图论专题选讲 (差分约束，同余最短路，强连通分量)

2025 CEIT 算法集训队暑期集训 #3

Orange

沈阳师范大学，人工智能学院，CEIT LAB

2025/08/3

目录 I

目录 II

目录 III

为什么需要 SPFA 算法?

- Dijkstra 算法：适用于非负权图的单源最短路径
- Bellman-Ford 算法：可处理负权边，但时间复杂度较高 $O(VE)$
- SPFA 算法：Bellman-Ford 的队列优化版本

核心思想

只有当一个顶点的最短距离估计值发生变化时，才需要更新其邻接顶点

SPFA 算法基本概念

算法全称

Shortest Path Faster Algorithm（最短路径快速算法）

- 由中国学者段凡丁于 1994 年提出
- 基于 Bellman-Ford 算法的队列优化
- 能够检测负权回路
- 平均时间复杂度优于 Bellman-Ford

SPFA 算法核心思想

1. 维护一个队列，存储待更新的顶点
2. 初始时将源点加入队列
3. 从队列中取出顶点，尝试松弛其所有邻接边
4. 如果松弛成功且目标顶点不在队列中，则将其加入队列
5. 重复直到队列为空

松弛操作

如果 $dist[u] + w(u, v) < dist[v]$，则更新 $dist[v] = dist[u] + w(u, v)$

SPFA 算法实现代码

```cpp
bool spfa(int s) {
    fill(dist, dist + n + 1, INF);
    fill(inq, inq + n + 1, false);
    fill(cnt, cnt + n + 1, 0);

    queue<int> q;
    dist[s] = 0;
    inq[s] = true;
    q.push(s);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;

        for (auto& edge : graph[u]) {
            int v = edge.to, w = edge.weight;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n) return false; // 负环
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
    return true;
}
```

负环的概念与判断

负环定义

图中存在一个环，环上所有边的权值之和为负数

- 如果图中存在负环，则不存在最短路径
- 可以通过负环无限减小路径长度

SPFA 中的负环判断

如果某个顶点被松弛的次数超过 $n-1$ 次，则存在负环

原理

最短路径最多包含 $n-1$ 条边，如果松弛次数超过此值，必然存在负环

SPFA 算法复杂度分析

- **最好情况：** $O(E)$ - 每条边只被松弛一次
- **平均情况：** $O(kE)$ - k 为较小常数（通常 $k \leq 2$）
- **最坏情况：** $O(VE)$ - 退化为 Bellman-Ford

空间复杂度

$O(V+E)$ - 存储图和辅助数组

注意

在稠密图或特殊构造的图中，SPFA 可能退化到 $O(VE)$

什么是差分约束?

问题背景

给定一组形如 $x_i - x_j \leq c_k$ 的不等式，求解是否存在满足所有约束的解

- 差分约束系统是线性规划的特殊情况
- 可以转化为图论中的最短路径问题
- 广泛应用于调度、时间安排等问题

实际应用

- 任务调度：任务间的时间约束
- 布局问题：元素间的距离约束
- 区间调度：时间区间的约束关系

差分约束系统的数学形式

标准形式

给定 m 个约束条件：

$$x_{i_1} - x_{j_1} \leq c_1$$

$$x_{i_2} - x_{j_2} \leq c_2$$

$$\vdots$$

$$x_{i_m} - x_{j_m} \leq c_m$$

- $x_1, x_2, \ldots, x_n$ 为未知变量
- $c_1, c_2, \ldots, c_m$ 为已知常数
- 目标：判断是否存在满足所有约束的解

差分约束的图论转化

核心思想

将约束 $x_i - x_j \leq c$ 转化为图中的边 $j \to i$，权值为 c

- 每个变量 x_i 对应图中的一个顶点
- 约束 $x_i - x_j \leq c$ 对应边 (j, i, c)
- 如果图中存在负环，则约束系统无解
- 否则，$dist[i]$ 就是 x_i 的一个可行解

为什么这样转化?

最短路径的三角不等式：$dist[i] \leq dist[j] + c$ 即：$dist[i] - dist[j] \leq c$

另一种转化方式

此外，也可以将系统转化为 $x_i - x_j \geq c$ 的形式，建图方式不变，同样是 $j \to i$，权值为 c。

注意

转化为 $x_i - x_j \geq c$ 后，SPFA 算法需要求最长路径，而不是最短路径。系统中若存在**正环**，则说明无解。

建图方法一：直接建图

适用情况

所有变量都有明确的约束关系，图连通

1. 为每个变量创建一个顶点
2. 对于约束 $x_i - x_j \leq c$，添加边 (j, i, c)
3. 选择任意顶点作为源点运行 SPFA
4. 如果存在负环则无解，否则有解

