Cryptohack刷题记录(三) Mathematics部分 Lattice WriteUp

原创

Mr_AgNO3 已于 2022-01-24 01:23:04 修改 230 收藏

文章标签： 线性代数 算法 几何学 python 密码学

于 2022-01-24 01:13:36 首次发布

本文链接：https://blog.csdn.net/u010883831/article/details/122660103

文章目录

Mathematics

Lattices

1. Vectors
2. Size and Basis
3. Gram Schmidt
4. What's Lattice?
5. Gaussian Reduction
6. Find the Lattice
7. Backpack Cryptography

backpack Cryptography没出

Mathematics

Lattices

1. Vectors

基本的向量于标量之间的运算

题目：

`v = (2,6,3)`, `w = (1,0,0)` and `u = (7,7,2)`, calculate `3*(2*v - w) · 2*u`.

直接使用sage计算

```
sage: v = vector([2,6,3])
sage: w = vector([1,0,0])
sage: u = vector([7,7,2])
sage: 3*(2*v-w)*2*u
702
```

flag是`702`

2. Size and Basis

一组向量 $v \ldots v \in V$ 是线性无关的，当且仅当

`基中的元素个数`是`向量空间的维数`

向量的大小，定义为`||v||`,是自己和自己的内积==$||v|| =$ ==

一组`正交基(orthogonal)` $v, v \ldots v \in V$ ，任意两个元素的内积为0——

一组`标准正交基(orthonormal)`,是对于所有$i \in [1, n]$ 都满足满足==$||v, ||$ — 1 ——的正交基

题目：

计算$v = (4, 6, 2, 5)$ 的大小(size)

$||v|| =$ =

flag就是`9`

3. Gram Schmidt

施密特算法

可以将向量组$v, v \ldots v \in V$ 正交化为

> =
> Set $v =$
> End Loop

题目：

`v1 = (4,1,3,-1), v2 = (2,1,-3,4), v3 = (1,0,-2,7), v4 = (6, 2, 9, -5)`

使用`Gram Schmidt算法`计算标准基

写了半天手搓线性代数，全是bug，不写了

寄了

一晚上全学线代了，

连个施密特都写不出来

Sage有内置的Gram Schmidt

```
sage: v0 = vector([4,1,3,-1])
sage: v1 = vector([2,1,-3,4])
sage: v2 = vector([1,0,-2,7])
sage: v3 = vector([6,2,9,-5])
sage: M = Matrix([v0,v1,v2,v3])
sage: M.gram_schmidt()
(
[         4          1          3         -1]
[     70/27      31/27     -23/9     104/27]
[  -287/397   -405/397    799/397    844/397]
[-1456/4023    273/298  1729/8046   455/4023],

[       1        0        0        0]
[   -4/27        1        0        0]
[    -1/3  468/397        1        0]
[   58/27 -659/794 439/4023        1]
)
sage: M
[ 4  1  3 -1]
[ 2  1 -3  4]
[ 1  0 -2  7]
[ 6  2  9 -5]
```

flag就是`0.91611`(四舍五入)

我也贴一个别人的solution，实现了题目所给的算法，作为学习参考

```
import numpy as np
v = [
    np.array([4,1,3,-1]),
    np.array([2,1,-3,4]),
    np.array([1,0,-2,7]),
    np.array([6,2,9,-5]),
]

"""
u1 = v1
Loop i = 2,3...,n
   Compute μij = vi · uj / ||uj||2, 1 ≤ j < i.
   Set ui = vi - μij * uj (Sum over j for 1 ≤ j < i)
End Loop
"""
u = [v[0]]
for vi in v[1:]:
    mi = [np.dot(vi, uj) / np.dot(uj, uj) for uj in u]
    u += [vi - sum([mij * uj for (mij, uj) in zip(mi,u)])]

print(round(u[3][1], 5))
```

---

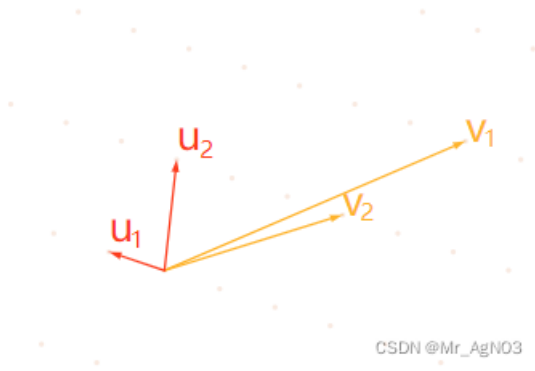
4. What's Lattice?

格的基本空间F

给定一组线性无关的向量

$v_1, v_2, \ldots, v_n \in R^n$，则由这组向量生成的格 L 是由 [illegible] 和整数系数的组合

> The choice of basis is far from unique



由一组基向量和与基向量相乘的数，可以到达任何一点。一组基向量定义了一块`基本空间(fundamental domain)`

$$F(v_1,\ldots,v_n) = \{t_1 v_1 + t_2 v_2 + \cdots$$

我们可以通过基向量来计算基本空间。

> We can calculate the volume of the fundamental domain from the basis vectors. As an example, let us take a two dimensional lattice with basis vectors `v = (2,5), u = (3,1)`. Create a matrix `A` with rows corresponding to the basis vectors: `A = [[2,5],[3,1]]`. The volume of the fundamental domain is the magnitude of the determinant of A: `Vol(F) = |det(A)| = |2*1 - 5*3| = |-13| = 13`.

