Project Lattices

Βασίλειος Τσιτώτας
ΑΕΜ: 225

20 Ιανουαρίου 2026

Κατάλογος Αρχείων Κώδικα ανά Άσκηση

Παραθέτω τη λίστα με τα αρχεία κώδικα Python που υλοποίησα και χρησιμοποίησα για την επίλυση των ασκήσεων, όπως αυτά είναι οργανωμένα στο φάκελο του έργου (`Lattices_project`):

1. **Άσκηση 1:** –
2. **Άσκηση 2:** –
3. **Άσκηση 3:** `ex3_gauss_lagrange.py`, `ex3_gauss_lagrange2.0.py`
4. **Άσκηση 4:** `ex4_LLL_1.py`, `ex4_LLL_2.py`, `ex4_LLL_3.py`, `ex4_LLL_4.py`
5. **Άσκηση 5:** `ex5_Knapsack.py`
6. **Άσκηση 6:** `ex6_Coppersmith.py`
7. **Άσκηση 7:** –
8. **Άσκηση 8:** `ex8_Matrixes_i.py`, `ex8_Matrixes_ii.py`, `ex8_Matrixes_iii.py`, `ex8_Matrixes_iv.py`
9. **Άσκηση 9:** –
10. **Άσκηση 10:** `ex10_Babai.py`
11. **Άσκηση 11:** `ex11_KFP.py`
12. **Άσκηση 12:** `ex12_Schnorr-Euchner.py`

1 Άσκηση 1

Πλέγμα (*Lattice*)

Ένα πλέγμα μπορούμε να το φανταστούμε ως ένα κανονικό «δίκτυο» σημείων που δημιουργείται αν ξεκινήσουμε από το μηδέν και προσθέσουμε ακέραια πολλαπλάσια από κάποια σταθερά διανύσματα, τα οποία ονομάζονται *διανύσματα βάσης*.

Ένα υποσύνολο $L \subset \mathbb{R}^n$ το καλούμε **πλέγμα**, αν υπάρχουν γραμμικώς ανεξάρτητα διανύσματα $b_1, b_2, \ldots, b_k \in \mathbb{R}^n$ (με $k \leq n$), τέτοια ώστε:

$$L = L(b_1, b_2, \ldots, b_k) = \left\{ \sum_{j=1}^{k} \alpha_j b_j : \alpha_j \in \mathbb{Z} \right\}$$

Τα διανύσματα $b_1, \ldots, b_k$ ονομάζονται **βάση** του πλέγματος L.

Διαδοχικά Ελάχιστα (*Successive Minima*)

Ορίζουμε τα **διαδοχικά ελάχιστα** ενός πλέγματος L ως τις μικρότερες τιμές/οι ελάχιστες ακτίνες $\lambda_i(L)$ (για $i = 1, 2, \ldots, n$), τέτοιες ώστε η σφαίρα $B(0, \lambda_i)$ να περιέχει τουλάχιστον i γραμμικά ανεξάρτητα διανύσματα του L.

Δηλαδή,

$\lambda_1(L)$: το μήκος του μικρότερου μη μηδενικού διανύσματος του πλέγματος,

$\lambda_2(L)$: η μικρότερη ακτίνα που περιέχει δύο ανεξάρτητα διανύσματα,

$\vdots$

$\lambda_n(L)$: η μικρότερη ακτίνα που περιέχει n ανεξάρτητα διανύσματα.

Έχουμε επομένως:

$$\lambda_1(L) \leq \lambda_2(L) \leq \cdots \leq \lambda_n(L)$$

Τα διαδοχικά ελάχιστα εκφράζουν πόσο «πυκνό» ή «αραιό» είναι το πλέγμα σε κάθε διάσταση.

Πρόβλημα Συντομότερου Διανύσματος (*Shortest Vector Problem – SVP*)

Στο πρόβλημα SVP ζητάμε να βρούμε το μη μηδενικό διάνυσμα του πλέγματος με το μικρότερο δυνατό μήκος. Δηλαδή, επιδιώκουμε να βρούμε

$$v \in L \setminus \{0\} \quad \text{τέτοιο ώστε} \quad \|v\| = \lambda_1(L)$$

Πρόβλημα Κοντινότερου Διανύσματος (*Closest Vector Problem – CVP*)

Στο πρόβλημα CVP δίνεται ένα πλέγμα $L \subset \mathbb{R}^m$ και ένα διάνυσμα-στόχος $t \in \mathbb{R}^m$. Ζητάμε να βρούμε ένα διάνυσμα $x \in L$ τέτοιο ώστε:

$$\|x - t\| \leq \gamma \|y - t\|, \quad \forall y \in L$$

όπου $\gamma \geq 1$. Αν $\gamma = 1$, έχουμε το **ακριβές** CVP, ενώ για $\gamma > 1$ έχουμε το **προσεγγιστικό** CVP.

Στόχος μας, λοιπόν, είναι να εντοπίσουμε ποιο σημείο του πλέγματος βρίσκεται πιο κοντά σε ένα δοσμένο εξωτερικό σημείο t.

Πρόβλημα Συντομότερων Ανεξάρτητων Διανυσμάτων (*Shortest Independent Vectors Problem – SIVP*)

Στο πρόβλημα $SIVP$ θεωρούμε ένα πλέγμα L διάστασης n και ζητάμε να βρούμε n γραμμικά ανεξάρτητα διανύσματα $x_1, x_2, \ldots, x_n \in L$ που ελαχιστοποιούν:

$$M = \max_i \|x_i\|$$

Αν επιτρέψουμε έναν παράγοντα προσέγγισης $\gamma > 1$, τότε έχουμε το προσεγγιστικό πρόβλημα $SIVP_\gamma$.

Με άλλα λόγια, στο $SIVP$ επιδιώκουμε να βρούμε μια βάση του πλέγματος με όσο το δυνατόν μικρότερες αποστάσεις από το μηδέν, δηλαδή μια «σφιχτή» και καλά ισορροπημένη βάση.

2 'Ασκηση 2

2.1 Συνάρτηση i is point in lattice

Algorithm 1 Συνάρτηση *is_point_in_lattice*(B, P)

```
1: procedure is_point_in_lattice(B, P)
2:     // Λύση του συστήματος Bx = P με ακέραιους περιορισμούς
3:     x ← solve_integer_system(B, P)
4:
5:     // Έλεγχος ύπαρξης ακέραιας λύσης
6:     if x είναι ακέραιο διάνυσμα then
7:         return True
8:     else
9:         return False
10:    end if
11: end procedure
```

Η λειτουργία του αλγορίθμου *Lattices Problems i* βασίζεται σε τρία βασικά βήματα:

1. **Επίλυση γραμμικού συστήματος:** Χρησιμοποιεί QR αποσύνθεση για να λύσει το σύστημα Bx = P, βρίσκοντας τους πραγματικούς συντελεστές x που εκφράζουν το P ως γραμμικό συνδυασμό των διανυσμάτων βάσης.
2. **'Ελεγχος ακεραιότητας:** Στρογγυλοποιεί τους συντελεστές x στους πλησιέστερους ακέραιους και ελέγχει αν ο γραμμικός συνδυασμός με αυτούς τους ακέραιους συντελεστές αναπαράγει το σημείο P.
3. **Εκτεταμένος έλεγχος:** Σε περίπτωση αποτυχίας, ψάχνει σε γειτονικές ακέραιες λύσεις μεταβάλλοντας κάθε συντελεστή κατά ± 1, για να καλύψει περιπτώσεις όπου η στρογγυλοποίηση μπορεί να οδηγήσει σε λάθος αποτέλεσμα λόγω αριθμητικών σφαλμάτων.

2.2 Συνάρτηση ii lattice intersection

Algorithm 2 Συνάρτηση lattice_intersection(B1, B2)

```
1: procedure LATTICE_INTERSECTION(B1, B2)
2:     Βήμα 1: Εύρεση κοινού χώρου στηλών
3:     A = [B1| − B2]                          ▷ Σύνθεση επαυξημένου πίνακα
4:
5:     Βήμα 2: Επίλυση ομογενούς συστήματος Ax = 0
6:     για εύρεση ακεραίων λύσεων
7:     Z = find_integer_solutions(A)
8:
9:     Βήμα 3: Διαχωρισμός λύσεων σε δύο μέρη
10:    Οι λύσεις έχουν μορφή [x; y] όπου B1*x = B2*y
11:    Z1 = πρώτες n_1 γραμμές του Z
12:
13:    Βήμα 4: Υπολογισμός βάσης τομής
14:    intersection_basis = B1 × Z1
15:
16:    Βήμα 5: Απομόνωση γραμμικά ανεξάρτητων διανυσμάτων
17:    basis = extract_linear_independent(intersection_basis)
18:
19:    return basis
20: end procedure
```

Επεξήγηση Λειτουργίας

1. **Δημιουργία Επαυξημένου Πίνακα:** Σχηματίζουμε τον πίνακα $[B_1|-B_2]$ που κωδικοποιεί την εξίσωση $B_1x = B_2y$.

