绝技：走到尽头，在“回溯退栈”时才记点！

- **Hierholzer**（套圈法）大白话：遇到岔路随便挑一条未走过的边往前冲，冲进函数时绝不记点。
- 只有当某个点 u 的所有出边都走光了、正要从函数返回（退栈）的那一瞬间，才写 `path.push_back(u)`！
- 这意味着：最先走到的死胡同终点 3，最先被压进 `path`；随后退回 2 吃掉小圈，最后才压入起点 1。
- 等全部退栈结束，把 `path` 整个反转（`reverse`），死胡同终点就自动落到了整条路径的最末尾！

本页要点

口诀：进函数只管走边，所有边走光准备退出时再压点；全部走完后把序列反转一遍！

图解 Hierholzer（1/2）：先冲向终点，退栈遇到未走小圈

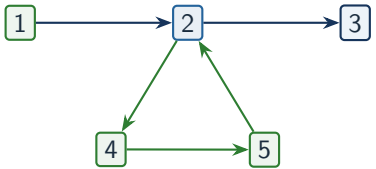


第 1 阶段：
$1 \to 2 \to 3$，点 3 无路可走！
点 3 退栈：path = [3]
退回到点 2，发现还有小圈未走！

本页要点

点 3 所有边已耗尽，它一定是整条路的终点，所以第 1 个被压入 path！

图解 Hierholzer（2/2）：吃掉小圈再继续退栈，最后反转



第 2 阶段：
从 2 走完 $2 \to 4 \to 5 \to 2$，
依次退栈压点：$2, 5, 4, 2, 1$！
`path = [3, 2, 5, 4, 2, 1]`
反转：`1-2-4-5-2-3`！

本页要点

小圈恰好嵌在了第一个 2 和第二个 2 之间，一次反转直接拼出完美一笔画！

手算跟踪表：5 点 5 边完整 Hierholzer 步进过程

步	当前状态	动作说明	递归栈	path（压栈）
1	进入 dfs(1)	走 $1-2$，标记已用	[1]	[]
2	进入 dfs(2)	先走 $2-3$，标记已用	[1, 2]	[]
3	进入 dfs(3)	无未走边，退栈记点	[1, 2]	[3]
4	回到 dfs(2)	走完小圈 $4-5-2$	[1,2,4,5,2]	[3]
5	圈尾 2,5,4 退栈	圈上各点依次退栈	[1, 2]	[3,2,5,4]
6	2，1 退栈（终止）	全部出边耗尽退出	[]（空）	[3,2,5,4,2,1]

- 终止后执行 reverse(path.begin(), path.end())，得到最终结果：1 -> 2 -> 4 -> 5 -> 2 -> 3。

本页要点

每次退栈都发生在“该点已无剩余出边”之时，保证绝不会漏掉任何挂在半路上的小环！

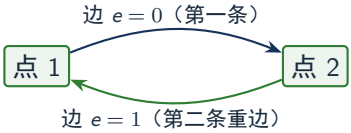
正确性论证：为什么“退栈压点 + 反转”一定合法？

论证阶段	核心不变量说明
初始化	从合法起点出发，每一步只走尚未标记 `used[e]` 的边，走过立即标记，保证每条边至多走一次。
保持	若点 u 在进入下条边前先走完了一条旁支环（起点终点都是 u），则环上所有点会先于 u 压栈；`reverse` 后整段环恰落在 u 之前并接回 u。
终止	当起点 `dfs(start)` 退栈结束，若所有 m 条边都已被消耗，则逆序后的序列相邻两点必有边相连，且总边数为 m。

本页要点

本质是“树状嵌套的环拼接”：先把主干终点压到栈底，再把沿途小环一层层嵌回对应节点。

重边陷阱：为什么不能用 vis[u][v] 矩阵判重？

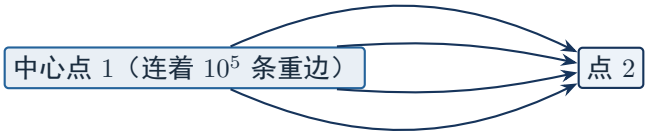


- 什么是重边？两个点之间有 2 条及以上独立的边（例如两座并排的桥）。
- 如果用二维数组 vis[1][2] = 1 标记“1 到 2 走过了”，那么第二条桥 $e=1$ 也会被误以为走过而跳过，直接漏边 **WA**！
- 正确做法：给每条读入的边编上唯一编号 $e=0,1,\ldots,m-1$，邻接表存 {邻居 v，边编号 e}，无向边两端存同一个 e，用 used[e] 判重！