示例

约束：$x_1 - x_2 \leq 3$, $x_2 - x_3 \leq 2$, $x_3 - x_1 \leq 1$
建图：$2 \to 1$ (权值 3), $3 \to 2$ (权值 2), $1 \to 3$ (权值 1)

建图方法二：超级源点

适用情况

图可能不连通，或需要求特定类型的解

1. 创建超级源点 s
2. 从 s 向所有变量顶点连边，权值为 0
3. 以 s 为源点运行 SPFA
4. 如果存在负环则无解，否则 $dist[i]$ 为 x_i 的解

超级源点的作用

- 保证图的连通性
- 提供统一的起始参考点
- 求得的解通常是字典序最小的解

超级源点的深入理解

设计原理

超级源点到所有顶点的距离初始为 0，相当于给所有变量一个统一的”基准值”

- **连通性保证：** 确保所有顶点都能从源点到达
- **解的唯一性：** 在多个可行解中选择特定的一个
- **数值稳定性：** 避免解中出现过大或过小的数值

为什么权值设为 0?

权值为 0 意味着不增加额外约束，只是提供连通性

注意

超级源点不参与原问题的约束，只是算法实现的技巧

差分约束系统的无解判断

无解的充要条件

差分约束系统无解当且仅当约束图中存在负权回路

- **必要性：**如果有解，则不能存在负环
- **充分性：**如果无负环，则 SPFA 求得的距离就是一组解

负环的含义

负环表示存在矛盾的约束，如：

$$x_1 - x_2 \leq -1$$

$$x_2 - x_3 \leq -1$$

$$x_3 - x_1 \leq -1$$

相加得：$0 \leq -3$，矛盾！

差分约束算法复杂度

- **时间复杂度：** $O(nm)$ - n 个变量，m 个约束
- **空间复杂度：** $O(n+m)$ - 存储图结构
- **实际表现：**通常远好于最坏情况

优化技巧

- 使用 SLF 优化（Small Label First）
- 使用 LLL 优化（Large Label Last）
- 合理选择源点

一个题目

小 K 的农场

一共有 n 个农场，编号 1 n，给定 m 条关系，每条关系是如下三种形式中的一种:

- 关系 1 a b c：表示农场 a 比农场 b 至少多种植了 c 个作物
- 关系 2 a b c：表示农场 a 比农场 b 至多多种植了 c 个作物
- 关系 3 a b：表示农场 a 和农场 b 种植了一样多的作物

如果关系之间能推出矛盾，输出"No"，否则输出"Yes"。数据范围:$1 \leq n, m \leq 5 * 10^3, 1 \leq c \leq 5 * 10^3$

思路

首先，形式化每种关系，我们统一使用小于等于的关系进行建模：关系 1 可以表示为 $a - b \geq c$，此时通过移项，可以转化成：$b - a \leq -c$，此时建立一条从 a 到 b 的边，权值为-c。

同理，关系 2 可以表示为 $a - b \leq c$，此时建立一条从 b 到 a 的边，权值为 c。

关系 3 可以表示为 $a - b \leq 0 \wedge a - b \geq 0$，此时建立一条从 a 到 b 的边和一条从 b 到 a 的边，权值均为 0。

至此，图论部分的建模结束，直接在图上跑 SPFA 判断是否有负环即可。

第二个题目

布局奶牛

编号 1 到编号 n 的奶牛**从左往右站**成一排，你可以决定任意相邻奶牛之间的距离。有 m1 条好友信息，有 m2 条情敌信息，好友间希望距离更近，情敌间希望距离更远。

- 好友信息表示为: u v w，表示希望 u 和 v 之间的距离 <= w，输入保证 u < v
- 情敌信息表示为: u v w，表示希望 u 和 v 之间的距离 >= w，输入保证 u < v

你需要安排奶牛的布局，满足所有的好友信息和情敌信息。如果不存在合法方案，返回-1，如果存在合法方案，返回 1 号奶牛和 n 号奶牛之间的最大距离，如果存在合法方案，并且 1 号奶牛和 n 号奶牛之间的距离可以无穷远，返回-2。

思路

非常裸的一道差分约束的题目，直接根据题意进行建图即可。对于每对情敌，建立一条从 u 到 v 的边，权值为-w。对于每对好友，建立一条从 v 到 u 的边，权值为 w。
此外，值得注意的是，题目强调了从左向右的关系，也就是 $\forall_{i<j} x_i \leq x_j$，因此考虑再构建每个节点 i，都向 i - 1 建立一条权值为 0 的边，即可约束顺序关系。

什么时候，1 号奶牛和 n 号奶牛之间的距离可以无穷远?

