

Informe de progrés II

Ampliació d'una llibreria en MAGMA per a codis Z2Z4 - lineals

Cristina Diéguez

22/05/2016



**Universitat Autònoma
de Barcelona**

Es presenten les modificacions realitzades en el treball de final de grau durant la seva execució. S'exposa la metodologia adoptada i el canvi que podria patir la planificació seguida, però es mantenen els objectius a assolir. Addicionalment, es detallen els resultats obtinguts i les conclusions provisionals que se'n desprenen. Al mateix temps, també es fa una introducció en la matèria i es detalla l'estat existent en el qual es troba actualment la temàtica.

Índex

1. Introducció	2
2. Estat de l'art	5
3. Objectius	6
4. Planificació i metodologia	7
4.1. Metodologia	7
4.2. Planificació temporal.....	9
5. Exposició de resultats.....	13
6. Conclusions provisionals	14
7. Bibliografia	15
8. Annex I.....	16

1. Introducció

Tot procés d'enviament i recepció de missatges es realitza a través d'un canal de comunicació. No obstant, el missatge rebut a la sortida del canal pot haver patit alguna alteració a causa del soroll i fa que aquest es converteixi en erroni. Aleshores, perquè el receptor pugui eliminar possibles errors es realitza el procés de codificació-descodificació.

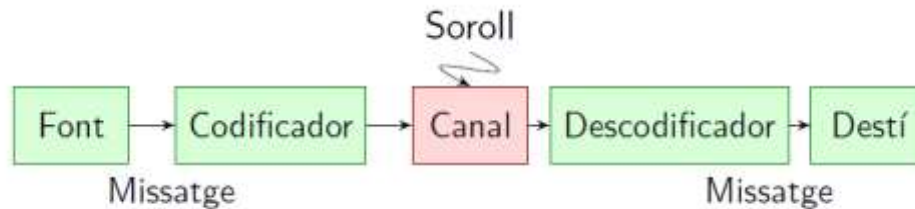


Figura 1. Procés de comunicació

En la codificació, s'assigna a cada símbol o conjunt de símbols de la font una paraula-codi afegint redundància. L'agrupació de totes aquestes paraules-codi formen un codi.

En el procés de descodificació, si s'ha produït algun error en la transmissió i el descodificador detecta l'error, es realitza la correcció aprofitant la redundància afegida. Aleshores el que es vol és poder seguir enviant missatges tenint la capacitat de detecció i correcció d'errors. El segon teorema de Shannon especifica aquest fet ja que declara que és possible transmetre informació a través d'un canal amb soroll si es transmet a una taxa de transmissió de la informació¹ inferior a la capacitat del canal. Aquest teorema dona peu a la teoria de codificació per a la correcció d'errors que té per objectiu la construcció de codificadors i descodificadors que permetin enviar el màxim d'informació amb el mínim d'errors.

L'escenari ideal seria trobar codis on cadascun dels vectors tingués una única descodificació possible, però trobar codis amb aquestes propietats no és una tasca banal. Una manera de poder minimitzar els errors seria augmentant la redundància en el missatge per poder, després, corregir els errors. El desavantatge que presenta és que existeix una penalització de la velocitat de transmissió.

Dins de l'àmbit informàtic, el tipus de codis que presenten un gran interès per les seves característiques són els codis lineals sobre anells perquè experimenten les següents propietats:

¹ Taxa de transmissió de la informació $R_T = \frac{\log_2 |C|}{n}$

1. Tancament per la suma: dues paraules codi sumades és igual a una altra paraula-codi.
2. Tancament per la multiplicació d'un escalar: si es multiplica una paraula-codi per un escalar, dóna una altra paraula-codi.

Gràcies a aquestes propietats, es pot trobar un conjunt mínim de paraules-codis linealment independents que són capaces de generar totes les paraules del codi. Aleshores, s'observa que un codi pot ser representat de forma matricial per les paraules-codi linealment independents. Es crea el que s'anomena matriu generadora que facilita el treball i la representació d'aquest tipus de codis. Amb aquest fet, es redueixen els costos de memòria i còmput a l'hora de treballar amb codis.

A part d'usar codis binaris que són un tipus de representació utilitzant l'anell d'enters mòdul 2, actualment es treballa amb codis Z_2Z_4 -additius, que són codis sobre l'anell $Z_2^\alpha \times Z_4^\beta$. Aquests codis també posseeixen el concepte de matriu generadora G que està formada per:

- γ vectors d'ordre 2: aquells formats per coordenades a $\{0,2\}$.
- δ vectors d'ordre 4: aquells que contenen alguna coordenada a $\{1,3\}$.

Coneixent aquesta informació, es determina que el codi C obtingut a partir de la matriu generadora G és d'ordre $2^\gamma \cdot 4^\delta$. Aquesta expressió, a la vegada, proporciona el número de paraules-codi de C . A l'hora de generar-les, s'opera $(x,y) \cdot G$ on x pertany a Z_2^γ i y a Z_4^δ ja que si x correspongués a Z_4 es duplicarien les mateixes paraules-codi.

Amb la finalitat de poder treballar a Z_2 , existeix una funció anomenada *Gray map* ϕ que proporciona les imatges de les paraules-codi en aquest conjunt; és a dir, realitza una bijecció:

$$\begin{array}{ccc} \phi: & Z_4 & \longrightarrow Z_2 \\ & 0 & \longrightarrow 00 \\ & 1 & \longrightarrow 01 \\ & 2 & \longrightarrow 11 \\ & 3 & \longrightarrow 10 \end{array}$$

Si $v=(v_1, v_2)$ que pertany a $Z_2^\alpha \times Z_4^\beta$, es defineix $\phi(v)=(v_1, \phi(v_2))$. Aquesta bijecció només afecta, realment, a les β coordenades quaternàries de les paraules-codi mentre que les α coordenades binàries es mantenen iguals. El codi binari $\phi(C)$ s'anomena Z_2Z_4 -lineal i, de forma general, no és lineal.

La distància de Hamming $d_H(u,v)$ entre dues paraules-codi u,v que pertanyen a Z_2 és el nombre de coordenades per les quals difereixen u i v . El pes de Hamming $w_H(u)$ és la distància de Hamming entre la paraula-codi u i la paraula-codi formada per un vector de tots 0. En codis quaternaris, que són un tipus de representació utilitzant l'anell d'enters mòdul 4, s'utilitza la mètrica de Lee. El pes de Lee $w_L(v)$ és la suma dels pesos de Lee de les seves coordenades. Els pesos d'aquestes coordenades estan definits de la següent manera:

- $w_L(0)=0$
- $w_L(1)=w_L(3)=1$
- $w_L(2)=2$

La funció Gray ϕ preserva les distàncies, transformant la distància de Lee a distància de Hamming. Gràcies a aquesta funció, s'observa que el pes de Lee d'un vector, v , sobre Z_4 coincideix amb el pes de Hamming de $\phi(v)$ sobre Z_2 . Un exemple d'aquest succés seria:

$$\phi(10230) = (0100111000)$$

$$w_L(10230) = w_L(1) + w_L(0) + w_L(2) + w_L(3) + w_L(0) = 4$$

$$w_H(0100111000) = 4$$

La distància mínima d'un codi C és el mínim valor de la distància de totes les parelles de paraules-codi diferents que formen C . Així mateix, el pes mínim d'un codi C és el mínim valor dels pesos de totes les paraules-codi. En aquest projecte, es treballarà amb aquests conceptes a l'hora de plantejar i implementar funcions que ajudin en el procés de descodificació, el qual pot ser descodificació via síndrome o descodificació per mínima distància.

2. Estat de l'art

El grup CCSG és un equip format dins del Departament d'Enginyeria de la Informació i de les Comunicacions, d'EIC, i dedicat, parcialment, a la investigació de codis Z_2Z_4 -lineals amb la finalitat dur a terme els següents objectius, entre d'altres:

- Construir i caracteritzar nous codis Z_2Z_4 -lineals, i computar els seus paràmetres estructurals.
- Caracteritzar i construir famílies de codis Reed-Muller Z_2Z_4 -lineals, i computar-ne el rang i la dimensió del kernel.
- Desenvolupar els algorismes necessaris per establir noves opcions en l'ocultació de dades i en l'autenticació de documents, utilitzant codis Z_2Z_4 -lineals.
- Expandir el software de MAGMA implementant nous paquets amb la finalitat de treballar eficientment amb codis Z_2Z_4 -lineals, codis regulars i codis no lineals.