本页要点

按“边编号 e”建一维数组 used[e] 判重，天然支持任意多条重边与自环！

暴力扫邻接表为何会 TLE？认识当前弧优化

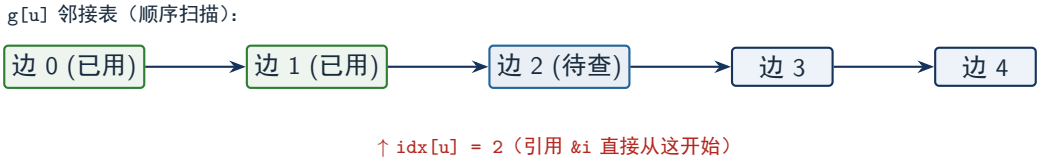


- 试想一个“菊花/重边图”：点 1 和点 2 之间有 $m = 2 \times 10^5$ 条重边，每次在 1 和 2 之间来回跳。
- 如果每次进 `dfs(1)` 都写 `for (int i = 0; i < g[1].size(); ++i)` 从头扫起：
- 第 1 次跳到第 0 条，第 2 次要跳过前 1 条……第 k 次要跳过前 $k-1$ 条！总扫描次数为 $1+2+\cdots+m \approx \frac{m^2}{2} = 2 \times 10^{10}$ 次，必 **TLE**！

本页要点

每次从下标 0 扫起遇到大量重边会退化成 $O(m^2)$ TLE；必须记录“上次扫到了第几条边”！

图解当前弧优化：int &i = idx[u] 永不回头



- 用一个数组 idx[u] 记录点 u 下一次应该看哪条边，初始全为 0。
- 在循环里写 for (int &i = idx[u]; i < (int)g[u].size();): &i 是 idx[u] 的引用（别名）！
- 每看一条边就执行 i++，不仅跳过已用边，连同 idx[u] 自己也永久往前推了一格，每条边最多被看 1 次，总复杂度降回 $O(n+m)$！

本页要点

记住“当前弧指针”写法：int &i = idx[u]，引用推进保证每条边只检查一次！

Hierholzer 完整模板（1/2）：存图与找起点

```cpp
vector<vector<pair<int, int>>> g(n + 1);
for (int e = 0, u, v; e < m; ++e) {
    cin >> u >> v;
    g[u].push_back({v, e});                  // 存 {邻居, 边编号}
    g[v].push_back({u, e});                  // 无向边两端用同一个编号 e
}
int odd = 0, start = 1;
for (int u = 1; u <= n; ++u) {
    if (!g[u].empty()) start = u;            // 保底：找任意有边点
    if (g[u].size() & 1) ++odd;
}
if (odd != 0 && odd != 2) { cout << "NO\n"; return 0; }
for (int u = 1; u <= n; ++u)
    if (g[u].size() & 1) { start = u; break; } // 若有奇度点必须选它
```

本页要点

无向图两条有向边共享同一个编号 e，才能做到“走过去顺便把反向也锁死”。

Hierholzer 完整模板（2/2）：当前弧 DFS 与输出

```
vector<char> used(m, 0);
vector<int> idx(n + 1, 0), path;
auto dfs = [&](auto &&self, int u) -> void {
    for (int &i = idx[u]; i < (int)g[u].size(); ) { // 当前弧引用推进
        auto [v, e] = g[u][i++];
        if (used[e]) continue;              // 这条边已被走过，跳过
        used[e] = 1;                        // 必须在递归前标记！
        self(self, v);
    }
    path.push_back(u);                      // 退栈才压点！
};
dfs(dfs, start);
if ((int)path.size() != m + 1) { cout << "NO\n"; return 0; } // 判连通性
reverse(path.begin(), path.end());          // 反转得正序路径
```

本页要点

第 13 行用 path.size() != m + 1 顺手检查图是否连通，一举两得！

为什么这样写？三个核心细节追问

1. 为什么 `used[e] = 1` 必须写在 `self(self, v)` 之前？
   如果写在递归之后，一旦环路绕回自己（或自环），就会重复走进同一条边导致死循环 **RE**（爆栈）！
2. 为什么 `idx[u]` 改成普通 `int i = idx[u]`（不加 `&`）会超时？
   不加 `&` 只是局部变量拷贝，函数返回后 `idx[u]` 还是 0，当前弧优化立刻失效退化成 $O(m^2)$ **TLE**！
3. 为什么用 `i++` 写在 `g[u][i++]` 里而不是循环末尾？
   因为进入递归 `self(self, v)` 之前，当前弧指针 `idx[u]` 必须已经推到下一条，这样别人再回到 u 时才不会重复看这一条！