题目：

已知$v_1 = (6,2,-3)$ $v_2 =$ $\qquad$ 计算基本空间的体积(the volume of the fundamental domain)

```
sage: v = vector
sage: v1 = v([6,2,-3])
sage: v2 = v([5,1,4])
sage: v3 = v([2,7,1])
sage: A = matrix([v1,v2,v3])
sage: A
[ 6  2 -3]
[ 5  1  4]
[ 2  7  1]
sage: det(A)
-255
```

flag就是`255`

---

5. Gaussian Reduction

如果你仔细观察，“格”开始出现在密码学中的每一个角落。 有时它们操纵一个加密系统，破坏了(生成)不够安全的参数。 最著名的例子是 Coppersmith 对 RSA 加密的攻击。

格也可用于构建加密协议，其安全性基于两个基本的“难”问题：

1. The **Shortest Vector Problem** (SVP)

给定的基向量和格，找到格L中的长度最短非零向量。换言之，找到一个非零向量$v\in L$使$||v||$最小

2. The **Closest Vector Problem** (CVP)

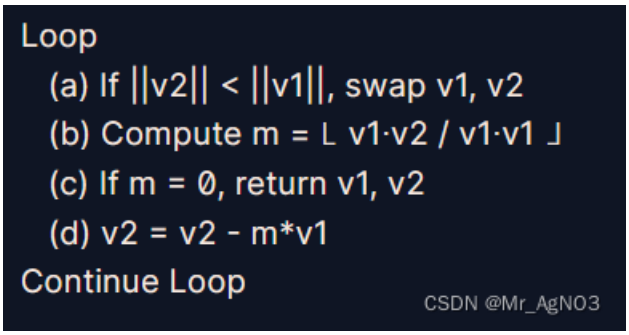
给定的基向量、格，和一个不在格上的目标向量，找到距离目标向量最近的格向量。

给定一个不在格L中的向量$v\in R$找到向量使||最小

对于一般的格，SVP问题十分困难，但对于相对简单的情况，我们有非常有效的算法可以计算出问题的解或者近似解。当格的维数小于等于4时，可以在多项式时间内计算出确定解，对于更高维，只能求出近似解。

高斯发明了一种算法来找到给定任意基的二维格 L 的最佳基。算法的输出 v1 是 L 中最短的非零向量，由此解决了二维的SVP问题。

高斯的算法大致通过从另一个基向量中减去一个基向量的倍数来达到目标，直到不再可能使它们变得更小。这适用于二维，因此可以很好地可视化。



> 一句一句的翻译已经很累了，不想再敲公式了，直接放图吧

脚本也不想写了

直接交互式命令行吧

```
import numpy as np
ar = np.array
v = ar([846835985, 9834798552],dtype='i8')
u = ar([87502093, 123094980],dtype='i8')
v1,v2 = u,v
```

```
>>> if siz2(v2) < siz2(v1):
 print('swap')
 v1,v2 = v2,v1

>>> m = int(v1.dot(v2)/v1.dot(v1));m
56
>>> v2 = v2 - m*v1;v2
array([-4053281223,  2941479672], dtype=int64)
>>> if siz2(v2) < siz2(v1):
 print('swap')
 v1,v2 = v2,v1

>>> m = int(v1.dot(v2)/v1.dot(v1));m
0
>>> v1
array([ 87502093, 123094980], dtype=int64)
>>> v2
array([-4053281223,  2941479672], dtype=int64)
>>> v1.dot(v2)
7410790865146821
```

flag即`7410790865146821`

---

6. Find the Lattice

这种题在各类比赛里比较常见了，可以参考一下这个

https://xz.aliyun.com/t/7163

题目

```python
from Crypto.Util.number import getPrime, inverse, bytes_to_long
import random
import math

FLAG = b'crypto{????????????????????}'
def gen_key():
    q = getPrime(512)
    upper_bound = int(math.sqrt(q // 2))
    lower_bound = int(math.sqrt(q // 4))
    f = random.randint(2, upper_bound)
    while True:
        g = random.randint(lower_bound, upper_bound)
        if math.gcd(f, g) == 1:
            break
    h = (inverse(f, q)*g) % q
    return (q, h), (f, g)

def encrypt(q, h, m):
    assert m < int(math.sqrt(q // 2))
    r = random.randint(2, int(math.sqrt(q // 2)))
    e = (r*h + m) % q
    return e

def decrypt(q, h, f, g, e):
    a = (f*e) % q
    m = (a*inverse(f, g)) % g
    return m


public, private = gen_key()
q, h = public
f, g = private

m = bytes_to_long(FLAG)
e = encrypt(q, h, m)

print(f'Public key: {(q,h)}')
print(f'Encrypted Flag: {e}')
```