Un dels principals articles escrits sobre codis Z_2Z_4 -lineals és [1] en què s'estudien els codis Z_2Z_4 -additius a partir dels quals s'extreu la corresponent imatge binària i s'obtenen els codis Z_2Z_4 -lineals. D'aquests nous codis, es detallen les seves propietats i les formes estàndard de les matrius generadores i de control. També esmentar l'article [5] ja que s'hi demostren les propietats necessàries pel càlcul de rangs i kernels dels codis Z_2Z_4 -lineals.

Degut que en aquest projecte serà necessari treballar amb les distàncies de Hamming dels codis ja mencionats, una de les referències a destacar és [7] perquè defineix i implementa mètodes algorítmics per la computació de la distància mínima de Hamming per codis Z_2Z_4 -lineals.

El llenguatge MAGMA porta incorporat llibreries sobre Z_2 i sobre Z_4 cosa que facilita el treball sobre aquests anells. Gràcies als membres d'aquest grup d'investigació CCSG, des de fa uns anys, es duu a terme el desenvolupament d'una nova llibreria en MAGMA per permetre treballar amb codis Z_2Z_4 -lineals, fent ús d'algunes llibreries ja existents en aquest llenguatge. Es pretén que aquesta estigui totalment integrada amb les llibreries esmentades.

Com s'ha esmentat anteriorment, s'usarà el llenguatge MAGMA amb la finalitat de realitzar la part pràctica del projecte. Gràcies a l'estructura Z_2Z_4 -lineals es pot treballar amb aquest llenguatge de manera lineal a $Z_2^\alpha \times Z_4^\beta$ i també es poden relacionar aquests codis amb codis sobre Z_2 i sobre Z_4 .

Les referències existents sobre l'ús d'aquest llenguatge en l'entorn desitjat són les proporcionades per MAGMA, [11] i [10]. Addicionalment, també es consultaran les guies que facilita el CCSG sobre MAGMA [9], per entendre els paquets de codis Z_2Z_4 -lineals [2], i codis Z_2Z_4 -additius [3].

El fet d'haver de dissenyar i implementar unes funcions que s'integraran dins de MAGMA implica que s'han de seguir els estàndards marcats per tal mantenir el format. Com a conseqüència, es consultarà la guia [8] ja que proporciona aquests estàndards.

3. Objectius

Els objectius que s'assoliran durant el transcurs del treball a realitzar es detallen a continuació:

- Realització d'un estudi dels codis Z_2Z_4 - lineals tant a nivell conceptual com de les propietats que els defineixen.
- Aprenentatge d'un nou llenguatge i entorn de programació anomenat MAGMA.
- Disseny i desenvolupament de tests de prova i tests d'integració de les funcions realitzades prèviament.
- Creació de noves funcions de codis Z_2Z_4 - lineals per realitzar els càlcul del pes mínim de Lee, la distància mínima de Lee, el nombre mínim de paraules-codi i la distribució de pesos .
- Ampliació de la llibreria de MAGMA existent sobre teoria de codis Z_2Z_4 -lineals amb les noves funcions desenvolupades per tal de facilitar la descodificació via síndrome.

Aquests objectius mencionats anteriorment van ser establerts i especificats en [4] i no es preveu cap modificació al respecte. Es conservaran els existents ja que són amb els quals es treballa pel desenvolupament complet del treball de fi de grau.

Durant l'execució del mateix, una part d'aquests objectius s'han anat assolint progressivament però la seva totalitat serà completada un cop el projecte arribi a la seva fi.

4. Planificació i metodologia

4.1. Metodologia

El procés de desenvolupament del projecte segueix un paradigma de programació d'enginyeria de software anomenat *Test-Driven Development (TDD)*, tal i com es va mencionar en [4] de forma més detallada.

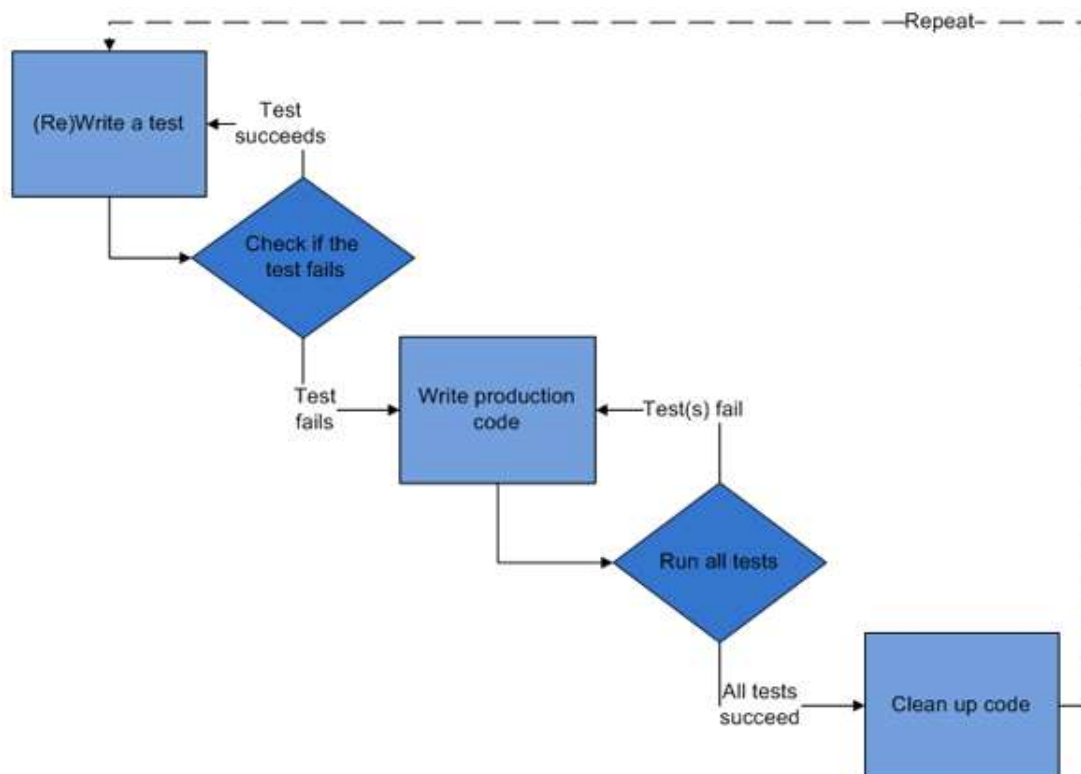


Figura 2: Cicle de vida de *Test-Driven Development*.

Les funcions previstes a desenvolupar durant tota l'execució del projecte són les següents, també esmentades amb anterioritat:

- ZZZ4MinimumLeeWeight
- ZZZ4MinimumLeeDistance
- ZZZ4MinimumWord
- ZZZ4MinimumWords
- WeightDistribution

En aquest segon període del projecte, s'estan duent a terme totes les funcions esmentades amb anterioritat, però depenent del tipus de codi introduït per les operacions aquestes poden requerir un temps de càlcul més elevat. Com a conseqüència, per cada funció s'estan implementant dos tipus d'algorismes diferents per poder estudiar sota quines característiques dels codis introduïts quin dels dos algorismes optimitza millor el temps de càlcul que requereix cadascuna.

El primer algorisme implementat es basa en la força bruta ja que recórrer totes les paraules-codi del codi C introduït realitzant els càlculs determinats per cada funció. El segon algorisme calcula el kernel i els cosets del codi C inserit mitjançant les funcions implementades a [3]. Gràcies a aquestes funcions, es pot treballar de forma lineal a Z_2 cosa que proporciona l'oportunitat d'usar funcions ja implementades sobre aquest anell, que estan optimitzades, per fer els càlculs de les funcions descrites amb anterioritat. Després els resultats pertinents es transformen a l'anell $Z_2^\alpha \times Z_4^\beta$ si és estrictament necessari.

