

Wydział Cybernetyki WAT

Przedmiot: Matematyka Konkretna

SPRAWOZDANIE

Temat: Rozwiązanie problemu wektora najkrótszego w bazie, za pomocą algorytmów

Gauss'a – Legendre'a
oraz
Lenstra–Lenstra–Lovász

Wykonał:

Paweł Witkowski
[K5X4S1]

Data wykonania ćwiczenia:

18.05.2017

Prowadzący ćwiczenie:

Kpt. Dr inż. Mariusz Jurkiewicz

Spis treści

1. Wstęp teoretyczny	3
2. Generator liczb pseudolosowych	4
2.1 Pseudokod	4
2.2 Realizacja C++/BOOST	5
3. Redukcja bazy Gauss'a – Legendre'a	7
3.1 Algorytm redukcji/Pseudokod	8
3.2 Realizacja C++/BOOST	9
4. Redukcja bazy Lenstra–Lenstra–Lovász	10
4.1 Algorytm redukcji LLL/Pseudokod	11
4.2 Realizacja C++	12
5. Łamanie NTRU za pomocą algorytmu LLL	14
5.1 Użycie LLL na przykładzie zadanego NTRU	15
6. Wnioski	16

1. Wstęp teoretyczny

Kraty

Zakłada się, że bezpieczeństwo szyfrowania asymetrycznego w oparciu o protokół klucza publicznego leży w wielkości klucza. Rozwiązywanie problemu logarytmu dyskretnego w odpowiednio dużych ciałach skończonych czy też faktoryzacja naprawdę dużych liczb.

Bezpieczeństwo krat opiera się w całkowicie innym miejscu. Kratą nazywany przestrzeń wektorów, rozpiętą nad \mathbb{R}^n (\mathbb{Z}^n bez zmniejszenia ogólności). Jedyną różnicą jest mnożenie wektorów przez liczby całkowite. Dzieje się tak z powodu niemożności uzyskania odpowiedniej precyzji na obecnych komputerach (float i double to przybliżenia, nie realne wartości).

Bazą kraty nazywamy zestaw wektorów liniowo niezależnych rozpiętych nad otoczką kraty. Można powiedzieć, że dowolny punkt na kracie można otrzymać przez liniową kombinację wektorów należących do bazy. Dwie bazy mogą generować tę samą kratę ($L(B) = L(C)$, gdzie B i C to bazy krat), w przypadku, gdy istnieje taka macierz unimodularna (o wyznaczniku ± 1), że zachodzi $C^T = AB^T$.

Na wyżej wymienionym twierdzeniu jest oparte bezpieczeństwo klucza publicznego krat, ponieważ bazy praktycznie prostopadłe (dalej ortogonalne) mogą być bazami tej samej kraty, co bazy bardzo „wygięte” – przypominające równoległobok. Będą one miały wtedy taką samą objętość obszaru fundamentalnego.

Obszar fundamentalny – obszar pomiędzy wektorami bazy. Jest ograniczony wektorami do niego należącymi. Objętość tego obszaru to objętość/pole tej figury.

Bezpieczeństwo krat

Istnieje wiele problemów na których oparte jest bezpieczeństwo krat. Fundamentalne dwa to SVP, CVP.

- SVP – (Shortest Vector Problem), problem znalezienia wektora najkrótszego, niezerowego, należącego do danej kraty. Minimalizuje on normę euklidesową z wektorów w danej kratce. $\mathbf{v} \in \mathbf{L}$, $\|\mathbf{v}\|$ jak najmniejszy.
- CVP – (Closest Vector Problem), zadany jest wektor $\mathbf{w} \in \mathbf{R}^m$, nienależący do kraty. Problemem jest znalezienie wektora $\mathbf{v} \in \mathbf{L}$, który jest najbliższy wektorowi \mathbf{w} . Można to opisać że norma euklidesowa $\|\mathbf{v} - \mathbf{w}\|$ jest jak najmniejsza.

Jest również odmiana powyższych, mianowicie **apprSVP** oraz **apprCVP**, gdzie przybliżamy wartość powyższych wektorów. Obydwa problemy są problemami NP-trudnymi, czyli takimi, w których zakładane rozwiązanie możemy sprawdzić w czasie wielomianowym.

Należy zwrócić uwagę, że może nie istnieć jeden wektor będący rozwiązaniem powyższych problemów, może być ich kilka z racji faktu, że rozpatrujemy tylko normy.

Dodatkowe pojęcia

- Ortogonalizacja Gramma-Schmidta – polega na zmianie wektorów w taki sposób, by wszystkie wobec siebie były „prostopadłe” – tj. ortogonalne. Od każdego wektora, z wyjątkiem pierwszego odejmujemy rzut wektora na wektor aktualnie rozpatrywany. Wzory służące do ortogonalizacji

$$v_i^* = v_i - \sum_{j=1}^{i-1} \mu_{i,j} v_j^*, \text{ gdzie } \mu_{i,j} = \frac{\langle v_i, v_j^* \rangle}{\|v_j^*\|^2}, \text{ dla } 1 \leq j \leq i-1$$

- Współczynnik Hadamara – oznaczany $H(B)$, miara ortogonalności bazy, przyjmuje wartości z przedziału $(0,1]$. Im bliżej „jedynek” tym bardziej ortogonalna jest baza

$$H(B) = \left(\frac{\det L}{\|v_1\| \cdot \|v_2\| \dots \|v_n\|} \right)^{\frac{1}{n}}$$

2. Generator liczb pseudolosowych

W celu stworzenia liczby **n** bitowej należy wykorzystać generację małej liczby pseudolosowej, za pomocą funkcji wbudowanej w podstawowe biblioteki (zależne od języka), a następnie, za pomocą przesunięcia bitowego przesunąć ją w lewo.