2. **Εύρεση Μηδενοχώρου:** Ο μηδενοχώρος του επαυξημένου πίνακα περιέχει όλες τις λύσεις $[x; y]$ που ικανοποιούν $B_1x = B_2y$.

3. **Διαχωρισμός Λύσεων:** Χρησιμοποιούμε τις λύσεις x για να παραγάγουμε τα διανύσματα τομής μέσω του πολλαπλασιασμού B_1x.

4. **Εξαγωγή Βάσης:** Με QR αποσύνθεση απομονώνουμε τα γραμμικά ανεξάρτητα διανύσματα που αποτελούν βάση για την τομή.

2.3 Συνάρτηση iii extended gcd

Υπολογισμός βάσης για το πλέγμα $L = \{x \in \mathbb{Z}^m : a \cdot x = 0\}$ Δοθέντος ενός ακέραιου διανύσματος $a \in \mathbb{Z}^m$, ζητείται να υπολογιστεί μια ακέραια βάση για το πλέγμα

$$L = \{x \in \mathbb{Z}^m : a \cdot x = 0\}.$$

Ψευδοκώδικας:

```
function EXTENDED_GCD(a, b)
    if b = 0 then
        return (|a|, 1, 0)
    else
        (g, s, t) ← EXTENDED_GCD(b, a mod b)
        return (g, t, s − (a ÷ b) · t)
    end if
end function
function FIND_LATTICE_BASIS(a)
    m ← μήκος(a)
    if όλα τα στοιχεία του a είναι μηδέν then
        return I_m
    end if
    U ← I_m
    v ← a
    for j = 0 έως m − 2 do
        for i = j + 1 έως m − 1 do
            if v[i] ≠ 0 then
                (g, s, t) ← EXTENDED_GCD(v[j], v[i])
                u ← −v[i] ÷ g,    w ← v[j] ÷ g
                col_j ← U[:, j],    col_i ← U[:, i]
                U[:, j] ← s · col_j + t · col_i
                U[:, i] ← u · col_j + w · col_i
                v[j] ← g,    v[i] ← 0
            end if
        end for
    end for
    if m = 1 then
        if a[0] = 0 then
            return [1]
        else
            return πίνακας 1 × 0
        end if
    else
        return U[:, 1 : m]
    end if
end function
```

Σύντομη εξήγηση: Ο αλγόριθμος εφαρμόζει διαδοχικά τον εκτεταμένο Ευκλείδειο αλγόριθμο σε κάθε ζεύγος στοιχείων του διανύσματος a, ώστε μέσω ακέραιων στοιχειωδών μετασχηματισμών να φέρει

το a στη μορφή $(g, 0, 0, \ldots, 0)$. Οι υπόλοιπες στήλες του προκύπτοντος ακέραιου αντιστρέψιμου πίνακα U αποτελούν ακέραια γραμμικά ανεξάρτητα διανύσματα που ικανοποιούν $a \cdot x = 0$, δηλαδή σχηματίζουν μια βάση του πλέγματος L.

3 'Ασκηση 3 Αλγόριθμος Gauss-Lagrange

3.1 Περιγραφή Αλγορίθμου

Ο αλγόριθμος Gauss-Lagrange είναι ένας αλγόριθμος για την εύρεση μιας ανηγμένης βάσης σε ένα δισδιάστατο πλέγμα. Δίνονται δύο γραμμικά ανεξάρτητα διανύσματα $\mathbf{b}_1$ και $\mathbf{b}_2$, ο αλγόριθμος παράγει μια νέα βάση $(\mathbf{b}'_1, \mathbf{b}'_2)$ που ικανοποιεί τη συνθήκη:

$$|\mathbf{b}'_1 \cdot \mathbf{b}'_2| \leq \frac{1}{2}\|\mathbf{b}'_1\|^2$$

και ελαχιστοποιεί τις νόρμες των διανυσμάτων.

Ο αλγόριθμος που υλοποίησα σε Python ακολουθεί τα παρακάτω βήματα:

1. Υπολόγισε $B_1 = \|\mathbf{b}_1\|^2$
2. Υπολόγισε $\mu = \frac{\mathbf{b}_1 \cdot \mathbf{b}_2}{B_1}$ και στρογγύλωσε στον πλησιέστερο ακέραιο.
3. Ανανέωσε $\mathbf{b}_2 = \mathbf{b}_2 - \lfloor \mu \rceil \mathbf{b}_1$
4. Υπολόγισε $B_2 = \|\mathbf{b}_2\|^2$
5. Εάν $B_2 < B_1$, ανταλλάσσεται $\mathbf{b}_1$ και $\mathbf{b}_2$ και επαναλαμβάνεται από το βήμα 1.

3.2 Υλοποίηση σε Python

Χρησιμοποίησα τη βιβλιοθήκη `numpy` για αριθμητικούς υπολογισμούς. Ο κώδικας περιλαμβάνει συναρτήσεις για:

- Τον αλγόριθμο Gauss-Lagrange (`gauss_lagrange`)
- Υπολογισμό της γωνίας μεταξύ δύο διανυσμάτων (`calculate_angle`)
- Δημιουργία 50 τυχαίων ζευγών διανυσμάτων στο $\mathbb{Z}^2$ και εκτέλεση του αλγορίθμου

3.3 Αποτελέσματα Πειράματος

Εκτέλεσα 50 πειράματα με τυχαία διανύσματα στο $\mathbb{Z}^2$ (με συντελεστές από -20 έως 20). Για κάθε πείραμα, εφάρμοσα τον αλγόριθμο Gauss-Lagrange και υπολόγισα τη γωνία μεταξύ των διανυσμάτων της ανηγμένης βάσης.

Τα αποτελέσματα έδειξαν μια αξιοσημείωτη τάση προς την ορθογωνιότητα:

- Μέση γωνία: $90.21°$ (πολύ κοντά στην ιδανική ορθογωνιότητα)
- Ελάχιστη γωνία: $63.50°$
- Μέγιστη γωνία: $114.20°$
- Τυπική απόκλιση: $12.58°$
- Ποσοστό γωνιών $> 80°$: 78.0%

Η κατανομή των γωνιών αποκαλύπτει ότι ο αλγόριθμος παράγει συστηματικά βάσεις που είναι πολύ κοντά στην ορθογωνιότητα. Το γεγονός ότι η μέση γωνία είναι ακριβώς $90.21°$ και ότι το 78% των περιπτώσεων έχει γωνίες άνω των $80°$ επιβεβαιώνει την αποτελεσματικότητα του αλγορίθμου στη βελτιστοποίηση της γωνίας μεταξύ των διανυσμάτων βάσης.

3.4 Απάντηση στο Θεωρητικό Ερώτημα

Σε περίπτωση που υπάρχει ορθογώνια βάση στο πλέγμα, ο αλγόριθμος Gauss-Lagrange **θα την βρει** με βεβαιότητα.

Οι λόγοι για αυτό είναι:

- Ο αλγόριθμος ελαχιστοποιεί τις νόρμες των διανυσμάτων βάσης. Αν υπάρχουν ορθογώνια διανύσματα με ελάχιστες δυνατές νόρμες για το πλέγμα, τότε αυτά θα προκύψουν ως τελικό αποτέλεσμα.
- Στις 2 διαστάσεις, ο αλγόριθμος είναι βέλτιστος και βρίσκει πάντα τα δύο μικρότερα γραμμικά ανεξάρτητα διανύσματα του πλέγματος.
- Η συνθήκη τερματισμού εγγυάται ότι τα διανύσματα είναι τοπικά βέλτιστα ως προς τη μείωση της νόρμας και τη μεγιστοποίηση της γωνίας.

Τα πειραματικά αποτελέσματα επιβεβαιώνουν αυτό το συμπέρασμα, καθώς η μέση γωνία των ανηγμένων βάσεων είναι πολύ κοντά στα $90°$, και σε πολλές περιπτώσεις ο αλγόριθμος βρίσκει ακριβώς ορθογώνιες βάσεις όταν αυτές υπάρχουν στο πλέγμα.