Inicialment, es va començar a programar l'estructura de les proves de test a realitzar per cada funció ja que seguiria la mateixa per totes elles amb l'objectiu de comparar els resultats obtinguts amb els esperats. L'estructura és la mencionada a continuació:

1. Generar diversos codis Z_2Z_4 -lineals amb característiques específiques.
 - Trivial ZZZ4-additive zero code
 - Trivial ZZZ4-additive universe code
 - Repetition ZZZ4-additive code
 - A ZZZ4-additive code with $\alpha=0$
 - A ZZZ4-additive code with $\beta=0$
 - A ZZZ4-additive code with $\gamma=0$
 - A ZZZ4-additive code with $\delta=0$
 - A ZZZ4-additive code with $\kappa=0$
 - A ZZZ4-additive code with $\kappa=\alpha$
 - A ZZZ4-additive code with $\kappa \neq \gamma$
2. Determinar els valors esperats dels paràmetres a provar.
3. Realitzar el càlcul de la funció pertinent pels anteriors tipus de codi.
4. Comparar el resultat obtingut amb la funció amb el valor previst segons cada tipus de codi.

Mentre s'anava perfilant l'estructura de les proves de test, es va començar a pensar i programar la funció "WeightDistribution" amb l'algorisme de la força bruta perquè el nivell de

coneixement requerit de codis Z_2Z_4 -lineals era inferior amb aquest tipus d'algorisme. Quan aquest va estar acabat i l'única part restant que quedava era l'estil de codificació de l'arxiu MAGMA, es van desenvolupar de forma paral·lela la resta de funcions utilitzant el mateix tipus d'algorisme.

Degut a problemes de compatibilitat respecte a les crides de les funcions creades a les proves de test, es va decidir començar a treballar paral·lelament amb la concepció de les funcions usant l'algorisme de kernel i cosets. Per tal de realitzar aquesta tasca, es va mantenir una sessió formativa amb la tutora Cristina Fernández amb l'objectiu d'assolir els coneixements necessaris al respecte.

Aquesta part de la programació ha implicat un repte més elevat ja que, primerament, s'han hagut d'adquirir conceptes nous i, posteriorment, implementar-los a nivell de codi corregint bastant errors sorgits durant la mateixa.

La metodologia que es va plantejar al principi del treball no ha sofert canvis ja que inicialment es van estructurar i programar les proves de test i posteriorment es van escriure les funcions amb els dos algorismes. Degut a problemes de compatibilitat entre les proves de test i les funcions, la part d'execució dels tests va ser més costosa i tediosa del que s'esperava en un primer moment i es van haver de modificar el codi de les funcions a mesura que aquestes s'executaven amb els tests de proves.

4.2. Planificació temporal

La planificació detallada a [12] no ha sofert cap canvi significatiu que afecti a la durabilitat del treball ni a la interdependència entre les tasques que en formen part. No obstant això, sí que és possible preveure una petita modificació d'una tasca concreta causada per errors en la implementació de la codificació de les funcions usades en el projecte que implementen l'algorisme de kernel i cosets.

Aquesta tasca esmentada és a l'anomenada *"Reconstrucció de les funcions i re-execució de les proves de test"* i, segons la planificació actual, aquesta hauria d'acabar el 30 de maig de 2016. Per aquesta raó, encara es tenen uns dies restants de marge per poder acabar la tasca considerant els errors mencionats ja que s'està treballant en la gestió de la solució. Tot i aquesta petita franja de temps, podria ser possible que la tasca es demorés i s'acabés uns dies més tard dels planificats anteriorment. Si s'efectués l'ús d'aquests dies extres comptabilitzant

la tardança, aquesta no afectaria a altres tasques ja que les posteriors no estan vinculades directament amb ella, com es pot observar en les imatges següents.

La resta de tasques planificades es podrien dur a terme en les dates previstes i treballant paral·lelament si fos necessari per la seva adequada execució.

Tot i la modificació que es té en consideració a la planificació, es manté la durabilitat de tot el projecte amb l'objectiu de respectar la data de finalització del mateix.

Aquesta versió és la mateixa que també es va realitzar mitjançant un diagrama de Gantt on es van especificar els recursos del projecte i, a la vegada, la interdependència de tasques. Juntament amb aquest informe, s'afegeix el diagrama amb el format *pdf* per una millor apreciació de la planificació.






















		Nombre	Duración	Inicio	Fin	Predecessoras	Recursos
1		Primera reunió de presentació de la proposta de TFG	1d	05/02/2016	05/02/2016		Cristina Diéguez
2		Reunió: plantejament del projecte i coneixements previs	1d	11/02/2016	12/02/2016	1	Cristina Diéguez
3		Creació i lliurament: Reunió inicial	2d?	12/02/2016	15/02/2016	2	Cristina Diéguez
4		Fonaments teòrics: codis Z4 i codis ZZZ4-lineals	10d	15/02/2016	26/02/2016	2	Cristina Diéguez
5		Creació i lliurament: Informe inicial	4d?	02/03/2016	07/03/2016	4	Cristina Diéguez
6		Implementació final dels tests existents	24.88d?	06/03/2016	11/04/2016	4	Cristina Diéguez
7		Disseny i implementació dels tests per les funcions pensades	22.75d?	23/03/2016	22/04/2016	6II 6d	Cristina Diéguez
8		Disseny i implementació de les funcions (determinades informe)	29.63d?	04/04/2016	13/05/2016	6II 17d	Cristina Diéguez
9		Creació i lliurament: Informe progrés I	3.88d?	12/04/2016	15/04/2016	6,7II,88	Cristina Diéguez
10		Execució de les proves de test	20d?	25/04/2016	20/05/2016	7	Cristina Diéguez
11		Reconstrucció de les funcions i reexecució de les proves de test	16d?	09/05/2016	30/05/2016	8II,10II	Cristina Diéguez
12		Creació i lliurament: Informe progrés II	3d?	18/05/2016	20/05/2016	6	Cristina Diéguez
13		Creació article	14d?	23/05/2016	09/06/2016	7,8,10	Cristina Diéguez
14		Lliurament proposta article	1d?	10/06/2016	10/06/2016	13	Cristina Diéguez
15		Memòria final	12.88d?	06/06/2016	22/06/2016	12	
16		Lliurament final	2d?	22/06/2016	24/06/2016	14,15	Cristina Diéguez
17		Creació pòster	6d?	22/06/2016	30/06/2016	15	Cristina Diéguez
18		Lliurament pòster	1d?	01/07/2016	01/07/2016	17	Cristina Diéguez
19		Preparació de la presentació	6d?	23/06/2016	30/06/2016	15	Cristina Diéguez
20		Presentació	4d?	01/07/2016	06/07/2016	19	Cristina Diéguez