Na wejście zostaje podana wielkość **n**, która określa ilość bitów wygenerowanej liczby. W zależności od języka trzeba brać pod uwagę jakiej wielkości liczby, generują podstawowa funkcja pseudolosowa. Należy pobrać tę wartość i do niej dopasować algorytm.

2.1 Pseudokod

Input n

$$\text{iterator} = \left\lfloor \frac{n}{\text{RNG_SIZE}} \right\rfloor$$

IF ($n < \text{RNG_SIZE}$)

L = random

ELSE

FOR i IN iterator

X = random

$X = 2^{\text{RNG_SIZE}-1}$ or X

$X = X \cdot 2^{n - (\text{RNG_SIZE} * i)}$

L = L or X

END FOR

mode = random%2

IF (mode==1) L = -L

return L

2.2 Realizacja C++/BOOST

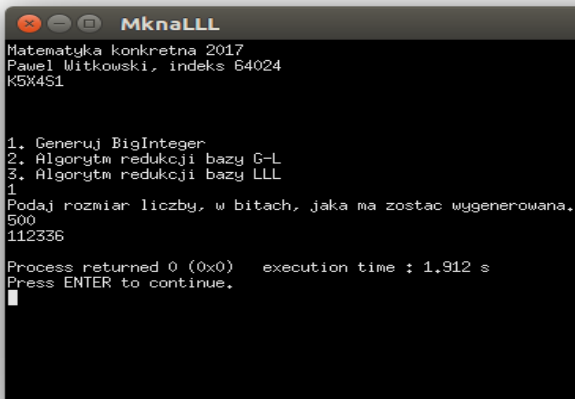
Implementacja w języku C++, z wykorzystaniem biblioteki BOOST.

Typ `boost::multiprecision::cpp_int` to nieograniczony z góry integer, rozszerzający się w miarę potrzeby.

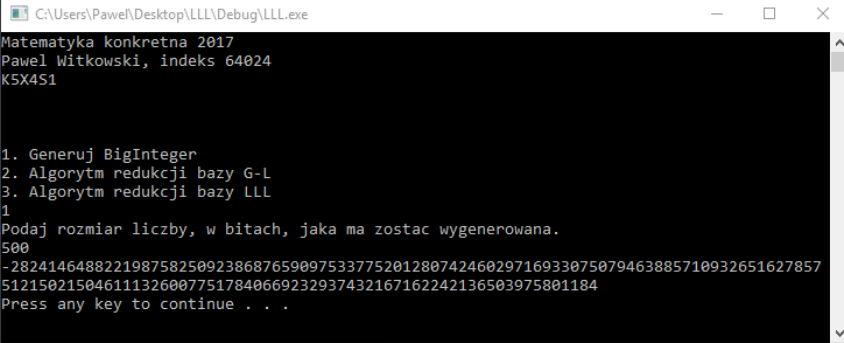
```
cpp_int Generuj_BigInteger(int n) {
    cpp_int BigInt = 0, y;
    cpp_int Iterations = n / 16;
    short x;
    if (Iterations == 0) {
        x = rand();
        return x;
    }
    else {
        for (int i = 1; i <= Iterations; i++) {
            x = rand();
            y = Szybkie_pot(2, 15) | x;
            BigInt |= (y << (n - (16 * i)));
        }
        short rd = rand() % 2;
        if (rd == 1) BigInt *= (-1);
        return BigInt;
    }
}
```

Adnotacja:

Poniższe zrzuty ekranowe zostały wykonane na implementacji pod systemem Windows i Linux. Ten sam kod zachowywał się inaczej na Linuksie i Windowsie. Nie wiem czy jest to spowodowane różnicą w kompilatorach (GNU/MSVC) czy systemie. Zamieszczam zrzuty ekranowe:



```
temp *= temp;
}
return value;
}
cpp_int Generuj_BigInteger(int n) {
    cpp_int BigInt = 0, y;
    cpp_int Iterations = n / 16;
    short x;
    if (Iterations == 0) {
        x = rand();
        return x;
    }
    else {
        for (int i = 1; i <= Iterations; i++) {
            x = rand();
            y = Szybkie_pot(2, 15) | x;
            BigInt |= (y << (n - (16 * i)));
        }
        short rd = rand() % 2;
        if (rd == 1) BigInt *= (-1);
        return BigInt;
    }
}
```



```
cpp_int Generuj_BigInteger(int n) {
    cpp_int BigInt = 0, y;
    cpp_int Iterations = n / 16;
    short x;
    if (Iterations == 0) {
        x = rand();
        return x;
    }
    else {
        for (int i = 1; i <= Iterations; i++) {
            x = rand();
            y = Szybkie_pot(2, 15) | x;
            BigInt |= (y << (n - (16 * i)));
        }
        short rd = rand() % 2;
        if (rd == 1) BigInt *= (-1);
        return BigInt;
    }
}
```