本页要点

“先自增 `i++`，再标记 `used[e]=1`，最后进递归 `self(v)`”，顺序绝对不能乱！

如何求“字典序最小”的一笔画路径？

- 题目常要求：如果有多条一笔画路径，输出字典序最小（即第 1 个点尽量小，相同再比第 2 个点……）的那条。
- 两条铁律：
  1. 选起点：若有 2 个奇度点，选编号较小的奇度点；若全是偶度点，选编号最小的有边点。
  2. 排边序：对每个点的邻接表 `sort(g[u].begin(), g[u].end())`，按邻居编号升序排列，DFS 优先走小邻居！
- 常见疑惑：为什么“先走小邻居”，在退栈压入、最后 `reverse` 之后，小邻居反而排在最前面？

本页要点

记住动作：起点挑最小，每个点的出边按邻居编号从小到大排序即可！

手算对比：为什么“先走小邻居 + 退栈反转”字典序最小？

测试小图	邻接表排序做法	手算退栈与反转结果
点 1 有两条分支：$1-2$ 与环 $1-3-1$	先走大邻居 3（错误）	退栈顺序 [2，1，3，1] $\rightarrow$ 反转 1-3-1-2
	先走小邻居 2（正确）	点 2 先入栈 [2]，后走环得 [2，1，3，1]

- 精确规律：能早早走进死胡同的点（唯一终点），无论怎么走都在最后；在能绕回来的多个环里，先走的小环会被留在栈的更深处还是更浅处？
- 从 u 先走 $u \rightarrow a \rightarrow \cdots \rightarrow u$（小邻居 a），它会立刻走完并压栈；然后走大邻居 b 的环。若只看环 $1-2-1$ 和 $1-3-1$：先走 2 回来时，1 还在循环里没退栈！走完 3 才在深层压入 $1,3,1$，最外层再压 $2,1$，反转后究竟是 1-2-1-3-1 还是 1-3-1-2-1？

本页要点

下页用 4 步逐行跟踪，彻底验证“小邻居在栈里到底排哪儿”！

逐行跟踪验证：两个环挂在点 1 时谁先被压栈？

步骤	升序排序（先走邻居 2，再走邻居 3）	当前 path 内容
1	dfs(1) 选边 $1-2$ → dfs(2) 选边 $2-1$ → 进 dfs(1)	[]
2	此时在深层 dfs(1) 里，idx[1] 已经指到边 $1-3$！走 $1-3-1$	[]
3	深层 dfs(1) 无边可走，依次退栈压入：$1, 3, 1$	[1, 3, 1]
4	回到 dfs(2) 退栈压入 2；回到最外层 dfs(1) 压入 1	[1, 3, 1, 2, 1]

- 把 path = [1, 3, 1, 2, 1] 反转：1 -> 2 -> 1 -> 3 -> 1！
- 看！先走的小邻居 2 挂在最外层递归里，最后才退栈，反转后恰恰稳居第 2 位！

本页要点

正是因为先走小邻居 2 深入后把大邻居 3 吞在深层先退栈，反转后 2 才排在 3 前面！

上机验证：Hierholzer 完整代码 3 组实测

测试场景	输入数据（n, m 与边）	预期输出	检验点
纯偶点含重边	$n=2, m=2$：`1 2`，`1 2`	`1 2 1`	两条重边都能走完
非 1 号作起点	$n=3, m=2$：`2 3`，`3 2`（1 孤立）	`2 3 2`	跳过孤立点 1 选 2
字典序分叉	$n=4, m=4$：`1 3`，`3 1`，`1 2`，`2 4`	`1 3 1 2 4`	终点 4 必在最后

反事实追问

1. 删掉 `reverse(path)` 直接正序输出：若是回路倒着走仍合法但字典序偏大；若是欧拉路起终点颠倒会 **WA**！
2. 在 $m = 2 \times 10^5$ 的重边链上删掉 `idx[u]` 的 `&` 符号，复杂度立刻退化成 $O(m^2)$ **TLE**！