4 Άσκηση 4

1. Εκφώνηση

1. Να υλοποιηθεί ο αλγόριθμος LLL για ακέραια πλέγματα, χωρίς χρήση floating point στους ενδιάμεσους υπολογισμούς.
2. Βρείτε ένα πλέγμα όπου το πρώτο διάνυσμα που δίνει ο LLL έχει μεγαλύτερο μήκος από το δεύτερο διάνυσμα.
3. Εκτελέστε τον LLL και τον Gauss-Lagrange σε τυχαία πλέγματα μέγιστης βαθμίδας 2. Τι παρατηρείτε;
4. Κατασκευάστε τυχαία πλέγματα μέγιστης βαθμίδας 4 και υπολογίστε μια ανηγμένη βάση LLL. Το πρώτο διάνυσμα έχει μήκος ίσο με $\lambda_1(\mathcal{L})$;

Υλοποίηση και Θεωρητικό Υπόβαθρο

4.1 Αλγόριθμος LLL

Ο αλγόριθμος Lenstra–Lenstra–Lovász (LLL) είναι ένας πολυωνυμικός αλγόριθμος για την εύρεση μιας *κατά προσέγγιση* βέλτιστης βάσης ενός πλέγματος. Δεδομένης μιας βάσης $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$, ο LLL επιστρέφει μια βάση που ικανοποιεί τις συνθήκες Size Reduction και τη συνθήκη Lovász (με $\delta = 3/4$).

Αποφυγή Floating Point

Για να αποφευχθούν σφάλματα στρογγυλοποίησης, υλοποίησα τον αλγόριθμο χρησιμοποιώντας την κλάση `Fraction` της Python. Όλες οι πράξεις Gram-Schmidt και οι υπολογισμοί των συντελεστών μ_{ij} γίνονται με ρητή αριθμητική, εξασφαλίζοντας απόλυτη ακρίβεια.

Σύγκριση με τη συνάρτηση LLL του SageMath

Σύγκρινα την υλοποίησή μου (`lll_exact`) με την ενσωματωμένη συνάρτηση LLL του υπολογιστικού συστήματος SageMath (η οποία βασίζεται στη βιβλιοθήκη `fplll`). Οι κυριότερες διαφορές συνοψίζονται στον παρακάτω πίνακα:

Χαρακτηριστικό	**Η Υλοποίησή μου (Python)**	**SageMath (fplll)**
Αριθμητική	Ακρίβεια Ρητών (Fraction). Δεν χρησιμοποιούνται καθόλου δεκαδικοί.	Βελτιστοποιημένη αριθμητική (συχνά χρησιμοποιεί floating-point προσεγγίσεις για τους συντελεστές μ_{ij} για ταχύτητα, με διορθώσεις).
Ταχύτητα	Αργή. Οι πράξεις με αντικείμενα `Fraction` και οι υπολογισμοί ΜΚΔ (GCD) αυξάνουν την πολυπλοκότητα.	Εξαιρετικά γρήγορη. Υλοποιημένη σε C/C++ με βελτιστοποιημένες βιβλιοθήκες (GMP, mpfr).
Αποτέλεσμα	Επιστρέφει μια έγκυρη LLL βάση.	Επιστρέφει μια έγκυρη LLL βάση (μπορεί να διαφέρει από τη δική μου λόγω σειράς πράξεων, αλλά ισοδύναμη).

Table 1: Σύγκριση της Python υλοποίησης με το SageMath

Παρατήρηση: Εκτελώντας τα ίδια παραδείγματα και στα δύο συστήματα, επιβεβαίωσα ότι η υλοποίησή μου είναι ορθή, καθώς παρήγαγε βάσεις που ικανοποιούν τις συνθήκες LLL και έχουν διανύσματα με μήκη παραπλήσια (ή και ταυτόσημα) με αυτά που επέστρεψε το SageMath. Η κύρια διαφορά εντοπίστηκε στον χρόνο εκτέλεσης, όπου η Python υλοποίηση καθυστερούσε αισθητά σε πλέγματα μεγαλύτερων διαστάσεων.

4.2 Πειραματικά Αποτελέσματα

Ένα Πλέγμα όπου $\|\mathbf{b}_1\| > \|\mathbf{b}_2\|$

Αναζήτησα ένα πλέγμα όπου η συνθήκη Lovász ικανοποιείται, αλλά το Ευκλείδειο μήκος του πρώτου διανύσματος είναι μεγαλύτερο του δευτέρου.

Παράδειγμα που βρέθηκε (2D): Αρχική βάση: $B = \{[-9, 4], [-6, 13]\}$. Μετά την αναγωγή LLL προέκυψε η βάση:

$$\mathbf{b}_1 = [-9, 4], \quad \mathbf{b}_2 = [3, 9]$$

Τα μήκη των διανυσμάτων είναι:

- $\|\mathbf{b}_1\| = \sqrt{(-9)^2 + 4^2} = \sqrt{97} \approx 9.85$
- $\|\mathbf{b}_2\| = \sqrt{3^2 + 9^2} = \sqrt{90} \approx 9.49$

Παρατήρησα ότι $\|\mathbf{b}_1\| > \|\mathbf{b}_2\|$. Αυτό συμβαίνει διότι η συνθήκη Lovász ελέγχει τα μήκη των *προβολών* (Gram-Schmidt) και όχι τα συνολικά μήκη των διανυσμάτων.

Σύγκριση LLL και Gauss-Lagrange σε Πλέγματα Βαθμίδας 2

Εκτέλεσα πειράματα σε 500 τυχαία πλέγματα 2×2. Τα αποτελέσματα έδειξαν τα εξής:

1. Στο **79.0%** των περιπτώσεων, οι δύο αλγόριθμοι επέστρεψαν την ίδια βάση.
2. Στο **21.0%** των περιπτώσεων, ο αλγόριθμος Gauss-Lagrange επέστρεψε διάνυσμα μικρότερου μήκους από τον LLL.
3. Ο LLL δεν ήταν ποτέ καλύτερος από τον Gauss-Lagrange.

Συμπέρασμα: Στις 2 διαστάσεις, ο αλγόριθμος Gauss-Lagrange είναι βέλτιστος (επιλύει ακριβώς το SVP), ενώ ο LLL είναι προσεγγιστικός. Σε ένα σημαντικό ποσοστό των περιπτώσεων, παρατήρησα ότι ο LLL «κολλάει» σε ένα τοπικό ελάχιστο που ικανοποιεί τη συνθήκη Lovász, αλλά δεν είναι το απόλυτο ελάχιστο.

4.3 Ελέγχοντας αν $\|\mathbf{b}_1\| = \lambda_1(\mathcal{L})$ σε Βαθμίδα 4

Για 15 τυχαία πλέγματα 4×4, σύγκρινα το αποτέλεσμα του LLL με την πραγματική τιμή του πρώτου ελαχίστου λ_1, την οποία υπολόγισα μέσο εξαντλητικής αναζήτησης (Enumeration).

Αποτέλεσμα	**Πλήθος**	**Ποσοστό**
$\|\mathbf{b}_1\|_{LLL} = \lambda_1$ (Επιτυχία)	3	20.0%
$\|\mathbf{b}_1\|_{LLL} > \lambda_1$ (Αποτυχία)	12	80.0%

Table 2: Αξιοπιστία του LLL στην εύρεση του SVP σε 4D

Παράδειγμα μεγάλης απόκλισης: Σε ένα από τα πειράματα βρέθηκε:

- LLL αποτέλεσμα: $\|\mathbf{b}_1\| = 8.660$
- Πραγματικό ελάχιστο: $\lambda_1 = 2.449$

Η διαφορά είναι σημαντική.

Τελικά Συμπεράσματα

- Η ακριβής υλοποίηση με κλάσματα αποφεύγει σφάλματα αριθμητικής αλλά δεν διορθώνει την προσεγγιστική φύση του αλγορίθμου.
- Ο LLL δεν ταξινομεί τα διανύσματα αυστηρά κατά μήκος.
- Ενώ στις 2 διαστάσεις ο LLL είναι κοντά στο βέλτιστο (79% επιτυχία), στις 4 διαστάσεις η απόδοσή του στην εύρεση του συντομότερου διανύσματος πέφτει δραματικά (μόλις 20% επιτυχία).
- Αυτό επιβεβαιώνει ότι ο LLL λύνει το SVP κατά προσέγγιση με παράγοντα που αυξάνεται εκθετικά με τη διάσταση n.

5 Άσκηση 5

5.1 Απόδειξη του Knapsack Lemma

Θεωρώ το πλέγμα $L(B_{N,H})$ που παράγεται από τις γραμμές του πίνακα $B_{N,H} \in \mathbb{Z}^{(n+1)\times(n+3)}$, όπου $N, H \in \mathbb{N}$ με $N > \sqrt{n}$. Θα δείξω ότι υπάρχει αμφίδρομη αντιστοιχία μεταξύ λύσεων του προβλήματος knapsack και διανυσμάτων του πλέγματος με τις δοσμένες ιδιότητες.

Από λύση knapsack σε διάνυσμα πλέγματος. Έστω ότι υπάρχει λύση $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ του προβλήματος knapsack

$$a_1 x_1 + \cdots + a_n x_n = s,$$

με βάρος Hamming $H = \sum_{i=1}^{n} x_i$.