Figura 3: Planificació de tasques

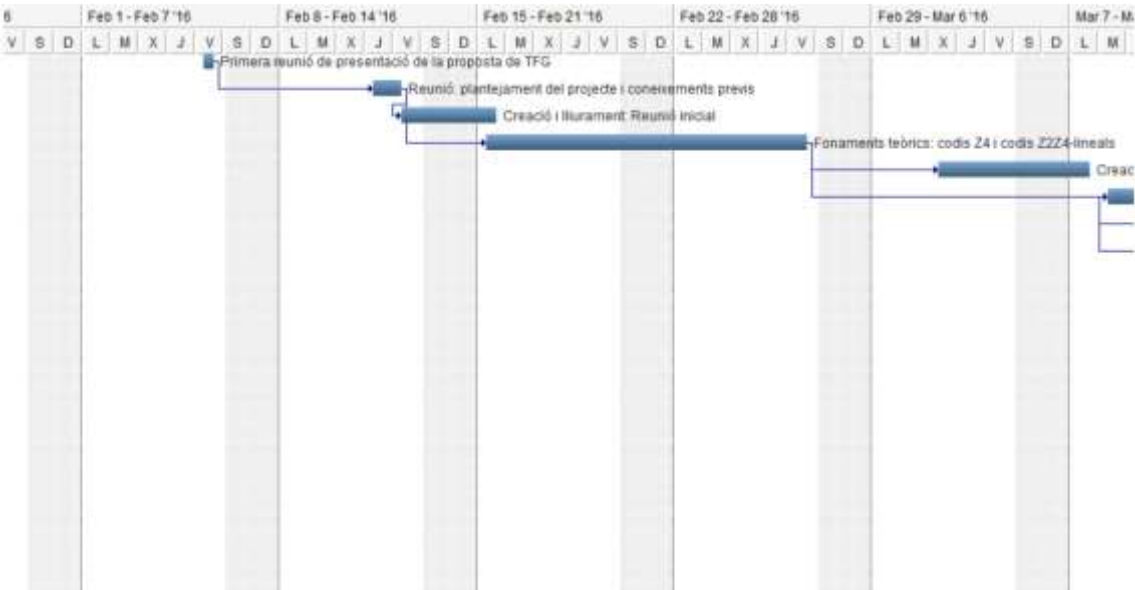


Figura 4: Diagrama temporal

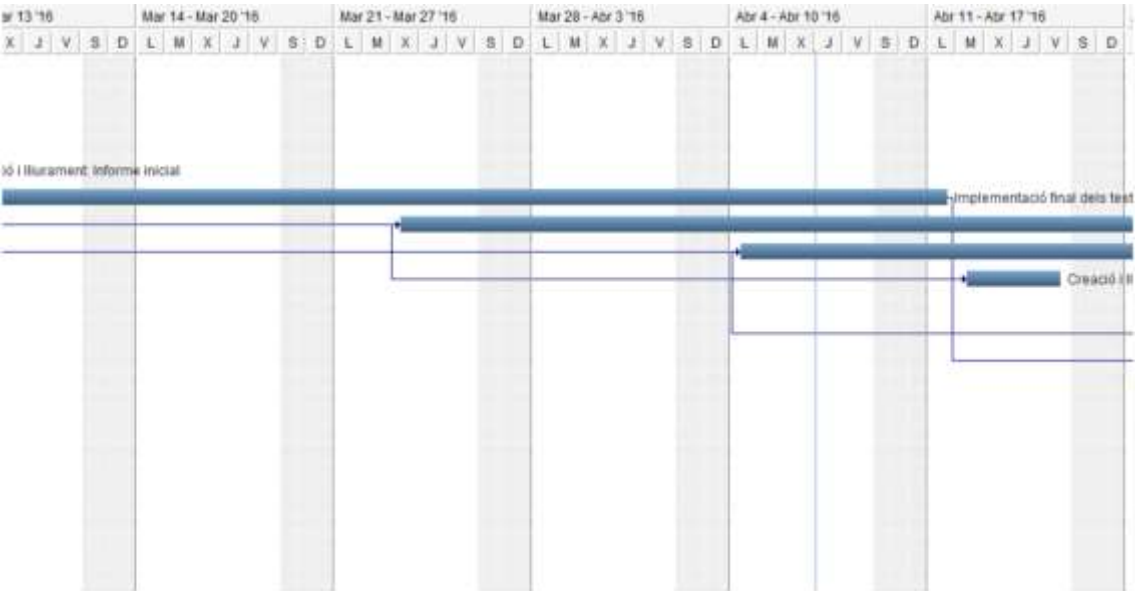


Figura 5: Diagrama temporal

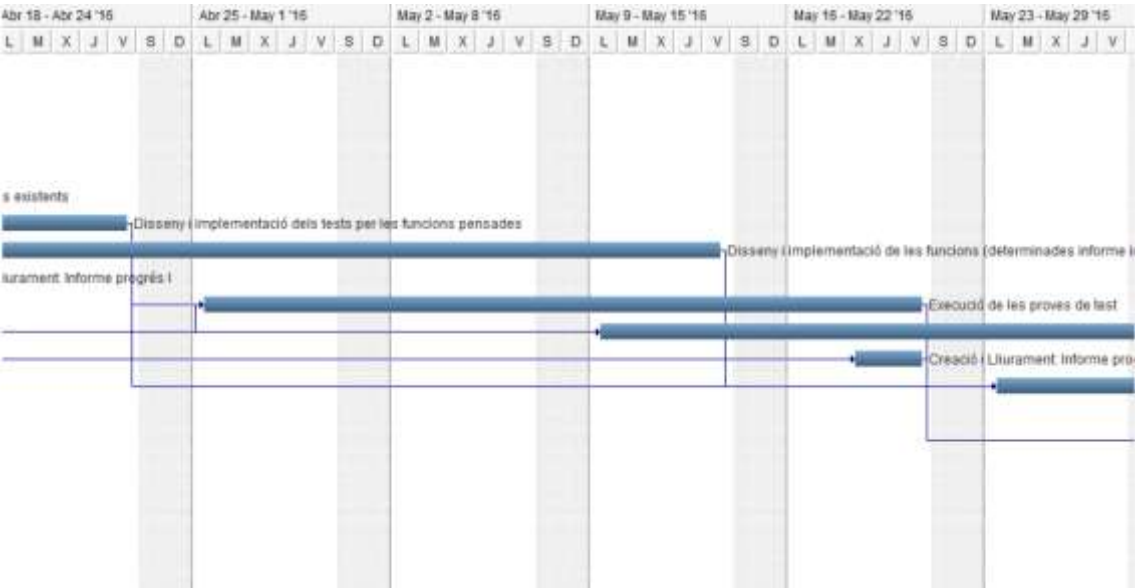


Figura 6: Diagrama temporal

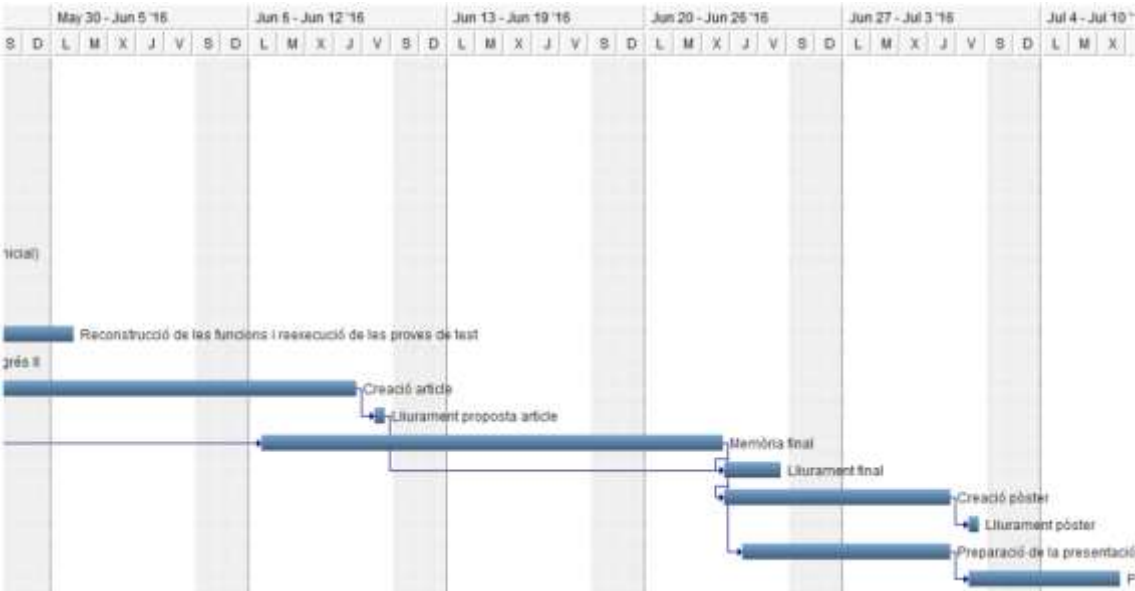


Figura 7: Diagrama temporal

5. Exposició de resultats

En aquest projecte, ha estat necessari programar diverses funcions amb les quals poder cobrir la necessitat de càlcul de les següents característiques d'un codi ZZZ4-lineal:

- Distribució de pesos.
- Pes mínim de Lee.
- Distància mínima de Lee.
- Paraula-codi mínima.
- Distribució de paraules-codi mínimes.

Primerament, es va dur a terme la programació de les funcions mencionades amb anterioritat amb els dos tipus d'algorismes: la força bruta i el càlcul de kernel i cosets. L'execució dels testos ha fet sortir a la llum errors en les funcions de kernel i cosets que estan sent gestionats per trobar la solució i aplicar-la el més aviat possible. Tot i això, les funcions estan ben estructurades per realitzar les operacions pertinents i, així, poder testejar que el resultat obtingut d'una funció per ambdós algorismes sigui el mateix.

Durant les proves dutes a terme a mesura que es programaven les funcions, es va observar que el temps d'execució dels dos algorismes era similar, sense cap diferència rellevant entre ells. En els dos casos, el procés va ser notablement ràpid exceptuant el cas del codi "ZZZ4AdditiveUniverseCode(α , β)" ja que al retornar un codi genèric ZZZ4-additiu format per totes les paraules-codi possibles implicava que el temps de càlcul augmentés considerablement respecte els altres casos. Aquesta primera mostra es va obtenir de l'execució de certs codis ZZZ4-lineal, no de tots els codis de les proves de test.