3. Redukcja bazy Gauss'a – Legendre'a

Zadana jest krata pełnego rzędu w \mathbf{R}^2 (\mathbf{Z}^2 bez zmniejszania ogólności). Możliwe jest rozwiązanie problemu, klasy NP, znalezienia wektora najkrótszego, sprawdzając czy zadane wektory są redukcją Gaussa – Legendre'a (dalej G – L). To znaczy, że nie ma od nich wektorów krótszych, a więc realizują dwa kolejne minima

Zadane wektory są redukcją G – L, wtedy i tylko wtedy, kiedy spełniony zostaje warunek:

$$\|v_1\| < \|v_2\| \leq \|v_2 - qv_1\|, q \in \mathbf{Z}$$

Co więcej, wystarczy założyć $q \in \{-1, 1\}$, a powyższy warunek zostanie spełniony.

$$\|v_1\| < \|v_2\| \leq \|v_2 \pm v_1\| \Leftrightarrow \|v_1\| < \|v_2\| \leq \|v_2 - qv_1\|, q \in \mathbf{Z}$$

D-d (\Rightarrow) oczywisty

D-d (\Leftarrow)

Przekształcając w funkcje kwadratową otrzymujemy równanie funkcji w postaci:

$$F(q) = \|v_2 - qv_1\|^2, q \in \mathbf{R}$$

$$F(q) = \|v_2\|^2 - 2q \langle v_1, v_2 \rangle + q^2 \|v_1\|^2$$

Po podstawieniu do powyższego równania $q = 0$ \vee $q = 1$ \vee $q = -1$ możemy wysunąć wniosek, że:

$$F(0) \leq F(-1) \wedge F(0) \leq F(1)$$

Co dowodzi, że $\min F(q) \in (-1, 1)$

Z powyższego widzimy, że najmniejsze wartości leżą w tym przedziale, dlatego jeżeli q się w nim znajdzie, będzie spełniony ogólniejszy warunek.

3.1 Algorytm redukcji/Pseudokod

Algorytm opiera się na ciągłym ortogonalizowaniu bazy oraz jej przekrzywianiu, tj. zaokrągłaniu do najbliższej całkowitej tak, aby wektor wpadł do tej samej bazy.

Zakładamy, że

$$\|v_1\| < \|v_2\|$$

Wektor v_1 zortogonalizować względem wektora v_2 . Do ortogonalizacji używamy ilorazu $\frac{\langle v_1, v_2 \rangle}{\|v_1\|^2}$, co powoduje powstanie ułamka. Krata jest w \mathbf{Z} dlatego należy „przekrzywić” zortogonalizowany wektor, tj. zaokrąglić ten ułamek do najbliższej liczby całkowitej tak, aby wektor wpadł do kraty.

Algorytm należy powtarzać aż do momentu, gdy $q = \left\lfloor \frac{\langle v_1, v_2 \rangle}{\|v_1\|^2} \right\rfloor = 0$, wtedy nie da się już zredukować bazy, ponieważ $v_2 = v_2 - qv_1 \Rightarrow v_2 = v_2$.

Input: v_1, v_2

LOOP

IF $\|v_2\| < \|v_1\|$ swap (v_1, v_2)

$$q = \left\lfloor \frac{\langle v_1, v_2 \rangle}{\|v_1\|^2} \right\rfloor$$

IF $q \neq 0$

$$v_2 = v_2 - qv_1$$

ELSE

return $v_1, v_2 \rightarrow \min(\lambda_1, \lambda_2)$

END LOOP

3.2 Realizacja C++/BOOST

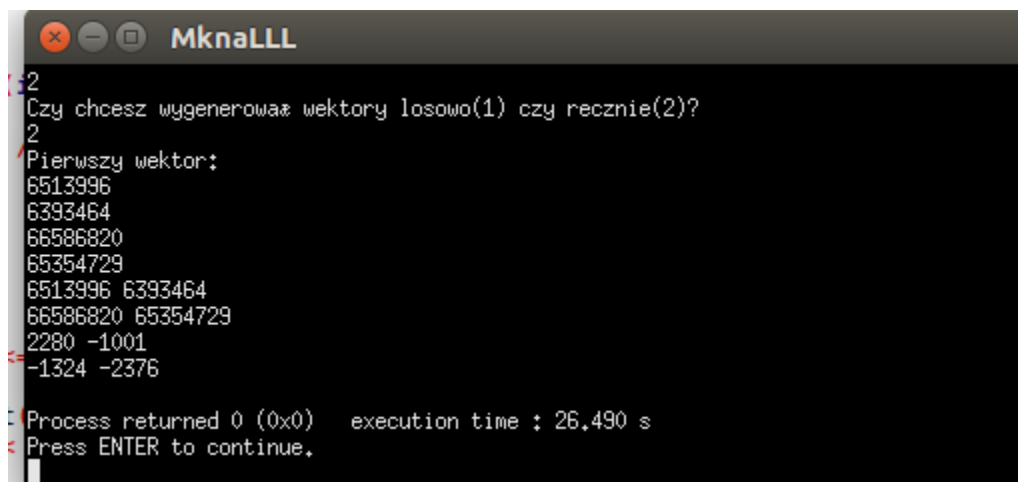
Implementacja w języku C++, z wykorzystaniem biblioteki BOOST.

