# 最短路与差分约束

## 堆优化 Dijkstra、Bellman-Ford / SPFA、Floyd 与不等式建模

HGNU-ACM 实验室算法组

前置：欧拉回路与图遍历（第 1～2 章）｜配套：《香甜的黄油》《虫洞》精讲






# 本章目录

从 BFS 到堆优化 Dijkstra 与懒删除

Bellman-Ford、SPFA 队列松弛与判负环

Floyd-Warshall 全源最短路

第 1～3 节依次掌握正权图主力 Dijkstra 懒删除、含负权与判负环的 SPFA、以及全源最短路 Floyd-Warshall。

# 本章目录（续）

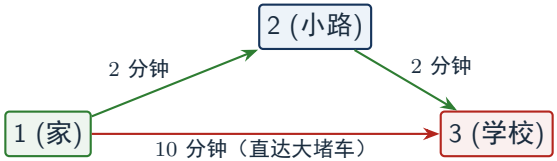
差分约束系统：不等式组转有向图

两大最短路与负环真题精讲

选型总表、常见错误与练习安排

第 4～6 节掌握不等式组转图的差分约束系统，精讲《香甜的黄油》《虫洞 Wormholes》，并总结常见 WA/TLE 排查清单。

# 生活引入：少转弯不等于路程最短！




- 第 1 章学的普通 BFS 只管“经过几条边”：它第 1 步摸到直达边 $1 \to 3$ 就以为到了，得出答案 10。
- 但绕道 $1 \to 2 \to 3$ 虽然多走 1 条边，总耗时只有 $2 + 2 = 4$ 分钟！
- 核心变化：带权图里，边上有长短不一的权重 $w$，我们必须按累计距离从小到大去扩散，而不再是“边数少先出队”。

## 本页要点

普通 BFS 只适用于边权全相等（如全为 1）的图；边权不同时必须改用带权最短路算法！

# 暴力做法为什么会 TLE？

- 暴力方案一（**DFS** 搜全路径）：枚举从起点到终点的所有简单路径取最小值，最坏是指数级 $O(n!)$，点数 $n = 20$ 就超时。
- 暴力方案二（朴素 **Dijkstra**）：一共做 $n$ 轮，每轮暴力遍历全图 $n$ 个点找出“尚未确定、且当前距离最小”的点 $u$，共需 $O(n^2)$ 次操作。
- 当考场给稀疏大图 $n = 10^5, m = 2 \times 10^5$ 时，$n^2 = 10^{10}$ 远超每秒 $10^8$ 次运算上限，必然 **TLE**！
- 提速思路：用小根堆（第 1 篇学过的 `priority_queue`）代替“每轮傻扫 $n$ 个点”，每次 $O(\log n)$ 弹出最近点，总复杂度降为 $O(m \log n)$！

## 本页要点

当 $n = 10^5$ 时，$O(n^2)$ 朴素 Dijkstra 必 TLE，必须上小根堆优化成 $O(m \log n)$！

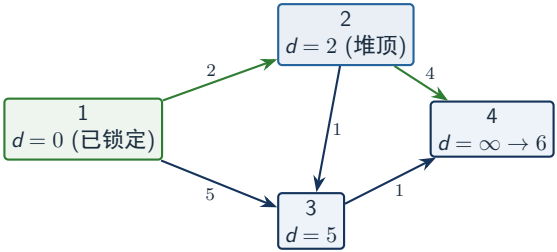
# 两个核心大白话术语：松弛与小根堆

- 松弛操作（**Relaxation**）：把边 $u \to v$（权为 $w$）想象成一根橡皮筋。如果“起点先到 $u$ 再走这条边到 $v$”比“目前已知到 $v$ 的距离”更短，即 `d[u] + w < d[v]`，就立刻把 `d[v]` 拉紧更新成 `d[u] + w`！
- 小根堆（**Min-Heap**）：C++ 里的 `priority_queue` 默认是大根堆（大的在顶），加上 `greater<>` 后就变成小根堆，永远把当前 `d[]` 最小的点放在堆顶第一个吐出来。
- 堆里存一对数 {当前距离，点编号}，因为 `pair` 默认先比第一维，距离最小的自动浮到堆顶！

## 本页要点

“小根堆挑最近点 + 用它的出边去松弛邻居”，这就是堆优化 Dijkstra 的全部核心！

# 图解 Dijkstra 执行流程：步步锁定最近点




- 起点 1 出堆，松弛得 $d[2] = 2, d[3] = 5$，双双入堆。
- 下一步堆顶弹出更小的 2（$d = 2$），用 2 松弛邻居：$d[3]$ 从 5 缩成 $2 + 1 = 3$，$d[4]$ 更新成 $2 + 4 = 6$！

## 本页要点

每次只让“当前总距离最小”的点出堆去扩散，保证它自己已经被锁到最优！

# 手算跟踪表：4 点图堆优化 Dijkstra 全过程

| 轮次 | 弹出堆顶 (du, u) | 是否有效 | 更新后 d[1..4] | 动作后小根堆内元素 |
|---|---|---|---|---|
| 初态 | — | — | [0, ∞, ∞, ∞] | {(0, 1)} |
| 1 | (0, 1) | 有效 | [0, 2, 5, ∞] | {(2, 2), (5, 3)} |
| 2 | (2, 2) | 有效 | [0, 2, 3, 6] | {(3, 3), (5, 3), (6, 4)} |
| 3 | (3, 3) | 有效 | [0, 2, 3, 4] | {(4, 4), (5, 3), (6, 4)} |
| 4 | (4, 4) | 有效 | [0, 2, 3, 4] | {(5, 3), (6, 4)} |
| 5 | (5, 3)、(6, 4) | 过期跳过 | [0, 2, 3, 4] | {}（堆空，算法终止） |

- 请注意第 5 行：堆里残留的旧值 (5, 3) 和 (6, 4) 弹出时，由于 $5 > d[3] = 3$，被直接跳过！

## 本页要点

同一个点可能带着不同距离多次入堆，只处理第一次出堆（最小那次），其余旧记录全跳过！

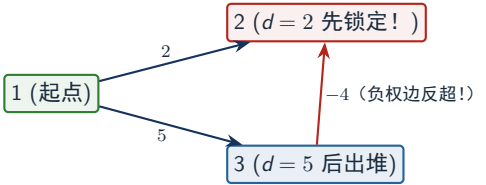
# 正确性论证：为什么非负权下“最近点”出堆就能锁定？

| 论证阶段 | 循环不变量说明 |
| --- | --- |
| 初始化 | 起点 `d[s] = 0`，其余点为 $\infty$；此时所有“已出堆并处理过”的点集为空，显然都是最短路。 |
| 保持 | 每次选当前未处理点中 `d[u]` 最小的 $u$。若存在另一条绕道其他未处理点 $x$ 再到 $u$ 的路，因为 $d[x] \geq d[u]$ 且所有后续边权 $\geq 0$，绕路总长必 $\geq d[u]$，绝不可能反超！ |
| 终止 | 当堆空时，所有从起点可达的点都已处理完毕，`d[]` 即最终单源最短路。 |