实际上，我们差分约束求出的是若干组解系组合而成的一组解系。对于两个变元 x 和 y，若其之间存在约束关系，则我们称其位于相同解系之中，在相同解系中，其所有解可以加减任意实数。而不同解系之间，不存在任何关系限制。所以，假设 1 和 n 不在同样的解系之中，则其就可以无穷远。**在图论中，约束关系的本质就是边，因此在同一个连通块内的节点，就位于相同解系。**

思路

最终，我们在图上跑 SPFA，判断是否存在负环即可。若存在则答案为-1，此外，若无负环，还需要判断 1 和 n 是否在同一个连通块内，若不在，则答案为-2，否则答案为 dist[n]-dist[1]。

带常量的差分约束系统

在差分约束系统中，若部分变量给定了具体值，我们可以将其视为一个带常量的差分约束系统。首先，需要明确，差分约束系统约束的主要是相对关系，与是否是常量无关，只要能将常量转化为正确的约束条件，即可建立正确的差分约束系统。

例

a $=4$, c $=10$，求解如下约束关系：

$$\begin{cases} b \leq a \\ c \leq b \end{cases}$$

实际上，我们根本无需考虑 a 和 b 的值，我们只需要考虑 a 和 b 之间的约束关系。由于 a $=4$, c $=10$，因此其必然存在如下约束：$a - c \leq -6 \wedge a - c \geq -6$。将其转化为图论中的两条边即可。

带常量的差分约束系统

此时，显然存在一个严重的问题，我们任意两组常量之间都存在互相的约束关系，假设常量趋近于 n，我们新建立的边的数量级可能到达 $\binom{n}{2}$ 也就是 $O(n^2)$ 级别，这显然是无法接受的。因为在这样的图上跑一次 SPFA 的复杂度会来到 $O(n^3)$。

我们可以考虑引入一个基准点 p，用来统一衡量所有的常量，即所有常量都是相对 p 成立的，此时，我们最多建立的边就只会到达 $O(n)$ 这个数量级，允许 SPFA 通过。最终，求出的解再假设 dist[p] 的偏移即可。

一个题目

倍杀测量者

如果 A 的分数 >= B 的分数 * k，k 是正实数，就称 A k 倍杀 B，或称 B 被 A k 倍杀了。一场比赛中，一共有 n 个选手，有 m1 条誓言记录，有 m2 条选手得分记录，得分只可能是正实数。

- 类型 1 的誓言 u v k：选手 u 没有 k 倍杀选手 v，那么选手 u 就穿女装
- 类型 2 的誓言 u v k：选手 u 被选手 v k 倍杀了，那么选手 u 就穿女装
- 选手的得分 u w：选手 u 得了 w 分，如果某选手没有得分记录，按照尽量不穿女装的情况推测

你希望**看到比赛后有人穿女装，但不想看到很多人穿女装**，于是想制定正实数 R，效果如下:
类型 1 的誓言，比例调整成 (k-R)，类型 2 的誓言，比例调整成 (k+R)，即提高了穿女装的条件。

计算 ans 最大多少，依然有人穿女装，保留小数点后 4 位，如果不干预也没人穿女装，

返回-1

思路

根据两种类型的誓言，我们统一采用不穿女装作为约束条件：
对于给出的若干个约束条件，如果约束条件都成立，则没有人会女装。你需要求出最大的 R，使得至少有一个人穿女装，即约束条件不成立。**很显然，R 越大，约束条件肯定是越难成立的。**

很显然可以得出 R 具有单调性，考虑二分答案。

此时，观察约束条件，其与我们差分约束系统的条件并不相同，我们期望的约束是二者之差，而非二者的倍数关系。
不妨观察誓言 1 的不等关系，假设 u 不穿女装，u 应该满足:

$$u \geq v \times (k - R)$$

思路

此时对不等式移项：

$$\frac{u}{v} \geq k - R$$

两边同时取自然对数:

$$\ln u - \ln v \geq \ln k - R$$

此时，我们就得出了不等关系。建图时，若 u/v 的值给定，则考虑用之前设立基准点的方式，与基准点 P 建立一条权值为 $\ln u/v$ 的边。而另外的关系则直接用之前的方式建图即可，注意边权取自然对数即可。

注意

对于誓言 2 的形式，其转化为不等关系为：$u < v \times (k + R)$，此时需要转化为带等号的约束关系，可以考虑在右边加上一个极小值 eps，使得其转为：$u \leq v \times (k + R) + eps$。
eps 一般取题目要求精度的 $\frac{1}{100}$ 左右即可。

问题的起源与背景

弗罗贝尼乌斯硬币问题 (Frobenius Coin Problem)

给定一组互质的正整数 $a_1, a_2, \ldots, a_n$，求不能由这组数的非负整数线性组合构成的最大整数，这个最大数被称为弗罗贝尼乌斯数，记作 $g(a_1, a_2, \ldots, a_n)$

