第五届（2020 年）
全国高校密码数学挑战赛
参赛论文

赛题三：子集和问题

战队名称 covteam

学　　校 西华大学

学　　院 计算机与软件工程学院

专　　业 信息安全

参赛选手 杨佳立,白泽兴,赵之奕

指导教师 周洁

二零二零六月

摘　要

子集和问题又称背包问题，其已经被证明为 NP-完全问题，在 P≠ NP 的基本假设下，不可能找到求解子集和问题的多项式时间算法。但是，精心设计的求解算法往往能降低求解的计算复杂度。维数越高的子集和问题越难求解，根据维数的不同，本论文采用三种方法对赛题进行求解。对于维数 n=40 到 n=80 的低维子集和问题，通过对原始的 Schroeppel-Shamir 算法进行改进，将赛题给定的汉明重量 k 应用到算法中，大大降低原始算法的时间和空间复杂度。此算法主要基于生日攻击碰撞的思想，其对于某个集合只需要遍历一半的搜索空间即可找到解，因此对于低维度的背包问题能准确的得出答案并且其时间和空间复杂度较低。对于维数 n=100 到 n=160 的中维子集和问题，采用基于给定汉明重量 k 的 HGJ (Howgrave-Graham and Joux)改进算法，降低原始 HGJ 算法的时间和空间复杂度。此算法的思想和 Schroeppel-Shamir 改进算法大体相同，只是其构造搜索空间时采用组合进行构建，该构造方法可以减少搜索空间的大小，提高找到碰撞的机率。利用 HGJ 改进算法，在可接受的时间和空间消耗中得到了中维子集和问题的解。对于 n=180 及以上的高维子集和问题，论文采用格基规约的方法，通过构造向量组，把子集和问题转化为最短向量问题，找到的最短向量即是子集和问题的解。

关键词：子集和问题，背包问题，格基规约，LLL 归约，最短向量问题

目 录

第一章　　引言

1.1 赛题分析

定义子集和问题为：给定一个 n 元的正整数集合 $A=\{a_1,a_2,\ldots\ldots,a_n\}$ 和同为 n 元的 0-1 集合 $X=\{x_1,\ldots\ldots,x_n\}$ 以及 S，其中 A 集合已知，S 已知，求解 X 使得对于 $i=0,\ldots,n-1,x_i=0$ 或 1，满足 $a_0x_0+\ldots+a_{n-1}x_{n-1}=S$。

X 集合中的 x_i 的值为 0，1 则确定了相应的 a_i 是否需要放到背包里。通俗一点来说，S 相当于背包总重量，a_i 相当于备选装入背包的值，而 x_i 则决定 a_i 是否应该装入背包中。解决背包问题就是要从全部的 a_i 值中选出其中的一部分使其加起来等于 S。x_i 为 0 时，表示该 a_i 值并未被选中，为 1 时表示该 a_i 值被选中。题中总共有九个挑战，从 40 维到 200 维，每 20 维一个等级。每个等级根据密度 d 分为四个题目，并且密度越高，a_i 与 S 的比特位数越小。

1.2 解决方法

求解子集和问题的方法大致可以分为确定性算法和概率性算法两类。确定性算法在理论上一定可以求得子集和问题的解，但一般是非多项式时间算法，时间和空间复杂度高，仅适用于维数较低的子集和问题。概率性算法则是以一定的概率得到子集和问题的解，效率高但不一定能得到解。

根据子集和问题的维数 n，我们将赛题分为三类：低维子集和问题（n<=80），中维子集和问题（80<n<=160），高维子集和问题(n>180)。

对于低维子集和问题，采用确定性的 Schroeppel-Shamir 改进算法[5]。其算法首先对背包进行整数划分，随机分为四个子背包并完全枚举四个子背包的所有可能的和值，并通过 four-way-merge 算法从中有效且确定地寻找答案，此外如果题目已知汉明重量可以进一步缩小范围从而提高找到解的可能性。

对于中维子集和问题，采用确定性的 Howgrave-Graham-Joux 改进算法[6]。其算法更关注子背包的构造情况，如果假定汉明重量为 $\lfloor n/2 \rfloor$（可证明此时已经是最复杂的情况）那么我们从 $\lfloor n/2 \rfloor$ 中构建所有的组合情况在其中选四个组合数，那么这样同样可以得到答案并且如此构造时间和空间复杂度相比 Schroeppel-Shamir 都要减少很多。

对于高维子集和问题，采用概率性的 BKZ 格基规约算法。因为确定性算法对于低维问题并不需要太多时间，并且总能找到答案，但缺点是时间复杂度太高，此外，空间复杂度对于高维度的问题同样无法承受。而格基规约算法则是通过背包元素来构造向量矩阵，之后运用格基规约算法对其进行规约将问题转化为 SVP 问题，并使得得到的归约基进尽可能可正交

化，而当得到的归约基只包含 0 和 1 时，就是我们的背包的解。该算法只使用于高密度的子集和问题，在使用规约算法时，我们主要采用 BKZ 算法

1.3 结题结果

我们求出了第一级到第七级挑战的 0-1 解，第八级和第九级挑战的非 0-1 解。按照赛题的计分原则，总得分为：

200+400+600+800+1000+1200+1400+(1600/2+22-162)+(1800/2+25-223)=6962，

其中 22 为第八级挑战题目 8.4 给定的汉明重量，162 为题目 8.4 所得解的欧几里德范数平方；25 为第九级挑战题目 9.4 给定的汉明重量，223 为题目 9.4 所得解的欧几里德范数平方。每一级挑战的题目及答案如下。

1. **挑战难度一：题目 1.1.**
   n=40，d=0.8，k=5，s=2038015735279982.
   本题的解向量为:

$$\mathbf{X}=[0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0]$$

即：a1+a19+a20+a28+a33=s.

2. **挑战难度二：题目 2.4.**
   n=60，d=1.1，k=7，s=75355236542335061.
   本题的解向量为:

$$\mathbf{X}=[0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0]$$

即：a8+a23+a36+a52+a53+a56+a58=s.

3. **挑战难度三：题目 3.1.**
   n=80，d=0.8，k=10，s=380874853941482352011927516300 7.
   本题的解向量为:

$$\mathbf{X}=[0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,1,0,0,1,0,0,0,0,0,0,0]$$

即：a6+a10+a15+a16+a35+a58+a62+a67+a68+a72=s.

4. **挑战难度四：题目 4.1.**
   n=100，d=0.8，k=12，s=21432689612707032947115881054702567 0819.
   本题的解向量为:

$$\mathbf{X}=[0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0]$$

0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

即：a2+a5+a8+a11+a24+a30+a54+a66+a75+a84+a86+a94=s.

5. **挑战难度五：题目 5.4.**

   n=120，d=1.1，k=15，s=526745943177944311098874708308626 6.

   本题的解向量为:

X=[0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

即：a2+a9+a25+a29+a30+a33+a34+a36+a42+a45+a61+a91+a100+a102+a118=s.

6. **挑战难度六：题目 6.4.**

   n=140，d=1.1，k=17，s=162409110295251424612219935908925736 4515.

   本题的解向量为:

X=[1, 0, 0, 0, 1, 0, 0, 0, 1, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

即：a0+a4+a8+a9+a15+a20+a38+a59+a68+a69+a73+a77+a87+a99+a124+a125+a128=s.

7. **挑战难度七：题目 7.4.**

   n=160，d=1.1，k=20，s=3543386607007586113441061652401373628984475 64.

   本题的解向量为:

X=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 0, 0, 0, 1,

0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1, 1, 0, 0, 0, 0, 1, 0, 1, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

即:a15+a27+a30+a33+a39+a43+a50+a51+a52+a61+a85+a88+a98+a100+a101+a106+a108+a112+a122+a150=s.

8. **挑战难度八：题目 8.4.**

n=180，d=1.1，k=22，

s=1007692368920981506120960449198658867909375080033347

本题的整数解向量为:

X= [0, 1, 2, 2, 0, 0, 0, 0, −1, 0,
−2, 4, −1, 1, −1, 1, 0, 0, −1, 0,
1, 0, −1, 2, 0, −2, 2, 1, −1, 0,
−1, 1, 2, 1, 2, 0, −1, −1, 0, 1,
2, 1, 0, −1, 2, 0, −1, 0, −2, 0,
2, 0, −1, 1, −1, 2, 3, 0, 0, −1,
−1, 0, 0, −2, 2, 1, 1, 0, 2, −2,
0, −1, 1, 1, 0, 1, 2, −1, 0, 1,
0, −1, 0, 0, 1, 0, 0, 0, 0, 2,
0, 1, −2, 0, 1, 0, 0, 0, 0, 1,
−1, 0, 0, −1, −2, 0, 0, 2, 1, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, −2, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, −1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, −1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

即：a1+2×a2+2×a3−a8−2×a10+4×a11−a12+a13−a14+a15−a18+a20−a22+2×a23−2×a25 + 2×a26+a27−a28−a30+a31+2×a32+a33+2×a34−a36−a37+a39+2×a40+a41−a43+2×a44−a46−2×a48+2×a50−a52+a53−a54+2×a55+3×a56−a59−a60−2×a63+2×a64+a65+a66+2×a68−2×a69−a71+a72+a73+a75+2×a76−a77+a79−a81+a84+2×a89+a91−2×a92+a94+a99−a100−a103−2×a104+2×a107+a108+a115−2×a122−a135−a162+a174 = s.