Typ `boost::multiprecision::cpp_int` to nieograniczony z góry integer, rozszerzający się w miarę potrzeby.

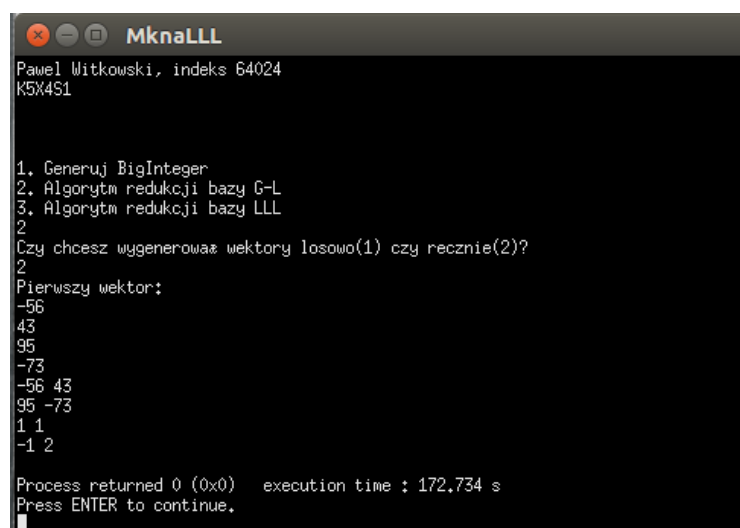
Funkcja `Iloczyn_Skalarny` jest trywialna, nie wymaga wyjaśnienia.

Funkcja `boost_round` została napisana przeze mnie, polega na zaokrągleniu do najbliższej całkowitej ilorazu liczb podanych w nawiasie. Nie występuje tam rzutowanie na typ zmiennoprzecinkowy.

```
void GL(vector<cpp_int> &v1, vector<cpp_int> &v2) {
    cpp_int m;
    do {
        if (Iloczyn_Skalarny(v2, v2) < Iloczyn_Skalarny(v1, v1)) v1.swap(v2);
        m = boost_round(Iloczyn_Skalarny(v1, v2), Iloczyn_Skalarny(v1, v1));
        if (m!=0) v2 = Odejmowanie_Wektora(v2, v1, m);
    } while (m != 0);
}
```



```
MknaLLL
2
Czy chcesz wygenerować wektory losowo(1) czy ręcznie(2)?
2
Pierwszy wektor:
6513996
6393464
66586820
65354729
6513996 6393464
66586820 65354729
2280 -1001
-1324 -2376
Process returned 0 (0x0)   execution time : 26.490 s
Press ENTER to continue.
```



```
MknaLLL
Paweł Witkowski, indeks 64024
K5X4S1

1. Generuj BigInteger
2. Algorytm redukcji bazy G-L
3. Algorytm redukcji bazy LLL
2
Czy chcesz wygenerować wektory losowo(1) czy ręcznie(2)?
2
Pierwszy wektor:
-56
43
95
-73
-56 43
95 -73
1 1
-1 2
Process returned 0 (0x0)   execution time : 172.734 s
Press ENTER to continue.
```

4. Redukcja bazy Lenstra–Lenstra–Lovász

Dla krat o wymiarze większym niż dwa, nie jest możliwe zastosowanie algorytmu **Gauss’a – Legendre’a**, ponieważ wraz ze wzrostem rozmiaru, problem znalezienia wektora najkrótszego jest znacznie trudniejszy. Z pomocą przychodzi algorytm opracowany przez Panów, od nazwisk który wziął swoją nazwę.

Celem całej procedury jest znalezienie lepszej bazy, czyli takiej w której znajdują się najkrótsze wektory. Dodatkowo, dana baza będzie najbardziej ortogonalna jak się da, nie wychodząc ze zbioru liczb całkowitych, i to będziemy rozumieć dalej przez „dobrą bazę”. Jak najkrótsze wektory się w niej znajdują i której współczynnik Hadamara jest jak najwyższy.

Należy pamiętać, że zortogonalizowana baza, nie jest bazą tej samej kraty, ponieważ nie jest umieszczona w liczbach całkowitych. Należy zaokrąglić wartości wektorów do najbliższej liczby całkowitej, tak aby wartość należała do **Z**.

Procedura algorytmu opiera się na redukcji rozmiaru, ustawianiu odpowiednich kątów między wektorami oraz sprawdzaniu czy rozmiar wektorów jest odpowiedni. Do tego służą dwa warunki podane poniżej:

$$\textbf{Warunek rozmiaru} \quad |\mu_{i,j}| \leq 0.5 \quad \text{dla } 1 \leq j < i \leq n$$

$$\textbf{Warunek Lovasza} \quad \|v_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|v_{i-1}^*\|^2 \quad \text{dla } 1 < i \leq n.$$

Gdy wszystkie wektory będą spełniać powyższe warunki, będziemy mogli powiedzieć, że jest to baza LLL – zredukowana oraz, że zachodzi

$$\|v_1\| \leq \Psi(n) \|v_{\min}\|, \text{ gdzie przyjmujemy } \Psi(n) = 2^{\frac{n-1}{2}}$$

Złożoność obliczeniowa algorytmu LLL jest $O(\log n)$.

