

规划器的完整算法

应用工作台 (AW) 的合成规划器优化

1 简介

我们正在基于 Minecraft 中的所有配方进行合成规划。合成规划是指，给定要合成的单个物品以及合成它的数量，找出每个配方需要使用多少次。合成可能不止一种方式，而我们想要总合成步数最少的那一个。自然，在游戏中玩家其实注意不到最优，所以较优的可行解也是能接受的。

这里“步数”指的是底层 Minecraft 配方真正被执行的次数，而不是规划器里配方列的个数。一个普通配方执行一次就是一步；tag 边（合成 tag 的那个合成配方）不计步数，因为选哪个成员来填充 tag 在游戏里是免费的；而 Section 1.2 一节提到的概率配方则是被批量折叠起来的，每一个代表 c_r 次真实执行，也就需要 c_r 步。所以每条配方 r 都有一个成本 c_r ，而目标是最小化 $\sum_r c_r x_r$ 。

传统的规划方法完全失效。AE2 认为配方是一棵树，在这棵树上进行深度优先搜索。它的各类优化模组认为配方是一个 DAG，并在图上利用图论算法传播信息。然而，下文会告诉我们巨型 SCC 是不可避免的。

一部分求解器会选择将 SCC 切断成 DAG。这不保证最优性，所以自然也是不完备的，但它非常快速。所以我们采用的方法是，先通过预处理剪枝，再用一次 Section 3 的贪心 DAG 预扫描快速给出一个可行解，最后才交给 CP-SAT 求解器去优化。

1.1 SCC 为什么这么大？

考虑一个随机的二分图，分为 A, B 两部分，占比分别为 α, β ($\alpha + \beta = 1$)。设 A 指向 B 的平均出度为 m ， B 指向 A 的平均出度为 n 。

先求解巨型出分量的概率。设从属于 A 或 B 的节点开始，能走到无穷远处的概率分别是 x_A, x_B ，而灭绝（也就是最终有限）的概率是 u_A, u_B 。既然 A, B 节点的出度符合泊松分布，那么如果 $a \in A$ 要灭绝，它的每一个 B 邻居都必须灭绝，所以

$$u_A = \sum_{i=0}^{\infty} e^{-m} \frac{m^i}{i!} u_B^i$$

其中 $e^{-m} \frac{m^i}{i!}$ 正是泊松分布中 a 有 i 个邻居的概率，而 u_B^i 则是邻居全部灭绝的概率。我们熟知

$$u_A = e^{-m} \sum_{i=0}^{\infty} \frac{(m u_B)^i}{i!} = e^{-m + m u_B}$$

同理， $u_B = e^{-n + n u_A}$ 。

对于巨型入分量的概率，原理类似，只是要注意 A 和 B 的入度现在是 $\beta \frac{n}{\alpha}$ 和 $\alpha \frac{m}{\beta}$ 。这说明灭绝概率 $v_A = e^{-\frac{\beta}{\alpha} n (1 - v_B)}$ ，而 $v_B = \exp\left(-\frac{\alpha}{\beta} m (1 - v_A)\right)$ ，自然无限概率即为 $y_A = 1 - v_A, y_B = 1 - v_B$ 。

那么 A 中节点落在 SCC 中的概率自然是 $x_A y_A$ ，有 α 的部分属于 A 节点； B 中节点落在 SCC 中的概率是 $x_B y_B$ ，有 β 的部分属于 B 节点。从而整个 SCC 的大小是 $\alpha x_A y_A + \beta x_B y_B$ 。

现在我们从物品出发，检查其反向可达的子图，那么可达的概率应当是入分量的概率。所以在这个子图中，SCC 的占比是

$$\frac{\alpha x_A y_A + \beta x_B y_B}{\alpha y_A + \beta y_B}$$

在我们所用的数据集上，这个值大概在 75% - 80% 之间，除了原版只有 50%。在 ATM 上占比实测为 14%，然而我也不知道为什么这么小。

1.2 概率配方

在求解器内部，我们并不处理概率配方。所以，这里描述的是从 Minecraft 中的实际配方编码为 AW 配方的过程，不归 JNI 管。

首先，概率是小数，所以我们当然要放大成整数，而且可以逐级放大：乘以 1, 4, 16, ... 之类的，当做一个四进制数来凑出最终的需求量。这个底数 $m = 4$ 是可以调整的。接下来就是具体如何放大了。

假设原始配方以 p 的概率产出 1 个物品，而放大 i 级的配方应该产出 n 个物品。我们希望以 $c = 90\%$ 的概率保证这条配方在实际执行时可以产出至少 n 个，并将 c 称作**置信度**。从而

$$P(x < n) < \varepsilon = 1 - c$$

这里的 ε 就是容错。然而，这是给全体放大配方一起执行的容错，而分到每个配方头上的自然要少一些。假设分到 i 级配方的是 m_i ，而一共有 k 级，它们的输出是独立的。那么最坏的情况就是所有的配方都会执行 $m - 1$ 次，所以必须满足

$$\prod (1 - \varepsilon_i)^{m-1} \leq \varepsilon$$

我们知道最高级配方可能执行不止 $m - 1$ 次，但这也没有什么办法，所以选择妥协。更何况最高级配方批量已经足够大，可以稳定在期望附近，一般不会出太大问题的。

至于 ε_i 的分配方式，我们希望最小化总的**相对误差**。

我们要在执行 n 次的情况下以置信度 c 产出 d 个物品，也就是 $P(x < d) < c$ ，这时**绝对误差**是 $np - d$ ，而**相对误差**则是 $R = \frac{np-d}{n}$ 。我们高中就学过，在 np 足够大时，二项分布近似于正态分布 $N(np, np(1-p))$ 。设标准正态分布的概率密度函数为 Φ ，自然有

$$\Phi\left(\frac{d - np}{\sqrt{np(1-p)}}\right) < 1 - c$$

考虑到 $\Phi(x) + \Phi(-x) = 1$ ，可以解出

$$d \leq np - \Phi^{-1}(c)\sqrt{np(1-p)}$$

不妨将 $\Phi^{-1}(c)$ 记作常数 z ，那么我们有

$$(np - d)^2 \geq z^2 p(1-p)n$$

也就是

$$p^2 n^2 - (2pd + z^2 p(1-p))n + d^2 \geq 0$$

考虑到 $n > d$ ，自然有

$$n \geq \frac{2d + z^2(1-p) + \sqrt{z^2(1-p)(4d + z^2(1-p))}}{2p}$$

在零附近的泰勒展开告诉我们， $\sqrt{1+a} = 1 + \frac{a}{2} - \frac{a^2}{8} + \dots$ 。现在 d 很大，所以 $z^2 \frac{1-p}{d}$ 应当足够小。那么

$$\sqrt{4d(1-p) + z^2(1-p)^2} = \sqrt{4d(1-p)} \left(1 + \frac{z^2(1-p)^2}{8d(1-p)} + O(d^{-2})\right)$$

接下来再将外面的根号乘进去，化简，就可以得到

$$np \geq d + \frac{z^2(1-p)}{2} + z\sqrt{d(1-p)} + \frac{z^3(1-p)^{\frac{3}{2}}}{8\sqrt{d}}$$

所以，相对误差 R 其实是 $z\sqrt{\frac{d(1-p)}{n}} + C + O(n^{-1}d^{-\frac{1}{2}})$ ，其中 $C = \frac{z^2(1-p)}{2}$ ，是个常数，跟我们最小化的目标无关。我们忽略小量，并且认为 $d \sim np$ ，那么 $R \sim z\sqrt{\frac{p(1-p)}{n}}$ 。具体忽略了多少其实我也懒得算了，就先这样吧，恐怕有个 $O(n^{-\frac{1}{2}})$ ？但是大概问题不大（不大在哪，这玩意也是 $O(n^{-\frac{1}{2}})$ 的）。

总之，接下来就是拉格朗日乘数法了。我们的约束是

$$\prod_i (1 - \varepsilon_i)^{m-1} = \varepsilon$$

需要最小化 $\sum_i R_i$ 。

那么我们自然直接求导。这里 p 和 n 都和 ε_i 没有关系，但 $z = \Phi^{-1}(c) = 1 - \Phi^{-1}(\varepsilon)$ 。这时 $\frac{dz}{d\varepsilon} = -\frac{1}{\varepsilon}$ 。

$$\frac{\partial R_i}{\partial \varepsilon_i} = -\frac{1}{\varepsilon_i} \sqrt{\frac{p(1-p)}{n}}$$