Tot i els resultats obtinguts en la part de comparació d'algorismes, serà necessari augmentar la quantitat de codis usats per l'obtenció del temps de càlcul de les funcions amb els dos algorismes diferents. D'aquesta forma, es podrà mostrar una taula de resultats més contrastada i que proporcioni més informació amb l'objectiu de comparar el comportament dels dos algorismes executant diversos tipus de codis.

L'objectiu final és extreure suficient dades rellevants que facilitin la presa de decisions a l'hora d'escollir quins dels dos algorismes és més òptim i ajudar, així, a extreure les pertinents conclusions.

6. Conclusions provisionals

En la primera etapa del projecte, mentre es programaven les funcions amb l'algorisme de la força bruta, el que es pensava a nivell teòric és que aquest tipus d'algorisme és poc eficient per realitzar les operacions ja que sempre cal recórrer totes les paraules-codi per obtenir la informació final desitjada. Aquesta forma no era massa òptima per estalviar temps de càlcul.

Posteriorment, quan es van implementar les funcions amb l'algorisme de kernel i cosets, s'esperava que aquest fos més eficient ja que el kernel i els cosets d'un codi Z2Z4-lineal es traslladen a l'anell d'enters mòdul 2 i, en aquest anell, s'usen les funcions binàries que ja estan optimitzades.

Després de les conclusions teòriques esperades sobre el comportament dels algorismes, quan es van poder dur a terme les execucions es va comprovar que el temps de càlcul d'una funció per un mateix tipus codi entre els dos algorismes era similar. És a dir, entre les funcions no hi havia una diferència rellevant a l'hora d'executar les operacions pertinents ja que, fins i tot, en el cas del codi "Z2Z4AdditiveUniverseCode(α , β)" on el temps de càlcul era molt elevat respecte la resta de codis, els dos algorismes tardaven el mateix.

Encara queden pendents més proves amb codis de diversos tipus per observar si existeix el mateix patró anterior o hi ha una diferència de temps entre els dos algorismes que faci pensar que un d'ells és més òptim i eficient per codis amb característiques determinades.

7. Bibliografia

- [1] J. Borges, C. Fernández-Córdoba, J. Pujol, J. Rifà and M. Villanueva, " $\mathbb{Z}_2\mathbb{Z}_4$ -linear codes: generator matrices and duality," *Designs, Codes and Cryptography*, vol. 54, pp. 167-179, 2010.
- [2] J. Borges, C. Fernández, J. Pujol, J. Rifà and M. Villanueva, " $\mathbb{Z}_2\mathbb{Z}_4$ -linear codes. A Magma package," Universitat Autònoma de Barcelona, 2007.
- [3] J.Borges, C. Fernández, B. Gastón, J. Pujol, J. Rifà and M. Villanueva, " $\mathbb{Z}_2\mathbb{Z}_4$ -Additive Codes. A MAGMA Package", Univeristat Autònoma de Barcelona, 2009.
- [4] C. Diéguez, "Informe inicial. Ampliació d'una llibreria en MAGMA per a codis $\mathbb{Z}_2\mathbb{Z}_4$ -lineals", Informe inicial del projecte de final de carrera, Universitat Autònoma de Barcelona, 2016.
- [5] C. Fernández-Córdoba, J. Pujol, and M. Villanueva, " $\mathbb{Z}_2\mathbb{Z}_4$ -linear codes: rank and kernel," *Designs, Codes and Cryptography* (July 2010) vol. 56. no. 1, pp. 43-59, ISSN: 0925-1022. DOI: 10.1109/TIT.2011.2119465.
- [6] B.Gastón, "Codis $\mathbb{Z}_2\mathbb{Z}_4$ -Additius en MAGMA", Projecte de final de carrera, Universitat Autònoma de Barcelona, 2008.
- [7] M. Pujol, "Computing the Minimum Hamming Distance for $\mathbb{Z}_2\mathbb{Z}_4$ -linear codes", Projecte de final de màster, Universitat Autònoma de Barcelona, 2012.
- [8] CCSG Development Group, "CCSG Style Guide", Univeristat Autònoma de Barcelona, 2011.
- [9] CCSG Development Group, "Breu Introducció al MAGMA", Univeristat Autònoma de Barcelona, 2004.
- [10] Computational Algebra Group, "Overview of MAGMA V2.19 Features", University of Sydney, 2016, <https://magma.maths.usyd.edu.au/magma/overview/pdf/overv219.pdf>.
- [11] "Handbook MAGMA", <https://magma.maths.usyd.edu.au/magma/handbook/>
- [12] C. Diéguez, "Informe de progrés I. Ampliació d'una llibreria en MAGMA per a codis $\mathbb{Z}_2\mathbb{Z}_4$ -lineals", Informe de progrés I del projecte de final de carrera, Universitat Autònoma de Barcelona, 2016.

8. Annex I

- El codi de la funció implementada com a “WeightDistribution” és el següent:

```
/* **** */
/*
/* Package or project name: Z2Z4AdditiveCodes package
/* Test file name: weightDistribution.m
/*
/* Comments:
/*
/* Authors: C. Fernández and C. Diéguez
/*
/* Revision version and last date: 1.0, 2016/03/29
```

```
/*                                                    */  
/*****/
```

intrinsic WeightDistribution (C)

```
/*Definir la longitud maxima de la llista*/  
type_code=Z2Z4Type(C);  
MAX_L:=type_code[1]+2*type_code[2]+1;  
  
/*Inicialitzar la llista a 0*/  
/*Llistes comencen a la posicio 1*/  
listLeeWeight:=[* *];  
for i:=0 to MAX_L do  
    listLeeWeight[i+1]:=0;  
end for;  
  
/*Per cada paraula-codi de C, es busca el pes de Lee i es guarda a la llista */  
/*la quantitat de paraules-codi amb pes X*/  
for c in C do  
    leeWeight:=Z2Z4LeeWeight(c, type_code[1]);  
    listLeeWeight[leeWeight+1]:=listLeeWeight[leeWeight+1]+1;  
end for;  
  
/*Es crea una llista que conte el pes i la quantitat de paraules-codi d'aquell pes*/  
listWeightDistribution:=[];  
for i:=1 to MAX_L do  
    if listLeeWeight[i] ne 0 then /*No mostrar els valors <0,0>*/
```

```

    a:=< i-1, listLeeWeight[i] >;

    listWeightDistribution:=Append( listWeightDistribution, a);

end if;

end for;

return listWeightDistribution;

end intrinsic;

```

- El codi de les proves de test per la funció anomenada “WeightDistribution” és el que es mostra a continuació:

```

/*****/

/*                                          */
/* Package or project name: ZZZ4AdditiveCodes package */
/* Test file name: weightDistribution_test.m */
/*                                          */
/* Comments: Black-box tests for the intrinsic functions */
/*      WeightDistribution() included in the */
/*      weightDistribution.m file */
/*                                          */
/* Authors: C. Fernández and C. Diéguez */
/*                                          */

```

```

/* Revision version and last date: 1.0, 2016/04/11 */

/* */

/*****/

//needs ZZZ4AdditiveCode package

SetAssertions(true);

Alarm(30*60);

/*****

GLOBAL VARIABLES

*****/

Z4 := Integers(4);

/*****/

/* */

/* Function name: ZZZ4AdditiveCode */

/* Parameters: C */

/* Function description: Create all the codewords of code C */

/* and a vector v with N positions. For each codeword u, */

/* Lee weight, wL(u), is looked for and it is added one */

/* in v where position "i" is equal to Lee weight in each */

/* position adds one if Lee weight of u is equal to */

/* position "i". */

/* */

/* Input parameters description: */

/* - C: a ZZZ4-additive code */

```

```

/* Output parameters description: */
/* - a vector "weighthDistribution" with weight distribution of code C */
/* */
/* Signature: */
/* */
/*****/

print "WEIGHT DISTRIBUTION OF GENERAL ZZZ4-ADDITIVE CODES AND SOME TRIVIAL ONES";
print "test 1: Trivial ZZZ4-additive zero code

      alpha = 2, beta = 4, gamma = 0, delta = 0, kappa = 0, length = 6, #C = 1";

R := RSpace(Z4, 6);
L := [R!0];
C := ZZZ4AdditiveCode(L : Alpha := 2);

weighthDistribution := WeightDistribution(C);
weighthDistribution;

/*Demostració a Z2*/
if HasZZZ4LinearGrayMapImage(C) then
    grayMapImage:= HasLinearGrayMapImage(C);

    print "Distribucio de pesos de Hamming:";
    expectedOutputWeightDstribution := WeightDistribution(grayMapImage);

    assert weighthDistribution eq expectedOutputWeightDstribution;
else
    print "No image Z2";

```

end if;

/******

print "test 3: Trivial Z2Z4-additive universe code

alpha = 4, beta = 8, gamma = 4, delta = 8, kappa = 4, length = 12, #C = 1048576";

C := Z2Z4AdditiveUniverseCode(4, 8);

weigthDistribution := WeightDistribution(C1);

weigthDistribution;

/*Demostració a Z2*/

if HasZ2Z4LinearGrayMapImage(C) then

grayMapImage:= HasLinearGrayMapImage(C);

print "Distribucio de pesos de Hamming:";

expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);

assert weigthDistribution eq expectedOutputWeightDsitribution;

else

print "No image Z2";

end if;

/******

print "test 4: Repetiton Z2Z4-additive code

alpha = 4, beta = 8, gamma = 1, delta = 0, kappa = 1, length = 12, #C = 2";

C := Z2Z4AdditiveRepetitionCode(4, 8);

```
weighthDistribution := WeightDistribution(C1);
```

```
weighthDistribution;
```

```
/*Demostració a Z2*/
```

```
if HasZ2Z4LinearGrayMapImage(C) then
```

```
    grayMapImage:= HasLinearGrayMapImage(C);
```

```
    print "Distribucio de pesos de Hamming:";
```

```
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);
```

```
    assert weighthDistribution eq expectedOutputWeightDsitribution;
```

```
else
```

```
    print "No image Z2";
```

```
end if;
```

```
/******
```

```
print "test 5: A Z2Z4-additive code with alpha, beta even, p=Id.
```

```
    alpha = 10, beta = 20, gamma = 6, delta = 2, kappa = 1, length = 30, #C = 1024";
```

```
M := Matrix(Z4,[[2,0,2,0,0,0,2,0,2,2,0,0,0,1,0,0,0,3,2,1,2,0,1,2,0,1,2,1,0,3],
```

```
    [0,2,2,0,0,2,0,0,2,2,1,1,1,1,0,1,0,3,2,0,0,2,2,0,1,0,3,2,1,1],
```

```
    [0,0,0,0,2,0,2,2,2,0,0,0,0,1,0,0,0,1,2,3,0,0,1,0,2,3,0,3,2,3],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,2,0,0,2,0,2,2,0,0,2,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0,2,2,2,2,0,0,2,2,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,2,0,2,0,0,0,2,2,0,2,2,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,0,0,2,0,0,2,0,2,0,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,2,0,0,2,2,0,2,0,0,2,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,2,2,2,0,2,0,2,2,0,0],
```

```

[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,0,0,0,0,2,2,0,2,2,2]]);

C := Z2Z4AdditiveCode(M : Alpha := 10);

weighthDistribution := WeightDistribution(C1);

weighthDistribution;

/*Demostració a Z2*/

if HasZ2Z4LinearGrayMapImage(C) then

  grayMapImage:= HasLinearGrayMapImage(C);

  print "Distribucio de pesos de Hamming:";

  expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);

  assert weighthDistribution eq expectedOutputWeightDsitribution;

else

  print "No image Z2";

end if;

/*****/

print "test 6: A Z2Z4-additive code with alpha even, beta odd, p <> Id in delta quaternary part

alpha = 10, beta = 19, gamma = 7, delta = 2, kappa = 3, length = 29, #C = 2048";

M := Matrix(Z4,[[2,0,2,0,0,0,2,0,2,2,0,0,0,1,0,0,0,3,2,1,2,0,1,2,0,1,2,1,0],

[0,2,2,0,0,2,0,0,2,2,1,1,1,0,1,0,3,2,0,0,2,2,0,2,0,3,2,2],

[0,0,0,0,2,0,2,2,2,0,0,0,0,1,0,0,0,1,2,3,0,0,1,0,2,3,0,3,2],

[0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,2,0,0,2,0,2,2,0,0,2],

[0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0,2,2,2,2,0,0,2,2,2],

```



```

[0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,2,0,2,0,0,0,2,2,0,2,2,2],
[0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,0,0,2,0,0,2,0,2,0],
[0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,0,2,0,0,2,2,0,2,0,0,2],
[0,2,0,2,0,0,0,0,0,0,0,0,2,0,0,2,0,2,2,2,0,2,0,2,2,0],
[0,0,2,0,2,0,2,0,2,0,2,0,0,0,2,2,0,0,0,0,2,2,0,2,2,2]]);
C := ZZ4AdditiveCode(M : Alpha := 10);

weighthDistribution := WeightDistribution(C1);

weighthDistribution;

/*Demostració a Z2*/
if HasZZ4LinearGrayMapImage(C) then
  grayMapImage:= HasLinearGrayMapImage(C);

  print "Distribucio de pesos de Hamming:";
  expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);

  assert weighthDistribution eq expectedOutputWeightDsitribution;
else
  print "No image Z2";
end if;

/*****/

print "test 7: A ZZ4-additive code with alpha even, beta odd, p <> Id in non delta quaternary
part.

alpha = 10, beta = 19, gamma = 7, delta = 2, kappa = 3, length = 29, #C = 2048";

M := Matrix(Z4,[[2,0,0,0,0,0,2,2,2,2,0,0,2,2,0,0,2,0,2,0,2,0,0,0,0,0,0,0,0],

```

```

[0,2,0,2,0,0,0,0,0,0,0,2,0,2,2,0,0,2,0,2,2,2,0,0,2,0,0,0],
[0,0,2,0,2,0,2,0,2,0,0,0,2,0,0,2,0,2,0,0,0,2,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,2,0,0,0,2,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,2,2,0,0,2,2,0,0,0,2,0,0,0,2,0,0,0,0],
[0,0,0,0,0,0,0,0,0,2,0,2,0,2,2,0,0,0,2,2,2,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0,0,2,2,2,0,0,0,0,2,2,2,2,0,0,0,2,0,0],
[0,0,0,2,2,2,0,0,2,3,3,3,0,3,2,3,0,2,0,2,0,0,0,0,1,0],
[0,0,0,0,2,0,2,2,2,0,2,0,0,3,0,0,0,3,0,3,0,0,3,0,0,1,0,1]])

C := ZZZ4AdditiveCode(M : Alpha := 10);

weightDistribution := WeightDistribution(C1);
weightDistribution;

/*Demostració a Z2*/
if HasZZZ4LinearGrayMapImage(C) then
  grayMapImage:= HasLinearGrayMapImage(C);

  print "Distribucio de pesos de Hamming:";
  expectedOutputWeightDstribution := WeightDistribution(grayMapImage);

  assert weightDistribution eq expectedOutputWeightDstribution;
else
  print "No image Z2";
end if;

/*****/
print "test 8: A ZZZ4-additive code with alpha even, beta odd, p <> Id in alpha part.

```

```

alpha = 10, beta = 19, gamma = 7, delta = 2, kappa = 3, length = 29, #C = 2048";

M := Matrix(Z4,[[0,0,2,0,2,0,2,0,2,0,0,0,2,0,0,0,2,0,2,0,0,0,0,2,2,2,0],
[2,0,0,0,0,0,2,2,2,0,0,0,0,0,2,0,2,0,2,0,2,0,2,2,2,0],
[2,0,2,0,2,0,0,2,0,2,0,0,0,0,2,2,0,0,2,0,0,0,0,2,0,2,0,0],
[0,0,2,0,2,0,2,0,2,0,2,0,0,0,2,0,2,0,2,0,0,2,2,0,2,0],
[0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,2,0,0],
[2,0,0,0,0,0,2,2,2,0,0,0,0,2,0,2,2,2,2,0,0,0,0,0,2,2,0],
[2,0,2,2,2,0,0,2,0,2,0,0,0,0,2,0,0,0,0,0,2,0,0,0,2,0,2,2],
[0,0,2,0,0,0,0,2,0,0,0,0,0,1,2,0,2,1,0,3,0,1,1,0,2,0,2,0,1],
[0,2,0,2,0,2,0,2,2,2,1,1,0,0,1,2,0,2,3,0,1,1,2,0,2,2,3,1]]);

C := Z2Z4AdditiveCode(M : Alpha := 10);

weighthDistribution := WeightDistribution(C1);

weighthDistribution;

/*Demostració a Z2*/
if HasZ2Z4LinearGrayMapImage(C) then
    grayMapImage:= HasLinearGrayMapImage(C);

    print "Distribucio de pesos de Hamming:";
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);

    assert weighthDistribution eq expectedOutputWeightDsitribution;
else
    print "No image Z2";
end if;

```

```

/*****/

print "test 9: A Z2Z4-additive code with alpha=0

      alpha = 0, beta = 19, gamma = 7, delta = 2, kappa = 0, length = 19, #C = 2048";

M := Matrix(Z4,[[0,0,2,0,0,0,2,0,2,0,0,0,0,2,2,2,2,0],
               [0,0,0,0,0,0,2,0,2,0,2,0,2,2,2,2,0],
               [0,0,0,0,2,2,0,0,2,0,0,0,0,2,0,2,0,0],
               [0,2,0,0,0,0,2,0,2,0,2,0,0,2,2,2,0,0],
               [0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,2,0,0],
               [0,0,0,0,2,0,2,2,2,2,2,0,0,0,0,0,2,2,0],
               [0,0,0,0,2,0,0,0,0,0,2,0,0,0,0,0,2,0,2,2],
               [0,0,0,1,2,0,2,1,0,3,0,1,1,0,2,0,2,0,1],
               [1,1,1,0,0,1,2,0,2,3,0,1,1,2,0,2,2,3,1]]);

C := Z2Z4AdditiveCode(M : Alpha := 0);

weighDistribution := WeightDistribution(C1);

weighDistribution;

/*Demostració a Z2*/

if HasZ2Z4LinearGrayMapImage(C) then

  grayMapImage:= HasLinearGrayMapImage(C);

  print "Distribucio de pesos de Hamming:";

  expectedOutputWeightDstribution := WeightDistribution(grayMapImage);

  assert weighDistribution eq expectedOutputWeightDstribution;

else

  print "No image Z2";

```

```
end if;
```

```
/******
```

```
print "test 10: A ZZZ4-additive code with beta=0
```

```
alpha = 10, beta = 0, gamma = 5, delta = 0, kappa = 5, length = 10, #C = 32";
```

```
M := Matrix(Z4,[[0,0,2,0,2,0,2,0,2,0],
```

```
[2,0,0,0,0,0,2,2,2,2],
```

```
[2,0,2,0,2,0,0,2,0,2],
```

```
[0,0,2,0,2,0,2,0,2,0],
```

```
[0,0,0,2,0,0,0,0,0,0],
```

```
[2,0,0,0,0,0,2,2,2,2],
```

```
[2,0,2,2,2,0,0,2,0,2],
```

```
[0,0,2,0,0,0,0,2,0,0],
```

```
[0,2,0,2,0,2,0,2,2,2]]);
```

```
C := ZZZ4AdditiveCode(M : Alpha:=10);
```

```
weighDistribution := WeightDistribution(C1);
```

```
weighDistribution;
```

```
/*Demostració a Z2*/
```

```
if HasZZZ4LinearGrayMapImage(C) then
```

```
grayMapImage:= HasLinearGrayMapImage(C);
```

```
print "Distribucio de pesos de Hamming:";
```

```
expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);
```

```
assert weighDistribution eq expectedOutputWeightDsitribution;
```

```
else
```

```
    print "No image Z2";
```

```
end if;
```

```
/******
```

```
print "test 11: A Z2Z4-additive code with gamma=0
```

```
    alpha = 10, beta = 19, gamma = 0, delta = 7, kappa = 0, length = 19, #C = 16384";
```

```
M := Matrix(Z4,[[2,2,0,0,2,2,0,2,2,2,0,0,0,1,0,0,0,0,2,3,2,1,1,3,3,1,2,3,0],
```

```
    [0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,1,0,3,0,2,2,1,3,1,1,0,1],
```

```
    [0,0,2,2,0,0,2,2,2,2,0,1,0,0,0,0,0,0,2,3,1,1,0,2,1,0,0,0,1],
```

```
    [2,0,0,0,2,0,2,0,0,0,0,0,0,0,0,0,1,0,3,3,1,3,1,0,0,3,1,0,2],
```

```
    [2,2,2,2,2,0,0,0,0,0,1,0,0,0,0,1,0,0,0,2,3,0,0,1,1,0,2,2,1],
```

```
    [2,0,2,2,2,0,2,2,0,2,0,0,1,0,0,3,0,0,0,1,2,3,1,3,0,2,1,3,2],
```

```
    [0,2,2,0,2,0,2,2,0,0,0,0,0,0,1,1,0,0,0,3,3,2,0,2,0,3,1,1,3]]);
```

```
C := Z2Z4AdditiveCode(M : Alpha := 10);
```

```
weighDistribution := WeightDistribution(C1);
```

```
weighDistribution;
```

```
/*Demostració a Z2*/
```

```
if HasZ2Z4LinearGrayMapImage(C) then
```

```
    grayMapImage:= HasLinearGrayMapImage(C);
```

```
    print "Distribucio de pesos de Hamming:";
```

```
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);
```

```
    assert weighDistribution eq expectedOutputWeightDsitribution;
```

```
else
```

```
    print "No image Z2";
```

```
end if;
```

```
/******
```

```
print "test 12: A Z2Z4-additive code with delta=0
```

```
    alpha = 10, beta = 19, gamma = 7, delta = 0, kappa = 4, length = 19, #C = 128";
```

```
M := Matrix(Z4,[[2,0,0,0,0,0,2,2,2,0,0,0,0,0,2,2,0,0,0,0,2,0,2,0,2,2],
```

```
    [0,2,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,2,2,2],
```

```
    [0,0,2,0,0,2,2,0,2,0,0,0,0,2,0,2,2,0,2,0,0,0,2,2,2,0,0],
```

```
    [0,0,0,2,0,2,0,0,2,0,0,0,0,2,0,0,0,2,0,2,2,0,0,2,0,2,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,0,2,0,2,0,0,0,0,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,2,0,2,0,2,2,2,0,2,0,2,2,2,0,2],
```

```
    [0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,2,0,0,0,0,2,2,0,2,2]]);
```

```
C := Z2Z4AdditiveCode(M : Alpha := 10);
```

```
weighDistribution := WeightDistribution(C1);
```

```
weighDistribution;
```

```
/*Demostració a Z2*/
```

```
if HasZ2Z4LinearGrayMapImage(C) then
```

```
    grayMapImage:= HasLinearGrayMapImage(C);
```

```
    print "Distribucio de pesos de Hamming:";
```

```
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);
```

```
    assert weighDistribution eq expectedOutputWeightDsitribution;
```

```
else
```

```
    print "No image Z2";
```

```
end if;
```

```
/******
```

```
print "test 13: A Z2Z4-additive code with kappa=0
```

```
    alpha = 10, beta = 19, gamma = 7, delta = 2, kappa = 0, length = 29, #C = 2048";
```

```
M := Matrix(Z4,[[0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,0,0,0,0,2,2,2,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,2,0,2,0,2,2,2,2,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,0,2,2,0,0,2,0,0,0,2,0,2,2,0,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,0,2,0,2,0,2,2,2,0,2,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,2,2,0,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,0,2,0,2,2,2,2,2,0,0,0,0,2,2,0],
```

```
    [0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,2,0,0,0,0,2,0,2,2],
```

```
    [0,0,2,0,0,0,2,0,0,0,0,0,0,1,2,0,2,1,0,3,0,1,1,0,2,0,2,0,1],
```

```
    [0,2,0,2,0,2,0,2,2,2,1,1,1,0,0,1,2,0,2,3,0,1,1,2,0,2,2,3,1]]);
```

```
C := Z2Z4AdditiveCode(M : Alpha := 10);
```

```
weightDistribution := WeightDistribution(C1);
```

```
weightDistribution;
```

```
/*Demostració a Z2*/
```

```
if HasZ2Z4LinearGrayMapImage(C) then
```

```
    grayMapImage:= HasLinearGrayMapImage(C);
```

```
    print "Distribucio de pesos de Hamming:";
```

```
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);
```



```

    assert weigthDistribution eq expectedOutputWeightDsitribution;
else
    print "No image Z2";
end if;

/*****/

print "test 14: A Z2Z4-additive code with kappa=alpha

    alpha = 7, beta = 12, gamma = 8, delta = 2, kappa = 7, length = 19, #C = 4096";

M := Matrix(Z4,[[0,0,0,2,0,2,2,0,0,0,2,2,0,0,2,2,0,0,0],
    [0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,2,2,0,0],
    [0,0,2,0,2,2,2,0,0,0,2,2,0,0,0,0,0,0,0],
    [0,0,0,2,0,0,0,0,0,2,2,2,0,0,0,2,0,0,0],
    [2,0,0,0,2,2,0,0,0,0,2,0,0,0,0,0,0,0,0],
    [0,2,0,0,2,2,0,0,0,0,0,2,0,2,0,0,0,0,0],
    [0,2,0,2,2,0,0,0,2,0,2,2,0,0,0,0,0,0,2],
    [0,2,0,2,0,0,0,0,0,0,2,2,0,0,0,0,0,2,2],
    [0,2,0,0,0,0,2,0,0,0,0,0,1,1,1,2,1,1,2],
    [0,2,0,2,0,0,2,1,1,1,2,0,0,1,0,0,1,0,3]]);

C := Z2Z4AdditiveCode(M : Alpha:=7);

weigthDistribution := WeightDistribution(C1);

weigthDistribution;

/*Demostració a Z2*/

if HasZ2Z4LinearGrayMapImage(C) then
    grayMapImage:= HasLinearGrayMapImage(C);

