LLL格基约化算法与基于LLL的一种字符串哈希攻击方法

在算法竞赛语境中，“哈希”一般指Rabin-Karp算法，“n 哈希”指在比较两字符串时，各运行 n 个不同底数和模数的多项式的Rabin-Karp算法给出的结果，只有每队结果相等时才认为两字符串相等。为了讨论方便，我们假设字符集是$[0,\sigma)$中的整数，并且字符串的下标从右往左递增，且最后一位下标为 0。因此，设哈希用的底数是 b，模数是 p，原字符串为 s，则有 s 在参数如此的Rabin-Karp算法输出的哈希值为 $f(s,b,p)=(\sum_{0\le i<|s|}s_ib^i)\bmod p$。

若一个哈希是安全的，我们要求它有2-独立性和抗碰撞性。前者要求 $\Pr[f(s,b,p)=f(t,b,p)]\to\frac{1}{p^2},\text{when }(n\to+\infty)$，后者要求已知哈希算法的参数时构造使得哈希值碰撞的字符串是困难的。

在算法竞赛中，由于造测试数据的人通常不知道选手的程序所使用哈希算法的参数，而通常随机生成，有时候我们选用Rabin-Karp哈希算法只要求2-独立性而不要求抗碰撞，这使得一些选用常见参数的哈希能轻易地被碰撞攻击。（但如果参数在每轮运行时采用真随机数生成器生成，那么只能使用生日攻击。）

LLL格基约化算法原理简介

格的定义

对于线性无关的 n 维列向量组 $\mathbf{B}=\{\mathbf{b_1},\mathbf{b_2},\dots,\mathbf{b_k}\}$，其所有整系数线性组合构成的集合称为它的格，即

$$\mathcal{L}=\{\sum_i z_i\mathbf{b_i}\mid z_i\in\mathbb{Z}\}$$

称 $\mathbf{B}$ 为格 $\mathcal{L}$ 的一组基，或 $\mathbf{B}$ 张成格 $\mathcal{L}$。

定义格 $\mathcal{L}$ 的最短向量 $\lambda(\mathcal{L})$ 为 $\mathcal{L}$ 中所有向量中长度最短的那个向量。

最短向量问题（SVP）：给定 $\mathcal{L}$ 求出 $\lambda(\mathcal{L})$，给出准确解是NP困难的，但LLL算法可以找出近似解。

Gram-Schmidt正交化

读者可能在线性代数课本上已了解相关概念。

对于一组向量 $(\mathbf{b_1},\mathbf{b_2},\dots,\mathbf{b_n})$ 其正交化结果为：

$$\mathbf{b_i^*}:=\mathbf{b_i}-\sum_{j=1}^{i-1}\mu_{j,i}\mathbf{b_j^*}$$

其中 $\mathbf{b_i}$ 在 $\mathbf{b_j^*}$ 上的正交投影系数（注：参考文献[3]的“投影”两个字的说法是错的）$\mu_{j,i}=\dfrac{<\mathbf{b_i},\mathbf{b_j^*}>}{<\mathbf{b_j^*},\mathbf{b_j^*}>}$。

正交向量组满足任意向量在其他向量上的投影长度为0。此外，对于正交向量组张成的格，其最短向量 $\lambda(\mathcal{L})$ 有平凡解——直接取向量组中最短的即可，此外，近似正交（μ 很小）的向量组也可以如此——详见下文。

过程

我们首先考虑 $n=2$，即只有2个向量 $\mathbf{B}=(\mathbf{b_1},\mathbf{b_2})$ 的特殊情况，不妨设 $|\mathbf{b_1}|\leq|\mathbf{b_2}|$。

然后我们通过证明或画图可以发现，当 $|\mu_{1,2}|\leq\frac{1}{2}$ 时，一定有 $\mathbf{b}_1$ 为该格的最短向量。

但假如该向量组不满足上述条件呢?

我们注意到，运行Gram-Schmidt正交化后的向量组一定有所有向量正交因此可以直接取最短向量，但Gram-Schmidt正交化不能保证格的不变——因为各正交投影系数可能不是整数，得到的向量甚至可能不在原来的格中。对向量组中一个向量加上其他向量的整系数线性组合才能保证张成的格不变，这启发我们把Gram-Schmidt正交化过程的各 $\mu_{1,2}$ 舍入到最近整数后得到新的 $\mu_{1,2}$ 足够小的向量组。由此可见，我们的 $\mathbf{b}_1$ 应该选择较短的那个（否则交换两向量），保证 $\mu_{1,2}$ 较大以至于舍入造成的相对误差较小（可以证明，这使得运行完后一定有新的 $|\mu_{1,2}|\leq\frac{1}{2}$，满足了第一个条件，只要仍然 $|\mathbf{b_1}|\leq|\mathbf{b_2}|$ 第二个条件便可满足）。之后我们反复运行上述算法，直到满足条件然后再取平凡解即可。这即是高斯格基约化算法。