而不同层级的 z 之间没有关系，所以 $\sum_i R_i$ 求导之后只剩这一项。从而 ε_i 应当正比于 $n^{-\frac{1}{2}}$ 。这样就好了，最后归一化就完了。

2 基于支配的剪枝

2.1 Tag 剪枝

2.1.1 合成图

合成图是一个二分图。物品只能指向配方，而配方只能指向物品。

它还是一个 AND-OR 图，同时配方指向物品的边是**超边**（AND 边）：我们熟悉的边（OR 边）是任选一条边即可，而这里的超边则需要同时走完所有边。

2.1.2 支配

我们考虑这样的例子：假设我们要合成钢制机壳，它需要 4 个任意种类的玻璃。我们知道 Minecraft 中有 16 种染色玻璃，它们通过普通玻璃和染料合成。注意到在合成任何一种染色玻璃时，我们都需要经过普通玻璃，所以普通玻璃**支配**了所有的合成路径。这时，额外花费物品去合成染色玻璃自然是不划算的。

在编译原理中，我们熟知计算支配关系的算法。然而，那个算法没法直接用在这里，因为它只能处理普通的图，而这是一个 AND-OR 图。

所以，我们接下来就需要根据我们的使用场景，找出计算**支配**关系的算法，然后基于它进行**剪枝**。

我们的直觉可能会 w 支配 m 定义为：合成 m 时，一定绕不开 w 。然而，这样其实还不够：假设 $8m \leftarrow w$ 是 m 唯一的合成方式。这时，合成 m 时确实一定绕不开 w ，但这能说明我们不需要 m 、直接用 w 了吗？其实是不可以的，因为如果有一个配方中 m, w 可以互换，那么显然用 m 更赚。

所以，我们给出**支配**的定义。这两种情况都算作 w 支配 m ：

1. 合成 m 的任何一种方法，至少会消耗**等量**的 w 。
2. 合成 m 的任何一种方法，至少会消耗**等量**的某个 tag，而其中所有不是 w 的资源都支配 w 。

我们将这个关系记作 $\text{dominate}(m, w)$ 。

2.1.3 支配关系的计算

最直接的计算方式是直接遍历所有的 (m, w) ，然后根据定义判断。这肯定很慢，所以为了加速，我们只想收集有用的 (m, w) ，排除一些明显不可能支配 m 的 w 。

具体怎么算有用呢？对于一个物品 m 与合成 a 个它的配方 r 而言，如果 r 中含有至少 a 个 w 作为输入，那么 w 确实有可能支配 m ，这时 (m, w) 是有用的。如果 r 中含有的不是某个具体的资源，而是

一个 tag J ，那么 J 的每一种资源都可以当做上述的 w ，这时所有的 (m, w) 都是有用的。我们将这个集合称作 $witness(r)$ 。

接下来，将 m 所有配方 r 的 $witness(r)$ 取交集，得到的结果称作 $c_1(m)$ 。这里取交集是因为根据定义，**每一个**配方都必须用到 w 。

有了所有的 (m, w) ，我们又该如何精确判断 w 是否支配 m 呢？

我们的想法是，首先假设所有的 (m, w) 都满足 $dominate(m, w)$ ，然后逐渐删除明显不正确的那些。我们将算法过程中逐渐缩小的这个关系称作 $alive(m, w)$ ，在结束时它应当收敛于 $dominate(m, w)$ 。由于 $dominate$ 的定义是递归的，不动点迭代是常规方法。

我们不妨将支配定义的条件记作 $valid(m, w)$ ，展开写就是

$$\forall r \in recipes(m). \exists (j, a) \in input(r). \underbrace{j = w}_{\text{条件 1}} \vee \underbrace{isTag(j) \wedge \forall z \in M(j). alive(z, w)}_{\text{条件 2}}$$

注意第二个条件并不要求 $w \in M(j)$ ，这指的正是 j 这个 tag 的每一个成员都绕不开 w 。

注意到，这和 $dominate$ 的定义是一样的，只是将递归检测 $dominate$ 的部分改换为了检测 $alive$ 。下面的代码实现中我们将用到这个辅助函数。

在删除 $valid(m, w)$ 不成立的节点时，一些节点可能受到影响，不再成立。对于普通的资源倒还好说，但如果遇到的是 tag 就比较麻烦了。将 tag 记作 J ，它的成员记作 M ，其中 $w = M[i]$ 是 M 中的第 i 个资源。根据定义，如果 M 里除了 w 以外，使得 $dominate(z, w)$ 成立的 z 数量恰好是 $|M| - 1$ ，那么 w 就支配 m 。我们不妨将这个数量称作 $cnt(J, i)$ 。当且仅当 $cnt(J, i)$ 从 $|M| - 1$ 掉到 $|M| - 2$ 的时候， $alive(m, w)$ 从成立变为不成立，才会产生影响。这就是维护 cnt 的作用。

我们维护一个 `worklist`，来将 $alive$ 迭代至不动点：

```
for (m, w) in universe:
    worklist.push(m, w)

# 不断删除不可能的节点
while not empty(worklist):
    m, w = worklist.pop()
    if alive(m, w) and not valid(m, w):
        kill(m, w)

def kill(m, w):
    dominate(m, w) = 0
    for (J: tag) in tagsOf(m).intersect(tagsOf(w)):
        M = members(J)
        i = lowerBound(M, w)
        cnt(J, i) -= 1
        if cnt(J, i) + 1 == len(M) - 1:
            for m2 in consumers(J):
                push(m2, w) # may not exist; skip if absent
```

2.1.4 传递闭包

上面的 $valid$ 还有一个递归没有展开：第一个条件只接受 $j = w$ ，但如果 j 自己也被 w 支配，那么合成 m 同样绕不开 w 。补上这一条后， $valid$ 就是 $dominate$ 的传递闭包：

$$valid(m, w) = \forall r \in recipes(m). \exists (j, a) \in input(r). j = w \vee alive(j, w) \vee (isTag(j) \wedge \forall z \in M(j). alive(z, w))$$

实现上，tag 那一条是 $alive(j, w)$ 在 tag 上的特例，所以把 (m, w) 与 (j, w) 放进同一个不动点：前者是配方上的 OR，后者是其成员上的 AND。`kill` 的传播也由维护 cnt 改为维护这两个队列。

这两个放宽都会破坏最优性： j 链条可能靠批量余料绕过 w ，tag 那条则可能剪掉整数规划真正需要的边。所以它们**不会在最优模式下**启用。这样一来，关系不再是精确的 dominate，而只是它的一个近似。

2.1.5 关系索引

不动点把所有的 (x, w) 混在一个 worklist 里，但它们之间其实有一个很强的结构：上面每一步传播都只改变 x ，而 w 始终不变。valid 里查的 (j, w) 、tag 的 (z, w) 、kill 里通知消费者的 (m_2, w) ，用的都是同一个 w 。所以，我们可以遍历 w ，并且对每个 w 生成需要的 m 与 z 。

Section 2.1.7 的 SCC 阶段是唯一需要跨行信息的步骤，我们只要存储 m 即可。

2.1.6 列支配

支配定义里的 w 必须出现在 m 的每一个配方里。除此之外，还有一类更简单的情况：只要 s 的产出不少于 r ，每种输入都不多于 r ，且 $c_s \leq c_r$ ，那么 m 的某条配方 r 可以被 w 的某条配方 s 直接替换。实现上，在 valid 里把 r 记为已满足即可。

值得注意的是，只有 m 的配方一次只产生一个 m 时，逐次替换才能把 $T \leftarrow m$ 换成 $T \leftarrow w$ 而不增加总步数。如果 m 产出了不止一个，那么对其他消耗者来说，剩下的部分相当于是免费的，这时如果删掉 tag，就得拿更多的 w 去补。所以**在最优模式下**，只有当 m 的**每一条**配方产出都恰好为 1 时，才允许剪掉 $T \leftarrow m$ 。

FIXME：列支配会保证收支平衡，却不保证计划可以启动。

2.1.7 剪枝

对于含有成员 M 的 tag T ，在 M 中对所有留下的 $\text{alive}(m, w)$ 建立一条边 $m \rightarrow w$ 。注意这可能成环，所以我们需要先缩点：计算强连通分量 (SCC)，然后在所有出度为 0 的 SCC 中挑选一个代表元。这时，其他的所有标签都可以到达这个代表元，只要产出恰好为 1，就可以将它们替换。

2.2 配方剪枝

前面的 tag 剪枝只会删去人工添加的 tag 边。同样的想法自然也可以推广到真实存在的配方。

2.2.1 动机

考虑下面这组配方：

```
黑色蜡烛 x1 <- 黑染料 x1, 蜡烛 x1
黑色蜡烛 x1 <- 黑颜料 x256, 蜡烛 x1
黑颜料 x256 <- 黑染料 x1
黑颜料 x224 <- 黑色蜡烛 x1
```

如果你熟悉通用机械，这里的黑颜料就是来自它，由颜料提取机产出。不过与 tag 剪枝不同，这里的支配只是消耗量的比较： R 和 S 完全可能使用不同的机器。若玩家只有 R 的机器而没有 S 的，删去 R 就丢掉了合成 X 的唯一途径，所以机器在这里不能忽略。

注意区分黑颜料 (mek 模组) 和黑染料 (原版 MC)。

在合成黑颜料时，如果玩家没有黑蜡烛的存货，那么通过合成黑蜡烛来合成黑颜料的方式永远不划算：黑蜡烛要么由黑染料合成，那还不如直接把黑染料变成 256 个黑颜料；要么由黑颜料合成，但那净亏 32 个黑颜料。所以它可以被删掉。

注意 $\text{dominate}(\text{黑染料}, \text{黑蜡烛})$ 并不成立，因为合成黑蜡烛并不是总需要黑染料的，还可以通过黑颜料合成。所以，为了删除配方 $r: X \leftarrow Y$ ，我们需要把 Y 的生产配方**内联**进 r 之后，再和其他的配方比较。

2.2.2 定义

考虑配方 $r: p_r \times o \leftarrow q_1 \times i_1, q_2 \times i_2, \dots$ ，其中 $o, i_1, i_2, \dots$ 是物品，而 $p, q_1, q_2, \dots$ 是固定的系数。

它可以写成一个列向量 v_r ：在输出 o 这一行取 p ，在每个输入 i_n 这一行取 $-q_n$ ，其余为 0。两个向量比较时按分量进行。

设 $R: p \times X \leftarrow q \times Y, \dots$ 是产出 X 的一条配方，其中 Y 是真实资源； $S: p_S \times X \leftarrow \dots$ 是 X 的另一条配方。对 Y 的每一条生产配方 r ，令 $\alpha_r = \left\lceil \frac{q}{p_r} \right\rceil$ ，并称

$$c_r = \alpha_r v_r + v_R$$

为把 r 内联进 R 之后的组合。若对 Y 的每一条生产配方 r ，都有

$$c_r \leq v_S \vee c_r \leq 0$$

就称 S 支配 R 。

c_r 本身表示用 r 造出至少 q 个 Y 、再用 R 处理掉，这一整套下来的资源变化量。如果它 $\leq v_S$ ，那么这 $\alpha_r + 1$ 步可以整体换成 S ：这只需一步，而且产出不减少、消耗不增多。而 $c_r \leq 0$ 的意思是，这是纯亏的，可以直接删掉。第二种情况只删掉执行，成本非负，所以一定不会变差；第一种情况是一次执行换 $\alpha_r + 1$ 次执行，所以要额外要求

$$c_S \leq \alpha_r c_r + c_R$$

也就是换过去不会更贵。

除了这个消耗条件以外，还必须要让替换发生时 S 也能开工。每个查询都会带上可用的机器集合，所以剪枝时再检查即可；在预处理时，为每条被支配的 R 记下可能的替换者就行了。

2.2.3 支配关系的计算

对每个真实资源 X ，在它的配方集合上建一张有向图： $R \rightarrow S$ 表示 S 支配 R 。因为 R 与 S 产出同一个 X ，比较只需要在同一个 X 的配方之间进行。

判断一条边 $R \rightarrow S$ 时，依次尝试 R 的每个真实输入 Y ，并对 Y 的每条生产配方 r 做一次稀疏向量的分量比较。只要存在一个 Y 使条件对所有 r 成立，就记下这条边，并记下这个 Y 作为守卫。若某个 Y 下没有合法替换，就换 R 的下一个输入继续尝试。注意这里不再检查机器，因为可用性已经推迟到查询时。

和 tag 剪枝一样，这些边可能成环，所以对每个 X 的支配子图按照 SCC 缩点，在每个出度为 0 的 SCC 中保留一个代表元即可。被删去的 R 可以到达若干个保留的代表元，其中任意一个都能替换它，于是预处理时为 R 记下这些代表元的可用机器集合的并集 $W(R)$ ：只要它与玩家的机器相交，就有一个能开工的代表元可以接手。

在真正剪枝的时候，配方的支配还需要检查两件事。其一， R 可能正是玩家消耗存货 Y 的途径，所以只有当 Y 没有存货时才能真正删去 R ：预处理时除了标记被支配的配方，还要记下 Y 到底是哪一个。其二，替换者必须真的能开工，也就是玩家持有的机器要与 $W(R)$ 相交。只要这两条同时满足，就可以删去 R 。

2.2.4 一个无法剪枝的例子

考虑

```
黑色玻璃 x8 <- 黑染料 x1, 玻璃 x8
黑色玻璃 x1 <- 黑颜料 x32, 玻璃 x1
黑颜料 x256 <- 黑染料 x1
黑颜料 x16 <- 黑色玻璃 x1
```

第一眼看上去似乎由黑色玻璃合成黑颜料的边也可以删除，毕竟这要么纯亏，要么被黑染料支配。但这条边是不能删的，因为如果有一个什么东西需要 16 黑颜料和 7 个黑色玻璃，那么显然是使用黑染料 1 次更划算，而删掉之后就需要用两次了。这本质上是整数限制所导致的。

不在最优模式下时，我们会删除这条边。

2.3 列支配剪枝

上一篇的支配关系要先内联输入的生产配方，比较的是“走完整条子链”之后的列。除此之外，和 Section 2.1.6 类似，也有更简单的情况：如果同一物品的两条配方中，其中一条的列向量逐分量地不小于另一条，那么可以直接删去后者。

考虑产出同一物品 X 的两条配方：

$$R : p \times X \leftarrow q_1 \times I_1, q_2 \times I_2, \dots$$

$$S : p' \times X \leftarrow q'_1 \times I_1, q'_2 \times I_2, \dots$$

若 $p' \geq p$ ，对每个输入都有 $q'_n \leq q_n$ （换言之， $v_S \leq v_R$ ），且 $c_S \leq c_R$ ，则把任何一次 R 换成一次 S ，所有物品的收支状况都不会变差，目标也不会变大，所以最优解一定可以不用 R 。这不需要守卫输入：即使玩家持有某个输入，换成 S 也不会更差。

在实现时，可以将偏序 $v_S \geq v_R$ 定义的范畴直接看做一个图，对它缩点后，在每个出度为 0 的 SCC 中保留一个代表元，就保留了所有极大列。

替换者可能需要 R 没有的机器，所以和其他剪枝一样，可用性推迟到查询时判定： R 只有在某个代表元的机器可用时才能真正删去。

2.4 替换剪枝

前面两次真实配方支配都要把输入 Y 的生产配方内联进来。但有一类情况其实不用真的内联：如果合成 Y 的任何一种方法都至少消耗等量的某个真实资源 w ，那么在比较列的时候，就可以把 R 里的一份 Y 直接换成一份 w 。

2.4.1 定义

设 $R : p \times X \leftarrow q \times Y + Z$ ，其中 Z 是其余输入。若每生产一个 Y 至少要消耗一个 w ，令 R_w 为把 R 中的 $q \times Y$ 换成 $q \times w$ 得到的列：

$$v_{R_w} = v_R + q(e_Y - e_w).$$

对 Y 的任意一组恰好产出 q 个 Y 的生产配方 r ，每条配方产出 p_r 个 Y 就要消耗至少 p_r 个 w ，所以 w 那一行不高于 $-q$ ；又因为配方只有唯一产出，其余行都不高于 R 。于是

$$\sum_r v_r + v_R \leq v_{R_w}.$$

R 的每一次执行都必须消耗 q 个 Y ，所以整条「生产 Y + 执行 R 」的链的列都不高于 R_w 。因此，若 R 的同门配方 S 满足 $v_S \geq v_{R_w}$ ，或 $v_{R_w} \leq 0$ ，就可以删去 R 。