- 在算法竞赛的数学部分中，同余理论是重要组成部分
- 以优雅的方式将无限的整数世界映射到有限的剩余类中
- 为处理与整除、余数、周期性相关的问题提供独特视角
- 当 $n = 2$ 时，有著名结论：$g(a_1, a_2) = a_1 a_2 - a_1 - a_2$
- 当 $n \geq 3$ 时，不存在简单的封闭解公式

具体问题实例

购物凑单问题

小明去商店购物，商店只接受面值为 3 元和 5 元的硬币。问：

1. 哪些金额无法精确支付？
2. 无法支付的最大金额是多少？
3. 在区间 $[1, 100]$ 内有多少个金额可以支付？

手工分析

- 可以支付：$0, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, \ldots$
- 无法支付：$1, 2, 4, 7$
- 最大无法支付金额：7（验证：$g(3,5) = 3 \times 5 - 3 - 5 = 7$）

邮票面值问题

邮票组合问题

邮局发行了面值为 4 分、6 分、10 分的邮票，每种邮票数量无限。问能够组成哪些邮资？不能组成的最大邮资是多少？

分析思路

- 注意到 $\gcd(4,6,10)=2$，所以所有奇数邮资都无法组成
- 问题转化为：用 $2,3,5$（除以 2 后的值）能组成哪些数？
- 根据 $g(2,3)=2\times 3-2-3=1$，大于等于 2 的偶数都能组成
- 因此原问题中，大于等于 4 的偶数邮资都能组成
- 不能组成的邮资：所有奇数 $+\ \{2\}$

动态规划方法的分析

传统 DP 方法

设 $dp[i]$ 表示能否凑出数值 i，状态转移：

$$dp[i] = \bigvee_{j=1}^{n} dp[i - a_j] \quad (i \geq a_j)$$

边界条件：$dp[0] = \text{true}$

以 $a = \{3, 5\}$ 为例

- $dp[0] = \text{true}$（基础情况）
- $dp[3] = dp[0] = \text{true}$，$dp[5] = dp[0] = \text{true}$
- $dp[6] = dp[3] = \text{true}$，$dp[8] = dp[5] = \text{true}$
- $dp[9] = dp[6] \vee dp[4] = \text{true}$
- $dp[1] = dp[2] = dp[4] = dp[7] = \text{false}$

DP 方法的局限性

主要问题

- **空间复杂度过高**：当目标数值达到 10^9 级别时，需要 $O(V)$ 的空间
- **时间复杂度高**：$O(V \cdot n)$，其中 V 是值域，n 是硬币种类数
- **无法处理大范围询问**：如询问区间 $[10^6, 10^9]$ 内的可达数个数
- **内存限制**：实际比赛中往往受到内存限制的约束

同余最短路的突破

- 利用**同余性质**，将问题规模从 $O(V)$ 降到 $O(\min(a_i))$
- 通过**图论算法**求解，避免大数组的使用
- 能够高效处理**值域很大**的情况
- 空间复杂度降为 $O(\min(a_i))$，通常在可接受范围内

同余最短路的核心洞察

关键观察

对于任意可达数 x，如果 $x \equiv r \pmod{m}$（其中 $m = \min\{a_1, a_2, \ldots, a_n\}$），那么：

- 所有形如 $x + km$（$k \geq 0$）的数都是可达的
- 我们只需要找到每个余数类中的**最小可达数**

以 $a = \{3, 5\}$，$m = 3$ 为例

- 余数 0：最小可达数是 0，所有 $3k$（$k \geq 0$）都可达
- 余数 1：最小可达数是 ∞（不存在），所有 $3k + 1$ 都不可达
- 余数 2：最小可达数是 5，所有 $3k + 2$（$k \geq 1$）都可达

核心思想

将**无限的数值问题**转化为**有限的图论问题**！

构建同余图的基本框架

选取模数

选择 $a_1, a_2, \ldots, a_n$ 中的最小值作为模数：

$$m = \min\{a_1, a_2, \ldots, a_n\}$$

原因：最小值能保证图的规模最小，且不会遗漏解

定义图的顶点

图 $G = (V, E)$ 中的顶点集合为：

$$V = \{0, 1, 2, \ldots, m-1\}$$

每个顶点 i 代表模 m 意义下的余数 i

图论解释

- 任何正整数 x 都可以表示为 $x = qm + r$，其中 $0 \leq r < m$
- 顶点 r 对应所有模 m 余 r 的数的集合
- 通过图上的**最短路径**来找到每个余数类的最小可达数

构建图的边：基本规则

边的构建规则

对于每个硬币面值 a_i $(i = 1, 2, \ldots, n)$，在图中添加边：

$$u \xrightarrow{a_i} v$$

其中 $v = (u + a_i) \bmod m$，边权为 a_i

重要说明

- 包括模数 m 本身对应的硬币（通常是 $a_1 = m$）
- 每条边代表" 使用一枚面值为 a_i 的硬币"
- 边权 a_i 表示使用该硬币增加的总价值
- 从余数 u 到余数 $(u + a_i) \bmod m$ 的转移