## 本页要点

贪心封盘成立的唯一靠山就是“后续边权 $\geq 0$”，路只会越走越长，不会突然变短！

# 致命反例：为什么 Dijkstra 绝对不能有负权边？




- 起点 1 先松弛出 $d[2] = 2, d[3] = 5$。
- 小根堆弹出最小的 2（以为它锁定了 2）；等后面 3 出堆走 $-4$ 这条边时，算出 $1 \to 3 \to 2$ 总长只有 $5 + (-4) = 1 < 2$！
- 若存在负环或复杂负边网，点会反复出堆破坏贪心单调性，甚至在有负环时死循环！

## 本页要点

铁律：只要图里哪怕只有一条负数边权，就绝不能用 Dijkstra，立刻换 Bellman-Ford / SPFA！

# 堆优化 Dijkstra 标准模板（12 行直接背会）

```
const long long INF = LLONG_MAX / 4;
using P = pair<long long, int>;
vector<long long> d(n + 1, INF);
priority_queue<P, vector<P>, greater<>> pq; // 小根堆：最小距离在顶
d[s] = 0; pq.push({0, s});
while (!pq.empty()) {
    auto [du, u] = pq.top(); pq.pop();
    if (du > d[u]) continue;                // 懒删除：跳过已过期的旧堆顶！
    for (auto [v, w] : g[u]) if (d[u] + w < d[v]) {
        d[v] = d[u] + w; pq.push({d[v], v});
    }
}
```

## 本页要点

记住第 8 行 `if (du > d[u]) continue;`，它是堆优 Dijkstra 保持 $O(m \log n)$ 的命门！

# 为什么这样写？三个关键设计追问

1. 为什么需要第 **8** 行的“懒删除” `if (du > d[u]) continue;`？
   C++ 的 `priority_queue` 不支持直接修改堆中旧元素。每次点 $v$ 被刷新更小值时，我们直接把新值再塞进堆里。等旧的、较大的 `(du, u)` 浮上来时，发现它比最新的 `d[u]` 大，直接丢掉不扫邻接表！如果漏写这行，一个点被松弛 $k$ 次就会重扫 $k$ 次出边，菊花图直接卡成 $O(nm)$ **TLE**！
2. 为什么 `INF` 取 `LLONG_MAX / 4` 而不是 `INT_MAX`？
   因为边权之和很容易超过 $2 \times 10^9$（爆 `int`）；且写 `d[u] + w` 时如果 `d[u] = LLONG_MAX`，加个正数会溢出成负数！除以 $4$ 最安全。
3. 为什么不用单独开 `vis[u]` 数组？
   `if (du > d[u]) continue;` 已经把所有过期元素全拦住了，完全等效于“每个点只处理一次”。（若图有 $0$ 权边且多次推入相等距离，也可写 `du != d[u]` 或加 `vis[u]`，效果更严密！）

## 本页要点

懒删除靠 `du > d[u]`（或加 `vis[u]`），防溢出靠 `long long` 与 `LLONG_MAX / 4`！

# 上机验证：3 组测试数据与反事实追问

| 测试场景 | 构造输入（起点 1） | 预期 d[1..n] | 检验点 |
|---|---|---|---|
| 普通正权图 | $n=4, m=5$（前页例图） | 0，2，3，4 | 松弛反超直达边 |
| 不可达孤点 | $n=3, m=1$：1 2 5 | 0，5，INF | 点 3 保持初始无穷大 |
| 重边与自环 | $n=2, m=3$：1 2 10，1 2 3，1 1 5 | 0，3 | 自动取最小重边，不被自环误导 |

## 反事实追问

1. 如果把 greater<> 忘写了，变成默认的大根堆，程序还能算出正确答案吗？复杂度会怎样？
2. 如果输入里出现 d[s] 到终点距离是 $3 \times 10^9$，但你把 d[] 开成了 int，会变成什么现象？

## 本页要点

忘写 greater<> 会让最远的点先出堆导致一个点反复进出退化成指数级 TLE！

# 易错对照：Dijkstra 阶段的 4 个高频报错

| 错误类型 | 错误写法 | 正确做法与原因 |
|---|---|---|
| **TLE** 漏懒删除 | 弹出 (du, u) 后直接扫邻接表 | 必须先判 `if (du > d[u]) continue;` 跳过过期旧值 |
| **WA** 负权强上 | 含负边图仍跑小根堆 Dijkstra | 违背贪心单调性，含负边换 Bellman-Ford / SPFA |
| **WA** 距离爆 `int` | `vector<int> d(n+1, 1e9);` | 累计路径超 $2 \times 10^9$ 溢出变负，改用 `long long` |
| **TLE** 忘写 `greater<>` | `priority_queue<pair<long long,int>>` | 默认是大根堆，必须传 `greater<>` 才是小根堆 |

## 本页要点

Dijkstra 四查：正权？小根堆？懒删除？`long long`？

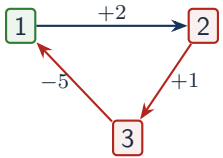
# 引入：遇到负权边怎么办？

- 现实问题中，有些边权可能是负数：例如过某条路可以捡到金币（费用减少），或者穿过时空虫洞回到过去。
- 我们刚证明过 Dijkstra 碰到负权边会翻车。既然不能“走一步锁死一个点”，那就改用老实办法：反复松弛所有边，直到没有任何距离还能变短为止！
- 这就是 **Bellman-Ford** 算法的思路：一共进行 $n-1$ 轮扫描，每轮把全图 $m$ 条边全部尝试松弛一遍。

## 本页要点

Dijkstra 是“贪心一步锁定”，Bellman-Ford 是“多轮耐心松弛”，天然不怕负权边！

# 为什么最多 $n-1$ 轮就够了？什么是负权环？




- 如果图里没有负环，任意一条最短路最多经过 $n$ 个点、即至多 $n-1$ 条边（重复走点只会白白绕远）。每轮松弛至少多确定 $1$ 条边，因此 $n-1$ 轮必定全部收敛！
- 如果图里存在负权环（负环）：像上图绕一圈总权值为 $2+1-5=-2<0$，每绕一圈路径就减 $2$，距离可以无限变小，根本不存在“最短路”！
- 反过来想：如果第 $n$ 轮还能松弛成功，就说明图里一定藏着负环！

## 本页要点

没有负环最多走 $n-1$ 条边；一旦能松弛到第 $n$ 条边，必有负环！

# Bellman-Ford 为何慢？SPFA 队列松弛怎么提速

- **Bellman-Ford** 缺点：每轮都要把全部 $m$ 条边傻傻扫一遍，哪怕 $99\%$ 的点距离根本没变。复杂度高达 $O(nm)$，在 $n = 5000, m = 10^4$ 时要 $5 \times 10^7$ 次，容易超时。
- **SPFA** 核心想法（大白话）：只有“刚刚变短了的点”，它的出边才有可能把邻居拉短！
- 具体做法：开一个普通队列 `queue<int> q` 和标记数组 `inq[]`（记录是否在队列里，防止同一个点反复进队）：
  1. 每次从队头取出点 $u$（置 `inq[u] = 0`），用它的出边去松弛邻居 $v$；
  2. 只要 `dist[u] + w < dist[v]` 成立，更新 `dist[v]`，若 `!inq[v]` 就把 $v$ 塞进队尾！

