Problema 2

Na criptografia pós-quântica os reticulados inteiros (“$hardlattices$”) e os problemas a eles associados são uma componente essencial. Um reticulado inteiro pode ser definido por uma matriz $\mathsf{L} \in \mathbb{Z}^{m\times n}$ (com $m > n$) de inteiros e por um inteiro primo $q \geq 3$. O chamado problema do vetor curto (SVP) consiste no cálculo de um vetor de inteiros

$$e \in \{-1, 0, 1\}^m$$

não nulo que verifique a seguinte relação matricial

$$\forall\, i < n\, \centerdot\, \sum_{j<m} e_j \,\times\, \mathsf{L}_{j,i} \;\equiv\; 0 \mod q$$

1. Pretende-se resolver o SVP por programação inteira dentro das seguintes condições
    A. Os valores m , n , q são escolhidos com $n > 30$, $|m| > 1 + |n|$ e $|q| > |m|$.
    B. Os elementos $\mathsf{L}_{j,i}$ são gerados aleatória e uniformemente no intervalo inteiro $\{-d \cdots d\}$ sendo $d \equiv (q-1)/2$.
2. Pretende-se determinar, em primeiro lugar, se existe um vetor e não nulo (pelo menos um dos e_j é diferente de zero). Se existir e pretende-se calcular o vetor que minimiza o número de componentes não nulas.

Notas

Se $x \geq 0$, representa-se por $|x|$ o tamanho de x em bits: o menor ℓ tal que $x < 2^\ell$.

- Um inteiro x verifica $x \equiv 0 \mod q$ sse x é um múltiplo de q . $x \equiv 0 \mod q$ sse $\exists\, k \in \mathbb{Z}\, \centerdot\, x \,=\, q \times k$.

  Por isso, escrito de forma matricial, as relações que determinam o vetor $e \neq 0$ são

$$\begin{cases} \exists\, e \in \{-1,0,1\}^m \,\centerdot\, \exists\, k \in \mathbb{Z}^n \;\;\centerdot\;\; e \times \mathsf{L} \,=\, q\,k \\ \qquad\qquad\qquad\qquad \exists\, i < n \;\;\centerdot\;\; e_i \,\neq\, 0 \end{cases}$$

Resolução do problema

$$\begin{bmatrix} x_{0,-1} & x_{0,0} & x_{0,1} \\ x_{1,-1} & x_{1,0} & x_{1,1} \\ . & . & . \\ . & . & . \\ . & . & . \\ x_{m-1,-1} & x_{m-1,0} & x_{m-1,1} \end{bmatrix}$$

Utilização de uma matriz com m linhas e 3 colunas em que os valores das colunas estão dentro do intervalo [-1, 1] e onde os valores de $x_{i,j}$ são 0(False) ou 1(True), com $0 \leq i \leq m-1$ e $-1 \leq j \leq 1$.

A seguinte expressão, para i = 0

$$\sum_{j=-1}^{j<2} x_{0,j} \,\times\, \mathsf{j}$$

representa o elemento e_0 do vetor e.

Assim, obtemos o vetor e a partir da seguinte expressão:

$$\forall_{i<m}\,(e_i = \sum_{j=-1}^{j<2} x_{i,j} \,\times\, \mathsf{j}\,)$$

Restrições

```
1 - Cada linha da matriz tem de ter um e um só valor a 1
```

Para isso utilizamos a seguinte restrição:

$$\forall_{i<m}(\sum_{j=-1}^{j<2} x_{i,j}) = 1$$

```
2 - O vetor nulo não pode ser uma solução
```

Para isso utilizamos a seguinte restrição:

$$\sum_{i=0}^{i<m} x_{i,0} = m$$

```
3- Relação de congruência
```

$$\forall\, i < n\, \centerdot\, \sum_{j<m} e_j \,\times\, \mathsf{L}_{j,i} \;\equiv\; 0 \mod q$$

Para isso utilizamos a seguinte restrição:

$$\forall_{i<n}(\sum_{j=0}^{j<m}(\sum_{t=-1}^{t<2} x_{j,t} \times t) \times L_{j,i}) = q \times k_i, \quad k_i \in \mathbb{Z}$$

Em que k toma diferentes valores consoante o i que toma para validar a congruência

Minimizar soluções

Para minimzar o número de elementos de e não nulos utilizamos a seguinte expressão: \

$$\sum_{i=0}^{i<m} x_{i,0}$$

\ Ou seja, vamos à matriz m e minimizamos a coluna de elementos a 0.

Função erastostenes_crive(n)

```
n - limite superior
```

Esta função é utilizada para saber quais os números primos entre 1 e n.

In [1]:

```python
def erastostenes_crive(n):
    numeros = [True] * (n + 1)

    numeros[0] = False
    numeros[1] = False

    primos = []

    for numero, primo in enumerate(numeros):
        if primo:
            primos.append(numero)

            for i in range(numero * 2, n + 1, numero):
                numeros[i] = False

    print(primos)
    return primos

erastostenes_crive(255)
pass
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 18
1, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251]
```

Função generate_matrix(m, n, d)

```
m - número de linhas da matriz
n - número de colunas da matriz
d - inteiro que serve de referência para o intervalo de valores [-d, d] da matriz L
```

Utilizada para gerar uma matriz com m linhas e n colunas com valores entre d e -d.

In [2]:

```python
from random import randint

def generate_matrix(m, n, d):
    L = {}
    for i in range(0, m):
        L[i] = {}
        for j in range(0, n):
            random_number = randint(-d,d)
            L[i][j] = random_number
    return L
```

Função print_matrix(table, solver)

```
m - matriz a que vamos dar print
n - solver passado como parâmetro
```

Utilizada para dar print à matriz resultante.