Ορίζω το διάνυσμα $\mathbf{b} \in \mathbb{Z}^{n+3}$ ως τον ακόλουθο γραμμικό συνδυασμό των γραμμών του $B_{N,H}$:

$$\mathbf{b} = \sum_{i=1}^{n} x_i \cdot \mathbf{b}^{(i)} \; - \; \mathbf{b}^{(n+1)},$$

όπου $\mathbf{b}^{(i)}$ είναι η i-οστή γραμμή του $B_{N,H}$ (για $i = 1, \ldots, n$) και $\mathbf{b}^{(n+1)}$ η τελευταία γραμμή.

Υπολογίζοντας τις συνιστώσες του $\mathbf{b}$:

- Για $i = 1, \ldots, n$: $b_i = x_i \cdot 2 - 1 = 2x_i - 1 \in \{\pm 1\}$, άρα $|b_i| = 1$.
- Για τη συνιστώσα $n+1$: $b_{n+1} = \sum_{i=1}^{n} x_i(Na_i) - Ns = N\big(\sum_{i=1}^{n} a_i x_i - s\big) = 0$.
- Για τη συνιστώσα $n+2$: $b_{n+2} = \sum_{i=1}^{n} x_i \cdot 0 - 1 = -1$, άρα $|b_{n+2}| = 1$.
- Για τη συνιστώσα $n+3$: $b_{n+3} = \sum_{i=1}^{n} x_i N - HN = N\big(\sum_{i=1}^{n} x_i - H\big) = 0$.

Επομένως, το $\mathbf{b}$ ανήκει στο πλέγμα $L(B_{N,H})$ και ικανοποιεί τις συνθήκες

$$|b_1| = \cdots = |b_n| = 1, \quad b_{n+1} = b_{n+3} = 0, \quad |b_{n+2}| = 1.$$

Επιπλέον, για κάθε $i = 1, \ldots, n$,

$$\frac{|\, b_i - b_{n+2} \,|}{2} = \frac{|\, (2x_i - 1) - (-1) \,|}{2} = \frac{|\, 2x_i \,|}{2} = x_i,$$

οπότε η αρχική λύση $\mathbf{x}$ ανακτάται από το $\mathbf{b}$.

Από διάνυσμα πλέγματος σε λύση knapsack. Αντίστροφα, έστω ότι υπάρχει διάνυσμα $\mathbf{b} = (b_1, \ldots, b_{n+3}) \in L(B_{N,H})$ με

$$|b_1| = \cdots = |b_n| = 1, \quad b_{n+1} = b_{n+3} = 0, \quad |b_{n+2}| = 1.$$

Ορίζω

$$x_i = \frac{|\, b_i - b_{n+2} \,|}{2}, \qquad i = 1, \ldots, n.$$

Επειδή $b_i, b_{n+2} \in \{\pm 1\}$, η διαφορά $b_i - b_{n+2}$ παίρνει τιμές $-2, 0, 2$, άρα $|b_i - b_{n+2}| \in \{0, 2\}$ και συνεπώς $x_i \in \{0, 1\}$.

Εφόσον το $\mathbf{b}$ είναι γραμμικός συνδυασμός των γραμμών του $B_{N,H}$, από τη μορφή του προκύπτει ότι:

- Η συνθήκη $b_{n+1} = 0$ δίνει $\sum_{i=1}^{n} a_i x_i = s$.
- Η συνθήκη $b_{n+3} = 0$ δίνει $\sum_{i=1}^{n} x_i = H$.

Άρα, το $\mathbf{x} = (x_1, \ldots, x_n)$ είναι λύση του προβλήματος knapsack με βάρος Hamming H.

Συμπέρασμα. Απέδειξα ότι υπάρχει αμφίδρομη αντιστοιχία μεταξύ λύσεων του knapsack και διανυσμάτων του πλέγματος $L(B_{N,H})$ με τη δομή που περιγράφεται στο λήμμα, ολοκληρώνοντας την απόδειξη.

Επίλυση Knapsack Προβλημάτων με LLL

Εισαγωγή

Η εργασία αφορά την επίλυση προβλημάτων knapsack με συγκεκριμένες παραμέτρους:

- Διάσταση: $n = 30$
- Hamming weight: $H = 15$
- Πυκνότητα: $d \approx 1$

Μέθοδος Επίλυσης

Χρησιμοποίησα τον αλγόριθμο LLL (Lenstra-Lenstra-Lovász) σε πλέγμα που ορίζεται από τον πίνακα:

$$B_{N,H} = \begin{bmatrix} 2 & 0 & \cdots & 0 & Na_1 & 0 & N \\ 0 & 2 & \cdots & 0 & Na_2 & 0 & N \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 2 & Na_n & 0 & N \\ 1 & 1 & \cdots & 1 & Ns & 1 & HN \end{bmatrix} \in \mathbb{Z}^{(n+1)\times(n+3)}$$

όπου $N > \sqrt{n}$ (εδώ $N = 6 > \sqrt{30} \approx 5.48$).

Υλοποίηση και Αλγόριθμος

1. Δημιουργία τυχαίου προβλήματος με τη συνάρτηση `find()` (σύμφωνα με τα notes)
2. Κατασκευή πίνακα $B_{N,H}$ για $N = 6$
3. Εφαρμογή LLL στον πίνακα
4. Εξαγωγή λύσης από τα διανύσματα μικρού μήκους
5. Επαλήθευση ότι η λύση ικανοποιεί $\sum a_i x_i = a_0$ και $\sum x_i = H$

Αποτελέσματα και Στατιστική Ανάλυση

Στις 5 εκτελέσεις του προγράμματος:

Εκτέλεση	Πυκνότητα	Προσπάθειες	Τύπος Λύσης	Ταύτιση
1	1.000	1	Μορφή 2	ΝΑΙ
2	1.001	1	Μορφή 2	ΟΧΙ
3	1.000	1	Μορφή 2	ΝΑΙ
4	1.003	2	Μορφή 2	ΝΑΙ
5	1.001	1	Μορφή 1	ΝΑΙ

Table 3: Αποτελέσματα από 5 εκτελέσεις του προγράμματος

Στατιστικά:

- Ποσοστό επιτυχίας: 100% (5/5 εκτελέσεις βρήκαν λύση)
- Μέσος αριθμός προσπαθειών ανά εκτέλεση: 1.2
- Τύπος λύσης που βρέθηκε: 80% Μορφή 2, 20% Μορφή 1
- Ταύτιση με πραγματική λύση: 80% των περιπτώσεων

Επιτυχής Επίλυση (Παράδειγμα 1)

- **Παράμετροι**: $n = 30$, $H = 15$, πυκνότητα $= 1.000$
- **Στόχος**: $a_0 = 9969058017$
- **Πραγματική λύση**: $[0, 0, 1, 1, 1, 0, 1, 0, 1, 0, \dots]^T$
- **Βρεθείσα λύση**: $[0, 0, 1, 1, 1, 0, 1, 0, 1, 0, \dots]^T$
- **Επαλήθευση**:

$$\sum_{i=1}^{30} a_i x_i = 9969058017 = a_0, \quad \sum_{i=1}^{30} x_i = 15$$

Σχολιασμός Αποτελεσμάτων

- **Αποδοτικότητα**: Στις 4 από 5 περιπτώσεις, η λύση βρέθηκε στην πρώτη προσπάθεια
- **Ορθότητα**: Σε όλες τις περιπτώσεις, η λύση ικανοποιεί τον στόχο a_0
- **Πολλαπλές μορφές**: Ο αλγόριθμος αναγνωρίζει δύο μορφές λύσεων:
  - Μορφή 1: $(2x_i - 1, 0, -1, 0)$
  - Μορφή 2: $(1 - 2x_i, 0, 1, 0)$ (συμπλήρωμα)
- **Σταθερότητα**: Ο αλγόριθμος είναι σταθερός - αν δεν βρει λύση σε ένα πρόβλημα, δημιουργεί νέο και συνεχίζει
- **Πυκνότητα**: Η πραγματική πυκνότητα παραμένει κοντά στο 1 (1.000-1.003)

Συμπεράσματα

- Ο αλγόριθμος LLL σε συνδυασμό με το πλέγμα $B_{N,H}$ επιλύει αποτελεσματικά προβλήματα knapsack με τις δοσμένες παραμέτρους
- Η μέθοδος απαιτεί μόνο $N > \sqrt{n}$ (εδώ $N = 6$) για να λειτουργεί αξιόπιστα
- Η επίλυση είναι ταχεία (1-2 LLL αναγωγές ανά πρόβλημα)
- Ο αλγόριθμος έχει 100% ποσοστό επιτυχίας, απαιτώντας κατά μέσο όρο 1.2 προσπάθειες ανά εκτέλεση
- Ο κώδικας υλοποιεί πλήρως τις απαιτήσεις της άσκησης και επαληθεύει τη θεωρητική προσέγγιση

6 Άσκηση 6

6.1 Περιγραφή Προβλήματος

Δίνεται ένα RSA modulus N μεγέθους 251 bits:

$$N = 2122840968903324034467344329510307845524745715398875789936591447337206598081$$

και ciphertext c που προέκυψε από την κρυπτογράφηση μηνύματος m με $m < 2^{36}$:

$c = (2^{500}+m)^3 \mod N = 17929634596906001924003559884681302712481713818274627498706514089439934808 16$

Στόχος είναι η ανάκτηση του μηνύματος m χρησιμοποιώντας την μέθοδο Coppersmith.