## 本页要点

SPFA $=$ 用队列记住“刚被更新过的点”，只让这些点去扩散更新别人！

# 手算跟踪表：含负权边（无负环）的 SPFA 全过程

| 轮次 | 出队 $u$ | 尝试松弛出边 | dist[1..3] | 队列 q |
|---|---|---|---|---|
| 初态 | — | 起点 1 入队 | [0, ∞, ∞] | [1] |
| 1 | 1 | $1\to2(2), 1\to3(5)$ 变短 | [0, 2, 5] | [2, 3] |
| 2 | 2 | 无出边（图：$1\to2(2), 1\to3(5), 3\to2(-4)$） | [0, 2, 5] | [3] |
| 3 | 3 | $3\to2(-4)$：$5-4=1<2$，刷新 $d[2]=1$ | [0, 1, 5] | [2] |
| 4 | 2 | 无出边，弹出结束（终止） | [0, 1, 5] | []（空） |

- 刚才把 Dijkstra 坑惨的负权边 $3\to2(-4)$，在 SPFA 里因为点 2 被改小后允许重新入队，轻松算出 $d[2]=1$！

## 本页要点

SPFA 允许节点被改短后多次重新入队，因此能正确处理任意合法的负权边！

# SPFA 怎样判负环？记住 cnt[v] >= n！

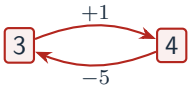
- 开一个计数数组 cnt[v]，记录到达点 $v$ 的当前最短路一共经历了多少条边（或点 $v$ 被成功松弛了多少次）。
- 每当通过边 $u \to v$ 成功松弛 dist[v] 时：
  - 写法 A（边数传递）：cnt[v] = cnt[u] + 1;
  - 写法 B（松弛次数累加）：++cnt[v];
- 由抽屉原理：全图只有 $n$ 个点，如果某个点最短路包含的边数（或被松弛次数）达到 >= n，根据鸽巢原理必然至少有一个点被走了两次，图里一定存在负环！

## 本页要点

两种判法均成立：cnt[v] >= n 时立刻返回“存在负环”，不用再继续绕圈！

# 致命盲区：只把起点 1 入队为什么会漏掉孤立负环？




- 若题目问“整张图里是否存在负环?”（比如《虫洞》或差分约束系统）：
- 上图中 $3 \leftrightarrow 4$ 显然是个总权 $-4$ 的负环，但它和起点 $1$ 不连通！如果只写 `q.push(1)`，队列走完 $1 \to 2$ 就空了，直接漏判输出 `NO`（**WA**）！
- 标准解法：初始把所有点 $1 \sim n$ 全部设 `dist[i] = 0` 并同时塞进队列（相当于加了一个虚拟超级源点 $0$ 向每个点连了权为 $0$ 的边）！

## 本页要点

只要是查全图负环，务必“全点设 0 同时入队”，绝不能只塞一个 1 号点！

# SPFA 判全图负环标准模板（12 行）

```
vector<long long> dist(n + 1, 0);
vector<int> cnt(n + 1, 0); vector<char> inq(n + 1, 1);
queue<int> q;
for (int i = 1; i <= n; ++i) q.push(i);    // 全点同时入队防孤立负环漏判！
bool has_neg = false;
while (!q.empty() && !has_neg) {
    int u = q.front(); q.pop(); inq[u] = 0;
    for (auto [v, w] : g[u]) if (dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        if (++cnt[v] >= n) { has_neg = true; break; } // 抓到负环！
        if (!inq[v]) { inq[v] = 1; q.push(v); }
    }
}
```

## 本页要点

判负环 `dist` 初值全为 $0$；求从 $s$ 出发的单源最短路则只把 `s` 入队且初值设 `INF`！

# 正确性论证：为什么“全点初值 0 + 松弛 $\geq n$ 次”能抓全图负环？

| 论证阶段 | 核心不变量说明 |
| --- | --- |
| 初始化 | 所有点初始 `dist[i] = 0` 且都在队中，等价于虚拟点 $0$ 连向 $1 \ldots n$ 权为 $0$ 的边。 |
| 保持 | 若某个环总权值 $< 0$，环上至少存在一条边能不断让 `dist[v]` 严格变小（从 $0$ 降成更小的负数），因此它永远不会停下来。 |
| 终止 | 无负环时所有点松弛次数 $< n$，队列必清空；有负环时必有环上一点的 `cnt[v]` 率先达到 $n$ 而终止！ |

## 本页要点

正权环或零权环绝对不会让 `dist[v]` 严格变小（因为有 <），只有总权 $< 0$ 的负环才会无限绕！

# 警告：为什么正权图千万不要写 SPFA？

- 很多同学觉得“SPFA 既能算正权又能算负权，那我考场全写 SPFA 不就行了？”——大错特错！
- SPFA 的均摊复杂度虽然像 $O(km)$，但在网格图、菊花图、或者出题人特意构造的“卡 SPFA 数据”下，每个点会被反复松弛入队，最坏退化成 $O(nm)$！
- 当 $n = 10^5, m = 2 \times 10^5$ 时，$O(nm) \approx 2 \times 10^{10}$ 必 **TLE**（著名梗：“关于 SPFA，它死了”）。
- 考场铁律：只要确定边权全是非负数，无脑写堆优化 **Dijkstra**；只有存在负权边、判负环、或求解差分约束时，才请出 SPFA！

## 本页要点

正权图写 Dijkstra，负权/判负环写 SPFA；永远不要在正权大图里赌 SPFA 不被卡！

# 上机验证：SPFA 3 组边界测试与反事实追问

| 测试场景 | 构造图（$n=3$） | 预期判负环 | 检验点 |
|---|---|---|---|
| 纯负边无负环 | $1\to 2(-5), 2\to 3(-3)$ | NO | 负边但无环，能正常结束 |
| 零权环 | $1\to 2(2), 2\to 1(-2)$ | NO | 总权为 0，严格小于 < 不会死循环 |
| 孤立负环 | 边只有 $2\to 3(1), 3\to 2(-4)$ | YES | 全点初始入队能抓出不与 1 连通的负环 |

## 反事实追问

1. 如果把 `if (dist[u] + w < dist[v])` 误写成 `<=`，遇到零权环会发生什么？（提示：零权环也会无限松弛导致误判有负环 **WA**！）
2. 如果出队时忘了写 `inq[u] = 0;`，对结果有什么影响？（提示：点 $u$ 以后再也没法重新入队，含负权边时直接算错 **WA**！）

## 本页要点

记住两点：松弛条件必须是严格 <（防零权环），出队必须立刻置 `inq[u] = 0`！

# 易错对照：Bellman-Ford / SPFA 高频坑

| 错误类型 | 错误写法 | 正确做法与原因 |
|---|---|---|
| **WA** 只入队 1 号 | 判全图负环只写 `q.push(1)` | 孤立负环搜不到，必须全点初始入队 |
| **WA** 忘了置 `inq[u]=0` | 弹出 $u$ 后不清除队列标记 | $u$ 之后被改小也进不了队，导致松弛中断 |
| **WA** 写成了 `<=` | `if (dist[u] + w <= dist[v])` | 零权边/零权环被反复松弛，误判为存在负环 |
| **TLE** 正权图用 SPFA | 见最短路就写 SPFA | 正权稠密/网格图会被卡到 $O(nm)$ 超时 |