```

```

print "Distribucion de pesos de Hamming:";

expectedOutputWeightDistribution := WeightDistribution(grayMapImage);

assert weigthDistribution eq expectedOutputWeightDistribution;

else

    print "No image Z2";

end if;

/*****/

print "test 15: A Z2Z4-additive code with kappa=gamma

    alpha = 9, beta = 12, gamma = 8, delta = 2, kappa = 8, length = 21, #C = 4096";

M := Matrix(Z4,[[0,0,0,2,0,2,2,2,2,0,0,2,0,0,0,0,2,0,2,0,0],

    [0,0,0,0,2,2,2,0,2,0,0,0,0,0,0,2,2,0,0,0,0],

    [2,2,2,0,0,0,2,2,2,0,0,0,0,0,2,0,0,0,0,0,0],

    [0,0,0,0,2,0,2,2,0,0,0,0,2,0,0,0,2,0,0,0,0],

    [0,0,0,2,0,0,0,2,0,0,0,0,0,0,0,0,0,0,2,2,0],

    [2,0,0,0,0,0,0,2,2,0,0,0,0,2,0,0,0,0,0,0,0],

    [2,0,2,2,2,0,2,0,2,0,0,0,0,0,0,0,0,2,0,0,0],

    [2,0,0,2,0,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2],

    [2,2,0,2,0,2,0,0,2,0,1,0,0,1,1,0,2,1,2,1,1],

    [2,2,0,0,0,2,0,2,0,1,0,1,1,1,0,1,0,1,2,1,0]]);

C := Z2Z4AdditiveCode(M : Alpha := 9);

weigthDistribution := WeightDistribution(C1);

weigthDistribution;
```

```

/*Demostració a Z2*/

if HasZ2Z4LinearGrayMapImage(C) then

    grayMapImage:= HasLinearGrayMapImage(C);

    print "Distribucio de pesos de Hamming:";

    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);

    assert weigthDistribution eq expectedOutputWeightDsitribution;

else

    print "No image Z2";

end if;

/*****/

print "test 16: A Z2Z4-additive code with a zero column in alpha part.

    alpha = 9, beta = 12, gamma = 6, delta = 3, kappa = 4, length = 21, #C = 4096";

M := Matrix(Z4,[[2,0,0,0,2,2,2,0,2,0,0,0,0,2,0,0,0,0,0,0,2],

    [2,0,0,2,0,0,2,0,2,0,0,2,0,2,0,0,2,0,0,2,0],

    [0,0,2,2,0,2,2,0,0,0,0,0,0,2,2,0,0,0,2,2,0],

    [0,0,2,2,0,2,2,0,0,0,0,0,0,0,0,2,2,0,2,2,0],

    [0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0],

    [2,0,0,2,0,0,2,0,2,0,0,0,0,0,0,0,0,2,0,2,0],

    [0,0,0,0,0,2,2,2,2,0,0,0,1,0,1,1,0,0,1,2,1],

    [0,0,2,0,2,2,0,0,2,1,0,1,0,2,0,1,0,1,0,3,0],

    [0,0,0,2,0,2,2,0,2,0,1,1,0,1,0,0,2,0,1,0,0]]);

C := Z2Z4AdditiveCode(M : Alpha := 9);

weigthDistribution := WeightDistribution(C1);

```

```
weigthDistribution;
```

```
/*Demostració a Z2*/
```

```
if HasZZZ4LinearGrayMapImage(C) then
```

```
    grayMapImage:= HasLinearGrayMapImage(C);
```

```
    print "Distribucio de pesos de Hamming:";
```

```
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);
```

```
    assert weigthDistribution eq expectedOutputWeightDsitribution;
```

```
else
```

```
    print "No image Z2";
```

```
end if;
```

```
/******
```

```
print "test 17: A ZZZ4-additive code with a zero column in beta part.
```

```
    alpha = 9, beta = 12, gamma = 6, delta = 3, kappa = 4, length = 21, #C = 4096"
```

```
M := Matrix(Z4,[[2,0,0,0,2,2,2,0,2,0,0,0,2,0,0,0,0,0,2],
```

```
    [2,0,0,2,0,0,2,0,2,0,0,2,0,2,0,0,2,0,0,2,0],
```

```
    [0,0,2,2,0,2,2,0,0,0,0,0,0,2,2,0,0,0,2,2,0],
```

```
    [0,0,2,2,0,2,2,0,0,0,0,0,0,0,0,2,2,0,2,2,0],
```

```
    [0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0],
```

```
    [2,0,0,2,0,0,2,0,2,0,0,0,0,0,0,0,0,0,0,2,0],
```

```
    [0,0,0,0,0,2,2,2,2,0,0,0,1,0,1,1,0,0,1,2,1],
```

```
    [0,0,2,0,2,2,0,0,2,1,0,1,0,2,0,1,0,0,0,3,0],
```

```
    [0,0,0,2,0,2,2,0,2,0,1,1,0,1,0,0,2,0,1,0,0]]];
```

```
C := ZZZ4AdditiveCode(M : Alpha := 9);
```

```

weighthDistribution := WeightDistribution(C1);
weighthDistribution;

/*Demostració a Z2*/
if HasZ2Z4LinearGrayMapImage(C) then
    grayMapImage:= HasLinearGrayMapImage(C);

    print "Distribucio de pesos de Hamming:";
    expectedOutputWeightDsitribution := WeightDistribution(grayMapImage);

    assert weighthDistribution eq expectedOutputWeightDsitribution;
else
    print "No image Z2";
end if;

/*****/

print "test 18: Another ZZZ4-additive code with two zero columns in beta part.
      alpha = 9, beta = 12, gamma = 6, delta = 3, kappa = 4, length = 21, #C = 4096";
M := Matrix(Z4,[[2,0,0,0,2,2,2,0,2,0,0,0,0,2,0,0,0,0,0,2],
               [2,0,0,2,0,0,2,0,2,0,2,0,2,0,2,0,0,0,0,0],
               [0,0,2,2,0,2,2,0,0,0,0,0,0,2,2,0,0,0,2,0,0],
               [0,0,2,2,0,2,2,0,0,0,0,0,0,0,0,2,2,0,2,0,0],
               [0,0,0,0,2,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0],
               [2,0,0,2,0,0,2,0,2,0,0,0,0,0,0,0,0,0,0,0,0],
               [0,0,0,0,0,2,2,2,2,0,0,0,1,0,1,1,0,0,1,0,1],
               [0,0,2,0,2,2,0,0,2,1,0,1,0,2,0,1,0,0,0,0,0],

```

```
[0,0,0,2,0,2,2,0,2,0,0,0,1,0,1,1,0,0,1,0,0]]);  
  
C := Z2Z4AdditiveCode(M : Alpha := 9);  
  
weighthDistribution := WeightDistribution(C1);  
  
weighthDistribution;  
  
/*Demostració a Z2*/  
  
if HasZ2Z4LinearGrayMapImage(C) then  
  grayMapImage:= HasLinearGrayMapImage(C);  
  
  print "Distribucio de pesos de Hamming:";  
  expectedOutputWeightDistribution := WeightDistribution(grayMapImage);  
  
  assert eighthDistribution eq expectedOutputWeightDistribution;  
  
else  
  print "No image Z2";  
  
end if;
```