6.2 Μέθοδοι Επίλυσης

Εφαρμόστηκαν οι ακόλουθες μέθοδοι:

6.2.1 Μέθοδος Coppersmith

Χρησιμοποιήθηκε η μέθοδος Coppersmith για εύρεση μικρών ριζών πολυωνύμων modulo N. Το πολυώνυμο ορίστηκε ως:

$$f(x) = (2^{500} + x)^3 - c \mod N$$

ή ισοδύναμα:

$$f(x) = x^3 + 3 \cdot 2^{500}x^2 + 3 \cdot 2^{1000}x + (2^{1500} - c) \mod N$$

Για την εφαρμογή της μεθόδου χρησιμοποιήθηκαν:

- Η συνάρτηση `small_roots()` της SageMath με διάφορες παραμέτρους β (0.1 έως 0.6) και ϵ (0.01 έως 0.1)
- Η προσαρμοσμένη συνάρτηση `univ_cop()` με διάφορες τιμές ϵ
- Δοκιμή με διαφορετικά άνω όρια X (από 2^{36} έως $5 \cdot 2^{36}$)

6.2.2 Μαθηματική Ανάλυση

Αναλύθηκε το θεωρητικό όριο της μεθόδου Coppersmith:

$$|m| < N^{1/d-\epsilon} \quad \text{με } d = 3$$

Για $N \approx 2^{251}$, έχουμε $N^{1/3} \approx 2^{84} \approx 1.29 \times 10^{25}$, ενώ το πραγματικό όριο είναι $2^{36} \approx 6.87 \times 10^{10}$. Η αναλογία είναι μόνο 0.00053% του θεωρητικού ορίου, γεγονός που καθιστά το πρόβλημα ιδανικό για επίθεση Coppersmith.

6.2.3 Βελτιστοποιημένη Εξαντλητική Αναζήτηση

Προσεγγίστηκε εξαντλητική αναζήτηση με μαθηματικές βελτιστοποιήσεις:

- Υπολογισμός $2^{500} \mod N$ για μείωση πράξεων
- Έλεγχος ισοτιμιών (άρτιος/περιττός)
- Έλεγχος modulo μικρών πρώτων (3, 5, 7, 11, 13, 17, 19)

6.3 Αποτελέσματα

Καμία από τις μεθόδους δεν κατάφερε να βρει το μήνυμα m. Συγκεκριμένα:

- Η μέθοδος Coppersmith με όλες τις παραμέτρους δεν επέστρεψε καμία ρίζα
- Ο έλεγχος modulo 7 έδειξε ότι δεν υπάρχει έγκυρο υπόλοιπο, υποδεικνύοντας πιθανό λάθος στη διατύπωση
- Η εξαντλητική αναζήτηση διακόπηκε λόγω του μεγάλου χρονικού διαστήματος που απαιτείται ($\sim$19 ώρες)

6.4 Πιθανοί Λόγοι Αποτυχίας

1. **Λάθος στη διατύπωση:** Ίσως υπάρχει λάθος στον αριθμό N, στο c, ή στην εκφώνηση ($m < 2^{36}$)
2. **Μεγαλύτερο m:** Το m μπορεί να είναι μεγαλύτερο από 2^{36}, καταρρίπτοντας τις υποθέσεις της μεθόδου Coppersmith
3. **Πρόβλημα με τις παραμέτρους:** Η μέθοδος Coppersmith μπορεί να απαιτεί διαφορετική παραμετροποίηση (μεγαλύτερο πλέγμα, διαφορετικό ϵ)
4. **Αδύνατη εξίσωση:** Η εξίσωση $(2^{500} + m)^3 \equiv c \pmod{N}$ μπορεί να μην έχει λύση για $m < 2^{36}$
5. **Υπολογιστικά ζητήματα:** Η SageMath μπορεί να μην χειρίζεται σωστά πολυώνυμα με τόσο μεγάλους συντελεστές

6.5 Συμπεράσματα

Παρόλο που το πρόβλημα φαινόταν ιδανικό για επίθεση Coppersmith λόγω της μικρής ρίζας σε σχέση με το μέτρο, δεν ήταν δυνατή η εύρεση λύσης. Αυτό υποδηλώνει πιθανά ζητήματα με τα δεδομένα του προβλήματος ή την ανάγκια για πιο εξελιγμένες τεχνικές. Η αποτυχία του ελέγχου modulo 7 είναι ιδιαίτερα ενδεικτική πιθανού λάθους στη διατύπωση.

7 Άσκηση 7 (14.4)

Υπολογίστε τον όγκο του πλέγματος που έχει βάση $\mathbf{b}_1 = (1, 1, 1)$ και $\mathbf{b}_2 = (0, -1, 2)$.

Λύση

Ο όγκος του πλέγματος (που στην προκειμένη περίπτωση είναι το εμβαδόν του θεμελιώδους παραλληλογράμμου στον $\mathbb{R}^3$) δίνεται από το μέτρο του εξωτερικού γινομένου των διανυσμάτων της βάσης:

$$\text{vol}(\mathcal{L}) = \|\mathbf{b}_1 \times \mathbf{b}_2\| \tag{1}$$

Υπολογίζουμε το εξωτερικό γινόμενο:

$$\begin{aligned}
\mathbf{b}_1 \times \mathbf{b}_2 &= \det \begin{pmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 1 & 1 & 1 \\ 0 & -1 & 2 \end{pmatrix} \\
&= \mathbf{i}(2 - (-1)) - \mathbf{j}(2 - 0) + \mathbf{k}(-1 - 0) \\
&= 3\mathbf{i} - 2\mathbf{j} - \mathbf{k} \\
&= (3, -2, -1)
\end{aligned}$$

Συνεπώς, ο ζητούμενος όγκος είναι:

$$\text{vol}(\mathcal{L}) = \sqrt{3^2 + (-2)^2 + (-1)^2} = \sqrt{9 + 4 + 1} = \sqrt{14}$$

8 Άσκηση 8 (14.8)

Ερώτημα (i): Απόδειξη Ισοδυναμίας Πλεγμάτων

Μαθηματική Θεμελίωση

Δίνονται οι πίνακες βάσης A και B. Για να αποδείξουμε ότι τα πλέγματα που παράγονται από τις γραμμές τους, $L(A)$ και $L(B)$, είναι ταυτοτικά, πρέπει να ισχύουν οι εξής δύο συνθήκες:

1. Κάθε γραμμή του πίνακα B πρέπει να ανήκει στο πλέγμα $L(A)$. Αυτό σημαίνει ότι κάθε γραμμή του B πρέπει να μπορεί να γραφτεί ως **ακέραιος** γραμμικός συνδυασμός των γραμμών του A.
2. Κάθε γραμμή του πίνακα A πρέπει να ανήκει στο πλέγμα $L(B)$ (αντίστροφη σχέση).

Μαθηματικά, αναζητούμε αν υπάρχουν πίνακες ακεραίων U και V τέτοιοι ώστε:

$$B = U \cdot A \quad \text{και} \quad A = V \cdot B$$

Ανάλυση Κώδικα και Αποτελέσματα

Για την υπολογιστική επαλήθευση, αναπτύχθηκε κώδικας σε Python με χρήση της βιβλιοθήκης `SymPy` για συμβολικούς υπολογισμούς, ώστε να διασφαλιστεί η ακρίβεια των πράξεων και να αποφευχθούν σφάλματα κινητής υποδιαστολής.

Η διαδικασία που ακολουθήθηκε είναι η εξής:

- **Έλεγχος Τάξης:** Επιβεβαιώθηκε ότι $\text{rank}(A) = \text{rank}(B) = 4$, συνεπώς και οι δύο βάσεις είναι πλήρους τάξης στον $\mathbb{R}^4$.
- **Έλεγχος $L(B) \subseteq L(A)$:** Για κάθε γραμμή b_i του πίνακα B, λύθηκε το γραμμικό σύστημα $x \cdot A = b_i$. Ο αλγόριθμος επιβεβαίωσε ότι οι λύσεις είναι μοναδικές και αποτελούνται αποκλειστικά από ακέραιους αριθμούς.
- **Έλεγχος $L(A) \subseteq L(B)$:** Αντίστοιχα, για κάθε γραμμή a_i του A, λύθηκε το σύστημα $y \cdot B = a_i$, το οποίο επίσης έδωσε ακέραιες λύσεις.