## 本页要点

SPFA 四防：全点入队、出队清标记、严格小于号、正权绝不用！

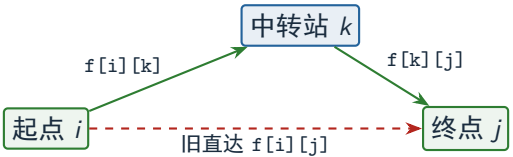
# 引入：想知道任意两点间的最短路怎么办？

- 如果题目不固定起点，而是有 $10^5$ 次询问每次问 “$u$ 到 $v$ 最短多少”，或者需要一张任意两点的距离表：
- 当点数 $n \le 400$ 时，不需要写 $n$ 遍堆优化 Dijkstra！**Floyd-Warshall** 算法只需三层 ‘for‘ 循环、总共 5 行核心代码就能算出所有点对 $(i, j)$ 的最短路，复杂度 $O(n^3)$（$400^3 = 6.4 \times 10^7$）。
- 大白话核心：每次多允许一个点 $k$ 当 “中转站”，看看 $i \to k \to j$ 能不能比原来的 $i \to j$ 更短！

## 本页要点

点数 $n \le 400$ 求全源最短路，首选 5 行代码搞定的 Floyd-Warshall 算法！

# 图解中转思想：为什么最外层循环必须是 k？




- 状态 `f[i][j]` 的真实含义（二维压缩掉了第一维 $k$）：只允许经过编号 $1 \sim k$ 的点作中转时，$i$ 到 $j$ 的最短路。
- 当我们用 `f[i][k] + f[k][j]` 更新 `f[i][j]` 时，必须保证 `f[i][k]` 和 `f[k][j]` 已经算完了前 $k-1$ 个中转点的最优解！
- 如果把 `k` 写到最内层（`i`，`j`，`k`），计算 `f[1][4]` 时用到了 `f[1][2]`，但此时经过点 3 的中转 `1->3->2` 还没算出来，路径拼不起来导致 **WA**！

## 本页要点

铁律：Floyd 三重循环必须写 `for(k) for(i) for(j)`，中转点 `k` 永远在最外层！

# 手算跟踪表：依次开放中转点 $k = 1, 2$ 的距离表

| 阶段 | 允许中转点 | 关键更新（假设有边 $1 \to 3(10), 1 \to 2(2), 2 \to 3(3), 3 \to 1(1)$） |
|---|---|---|
| 初态 | 不允许任何中转 | $f[1][3] = 10, f[1][2] = 2, f[2][3] = 3, f[3][1] = 1$，其余 $\infty$ |
| $k = 1$ | 只允许经过点 $1$ | 发现 $3 \to 1 \to 2$：$f[3][2] = \min(\infty, 1 + 2) = 3$！ |
| $k = 2$ | 允许经过 $\{1, 2\}$ | 发现 $1 \to 2 \to 3$：$f[1][3] = \min(10, 2 + 3) = 5$！ |
| $k = 3$ | 允许经过 $\{1, 2, 3\}$ | 检查所有 $i \to 3 \to j$，无更短路径，算法终止 |

- 注意看 $k = 2$ 时，`f[3][2]` 已经在 $k = 1$ 阶段算成 $3$ 了（经过了 $1$），所以 $k = 2$ 还能继续拼出更长的中转路径！

## 本页要点

每多开放一个中转点 $k$，所有点对都有机会借道 $k$ 拼出更短的路！

# 正确性论证：Floyd 为什么能省略掉第一维 $k$？

| 论证阶段 | 核心不变量说明 |
| --- | --- |
| 初始化 | 对角线 `f[i][i] = 0`，直接连边取最小边权，其余为 `INF`（不允许任何中转）。 |
| 保持 | 在第 $k$ 轮中，由于我们正试图用 $k$ 当中转站，所以到达 $k$ 和从 $k$ 出发的子路径不可能再把 $k$ 当中转点；此时 `f[i][k]` 和 `f[k][j]` 只包含 $1 \sim k-1$ 的中转，是否被本轮覆盖顺序不影响正确性。 |
| 终止 | 当 $k = n$ 执行完，允许所有点 $\{1 \dots n\}$ 中转，全图任意两点最短路全部求完！ |

## 本页要点

因为第 $k$ 轮绝不会用 $k$ 自己去更新 $f[i][k]$ 和 $f[k][j]$，所以二维原地覆盖完全安全！

# Floyd-Warshall 标准模板与初始化

```cpp
const long long INF = LLONG_MAX / 4;
vector<vector<long long>> f(n + 1, vector<long long>(n + 1, INF));
for (int i = 1; i <= n; ++i) f[i][i] = 0;   // 自己到自己距离必须为 0
for (int i = 0, u, v, w; i < m; ++i) {
    cin >> u >> v >> w;
    f[u][v] = min(f[u][v], (long long)w);   // 防重边！一定要取 min
}
for (int k = 1; k <= n; ++k)                // 中转点 k 放最外层！
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            f[i][j] = min(f[i][j], f[i][k] + f[k][j]);
```

## 本页要点

第 6 行读入边时一定要写 `min(f[u][v], w)`，直接赋值 = 碰到重边会覆盖掉短边！

# 上机验证与易错对照：Floyd 的 3 个陷阱

| 测试 / 错误点 | 常见错误写法 | 正确做法与后果 |
|---|---|---|
| **WA** 循环顺序错 | 把 `k` 放最内层 `i-j-k` | 漏掉多点中转拼接，答案偏大 |
| **WA** 读入不取 `min` | 读入直接写 `f[u][v] = w` | 重边时后读的较长边会覆盖掉短边 |
| **WA** 初值太大溢出 | `INF` 设为 `LLONG_MAX` | `f[i][k] + f[k][j]` 相加直接溢出变负数！ |

## 反事实追问

1. 如果图中有负权边但没有负环，Floyd 还能算对吗？（能！因为 Floyd 本质是 DP 而不是贪心）
2. 跑完 Floyd 后如何顺便检测有没有负环？（只需看对角线有没有出现 `f[i][i] < 0`！）

## 本页要点

记住：Floyd 也能处理无负环的负权边，且 `f[i][i] < 0` 直接等价于点 $i$ 在负环上！

# 生活引入：怎么用图论解一组不等式？

## 手算小问题

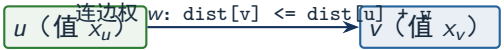
三个同学比身高：$x_1, x_2, x_3$ 满足：(1) 小明比小红高最多 $5$ cm：$x_1 - x_2 \le 5$；(2) 小红比小华高最多 $3$ cm：$x_2 - x_3 \le 3$；(3) 小华比小明高最多 $-10$ cm（即小华至少矮 $10$ cm）：$x_3 - x_1 \le -10$。这三条要求能同时满足吗？

- 把三条不等式直接相加：$(x_1 - x_2) + (x_2 - x_3) + (x_3 - x_1) \le 5 + 3 - 10 = -2$，得出 $0 \le -2$，互相矛盾、无解！
- 当变量有 $5000$ 个、不等式有 $10^4$ 条时，手动消元根本算不过来，这就要靠差分约束系统！