已知的值有

q,h,e

其中

$$h \equiv f \quad g \quad \mathrm{mod} p$$

> q 512位
> f,g 低于256位
> h，e 512位
> r 256位
> m为flag，大小约为231bit

要求得m，需要获得函数`decrypt`的参数，还剩`f`和`g`是未知的

而，我们唯一知道的关于`f`和`g`的关系式，只有

$$f \cdot h \equiv$$

向量`(f,g)`可以由两组基向量`M`的某种整系数线性组合`(f, -u)`来表示，因此向量`(f,g)`就在这个lattice上。

对于两个基底向量，

(1,h) : 512位

(0,q) : 512位

而向量 (f,g) 的长度 : 大约256位

相比于基底向量，是非常小的。

很大概率上，**这个`(f, g)`就是这个lattice的最短向量**，(Gaussian heurstic)。

上一题我没写程序，现在补回来...

```
# sage
# GaussLatticeReduction
def Gauss(v1, v2):
    while True:
        if v2.norm() < v1.norm():
            v1, v2 = v2, v1
        m = round( v1*v2 / v1.norm()^2 )
        if m == 0:
            return (v1, v2)
        v2 = v2 - m*v1
```

```
sage: v = vector([1,h])
sage: u = vector([0,q])
sage: Gauss(u,v)
((472518176144313694681510883019487227616946226062205789815612365633258081787 56, 439979578851470781158511474563708800896962564703897823482933419379155042545 89),
(-672690102502127170754321826930439631840976488801650086219078312846471160259 01, 990127634595298588096084361335646305243501060002420703368183040534679422691 78))
```

输出的第一个向量即所求的`(f,g)`

```
>>> len(bin(f)[2:])
255
>>> len(bin(g)[2:])
255
```

长度也满足

现在直接运行`decrypt`函数即可得到flag

```
q, h = (763823212045492587923155423401184234764101788821902117530421735871587863618325243345489649067749651614989316745664606749499241420160898019203925115292257, 216326890219456009384369357217019970750178779749799846346212959223997358146265162297828263751386527419937445280529263958626479131743902953592640110907480 0)
e = 560569649525372066414288195690862430757067185847748211965743616366366384473116903568234497428637904912373335600912567192428031253275524116226726912348652 3
f,g = 472518176144313694681510883019487227616946226062205789815612365633258081787 56, 439979578851470781158511474563708800896962564703897823482933419379155042545 89
print(l2b(decrypt(q,h,f,g,e)))
```

`crypto{}`

---

7. Backpack Cryptography

背包加密算法

这题我没做出来

参考密码学——公钥密码体系之背包算法1_摆渡沧桑-CSDN博客

这是一种早期的公钥算法，背包算法的安全性起源于背包难题，他是一个**NP完全**问题，但后来发现该算法并不安全，但是由于它证明了如何将NP完全问题用于公开秘钥密码学

Merkle-Hellman背包算法的思想是将消息编码为背包问题的解。明文分组长度等于堆中物品的个数，并且明文位与b的值相对应，密文则是计算得到的和值，下图给出了一段用例子中背包问题来加密的明文；

明文：	1 1 1 0 0 1	0 1 0 1 1 0	0 1 1 0 0 0
密钥：	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20
密文：	1+5+6+20 = 32	5+11+14 = 30	5+6 = 11

对于易解的背包问题，可以选择**超递增序列**，那么相应的背包问题容易求解。
超递增序列：**它的每一项都大于它之前所有项之和**，例如{1，3,6，13，27，52}是一个超递增序列，而{1，3，4，9，15，25}则不是。

非超递增序列的背包是困难的问题，它们没有快速算法。

背包加密算法在于，公钥是一个非超递增序列背包，私钥是超递增序列和将两者互相转换的 r

背包问题的解密及破解_Taotaoboke的博客-CSDN博客_超递增序列背包问题

破译的基本思想是不必找出正确的模数 q 和乘数 r（即陷门信息）只需要找到出任意模数 k 和乘数 t，然后用 k 和 t 对公开的背包向量 b 求出新的超递增向量即可。

我使用的脚本来自这里： 背包加密 - CTF Wiki (ctf-wiki.org) 用wiki上的脚本

```
import binascii
# open the public key and strip the spaces so we have a decent array
pubKey = a
nbit = len(pubKey)
# open the encoded message
encoded = c
print("start")
# create a large matrix of 0's (dimensions are public key length +1)
A = Matrix(ZZ, nbit + 1, nbit + 1)
# fill in the identity matrix
for i in range(nbit):
    A[i, i] = 1
# replace the bottom row with your public key
for i in range(nbit):
    A[i, nbit] = pubKey[i]
# last element is the encoded message
A[nbit, nbit] = -int(encoded)

res = A.LLL()
for i in range(0, nbit + 1):
    # print solution
    M = res.row(i).list()
    flag = True
    for m in M:
        if m != 0 and m != 1:
            flag = False
            break
    if flag:
        print(i, M)
        M = ''.join(str(j) for j in M)
        # remove the last bit
        M = M[:-1]
        M = hex(int(M, 2))[2:-1]
        print(M)
```

但是没有结果