解的欧几里德范数平方为$\|x\|_2^2$=162。

9. **挑战难度九：题目 9.4.**

n=200，d=1.1，k=25，

s=34194585308509120772031528249247006704390121910932930512

本题的整数解向量为:

X= [1, 0, 0, 1, −1, 3, 0, 3, −1, 0,
2, 0, −3, 0, 0, 1, 1, 1, 0, 2,
1, 1, 0, −1, 4, −1, −2, −2, 0, −1,

$$
\begin{matrix}
-4, & 0, & 0, & -1, & 2, & 0, & 0, & 2, & 2, & -1, \\
0, & 2, & -2, & 0, & 0, & -1, & -2, & 2, & -1, & 2, \\
-1, & 0, & 1, & 0, & -2, & 0, & 2, & 0, & -5, & -1, \\
2, & 0, & 1, & 2, & 0, & 1, & 1, & 0, & 0, & -1, \\
0, & 1, & -1, & 1, & 2, & 3, & 1, & 2, & 1, & 0, \\
0, & 0, & 1, & 1, & 0, & 2, & 1, & -1, & 0, & 0, \\
1, & 0, & 0, & 0, & 1, & -1, & -1, & 0, & -2, & 2, \\
0, & 0, & 0, & 0, & 3, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & -1, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, \\
0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0, & 0]
\end{matrix}
$$

即：$a0+a3-a4+3\times a5+3\times a7-a8+2\times a10-3\times a12+a15+a16+a17+2\times a19+a20+a21-a23+4\times a24-a25-2\times a26-2\times a27-a29-4\times a30-a33+2\times a34+2\times a37+2\times a38-a39+2\times a41-2\times a42-a45-2\times a46+2\times a47-a48+2\times a49-a50+a52-2\times a54+2\times a56-5\times a58-a59+2\times a60+a62+2\times a63+a65+a66-a69+a71-a72+a73+2\times a74+3\times a75+a76+2\times a77+a78+a82+a83+2\times a85+a86-a87+a90+a94-a95-a96-2\times a98+2\times a99+3\times a104-a132=s.$

解的欧几里德范数平方为$\|x\|_2^2=223$。

第二章　　子集和问题

2.1 子集和问题研究现状

子集和问题或 0-1 背包问题是一个著名的 NP-完全问题[4]，其经常被用于构建加密体系。近年来，出现了许多针对于背包加密体系的攻击算法，根据背包的特性，其攻击算法可分为两大类。一类是通用算法，该算法适用于任何背包问题。从 Schroeppel-Shamir 的生日攻击算法[5]，到更加优化的 Howgrave-Graham-Joux 的通用算法[6]和 Becker 的改进的背包通用算法[7]。这类算法的优点是，其降低了破解所需花费的时间，是一种确定性的算法，并且可以适用于任何的背包问题。缺点是，因该算法的时间复杂度和空间复杂度是指数时间的，所需要的时间和空间呈指数增长，当背包的维数太大时，无法在有效时间和空间内进行求解。另一类是算法是基于格基规约来进行求解。Coster 等人证明了[8]，当背包的密度 $d = n/\log_2 \max\{a_1,\ldots,a_n\} < 0.9408$ 时，子集和问题的解以极大的概率可以通过寻找某个格上的最短向量来进行求解。此外，Ping 等人证明了[9]，低汉明重量、高密度的背包问题可以确定性的规约到格上的最短向量问题。Schnorr 等人[10]提出了基于格基规约的算法来求解子集和问题，其构建了基于 LLL 格基规约和基于 BKZ 格基规约的求解子集和问题的算法，并对其进行了改进。此后，Schnorr 等人[11]又提出了使用随机化的 BKZ 归约来求解密度接近于 1 的子集和问题。周娜[1]阐述了，通过合理的构建向量矩阵，使用 LLL 算法或 BKZ 算法可以有效的解决低密度的子集和问题。此外，近年来又出现了一些新的解法，例如王蔚[2]等人提出的求解子集和的快速算法。王保仓等人[3]提出的基于联立丢番图逼近的子集和问题的启发式求解算法等。

2.2 Schroeppel-Shamir 算法

2.2.1 Schroeppel-Shamir 算法

Schroeppel-Shamir 算法[5]是 Schroeppel 和 Shamir 于 1981 年提出的一个背包算法，可以在 $\tilde{O}\left(2^{n/4}\right)$ 的内存和在 $\tilde{O}\left(2^{n/2}\right)$ 的时间内解决 n 个元素的一般整数背包问题。该算法是对 Horowitz 和 Sahni 的生日算法[12]的改进，使其能够应用于背包问题。生日算法的目是按照如下的等式重写背包:

$$\sum_{i=1}^{\lfloor n/2 \rfloor} \varepsilon_i a_i = \mathrm{S} - \sum_{i=\lfloor n/2 \rfloor+1}^{n} \varepsilon_i a_i$$

其中所有的 ε 都是 0 或 1。因此，为了解决背包问题，我们先构造集合 $\mathrm{S}^{(1)}$，其包含前 n/2 个元素的的所有可能和，$\mathrm{S}^{(2)}$ 是目标值 S 减去最后一个[n/2]元素的所有可能和得到的集合。

通过寻找两个集合之间的碰撞，我们就能找到背包问题的所有解。该算法可以在$\tilde{O}\left(2^{n/2}\right)$的时间内通过排序和查找碰撞完全计算如上两个集合。在文献[5]中，Schroeppel 和 Shamir 指出，为了找到这些碰撞，不需要存储$S^{(1)}$和$S^{(2)}$集合，而使用(基于堆或 Adelson-Velsky 和 Landis 树)优先级队列，这样就只需要使用$\tilde{O}\left(2^{n/4}\right)$的内存。在实际应用中，由于对大型优先队列的需求，使得 Schroeppel-Shamir 算法难以实现和优化。实际上，使用大型优先级队列要么会在内存使用中引入额外的因素，要么会有额外的缓存信息。因此，我们希望完全避免采用优先队列。为了做到这一点，我们使用基于文献[13]的算法，该算法解决了从 4 个不同的列表中寻找 4 个元素的问题。但从理论角度来看，对于一些特殊背包需要更多的内存。这个新算法的基本思想是，我们首先选择一个在$2^{(1/4-\varepsilon)n}$附近的模数，随后我们把背包表示为如下等式：

$$\sigma_L^{(1)}+\sigma_R^{(1)}\equiv S-\sigma_L^{(2)}-\sigma_R^{(2)}(\mathrm{mod}\,M)$$

在对于第一和第二个数列的时候，我们找到是否存在这样一个σ_M满足:

$$\sigma_M=\left(\sigma_L^{(1)}+\sigma_R^{(1)}\right)\mathrm{mod}\,M=\left(S-\sigma_L^{(2)}-\sigma_R^{(2)}\right)\mathrm{mod}\,M$$

对此我们可以通过循环 0 到 M-1 的所有的值实现。首先我们对第二个数列进行排序并全部模 M，随后我们循环每一个第一个数列中的值减去σ_M然后在第二个列表中寻找是否存在这样的差值。如果找到我们则把它们压入 S 中，于是我们能够构建一个数列 S，由第一个列表和第二个列表中的某值构成且满足其和模掉 M 后等于σ_M。对于列表 3 和列表 4 我们采用同样的方法，不过这次是寻找目标 S 减去数列中的和值后，其是否在列表 4 中存在。Schoreppel-Shamir 算法的具体步骤见图 2-1。