本页要点

用“重边图 + 孤立点 + 字典序分叉”这三组小数据，1 分钟就能把 Hierholzer 测透！

易错对照：Hierholzer 实现阶段的 4 个经典错误

错误类型	错误写法	正确做法与原因
WA 进函数就压栈	在 `for` 循环之前写 `path.push_back(u)`	遇到半路小环会断路，必须在循环结束后压栈再反转
TLE 漏掉当前弧	循环写成 `for (int i = 0; i < g[u].size(); ++i)`	大量重边退化为 $O(m^2)$，必须写 `int &i = idx[u]`
WA 矩阵判重	用 `vis[u][v]` 标记两点之间已走过	重边直接被跳过丢边，必须按唯一边编号 `used[e]` 判重
RE 递归栈溢出	忘了在递归前写 `used[e] = 1`	在环上无限绕圈直到系统栈爆掉（段错误）

本页要点

记牢四防线：退栈压点、`&i = idx[u]`、边编号 `used[e]`、先标记再递归！

为什么需要非递归？防止 2×10^5 层递归爆栈

- 当 $m = 2 \times 10^5$ 且图是一条长链时，递归调用深度可达 2×10^5。
- 在栈空间受限的评测环境或 Windows 本机测试时，过深的递归会直接触发 **RE**（爆栈/段错误）！
- 解决方法：自己开一个 `vector<int> st` 放在堆内存里当作显式栈，手动模拟函数“进栈继续走”与“无边可走才退栈记点”！

本页要点

数据规模达 10^5 时，把递归改成显式栈能彻底避免系统栈溢出 RE！

防爆栈技巧（1/2）：显式栈 14 行模板代码

```
vector<int> st = {start}, path;
while (!st.empty()) {
    int u = st.back();                          // 看栈顶元素（先不弹出！）
    int &i = idx[u];
    while (i < (int)g[u].size() && used[g[u][i].second]) ++i;
    if (i < (int)g[u].size()) {
        auto [v, e] = g[u][i++];
        used[e] = 1;
        st.push_back(v);                        // 还有边可走，把邻居压栈继续冲
    } else {
        path.push_back(u); st.pop_back();       // 无边可走才弹出并记点！
    }
}
reverse(path.begin(), path.end());
```

本页要点

显式栈写法只有 14 行，逻辑和递归一模一样，但彻底免疫递归栈溢出！

防爆栈技巧（2/2）：显式栈对比递归的三个对应点

动作	递归版写法	显式栈迭代版写法
往前走一条新边	调用 `self(self, v)`	`st.push_back(v)`（不弹出 u）
跳过已走过的边	`if (used[e]) continue;`	`while (... && used[...]) ++i;`
无路可走回溯记点	循环结束后 `path.push_back(u)`	`else { path.push_back(u); st.pop_back(); }`

- 为什么走新边时不能先把 u 弹出？因为走完旁支小圈还要退回 u 继续找下一条边！
- 只有当 `i == g[u].size()` 确认点 u 无路可走时，才把 u 弹出并放进 path。

本页要点

栈顶只在“无边可走”时才弹出；只要还有边，就只压入新点不弹出旧点！

有向图极简写法：用 pop_back() 替代当前弧

- 在有向图里，每条边只出现一次（不像无向图在两点各存一份），不需要 used[e] 数组！
- 只要每次从 g[u] 的末尾取一条边并直接 pop_back() 删掉，走过的边就永久消失了，天然实现 $O(1)$ 当前弧！

```
auto dfs = [&](auto &&self, int u) -> void {
    while (!g[u].empty()) {
        int v = g[u].back();
        g[u].pop_back();                    // 直接弹掉这条有向边，永不再走！
        self(self, v);
    }
    path.push_back(u);
};
```

- 如果要字典序最小怎么办？因为每次从 back() 尾部弹出，所以存完边后要把 g[u] 按降序排序（最小的编号留在尾巴最后被最先弹出）！

本页要点

有向图不需要 used[e] 和 idx[u]，直接 pop_back() 删边省一半代码！

综合快问快答：考场 30 秒排雷小测验

请先口头作答，下一页核对答案

1. 图有 4 个点，边为 $(1,2),(2,3),(3,1)$，点 4 没有连边。问：它存在欧拉回路吗？起点该选几号？
2. 某同学写 Hierholzer 忘了写 `int &i = idx[u]` 而写了 `int i = 0`，样例过了但交题 **TLE**，为什么？
3. 求有向图字典序最小欧拉路，如果采用 `g[u].pop_back()` 删边法，邻接表应该升序排序还是降序排序？

本页要点

先想 10 秒再翻页，考场遇到这三类坑能救回 100 分！

综合快问快答：答案与核心解析

1. 存在！点 4 是孤立点，不影响所有有边点 $\{1,2,3\}$ 构成的连通偶环。起点选 $1,2,3$ 都可以（选最小有边点 1），绝对不能选 4！
2. 遇到大量重边或稠密图时，每次递归进点 u 都从下标 0 重新扫起，总扫描次数高达 $O(m^2)$，必然 **TLE**。
3. 必须降序排序！因为 `back()` 拿的是容器末尾元素，把最小邻居放在末尾，才能第一个被弹出走出去！