边集合的数学表示

$$E = \{(u, (u + a_i) \bmod m, a_i) : u \in V, i = 1, 2, \ldots, n\}$$

具体例子：$a = \{3, 5\}$ 的图构建

参数设置

- 硬币面值：$a_1 = 3, a_2 = 5$
- 模数：$m = \min\{3, 5\} = 3$
- 顶点集合：$V = \{0, 1, 2\}$

边的构建过程

使用面值 3 的硬币：

- $0 \xrightarrow{3} (0 + 3) \bmod 3 = 0$
- $1 \xrightarrow{3} (1 + 3) \bmod 3 = 1$
- $2 \xrightarrow{3} (2 + 3) \bmod 3 = 2$

使用面值 5 的硬币：

- $0 \xrightarrow{5} (0 + 5) \bmod 3 = 2$
- $1 \xrightarrow{5} (1 + 5) \bmod 3 = 0$
- $2 \xrightarrow{5} (2 + 5) \bmod 3 = 1$

最短路在同余最短路中的目标

目标定义

在构建的同余图中，我们要求从源点 0 到每个顶点 i 的最短路径长度 $dist[i]$

图论解释

- $dist[i]$ 表示能够凑出的、模 m 余 i 的**最小非负整数**
- 如果 $dist[i] = \infty$，说明不存在模 m 余 i 的可凑数
- 一旦知道了 $dist[i]$，所有形如 $dist[i] + k \cdot m$（$k \geq 0$）的数都可以凑出
- 这就是同余最短路的**核心思想**：用有限信息表示无限集合

源点设置

- 源点选择顶点 0，因为数值 0 可以通过”不选择任何硬币”来凑出
- $dist[0] = 0$，表示凑出数值 0 的最小代价为 0
- 这是算法的**边界条件**

同余最短路算法详细步骤

第一步：预处理与建图

1. 选择模数：$m = \min\{a_1, a_2, \ldots, a_n\}$
2. 创建图 $G = (V, E)$，其中 $V = \{0, 1, \ldots, m-1\}$
3. 对每个硬币面值 a_i 和每个顶点 $u \in V$，添加边：$u \xrightarrow{a_i} (u + a_i) \bmod m$

第二步：最短路计算

1. 初始化：$dist[0] = 0$，$dist[i] = \infty$ $(i = 1, 2, \ldots, m-1)$
2. 使用 Dijkstra 算法求从顶点 0 到所有其他顶点的最短路
3. 得到数组 $dist[]$，其中 $dist[i]$ 表示模 m 余 i 的最小可达数

第三步：问题求解

根据具体问题类型，利用 $dist[]$ 数组计算答案

常见问题类型与解法

问题类型一：判断数 x 是否可达

解法： 检查 $dist[x \bmod m]$ 是否有限且 $dist[x \bmod m] \leq x$

问题类型二：求弗罗贝尼乌斯数（最大不可达数）

解法： 遍历所有余数类，找到最大的不可达数

- 如果 $dist[i] = \infty$，则所有 $mk + i$ 都不可达
- 如果 $dist[i] < \infty$，则 $dist[i] - m + i$ 是该余数类中最大的不可达数

问题类型三：统计区间 $[L, R]$ 内可达数个数

解法： 对每个余数类分别计算，然后求和

- 对于余数 i，如果 $dist[i] \leq R$，计算区间内该余数类的可达数个数
- 公式：$\max(0, \lfloor \frac{R-dist[i]}{m} \rfloor + 1 - \max(0, \lceil \frac{L-dist[i]}{m} \rceil))$

复杂度分析

时间复杂度详细分析

- **建图阶段**：$O(m \cdot n)$，其中 $m = \min(a_i)$，n 是硬币种类数
- **Dijkstra 算法**：$O((m \cdot n) \log m)$，使用优先队列优化
- **总体时间复杂度**：$O(mn \log m)$

空间复杂度

$O(m \cdot n)$，主要用于存储图的邻接表和距离数组

与传统 DP 的对比

- **传统 DP**：时间 $O(V \cdot n)$，空间 $O(V)$
- **同余最短路**：时间 $O(mn \log m)$，空间 $O(mn)$
- 当 $V \gg m$ 时（如 $V = 10^{18}, m = 100$），同余最短路有**压倒性优势**

算法优势与适用场景

核心优势

- **处理大值域**：能处理值域达到 10^{18} 的问题
- **一次计算多次查询**：预处理后可高效回答各种询问
- **理论优雅**：将数论问题转化为图论问题
- **扩展性强**：可处理带权重、多维等变种问题