这本质上就是把配方剪枝里的 c_r 从“内联每一个生产者”换成“用代价下界 w 代替”。这不需要递归展开，也不会因为生产链成环而发散。

这里同样是一次执行换一次执行，只是换掉的 Y 改由 w 顶替，所以也需要 $c_S \leq c_R$ 。

2.4.2 用哪条支配关系

这里要的是“生产 Y 至少消耗 w ”的代价关系。在 tag 剪枝里，我们的确计算了看起来差不多的 alive，但实际上 alive 还接受 colCover，因此不能直接复用。这是因为，colCover 说的是 Y 的配方可以改造成 w 的配方，但这不等于生产 Y 会消耗 w ，例如 $Y \times 1 \leftarrow A \times 5$ 与 $w \times 1 \leftarrow A \times 1$ 会通过 colCover 让 w 支配 Y ，但 Y 根本不消耗 w 。代价关系的递归形式和 valid 一样：

$$\text{costs}(Y, w) = \forall r \in \text{recipes}(Y). \exists (j, a) \in \text{input}(r). j = w \vee \text{costs}(j, w),$$

其中被消费的 tag 取其全体成员的交。注意这和 tag 剪枝的候选集方向相反：候选集是从种子向下展开的，也就是从配方到输入；而真实输入链必须向上传递。所以实现里不是逐个查询 (Y, w) ，而是逐行求解：固定 w ，从 w 出发沿合格输入的逆向边向上闭包，只记录真正成立的边。闭包规则与 costs 一一对应：某个物品进入行内，就让它作为合格输入的每条配方立即被满足（递归只需要一条这样的

输入)；一个真实物品在它的**每一条**配方都被满足时进入，一个 tag 在所有成员都进入时进入。层数记录推导高度，因此在 `maxCostDepth` 处停下正好对应原来的深度上限。这样只会访问真正成立的少数对（在大型包上只有几千对），而关系里占绝大多数的假对完全不展开，也不会因为环而反复重算。

2.4.3 守卫

和其他剪枝一样，删除推迟到查询时判定：

- 若 Y 有存货，玩家可以直接花掉它，所以只有当 Y 没有存货时才能删去 R 。对于 tag 来说，需要所有的成员都没有存货，所以守卫是一组物品，而不是单个输入。
- 替换者 S 可能只在别的机器上可用，所以和列支配一样要记录代表元的机器并集。
- **在最优模式下**，链上每个物品的所有配方都必须恰好产出一个单位。否则一条批量配方留下的余料可以反过来满足 R ，删掉它反而抬高最优值；这与 tag 篇的批量守卫同源。非最优模式放宽这一条。

3 贪心 DAG 预扫描

我们下面提到了很多基于上界的剪枝，所以我们迫切地想要一个可行的上界；更何况 CP-SAT 也可以通过可行解来加速。所以，我们可以采用启发式算法，也就是贪心，来尝试获得第一个可行解。

我们需要先把 SCC 切成 DAG，然后再在 DAG 上推理。

3.1 切环

最直接的切法是从目标出发做一次拓扑排序。它可以删掉回边，但是删完之后可能会让某些物品不再可达，导致漏解。所以，我们需要考虑一个尽量切得更大的方法。

我们先对物品的投影图做 SCC 分解，再拓扑排序。我们知道拓扑排序是一个不断移除入度为 0 的点的过程，而如果有多于一个可以移除，我们需要按照某种顺序来决定到底动哪个。

为此，我们为每个物品分级：

- 如果有配方可以无限产出，那么称为 2 级；
- 如果有库存，那么称为 1 级；
- 否则，称为 0 级。

我们优先选择级别高的物品。我们还会基于当前 DAG 的深度以及原图中到目标物品的最短距离来排序。如果直接这么写的话需要比较四次，而为了减少比较数量，我们将它们打包在一个 `uint64_t` 中。这意味着物品数量必须小于 2^{22} ，而 DAG 深度必须小于 2^{18} ，不过在达到这么多之前恐怕 Minecraft 自己先崩溃了。

有了排序之后，在同一个 SCC 内部，对于边 $u \rightarrow v$ ，如果 u 排在 v 后面就删除，而在 SCC 之间的边不会成环，可以全部留下。这时我们就留下了 SCC 中的所有点，不会出现 DFS 中让某些点不可达的情况。

我们会跑两次：第一次启用第二条规则，第二次禁用。两条规则给出的是两种不同的切点。在实验之后发现，两种规则无法相互覆盖，所以我们都保留了。

3.2 乐观容量

在朴素的 DFS 下，结果可能会差距七个数量级。我们需要修复这一点，所以得搞个上限。换言之，我们需要为每一个物品 x 计算：为了合成足够数量的目标物品，到底最多需要多少个 x 。我们把它叫做 x 的**乐观容量**，记作 $\text{cap}(x)$ 。

首先，我们考虑生产 x 的配方 $r: p_r \times x \leftarrow q_{r1} \times i_1, q_{r2} \times i_2, \dots$ 。它能生产 x 的个数自然是

$$\text{producible}(r) = \min_{i \in \text{input}(r)} \left\lfloor \frac{\text{cap}(i)}{q_{ri}} \right\rfloor \times p_r$$

x 的最大需求量一定不会超过它的最大生产量加上库存。那么

$$\text{cap}(x) = \text{stock}(x) + \max_{r \in \text{recipes}(x)} \text{producible}(r)$$

这里我们采用的是 $\max$ 而不是求和，这可能是一个问题。对于下文的两种遍历方式来说，朴素遍历只挑一条配方，所以确实不必求和。但容量遍历则不一样，例如 $A \leftarrow B$, $A \leftarrow C$ 的情况下，假如 B 和 C 各有 5 个，那这样算出来的 $\text{cap}(A)$ 等于 5。如果之后有一个物品 T $x1 \leftarrow A \times 10$ ，就会被误认为无法合成。然而，贪心本就是不完备的，而且实验之后我们发现这样拒绝的内容并不多，所以我们依然保留这个方式。

3.3 朴素遍历

我们从目标出发进行逆后序遍历 (RPO)。

对于当前的物品 x ，如果它有亏空，也就是需要的量大于库存的量，那么我们就去找一个“好”的配方 r ，然后全情投入。我们采用的判别方式有两种：

- r 产出的 x 数量越多越好；
- r 需要的物品数量越少越好。

我们会用这两个判断方式分别遍历一次。不过不论哪一种，我们都要求配方的所有输入都是可达的。

这里的可达和可达性剪枝处计算的可达并不相同。在剪枝中，我们考虑的是未切环的完整子图，而这里则是在询问切出的 DAG 中是否可达。

3.4 容量遍历

这是朴素遍历的升级版。

我们不再为 x 选择一条最好的配方，而是按能合成的 x 的总数从大到小逐个使用 x 的生产配方，能做多少就做多少。我们把这个值称作配方的容量：

$$\text{craftable}(r) = \min_{i \in \text{input}(r)} \left\lfloor \frac{d}{q_{ri}} \right\rfloor \times p_r$$

这里的 d 是 x 的亏空，每合成完一条配方都会减少。

如果两条配方的容量相同，我们按照 Section 3.3 所说的两种方式来打破平局。

这可以解决一些朴素方法中的问题。如果朴素遍历选中了数值很大的转换，例如 $1x A \leftarrow 1000x B$ ，就会大幅度提高需求量，这也就是我们刚开始提到的结果差距七个数量级的来源。而基于容量的方法会发现这条路容量不够。

3.5 回退

我们会先使用最普通的 DFS 切环法，最后取 $\sum_r c_r x_r$ 最小的那个。如果普通视图一种可行解都没有找到，再用基于分级的切环重新跑一遍。如果全部失败了，就交给 CP-SAT。

这样求出来的解不一定收支平衡。在 Section 3.4 中，我们求出的只是乐观容量，而且传播过程中也可能因为按批取整而偏离实际。所以，我们必须再检查一次。

4 CP-SAT 求解

这个部分其实没什么太多说的，主要是调用 Google OR-Tools。

我们依然进行了一些优化。虽然每个数值都有 $[0, 2^{63})$ 的自然范围，但是对于 CP-SAT 来说，这个范围太大了，初值经常和最优解偏离十万八千里，甚至可以偏差七个数量级。为了减轻搜索的压力，我们会优先计算出一个更小的上限 cap 。

我们知道，如果使用普通的线性规划 (LP)，可以求出一个有理数域上的解。这时这个解自然可以作为下界估计，然而我们并不知道上界在哪里。所以，我们将这个值乘以 $\frac{9}{8}$ 作为第一次尝试的上界。如果无解，就持续翻倍上界，直到达到预先计算好的、保证中间结果不可能溢出的最大上界。