接下来是高位向量组的情况。我们推广2维时使用的使用近似正交化的思路。（注：参考文献[3]貌似在介绍LLL原理这里出锅了。）

另外，我们同样需要一个可以直接输出平凡近似解的算法的终止条件。仿照以上的定义，设常数 $\frac{1}{4}<y<1$，则当且仅当：

- $\forall 1\leq j<i\leq n$, 有$|\mu_{j,i}|\leq\frac{1}{2}$
- $\forall 1\leq i<n$, 有$y\cdot|\mathbf{b_i^*}|^2\leq|\mathbf{b_{i+1}^*}+\mu_{i,i+1}\mathbf{b_i^*}|^2$（即向量的长度近似递增）

我们称满足上述2个条件的向量组 $\mathbf{B}$ 为LLL既约的。

对第二个条件进行放缩：

$$y\cdot|\mathbf{b_i^*}|^2\leq|\mathbf{b_{i+1}^*}+\mu_{i,i+1}\mathbf{b_i^*}|^2\leq|\mathbf{b_{i+1}^*}|^2+\mu_{i,i+1}^2|\mathbf{b_i^*}|^2<|\mathbf{b_{i+1}^*}|+\frac{1}{4}|\mathbf{b_i^*}|^2$$

从而有：$\forall 1\leq j<i\leq n, |\mathbf{b_j^*}|^2\leq(\frac{4}{4y-1})^{i-j}|\mathbf{b_i^*}|^2$

又由于在正交化时有$\mathbf{b_i}=\mathbf{b_i^*}+\sum_{j=1}^{i-1}\mu_{j,i}\mathbf{b_j^*}$，且 $\mathbf{b_j^*}$ 两两正交，展开并求和得

$$|\mathbf{b_i}|^2\leq(\frac{4}{4y-1})^{i-1}|\mathbf{b_i^*}|^2\leq(\frac{4}{4y-1})^{h-1}|\mathbf{b_h^*}|^2,\forall h>i$$

最终可以得到：$|\mathbf{b_1}|^2\leq(\frac{4}{4y-1})^{n-1}\lambda_\mathbf{1}$

其中 $\lambda_\mathbf{1}$ 是格中最短非零向量的长度的平方。

取 $y=\frac{1}{4}+(\frac{3}{4})^{\frac{n}{n-1}}$，则有：

$$|\mathbf{b_1}|^2\leq(\frac{2}{\sqrt{3}})^{n-1}\lambda_\mathbf{1}$$

也就是说，只要找到了这样的一组基，我们就有了格中最短向量的近似解。

因此，我们仿造高斯格基约化算法得到LLL算法的流程：

1. Build: 运行Gram-Schmidt正交化记录其 $\mu_{i,j}$。
2. Reduction：将 $\mu_{i,j}$ 舍入后更新 $\mathbf{B}$，容易证明更新后一定满足第一个条件。
3. Swap：找到任意一个 $i\in[1,n)$ 不满足第二个条件，然后交换 $\mathbf{b_i},\mathbf{b_{i+1}}$。如果无法找到这样的 i，则终止算法并输出 $\mathbf{B}$（一定是LLL既约基），否则第一个条件可能被破坏然后重新回到第一步Build。

设 n 为输入矩阵大小，V 为 $\mathbf{B}$ 中元素值域。可以证明，算法一定在迭代有限次（进一步证明为 $O(n^2 \log V)$ 次）后终止，并且时间瓶颈在迭代时的运算，因此总共需要 $O(n^4 \log V)$ 次理论运算，而若要防止精度损失算法过程中的每个数需要占用 $O(n^2 \log V)$ 位空间，且时间复杂度为 $O(n^{5+\epsilon} \log^{2+\epsilon} V)$。

LLL算法通常用于求解线性同余方程式的一组较小解：

设有一 m 元线性同余方程 $(\sum_i a_i x_i) \bmod p = 0$（在算法竞赛中 p 通常为一 10^9 左右的质数），我们可以希望在可接受的时间内得到绝对值较小的一组 x 的值，这正好贴合LLL得到近似最小向量的功能，因此我们构造向量组：

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ Ma_1 & Ma_2 & Ma_3 & \dots & Ma_m & Mp \end{bmatrix}$$