Υπολογιστικά Ευρήματα

Ο κώδικας παρήγαγε τους εξής συντελεστές μετάβασης, αποδεικνύοντας την ισοδυναμία:

1. Μετάβαση από A σε B: Οι γραμμές του B παράγονται από το A με τους εξής συντελεστές:

$$\begin{aligned} b_1 &= 2a_1 + 1a_2 + 1a_3 + 1a_4 \\ b_2 &= 1a_1 + 1a_2 + 1a_3 + 1a_4 \\ b_3 &= 1a_1 + 1a_2 + 2a_3 + 1a_4 \\ b_4 &= 1a_1 + 1a_2 + 1a_3 + 2a_4 \end{aligned}$$

2. Μετάβαση από B σε A: Οι γραμμές του A παράγονται από το B με τους εξής συντελεστές:

$$\begin{aligned} a_1 &= 1b_1 - 1b_2 \\ a_2 &= -1b_1 + 4b_2 - 1b_3 - 1b_4 \\ a_3 &= -1b_2 + 1b_3 \\ a_4 &= -1b_2 + 1b_4 \end{aligned}$$

Συμπέρασμα: Εφόσον οι πίνακες μετάβασης είναι ακέραιοι και προς τις δύο κατευθύνσεις, αποδεικνύεται ότι $L(A) = L(B)$.

Ερώτημα (ii): Έλεγχος Συμμετοχής Διανύσματος στο Πλέγμα

Μαθηματική Διατύπωση

Ζητείται να ελεγχθεί αν το διάνυσμα $v = (43, 69, 95, 110)$ ανήκει στο πλέγμα $L(A)$. Εξ ορισμού, το πλέγμα $L(A)$ αποτελείται από όλους τους ακέραιους γραμμικούς συνδυασμούς των γραμμών του πίνακα βάσης A. Επομένως, το $v \in L(A)$ αν και μόνο αν υπάρχουν ακέραιοι συντελεστές $x_1, x_2, x_3, x_4 \in \mathbb{Z}$ τέτοιοι ώστε:

$$v = x_1a_1 + x_2a_2 + x_3a_3 + x_4a_4$$

όπου a_i είναι η i-οστή γραμμή του πίνακα A.

Αυτό ανάγεται στην επίλυση του γραμμικού συστήματος:

$$(x_1, x_2, x_3, x_4) \cdot \begin{pmatrix} 1 & 2 & 3 & 4 \\ -1 & 1 & 2 & 3 \\ 4 & 3 & 2 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} = (43, 69, 95, 110)$$

Ανάλυση Κώδικα και Αποτελέσματα

Χρησιμοποιώντας τη βιβλιοθήκη `SymPy`, ορίσαμε τις συμβολικές μεταβλητές x_i και λύσαμε το σύστημα των 4 γραμμικών εξισώσεων (μία για κάθε συνιστώσα του διανύσματος v).

Η επίλυση του συστήματος έδωσε τη μοναδική λύση:

$$x_1 = 14, \quad x_2 = 11, \quad x_3 = 10, \quad x_4 = 11$$

Συμπέρασμα

Παρατηρούμε ότι:

- Υπάρχει μοναδική λύση στο σύστημα.
- Όλοι οι συντελεστές x_i είναι **ακέραιοι** αριθμοί.

Συνεπώς, το διάνυσμα v **ανήκει** στο πλέγμα $L(A)$, καθώς μπορεί να εκφραστεί ως:

$$v = 14a_1 + 11a_2 + 10a_3 + 11a_4$$

Ερώτημα (iii): Υπολογισμός Όγκου Πλέγματος

Θεωρητικό Υπόβαθρο

Ο όγκος (ή suovolume) του θεμελιώδους χωρίου ενός πλέγματος $L \subset \mathbb{R}^n$, το οποίο παράγεται από έναν τετραγωνικό πίνακα βάσης M διάστασης $n \times n$, δίνεται από την απόλυτη τιμή της ορίζουσας του πίνακα:

$$\mathrm{vol}(L) = |\det(M)|$$

Ο όγκος αυτός είναι αναλλοίωτος, δηλαδή δεν εξαρτάται από την επιλογή της βάσης του πλέγματος.

Υπολογισμός

Για τον υπολογισμό χρησιμοποιήσαμε τον πίνακα A. Η ορίζουσα του A είναι:

$$\det(A) = \det \begin{pmatrix} 1 & 2 & 3 & 4 \\ -1 & 1 & 2 & 3 \\ 4 & 3 & 2 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

Μέσω του κώδικα υπολογίστηκε ότι:

$$\det(A) = 5$$

Συνεπώς, ο ζητούμενος όγκος του πλέγματος είναι:

$$\mathrm{vol}(L) = |5| = 5$$

Επαλήθευση Συνοχής

Ως επιβεβαίωση των αποτελεσμάτων του ερωτήματος (i), υπολογίστηκε και η ορίζουσα του δεύτερου πίνακα βάσης B. Βρέθηκε ότι $\det(B) = 5$. Το γεγονός ότι $|\det(A)| = |\det(B)| = 5$ επιβεβαιώνει ότι οι δύο πίνακες παράγουν πλέγματα με τον ίδιο όγκο, κάτι που είναι απαραίτητη προϋπόθεση (αν και όχι ικανή από μόνη της) για την ισότητα των πλεγμάτων.

Ερώτημα (iv): Απαρίθμηση Σημείων Πλέγματος σε Γεωμετρικά Σχήματα

Μεθοδολογία: Αναγωγή Βάσης (LLL Reduction)

Ζητείται ο υπολογισμός του πλήθους των σημείων του πλέγματος $x \in L(A)$ που περιέχονται:

1. Σε σφαίρα στον $\mathbb{R}^4$ με κέντρο την αρχή των αξόνων και ακτίνα $R = 10$.
2. Σε υπερκύβο με κέντρο την αρχή των αξόνων και πλευρά $a = 10$ (δηλαδή $|x_i| \leq 5$).

Η αρχική βάση A αποτελείται από διανύσματα που δεν είναι ορθογώνια μεταξύ τους (skewed basis). Αυτό καθιστά την εξαντλητική αναζήτηση (brute-force) εξαιρετικά αργή και αναποτελεσματική, καθώς απαιτείται έλεγχος πολύ μεγάλου εύρους συντελεστών για να εντοπιστούν σημεία κοντά στο κέντρο.

Για την επίλυση του προβλήματος, εφαρμόστηκε ο αλγόριθμος **LLL (Lenstra–Lenstra–Lovász)**. Ο αλγόριθμος αυτός παράγει μια νέα βάση A_{LLL} για το *ίδιο* πλέγμα, η οποία αποτελείται από διανύσματα μικρότερου μήκους και σχεδόν ορθογώνια μεταξύ τους.

Η νέα βελτιστοποιημένη βάση που προέκυψε είναι:

$$A_{LLL} = \begin{pmatrix} 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & -1 & 1 \\ -1 & -1 & 1 & -1 \end{pmatrix}$$

Με τη χρήση της A_{LLL}, αρκεί ένα μικρό εύρος αναζήτησης ακεραίων συντελεστών ($z_i \in [-6, 6]$) για να καλυφθεί πλήρως ο ζητούμενος χώρος.

Αποτελέσματα Καταμέτρησης

Ο κώδικας εκτέλεσε αναζήτηση σε $13^4 = 28,561$ συνδυασμούς συντελεστών και εντόπισε τα εξής πλήθη σημείων:

Γεωμετρικό Σχήμα	**Θεωρητική Εκτίμηση** ($N \approx \frac{V}{\det(L)}$)	**Εύρημα Κώδικα**
Σφαίρα ($R = 10$)	≈ 9870	**8719**
Κύβος (Πλευρά 10)	≈ 2000	**2879**

Table 4: Σύγκριση θεωρητικής πρόβλεψης και υπολογιστικού αποτελέσματος.

Ανάλυση Αποτελεσμάτων

Η θεωρητική εκτίμηση βασίζεται στην Γκαουσιανή ευρετική (Gaussian Heuristic): $N \approx \frac{\text{Vol}(S)}{\det(L)}$.

- **Σφαίρα:** Το αποτέλεσμα (8719) πλησιάζει τη θεωρητική πρόβλεψη (9870) με απόκλιση περίπου 11%. Η απόκλιση αυτή είναι αναμενόμενη στις 4 διαστάσεις λόγω του «φαινομένου των συνόρων» (boundary effect), καθώς μεγάλο μέρος του όγκου της υπερσφαίρας βρίσκεται κοντά στην επιφάνειά της, όπου η διακριτή φύση του πλέγματος δημιουργεί σφάλματα στην εκτίμηση μέσω του συνεχούς όγκου.
- **Κύβος:** Παρατηρείται ότι βρέθηκαν περισσότερα σημεία (2879) από την εκτίμηση όγκου (2000). Αυτό οφείλεται στον ειδικό προσανατολισμό του πλέγματος ως προς τους άξονες του κύβου. Επειδή ο κύβος δεν είναι αναλλοίωτος σε στροφές (σε αντίθεση με τη σφαίρα), η ευθυγράμμιση των πυκνών περιοχών του πλέγματος με τις ακμές του κύβου μπορεί να οδηγήσει σε αυξημένο αριθμό σημείων εντός αυτού.