LP 是非常迅速的，在我们的问题规模下通常仅需数十毫秒，因此这种上界估算一般不会出现性能问题。

不过，在只合成一个目标物品的时候，LP 和 ILP 之间的差距通常也很大，而乘法能增加的并不够多。因此，如果 LP 太小，我们就转而增加一个固定的数值（这里选择的是 64），而不是乘以 $\frac{9}{8}$ 。

4.1 基于价值的剪枝【无法启用】

考虑我们规划的对偶。我们原本的目标是：在约束 $\sum_r A_{i,r} x_r \geq b_i$ 下，求出 $\sum_r c_r x_r$ 的最小值，其中 c_r 是简介里说的配方成本（普通配方为 1，批次级为批大小，tag 边为 0）。其中对于配方 $r: p \times o \leftarrow q_1 \times i_1, \dots$ ，有 $A_{o,r} = p$ ， $A(i_n, r) = -q_n$ ， b_i 是物品 i 的净需求（目标物为正，库存取负）。

它的对偶是：在 $\sum_i A_{i,r} y_i \leq c_r$ （每条配方一个约束，右边就是它的成本）下最大化 $\sum_i b_i y_i$ 。我们知道， y_i 可以理解为客户物品 i 的边际价值，也就是多得到一个 i 所需要付的成本。于是 r 的等效成本是 $(A^T y)_r = p y_o - \sum_n q_n y_{i_n}$ ，从直觉上来说，是它的工作量。这个值称作**折合成本**，但 r 本身执行一次的花费是 c_r ，与它的产出结构无关。二者之差

$$d_r = c_r - \left(p y_o - \sum_n q_n y_{i_n} \right) \geq 0$$

就是配方 r 亏掉的工作量，称作**缩减成本**，在 LP 最优处可以直接读出。

由对偶可行性，对任意可行解 x 都有 $\sum_r c_r x_r \geq l + \sum_r d_r x_r$ ，其中 l 是 LP 的最优值。设 u 是某个已知可行解的成本。若某个满足 $\sum_r c_r x_r \leq u$ 的可行解用到了配方 r （即 $x_r \geq 1$ ），则 $u \geq \sum_r c_r x_r \geq l + d_r x_r \geq l + d_r$ 。所以只要

$$d_r > u - l$$

就必有 $x_r = 0$ ：步数不超过 u 的解一定不会用到配方 r 。我们可以直接把这一列删掉。

更进一步地说，即使不能直接得到 $x_r \leq 0$ ，我们也可以得到这个结果，加快 CP-SAT 的处理速度：

$$x_r \leq \left\lfloor \frac{u - l}{d_r} \right\rfloor$$

实现上，LP 松弛本来就要算一次来估计 cap，我们顺便取出所有 d_r 。 u 则由一次**紧上界探测**给出：拿 $[l + \text{gap}]$ 当作 cap 跑一次 CP-SAT。如果它直接不可行，就说明 $u - l > \text{gap}$ ，方法不划算，跳过；否则拿到 u ，按上面的条件删列，再在删过的模型上以 u 为 cap 正式求解。返回的 x 会补回被删的列（取 0），所以对外仍然是一列一个配方。

在一些额外的测试之后，我发现这个东西反而会拖慢速度：最开始想要拿到 u 并不是一件容易的事情。文档后面也有一些基于 u 的想法，也一并都不启用。

5 针对多个零库存物品的推广

前面的剪枝只针对零库存的单个物品。我们自然想知道能否同时利用多个零库存物品，剪掉某个配方 R ，也就是说明 $x_R = 0$ 。

我们考虑这样的一组配方：

```
铜镝 x1 <- 铜锭 x3, 木棍 x2
铜粒 x1 <- 铜镝 x1
铜锭 x1 <- 铜粒 x9
```

如果这三个配方同时被用到，那就亏到姥姥家了：按照 1:9:1 的情况计算，我们亏掉了 18 个木棍和 26 个铜锭。但是如果按照 1:1:1 的比例计算，我们反而“赚到”了 8 个铜粒，尽管这仍然是亏的，不如直接通过铜锭合成来得多。

我们希望通过“打包配方”的形式，来用数学描述这个事实。

沿用 Section 4.1 中的记号： A 是以物品为行、以配方为列的收支矩阵，其中 $A_{i,r}$ 为配方 r 对物品 i 的净产出（产出取正、消耗取负）。可行解 x 满足 $Ax \geq b, x \geq 0$ 且 x 取整数，目标是最小化 $\sum_r c_r x_r$ 。

我们将上述的配方包记作向量 $z = [z_1, z_2, \dots, z_n]$ ，其中 z_i 是第 i 个配方在这个配方包里执行的次数。

引理. 若 $z \neq 0$ 是非负整数向量且 $Az \leq 0$ ，则任何满足 $x \geq z$ 的可行解 x 都不是最优解。

证明. $A(x - z) = Ax - Az \geq Ax \geq b$ ，同时 $x - z$ 非负，所以 $x - z$ 是可行解。它和 x 相比则减少了 $c^T z = \sum_r c_r z_r$ 的成本。 z 里一定含有被认证的那条真实配方 ($z_R = 1$)，而真实配方的 $c_R \geq 1$ ，其余 $c_r \geq 0$ ，所以 $c^T z > 0$ ，一定更优。

这个引理在说， z 这个配方包在最优解中一次都不能执行。那么，我们可以做出如下推论：

对于配方 R ，如果能找到某个正整数 k 并证明对所有的可行解 x 都有 $x_R \geq k \Rightarrow x \geq z$ ，且 $Az \leq 0$ ，则任何最优解都有 $x_R \leq k - 1$ 。当 $k = 1$ 时结论最强，这时我们可以直接剪掉配方 R 。

5.1 蕴含关系的证明

这个蕴含关系 $x_R \geq k \Rightarrow x \geq z$ 是最大的难点。既然是自动证明，我们不妨考虑一个形式证明系统，也就是一组推理规则。

维护一个下界向量 z ，初始值为 $z_R = k$ ，而其他的 $z_i = 0$ 。容易说明此时必然有 $Az \leq 0$ 。我们希望保证 $z \leq x$ 。我们将当前 z 对物品 i 的消耗量记作 $c_i(z) = \sum_r \max(-A_{i,r}, 0) z_r$ ，并将零库存的物品集合称作 G 。

R1. 唯一生产者的下界

如果只有一个生产物品 $i \in G$ 的配方 p ，并且生产 a 个，那么 $x_p \geq \left\lceil \frac{c_i(z)}{a} \right\rceil$ 。这是因为 i 的消耗量至少是 $c_i(z)$ ，而产量减去消耗量必须大于零。

我们可以把下界 z_p 提高到这个值，依然能满足 $z \leq x$ 。

在更新 x_p 后，我们自然会更新 x 对各种资源的消耗量，从而更新 x 的生产量，那么就可以继续使用其他基于生产量的规则了。

R2. 被支配的输入

假设物品 i, j 都在 G 中。如果生产 i 的每个配方 p 都以不小于 α 的比率消耗另一种资源 j ，则 $c_j(x) \geq \alpha c_i(z)$ 。这是对 R2 在生产者不唯一时的补充。

R3. 分支

如果前两条都不能处理，说明 i 有多条配方，需要一起产出需求。设 i 有 m 条配方 $p_1, \dots, p_m$ ，分别产出 $a_1, \dots, a_m$ 个。对任意可行解都有

$$\sum_j a_j x_{p_j} = \text{produced}_i(x) \geq \text{consumed}_i(x) \geq l_i$$

因此至少有一条配方满足 $a_j x_{p_j} \geq \frac{l_i}{m}$ ，也就是 $x_{p_j} \geq \left\lceil \frac{l_i}{ma_j} \right\rceil$ 。我们对每条配方分出一支：在第 j 支中把 z_{p_j} 提高到 $\left\lceil \frac{l_i}{ma_j} \right\rceil$ ，再继续用 R1、R2 传播。最后取所有分支算出的 z 的逐分量最小值，作为这一层的结果。这是因为，对任意满足 $x_R \geq k$ 的可行解 x ，一定存在某个 j 可以给出下界 $z^{(j)} \leq x$ 。而逐分量取最小值可以使得 z 不超过任何 $z^{(j)}$ ，所以它也 $\leq x$ 。所以，分支必然是有限的：每一次分支都至少把某条配方的下界抬高一点，而 z 的总量被预算限制。