适用条件

- 硬币面值的最小值不太大（通常 $m \leq 10^5$）
- 需要处理值域很大的问题（$V \geq 10^6$）
- 硬币种类数不太多（通常 $n \leq 100$）
- 需要回答多种类型的询问

不适用场景

当最小硬币面值很大（如 $m > 10^6$）时，建图开销过大，不如直接使用其他方法

经典例题一：弗罗贝尼乌斯硬币问题

问题描述

给定 n 个互质的正整数 $a_1, a_2, \ldots, a_n$，求不能表示的最大数（弗罗贝尼乌斯数）

解题思路

1. 构建同余图，计算 $dist[]$ 数组
2. 对每个余数类 i，找到该类中最大的不可达数：
   - 如果 $dist[i] = \infty$，则所有 $mk + i$ 都不可达，答案可能是 ∞
   - 如果 $dist[i] < \infty$，则该类中最大不可达数是 $dist[i] - m$
3. 取所有余数类中最大不可达数的最大值

经典例题二：区间计数问题

问题描述

给定硬币面值 $a_1, a_2, \ldots, a_n$，询问区间 $[L, R]$ 内能凑出的数的个数

解题思路

1. 构建同余图，计算 $dist[]$ 数组
2. 对每个余数类 i，计算该类在区间 $[L, R]$ 内的可达数个数
3. 对于余数类 i，可达数形如 $dist[i], dist[i] + m, dist[i] + 2m, \ldots$
4. 计算公式：设 $d = dist[i]$，则该类在 $[L, R]$ 内的可达数个数为：

$$\max\left(0, \left\lfloor \frac{R-d}{m} \right\rfloor - \left\lceil \frac{L-d}{m} \right\rceil + 1\right)$$

5. 将所有余数类的结果求和

拓展应用与变种问题

多维同余最短路

当问题涉及多个约束条件时，可以构建多维状态空间：

- 状态：$(x \bmod m_1, x \bmod m_2, \ldots)$
- 适用于有多个模数约束的问题

带权重的硬币问题

每种硬币有不同的" 价值" 或" 代价"，目标是最小化总代价：

- 边权设为硬币的代价而非面值
- 最短路的含义变为最小代价

限制硬币使用次数

每种硬币最多使用 k 次的变种：

- 状态扩展为 (余数, 各硬币使用次数)
- 状态空间变大，但思想相同

模板题

跳楼机

你在一座 h 层高的大楼的第 1 层。有三种移动方式：向上移动 x 层、向上移动 y 层、向上移动 z 层。问你总共能到达多少个不同的楼层（包括第 1 层）。

思路

一个同余最短路的模板题，本质上是求解 [0, h-1] 内有多少个数能被表示成形如: $k_1x + k_2y + k_3z$(其中，$k_1, k_2, k_3 \geq 0$) 。
考虑选择 xyz 三者中最小的作为基数，将其他三者的关系进行同余建图，使用 dijk 求出基数每种余数的最短路即可。
答案即为 $\sum[dist[i] \leq h] \cdot (1 + \frac{h-dist[i]}{base})$，其中 base 为基数。

一个题目

牛场围栏

有 N 种长度为 a_i 的木料，每种无限。每根木料可以被截短 0 到 M 的任意整数长度。问用这些处理后（或不处理）的木料拼接，无法组成的最大围栏长度是多少。如果不存在无法构成的围栏长度或者无法构成的围栏长度无穷大，输出-1。

思路

本质上还是若干种不同的数不能凑出的最大值，对于每个 a_i，其本质代表了 $[\max(1, a_i - m), a_i]$ 这么多种不同的数（注意去重），将所有不同的数统计后，跑同余最短路即可。

特殊情况

如果集合中存在 1，则说明任何长度都可以被构建，不存在无法构成的围栏长度。
如果存在某个 $dist[i] = \infty$，则说明所有余数为 i 的长度的围栏均无法被构建，因此无法构成的围栏长度可以达到无穷大。

一个题目

Small Multiple

给定一个整数 K。求一个 K 的正整数倍中，数位之和最小的那个，并输出这个最小的数位和。

思路

本质上还是一个最短路问题。我们想要找到一个目标数，这个数需要满足两个条件：是 K 的倍数，且数位和最小。
不妨考虑用模 K 的余数作为图的状态。我们建立一个有 K 个顶点的图，顶点编号为 0,1,...,K-1。定义 d[i] 为：所有模 K 余 i 的数中，最小的数位和是多少。我们的最终目标就是 d[0]。
如何从一个数得到另一个数？可以考虑两种基本操作来构建任意正整数：

- 乘以 10：如果我们有一个数 S，我们可以得到 $S\times 10$。如果 $S\ i(mod K)$，那么 $S\times 10\ (i\times 10)(mod K)$。这个操作不改变数位和，对应从顶点 i 到 $(i\times 10)(mod K)$ 的边，权重为 0。
- 加 1：如果我们有一个数 S，我们可以得到 $S+1$。如果 $S\ i(mod K)$，那么 $S+1\ (i+1)(mod K)$。这个操作使数位和增加 1，对应从顶点 i 到 $(i+1)(mod K)$ 的边，权重为 1。

图中只有权重为 0 和 1 的边，可以考虑使用 01bfs 优化 dijkstra，使得遍历图的复杂度降低为 $O(n)$，使用 01bfs 求解 d[0] 即为答案。

一个题目

[国家集训队] 墨墨的等式

墨墨突然对等式很感兴趣，他正在研究 $\sum_{i=1}^{n} a_i x_i = b$ 存在非负整数解的条件，他要求你编写一个程序，给定 $n, a_{1\ldots n}, l, r$，求出有多少 $b \in [l, r]$ 可以使等式存在非负整数解。

思路

跳楼机的升级版，从 3 个操作升级为 n 个操作。直接套用同余最短路即可。
我们要计算 [l,r] 的可达数，可以转化为 r 以内的可达数减去 l-1 内的可达数。

强连通分量的概念引入

有向图的连通性

在有向图中，连通性比无向图更复杂，需要考虑边的方向

- 无向图：两点间存在路径即连通
- 有向图：需要考虑双向可达性
- 强连通：两点间互相可达
- 强连通分量：最大的强连通子图

实际应用

- 网络分析：识别紧密连接的节点群
- 编译器：检测循环依赖
- 社交网络：发现社区结构

强连通分量的严格定义

强连通

对于有向图 $G = (V, E)$，如果顶点 u 和 v 之间存在从 u 到 v 的路径，也存在从 v 到 u 的路径，则称 u 和 v 强连通

强连通分量

有向图 G 的强连通分量是一个最大的顶点集合 $C \subseteq V$，使得对于任意 $u, v \in C$，u 和 v 都强连通

- 强连通关系具有等价关系的性质（自反、对称、传递）
- 强连通分量构成对顶点集的划分
- 每个顶点恰好属于一个强连通分量

强连通分量的重要性质

1. **划分性质**：所有强连通分量构成对顶点集的划分
2. **缩点性质**：将每个强连通分量缩成一个点，得到的图是 DAG
3. **路径性质**：同一强连通分量内任意两点互相可达
4. **最大性质**：强连通分量是包含关系下的最大强连通子图

缩点图的重要性

缩点后的 DAG 保留了原图的"层次结构"，常用于：

- 拓扑排序
- 最长路径问题
- 可达性分析

Tarjan 算法概述

算法特点

Tarjan 算法是一种基于 DFS 的线性时间算法，能够在一次遍历中找出所有强连通分量

- 由 Robert Tarjan 于 1972 年提出
- 时间复杂度：$O(V+E)$
- 空间复杂度：$O(V)$
- 使用 DFS 和栈的巧妙结合

核心思想

利用 DFS 的回溯性质，在遍历过程中识别强连通分量的”根节点”

Tarjan 算法的关键概念

- **DFS 序：** $dfn[u]$ - 顶点 u 的 DFS 访问时间戳
- **Low 值：** $low[u]$ - 从 u 出发能到达的最小 DFS 序
- **栈：** 存储当前路径上的顶点
- **在栈标记：** $instack[u]$ - 顶点 u 是否在栈中

Low 值的计算

$$low[u] = \min \begin{cases} dfn[u] & \\ low[v] & \text{对于树边}(u, v) \\ dfn[v] & \text{对于后向边}(u, v) \text{ 且}v \text{ 在栈中} \end{cases}$$

强连通分量的根

当 $dfn[u] = low[u]$ 时，u 是一个强连通分量的根

Tarjan 算法详细步骤

1. **初始化：**设置时间戳计数器，初始化数组
2. **DFS 遍历：**对每个未访问的顶点进行 DFS
3. **访问顶点：**设置 $dfn[u]$ 和 $low[u]$，将 u 入栈
4. **遍历邻接点：**对于每个邻接点 v：
   - 如果 v 未访问：递归访问，更新 $low[u]$
   - 如果 v 在栈中：更新 $low[u] = \min(low[u], dfn[v])$
5. **判断根节点：**如果 $dfn[u] = low[u]$：
   - 不断弹栈直到弹出 u
   - 弹出的所有顶点构成一个强连通分量

Tarjan 算法实现代码

```
class Tarjan {
public:
    vector<vector<int>> graph;
    vector<int> dfn, low, scc_id;
    vector<bool> instack;
    stack<int> st;
    int timestamp, scc_count;

    void tarjan(int u) {
        dfn[u] = low[u] = ++timestamp;
        st.push(u);
        instack[u] = true;

        for (int v : graph[u]) {
            if (dfn[v] == 0) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            } else if (instack[v]) {
                low[u] = min(low[u], dfn[v]);
            }
        }

        if (dfn[u] == low[u]) {
            scc_count++;
            while (true) {
                int v = st.top();
                st.pop();
                instack[v] = false;
                scc_id[v] = scc_count;
```

Tarjan 算法实现代码

```
                if (v == u) break;
            }
        }
    }

    void findSCC(int n) {
        dfn.assign(n + 1, 0);
        low.assign(n + 1, 0);
        scc_id.assign(n + 1, 0);
        instack.assign(n + 1, false);
        timestamp = scc_count = 0;

        for (int i = 1; i <= n; i++) {
            if (dfn[i] == 0) {
                tarjan(i);
            }
        }
    }
};
```

Tarjan 算法复杂度分析

- **时间复杂度：** $O(V+E)$
  - 每个顶点被访问恰好一次
  - 每条边被检查恰好一次
  - 栈操作总次数为 $O(V)$
- **空间复杂度：** $O(V)$
  - DFS 递归栈：$O(V)$
  - 辅助数组：$O(V)$
  - Tarjan 栈：$O(V)$

算法优势

- 线性时间复杂度，最优
- 一次遍历完成，效率高
- 实现相对简单，代码量少

一个题目

受欢迎的牛

每一头牛的愿望就是变成一头最受欢迎的牛。现在有 N 头牛，给你 M 对整数 (A,B)，表示牛 A 认为牛 B 受欢迎。这种关系是具有传递性的，如果 A 认为 B 受欢迎，B 认为 C 受欢迎，那么牛 A 也认为牛 C 受欢迎。你的任务是求出有多少头牛被除自己之外的所有牛认为是受欢迎的。

思路

首先，我们可以把受欢迎看作一种有向关系，那么每个 scc 中的牛一定都互相认为是受欢迎的。其次，如果要受到所有牛的欢迎，那么在 scc 缩点后，当前的点一定要能够被所有点可达。由于缩点后是一个 DAG，因此只需要考虑出度为 0 的 scc 即可。注意！如果有多个出度为 0 的点，则一定不存在答案。否则，唯一的出度为 0 的 scc 的大小就是答案。

一个题目

网络协议

一些学校连接在一个计算机网络上。学校之间存在软件支援协议。每个学校都有它应支援的学校名单（学校 a 支援学校 b，并不表示学校 b 一定支援学校 a）。当某校获得一个新软件时，无论是直接得到还是网络得到，该校都应立即将这个软件通过网络传送给它应支援的学校。因此，一个新软件若想让所有连接在网络上的学校都能使用，只需将其提供给一些学校即可。

任务 1: 根据学校间支援协议（各个学校的支援名单），计算最少需要将一个新软件直接提供给多少个学校，才能使软件通过网络被传送到所有学校；

任务 2: 如果允许在原有支援协议上添加新的支援关系。则总可以形成一个新的协议，使得此时只需将一个新软件提供给任何一个学校，其他所有学校就都可以通过网络获得该软件。编程计算最少需要添加几条新的支援关系。

思路

首先，对于任务 1，由于软件可以相互传递，因此只要一个点有入度，我们就可以将软件给他的上游学校，这样能够使得其能够间接获得软件。最终，只需要考虑 scc 缩点后，入读为 0 的点数即可。
对于任务 2，本质上是在询问，再加多少条边，可以使得整个图变成一整个 scc。考虑图中的特殊点，入读为 0 的点：不能被其他点到达；出度为 0 的点：不能到达其他点；且他们一定是整张图的端点，不妨考虑如下方案，将出度为 0 的点都与“下一个“入度为 0 的点进行配对，这样一定能保证整张图联通。此外，如果有多余的出度为 0 或者入读为 0 的点，统一连接到最后一个入度为 0 的点上。最终，答案为 max(入度为 0 的点数，出度为 0 的点数)。

课程内容总结

- **SPFA 算法：**处理负权边的单源最短路径，能检测负环
- **差分约束：**将不等式组转化为图论问题，利用最短路求解
- **同余最短路：**利用模运算的周期性解决数论问题
- **强连通分量：**有向图的重要结构，Tarjan 算法高效求解

共同特点

这些算法都体现了图论与其他数学分支的深度结合

学习建议

- 理解算法的数学原理
- 掌握建图的技巧
- 多做相关练习题
- 注意算法的适用条件

Thank You!

Reference

1. Thomas H. Cormen, et al. Introduction to Algorithms, 3rd Edition
2. 刘汝佳. 算法竞赛入门经典（第 2 版）
3. 胡伯涛. 最短路径算法及其应用
4. Robert Tarjan. Depth-First Search and Linear Graph Algorithms
5. 算法竞赛进阶指南