## 本页要点

形如 $x_v - x_u \le w$ 的不等式组，叫做“差分约束系统”，它能直接变成图上的最短路！

# 核心等价转化：$x_v - x_u \le w \iff u \xrightarrow{w} v$




- 把不等式 $x_v - x_u \le w$ 移项，得到：$x_v \le x_u + w$。
- 看看这长得像什么？这正是图论最短路里松弛结束后必然满足的三角形不等式：`dist[v] <= dist[u] + w`！
- 因此，每一条不等式 $x_v \le x_u + w$，就对应图中一条从 $u$ 指向 $v$、边权为 $w$ 的有向边！
- 刚才三条矛盾不等式加起来权和为 $-2 < 0$，在图上恰好就是一个负权环！

## 本页要点

口诀：看到 $x_v - x_u \le w$，把减数 $u$ 当起点、被减数 $v$ 当终点，连有向边 $u \to v$ 权 $w$！

# 不等式变形大全：遇到 $\geq$、$=$、$<$ 怎么统一？

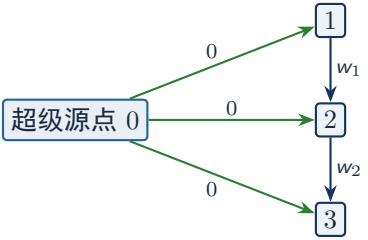
| 原不等式写法 | 等价变形为 $\leq$ 型 | 对应连边（起点 $\to$ 终点，边权） |
|---|---|---|
| $x_v - x_u \leq w$ | $x_v \leq x_u + w$ | $u \to v$，权 $w$ |
| $x_v - x_u \geq w$ | $x_u - x_v \leq -w \iff x_u \leq x_v + (-w)$ | $v \to u$，权 $-w$ |
| $x_v - x_u = w$ | 拆成 $x_v - x_u \leq w$ 且 $x_v - x_u \geq w$ | 连两条边：$u \xrightarrow{w} v$ 与 $v \xrightarrow{-w} u$ |
| $x_v - x_u < w$（整数解） | $x_v - x_u \leq w - 1$ | $u \to v$，权 $w-1$ |

- 如果不等式只出现单变量限制，比如 $x_i \leq c$，可以引入 $0$ 号虚拟点（固定 $x_0 = 0$），写成 $x_i - x_0 \leq c$，连边 $0 \to i$ 权 $c$！

## 本页要点

无论题面给 $\geq$、$=$ 还是严格 $<$（整数），全部先统一化成 $x_v - x_u \leq w$ 再连边！

# 建模图解：为什么必须加“超级源点 0”？




- 新建虚拟点 $0$，向每个变量点 $1 \sim n$ 连一条权为 $0$ 的有向边（对应条件 $x_i \le x_0 + 0$，即所有变量 $\le 0$）。
- 这样从 $0$ 出发不仅能到达全图所有连通块（不会漏掉孤立不等式组里的负环），跑完 SPFA 后得到的 `dist[1..n]` 就是一组合法实数/整数解！

## 本页要点

加了超级源点 $0 \to i(0)$，等效于“全点设 0 同时入队”，既能判无解又能求出一组解！

# 手算跟踪表：3 变量不等式组的 SPFA 求解与判无解

| 不等式组 | 建边后从 0 跑 SPFA 过程 | dist[1..3] | 结论与验算 |
|---|---|---|---|
| $x_2 - x_1 \le 3$ | $1 \xrightarrow{3} 2$，$2 \xrightarrow{-1} 3$，$3 \xrightarrow{2} 1$ | [0, 0, -1] | 环总权 $4 \ge 0$，无负环 |
| $x_3 - x_2 \le -1$ | $2 \to 3$ 刷成 $d[3] = -1$ | （收敛终止） | 验算 3 式全部成立 |
| $x_1 - x_3 \le 2$ | 队列清空，无点松弛 $\ge n$ 次 | 返回 true | $0 \le 3, -1 \le -1, 1 \le 2$ |
| 第 3 式改为 | 环总权 $3 - 1 - 4 = -2 < 0$ | cnt[] 超限 | 抓出负环 $\to$ 无解！ |
| $x_1 - x_3 \le -4$ | $d$ 不断减小触发 >= n | （终止） | 3 式相加得 $0 \le -2$ |

## 本页要点

无负环时收敛的 dist[] 就是一组可行解；有负环就一定无解！

# 考场必考辨析：什么时候求“最大解”？什么时候求“最小解”？

| 题目要求 | 不等式统一化为 | 跑哪种算法 | 大白话物理直觉 |
|---|---|---|---|
| 求变量最大值 | $x_v \le x_u + w$ | 最短路 | 所有上界紧逼下来，求最高能顶到多高 |
| 求变量最小值 | $x_v \ge x_u + w$ | 最长路 | 所有下界往上托起，求最低不能低于多少 |

- 如果题目既要求最小总和，又想跑最短路？其实“求最小解 $\iff$ 跑最长路”和“给所有变量取反后跑最短路”是一样的！
- 写最长路只需要把 SPFA 松弛改成：`if (dist[u] + w > dist[v]) { dist[v] = dist[u] + w; ... }`，无解对应出现正环（同样用 `cnt[v] >= n` 抓出来）！

## 本页要点

记忆口诀：不等式全用 $\le$ 跑最短路（天花板压下来 $\to$ 最大解）；全用 $\ge$ 跑最长路（地板顶上去 $\to$ 最小解）！

# 差分约束 SPFA 模板代码（以全化 $\leq$ 求最大解为例）

```
// 已按 x_v - x_u <= w 连边 u -> v 权 w; 超级源点 0 连向 1..n 权 0
vector<long long> dist(n + 1, LLONG_MAX / 4);
vector<int> cnt(n + 1, 0); vector<char> inq(n + 1, 0);
queue<int> q; dist[0] = 0; q.push(0); inq[0] = 1;
while (!q.empty()) {
    int u = q.front(); q.pop(); inq[u] = 0;
    for (auto [v, w] : g[u]) if (dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        if (++cnt[v] >= n + 1) { cout << "NO\n"; return 0; }
        if (!inq[v]) { inq[v] = 1; q.push(v); }
    }
}
```

## 本页要点

加了超级源点 0 后总点数是 n + 1，所以判负环的阈值写 >= n + 1！

# 上机验证与易错对照：差分约束的 3 个坑

| 常见失误 | 错误写法 | 正确做法与后果 |
| --- | --- | --- |
| **WA** 边方向连反 | 看见 $x_v - x_u \le w$ 连 $v \to u$ | 移项得 $x_v \le x_u + w$，必须从 $u$ 指向 $v$ |
| **WA** 漏加超级源 | 不连 $0 \to i$ 只从 $1$ 号点开跑 | 不连通负环漏判，且孤立变量仍为 `INF` |
| **WA** 极值求反 | 求最小总和却全化 $\le$ 跑最短路 | 跑出的是最大解而非最小解，应化 $\ge$ 跑最长路 |

## 反事实追问

在超级源点 $0 \to i(0)$ 的图里，若变量要求必须 $\ge 0$，可加边 $0 \to i$ 权 $0$（跑最长路）或 $i \to 0$ 权 $0$（跑最短路）！