4.1 Algorytm redukcji LLL/Pseudokod

Przed rozpisanem pseudokodu, należy wspomnieć, że sprawdzanie warunku rozmiaru jest zbyteczne. Dzieje się tak, ponieważ odpowiednie $\mu_{i,j}$ zawsze będzie spełniało nierówność.

Najpierw dokonywana jest ortogonalizacja, następnie jest przekrzywienie i ponowna ortogonalizacja w celu pozyskania nowego $\mu_{i,j}$, które można nazwać współczynnikiem odchylenia. Łatwo zauważyć, że po ponownej ortogonalizacji będzie ono bardzo małe z racji małego odchylenia. Oczywiście pamiętamy, że ortogonalizacja „surowych” wektorów, jak i tych już przekrzywionych po pierwszym prostowaniu da te same wektory. Co można zapisać w sposób następujący

$v_i^* = \bar{v}_i^*$ dla $1 \leq i \leq n$, gdzie za wektor z daszkiem rozumiemy „prawie ortogonalny, taki co wpada do \mathbf{Z} ”

input: $v_1, v_2, \dots, v_n \in \mathbf{Z}^n$

output: baza LLL zredukowana

k=2

WHILE k ≤ n

FOR j=k-1 TO 1 DO

GRAM-SCHMIDT //w celu zdobycia współczynników odchylenia

$q = \lfloor \mu_{kj} \rfloor$ //zaokrąglenie, żeby wektor wpadł do \mathbf{Z}

$\bar{v}_k = \bar{v}_k - q\bar{v}_j$ //skracać długość wektora

END FOR

GRAM-SCHMIDT //w celu uaktualnienia współczynników odchylenia

IF $\|v_k^*\|^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) \|v_{k-1}^*\|^2$

k = k + 1 //jeżeli przeszło war Lav. to można rozpatrywać następny wektor

ELSE

swap(\bar{v}_k, \bar{v}_{k-1}) //wpw. zamień wektory i cofnij sprawdzanie

k = max(2, k - 1)

END IF

END WHILE

4.2 Realizacja C++

- Macierze „ort”, oraz „mi” zawierają kolejno: zortogonalizowaną aktualną bazę oraz współczynniki odchyłeń.
- Funkcja „Ortogonalizacja” zwraca zortogonalizowany układ wektorów podanych w argumencie oraz wszystkie współczynniki odchyłeń do macierzy „mi”.
- Funkcja „Odejmowanie_Wektora” zwraca wektor, będący wynikiem odejmowania od pierwszego wektora w argumencie, drugiego pomnożonego przez „t”.
- Funkcje „max”, „pow”, „Iloczyn skalarny” są trywialne i nie wymagają wyjaśnienia.

```
void LLL(vector < vector <long double> > &base) {
    vector < vector <long double> > ort;
    vector <vector<long double> > mi;
    vector <long double>k_values;
    for (int i = 0; i < base.size(); i++) {
        vector <long double> temp;
        for (int j = 0; j < base[i].size(); j++) {
            temp.push_back(0);
        }
        mi.push_back(temp);
    }
    int k = 1;
    long double t;
    long double Left_Lovasz, Right_Lovasz;
    int swaps = 0;
    int iterator = 0;
    while (k < base.size()) {
        k_values.push_back(k + 1);
        for (int j = k - 1; j >= 0; j--) {
            ort = Ortogonalizacja(base, mi);
            t = floor(mi[k][j] + 0.5);
            base[k] = Odejmowanie_Wektora(base[k], base[j], t);
        }

        ort = Ortogonalizacja(base, mi);
        Left_Lovasz = Iloczyn_Skalarny(ort[k], ort[k]);
        Right_Lovasz = 0.75 - pow(mi[k][k - 1], 2);
        Right_Lovasz *= Iloczyn_Skalarny(ort[k - 1], ort[k - 1]);
        if (Left_Lovasz > Right_Lovasz) {
            k++;
        }
        else {
            base[k - 1].swap(base[k]);
            swaps++;
            k = max(k - 1, 1);
        }
        iterator++;
    }
    cout << endl;
    cout << "k's : ";
    show_vector(k_values);
    cout << "swaps: " << swaps << endl;
    cout << "iterations: " << iterator << endl;
}
```

```
MknaLLL
Czy chcesz samemu wprowadzić dane(1) czy wygenerować bazę losowo(2)?
1
1 wektor:
1
0
0
2 wektor:
4
2
15
3 wektor:
0
0
3
1 0 0
4 2 15
0 0 3

k's : 2 3 2 3 2 3
swaps: 2
iterations: 6
1 0 0
0 2 0
0 0 3
```

```
MknaLLL
9
1
29
19 2 32 46 3 33
15 42 11 0 3 24
43 15 0 24 4 16
20 44 44 0 18 15
0 48 35 16 31 31
48 32 32 9 1 29

k's : 2 2 3 2 3 4 3 2 2 3 4 5 4 3 2 3 4 5 4 3 4 5 6 5 4 3 4 5 6 5 4 3 2 2 3 2 3 4 5 6
swaps: 19
iterations: 40
7 -14 -8 4 19 9
-20 4 -9 16 13 16
5 2 33 0 15 -9
-6 -8 -20 -21 8 -12
-10 -25 21 -15 -6 -11
7 1 -9 -11 1 31
```

5. Łamanie NTRU za pomocą algorytmu LLL

Znając klucz publiczny $(N, p, q, h(x))$ można użyć algorytmu LLL do obliczenia klucza prywatnego. Należy stworzyć tzw. „Macierz NTRU” o postaci:

$$M_h^{NTRU} = \begin{bmatrix} I & h \\ 0 & qI \end{bmatrix}$$

Która ma wymiary $2N \times 2N$, oraz rząd $2N$.