2.2.2 子集和低维（n<=80）问题求解

在求解低维的子集和问题时，我们采用 Schoreppel-Shamir 算法，并根据赛题中所给出的汉明重量 k 对算法进行了优化。在改进算法中，我们采用组合数来对背包元素进行遍历，但给出了限制条件，即只有满足在所给的汉明重量的集合才能被选取出来进行碰撞。该算法适合求解 80 维及以下的背包问题，当维数达到 100 时，其时间复杂度和空间复杂度便难以接受。考虑到时间效率问题，我们采用 C++来实现该算法，因所给数据均为大整数，我们选择导入 GMP 库来进行编程。最终，通过不断的尝试来选取时间最优的模数 M 的值来完成解题。具体实现代码见附录“Schroeppel-Shamir 改进算法代码”。

赛题的时间和空间消耗情况如下：对于 40 维（题目 1.1）在可忽略的内存和 0.1s 内完成了计算；对于 60 维（题目 2.4），在可忽略的内存和 59s 内完成了计算；对于 80 维（题目 3.1），我们在 intel® 至强® E5-2403 处理器上，使用 20 个线程并行计算，在 2150MB 的内存和 4h 24m 25s 时间内完成了计算。

Require: Four input lists $(\mathcal{S}_L^{(1)}, \mathcal{S}_R^{(1)}, \mathcal{S}_L^{(2)}, \mathcal{S}_R^{(2)})$, size n, target sum $\mathcal{T}$
Require: Memory margin parameter: ε
  Let M be a random modulus in $[2^{(1/4-\varepsilon)\,n}, 2 \cdot 2^{(1/4-\varepsilon)\,n}]$
  Create list $\mathcal{S}_R^{(1)}(M)$ containing pairs $(\mathcal{S}_R^{(1)}[i] \bmod M, i)$ where i indexes all of $\mathcal{S}_R^{(1)}$
  Create list $\mathcal{S}_R^{(2)}(M)$ containing pairs $(\mathcal{S}_R^{(2)}[i] \bmod M, i)$ where i indexes all of $\mathcal{S}_R^{(2)}$
  Sort $\mathcal{S}_R^{(1)}(M)$ and $\mathcal{S}_R^{(2)}(M)$ by values of the left member of each pair
  Create empty list *Sol*
  for σ_M from 0 to $M-1$ **do**
    Empty the list $\mathcal{S}^{(1)}$ (or create the list if $\sigma_M = 0$)
    for i from 1 to size of $\mathcal{S}_L^{(1)}$ **do**
      Let $\sigma_L^{(1)} = \mathcal{S}_L^{(1)}[i]$ and $\sigma_t = (\sigma_M - \sigma_L^{(1)}) \bmod M$
      Binary search first occurrence of σ_t in $\mathcal{S}_R^{(1)}(M)$
      for each consecutive (σ_t, j) in $\mathcal{S}_R^{(1)}(M)$ **do**
        Add $(\sigma_L^{(1)} + \mathcal{S}_R^{(1)}[j]), (i,j))$ to $\mathcal{S}^{(1)}$
      end for
    end for
    Sort list $\mathcal{S}^{(1)}$ by values of the left member of each pair
    for k from 1 to size of $\mathcal{S}_L^{(2)}$ **do**
      Let $\sigma_L^{(2)} = \mathcal{S}_L^{(2)}[k]$ and $\sigma_t = (\mathcal{T} - \sigma_M - \sigma_L^{(2)}) \bmod M$
      Binary search first occurrence of σ_t in $\mathcal{S}_R^{(2)}$
      for each consecutive (σ_t, l) in $\mathcal{S}_R^{(2)}(M)$ **do**
        Let $\mathcal{T}' = \mathcal{T} - \sigma_L^{(1)} - \mathcal{S}_R^{(2)}[l]$
        Binary search first occurrence of $\mathcal{T}'$ in $\mathcal{S}^{(1)}$
        for each consecutive $(T, (i,j))$ in $\mathcal{S}^{(1)}$ **do**
          Add (i,j,k,l) to *Sol*
        end for
      end for
    end for
  end for
  Return list of solutions *Sol*

图 2-1 The Algorithm of Schroeppel and Shamir

2.3 Howgrave-Graham-Joux 算法

2.3.1 Howgrave-Graham-Joux 算法

Howgrave-Graham-Joux（HGJ）算法[6]是 Howgrave-Graham 和 Joux 于 2010 年提出的通用背包算法。该算法可以解决通用的背包问题，即形如：

$$S = \sum_{i=1}^{n} \grave{o}_i a_i$$

其汉明重量和密度可任意给定，我们假定其汉明重量为 ℓ。

定义集合 $S_{\lceil \ell/2 \rceil}$ 为 $\lfloor \ell/2 \rfloor$ 或 $\lceil \ell/2 \rceil$ 个背包元素的部分和。则存在一对集合 $S_{\lceil \ell/2 \rceil}$ 中的元素 (σ_1, σ_2)，其满足 $S = \sigma_1 + \sigma_2$。实际上，根据 S 可能的分解，存在许多的这样的配对。当 ℓ 为偶数时，存在这样的分解的数量 $N_n = (\frac{\ell}{\ell/2})$，而当 ℓ 为奇数时，其数量 $N_n = 2(\frac{\ell}{(\ell/2)/2})$。

HGJ 的基本思想就是关注于集合 $S_{\lceil \ell/2 \rceil}$，在其中会找到一个我们所需要的解。我们首先在 N_n 附近选择一个素数 M，和一个随机的元素 R 模 M。则有一定的可能性存在一个 S 的分解 $\sigma_1 + \sigma_2$，其满足 $\sigma_1 \equiv R(\mathrm{mod}\, M)$ 并且 $\sigma_2 \equiv S - R\ (\mathrm{mod}\, M)$。为了发现这样的一个分解，我们需要构建出 $S_{\lceil \ell/2 \rceil}$ 的两个子集，其各自含有同余于 M 的 R 和 R-S。这样的集合的预期大小为：

$$\frac{\binom{n}{\lceil \ell/2 \rceil}}{M} \approx \frac{\binom{n}{\lceil \ell/2 \rceil}}{\binom{\ell}{\lceil \ell/2 \rceil}} = \tilde{O}\left(2^{0.3113n}\right)$$

该指数是假定最坏的时候，即 $\ell \approx n/2$ 时所得出的。一旦这两个子集 $S^{(1)}_{\lceil \ell/2 \rceil}$ 和 $S^{(2)}_{\lceil \ell/2 \rceil}$ 构建好后。我们就去寻找在 σ_1 和 $S - \sigma_2$ 之间的一个碰撞，其中 σ_1 属于 $S^{(1)}_{\lceil \ell/2 \rceil}$，$\sigma_2$ 属于 $S^{(2)}_{\lceil \ell/2 \rceil}$。接下来，使用排序和查找算法即可在 $\tilde{O}\left(2^{0.3113n}\right)$ 时间内完成通用算法的求解。

2.3.2 子集和中维（80<n<=160）问题求解

在求解中维的子集和问题时，此前的 Schroeppel-Shamir 算法因空间复杂度较高，普通计算机已无法完整的执行该算法。之后，我们找到了 HGJ 的算法，其算法内部也使用了 Schroeppel-Shamir 算法，但对其进行了一定改进。正如上文所述，其通过构造子集然后再进行碰撞，将其时间复杂度降低到了 $\tilde{O}\left(2^{0.3113n}\right)$，可惜的是，其算法只对一轮碰撞有效，其构建部分子集和时并不能完全的实现元素全覆盖，因此该算法能够找出子集和解的概率并不高。但我们采取了 HGJ 算法的思想，通过构建组合数，发现通过构造组合数去寻找道碰撞的几率极高，程序往往不需要遍历整个空间。实际运行过程中我们寻找完成了 160 维以内子集和问题的计算。考虑到时间效率问题，HGJ 改进算法同样采用 C++实现。其具体实现代码见附录“Howgrave-Graham-Joux 改进算法代码”。

赛题的时间和空间消耗情况如下：对于 100 维（题目 4.1），我们选择了 M= 2^16，它在 intel® Xeon® E5-240（以下均使用此配置）上使用了小于 896MB 的内存和 0.28s 时间内完成了计算；对于 120 维（题目 5.4），我们选择了 M = 2^18，它在 10 个线程中在 1496MB 的内存和 3h 10m 25s 时间内完成了计算；对于 140 维（题目 6.4），我们选择了 M = 2^20，它在 20 个线程中在 3584MB 内存和 3h 54m 21s 时间内完成了计算；对于 160 维（题目 7.4），我们选择了 M= 2^29，它在 15 个线程中在 201728MB 内存和 1h 4m 48s 时间内完成了计算。