将 R1 到 R3 展开成一棵搜索树；每个叶子给出一个对相应分支假设成立的下界，取所有叶子的逐分量最小值，就得到对每个满足 $x_R \geq k$ 的可行解都成立的下界 z 。最后检查 $Az \leq 0$ ，若成立则记录证书 (R, k, z, G) 。

在实测中，我注意到 R3 效果不佳，而过于耗时。所以我选择禁用 R3，仅使用 R1 和 R2 处理，效果反而更好。

5.2 预处理与查询

目前而言，我的想法是只保留 $k = 1$ ，因为这是最有用的。

预处理时，为每个配方 R 保存一组 (z, G, W) ： z 是上述得到的配方包， G 是零库存的物品集合，而 W 是 z 中配方所需的工作台集合。查询时，如果 $\text{supp}(z) = \{i \mid z_i \neq 0\}$ 落在可达子图内、 G 与有库存物品不交、 W 可用，即可删除 R 这条边。

5.3 利用配方的价值

对任意对偶可行的 $y \geq 0$ ，我们有 $A^T y \leq c$ 。仍然考虑缩减成本 $d = c - A^T y \geq 0$ ，那么对任意的 $z \geq 0$ 且 $Az \leq 0$ 有

$$\sum_r d_r z_r = d^T z = c^T z - y^T Az \geq \sum_r c_r z_r > 0$$

也就是说，每个处置环里一定含有缩减成本为正的列。这带来一个有用的改写：设 $H = \{r : d_r > u - l\}$ 是基于价值的剪枝所删去的列。如果能证明 $x_R \geq k$ 必然会用到 H 中的某个配方至少 t 次，那么自然有 $\sum_r c_r x_r \geq l + d_R k + t \min_{h \in H} d_h$ ；第二项 $d_R k$ 是执行了 k 次配方 R 导致的，而第三项 $t \min_{h \in H} d_h$ 则是执行了 t 次 H 中的配方导致的。这时，既然 $u \geq \sum_r c_r x_r \geq l$ ，自然有

$$x_R \leq k \leq \left\lfloor \frac{u - l - t \min_{h \in H} d_h}{d_R} \right\rfloor$$

比起之前的 $\left\lfloor \frac{u-l}{d_R} \right\rfloor$ 更紧了。

我们可以直接用上一节求出 z 的方法来求解，只需要把得到的 z 里属于 H 的配方次数加起来就是 t 了。

5.4 与线性规划变量上界的关系

上面的 $Az \leq 0$ 是一个纯组合的最优性论证，但它不总是强过线性规划。事实上，更强的界是直接求解线性规划

$$U_R = \max \left\{ x_R : Ax \geq b, \sum_r c_r x_r \leq u, x \geq 0 \right\}, \quad x_R \leq \lfloor U_R \rfloor.$$

我们知道 $u \geq \sum_r c_r x_r \geq l + \sum_r d_r x_r$ ，其中 $l = b^T y$ 是对偶目标。那么自然有

$$\sum_r d_r x_r < u - b^T y$$

然而，这是有理数。我们乘以一个 D 使得左边的每一个 d_r 都是整数：

$$\sum_r (D d_r) x_r \leq \lfloor D u - D b^T y \rfloor = D u - \lceil D b^T y \rceil$$

将 x_R 单独留在左边，移项之后就可以得到

$$x_R \leq \frac{u - \frac{\lceil D b^T y \rceil}{D} - \sum_{h \neq R} d_h x_h}{d_R}$$

这其实就是一阶 CG 切割。之前剪枝时的 $\left\lfloor \frac{u-l}{d_R} \right\rfloor$ 只是它的一个特例：不考虑其他项的亏损 $d_h x_h$ ，同时令 $D = 1$ ，就会得到这个结果。

6 孤岛消除

6.1 动机

你一定知道为大家所“津津乐道”的更新：Mojang 更新了铜块、斑驳的铜块、锈蚀的铜块、氧化的铜块、涂蜡的铜块、涂蜡的斑驳铜块、涂蜡的锈蚀铜块、涂蜡的氧化铜块等数百种排列组合（但是仍然没有竖半砖）。它们和普通铜块可以转换，所以和铜块这个基础形态一起被卷进同一个强连通分量，而它们自己却除了作为装饰品以外没有什么作用。

以此为引，直觉告诉我们：如果一块区域只通过一个出口 i 与外界相连，而它没法凭空产出 i ，那么它不可能出现在最优解里。

“与外界相连”不必是（无向图中的）割点。各种氧化程度的铜在通用机械中可以通过氧气转换，所以铜块不是割点。但循环没有净产出任何物品，只是在额外消耗氧气，所以它仍然不该出现在最优解里。所以我们给出这个比较通用的引理：

引理. 取一些不包含合成目标的物品 I ，记 G 为所有消费或生产 I 中物品的配方。设 I 中每个物品的库存都是 0，并存在一个出口物品 $i \notin I$ ，使得 G 中每个配方产出的物品都在 $I \cup \{i\}$ 中，也就是**对产出封闭**。这时，若不存在 G 上的配方向量 $z \geq 0$ ，使得

$$(Az)_i > 0 \wedge \forall j \in I. (Az)_j \geq 0$$

则任何最优解都满足 $\forall r \in G. x_r = 0$ 。

证明. 假设 x 是最优解，而 $z = x|_G$ 是 x 被限制在 G 上的那一部分。因为 G 包含了所有与 I 中物品相邻的配方， I 中物品的收支行只与 G 内的配方有关；又因为库存为 0，可行性给出 $(Az)_j = (Ax)_j \geq b_j = 0$ ($j \in I$)。根据条件， z 不能同时满足该条件与 $(Az)_A > 0$ ，那么自然有 $(Az)_i \leq 0$ 。令 $x' = x - z$ ：在 I 内 $(Ax')_j = 0 \geq b_j$ ；对于出口本身有 $(Ax')_i = (Ax)_i - (Az)_i \geq (Ax)_i \geq b_A$ ；而对于不在 $I \cup \{A\}$ 中的物品 h ，条件告诉我们 G 不会产出 h ，即 $(Az)_h \leq 0$ ，所以 $(Ax')_h \geq (Ax)_h \geq b_h$ 。于是 x' 可行且执行次数更少，矛盾。

出口 i 并不是本质的，它只是为了描述岛漏出的产出。若 I 自身就**对产出封闭**，也就是 G 中每个配方产出的物品都在 I 中，那么任取一个 I 外的物品作为出口 i ， G 都产不出它，判定条件自动成立，对偶解取 $y = 0$ 即可。真正被用到的是 G 在 I 之外没有净产出，而不是“只有一个出口”。

6.2 判定

值得注意的是，正产出环是不能用 Bellman-Ford 的方法计算的，这是因为合成图是 AND-OR 图，和前面无法使用 LT 算法计算**代价**支配的理由一样：一条配方需要它的全部输入，于是环的利润是若干配方共享子产物之后的结果，不能逐物品地用一个势函数刻画。注意这里被卡住的是“ G 能否净产出 A ”这个**判定**，而不是候选 I 的寻找： I 只关于可达性，而可达性在配方上是 OR，下面会看到一个普通的支配树就够了。

要证明不存在是很困难的。所以，我们不如将它转化为 Farkas 对偶形式，证明存在就简单了。它的对偶是：存在 I 上的 $y \geq 0$ 使得

$$\forall r \in G. \sum_{j \in I} y_j A_{j,r} + A_{A,r} \leq 0$$

所以，我们只需要以 $\sum_j y_j$ 为目标求解上述线性规划，得到任何一个可行的 y 即可。这样算出来的 y 是有理的，所以我们得乘以一个数去消除浮点误差，确认是否可行。（TODO：这种猜一个分母 D 再用 int128 验算的方法看起来好难受。真的没有别的方法了吗？）

实现上，我们先跑无向图的割点枚举，取每个连通块作为 I ；再取每个候选出口 A 所能支配的最大集合。后者就是依赖图 $x \rightarrow$ 生产 x 的配方的每个输入上 A 所支配的物品集合：不经过 A 就不能到达目标或库存物品的物品，自动满足输出条件。它能把只有输入的岛屿也一并收进来。

这个支配关系是普通有向图上的可达性，所以实现时给依赖图加一个连到目标和每个有库存的物品的虚拟根，然后跑一次 Lengauer-Tarjan。这时， A 的岛就是它在这棵树里的真子树。

出口为空的情形既不需要线性规划，也不需要枚举，而且有一个唯一的最大岛。记 B 为从目标和所有有库存的物品出发、在下面这条规则下闭包得到的最小集合：**若某条配方产出 B 中的物品，就把该配方的所有物品并入 B** 。 B 的补集就是对产出封闭的岛：某条配方若触及这个补集，就不可能产出 B 中的物品，否则闭包会把被它触及的那个物品也拉进 B 。反过来，若 I 避开目标与库存且对产出封闭，令 B' 为 I 的补集：任何产出 B' 中物品的配方都不能触及 I （否则它就产出了一个 I 外的物品），

所以它的所有物品都在 B' 里，即 B' 也满足这条闭包规则；而 B 是最小的，于是 B 包含于 B' ，即 I 包含于 B 的补集。所以 B 的补集是最大的岛。

6.3 不止一个出口的孤岛

我们已经说过出口 i 并不本质。对于多出口的岛 I ，如果出口集合是 O ，只需要满足对任意的 $o \in O$ 都没有 z 使得

$$(Az)_o > 0 \wedge \forall j \in I. (Az)_j \geq 0$$

就同样可以剪枝。所以，现在问题变成了我们到底如何找到这样的岛。

我们可以考虑先固定一个种子 s ，首先令 $I = \{s\}$ ，并记 $N(I)$ 为所有产出或消耗任意 I 中元素的所有配方。令 $O = \text{out}(N(I)) - I$ ，也就是 I 当前产出的不属于 I 的部分。如果对于某个 $o \in O$ 可以找到上述的 z ，那么 o 就必须加入 I ，直到遇到目标 T 或者 o 有库存的时候失败。如果一直都没有遇到，最终成型的闭包 G 就是我们想要的孤岛，并且无需额外证明。

对每一个物品都这么跑一遍太久了，而且结果无法复用：即使从 s 开始得到的闭包 G 失败了，也不意味着任何 $s' \in G$ 都会失败，同时也无法推出 s 的输入一定会失败。考虑这个例子：

```
X <- S
S + A <- Y
T + B <- A
Z <- X
```

如果目标为 T ，那么自然从 S 开始会失败，因为闭包会包含 T 。然而闭包已经包含了 X ，从 X 开始则不会失败。这是因为 S 产出了多个物品，其中牵扯到了 A ，但 X 则没有这个烦恼。同时， Y 产出了 S ，但是如果从 Y 开始，我们会发现闭包在 S 处就已经结束了，而 Y 本身就是一个孤岛。

实际上，这并不是由多产出所带来的。考虑：

```
S <- A
U <- S
V <- S
T <- V
```

目标同样为 T ，从 S 开始会失败，而 U 不会。这大概是因为这个闭包的扩张不是对称的。不过，这两个例子也说明了正向和反向的可达性分析都是孤岛消除的一个特例。

所以为了避免在求解之前花费太多时间，我在考虑启发式地跑：优先尝试处于二元环中的物品。一般这种压缩/分解或者互相转换的二元环比较多。

在尝试之后，即使这样似乎也只能多剪掉 0.2% 左右的边，完全不值得。所以这个想法被放弃了。

7 变体消除

7.1 动机

Minecraft 的 16 种染色旗帜都出现在同一个 tag 里，而这个 tag 的唯一用途是合成线；任意一种旗帜又能通过染料和“除它以外的旗帜”的 tag 来染色成别的旗帜，但这些旗帜最终还是只能合成线，所以这样的染色是没有用的。

因此，我们以这两个条件来定义**物品变体集合** V ：

- **不可区分**：存在一个 tag T ，使得 V 不在 V 中的消费者必定是 T 。
- **无正产出**： V 中配方产出的量不超过它的消耗。

那么， V 中的互相转换就可以删除。下面我们来看更加形式化的定义。

7.2 定义与引理

取一组物品 V 。记配方 r 对 V 的净贡献为 $C_V(r) = \sum_{v \in V} A_{v,r}$ 。如果一个 tag J 的成员 $M(J)$ 都在 V 中，就称它是 V -tag；反过来，如果 $V \subseteq M(J)$ ，那么称 J 是出口 tag。

我们把 V 中的配方叫做**内部配方**。具体的定义是这样的：如果满足下列两条之一，那么 r 就是内部配方。

- r 是产出 V 中物品的真实配方，而且它的全部产出都在 V 中，并且 $C_V(r) \leq 0$ 。我们把最后一个条件称为 r 对 V **中性**。
- r 是 V -tag 的成员边。

引理. 设目标 $t \notin V$ 。若

1. V 中物品的每个消费者要么是内部配方，要么是通向出口 tag 的成员边；
2. 每条内部实配方都对 V 中性，

则任何最优解都不执行内部配方。

证明. 设 x 可行， $z = x|_G$ 是它限制在内部配方集 G 上的部分。先看 V 的总量。由条件 2，内部实配方的 $C_V \leq 0$ ；对每个 V -tag J ，既然它的收支 $(Az)_J \geq 0$ ，那么消耗 J 的内部配方次数不超过成员边产出 J 的次数。我们又知道每条成员边恰好消耗 1 个 V 中的物品，所以成员边消耗的 V 不少于内部配方产出的 V ，即 $(Az)_V \leq 0$ 。所以 $(A(x-z))_V \geq (Ax)_V \geq 0$ 。□

这和孤岛消除不一样，就算内部能净产出外部物品也不会破坏引理。

7.3 寻找 V

我们在每次查询时都检查一次，而不是预处理，因为这个 pass 够快。

我们从每个可能作为出口的 tag t_0 开始，先把 V 设置成 $M(t_0)$ ，然后反复删去破坏条件的物品，直到不再变化。删除的规则是

- 物品被某条真实配方**直接**消费。
- 物品所在的某个 tag 既不是 V -tag，也不是出口 tag。
- 物品所在的某个 V -tag 还有存活的外部消费者。

最后，我们重新检查一遍 V 是不是满足我们引理。如果通过的话，就删掉所有内部配方，以及死掉的 V -tag 的成员边。

7.4 独占物品的推广

直觉. 我们不一定要找一个完整的 tag 消除。如果有一个物品 u 只在某个 tag 里使用，而合成它的所有配方都包含可以用于 tag 的物品，那么 u 其实就没有用了——直接把合成它的物品拿去满足配方会更好。

不过，这只是直觉，我们还要处理数量以及副产物之类的问题。下面我们形式化这个直觉。

考虑一个 tag T 。如果所有用到 u 的配方都是 T 的成员边，我们就称实物 u 被 T **独占**。

我们将 r 对 $M(T)$ 的净贡献记为 $C_{M(T)}(r)$ ，这个东西在 Section 7.2 里已经定义过一次了。简单理解就是“ r 产出 $M(T)$ 中物品的数量”。

引理. 如果实配方 r 的所有产出都被同一个 T 独占，且 $C_{M(T)}(r) \leq 0$ ，那么任何最优解都不会执行 r 。

证明与变体消除相同，只是把类换成整个 $M(T)$ 。设可行解为 x ，而 $z = x|_r$ 。我们可以重新指派 $x - z$ 中 T 的成员边，而 $M(T)$ 之外的物品收支只增不减。□

7.5 多出口的推广

普通的变体消除要求 V 存在一个唯一的出口 tag。实际上，这个条件并不必要。

动机. 在 ATM 中，有各种各样的玻璃：{Scratched, Connecting, Clear, ε } $\times$ 颜色 $\times$ {玻璃，玻璃板}（很遗憾，这些东西在游戏内部也没有中文翻译）。它们没有库存的时候，在合成图中一般都不会用到，直接替换为普通的玻璃和玻璃板就可以了。

所以，我们将这样的装饰物集合称作 D ，并考虑一个映射 $\sigma: D \rightarrow \text{Item} - D$ ，将 $\sigma(d)$ 映射到它的**本体**。举个例子，“连接的绿色染色玻璃板”将会被映射到“玻璃板”。我们可以将 σ 扩张到 $\text{Item} \rightarrow \text{Item}$ ，只要在原有的那些映射之外令 σ 恒等就可以了。

我们用 $\pi(v)$ 表示原始的收支向量 v 经过 σ 传播后的结果，也就是将 u 的部分转移到 $\sigma(u)$ 上。具体而言，

$$\forall i \notin D, \pi(v)_i = v_i + \sum_{j \in \sigma^{-1}(i)} v_j$$

而如果 i 在 D 中，那么 $\pi(v)_i = 0$ 。

这样，我们就可以提出消除的条件了。

引理. 如果对于任意 D 中的配方 r ，都存在一个向量 r^* ，使得 $\forall i \in D, (r^*)_i = 0$ （也就是 r^* 是 **D-free** 的），而且 $Ae_r = \pi(Ar^*)$ 并且 $c_{r^*} \leq c_r$ ，那么只要 D 不包含合成目标，最优解就可以不使用 D 。