Po zastosowaniu algorytmu redukcji bazy, na tejże macierzy uzyskamy aproksymatywny wektor najkrótszy. Z racji, że każdy wiersz macierzy to $2N$, to składa się on z dwóch wektorów o rozmiarach po N . Zostanie spełnione przybliżenie:

$$[f, g] \sim SVP \text{ lub } [f, g] \sim -SVP$$

Po uzyskaniu $[f, g]$ należy wykonać sprawdzenie czy zachodzi przystawanie:

$$fh \equiv g \pmod{q}$$

Jeżeli kongruencja jest spełniona, to $[f, g]$ jest kluczem prywatnym, bądź jego rotacją. Można wyznaczyć wielomian odwrotny do f tj $F_p \pmod{q}$ i rozpocząć odszyfrowywanie wiadomości.

- Wymnożyć $e \cdot f \pmod{q}$
- Na wyniku wykonać centralne podniesienie
- Wkładać do \mathbf{R}_p
- Następnie wymnożyć przez $F_p \pmod{p}$
- Centralne podniesienie
- Uzyskany wynik powinien być odszyfrowaną wiadomością.

5.1 Użycie LLL na przykładzie zadanego NTRU

$(N, p, q) = (11, 3, 97)$

$$h(x) = 39 + 9x + 33x^2 + 52x^3 + 58x^4 + 11x^5 + 38x^6 + 6x^7 + x^8 + 48x^9 + 41x^{10}$$

$$e(x) = 52 + 50x + 50x^2 + 61x^3 + 61x^4 + 7x^5 + 53x^6 + 46x^7 + 24x^8 + 17x^9 + 50x^{10}$$

1	0	0	0	0	0	0	0	0	0	39	9	33	52	58	11	38	6	1	48	41
0	1	0	0	0	0	0	0	0	0	41	39	9	33	52	58	11	38	6	1	48
0	0	1	0	0	0	0	0	0	0	48	41	39	9	33	52	58	11	38	6	1
0	0	0	1	0	0	0	0	0	0	1	48	41	39	9	33	52	58	11	38	6
0	0	0	0	1	0	0	0	0	0	6	1	48	41	39	9	33	52	58	11	38
0	0	0	0	0	1	0	0	0	0	38	6	1	48	41	39	9	33	52	58	11
0	0	0	0	0	0	1	0	0	0	11	38	6	1	48	41	39	9	33	52	58
0	0	0	0	0	0	0	1	0	0	58	11	38	6	1	48	41	39	9	33	52
0	0	0	0	0	0	0	0	1	0	52	58	11	38	6	1	48	41	39	9	33
0	0	0	0	0	0	0	0	0	1	33	52	58	11	38	6	1	48	41	39	9
0	0	0	0	0	0	0	0	0	0	1	9	33	52	58	11	38	6	1	48	41
0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	97