## 本页要点

连边方向看移项：`dist[终点] <= dist[起点] + 边权`，永远不会连反！

# 实用拓展：Floyd 如何记录并输出最短路径？

- 开一个数组 `mid[i][j]` 记录从 $i$ 到 $j$ 的中转点（初始全为 0）。
- 当发生更新 `f[i][k] + f[k][j] < f[i][j]` 时，记下 `mid[i][j] = k`；输出时递归先打左半段 `(i, k)`，打印 `k`，再打右半段 `(k, j)`！

```
void print_path(int i, int j) {
    int k = mid[i][j];
    if (k == 0) return;                    // 没有中转点，i 直接连向 j
    print_path(i, k);                      // 先输出前半段经过的中转点
    cout << k << ' ';
    print_path(k, j);                      // 再输出后半段经过的中转点
}
```

## 本页要点

记住只打印中转点 `k`，起点和终点由外层单独输出，防止中间点重复打印！

# 建模实战：区间和不等式怎样变成差分约束？

- 很多题目（如练习《出纳员问题 1510》）给的不是两个变量差，而是“某一段连续小时的总人数 $\geq R(i)$”。
- 前缀和妙招：令 $s[i]$ 表示前 $i$ 个小时一共雇佣的人数（$s[0]=0$）。
- 那么任意连续 $8$ 小时工作人数就能直接写成两个变量的差：$s[i]-s[i-8]\geq R(i)$！
- 这样一来，所有关于“区间总和”的约束全都变成了两两变量之差，直接套差分约束模板即可！

## 本页要点

看到“连续区间和满足 $\geq$ 或 $\leq$ 某数”，立刻设前缀和 $s[i]$ 转化成差分约束！

# 综合快问快答：最短路与差分约束考场排雷

## 请先口头作答，下一页核对答案

1. 一张无负环的图里有两条负权边，$n = 10^4, m = 2 \times 10^4$，该选 Dijkstra、SPFA 还是 Floyd？
2. 求全图任意两点最短路，如果只有 $n = 300$ 个点，写完 Floyd 后忘了把 `f[i][i]` 置 $0$ 会发生什么？
3. 差分约束里出现要求“$x_3 - x_1 = 5$（必须恰好等于 $5$）”，该连几条边？

## 本页要点

先想 10 秒再翻页，考场选型错误一题就是 100 分的差距！

# 综合快问快答：答案与核心解析

1. 选 **SPFA**！有负权边不能用 Dijkstra，$n = 10^4$ 用 $O(n^3)$ Floyd 会超时。
2. `f[i][i]` 保留了 `INF`，导致从 $i$ 出发绕回 $i$ 的值算错，甚至影响路径拼接。
3. 连两条相反的边！ $x_3 - x_1 = 5$ 等价于 $x_3 - x_1 \le 5$（连 $1 \xrightarrow{5} 3$）和 $x_3 - x_1 \ge 5 \iff x_1 - x_3 \le -5$（连 $3 \xrightarrow{-5} 1$）。

## 本页要点

有负边用 SPFA、Floyd 必设 `f[i][i]=0`、等号拆成一正一反两条边！

# 例 1 | [基础] 香甜的黄油（1345）： 先读懂要求

## 大白话题意

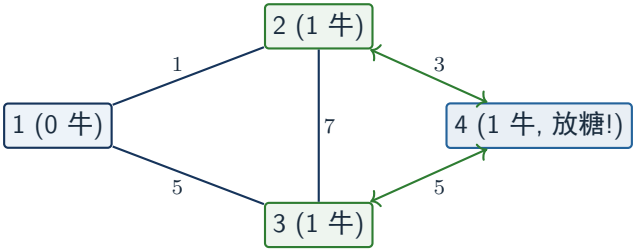
有 $N \le 500$ 头牛分布在 $P \le 800$ 个牧场（一个牧场可能有多头牛），牧场间有 $C \le 1450$ 条双向道路，边权 $D \in [1, 255]$。选一个牧场放糖，使所有牛走到该牧场的路程总和最小。

- 输入：第 1 行 $N, P, C$；接下来 $N$ 行给出每头牛所在牧场号；再接下来 $C$ 行给出双向边 $u, v, w$。
- 处理：找使 $\sum \text{cows}[i] \times \text{dist}(i, s)$ 最小的牧场 $s$。
- 输出：一行一个整数，最小总路程。
- 原题样例：$N = 3, P = 4, C = 5$，牛在 $2, 3, 4$，边 $(1,2,1), (1,3,5), (2,3,7), (2,4,3), (3,4,5)$ $\rightarrow$ 输出 8。

## 本页要点

无向图里“牛走到 $s$”和“从 $s$ 走到牛”距离完全相等，转化为枚举放糖点 $s$ 求单源最短路！

# 例 1 | 怎么想到的与建模 TikZ 图




- 为什么不用 Floyd？$P = 800$ 时 $P^3 = 5.12 \times 10^8$，很容易超时！
- 而边数 $C \le 1450$ 且全为正权，跑 $P$ 次堆优化 Dijkstra 的总操作数是 $P \times C \log P \approx 800 \times 1450 \times 10 \approx 1.1 \times 10^7$，轻松秒过！
- 统计每牧场牛数 `cows[i]`，每次算完 `d[]` 只需 $O(P)$ 求加权和。

## 本页要点

稀疏图 $P = 800$ 用“跑 $P$ 次堆优 Dijkstra”比 $O(P^3)$ Floyd 快了近 $50$ 倍！

# 例 1 | 样例手推：以 4 号牧场为放糖点

| 轮次 | 弹出堆顶 | 松弛动作 | d[1..4] | 堆内状态 |
|---|---|---|---|---|
| 初态 | — | 源点 $s=4$ 入堆 | $[\infty,\infty,\infty,0]$ | {(0,4)} |
| 1 | (0, 4) | $4\to 2(3)$、$4\to 3(5)$ | $[\infty,3,5,0]$ | {(3,2), (5,3)} |
| 2 | (3, 2) | $2\to 1(1)$ 更新 $d[1]=4$ | [4, 3, 5, 0] | {(4,1), (5,3)} |
| 3 | (4, 1) | $1\to 3(5)$ 算出 $9>5$ 不更新 | [4, 3, 5, 0] | {(5,3)} |
| 4 | (5, 3) | 无更短路径，堆空终止 | [4, 3, 5, 0] | {} |

- 计算总路程：$\texttt{tot}=\text{cows}[2]\cdot d[2]+\text{cows}[3]\cdot d[3]+\text{cows}[4]\cdot d[4]=1\times 3+1\times 5+1\times 0=\mathbf{8}$!
- 顺便看看如果放 1 号点（虽然 1 号没牛）：$d=[0,1,5,4]$，总和 $1+5+4=10>8$。

## 本页要点

即使某牧场本身没有牛（如 1 号），也必须把它作为放糖候选点之一去枚举！

# 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. 读牛位置并累加桶 | `cows[1..p]` | 同一个牧场有多头牛时是不是写了 `++cows[x]`？ |
| 2. 读 $c$ 条双向边建邻接表 | `g[1..p]` | 双向边有没有正反都压入？ |
| 3. 枚举 $s = 1 \dots p$ 跑 Dijkstra | 每次算出 `d[1..p]` | 堆里有没有写 `if (du > d[u]) continue;`？ |
| 4. 累加 `cows[i]*d[i]` 取最小 | 最终答案 `best` | `best` 初值是否开成 `LLONG_MAX`？ |