这里的 c 是代价，而 A 则是收支矩阵。

证明. 设 x 是最优解，我们下面证明 $x' = \sum_r x_r r^*$ 也是最优解。

$$Ax' = A \sum_r x_r r^* = \sum_r x_r \pi(Ae_r) = \pi(Ax)$$

最后一步是因为 π 是线性的。既然 $Ax \geq b$ ，对任意的 $i \notin D$ ，都有

$$\pi(Ax)_i = (Ax)_i + \sum_{j \in \sigma^{-1}(i)} (Ax)_j \geq (Ax)_i \geq b_i$$

对 $i \in D$ ， $\pi(Ax)_i \geq 0$ 。既然目标不在 D 中，必然有 $b_i \leq 0$ ，所以 x' 是可行解。

而 $c_{x'} = \sum_r x_r c_{r^*} \leq \sum_r x_r c_r = c_x$ ，所以 x' 是最优解。

7.6 寻找 D 与 σ

我们没有任何名字可用，所以只能从图的结构里找。（TODO：其实按照名字进行启发式搜索也是可以接受的，而且 JNI 本来就会传递 .names.tsv 给模糊搜索的部分。要这么做吗？）

大部分配方转换都会包含等量转换的配方 $a \times b \leftarrow a \times u$ 。如果合成图里存在这条边，我们就在新的图 G 中连边 $u \rightarrow b$ ，而 G 的弱连通分量就是互相可以转换的轨道，其中的成员应当被 σ 映射成同一个物品。至于本体，我们选**参与合成的配方最多**的那个成员。一般而言，装饰物只会经过 tag 或者参与转换配方，只有本体才会被别的配方指名道姓地需要。

TODO：如果没有任何成员被外部配方直接需求，那么孤岛消除应该已经删掉它了才对。但是好像还留下来了一点？

不过矿物锭、块和粒之间的转换并非只有装饰作用。所以，为了排除这种情况，我们只在 $M(u) \subset M(b)$ 的时候才让 $\sigma(u) = b$ ；此外， u 的某个 tag 还得真的被配方消费。（TODO：或许需要先进行死物品消除）

在确定了一系列 u 之后，它不见得就是完整的 D 了，还可能有更多的物品。如果配方 r 的投影 $\pi(Ar)$ 还不可实现，就在同型配方里找一个 r' ，把 r' 与 $\pi(Ar)$ 不一致的那几个输入或输出折叠到本体。这可以解决“连接玻璃板”与“连接玻璃”折叠不进去的问题。

最后，任何一次折叠都要验证一次引理条件。

8 启动可达性

8.1 动机

求解器看到的约束只有收支条件：

$$\forall i. \text{produced}(i) - \text{consumed}(i) \geq b_i.$$

它把整个计划里每一次产出和消耗加在一起，所以如果存在环，它可以把之后产出的物品用来补充之前的需求。这意味着 CP-SAT 的输出可能是无法执行的。考虑这个环：

```
1x A <- 1x B
1x B <- 1x A
```

如果在没有库存的情况下想要 4 个 A，那么 CP-SAT 会认为 $x = [8, 4]$ 是合法的，这时 A 产出 4 个，B 产出 0 个。但从空库存出发，两个配方都无法执行。

所以 $Ax \geq b$ 只是可达性的必要条件， x 还需要一个合法的执行序列。

容易想到的一点是，如果这个环从已有库存不可达，那么自然可以剪去，我们自然也这么做了。然而，这忽略了一点：即使环可达，CP-SAT 也不会规划为它点火的路线，而是单纯地让环自己补足自己，因为这样通常更赚。

所以我们必须再检查一次求解器的计划。我们建立一张仅由配方构成的图，如果配方 r 消耗 p 的产出，那么就从 r 往 p 建一条边，然后对 SCC 缩点，形成 DAG。而 DAG 一定是可行的，所以只需要判断 SCC 可以利用库存和已经拿到的净产出启动循环。DAG 上的传播使用贪心的方式，而 SCC 则是普通的搜索，通常都很小。

它不具有完备性，因为贪心本就不完备，而且就算不能通过这个规划启动环，也还可能存在一个其他的可行规划。所以如果失败，我们只会返回 CYCLE_UNFULFILLED，也就是推测无解，但未证明。

8.2 启动切断

我们可以预先排除一部分环路情况。

考虑一组配方 R 。我们需要给每个环 R 一套布尔变量 s_r ，表示配方 r 是其中最先启动的那一个。写成约束的话：

$$x_r \geq s_r, \quad x_r \leq \text{dom}_r \sum_{p \in R} s_p, \quad \text{gross}(r, i) s_r \leq \text{stock}_i + \sum_{k \notin R, \text{net}_{ki} > 0} \text{net}_{ki} x_k$$

其中 $\text{gross}(r, i)$ 是配方 r 的输入量。在收支矩阵中，我们存储的是输出减去输入的结果，但在这里我们只需要真正的输入量。不过， net_{ki} 是配方 k 对 i 的净输出，的确是输出减去输入的。这是因为要启动 r ，我们输入量一定不能超过存货加其他配方的净输出。我们将最后一项称作 $\text{avail}(i)$ ，也就是物品 i 的可用量。

这里考虑的 R 虽然对任何配方集合都有效，但有用的主要是净产出一元环和二元环。

以及，单独一对配方上的约束经常能被其他配方满足，比如说 tag 的合成：配方 P 消耗 tag 节点、产出 tag 的全部成员，而每个成员各有一条 tag 边 T 把成员变回 tag 节点。这时 (P, T) 这个二元环上的约束总可以由另一个 tag 元素来满足，尽管实际上这样来的配方也执行不了。所以真正要切的组不是一对，而是互相消耗关系的连通分量，一般就是所有的 tag 元素。

目前的切断方式仍然是不完备的。考虑配方

```
A x64 <- B x16
B x1 <- A x1
```

如果库存仅有 8 个 A，约束是可以满足的，因为启动一轮只需要 1 个 A，但第二、第三轮攒不够 16 个 B，模型仍然会返回执行不了的计划。

8.2.1 虚拟物品

为了解决上面提到的问题，我们考虑了引入**虚拟物品**。以这套配方作为例子：

```
# 环
A x64 <- B x16
B x1 <- A x1
# 种子
A x1 <- C x1
B x1 <- D x1
```

这个环需要 16 个 A 或者 16 个 B 才能无限运行。所以，直觉告诉我们可以额外加入两个新的配方，它们的 cost 变为原本的 16 倍：

```
A x16 + S x1 <- C x16
B x16 + S x1 <- D x16
```

引入一个额外变量 $y = \min(x_1, x_2)$ ，也就是环整体所运行的次数。在 CP-SAT 中，这次的约束就是 $y > 0 \Rightarrow S > 0$ 。

然而，这样做其实是不对的。如果只有 7 个 A 和 9 个 B，其实这个循环也是可以启动的。所以，我们没有真正引入物品 S ，而是使用了一个类似的方式。

设配方为

$$A : p_1 \times f \leftarrow q_1 \times e, L$$

$$B : p_2 \times e \leftarrow q_2 \times f, T$$

如果环启动了至少一次，而且在第一次启动 A 之前，已经执行了 m 次 B ，那么就有 $mq_2 \leq \text{avail}(f)$ 。这是因为 $\text{avail}(f)$ 是外界能提供的 f 总量，而既然 A 没有启动，那么 f 只能从外界输入。

同样，为了能够启动 A ，我们必须有 $p_1 \leq \text{avail}(e) + mp_2$ 。既然我们无法得到 m 的值，就应该尝试消去 m 。这也算是一步 Fourier-Motzkin，总之可以得到

$$\frac{p_1 - \text{avail}(e)}{p_2} \leq m \leq \frac{\text{avail}(f)}{q_2}$$

既然所有东西都是正的，我们会有

$$q_2 p_1 \leq q_2 \text{avail}(e) + p_2 \text{avail}(f)$$

同样的，如果在第一次启动 B 之前已经启动了 m 次 A ，我们自然可以写出对称的式子：

$$q_1 p_2 \leq q_1 \text{avail}(f) + p_1 \text{avail}(e)$$

它们就是在 $\min(x_1, x_2) > 0$ 的情况下必须要满足的条件。