6. Wnioski

Po zakończeniu pracy LLL otrzymałem następujące wyniki:

```
MknaLLL
k's : 2 2 3 2 3 4 3 4 5 4 5 6 5 6 7 6 7 8 7 8 9 8 9 10 9 10 11 10 11 12 11 10 9
8 9 10 11 12 11 10 9 8 7 6 5 4 3 2 2 3 4 5 6 5 4 3 2 3 4 5 6 7 6 5 6 7 8 7 6 7 8
9 8 7 8 9 10 11 12 11 10 9 8 9 10 11 10 11 12 13 12 11 10 9 8 7 6 5 4 3 2 2 3 4
5 6 7 6 5 4 5 6 5 6 7 6 7 8 7 6 5 4 3 4 5 6 7 8 9 8 7 8 9 10 9 10 11 10 9 8 7 6
5 4 5 6 7 8 9 10 11 12 11 12 13 12 11 10 9 8 7 6 5 4 3 2 2 3 2 3 4 3 4 5 4 5 6
7 8 9 10 11 12 13 12 11 10 9 8 7 6 5 4 5 6 7 8 9 10 11 12 13 14 13 12 11 10 9 8
7 6 7 8 9 10 9 10 11 10 9 10 11 12 13 12 11 10 11 12 13 14 13 12 11 10 9 8 7 8 9
10 11 12 13 14 13 12 11 12 13 14 13 14 15 14 13 12 11 10 9 8 7 6 5 4 3 2 2 3 4
5 6 7 8 9 10 11 12 11 12 13 12 11 10 9 8 7 6 5 4 5 6 7 8 9 8 7 8 9 10 11 12 13 1
4 13 12 11 10 9 8 7 6 5 4 3 4 5 6 7 8 9 10 11 10 11 12 13 14 15 14 13 12 11 12 1
3 14 15 14 13 12 11 10 9 8 7 6 5 4 3 2 3 4 3 4 5 6 7 8 9 10 11 12 13 14 15 16 15
14 13 12 11 10 9 8 7 6 5 4 5 6 7 6 7 8 9 8 7 8 9 10 11 10 9 8 9 10 11 10 11 12
11 12 13 12 11 10 9 10 11 12 13 14 13 12 11 10 9 8 7 6 5 6 7 6 7 8 9 10 9 8 7 8
9 10 11 12 13 14 13 12 11 10 11 12 13 14 15 16 17 16 15 14 13 12 11 10 9 8 9 10
11 12 13 14 15 14 13 12 11 10 9 10 11 12 13 14 13 14 15 14 15 16 15 16 17 16 15
14 13 12 11 10 11 12 13 12 11 12 13 14 13 14 15 14 15 16 15 16 17 16 17 18 17 16
15 14 13 12 11 10 9 8 7 6 5 4 3 2 2 3 4 5 6 7 8 9 10 9 8 7 6 7 8 7 8 9 10 11 12
13 14 13 14 15 14 15 16 15 16 17 16 15 16 17 18 17 16 15 14 13 12 11 10 9 8 9 1
0 11 12 11 10 9 8 7 6 5 4 3 2 3 4 5 6 7 8 9 10 11 12 11 12 13 12 13 14 15 14 13
12 11 10 11 12 13 14 15 16 15 14 13 14 15 16 17 18 19 18 17 16 15 14 13 12 11 10
9 8 9 10 11 12 13 14 13 12 13 14 15 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 3 4 5
6 7 8 9 10 11 10 11 12 13 12 11 12 13 14 13 14 15 14 15 16 15 14 13 14 15
16 17 18 19 18 17 16 17 18 19 20 19 18 17 16 15 14 13 12 13 14 13 14 15 16 15 1
17 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 4 5 6 7 8 7 8 9 10 11 12 11 10 9 8
9 10 11 12 13 12 11 10 9 8 7 6 5 4 5 6 7 8 9 10 11 10 11 12 13 14 15 16 17 16 1
16 17 18 19 18 17 16 17 18 19 20 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
6 7 8 9 10 11 10 11 12 13 12 11 12 13 14 13 12 13 14 15 14 15 16 15 16 17 16 15
14 13 12 11 10 9 8 7 6 5 4 3 2 2 3 4 5 6 7 8 9 10 11 12 13 12 13 14 15 16 17 18
19 20 21 22 21 20 19 18 17 16 15 14 13 14 15 16 17 18 19 18 19 20 19 20 21 22 21
20 19 18 17 16 15 14 13 12 11 10 9 8 7 8 9 10 11 10 9 8 9 10 11 12 13 14 15 16
17 18 19 20 19 20 21 20 21 22 21 20 19 18 17 18 19 20 21 22

swaps: 467
iterations: 948
-2 -4 0 3 -4 -1 2 7 1 -2 -2 -2 1 -6 1 2 4 3 3 4 -1 -2
4 1 1 3 -3 0 -3 2 -1 -2 0 1 3 -7 -1 3 -8 -1 5 -4 6 -4
1 -1 -2 4 -2 -2 1 -4 8 0 -1 4 6 -1 8 -6 -5 -3 -6 4 0 -8
6 2 3 1 4 -1 -4 1 -8 -1 -1 3 -1 -7 -5 -1 6 -4 4 -7 5 0
0 -2 -1 -4 5 3 0 5 -8 -2 -2 -6 6 1 -3 4 6 2 3 4 2 2
-6 -1 -2 -9 3 0 3 -2 2 7 1 -4 1 9 -1 1 -1 2 3 -4 6 2
5 -2 -2 3 -3 2 2 0 4 -6 -1 4 1 -3 5 -7 -3 -3 6 2 -3 -6
-4 0 5 5 0 -1 -1 4 0 -1 -5 0 -2 -2 -3 -6 6 -2 -6 8 0 0
2 -5 -3 -6 0 4 5 1 0 -3 1 -5 -1 -4 3 5 1 4 -1 2 3 7
8 -1 6 1 0 -6 3 -3 -1 2 -1 2 -3 -4 2 -9 -4 -1 -3 -4 0 -4
-1 3 1 1 -2 4 -7 4 1 1 -3 1 8 5 0 2 -10 -3 -4 -4 -1 -1
0 -2 -4 -2 4 0 3 -1 1 4 -1 -5 2 10 5 -2 -1 -7 -2 0 -5 -2
3 3 -3 -2 5 2 -5 4 -9 -1 3 0 0 -1 0 1 0 -6 6 -1 2 -1
5 4 3 -5 -1 5 -1 -3 1 -3 -7 -1 4 5 3 -3 -4 1 -4 4 -1 3
5 -2 -5 -5 -4 5 -2 -2 7 1 0 2 5 -5 2 7 -7 -1 -1 6 2 -3
3 -4 1 -9 -4 -2 4 0 5 0 0 2 3 6 4 -1 -1 3 1 6 -6 4
3 -4 1 1 -3 -4 2 3 -3 -5 0 -2 -4 -8 -4 6 -7 0 1 -1 -3 5
3 -3 4 -4 2 4 -5 3 -3 2 1 -4 5 -5 -4 -6 -2 -1 -2 1 4 0
4 -1 -3 -3 -4 0 -1 -5 6 -4 -2 -1 -2 1 1 -2 -1 -2 1 0 2 0
3 -1 1 -2 2 -6 -3 2 3 1 2 2 2 2 1 2 -3 -3 5 -10 -1 -4
3 4 0 1 5 -6 4 2 -4 1 3 -1 2 1 2 -1 0 -2 0 1 2 -1
2 -3 1 7 3 -3 -2 1 -6 3 -3 1 4 -1 -2 0 2 3 -3 -1 -4 1

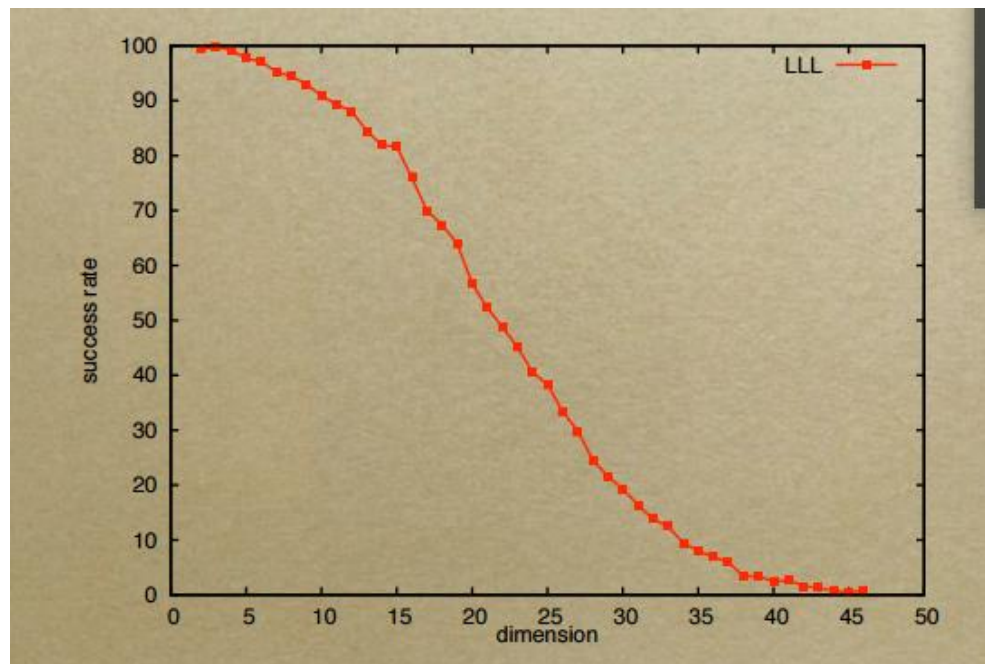
Zostalo spelnione f*h = g (mod q)
f: 95 93 0 3 93 96 2 7 1 95 95
g: 95 1 91 1 2 4 3 3 4 96 95
m: 0 0 0 0 -1 -1 0 1 -1 -1 1
```