## 相对基础模板改了哪里

1. 把单次 Dijkstra 放进 `for (int s = 1; s <= p; ++s)` 循环里枚举每个放糖点；
2. 新增桶数组 `cows[i]`，把原本需要 $N$ 次查距离变成 $P$ 次乘法累加。

## 本页要点

用 `cows[i]` 桶计数能把单源枚举后的统计复杂度稳稳控在 $O(P)$！

# 例 1 | 核心代码（1/2）：读入牛数桶与建双向图

```
int n, p, c; if (!(cin >> n >> p >> c)) return 0;
vector<int> cows(p + 1, 0);
for (int i = 0, x; i < n; ++i) {
    cin >> x;
    ++cows[x];                                  // 统计每个牧场上有多少头牛
}
vector<vector<pair<int, int>>> g(p + 1);
for (int i = 0, u, v, w; i < c; ++i) {
    cin >> u >> v >> w;
    g[u].push_back({v, w});
    g[v].push_back({u, w});                     // 牧场道路是双向的
}
long long best = LLONG_MAX;
```

## 本页要点

代码与 code/dijkstra_butter.cpp 一致；记得 best 必须开 long long！

# 例 1 | 核心代码（2/2）：枚举放糖点跑 Dijkstra

```
for (int s = 1; s <= p; ++s) {
    vector<long long> d(p + 1, LLONG_MAX / 4);
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<>>
    pq;
    d[s] = 0; pq.push({0, s});
    while (!pq.empty()) {
        auto [du, u] = pq.top(); pq.pop();
        if (du > d[u]) continue;            // 懒删除跳过过期旧堆顶
        for (auto [v, w] : g[u]) if (d[u] + w < d[v])
            { d[v] = d[u] + w; pq.push({d[v], v}); }
    }
    long long tot = 0;
    for (int i = 1; i <= p; ++i) tot += 1LL * cows[i] * d[i];
    best = min(best, tot);
}
```

# 例 1 | 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 |
|---|---|---|---|
| 多牛同牧场 | $N=3, P=2$：牛在 1,1,2，边 1 2 5 | 5 | 放 1 号最优，测 ++cows[x] |
| 中心无牛牧场 | 牛在 2，3，边 1-2(1),1-3(1),2-3(10) | 2 | 放没牛的 1 号比放 2,3 近 |
| 只枚举有牛点 | 同上 | 10 (错) | 绝不能只从有牛牧场开跑！ |

## 本页要点

记住：枚举放糖点必须是 $1 \sim P$ 所有牧场，不能只枚举有牛的牧场！

# 例 2 | [提高] 虫洞 Wormholes（1507）： 先读懂要求

## 大白话题意

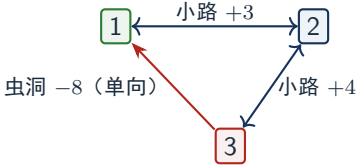
共有 $F \le 5$ 组农场地图。每张图有 $N \le 500$ 块田、$M \le 2500$ 条双向正权小路（用时 $T$ 秒）和 $W \le 200$ 条单向虫洞（可使时间倒流 $T$ 秒，即边权 $-T$）。问能否找到一条路径，在出发时刻之前回到出发点？

- 输入：第 $1$ 行 $F$；每组第 $1$ 行 $N, M, W$；随后 $M$ 行双向边 $S, E, T$；再 $W$ 行单向虫洞 $S, E, T$。
- 处理：判断每组图里是否存在总权值 $< 0$ 的有向环。
- 输出：每组一行，能回到过去输出 `YES`，否则输出 `NO`。
- 原题样例：第 $1$ 组（无负环）输出 `NO`；第 $2$ 组（$1 \leftrightarrow 2(3), 2 \leftrightarrow 3(4), 3 \to 1(-8)$）输出 `YES`。

## 本页要点

“在出发之前回到出发点” $\iff$ 沿有向图走一圈总权和为负数 $\iff$ 判定全图是否存在负环！

# 例 2 | 怎么想到的与建模 TikZ 图




- 把样例第 2 组画出来：走 $1 \to 2 \to 3 \to 1$ 总耗时是 $3 + 4 - 8 = -1 < 0$，这就是一个货真价实的负权环！
- 注意双向小路拆成两条有向边 $u \xrightarrow{c} v$ 和 $v \xrightarrow{c} u$；单向虫洞只连一条 $u \xrightarrow{-c} v$。
- 因为图可能不连通（孤立小块藏着虫洞负环），必须用全点设 0 同时入队的 SPFA 判负环！

## 本页要点

小路双向正权、虫洞单向负权，建完图跑一次全点入队的 SPFA 看 `cnt[v] >= n`！

# 例 2 | 样例手推：样例第 2 组 SPFA 抓负环过程

| 出队点 | 关键松弛边 | dist[1..3] 更新后 | cnt[1..3] | 队列状态 |
|---|---|---|---|---|
| 初态 | — | [0, 0, 0] | [0, 0, 0] | [1, 2, 3] |
| $1, 2$ | 无负边，$3$ 在后面 | [0, 0, 0] | [0, 0, 0] | [3] |
| $3$ | $3 \to 1(-8)$：刷新 $d[1] = -8$ | [-8, 0, 0] | [1, 0, 0] | [1] |
| $1$ | $1 \to 2(+3)$：刷新 $d[2] = -5$ | [-8, -5, 0] | [1, 1, 0] | [2] |
| $2$ | $2 \to 3(+4)$：刷新 $d[3] = -1$ | [-8, -5, -1] | [1, 1, 1] | [3] |
| $\ldots$ | 继续绕圈，$d$ 每圈减 $1$ | 直到 cnt[1] 变成 3 >= n | cnt[1]==3 | 立刻退出输出 **YES** |

## 本页要点

每绕负环一圈，环上各点的 cnt[] 就 $+1$，最多绕 $n$ 圈必有某点撞破 >= n 报警！

# 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. 每组建新邻接表 g(n+1) | 清爽干净的图 | 多组数据之间有没有清理干净？ |
| 2. 读 $m$ 条双向路与 $w$ 条虫洞 | 正权双向 + 负权单向 | 虫洞边权是不是取了 -c？ |
| 3. 全点设 0 入队跑 SPFA | neg 布尔标志 | inq[] 初始全为 1 了吗？ |
| 4. 输出 YES 或 NO | 每组一行结果 | 是否在发现 neg 后立刻 break？ |

## 相对基础模板改了哪里

1. 外层包裹 while (T--) 处理多组数据，邻接表和队列都在循环内局部重新创建，杜绝脏数据残留；
2. 分别用两个循环读入正权双向边和负权单向边。

## 本页要点

多组数据把 g，dist，cnt，inq，q 直接定义在 while (T--) 内部，天然免清空出错！

# 例 2 | 核心代码（1/2）：多组建图与全点入队初始化

```
int T; if (!(cin >> T)) return 0;
while (T--) {
    int n, m, w; cin >> n >> m >> w;
    vector<vector<pair<int, int>>> g(n + 1);
    for (int i = 0, u, v, c; i < m; ++i) {
        cin >> u >> v >> c;
        g[u].push_back({v, c}); g[v].push_back({u, c});
    }
    for (int i = 0, u, v, c; i < w; ++i) {
        cin >> u >> v >> c; g[u].push_back({v, -c}); // 虫洞是单向负权边
    }
    vector<long long> dist(n + 1, 0);
    vector<int> cnt(n + 1, 0); vector<char> inq(n + 1, 1);
```