本页要点

记住：孤立点不挡路、重边必加当前弧、尾部弹出需倒着排！

例 1 | [基础] 骑马修栅栏（1375）： 先读懂要求

大白话题意

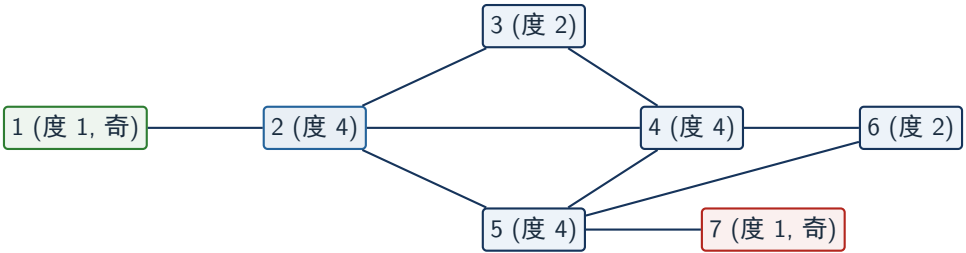
给 F 条栅栏（$1 \le F \le 1024$），每条栅栏连接 $1 \sim 500$ 中的两个顶点（可能存在重边）。要求从某个顶点出发，每条栅栏恰好经过一次，若有多种路线，输出字典序最小的一条。

- 输入：第 1 行整数 F；接下来 F 行每行两个整数 i, j，表示一条无向栅栏。
- 处理：找一条字典序最小的无向图欧拉路或欧拉回路。
- 输出：共 $F+1$ 行，依次输出经过的顶点编号。
- 原题样例：输入 $F = 9$ 条边 $(1,2), (2,3), (3,4), (4,2), (4,5), (2,5), (5,6), (5,7), (4,6)$，输出 `1 2 3 4 2 5 4 6 5 7`（每行一个）。

本页要点

500 进制数最小 $\iff$ 第 1 个顶点尽量小，其次第 2 个尽量小 $\iff$ 顶点序列字典序最小！

例 1 | 怎么想到的与建模 TikZ 图



- 把栅栏交点看作顶点，栅栏看作无向边；“每条栅栏走一次”就是找欧拉路。
- 点 1 与点 7 度数为 1（奇度点），其余点度数为偶，因此必须选较小的奇度点 1 当起点！
- 每个点邻居编号按升序排序，再用带当前弧和 `used[e]` 的 Hierholzer 即可。

本页要点

题目编号到 500 但未必 500 个点全出现，存图开 $N = 500$ 并把每个点邻居按升序排序！

例 1 | 样例手推：关键转折与退栈过程

阶段	走法选择（优先走小邻居）	关键动作说明	path 变化
1	$1 \to 2 \to 3 \to 4 \to 2$	在 2 选最小邻居 3，绕小圈回到 2	`[]`
2	$2 \to 5 \to 4 \to 6 \to 5 \to 7$	在 2 改走边 $2-5$，一直冲到奇点 7	`[]`
3	点 7 无未走边	首次遇到死胡同，7 最先退栈入列	`[7]`
4	依次退栈 $5,6,4,5,2,4,3,2,1$	沿途所有点均已耗尽出边，逐个压栈	`[7,5,6,4,5,2,4,3,2,1]`
5	结束执行 `reverse(path)`	反转得到最终序列	`1 2 3 4 2 5 4 6 5 7`

- 请留意第 2 阶段：在 4 号点时，边 $4-2, 4-3$ 都用过了，剩下 $4-5$ 和 $4-6$，按升序选了更小的 5！

本页要点

手推时牢记：每次在岔路口选编号最小的未用边，走到死胡同才开始退栈压点！

例 1 | 核心步骤表与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 读 m 条边并双向存编号 e	邻接表 g[1..500]	是否把重边编了不同 e?
2. 倒序找最小有边点并排序	邻接表升序、保底起点	若全偶点是否选了最小有边点?
3. 正序找首个奇度点	最终起点 start	若有奇度点，起点是不是最小奇点?
4. 跑当前弧 Hierholzer	倒序顶点序列 path	是否在退栈时压入，结束再反转?