Mimo, że przystawanie zostało spełnione, nie zgadzają mi się wielomiany f i g . Powinny to być wielomiany trinarne. W programie nie zamieszczam odwracania wielomianu w ciele, ponieważ nie udało mi się napisać jego stabilnej wersji, tak aby poprawnie zawsze działała. Dlatego dla konkretnego przypadku, jest zadeklarowany „na twardo”. Jednocześnie trzeba zaznaczyć, że złamany klucz prywatny może być rotacja oryginalnego, co spowoduje przejście warunku kongruencji, ale da niewłaściwy wynik.

Niedokładność redukcji NTRU jest prawdopodobnie spowodowana rozmiarem. Dla coraz to większych rozmiarów macierzy maleje jego skuteczność.

Opisuje to wykres dotyczący LLL, który znalazłem w internecie.

Jest on w prezentacji Pana Phong Nguyễn'a (?).



Phong Nguyễn, „The LLL Algoritm” (2010).

Nie znaczy to, że nie jest możliwe za pomocą algorytmu LLL rozwiązanie NTRU, jest to po prostu zadanie znacznie utrudnione z powodu coraz to większego przybliżenia.

Wcześniej zostało powiedziane, że LLL znajduje taką bazę, gdzie zachodzi:

$$\|v_1\| \leq \Psi(n)\|v_{\min}\|, \text{ gdzie przyjmujemy } \Psi(n) = 2^{\frac{n-1}{2}}$$

Im większy wymiar bazy, tym funkcja $\Psi(n)$ przyjmuje większe wartości więc przybliżenie staje się niedokładne.

Nie jestem w stanie podać czy zadany pierwszy wektor w bazie LLL zredukowanej jest szukany klucz prywatny, ale wokół niego kręci się algorytm więc z dużym prawdopodobieństwem tak jest. Odchyły są spowodowane coraz to większym wymiarem.

Przy opracowaniu tego sprawozdania pomocne okazały się pdf:

- „The LLL Algoritm” (2010) - Phong Nguyễn.
- „Introduction to Mathematical Cryptography”(2000) - Jospeh H. Silverman