2.4 LLL 与 BKZ 算法

2.4.1 LLL 与 BKZ

1982 年，Lenstra A. K.， Lenstra H. W. 和 Lovász L. [14]提出了经典的 LLL 格基规约算法。之后，Lagarias 和 Odlyzko[15]表明把特定的背包问题（密度为 0.64）转换成格中的最短向量问题（SVP）。紧接着，他们描述了使用 LLL 格基规约算法来求解该问题的思想。十九世纪二十年代初，Lagarias-Odlyzko 的方法被 Coster 等人[8]进一步将能被解决的背包问题的背包密度提高到 0.94，利用低密度子集和攻击就能在格中能够寻找到最短非零向量，那么算法的一个简单修改将解决几乎密度小于 0.94 的所有问题。

我们使用的格基规约算法来源于 Shoup 上的 NTL 库中的函数。我们使用库中的 chnorr-Euchner 基规约算法，使用了其中的 LLL 算法。在更新文档中，作者说明了这里的 LLL 不是最原始的 LLL，而是 LLL 衍生出的 deep LLL 算法。在 deep LLL 算法中，当条件

$$\delta\left\|b_{i-r}^{*}\right\|^{2} \leq\left\|b_{i}^{*}\right\|^{2}+u_{i, i-r}^{2}\left\|b_{i-r}^{*}\right\|^{2}$$

成立时，$k = k + 1$继续下面两个相邻向量之间的比较；如果不成立，则需要将当前向量与更前面的一个向量进行比较，直到$k = 1$时为止。在文献[10]中，实际使用的是L^3的改进算法$L^3FP(L^3FP-\text{reduction in floating point arthmetic})$，这样可以避免在改变格基时的正确性。算法中，将$b_1 \ldots b_m$定义为在格上的一组基，在设置好初始平台 k 之后，计算出$\mu_{k,1},\ldots,\mu_{k,k-1}$(具体计算方法见附录)，随后计算出$c_i$。第二步为 size-reduction，进一步将b_i进行缩减，最后通过一定条件进行交换。deepLLL 相较于一般的 LLL 算法在交换环节的次数更少，所以速度更加快。具体L^3FP算法步骤见图 2-2.

BKZ(Base Korkin Zolotarev)是分块的 deepLLL 算法。它将标准 LLL 算法中的交换步骤用一个分组约减步骤代替，该方法得到的约减基被成为 KZ 约减基。格中一组基$L(v_1, v_2, v_3, \ldots, v_n)$可以被称为 KZ 约减基应该当且仅当满足下面三条性质:

(1)v_1是格 L 中的最短非零向量。

(2)对于 i = 2,3...,n,向量v_i使得$\pi_{i-1}(v_i)$为$\pi_{i-1}(L)$中最短非零向量。

(3)对于所有的$1 \leq i \leq j \leq n$，都有$\left|\pi_{i-1}(v_i) \cdot \pi_{i-1}(v_j)\right| \leq \frac{1}{2}\left\|\pi_{i-1}(v_i)\right\|^2$。

通常，KZ 约减基的第一个向量就是 SVP 问题的一个解。

Algorithm L^3FP, L^3-reduction in floating point arithmetic.

INPUT $b_1, \ldots, b_m \in \mathbb{Z}^n$ (a lattice basis), δ with $\frac{1}{2} < \delta < 1$.

1. (*initiation*) $k := 2$, $F :=$ false
   (k is the *stage*. The following values are available upon entry of stage k: $\mu_{i,j}$ for $1 \leqslant j < i < k$ and $c_i = \|\hat{b}_i\|^2$ for $i = 1, \ldots, k-1$)
   FOR $i = 1, \ldots, m$ DO $b'_i := (b_i)'$
2. WHILE $k \leqslant m$ DO
   (computation of $\mu_{k,1}, \ldots, \mu_{k,k-1}$, $c_k = \|\hat{b}_k\|^2$)
   $c_k := = \|b'_k\|^2$, IF $k = 2$ THEN $c_1 := \|\hat{b}'_1\|^2$
   FOR $j = 1, \ldots, k-1$ DO
   IF $|\langle b'_k, b'_j \rangle| < 2^{-\tau/2} \|b'_k\| \, \|b'_j\|$ THEN $s := \langle b_k, b_j \rangle'$ ELSE $s := \langle b'_k, b'_j \rangle$
   $\mu_{k,j} := (s - \sum_{i=1}^{j-1} \mu_{j,i}\mu_{k,i}c_i)/c_j$
   $c_k := c_k - \mu_{k,j}^2 c_j$
3. (size-reduction of b_k)
   FOR $j = k-1, \ldots, 1$ DO
   IF $|\mu_{k,j}| > \frac{1}{2}$ THEN
   $\mu := \lfloor \mu_{k,j} \rceil$
   IF $|\mu| > 2^{\tau/2}$ THEN $F_c :=$ true
   FOR $i = 1, \ldots, j-1$ DO $\mu_{k,i} := \mu_{k,i} - \mu\mu_{j,i}$
   $\mu_{k,j} := \mu_{kj} - \mu$, $b_k := b_k - \mu b_j$, $b'_k := (b_k)'$
   END if $|\mu_{k,j}|$
   IF F_c THEN [$F_c :=$ false, $k := \max(k-1, 2)$, GOTO 2]
4. (swap b_{k-1}, b_k or increment k)
   IF $\delta c_{k-1} > c_k + \mu_{k,k-1}^2 c_{k-1}$
   THEN [swap b_k, b_{k-1} swap b'_k, b'_{k-1}, $k := \max(k-1, 2)$]
   ELSE $k := k+1$

OUTPUT $b_1, \ldots, b_m$ (a basis that is L^3-reduced with δ).

图 2-2 The Algorithm of L^3FP

改进了L^3FP 的 deep LLL 算法是将L^3FP的第四步交换的步骤做出如下修改。

New Step 4 (deep insertion of b_k).
$c := \|b'_k\|^2$, $i := 1$
WHILE $i < k$ DO
IF $\delta c_i \leqslant c$
THEN [$c := c - \mu_{k,i}^2 c_i$, $i := i+1$]
ELSE [$(b_1, \ldots, b_k) := (b_1, \ldots, b_{i-1}, b_k, b_i, \ldots, b_{k-1})$
rearrange the b'_j accordingly
$k := \max(i-1, 2)$, GOTO 2]
END while
$k := k+1$

图 2-3 The Algorithm of deep LLL

初始时，需要使用题目中的信息来构造出一个矩阵，将矩阵的每一行堪称一个向量，一共 n+1 个向量，构成格的一个基。参考文献[1]中构建矩阵的方式，矩阵总共 n+1 行，主对角线上除最后一个元素为 S 外，其余全部为 2。最后一列除最后一个元素为 S 外，其余为b_n。最后一行除最后一个元素为 S 外，其余元素均为 S。

$$B=\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \\ v_{n+1} \end{bmatrix}=\begin{bmatrix} 2 & 0 & 0 & \cdots & 0 & a_1 \\ 0 & 2 & 0 & \cdots & 0 & a_2 \\ 0 & 0 & 2 & \cdots & 0 & a_3 \\ \vdots & \vdots & \vdots & 2 & 0 & \vdots \\ 0 & 0 & 0 & \cdots & 2 & a_n \\ 1 & 1 & 1 & \cdots & 1 & S \end{bmatrix}$$

将矩阵构造完成之后，如果将矩阵最后一列的最后一行乘以 Constan 倍，能更有希望找到格中最短向量。在使用该算法时，可以使用 Hadamard 比率来评价一个矩阵是否是已经足够被规约，Hadamard 取值范围在(0,1]之间，当该值越接近于 1，说明规约出的格基越优质。找到其中最短的向量,

$$t=\left(2x_1-1,2x_2-1,\ldots 2x_n-1\right)$$

其中 t 为最短向量，而 $x_1 \ldots x_n$ 为所需要求的 0,1 解。

常见格基规约算法除了的 LLL 算法外，还有 BKZ(Block Korkin Zolotarev reduction)算法。其相比与 LLL 算法有了更近一步的优化，主要思想就是对向量分块来进行规约，当分块越小时，其规约速度越快，但其规约后格基的规约效果较差。我们可以通过调整其分块大小来规约出较高质量的基向量，当期分块大小较大时，将会有很大的概率得到只包含 0 和 1 的基向量，此时的基向量便是我们子集和问题的解向量。

2.4.2 子集和高维（n>160）问题求解

当面对高维的子集和问题时，之前的 Schroeppel-Shamir 算法与 Howgrave-Graham-Joux 算法因时间复杂度和空间复杂度过高，无法进行求解。因此，我们采用格基规约的方法来尝试求解。由于 BKZ 的规约效果通常比 LLL 的规约效果好，因此我们在解题时，主要是使用的 BKZ 算法来进行求解。通过用背包元素构建恰当的矩阵来进行规约，调整 BKZ 的分块大小来保证能规约出 01 基向量的同时保证时间小。在实际运行中，我们们的格基采用如下方式构建，其中 Δ 我们定义它大于汉明重量 k。

$$\mathbf{B}=\begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_{n-1} \\ \mathbf{b}_n \end{pmatrix}=\begin{pmatrix} 1 & 0 & \cdots & 0 & \Delta\mathbf{a}[0] & \Delta \\ 0 & 1 & \cdots & 0 & \Delta\mathbf{a}[1] & \Delta \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \Delta\mathbf{a}[n-1] & \Delta \\ 0 & 0 & \cdots & 0 & -c\Delta & -k\Delta \end{pmatrix}$$

如此一来我们得到了 n+1 个线性无关向量，如果我们把解 m 表示为 $(m[0],\ldots,m[n-1],0,0)$，那么我们一定有 $\pm\mathbf{m}\in L$。遗憾的是，当我们求解高维的子集和问题时，比赛即将结束，但我们通过格基规约，规约出了非 01 解向量的其他整数解向量。对于 180 维(赛题 8.4) 我们进行了分区大小为 21 的 BKZ 规约，对于 200 维（赛题 9.4）我们进行了分区大小为 24 的 BKZ 规约，在可忽略的内存内分别求出范数平方分别为 162 和 233 的解。

第三章　　结论

本文基于 Schroeppel-Shamir（Schroeppel and Shamir）改进算法、HGJ（Howgrave-Graham-Joux）改进算法和格基规约算法，成功解决了 160 维以内的子集和问题。通过实验发现维数较低 Schroeppel-Shamir 算法可以有效且快速的完成计算,如果知道汉明重量已知，改进的 HGJ 算法更为高效。通过理论分析，我们还得到了解决 180 维同样存在极高的概率找到答案，但是由于时间关系和计算资源的限制，最后我们未能求解。此外对于高维问题，目前最有效的方法是把该问题转为 SVP 问题，然后使用规约算法进行求解。但是此方法求解存在概率性，虽然空间复杂度上大大降低但是由于格基参数的不确定性，仍然需要较多的时间去尝试。所以对于构建一个时间复杂度和空间复杂度小，能够解决高维的子集和问题的算法，仍值得我们继续进行研究。

第四章　　参考文献

[1] 周娜. 格基规约相关算法的研究[D]. 深圳大学, 2017.
[2] 王蔚, 邱伟星. 求解子集和问题的快速算法[J]. 南京邮电大学学报: 自然科学版, 2012, 32(6): 92-95.
[3] 王保仓, 卢珂. 基于联立丢番图逼近的子集和问题启发式求解算法[J]. 密码学报, 2017, 4(5): 498-505.
[4] 周福才, 徐剑. 格理论与密码学. 北京:科学出版社.
[5] van Emde Boas P. Another NP-complete problem and the complexity of computing short vectors in a lattice[J]. Tecnical Report, Department of Mathmatics, University of Amsterdam, 1981.
[6] Schroeppel R, Shamir A. A T=O(2^n/2), S=O(2^n/4) algorithm for certain NP-complete problems[J]. SIAM journal on Computing, 1981, 10(3): 456-464.
[7] Howgrave-Graham N, Joux A. New generic algorithms for hard knapsacks[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2010: 235-256.
[8] Becker A., Coron J., Joux A., Improved generic algorithms for hard knapsacks, in Advances in Cryptology-EUROCRYPT 2011, Tallinn, Estonia, May 15-19, 2011 (LNCS, 6632), pp. 364-385.
[9] Coster M.J., Joux A., LaMacchia B.A., et al., Improved low-density subset sum algorithms, Computational Complexity, 1992, 2, (2), pp. 111-128.
[10] Yuan Ping, Baocang Wang, Shengli Tian, Yuehua Yang, Genyuan Du, Deterministic lattice reduction on knapsacks with collision-free properties, IET Information Security, 2018, 12(4), pp. 375-380.
[11] Schnorr C P, Euchner M. Lattice basis reduction: Improved practical algorithms and solving subset sum problems[J]. Mathematical programming, 1994, 66(1-3): 181-199.
[12] Schnorr C P, Shevchenko T. Solving Subset Sum Problems of Densioty close to 1 by" randomized" BKZ-reduction[J]. IACR Cryptol. ePrint Arch., 2012, 2012: 620.
[13] Horowitz E, Sahni S. Computing partitions with applications to the knapsack problem[J]. Journal of the ACM (JACM), 1974, 21(2): 277-292.
[14] Chose P, Joux A, Mitton M. Fast correlation attacks: An algorithmic point of view[C]//International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2002: 209-221.
[15] Lenstra A K, Lenstra H W, Lovász L. Factoring polynomials with rational coefficients[J]. Mathematische annalen, 1982, 261(ARTICLE): 515-534.
[16] Lagarias J C, Odlyzko A M. Solving low-density subset sum problems[J]. Journal of the ACM (JACM), 1985, 32(1): 229-246.

第五章　　谢辞

在本文的撰写过程中，周洁老师作为我们的指导老师，她治学严谨，学识渊博，视野广阔，为我们营造了一种良好的学术氛围。正是由于她在百忙之中多次为我们解答疑惑，并为本文的撰写提供了许多中肯而且宝贵的意见，本文才得以成型。另外在实际解题过程中，中国科学院软件所给与了我们宝贵的计算资源，因为有了他们的帮助我们的想法才能得以实现并最终解决问题。