相对基础模板改了哪里

1. 题面未给总点数 n，固定上限 $N = 500$;
2. 增加了对每个点邻接表 `sort(g[u].begin(), g[u].end())`，保证每次优先尝试小邻居;
3. 保证有解，只需按“倒序找非空点 $\rightarrow$ 正序找奇点”确定最小合法起点。

本页要点

模板微调两处：固定顶点上限 500，以及对每个点的邻接表排序求最小字典序!

例 1 | 核心代码（1/2）：建图、排序与选最小起点

```cpp
int m; if (!(cin >> m)) return 0;
const int N = 500;
vector<vector<pair<int, int>>> g(N + 1);
for (int e = 0, u, v; e < m; ++e) {
    cin >> u >> v;
    g[u].push_back({v, e}); g[v].push_back({u, e}); // 双向共用 e
}
int start = 0;
for (int u = N; u >= 1; --u) {
    if (!g[u].empty()) start = u;            // 全偶点停在最小有边点
    sort(g[u].begin(), g[u].end());          // 按邻居编号升序排序
}
for (int u = 1; u <= N; ++u)
    if (g[u].size() % 2 == 1) { start = u; break; } // 有奇点选最小奇点
```

本页要点

倒序扫描非空点，最后留在 start 里的一定是最小非空点！

例 1 | 核心代码（2/2）：当前弧 DFS 与输出

```cpp
vector<char> used(m, 0);
vector<int> idx(N + 1, 0), path;
auto dfs = [&](auto &&self, int u) -> void {
    for (int &i = idx[u]; i < (int)g[u].size(); ) { // 当前弧优化防重边 TLE
        auto [v, e] = g[u][i++];
        if (used[e]) continue;
        used[e] = 1;
        self(self, v);
    }
    path.push_back(u);                          // 退栈才记点！
};
dfs(dfs, start);
reverse(path.begin(), path.end());              // 反转得到字典序最小路径
for (int u : path) cout << u << '\n';
```

本页要点

代码与 code/euler_fence.cpp 完全一致，共 14 行核心逻辑搞定一切重边与字典序！

例 1 | 易错点与 3 组自测边界

自测场景	构造输入	预期输出	检验的易错点
两顶点重边	2 / 1 2 / 1 2	1 2 1	用 vis[u][v] 判重会漏掉第二条边
无 1 号顶点	3 / 2 3 / 3 4 / 4 2	2 3 4 2	起点写死 1 会输出孤零零的 1
2 个奇点倒序给	3 / 3 2 / 2 4 / 4 5	3 2 4 5	奇点是 3 和 5，不能错选 5 当起点

- 常见报错排查：输出行数不够 $F+1$ 行 $\rightarrow$ 查是否用点对判重漏了重边；输出字典序偏大 $\rightarrow$ 查邻接表是否忘了 sort 或奇点未取最小。

本页要点

提交前用“不含 1 号点 + 两条平行重边”的 3 行小样例自测，过关再交！

例 2 | [提高] 欧拉回路（1527）： 先读懂要求

大白话题意

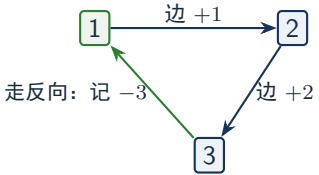
给定测试类型 $t \in \{1,2\}$（$t=1$ 表示无向图，$t=2$ 表示有向图），$n \le 10^5$ 个点、$m \le 2\times 10^5$ 条边（可能有重边、自环和孤立点）。判断是否存在欧拉回路，若存在输出 YES 及经过的边编号序列。

- 特殊输出要求：若 $t=1$（无向边 v_e-u_e），从 v_e 走到 u_e 输出正数 $+e$，反向从 u_e 走到 v_e 输出 $-e$；若 $t=2$，直接输出边编号 e。
- 无解判定：若不存在欧拉回路（含不连通），只输出一行 NO。
- 原题样例 **1**：$t=1, n=3, m=3$，边 $(1,2),(2,3),(1,3)\to$ 输出 YES 与 1 2 -3。

本页要点

本题三大升级：既要判无向/有向连通性，又要输出带正负号的“边编号”而非顶点编号！

例 2 | 怎么想到的与记录“边编号”图解



- 存无向边 i（u_i 连 v_i）时，`g[u_i]` 存 `{v_i, +i}`，`g[v_i]` 存 `{u_i, -i}`！
- 判重时用 `used[abs(e)] = 1`，走过一次就把这条边编号锁死。
- 怎么把边编号记下来？在 `dfs(u)` 中走完 `self(self, v)` 返回那一刻，把刚刚经过的带符号边编号 `ans.push_back(e)`！最后 `reverse(ans)` 即可！