对其运行LLL算法即可。

因为我们想让 $kp - \sum_i a_i x_i = 0$，该式形如 $(a_1, a_2, \dots, a_m, p)$ 的等于0的线性组合，而用LLL可能能消除到0。为了记录消除时用的线性组合（p我们不关心，因此除外），我们在 $\vec{a}$ 上添加一单位矩阵，做列变换时它也会记录等效的线性组合。而为了增大最下行消为 0 的可能性，我们将其乘以一个大整数 M 使得LLL会将其权重看得更大。在运行后，由于我们不一定要找最小解，因此我们可以记录所有最后一行为0并且其余行符合我们范围要求的列向量。

此外，该方法还可以扩展到方程数量多于1个的方程组上，假设有 n 个方程组成的 m 元线性方程组，第 i 个方程形如 $(\sum_j a_{i,j} x_j) \bmod p_i = 0$。同样令

$$\mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \dots & 1 & 0 & 0 & \dots & 0 \\ Ma_{1,1} & Ma_{1,2} & Ma_{1,3} & \dots & Ma_{1,m} & Mp_1 & 0 & \dots & 0 \\ Ma_{2,1} & Ma_{2,2} & Ma_{2,3} & \dots & Ma_{2,m} & 0 & Mp_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ Ma_{n,1} & Ma_{n,2} & Ma_{n,3} & \dots & Ma_{n,m} & 0 & 0 & \dots & Mp_n \end{bmatrix}$$

对其运行LLL算法再取后 n 行等于 0 的列向量即可。

攻击字符串哈希算法[3]

目标：构造两字符串 s 和 t，使得输入到算法后其哈希值碰撞。

我们首先考虑单哈希的情况。若两长度相等的字符串 s, t 要产生哈希值碰撞，则有 $f(s, b, p) = f(t, b, p)$，即 $(\sum_{0 \le i < |s|} (s_i - t_i) b^i) \equiv 0 \ (\bmod p)$，并且由于输入数据的字符集的限制，我们必须有 $-\sigma < s_i - t_i < \sigma$，也就是 $s_i - t_i$ 的绝对值要足够小。也就是要求一线性同余方程的较小解，根据上述LLL算法，我们只需要构造向量组：

$$\mathbf{B}=\begin{bmatrix}1&0&0&\dots&0&0\\0&1&0&\dots&0&0\\0&0&1&\dots&0&0\\\vdots&\vdots&\vdots&\ddots&\vdots&\vdots\\0&0&0&\dots&1&0\\Mb^0&Mb^1&Mb^2&\dots&Mb^{|s|-1}&Mp\end{bmatrix}$$

对其运行LLL算法即可。

但即使哈希函数的参数随机生成，由于生日攻击的原理，单哈希在数据范围较大的随机数据下就极易被攻击，因此通常使用若干个参数不同的哈希函数的值的有序对 $((\sum_{0\le j<|s|}s_jb_i^j)\bmod p_i,1\le i\le n)$ 作为哈希值，即多哈希。

由于多哈希的碰撞是若干个哈希函数作用于两串均得到相等的值（$(\sum_{0\le j<|s|}(s_j-t_j)b_i^j)\equiv 0\ (\bmod p_i\ ),1\le i\le n$），而LLL同样可用于解线性方程组（详见上文），因此我们构造

$$\mathbf{B}=\begin{bmatrix}1&0&0&\dots&0&0&0&\dots&0\\0&1&0&\dots&0&0&0&\dots&0\\0&0&1&\dots&0&0&0&\dots&0\\\vdots&\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&0\\0&0&0&\dots&1&0&0&\dots&0\\Mb_1^0&Mb_1^1&Mb_1^2&\dots&Mb_1^{|s|-1}&Mp_1&0&\dots&0\\Mb_2^0&Mb_2^1&Mb_2^2&\dots&Mb_2^{|s|-1}&0&Mp_2&\dots&0\\\vdots&\vdots&\vdots&\ddots&\vdots&\vdots&\vdots&\ddots&\vdots\\Mb_n^0&Mb_n^1&Mb_n^2&\dots&Mb_n^{|s|-1}&0&0&\dots&Mp_n\end{bmatrix}$$

并运行LLL算法即可。若 $n=5$，参数设置为 $|s|=32,M=2\sigma,y=0.994$ 表现较优秀。（注：参考文献[3]给出的程序中有Bug：变量`sz`和乘方函数参数等部分变量类型为`int`导致精度损失，不过迭代次数只增加了约0.84%）。

参考文献

[1] 谢梓涵. 浅谈格算法在整系数多项式分解中的应用 [J]. IOI中国国家候选队论文集, 2024.

[2] 程思元. 随机化和近似算法选讲 [C]. NOI2024冬令营讲义, 2024.

[3] 佚名. 如何有理有据地卡古典字符串哈希 [EB/OL]. 2024. https://www.luogu.com/article/artstb0e