在此向中国科学院软件所的同学表以衷心的谢意！此外向周洁老师的无可挑剔的敬业精神、严谨认真的治学态度、深厚的专业修养和平易近人的待人方式表示深深的敬意！

第六章　　附录

6.1 Schroeppel-Shamir 改进算法代码

```
#include <iostream>
#include <gmp.h>
#include <vector>
#include <random>
#include <string>
#include <ctime>
#include <algorithm>
#include <time.h>
#include <thread>
#include <stdlib.h>
#include <fstream>
#include <unistd.h>
using namespace std;
#define PBSTR "||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||"
#define PBWIDTH 60
typedef long long int ll;
typedef unsigned long int unint;
const unint N ;
const unint Hamingweight;
const int threadNumber;
const char *target_str;
bool result = false;
struct QuarterKnapsack
{
    mpz_t sum;
    unint weight;
};
struct ModuleKnapsck
{
    unint sum;
    unint index;
};
struct IndexPair
{
    unint i;
    unint j;
};
class Solution
{
public:
    mpz_t sum;
    unint sl1_index;
    unint sr1_index;
```

```
44          Solution() { mpz_init(this->sum); };
45          ~Solution();
46  };
47  Solution::~Solution()
48  {
49          // mpz_clear(this->sum);
50          // mpz_realloc2(this->sum, 0);
51  }
52  vector<QuarterKnapsack> init(int l, int r, vector<mpz_t *> elements)
53  {
54          unint last, lent = r - l + 1;
55          vector<QuarterKnapsack> slist;
56          for (ll i = 0; i < pow(2, lent); i++)
57          {
58                  unint count = 0, value = i, HMW = 0;
59                  mpz_t sum;
60                  mpz_init(sum);
61                  while (value > 0)
62                  {
63                          last = value & 0x01;
64                          if (last == 1)
65                          {
66                                  mpz_add(sum, sum, *elements[r - count]);
67                                  HMW++;
68                          }
69                          count++;
70                          value = value >> 1;
71                  }
72                  if (HMW > Hamingweight)
73                  {
74                          continue;
75                  }
76                  QuarterKnapsack oneKapk;
77                  mpz_init(oneKapk.sum);
78                  mpz_set(oneKapk.sum, sum);
79                  oneKapk.weight = HMW;
80                  slist.push_back(oneKapk);
81                  mpz_clear(sum);
82          }
83          return slist;
84  }
85  vector<unint> binarySearch(vector<ModuleKnapsck> &slist, unint x)
86  {
87          ll l = 0, r = slist.size() - 1, mid;
88          vector<unint> ans;
89          while (l <= r)
90          {
91                  mid = (l + r) / 2;
92                  if (slist[mid].sum == x)
```

```
93              {
94                  for (ll i = mid - 1; i > -1; i--)
95                  {
96                      if (slist[i].sum == x)
97                      {
98                          ans.push_back(slist[i].index);
99                      }
100                     else
101                     {
102                         break;
103                     }
104                 }
105                 ans.push_back(slist[mid].index);
106                 for (ll i = mid + 1; i < slist.size(); i++)
107                 {
108                     if (slist[i].sum == x)
109                     {
110                         ans.push_back(slist[i].index);
111                     }
112                     else
113                     {
114                         break;
115                     }
116                 }
117                 return ans;
118             }
119             else if (slist[mid].sum > x)
120             {
121                 r = mid - 1;
122             }
123             else
124             {
125                 l = mid + 1;
126             }
127         }
128         return ans;
129 }
130 vector<IndexPair> binarySearchSol(vector<Solution> &slist, mpz_t x)
131 {
132         int l = 0, r = slist.size() - 1, mid;
133         vector<IndexPair> ans;
134         IndexPair solpair;
135         while (l <= r)
136         {
137             mid = (l + r) / 2;
138             if (mpz_cmp(slist[mid].sum, x) == 0)
139             {
140                 for (ll i = mid - 1; i > -1; i--)
141                 {
```

```
142                         if (slist[i].sum == x)
143                         {
144                             solpair.i = slist[i].sl1_index;
145                             solpair.j = slist[i].sr1_index;
146                             ans.push_back(solpair);
147                         }
148                         else
149                         {
150                             break;
151                         }
152                     }
153                     solpair.i = slist[mid].sl1_index;
154                     solpair.j = slist[mid].sr1_index;
155                     ans.push_back(solpair);
156                     for (ll i = mid + 1; i < slist.size(); i++)
157                     {
158                         if (slist[i].sum == x)
159                         {
160                             solpair.i = slist[i].sl1_index;
161                             solpair.j = slist[i].sr1_index;
162                             ans.push_back(solpair);
163                         }
164                         else
165                         {
166                             break;
167                         }
168                     }
169                     return ans;
170                 }
171                 else if (mpz_cmp(slist[mid].sum, x) == 1)
172                 {
173                     r = mid - 1;
174                 }
175                 else
176                 {
177                     l = mid + 1;
178                 }
179             }
180 }
181 bool cmp_mod(ModuleKnapsck x, ModuleKnapsck y)
182 {
183     return x.sum < y.sum;
184 }
185
186 bool cmp_sol(Solution x, Solution y)
187 {
188     return (mpz_cmp(x.sum, y.sum) < 0);
189 }
190 void algorithm3(vector<QuarterKnapsack> &sl1, vector<QuarterKnapsack> &sr1,
```

```
    vector<QuarterKnapsack> &sl2,
191                         vector<QuarterKnapsack> &sr2, \
192                         vector<ModuleKnapsck> &sr1m, vector<ModuleKnapsck> &sr2m,
193                         mpz_t target,ll M, ll Ml, ll Mr, \
194                         int id, ofstream &record)
195 {
196     clock_t start, end;
197     start = clock();
198     cout << "algorm3 started" << endl;
199     ll step = 0;
200     // cout << Ml << ' ' << Mr << endl;
201     for (unint m = Ml; m <= Mr; m++)
202     {
203         step++;
204         if (step % 10 == 0)
205         {
206             double percentage = ((m - Ml) * 1.0) / (Mr - Ml);
207             int val = (int)(percentage * 100);
208 
209             cout << '\n'
210                  << "thread: " << id << ' ' << val << "% " << m << ' ' << Ml <<
    '/' << Mr << endl;
211             if (step % 500 == 0){
212                 record << '\n'
213                        << "thread: " << id << ' ' << val << "% " << m << ' ' << Ml
    << '/' << Mr << endl;
214             }
215 
216         }
217         vector<Solution> s;
218         for (unint sl1_index = 0; sl1_index < sl1.size(); sl1_index++)
219         {
220             mpz_t tmp;
221             mpz_init(tmp);
222             mpz_ui_sub(tmp, m, sl1[sl1_index].sum);
223             vector<unint> return_list = binarySearch(sr1m, mpz_fdiv_ui(tmp, M));
224             mpz_clear(tmp);
225 
226             for (int i = 0; i < return_list.size(); i++)
227             {
228                 unint sr1_index = return_list[i];
229                 unint HMW = sl1[sl1_index].weight + sr1[sr1_index].weight;
230                 if (HMW <= Hamingweight)
231                 {
232                     mpz_t sum;
233                     mpz_init(sum);
234                     mpz_add(sum, sl1[sl1_index].sum, sr1[sr1_index].sum);
235                     Solution sol;
236                     mpz_set(sol.sum, sum);
```

```
                        sol.sl1_index = sl1_index;
                        sol.sr1_index = sr1_index;
                        s.push_back(sol);
                        mpz_clear(sum);
                    }
                }
            }
            // gmp_printf("%Zd\n", s[i].sum);
            sort(s.begin(), s.end(), cmp_sol);
            for (int sl2_index = 0; sl2_index < sl2.size(); sl2_index++)
            {
                mpz_t tmp;
                mpz_init(tmp);
                mpz_sub_ui(tmp, target, m);
                mpz_sub(tmp, tmp, sl2[sl2_index].sum);
                vector<unint> return_list = binarySearch(sr2m, mpz_fdiv_ui(tmp, M));
                mpz_clear(tmp);
                for (int i = 0; i < return_list.size(); i++)
                {
                    unint sr2_index = return_list[i];

                    unint HMW = sl2[sl2_index].weight + sr2[sr2_index].weight;
                    if (HMW > Hamingweight)
                    {
                        continue;
                    }
                    mpz_t tt;
                    mpz_init(tt);
                    mpz_sub(tt, target, sl2[sl2_index].sum);
                    mpz_sub(tt, tt, sr2[sr2_index].sum);

                    vector<IndexPair> return_sol = binarySearchSol(s, tt);
                    mpz_clear(tt);
                    for (int sol_index = 0; sol_index < return_sol.size(); sol_index++)
                    {
                        FILE *file;
                        file = fopen("./outT", "w");
                        end = clock();
                        cout << "\rsolved in " << (double)(end - start) / CLOCKS_PER_SEC << endl;
                        mpz_out_str(file, 10, sl1[return_sol[sol_index].i].sum), fputc('\n', file);
                        mpz_out_str(file, 10, sr1[return_sol[sol_index].j].sum), fputc('\n', file);
                        mpz_out_str(file, 10, sl2[sl2_index].sum), fputc('\n', file);
                        mpz_out_str(file, 10, sr2[sr2_index].sum);
                        fclose(file);
                        result = true;
                        return;
```

```
283                         }
284                     }
285                 }
286                 for (int i = 0; i < s.size(); i++)
287                     mpz_clear(s[i].sum);
288                 s.clear();
289         }
290 }
291 void shamir(vector<mpz_t *> elements, mpz_t target)
292 {
293         int quarter = elements.size() / 4;
294         cout << "qater :" << quarter << endl;
295         vector<QuarterKnapsack> sl1 = init(0, quarter - 1, elements);
296         cout << "int 1 " << sl1.size() << endl;
297         vector<QuarterKnapsack> sr1 = init(quarter, 2 * quarter - 1, elements);
298         cout << "int 2 " << sr1.size() << endl;
299         vector<QuarterKnapsack> sl2 = init(2 * quarter, 3 * quarter - 1, elements);
300         cout << "int 3 " << sl2.size() << endl;
301         vector<QuarterKnapsack> sr2 = init(3 * quarter, elements.size() - 1, elements);
302         cout << "int 4 " << sr2.size() << endl;
303         double ll = pow(2, N / 4.0), rr = pow(2, (N / 4.0) + 1);
304         default_random_engine e(time(0));
305         uniform_real_distribution<double> u(ll, rr);
306         int M = (int)u(e);
307         cout << "Mod: " << M << endl;
308         cout << "init module" << endl;
309         vector<ModuleKnapsck> sr1m;
310         for (int i = 0; i < sr1.size(); i++)
311         {
312             ModuleKnapsck _t;
313             _t.sum = mpz_fdiv_ui(sr1[i].sum, M), _t.index = i;
314             sr1m.push_back(_t);
315         }
316         vector<ModuleKnapsck> sr2m;
317         for (int i = 0; i < sr2.size(); i++)
318         {
319             ModuleKnapsck _t;
320             _t.sum = mpz_fdiv_ui(sr2[i].sum, M), _t.index = i;
321             sr2m.push_back(_t);
322         }
323         cout << "sorting" << endl;
324         sort(sr1m.begin(), sr1m.end(), cmp_mod);
325         sort(sr2m.begin(), sr2m.end(), cmp_mod);
326         cout << "sort completed" << endl;
327         int ml = 0, mr = M / threadNumber;
328         // algorithm3(sl1, sr1, sl2, sr2, sr1m, sr2m, target, M, 0, M - 1, file);
329         // thread t1(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target,
    M, M /2 + 1, M-1, file);
330         ofstream record;
```

```
331         record.open("./record");
332         for (int i = 0; i < threadNumber; i++)
333         {
334             thread t(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target,
    M, ml, mr, i + 1, ref(record));
335             t.detach();
336             ml = mr + 1;
337             mr += M / threadNumber + 1;
338             if (abs(M - mr) < threadNumber)
339             {
340                 mr = M - 1;
341             }
342         }
343         while (result == false)
344         {
345             sleep(5);
346             /* code */
347         }
348 }
349 int main()
350 {
351         FILE *fp;
352         char StrLine[1024];
353         if ((fp = fopen("./data100", "r")) == NULL)
354         {
355             cout << ("[-]Error, can't open file") << endl;
356             return -1;
357         }
358         vector<mpz_t *> elements;
359         mpz_t arr[N];
360         for (int i = 0; i < N; i++)
361         {
362             mpz_init(arr[i]);
363             mpz_set_str(arr[i], fgets(StrLine, 1024, fp), 10);
364             elements.push_back(arr + i);
365         }
366         fclose(fp);
367         mpz_t target;
368         mpz_init(target);
369         mpz_set_str(target, target_str, 10);
370         shamir(elements, target);
371         mpz_clear(target);
372         for (int i = 0; i < N; i++){
373             mpz_clear(arr[i]);
374         }
375         return 0;
376 }
```