本页要点

回溯时压入“刚刚走完的那条边 e”，比压入顶点更容易还原每一步走的是哪条边！

例 2 | 样例手推：无向图 3 点 3 边跟踪表

步骤	当前点	选中边与方向	used[1..3]	ans（压入序列）
1	1	边 $1 \to 2$，记 e = +1	[1, 0, 0]	[]
2	2	边 $2 \to 3$，记 e = +2	[1, 1, 0]	[]
3	3	边 $3 \to 1$（反向），记 e = -3	[1, 1, 1]	[]
4	1	深层 dfs(1) 无边可走，退出	[1, 1, 1]	[]
5	退栈	依次返回并压边：-3, +2, +1	[1, 1, 1]	[-3, 2, 1]

- 终止检查：ans.size() == 3 == m（全部边走完），执行 reverse(ans) 得 1 2 -3，与标答完全一致！

本页要点

回溯压边是在 self(self, v); 的下一行执行 ans.push_back(e);，绝不能写在循环外面！

例 2 | 核心步骤表与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 按 $t = 1/2$ 存带符号边	邻接表与 `in/out`	无向反向边编号是否取了负号？
2. 判 $m = 0$ 与所有点度数	是否直接 `YES/NO`	$t = 1$ 查偶度，$t = 2$ 查 `in==out`
3. 选首个非空点跑当前弧 DFS	边编号序列 `ans`	压边是否写在 `self(v)` 之后？
4. 查 `ans.size() == m` 并反转	`YES` + 序列或 `NO`	若边没走完是不是输出 `NO`？

相对基础模板改了哪里

1. 必须先用度数条件过滤掉不合法点；
2. 因为 $m \le 2 \times 10^5$，绝对不能忘记 `int &i = idx[u]` 当前弧，否则遇到重边/菊花图直接 **TLE**；
3. 需要特判 $m = 0$（没有边时本身就是合法空回路，输出 `YES`）。

本页要点

面对 2×10^5 规模的大图，“当前弧 + 边编号判重 + 检查 `ans.size()==m`” 缺一不可！

例 2 | 核心代码（1/2）：建图、度数判定与选起点

```
for (int i = 1; i <= m; ++i) {
    int u, v; cin >> u >> v;
    if (t == 1) {
        g[u].push_back({v, i}); g[v].push_back({u, -i});
        ++outdeg[u]; ++outdeg[v];
    } else {
        g[u].push_back({v, i}); ++outdeg[u]; ++indeg[v];
    }
}
if (m == 0) { cout << "YES\n\n"; return 0; }
for (int u = 1; u <= n; ++u) {
    if (t == 1 && (outdeg[u] & 1)) { cout << "NO\n"; return 0; }
    if (t == 2 && indeg[u] != outdeg[u]) { cout << "NO\n"; return 0; }
}
```

本页要点

无向图用 `outdeg[u]` 顺便充当总度数，有自环时两侧各加 1 恰好加 2，完全正确！

例 2 | 核心代码（2/2）：记录边编号与输出

```cpp
int start = 1;
for (int u = 1; u <= n; ++u) if (!g[u].empty()) { start = u; break; }
vector<char> used(m + 1, 0); vector<int> idx(n + 1, 0), ans;
auto dfs = [&](auto &&self, int u) -> void {
    for (int &i = idx[u]; i < (int)g[u].size(); ) {
        auto [v, e] = g[u][i++];
        if (used[abs(e)]) continue;          // 绝对值编号判重
        used[abs(e)] = 1; self(self, v);
        ans.push_back(e);                    // 退栈记录带符号边编号
    }
};
dfs(dfs, start);
if ((int)ans.size() != m) { cout << "NO\n"; return 0; }
reverse(ans.begin(), ans.end());
```

本页要点

代码与 `code/euler_circuit_1527.cpp` 一致：边没走完则输出 `NO`！

例 2 | 易错点与 3 组自测边界

自测场景	构造输入	预期	检验的易错点
零条边空图	`1 / 3 0`	YES	没特判 $m = 0$ 会输出多余数
分离双偶环	`1 / 4 4 / 1 1, 2 3, 3 4, 4 2`	NO	靠 `ans.size()!=m` 拦截
有向图自环	`2 / 2 2 / 1 1, 1 2`	NO	点 1 出 2 入 1，判 `in!=out`