Συμπέρασμα: Η χρήση της LLL βάσης επέτρεψε την ακριβή καταμέτρηση των σημείων, επιβεβαιώνοντας ότι το πλέγμα έχει την αναμενόμενη πυκνότητα (περίπου 1 σημείο ανά 5 μονάδες όγκου).

9 Άσκηση 9 (14.14)

Για να υπολογίσουμε τον όγκο του πλέγματος A_n, χρειαζόμαστε μια βάση του A_n και τον πίνακα Gram αυτής της βάσης.

1. Εύρεση Βάσης

Το πλέγμα A_n ορίζεται ως $A_n = \{(x_0, \ldots, x_n) \in \mathbb{Z}^{n+1} : \sum x_i = 0\}$. Επομένως, έχει διάσταση n. Θεωρούμε τα εξής n διανύσματα $b_1, \ldots, b_n \in \mathbb{Z}^{n+1}$:

$$\begin{aligned} b_1 &= (1, -1, 0, 0, \ldots, 0) \\ b_2 &= (0, 1, -1, 0, \ldots, 0) \\ &\vdots \\ b_n &= (0, 0, \ldots, 0, 1, -1) \end{aligned}$$

Τα διανύσματα αυτά ανήκουν στο A_n και είναι γραμμικά ανεξάρτητα, άρα αποτελούν βάση του πλέγματος.

2. Υπολογισμός Πίνακα Gram

Ο πίνακας Gram G_n είναι πίνακας $n \times n$ με στοιχεία τα εσωτερικά γινόμενα $G_{ij} = b_i \cdot b_j$.

- Για $i = j$: $b_i \cdot b_i = 1^2 + (-1)^2 = 2$.
- Για $|i - j| = 1$: $b_i \cdot b_{i+1} = -1$.
- Για $|i - j| > 1$: $b_i \cdot b_j = 0$.

Ο πίνακας έχει τη μορφή:

$$G_n = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 2 \end{pmatrix}$$

3. Υπολογισμός Ορίζουσας

Έστω $D_n = \det(G_n)$. Αναπτύσσοντας ως προς την πρώτη γραμμή, παίρνουμε την αναδρομική σχέση:

$$D_n = 2D_{n-1} - D_{n-2}$$

Ελέγχουμε τις αρχικές τιμές:

- $D_1 = 2$
- $D_2 = \det \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} = 4 - 1 = 3$

Παρατηρούμε ότι ακολουθείται ο τύπος $D_n = n + 1$.

4. Τελικό Αποτέλεσμα

Ο όγκος του πλέγματος δίνεται από τον τύπο $\text{vol}(A_n) = \sqrt{\det(G_n)}$. Συνεπώς:

$$\text{vol}(A_n) = \sqrt{n+1}$$

10 Άσκηση 10

Υλοποίηση και Ανάλυση του Αλγορίθμου Nearest Plane (Babai)

Λειτουργία του Κώδικα

Η υλοποίηση του αλγορίθμου *Babai's Nearest Plane* βασίζεται στην αναδρομική εύρεση του κοντινότερου διανύσματος, χρησιμοποιώντας την ορθογωνιοποίηση Gram-Schmidt (GSO). Ο αλγόριθμος λειτουργεί ως εξής:

1. **Ορθογωνιοποίηση (Gram-Schmidt):** Αρχικά, η συνάρτηση `gram_schmidt` δέχεται τη βάση του πλέγματος $B = \{b_1, \dots, b_n\}$ και παράγει την ορθογώνια βάση $B^* = \{b_1^*, \dots, b_n^*\}$. Αυτό είναι κρίσιμο διότι οι προβολές πρέπει να γίνονται σε ορθογώνιους υποχώρους για να διαχωριστεί η συνεισφορά κάθε διανύσματος.

2. **Αναδρομική Προσέγγιση:** Η κύρια συνάρτηση ξεκινά θέτοντας μια μεταβλητή (στον κώδικα `b_vec`) ίση με το διάνυσμα στόχο t. Η επανάληψη εκτελείται από το τελευταίο διάνυσμα της βάσης (b_n) προς το πρώτο (b_1):

   - Υπολογίζεται ο συντελεστής προβολής $c_j \in \mathbb{R}$ του τρέχοντος διανύσματος στόχου στο ορθογώνιο διάνυσμα b_j^*:

$$c_j = \frac{\langle b_{\text{vec}}, b_j^* \rangle}{\|b_j^*\|^2}$$

- Ο συντελεστής c_j στρογγυλοποιείται στον πλησιέστερο ακέραιο:

$$k = \lfloor c_j \rceil$$

- Το διάνυσμα $k \cdot b_j$ αφαιρείται από το τρέχον διάνυσμα στόχο (`b_vec`), μειώνοντας την απόσταση στον συγκεκριμένο υπόχωρο.

3. **Εξαγωγή Αποτελέσματος:** Στο τέλος της διαδικασίας, η μεταβλητή `b_vec` περιέχει το διάνυσμα σφάλματος (την απόσταση του στόχου από το πλέγμα). Το κοντινότερο διάνυσμα του πλέγματος $v \in \mathcal{L}$ υπολογίζεται ως:

$$v = t - b_{\text{vec}}$$

Ανάλυση Αποτελεσμάτων Παραδείγματος

Εφαρμόσαμε τον αλγόριθμο με τα εξής δεδομένα εισόδου:

- **Βάση Πλέγματος:** $b_1 = (1, 2)$, $b_2 = (3, 5)$.
- **Διάνυσμα Στόχος:** $t = (12.1, 19.8)$.

Αποτέλεσμα: Ο αλγόριθμος επέστρεψε το διάνυσμα:

$$x = (12, 20)$$

Επαλήθευση: Παρατηρούμε ότι το αποτέλεσμα x είναι γραμμικός συνδυασμός των διανυσμάτων της βάσης με ακέραιους συντελεστές, συνεπώς ανήκει στο πλέγμα:

$$x = 0 \cdot b_1 + 4 \cdot b_2 = 0 \cdot (1, 2) + 4 \cdot (3, 5) = (0, 0) + (12, 20) = (12, 20)$$

Αυτό σημαίνει ότι ο αλγόριθμος επέλεξε συντελεστές $c_1 = 0$ και $c_2 = 4$.
Σφάλμα Προσέγγισης: Η Ευκλείδεια απόσταση μεταξύ του στόχου t και του ευρεθέντος διανύσματος x είναι:

$$\|t - x\| = \sqrt{(12.1 - 12)^2 + (19.8 - 20)^2} = \sqrt{0.1^2 + (-0.2)^2} = \sqrt{0.01 + 0.04} = \sqrt{0.05} \approx 0.2236$$

Η απόσταση αυτή είναι εξαιρετικά μικρή, υποδεικνύοντας ότι ο αλγόριθμος εντόπισε επιτυχώς το κοντινότερο σημείο του πλέγματος (CVP) για το συγκεκριμένο παράδειγμα.

11 Άσκηση 11

Αλγόριθμος Απαρίθμησης KFP: Υλοποίηση και Αποτελέσματα

11.1 Επισκόπηση του Αλγορίθμου

Ο αλγόριθμος Kannan-Fincke-Pohst (KFP) είναι ένας αλγόριθμος απαρίθμησης για πλέγματα που βρίσκει όλα τα διανύσματα ενός πλέγματος $L(B)$ μέσα σε μια σφαίρα ακτίνας R. Βασίζεται στην ιδέα του δένδρου απαρίθμησης και χρησιμοποιεί την ορθογωνοποίηση Gram-Schmidt για την αποκοπή απίθανων κλαδιών.