6.2 Howgrave-Graham-Joux 改进算法代码

```
#include <iostream>
#include <gmp.h>
#include <vector>
#include <random>
#include <string>
#include <ctime>
#include <algorithm>
#include <time.h>
#include <thread>
#include <stdlib.h>
#include <fstream>
#include <set>
#include <unistd.h>
using namespace std;
#define PBSTR "||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||"
#define PBWIDTH 60
typedef long long int ll;
typedef unsigned long int unint;
const unint N = 140;
const unint Hamingweight = 16;
bool result = false;
struct QuarterKnapsack
{
	mpz_t sum;
	vector<int> index;
};
struct ModuleKnapsck
{
	unint sum;
	unint index;
};
struct IndexPair
{
	unint i;
	unint j;
};
class Solution
{
public:
	mpz_t sum;
	unint sl1_index;
	unint sr1_index;
	Solution() { mpz_init(this->sum); };
	~Solution();
};
Solution::~Solution()
```

```
{
    // mpz_clear(this->sum);
    // mpz_realloc2(this->sum, 0);
}
vector<unint> binarySearch(vector<ModuleKnapsck> &slist, unint x)
{
    ll l = 0, r = slist.size() - 1, mid;
    vector<unint> ans;
    while (l <= r)
    {
        mid = (l + r) / 2;

        if (slist[mid].sum == x)
        {
            for (ll i = mid - 1; i > -1; i--)
            {
                if (slist[i].sum == x)
                {
                    ans.push_back(slist[i].index);
                }
                else
                {
                    break;
                }
            }
            ans.push_back(slist[mid].index);
            for (ll i = mid + 1; i < slist.size(); i++)
            {
                if (slist[i].sum == x)
                {
                    ans.push_back(slist[i].index);
                }
                else
                {
                    break;
                }
            }
            return ans;
        }
        else if (slist[mid].sum > x)
        {
            r = mid - 1;
        }
        else
        {
            l = mid + 1;
        }
    }
    return ans;
```

```
96  }
97  vector<IndexPair> binarySearchSol(vector<Solution> &slist, mpz_t x)
98  {
99      int l = 0, r = slist.size() - 1, mid;
100     vector<IndexPair> ans;
101     IndexPair solpair;
102     while (l <= r)
103     {
104         mid = (l + r) / 2;
105         if (mpz_cmp(slist[mid].sum, x) == 0)
106         {
107             for (ll i = mid - 1; i > -1; i--)
108             {
109                 if (slist[i].sum == x)
110                 {
111                     solpair.i = slist[i].sl1_index;
112                     solpair.j = slist[i].sr1_index;
113                     ans.push_back(solpair);
114                 }
115                 else
116                 {
117                     break;
118                 }
119             }
120             solpair.i = slist[mid].sl1_index;
121             solpair.j = slist[mid].sr1_index;
122             ans.push_back(solpair);
123             for (ll i = mid + 1; i < slist.size(); i++)
124             {
125                 if (slist[i].sum == x)
126                 {
127                     solpair.i = slist[i].sl1_index;
128                     solpair.j = slist[i].sr1_index;
129                     ans.push_back(solpair);
130                 }
131                 else
132                 {
133                     break;
134                 }
135             }
136             return ans;
137         }
138         else if (mpz_cmp(slist[mid].sum, x) == 1)
139         {
140             r = mid - 1;
141         }
142         else
143         {
144             l = mid + 1;
```

```
145             }
146         }
147 }
148 bool cmp_mod(ModuleKnapsck x, ModuleKnapsck y)
149 {
150         return x.sum < y.sum;
151 }
152
153 bool cmp_sol(Solution x, Solution y)
154 {
155         return (mpz_cmp(x.sum, y.sum) < 0);
156 }
157
158 void stdout_vector(vector<int> v)
159 {
160         for (int i = 0; i < v.size(); i++)
161             cout << v[i] << ' ';
162 }
163 void file_vector(vector<int> &v, ofstream &outfile)
164 {
165         for (int i = 0; i < v.size(); i++)
166             outfile << v[i] << ' ';
167 }
168
169 void stdout_vector(vector<mpz_t *> element)
170 {
171         for (int i = 0; i < element.size(); i++)
172             gmp_printf("%Zd\n", *element[i]);
173 }
174 void stdout_vector(vector<QuarterKnapsack> v)
175 {
176         for (int i = 0; i < v.size(); i++)
177         {
178             gmp_printf("%Zd ", v[i].sum);
179             for (int j = 0; j < v[i].index.size(); j++)
180             {
181                 cout << v[i].index[j] << ' ';
182             }
183             cout << endl;
184         }
185         cout << endl;
186 }
187 void insertset(set<int> &s, vector<int> &p)
188 {
189         for (int i = 0; i < p.size(); i++)
190             s.insert(p[i]);
191 }
192 void algorithm3(vector<QuarterKnapsack> &sl1, vector<QuarterKnapsack> &sr1,
    vector<QuarterKnapsack> &sl2,
```

```
193                     vector<QuarterKnapsack> &sr2,
194                     vector<ModuleKnapsck> &sr1m, vector<ModuleKnapsck> &sr2m,
195                     mpz_t target, ll M, ll Ml, ll Mr, int id)
196{
197     clock_t start, end;
198     start = clock();
199     cout << "algorm3 started" << endl;
200
201     ll step = 0;
202     // cout << Ml << ' ' << Mr << endl;
203     for (unint m = Ml; m <= Mr; m++)
204     {
205         step++;
206         if (step % 1 == 0)
207         {
208             double percentage = ((m - Ml) * 1.0) / (Mr - Ml);
209             int val = (int)(percentage * 100);
210
211             cout << '\n'
212                 << "thread: " << id << ' '
213                 << val << "% " << m << ' ' << Ml << '/' << Mr << endl;
214         }
215         vector<Solution> s;
216         for (unint sl1_index = 0; sl1_index < sl1.size(); sl1_index++)
217         {
218             mpz_t tmp;
219             mpz_init(tmp);
220             mpz_ui_sub(tmp, m, sl1[sl1_index].sum);
221             vector<unint> return_list = binarySearch(sr1m, mpz_fdiv_ui(tmp, M));
222             mpz_clear(tmp);
223             for (int i = 0; i < return_list.size(); i++)
224             {
225                 unint sr1_index = return_list[i];
226
227                 mpz_t sum;
228                 mpz_init(sum);
229                 mpz_add(sum, sl1[sl1_index].sum, sr1[sr1_index].sum);
230                 Solution sol;
231                 mpz_set(sol.sum, sum);
232                 sol.sl1_index = sl1_index;
233                 sol.sr1_index = sr1_index;
234                 s.push_back(sol);
235                 mpz_clear(sum);
236             }
237         }
238         // gmp_printf("%Zd\n", s[i].sum);
239         sort(s.begin(), s.end(), cmp_sol);
240         // cout << s.size() << endl;
241         // for (int i = 0; i < s.size(); i++)
```

```
242             // {
243             //         gmp_printf("%Zd ", s[i].sum);
244             //         stdout_vector(sl1[s[i].sl1_index].index);
245             //         cout << endl;
246             // }
247             // cout << "---" << endl;
248             for (int sl2_index = 0; sl2_index < sl2.size(); sl2_index++)
249             {
250                     mpz_t tmp;
251                     mpz_init(tmp);
252                     mpz_sub_ui(tmp, target, m);
253                     mpz_sub(tmp, tmp, sl2[sl2_index].sum);
254
255                     vector<unint> return_list = binarySearch(sr2m, mpz_fdiv_ui(tmp, M));
256                     mpz_clear(tmp);
257                     for (int i = 0; i < return_list.size(); i++)
258                     {
259                             unint sr2_index = return_list[i];
260
261                             mpz_t tt;
262                             mpz_init(tt);
263                             mpz_sub(tt, target, sl2[sl2_index].sum);
264                             mpz_sub(tt, tt, sr2[sr2_index].sum);
265                             vector<IndexPair> return_sol = binarySearchSol(s, tt);
266                             mpz_clear(tt);
267                             // cout << return_sol.size() << endl;
268                             for (int sol_index = 0; sol_index < return_sol.size(); sol_index++)
269                             {
270                                     // set<int> checkset;
271                                     // insertset(checkset, sl1[return_sol[sol_index].i].index);
272                                     // insertset(checkset, sr1[return_sol[sol_index].i].index);
273                                     // insertset(checkset, sl2[sl2_index].index);
274                                     // insertset(checkset, sr2[sr2_index].index);
275                                     // if (checkset.size() != Hamingweight){
276                                     //         continue;
277                                     // }
278                                     end = clock();
279                                     // gmp_printf("%Zd\n", sl1[return_sol[sol_index].i].sum);
280                                     // gmp_printf("%Zd\n", sr1[return_sol[sol_index].i].sum);
281                                     // gmp_printf("%Zd\n", sl2[sl2_index].sum);
282                                     // gmp_printf("%Zd\n", sr2[sr2_index].index);
283                                     cout
284                                             << "\rsolved in " << (double)(end - start) / CLOCKS_PER_SEC
    << endl;
285                                     stdout_vector(sl1[return_sol[sol_index].i].index);
286                                     stdout_vector(sr1[return_sol[sol_index].j].index);
287                                     stdout_vector(sl2[sl2_index].index);
288                                     stdout_vector(sr2[sr2_index].index);
289                                     cout << endl;
```

```
290                             ofstream outfile;
291                             outfile.open("./outH", ios::app);
292                             file_vector(sl1[return_sol[sol_index].i].index, outfile);
293                             file_vector(sr1[return_sol[sol_index].j].index, outfile);
294                             file_vector(sl2[sl2_index].index, outfile);
295                             file_vector(sr2[sr2_index].index, outfile);
296                             outfile << endl;
297                             result = true;
298                             return;
299                         }
300                     }
301                 }
302                 for (int i = 0; i < s.size(); i++)
303                     mpz_clear(s[i].sum);
304                 s.clear();
305             }
306 }
307 int combination(int n, int k)
308 {
309     if (n == k || k == 0)
310         return 1;
311     else
312         return combination(n - 1, k) + combination(n - 1, k - 1);
313 }
314 void initKnapsack(int n, int r, vector<mpz_t *> &elements,
315                           vector<QuarterKnapsack> &sl1, vector<QuarterKnapsack> &sr1,
316                           vector<QuarterKnapsack> &sl2, vector<QuarterKnapsack> &sr2)
317 {
318     vector<QuarterKnapsack> tt;
319     std::vector<bool> v(n);
320     std::fill(v.end() - r, v.end(), true);
321     int total_comb = combination(n, r);
322     int comb_step = 0;
323     do
324     {
325         double percentage = comb_step * 1.0 / total_comb;
326         int val = (int)(percentage * 100);
327         if (comb_step % 1000 == 0)
328         {
329             cout << '\r' << val << '%';
330         }
331         QuarterKnapsack tmp;
332         mpz_init(tmp.sum);
333         for (int i = 0; i < n; ++i)
334         {
335             if (v[i])
336             {
337                 mpz_add(tmp.sum, tmp.sum, *elements[i]);
338                 tmp.index.push_back(i);
```

```
339                 }
340             }
341             tt.push_back(tmp);
342             comb_step++;
343         } while (std::next_permutation(v.begin(), v.end()));
344         srand(time(0));
345         random_shuffle(tt.begin(), tt.end());
346         for (comb_step = 0; comb_step < tt.size(); comb_step++)
347         {
348             if (comb_step < total_comb / 4)
349             {
350                 sl1.push_back(tt[comb_step]);
351             }
352             else if (comb_step < total_comb / 2)
353             {
354                 sr1.push_back(tt[comb_step]);
355             }
356             else if (comb_step < 3 * total_comb / 4)
357             {
358                 sl2.push_back(tt[comb_step]);
359             }
360             else
361             {
362                 sr2.push_back(tt[comb_step]);
363             }
364         }
365 }
366 void howgrave(vector<mpz_t *> elements, mpz_t target, int K, int threadNumber)
367 {
368     vector<QuarterKnapsack> sl1, sr1, sl2, sr2;
369     cout << "init..." << endl;
370     initKnapsack(N, K / 4, elements, sl1, sr1, sl2, sr2);
371     cout << '\n';
372     cout << "quarter size: " << sl1.size() << ' ' << sr1.size() << ' '
373          << sl2.size() << ' ' << sr2.size() << ' ' << endl;
374     double ll = pow(2, N / 4.0), rr = pow(2, (N / 4.0) + 1);
375     default_random_engine e(time(0));
376     uniform_real_distribution<double> u(ll, rr);
377     int M = (int)u(e);
378     M = pow(2, 20);
379     cout << "Mod: " << M << endl;
380     cout << "init module" << endl;
381     vector<ModuleKnapsck> sr1m;
382     for (int i = 0; i < sr1.size(); i++)
383     {
384         ModuleKnapsck _t;
385         _t.sum = mpz_fdiv_ui(sr1[i].sum, M), _t.index = i;
386         sr1m.push_back(_t);
387     }
```

```
388     vector<ModuleKnapsck> sr2m;
389     for (int i = 0; i < sr2.size(); i++)
390     {
391         ModuleKnapsck _t;
392         _t.sum = mpz_fdiv_ui(sr2[i].sum, M), _t.index = i;
393         sr2m.push_back(_t);
394     }
395     cout << "sorting" << endl;
396     sort(sr1m.begin(), sr1m.end(), cmp_mod);
397     sort(sr2m.begin(), sr2m.end(), cmp_mod);
398     cout << "sort completed" << endl;
399     // stdout_vector(sl1);
400     // stdout_vector(sr1);
401     // stdout_vector(sl2);
402     // stdout_vector(sr2);
403     // algorithm3(sl1, sr1, sl2, sr2, sr1m, sr2m, target, M, 0, M - 1, 1);
404     int ml = 0, mr = M / threadNumber;
405     // algorithm3(sl1, sr1, sl2, sr2, sr1m, sr2m, target, M, 0, M - 1, file);
406     // thread t1(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target,
    M, M /2 + 1, M-1, file);
407     for (int i = 0; i < threadNumber; i++)
408     {
409         thread t(algorithm3, ref(sl1), ref(sr1), ref(sl2), ref(sr2), ref(sr1m), ref(sr2m), target,
    M, ml, mr, i + 1);
410         t.detach();
411         ml = mr + 1;
412         mr += M / threadNumber + 1;
413         if (abs(M - mr) < threadNumber)
414         {
415             mr = M - 1;
416         }
417     }
418     while (result == false)
419     {
420         sleep(5);
421         /* code */
422     }
423 }
424 int main(int argc, char **argv)
425 {
426     FILE *fp;
427     char StrLine[1024];
428     if ((fp = fopen("./data140", "r")) == NULL)
429     {
430         cout << ("[-]Error, can't open file") << endl;
431         return -1;
432     }
433     vector<mpz_t *> elements;
434     mpz_t arr[N];
```

```
435     for (int i = 0; i < N; i++)
436     {
437         mpz_init(arr[i]);
438         mpz_set_str(arr[i], fgets(StrLine, 1024, fp), 10);
439         elements.push_back(arr + i);
440     }
441     fclose(fp);
442     mpz_t target;
443     mpz_init(target);
444     if (argc < 2){
445         cout << "[-]Error, no target" << endl;
446         return 0;
447     }
448     char *target_str = argv[1];
449     mpz_set_str(target, target_str, 10);
450     gmp_printf("target %Zd\n", target);
451     if (argc < 3)
452     {
453         cout << "[-]Error, no threadNumber" << endl;
454         return 0;
455     }
456     int threadNumber = stoi(argv[2]);
457     cout << "threadNumber: " << threadNumber << endl;
458     howgrave(elements, target, Hamingweight, threadNumber);
459 
460     // mpz_clear(target);
461     // for (int i = 0; i < N; i++)
462     // {
463     //         mpz_clear(arr[i]);
464     // }
465     return 0;
466 }
```