- 防爆栈提醒：$m = 2 \times 10^5$ 时递归深度可达 2×10^5，栈受限环境可改用前文显式栈模拟。

本页要点

记住检查：`used[abs(e)]` 取绝对值、特判 `m==0`、`ans.size()==m` 判连通！

高频 WA 错误对照表

题目 / 场景	常见错误写法	正确做法与原因
记录路径顺序写反	在刚进入 `dfs(u)` 就记录 `u`	半路死胡同先于旁支环被记下会断路，必须退栈记点再反转
忽略重边按点判重	只用 `vis[u][v]` 矩阵判重	若两点间有 2 条重边会漏走第 2 条！必须按边编号 `e` 判重
起点选成固定 1 号	不找有边/奇度点直接 `dfs(1)`	图里 1 号可能根本没连边，或另有编号更小的奇度点必须作为起点
有向图忘判出/入度	拿无向图奇点数去套有向图	有向欧拉回路要求全图 $\text{in}[u] == \text{out}[u]$，欧拉路要求恰一对差 ± 1

本页要点

WA 四大高发区：正序记点、矩阵判重、写死起点 1、有向图套用无向奇点数！

高频 TLE / RE 错误对照与评测定位指南

评测结果与场景	常见错误写法	正确做法与原因
TLE 漏当前弧	每次进 u 都从下标 0 找边	大量重边/菊花图卡成 $O(m^2)$，必须用 `int &i = idx[u]`
TLE 拷贝非引用	写成 `int i = idx[u]` 忘了 `&`	指针没有真正推进，每次仍从同一位置扫起
RE 死循环爆栈	把 `used[e] = 1` 写在递归之后	在环或自环上无限绕圈，必须先标记再进 `self(v)`
WA 漏判不连通	只检查度数，没查 `ans.size() == m`	两个分离偶环度数也全偶，必须核对是否走完所有边

- 三秒定位口诀：**TLE** 先查 `&i = idx[u]`；**RE** 查是否递归前标了 `used[e]`；**WA** 查退栈压点、连通性与起终点。

本页要点

记住评测信号：TLE 找当前弧，RE 找标记时机，WA 找退栈顺序与连通性！

提交前检查清单（六必查）

1. 无向图判边使用过没，是不是用了每条边唯一的下标 `used[e]`（能自动处理重边与自环）？
2. 是否加入了当前弧指针 `int &i = idx[u]`（带引用符号 `&`）避免邻接表被反复从头扫描？
3. 记录顶点 `path.push_back(u)`（或边 `ans.push_back(e)`）是不是放在回溯退栈时，且最后执行了 `reverse`？
4. 若需要字典序最小一笔画，有没有对每个点的邻接表先 `sort`，且在找起点时优先找最小奇度点？
5. 若 1 号点没有任何边（孤立点），寻找起点循环是否找了第一个有边点而非写死 1？
6. 若图不连通导致边未走完（`ans.size() < m`），有没有按题意正确输出无解（如 `NO`）？

最后一步

造一组含 2 条重边、以及奇度点在 3 号而非 1 号的样例测试后再提交！

本章全部练习与完成标准

1. **[入门]** 学校 OJ：1341 ｜一笔画问题：无向图欧拉路/回路基础模板练习，输出一条合法路径即可。
2. **[基础]** 学校 OJ：1375 ｜骑马修栅栏 (fence)（`code/euler_fence.cpp`）：字典序最小欧拉路 + 重边处理 + 选最小起点。
3. **[提高]** 学校 OJ：1527 ｜欧拉回路 (UOJ #117)（`code/euler_circuit_1527.cpp`）：无向/有向双任务、带符号边编号输出与不连通判 `NO`。
4. **[提高]** 学校 OJ：1529 ｜欧拉回路：进阶无向/有向混合判定与多组数据清理练习。

完成标准

能一秒说出为什么必须退栈记录顶点再反转；能手写出带当前弧与边编号判重的 Hierholzer；独立 AC 上述 4 题。

课后安排与下一章预告

- 复习：画一个由 $1 \to 2 \to 3$ 且点 2 上挂着小三角形的图，手动走一遍退栈记录然后反转，体会小圈怎么嵌进路径。
- 练习：独立复写 `code/euler_fence.cpp` 与 `code/euler_circuit_1527.cpp`，并造含两条平行重边的小样例跑通。
- 下一章预告：第 3 章《最短路与差分约束》——从正权图小根堆 Dijkstra 懒删除，到含负边与判负环的 Bellman-Ford / SPFA，再把不等式组变成图上路径！

本章一句话

奇度点定起点，当前弧防重复扫描，走到尽头回溯压点再翻过来就是完整一笔画。

谢谢！