## 本页要点

注意 `inq` 初始值为 1，与下面把 $1 \sim n$ 全部压进队列保持严格同步！

# 例 2 | 核心代码（2/2）：SPFA 松弛与 cnt[v] >= n 判负环

```
    queue<int> q;
    for (int i = 1; i <= n; ++i) q.push(i); // 全点入队防孤立负环漏判
    bool neg = false;
    while (!q.empty() && !neg) {
        int u = q.front(); q.pop(); inq[u] = 0;
        for (auto [v, cost] : g[u]) if (dist[u] + cost < dist[v]) {
            dist[v] = dist[u] + cost;
            if (++cnt[v] >= n) { neg = true; break; }
            if (!inq[v]) { inq[v] = 1; q.push(v); }
        }
    }
    cout << (neg ? "YES" : "NO") << '\n';
}
```

## 本页要点

代码与 code/spfa_negcycle.cpp 完全一致；发现负环立刻跳出省时间！

# 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 |
|---|---|---|---|
| 孤立负环 | $n=3, m=1(1\leftrightarrow 2,5), w=1(3\to 3,1)$ | YES | 自环 $-1$ 在 3，只入队 1 会漏 |
| 虫洞误建双向 | $n=2, m=0, w=1(1\to 2,5)$ | NO | 双向会变成 $-10$ 环误报 YES |
| 多组第 2 组 | 第 1 组 YES，第 2 组无虫洞 | YES/NO | 全局变量未清空会受污染 |

## 本页要点

记住查三点：虫洞单向、全点入队、多组变量局部重置！

# 四大最短路算法考场两秒选型表

| 算法 | 时间复杂度 | 能否处理负边 | 考场核心用途 |
|---|---|---|---|
| 普通 **BFS** | $O(n+m)$ | 仅限边权全 $=1$ | 无权网格/状态空间最少步数 |
| 堆优 **Dijkstra** | $O(m \log n)$ | 不能！ | 正权稀疏图单源最短路绝对主力 |
| **SPFA / Bellman** | 均摊 $O(km) \sim O(nm)$ | 能，且判负环 | 含负权边、判负环、求解差分约束 |
| **Floyd-Warshall** | $O(n^3)$ | 能（无负环） | $n \leq 400$ 全源任意两点最短路 |

## 本页要点

正权单源选堆优 Dijkstra；负权/判负环/差分约束选 SPFA；小图全源选 Floyd！

# 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 |
| --- | --- | --- |
| 负权图强上 Dijkstra | 含负边权仍跑小根堆 Dijkstra | 出堆锁定导致负边反超无法挽回，含负边换 SPFA |
| 判负环只放起点 1 | 只写 `q.push(1)` 查全图负环 | 图不连通时孤立负环根本搜不到，必须全点初始入队 |
| Floyd `k` 写到内层 | 循环写成 `i - j - k` | 借道中转未提前算好，路径拼不起来导致答案偏大 |
| 差分约束连反方向 | 看见 $x_v - x_u \le w$ 错连 $v \to u$ | 移项得 $x_v \le x_u + w$，必须从 `u` 指向 `v` 权 `w` |

## 本页要点

WA 四大重灾区：负边用 Dijkstra、只入队 1 号、Floyd 循环序错、差分约束连反边！

# 高频 TLE / 溢出错误对照与定位指南

| 评测结果与场景 | 常见错误写法 | 正确做法与原因 |
|---|---|---|
| **TLE** 忘写懒删除 | 堆弹出 (du, u) 不写 `if (du > d[u])` | 过期废元素反复遍历邻接表，退化成 $O(nm)$ 超时 |
| **TLE** 正权用 SPFA | 非负权大图嫌麻烦直接写 SPFA | 网格图/菊花图卡成 $O(nm)$，正权务必换堆优 Dijkstra |
| **WA** `dist` 开 `int` | 累加距离或 `INF` 取 `INT_MAX` | 极易加爆溢出成负数！一律用 `long long` 与 `LLONG_MAX/4` |
| **WA** 零权环写 `<=` | SPFA 松弛写成 `<=` | 零权环无限入队导致误判成负环 |

- 三秒定位口诀：**TLE** 查懒删除与正权误用 SPFA；**WA** 查全点入队、`long long` 与差分约束方向。

## 本页要点

记住：正权 Dijkstra 记懒删，距离一律开 `long long`，零权环只用严格 `<`！

# 提交前检查清单（六必查）

1. 读入边权后先确认图里有没有负数边权——正权无脑写堆优 Dijkstra，负权或判负环才用 SPFA？
2. Dijkstra 弹出堆顶 `auto [du, u] = pq.top();` 后，第一行有没有写 `if (du > d[u]) continue;`？
3. 判负环题中，是否把 $1 \dots n$ 所有点都设距离 $0$ 并同时塞进了初始队列（防图不连通漏判）？
4. 距离数组 `d[]` 和累加答案是否全开成 `long long`，无穷大初值是否用 `LLONG_MAX / 4`？
5. 差分约束建模时，是否统一化成了 $x_v \le x_u + w$（或 $\ge$）并核对了起点终点方向？
6. Floyd 三重循环中，中转点 `k` 有没有乖乖放在最外层第一重循环？

## 最后一步

造一组不连通的双块小图（一块含负环、一块不含），验证判负环能抓出来后再提交！

# 本章全部练习与完成标准

1. **[基础]** 学校 OJ：1345 ｜香甜的黄油（`code/dijkstra_butter.cpp`）：枚举牧场 + 堆优化 Dijkstra + 桶计数累加。
2. **[基础]** 学校 OJ：1499 ｜最短路计数：在 BFS/Dijkstra 松弛过程中同步维护到达各点的最短路条数（等长相加、更短覆盖）。
3. **[提高]** 学校 OJ：1507 ｜虫洞 Wormholes（`code/spfa_negcycle.cpp`）：双向正权路 + 单向负权虫洞 + SPFA 全点入队判负环。
4. **[提高]** 学校 OJ：1510 ｜出纳员问题：把 24 小时工作时段转化为前缀和变量，建立差分约束系统并用二分或枚举总人数求解。

## 完成标准

能默写带懒删除的堆优 Dijkstra；能解释为什么负边不能用 Dijkstra 及全点入队原因；独立 AC 上述 4 题。

# 课后安排与下一章预告

- 复习：画一个 3 个点含一条 $-5$ 负权边的反例图，演练为什么 Dijkstra 会选错答案。
- 练习：独立复写 code/ 下两份源码，尝试把《最短路计数》（1499）的等长方案累加也写出来。
- 下一章预告：第 4 章《最小生成树与进阶应用》——如何用最省钱的边把整张城市网全部连起来？看 Kruskal 排序并查集与 Prim 怎么协同挑最小安全边！

## 本章一句话

正权路靠小根堆步步锁死最短，负权与不等式靠松弛次数抓出负环。

谢谢！