11.2 Βασικά Μαθηματικά Στοιχεία

Για μια βάση $B = \{b_1, \ldots, b_n\} \subset \mathbb{Z}^m$, ο αλγόριθμος υπολογίζει:

1. Τους συντελεστές Gram-Schmidt μ_{ij} και τα τετράγωνα των νορμών $B_i = \|b_i^*\|^2$

2. Για κάθε επιλογή συντελεστών x_i, το τετράγωνο της προβολής:

$$\ell_i = B_i(x_i - c_i)^2, \quad \text{όπου } c_i = -\sum_{j=i+1}^{n} x_j \mu_{ji}$$

3. Το άθροισμα $\text{sumli} = \sum_{j=i}^{n} \ell_j$ που αντιστοιχεί στο τετράγωνο της νόρμας του μερικού διανύσματος

11.3 Στρατηγική Διάσχισης του Δένδρου

Ο αλγόριθμος διασχίζει το δένδρο απαρίθμησης με την εξής λογική:

- **Κάθοδος:** Όταν sumli $\leq R^2$, κατεβαίνουμε στο επόμενο επίπεδο ($i \leftarrow i-1$) και θέτουμε το x_i στο αριστερό άκρο του διαστήματος:

$$x_i = \left\lfloor -\sum_{j=i+1}^{n} \mu_{ji} x_j - \sqrt{\frac{R^2 - \sum_{j=i+1}^{n} \ell_j}{B_i}} \right\rfloor$$

- **Ανάβαση:** Όταν sumli $> R^2$, ανεβαίνουμε στο προηγούμενο επίπεδο ($i \leftarrow i+1$) και αυξάνουμε το x_i κατά 1
- **Αποθήκευση:** Όταν φτάσουμε στο $i = 1$ και sumli $\leq R^2$, αποθηκεύουμε το διάνυσμα $\sum_{j=1}^{n} x_j b_j$

11.4 Υλοποίηση σε Python

Η υλοποίηση ακολουθεί πιστά τον ψευδοκώδικα και αποτελείται από δύο κύριες συναρτήσεις:

- Η συνάρτηση `gram_schmidt()` υπολογίζει τους συντελεστές μ_{ij} και τα B_i
- Η συνάρτηση `kfp_enumeration()` υλοποιεί τον κύριο αλγόριθμο απαρίθμησης
- Χρησιμοποιείται *1-based indexing* για ευθυγράμμιση με τον ψευδοκώδικα
- Προστίθεται αριθμητική ανοχή 10^{-10} για αποφυγή σφαλμάτων στρογγύλευσης

11.5 Αποτελέσματα για Πλέγμα $\mathbb{Z}^3$

Για την κανονική βάση του $\mathbb{Z}^3$ και ακτίνα $R = 1.1$:

11.5.1 Βασικά Διανύσματα (από τον αλγόριθμο)

Ο αλγόριθμος επέστρεψε 4 διανύσματα:

- $(0, 0, 0)$ με νόρμα 0
- $(1, 0, 0)$ με νόρμα 1
- $(0, 1, 0)$ με νόρμα 1
- $(0, 0, 1)$ με νόρμα 1

11.5.2 Πλήρες Σύνολο (με αντίθετα)

Προσθέτοντας τα αντίθετα των μη μηδενικών διανυσμάτων, προκύπτουν 7 διανύσματα:

- Μηδενικό διάνυσμα: $(0, 0, 0)$
- Διανύσματα νόρμας 1: $(1, 0, 0), (0, 1, 0), (0, 0, 1)$

11.5.3 Ανάλυση Αποτελεσμάτων

- Ο αλγόριθμος βρήκε ακριβώς το 50% των διανυσμάτων (4 από 7), όπως αναφέρεται στη θεωρία
- Όλα τα διανύσματα έχουν νόρμα $\leq R = 1.1$, πληρώντας τον στόχο του αλγορίθμου
- Η προσθήκη αντιθέτων δίνει πλήρη κάλυψη της σφαίρας
- Το συντομότερο μη μηδενικό διάνυσμα έχει νόρμα 1

11.6 Παρατηρήσεις και Βελτιώσεις

- **Αποτελεσματικότητα:** Ο αλγόριθμος έχει χωρική πολυπλοκότητα $O(n)$ και χρονική που εξαρτάται εκθετικά από τη διάσταση
- **Ορθότητα:** Η υλοποίηση επαληθεύτηκε με brute force για μικρά πλέγματα
- **Βελτιστοποιήσεις:** Στην πράξη, η βάση προεπεξεργάζεται με LLL ή BKZ για καλύτερη απόδοση
- **Επεκτάσεις:** Ο αλγόριθμος αποτελεί τη βάση για προβλήματα SVP και CVP

12 Άσκηση 12

Υλοποίηση και Αποτελέσματα της Άσκησης 14.23

12.1 Περιγραφή Υλοποίησης

Για την επίλυση της άσκησης υλοποιήθηκε ο αλγόριθμος απαρίθμησης **Schnorr-Euchner** (Αλγόριθμος 14.6.2), προσαρμοσμένος για την εύρεση του συντομότερου διανύσματος (Shortest Vector Problem - SVP). Η υλοποίηση πραγματοποιήθηκε σε γλώσσα Python και ενσωματώνει τη στρατηγική *pruning* όπως ορίστηκε στην εκφώνηση.

Βασικά χαρακτηριστικά της υλοποίησης:

- **Pruning Vector:** Εφαρμόστηκε ο τύπος της Άσκησης 14.23 για τον υπολογισμό των συντελεστών a_i:
$$a_i = \min\left\{1, \sqrt{1.05 \cdot \frac{i}{n}}\right\}, \quad 1 \le i \le n \tag{2}$$
- **Συνθήκη Κλαδέματος (Pruning Condition):** Στο επίπεδο i, ο έλεγχος του μερικού αθροίσματος γίνεται με βάση την προσαρμοσμένη ακτίνα, ώστε να αγνοούνται κλαδιά που δεν ικανοποιούν τη συνθήκη:
$$\text{sumli} \le a_{n+1-i}^2 \cdot R^2 \tag{3}$$
- **Δυναμική Ακτίνα:** Η μέγιστη ακτίνα αναζήτησης R δεν παραμένει σταθερή. Αρχικοποιείται με τη νόρμα του πρώτου διανύσματος της βάσης και ενημερώνεται δυναμικά κάθε φορά που εντοπίζεται ένα νέο, μικρότερο διάνυσμα στο φύλλο του δέντρου ($i = 1$). Αυτό επιταχύνει τη διαδικασία του pruning.
- **Ορθή Διαχείριση Κέντρου:** Χρησιμοποιήθηκε ο τύπος για το κέντρο του διαστήματος με αρνητικό πρόσημο ($c_i = -\sum_{j=i+1}^{n} \mu_{j,i} x_j$), διασφαλίζοντας τη σωστή κατεύθυνση της αναζήτησης προς το μηδέν.

12.2 Αποτελέσματα Επαλήθευσης (Test Cases)

Ο αλγόριθμος δοκιμάστηκε σε τρία σενάρια ελέγχου αυξανόμενης δυσκολίας για να επιβεβαιωθεί η ορθότητα της στρατηγικής backtracking και του pruning.

1. **Test 1 - Ορθογώνιο Πλέγμα 2D:** Σε βάση $\mathbf{B} = \{(5, 0), (0, 2)\}$, ο αλγόριθμος εντόπισε σωστά το διάνυσμα $(0, 2)$ με τετράγωνο νόρμας 4.0, αγνοώντας το μεγαλύτερο διάνυσμα $(5, 0)$.
2. **Test 2 - "Skewed" Βάση:** Χρησιμοποιήθηκε η βάση $\mathbf{B} = \{(10, 0), (11, 1)\}$. Παρόλο που τα διανύσματα της βάσης έχουν μεγάλες νόρμες (≈ 100 και 122), ο αλγόριθμος εκτέλεσε επιτυχώς backtracking και εντόπισε τον γραμμικό συνδυασμό $\mathbf{b}_2 - \mathbf{b}_1 = (1, 1)$, ο οποίος αποτελεί το συντομότερο διάνυσμα με τετράγωνο νόρμας 2.0.
3. **Test 3 - Πλέγμα 4 Διαστάσεων:** Σε πλέγμα 4D όπου το ελάχιστο διάνυσμα έχει τετραγωνικό μήκος 4, ο αλγόριθμος επέστρεψε σωστά το διάνυσμα $(2, 0, 0, 0)$ (με τετράγωνο νόρμας 4.0), επιβεβαιώνοντας τη λειτουργία του αλγορίθμου σε ανώτερες διαστάσεις.

Η επιτυχία, ιδιαίτερα στο Test 2, επιβεβαιώνει ότι η εφαρμογή του pruning vector a_i έγινε σωστά και δεν "έκοψε" το μονοπάτι που οδηγεί στη βέλτιστη λύση, παρά την αυστηρότητα των φραγμάτων στα ανώτερα επίπεδα του δέντρου.

Σημείωση

Για την ευκολία της συγγραφής και την αισθητική παρουσίαση της αναφοράς, χρησιμοποιήθηκε σε βαθμό η βοήθεια του $ChatGPT$**, Deepseek και Gemini**, κυρίως:

- για τη μορφοποίηση του αρχείου LaTeX ,
- για τη βελτίωση κάποιων κομματιών κώδικα, όπως οι περιττοί και χρονοβόροι έλεγχοι τα $labels$, σχόλια κλπ.
- για εύρεση επιπρόσθετων πληροφοριών

Η κατανόηση και η επίλυση των ασκήσεων έγιναν από εμένα και η χρήση του εργαλείου έγινε απλώς για διευκόλυνση και όχι για αντικατάσταση της διαδικασίας μάθησης.