In [3]:

```python
def print_matrix(table, solver):
    for i in table:
        for j in table[i]:
            print(int(solver.Value(table[i][j])), end="  ")
        print("")
```

Função print_vector(e)

```
e - vetor e passado como parâmetro
```

Utilizada para dar print ao vetor e resultante.

In [4]:

```python
def print_vector(e):
    for i in range(0, m):
        print(e[i], end=" ")
```

In [5]:

```python
from ortools.sat.python import cp_model
from pysmt.typing import INT

def SVP_Matrix(n, m, q, d, L):
    model = cp_model.CpModel()

    e_matrix = {}

    for i in range(0, m):
        e_matrix[i] = {}
        for j in range(-1, 2):
            e_matrix[i][j] = model.NewBoolVar(f'e_matrix[{i}][{j}]')



    # 1ª condição - Cada linha tem um valor
    for i in range(0,m):
        model.Add( sum( [e_matrix[i][j] for j in range(-1,2)]) == 1  )

    # 2ª condição - Não há o vetor nulo
    model.Add(sum([e_matrix[i][0] for i in range(0,m)]) <= (m-1))


    k = {}
    max_range = 2**52
    # 3ª condição - condição de congruência
    for i in range(0,n):
        k[i] = model.NewIntVar(-1*max_range, max_range, f'k[{i}]')
        model.Add(sum( sum( e_matrix[j][t]*t for t in range(-1,2)) *L[j][i] for j in range(0,m) )  == k[i]*q )

    # Minimizar o número de componentes não nulas
    model.Minimize( sum(e_matrix[i][0] for i in range(0, m)) )

    # Cria um solver CP-SAT a solver and solves the model.
    solver = cp_model.CpSolver()

    # Invoca o solver com o modelo criado
    status = solver.Solve(model)


    # Interpreta os resultados
    if status == cp_model.OPTIMAL:
        # conversão da matriz para vetor
        e = {}
        for i in range(0, m):
            acc = 0
            for j in range(-1, 2):
                acc += solver.Value(e_matrix[i][j]) * j
            e[i] = acc
        print_matrix(e_matrix, solver)
        print("")
        print_vector(e)
    else:
        print('No solution found.')
```

Exemplo 1

In [6]:

```python
n = 2
m = 12
q = 23
d = (q-1)/2
L = generate_matrix(m,n,d)

SVP_Matrix(n, m, q, d, L)
```

```
1  0  0
0  0  1
0  0  1
0  0  1
1  0  0
0  0  1
0  0  1
1  0  0
0  0  1
1  0  0
1  0  0
0  0  1

-1 1 1 1 -1 1 1 -1 1 -1 -1 1
```

Exemplo 2

In [7]:

```python
n = 3
m = 12
q = 29
d = (q-1)/2
L = generate_matrix(m,n,d)

SVP_Matrix(n, m, q, d, L)
```

```
0  0  1
1  0  0
1  0  0
0  0  1
0  0  1
0  0  1
0  1  0
0  0  1
0  1  0
0  0  1
1  0  0
1  0  0

1 -1 -1 1 1 1 0 1 0 1 -1 -1
```

Exemplo 3

In [8]:

```python
n = 4
m = 16
q = 37
d = (q-1)/2
L = generate_matrix(m,n,d)

SVP_Matrix(n, m, q, d, L)
```

```
1  0  0
1  0  0
1  0  0
1  0  0
1  0  0
1  0  0
1  0  0
0  1  0
1  0  0
0  0  1
0  0  1
0  0  1
1  0  0
1  0  0
1  0  0
0  0  1

-1 -1 -1 -1 -1 -1 -1 0 -1 1 1 1 -1 -1 -1 1
```

Exemplo 4

In [9]:

```python
n = 4
m = 28
q = 47
d = (q-1)/2
L = generate_matrix(m,n,d)

SVP_Matrix(n, m, q, d, L)
```

```
1  0  0
1  0  0
0  0  1
0  0  1
0  0  1
1  0  0
1  0  0
0  0  1
1  0  0
0  0  1
1  0  0
1  0  0
1  0  0
0  0  1
1  0  0
0  0  1
1  0  0
0  0  1
0  0  1
0  0  1
1  0  0
0  0  1
1  0  0
0  0  1
0  0  1
1  0  0
0  0  1
1  0  0

-1 -1 1 1 1 -1 -1 1 -1 1 -1 -1 -1 1 -1 1 -1 1 1 1 -1 1 -1 1 1 -1 1 -1
```

Exemplo 5

In [10]:

```python
n = 5
m = 28
q = 37
d = (q-1)/2
L = generate_matrix(m,n,d)

SVP_Matrix(n, m, q, d, L)
```

```
1  0  0
1  0  0
1  0  0
1  0  0
0  0  1
1  0  0
0  0  1
1  0  0
0  0  1
0  0  1
1  0  0
1  0  0
0  0  1
1  0  0
1  0  0
0  0  1
0  0  1
0  0  1
0  0  1
1  0  0
0  0  1
1  0  0
1  0  0
1  0  0
0  0  1
0  0  1
0  0  1
1  0  0

-1 -1 -1 -1 1 -1 1 -1 1 1 -1 -1 1 -1 -1 1 1 1 1 -1 1 -1 -1 -1 1 1 1 -1
```

Exemplo 6

In [ ]:

```python
n = 6
m = 28
q = 37
d = (q-1)/2
L = generate_matrix(m,n,d)

SVP_Matrix(n, m, q, d, L)
```

In [ ]: