Eines Informàtiques per a les Matemàtiques

Treball individual de llenguatge C

Programa a lliurar fins el 7 de juny de 2022

Resum

Les tasques que haurà de realitzar el programa seran codificar i descodificar fitxers usant el codi de Reed-Solomon.

Normes generals

Caldrà escriure, individualment, un programa en llenguatge C per dur a terme la tasca proposada. Un cop el treball lliurat i corregit, hi haurà una entrevista individual amb el professor, on caldrà mostrar i explicar el funcionament del programa.

Consells

- Recordeu que aquest és un treball *individual*. És possible, i fins i tot recomanable, intercanviar informació i idees amb els companys, però copiar està estrictament prohibit.
- En particular, *no compartiu arxius*. Entregar un programa copiat total o parcialment pot implicar suspendre *l'assignatura* (vegeu la guia docent).
- Us podeu també ajudar amb materials obtinguts d'internet, on podeu trobar codi, vídeos explicatius i altre material útil per algunes de les tasques que ha de fer el programa. Compte amb copiar *codi*! Haureu d'explicar com funciona el programa en una entrevista.
- Tots els fitxers que entregueu han de començar per `CognomNom` (on `Nom` vol dir el vostre nom i `Cognom` el vostre cognom). Per exemple, el fitxer font es pot dir `GarciaJoan.c`.
- Podeu usar qualsevol compilador i sistema operatiu en preparar el programa, però us haureu d'assegurar que useu C estàndard. El codi lliurat s'haurà de poder compilar amb la instrucció

  ```
  gcc -std=c11 -Wall -pedantic -o CognomNom CognomNom.c
  ```

  Observeu que les opcions `-std=c11`, `-Wall` i `-pedantic` es poden també activar en Code::Blocks des del menú `Settings→Compiler` activant les caselles Have gcc follow the 2011 ISO C language standard, Enable all common compiler warnings, i Enable warnings demanded by strict ISO C.
- La data límit és el 7 de juny de 2022 a les 23:59. El lliurament s'ha de fer a través del Campus Virtual.
- La nota sortirà del programa i l'entrevista obligatòria posterior a l'entrega.

1 Funcions bàsiques amb vectors de $\mathbb{F}_p$

El programa que heu de fer és força complex. Per aquest motiu hem desglossat la feina en tres seccions. En el primer bloc descrivim algunes funcions bàsiques necessàries per al programa, que ja podeu començar a programar. Es tracta de funcions que fan càlculs d'àlgebra lineal en el cos finit de p elements, $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$, on p és un nombre primer.

Per comprovar cada funció us recomanem escriure un programa amb una `main` molt simple, que llegeixi del teclat les dades (`scanf`) i cridi a cadascuna de les funcions amb els paràmetres adients.

Aritmètica a $\mathbb{F}_p$

1. Comprovació de si un nombre p és primer.
2. Operacions al cos $\mathbb{F}_p$. Cal programar una funció per trobar l'invers d'un nombre a que sigui diferent de zero mòdul p. Recordeu que la identitat de Bézout $ra + sp = \mathrm{mcd}(a, p)$, en el cas que a no és múltiple del primer p implica $ra \equiv 1 \pmod{p}$. Recordeu també que l'algoritme d'Euclides permet trobar els nombres r, s de la identitat de Bézout.
3. Càlcul de l'ordre multiplicatiu d'un element no nul a $\mathbb{F}_p$, és a dir, donat a, trobar el mínim enter $n > 0$ tal que $a^n \equiv 1 \pmod{p}$.
4. Determinació d'un element primitiu, és a dir, d'un $a \in \mathbb{F}_p$ d'ordre exactament $p - 1$. (Més endavant en el grau demostrareu que n'existeixen per a tot p primer; ara només cal fer un bucle que calculi els ordres dels elements diferents de zero fins que en trobi un d'ordre $p-1$.)

En general, a l'hora de mostrar en pantalla els elements de $\mathbb{F}_p$, podeu usar els seus representants entre 0 i $p - 1$; per exemple, els elements de $\mathbb{F}_{13}$ es poden mostrar com `0`, `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `10`, `11`, i `12`.

Àlgebra lineal

1. Funcions per llegir i escriure matrius i vectors, com les `llegeixvector`, `llegeixmatriu`, `imprimeixvector` i `imprimeixmatriu` de la pràctica 3.
2. La funció `matriuxvector` de la pràctica 3, adaptada per usar coeficients de $\mathbb{F}_p$.
3. Multiplicar un vector (o fila d'una matriu) per un element de $\mathbb{F}_p$.
4. Intercanviar dues files d'una matriu.
5. Sumar a una fila d'una matriu un múltiple d'una altra (sobre $\mathbb{F}_p$).
6. Reducció Gaussiana d'una matriu de $\mathbb{F}_p$.
7. Discussió de sistemes d'equacions lineals a $\mathbb{F}_p$ (Compatible Determinat / Compatible Indeterminat / Incompatible). Si el sistema és compatible determinat, la funció haurà de trobar la solució.

Proposta opcional, no necessària per al programa final

A partir del que teniu fet, no seria difícil programar la solució completa de sistemes lineals, és a dir, per als sistemes compatibles indeterminats donar una solució particular i una base de l'espai director.

2 Algorismes de Reed-Solomon i Berlekamp-Welch

Els codis de Reed-Solomon són una família de *codis correctors d'errors* introduïts per Irving S. Reed i Gustave Solomon el 1960. S'han usat en nombroses aplicacions, incloses tecnologies de comunicació, d'emmagatzematge de dades (discos durs, CDs, DVDs), gestió comercial (codis de barres) o màrqueting (codis QR). En tots els casos, la finalitat d'un codi corrector d'errors és permetre la lectura (o descodificació) correcta d'informació malgrat l'existència d'imperfeccions en el mitjà (per exemple, un CD/DVD es pot seguir usant encara que tingui algunes ratllades; els codis QR s'acostumen a descodificar correctament encara que estiguin bruts o les càmeres no enfoquin perfectament; les comunicacions via satèl·lit són fiables malgrat la presència d'interferències).

Normalment l'usuari no percep l'existència del codi corrector, ja que el procés de codificació i descodificació/correcció és automàtic; la seva presència només es manifesta en una fiabilitat superior. Així doncs, la finalitat dels codis correctors d'errors no és amagar o dificultar la lectura de la informació, sinó assegurar-ne la integritat. Per aconseguir aquest fi, s'introdueix en el "missatge" a escriure / transmetre una *redundància* que farà el missatge més llarg, però permetrà detectar i fins i tot corregir errors (si aquests no hi són en una quantitat excessiva.)

Des de la seva introducció el 1960, s'han proposat múltiples variants dels codis de Reed-Solomon, així com algorismes optimitzats per a la seva implementació. En aquest treball en programarem una versió simple, formalitzada en termes d'àlgebra lineal sobre el cos $\mathbb{F}_p$ de p elements, on p és un nombre primer. Així doncs, suposarem que la informació a codificar arriba en la forma d'una successió de nombres entre 0 i $p-1$ (que al seu torn pot representar un text, un so, una imatge...) L'algorisme opera en blocs de k nombres, que interpretem com vectors de $\mathbb{F}_p^k$. Multiplicar per una matriu adequada de $\mathcal{M}_{n\times k}(\mathbb{F}_p)$ afegeix $t = n-k$ *valors de comprovació* a les dades, els quals permeten al codi de Reed-Solomon detectar (sense corregir) fins a t valors erronis, o localitzar i corregir fins a $\lfloor t/2 \rfloor$ valors erronis.

En el camp dels codis, els elements de $\mathbb{F}_p$ s'anomenen *símbols* del codi, els vectors de $\mathbb{F}_p^k$ són les *paraules*, els de $\mathbb{F}_p^n$ són *paraules en codi* i la successió de paraules s'anomena *missatge* (encara que la informació transmesa no té per què ser un text.)

En les aplicacions pràctiques s'agafen valors fixats de p, n i k escollits en funció de les característiques del mitjà (freqüència i magnitud esperades dels errors). En el treball, el programa ha d'admetre valors variables de p i k, prenent en tot cas $n = p-1$.

Notem que, com que volem poder corregir $\lfloor t/2 \rfloor$ errors, cal que aquesta quantitat sigui positiva, per tant caldrà que $2 \leq n-k = p-k-1$, i per tant el primer p no pot ser ni 2 ni 3.

Codificació

La matriu que usarem en la codificació, de $n = p-1$ files per $k < n$ columnes, és:

$$RS_{p,k} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & a & a^2 & \dots & a^{k-1} \\ 1 & a^2 & a^4 & \dots & a^{2(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a^{p-2} & a^{(p-2)2} & \dots & a^{(p-2)(k-1)} \end{pmatrix}$$

on $a \in \mathbb{F}_p$ és un element primitiu. Observeu que la fila i-èsima consta de les potències de a^{i-1}; com que a és un element primitiu, els elements $1, a, a^2, \dots, a^{p-2}$ són tots diferents, i per tant estem davant d'una matriu de Vandermonde (rectangular) de rang k.

1. Partim d'una successió de r valors $x_1, \dots, x_r \in \mathbb{F}_p$, a codificar, que imaginem dividit en *paraules*, o vectors, de longitud k:

$$u_1 = (x_1, \dots, x_k), u_2 = (x_{k+1}, \dots, x_{2k}), \dots, u_q = (x_{(q-1)k}, \dots, x_{qk}),$$

En el cas que r no sigui inicialment múltiple de k, el programa haurà d'afegir al final $x_i = 0$, augmentant la longitud r tant com sigui necessari, per tal de poder assumir que $r = qk$ és múltiple de k a partir d'aquí.

2. Una funció del programa codificarà cada paraula (vector) u_i multiplicant-la per la matriu de codificació; és a dir, es calculen els productes $v_1 = RS_{p,k} \cdot u_1, v_2 = RS_{p,k} \cdot u_2, \ldots, v_q = R_{p,k} \cdot u_q \in \mathbb{F}_p^n$ com

$$v_1 = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & a & a^2 & \ldots & a^{k-1} \\ 1 & a^2 & a^4 & \ldots & a^{2(k-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a^{p-2} & a^{(p-2)2} & \ldots & a^{(p-2)(k-1)} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_k \end{pmatrix}, \text{ etc., i finalment:}$$

3. La sortida del programa serà la successió $y_1, \ldots, y_{qn}$ de tots els coeficients dels vectors v_i, que és el "missatge" codificat.

Descodificació

Si no hi ha errors en el procés d'emmagatzematge (o transmissió) dels y_i, el procés de descodificació és l'invers al de codificació. A partir d'una successió de valors y_i agrupats en vectors $v_i \in \mathbb{F}_p^n$, es resol, per a cada i, el sistema $RS_{p,k} \cdot u_i = v_i$, i els coeficients dels vectors u_i són els x_i del "missatge" inicial. El programa ho pot fer usant les funcions d'àlgebra lineal descrites a la primera secció d'aquest document. Si algun dels sistemes d'equacions és incompatible, vol dir que hi ha algun error en el missatge codificat rebut.

Si s'ha produït algun error, per contra, és a dir s'ha rebut un vector $w_i \neq v_i$ (però no més de $t = n - k$ coeficients del vector v_i estan afectats), aleshores el sistema d'equacions $RS_{p,k} \cdot u_i = w_i$ serà incompatible. En aquest cas aplicarem l'algorisme de correcció d'errors de Berlekamp-Welch, que expliquem a continuació.

1. Partim del vector rebut $w = w_i = (z_1, \ldots, z_n) \in \mathbb{F}_p^n$ (amb errors). Sigui $e = \lfloor \frac{n-k}{2} \rfloor$ (que és el nombre màxim de valors z_i erronis que es poden corregir).

2. Considerem el sistema de n equacions lineals en $2e + k$ incògnites (anomenades $g_1, \ldots, g_{e+k}$, $f_1, \ldots, f_e$) que en forma matricial s'escriu com:

$$BW_{p,k}(w) \cdot \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{e+k} \\ f_1 \\ f_2 \\ \vdots \\ f_e \end{pmatrix} = \begin{pmatrix} z_1 \\ a^e z_2 \\ a^{2e} z_3 \\ \vdots \\ a^{(n-1)e} z_n \end{pmatrix} \tag{1}$$

on la matriu del sistema és $BW_{p,k}(w) =$

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 & -z_1 & -z_1 & -z_1 & \ldots & -z_1 \\ 1 & a & a^2 & \ldots & a^{e+k-1} & -z_2 & -az_2 & -a^2 z_2 & \ldots & -a^{e-1} z_2 \\ 1 & a^2 & a^4 & \ldots & a^{2(e+k-1)} & -z_3 & -a^2 z_3 & -a^4 z_3 & \ldots & -a^{2(e-1)} z_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 1 & a^{n-1} & a^{(n-1)2} & \ldots & a^{(n-1)(e+k-1)} & -z_n & -a^{n-1} z_n & -a^{(n-1)2} z_n & \ldots & -a^{(n-1)(e-1)} z_n \end{pmatrix}.$$

El programa haurà d'analitzar aquest sistema:

- Si és incompatible *(s'han produït més de e errors)* s'haurà de donar un missatge de "error no corregible".
- Si és compatible indeterminat *(s'han produït menys de e errors)* disminuïm en 1 el valor de e i tornem al punt 2.
- Si és compatible determinat, passem al pas següent.

3. A partir de la solució $(g_1, \ldots, g_{e+k}, f_1, \ldots f_e)$ construïm un segon sistema d'equacions lineals:

$$\begin{pmatrix} f_1 & 0 & 0 & \ldots & 0 & 0 \\ f_2 & f_1 & 0 & \ddots & 0 & 0 \\ f_3 & f_2 & f_1 & \ddots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & f_e & f_{e-1} & \ddots & \vdots & \vdots \\ 0 & 1 & f_e & \ddots & \vdots & \vdots \\ 0 & 0 & 1 & \ddots & \vdots & \vdots \\ \vdots & \vdots & 0 & \ddots & f_e & f_{e-1} \\ \vdots & \vdots & \vdots & \ddots & 1 & f_e \\ 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_{e+k} \end{pmatrix}, \tag{2}$$

en què cadascuna de les k columnes de la matriu del sistema consta dels coeficients $(f_1, f_2, \ldots, f_e, 1)$, començant amb f_1 en la diagonal principal, i zeros a la resta. Si aquest sistema és compatible, aleshores és determinat i el vector solució $(x_1, \ldots, x_k)$ és el vector buscat; si és incompatible, novament significa que s'han produït més de $\lfloor \frac{n-k}{2} \rfloor$ errors, i s'haurà de donar un missatge de "error no corregible".

Alguns detalls avançats

Justificació teòrica de l'algorisme

Si pensem el vector $(x_1, \ldots, x_k)$ com els coeficients d'un polinomi $P(t) = x_1 + x_2 t + \cdots + x_k t^{k-1} \in \mathbb{F}_p[t]$, aleshores el producte amb la matriu $RS_{p,k}$ no és altra cosa que el vector

$$RS_{p,k} \cdot (x_1, \ldots, x_k) = (y_1, \ldots, y_n) = (P(1), P(a), P(a^2), \ldots, P(a^{p-2}))$$

format pels valors $P(a^i)$ del polinomi P avaluat en totes les potències de a, és a dir, en tots els elements no nuls de $\mathbb{F}_p$. Aquesta interpretació ens permetrà justificar l'algorisme de Berlekamp-Welch de correcció d'errors.

Suposem que s'han produït e errors en la transmissió del vector, i que concretament en el vector rebut $(z_1, \ldots, z_n)$, els elements $z_{p_1}, z_{p_2}, \ldots, z_{p_e}$ són diferents de $y_{p_1}, y_{p_2}, \ldots, y_{p_e}$. Aleshores, el polinomi $F(t) = \prod_{j=1}^{e} (t - a^{p_j - 1})$ compleix que, per a tot $i = 1, \ldots p-1$, la igualtat

$$F(a^{i-1}) \cdot z_i = F(a^{i-1}) \cdot y_i = F(a^{i-1}) P(a^{i-1}) \tag{3}$$

és certa (això és òbviament cert si $z_i = y_i$, però també es cert quan $y_i \neq z_i$, ja que llavors $F(a^{i-1}) = 0$). Desafortunadament, el polinomi F és desconegut, com ho és també P. Per determinar-los, anomenem f_i i g_i els coeficients de F i de $F \cdot P$ respectivament, que són

incògnites; és a dir

$$F(t) = \prod_{j=1}^{e}(t - a^{p_j - 1}) = f_1 + f_2 t + \cdots + f_e t^{e-1} + t^e,$$
$$F(t) \cdot P(t) = (F \cdot P)(t) = g_1 + g_2 t + \cdots + g_{e+k} t^{e+k-1}.$$

Si ara reescrivim les equacions (3) per $i = 1, \ldots, p-1$ en termes dels coeficients f_i, g_i, obtenim exactament les equacions lineals donades matricialment per (1).
Un cop resolt aquest sistema, coneixem (els coeficients de) F i $F \cdot P$, i estem buscant els x_i, que són els coeficients de P; per tant, ens cal buscar un polinomi que multiplicat per $f_1 + f_2 t + \cdots + f_e t^{e-1} + t^e$ doni $g_1 + g_2 t + \cdots + g_{e+k} t^{e+k-1}$. Si plantegem les equacions corresponents per als coeficients de P obtenim les equacions lineals (2).
D'altra banda, és clar que si el nombre d'errors realment comesos és $e' < e$ les equacions (3) no determinen un únic polinomi F de grau e ja que només e' de les seves arrels estan determinades (pels $z_i \neq y_i$), i això es detecta quan, en resoldre el primer dels sistemes lineals, aquest resulta ser indeterminat.

3 Manipulació de fitxers

Un cop tingueu escrites les funcions per codificar i descodificar, i les hàgiu comprovat amb una funció `main` senzilla que demana les dades amb `scanf`, haureu d'escriure un programa principal més complex.

El programa final està pensat per executar-lo des de la consola. Les diferents operacions que pot fer el programa (les més importants, codificar i descodificar corregint errors) i els fitxers als quals cal aplicar-les se seleccionaran mitjançant arguments de la funció `main`, introduïts en el moment d'executar el programa des de la consola (vegeu la pràctica 5 del curs, secció 5.3). Els "missatges" i "missatges codificats" estaran en fitxers, donats en uns formats numèrics concrets, que distingirem per la seva extensió, i que ara descrivim. Per defecte (és a dir, si no s'indica el contrari) el programa usarà $p = 257$ i $k = 200$.

- Fitxers `.dat` Aquests fitxers contenen nombres (en base 10) separats per espais o tabulacions i salts de línia. Es poden llegir amb `fscanf` de forma similar als `scanf`, i s'hi pot escriure amb `fprintf`.
- Fitxers `.bin` Aquests fitxers cal llegir-los amb la instrucció `fread`, contenen directament vectors de nombres `unsigned int`. Per escriure-hi cal usar `fwrite`.
- Fitxers `.txt` Aquests fitxers es llegiran caràcter a caràcter, posant cada `char` (pensat com un nombre entre 0 i 255 que ocupa un byte) en un `int` diferent.

A la pràctica 7 del curs es treballa la manipulació de fitxers en C.

Arguments de `main`

Ús bàsic del programa El programa ha de servir per codificar i descodificar. Els dos paràmetres més importants seran la *instrucció* (`codifica` o `descodifica`) i el *nom del fitxer* al qual s'aplica la instrucció. Recordem que el programa executable, un cop compilat amb la instrucció

```
gcc -std=c11 -Wall -pedantic -o CognomNom CognomNom.c
```

tindrà el nom `CognomNom` (en Linux o MacOS) o `CognomNom.exe` (en Windows). Considerem la següent instrucció de terminal:

```
./CognomNom codifica missatge.dat
```

(en Windows, ometrem el `./`).

Si escrivim aquesta instrucció, el paràmetre `argv[1]` de `main` conté `"codifica"` i `argv[2]` conté el nom del fitxer a codificar. Caldrà que al mateix directori del programa executable hi hagi un fitxer anomenat `missatge.dat`, per codificar-lo.

En aquest cas el programa ha de llegir el fitxer, codificar-lo segons l'algorisme de Reed-Solomon i escriure el missatge codificat en un nou fitxer, amb el nom `missatge.RS.dat`.

Caldrà que el programa identifiqui l'extensió en el nom del fitxer donat com a segon argument. Segons si l'extensió del fitxer donat és `.dat`, `.bin` o `.txt`, aquest s'haurà de llegir d'una manera o una altra. El nou fitxer on s'escriurà el missatge codificat haurà de tenir el format `.dat` si el d'entrada el té, i el format `.bin` si el d'entrada és `.bin` o `.txt`. Li donarem el mateix nom però acabant en `RS.dat`, `RS.bin` o `RS.txt.bin` segons el cas.

Si el fitxer no existeix, o si el nom del fitxer no té una de les extensions admeses, el programa haurà de donar un missatge d'error (per la nansa `stderr`), i no el processarà.

Si en els valors llegits del fitxer n'hi ha de més grans o iguals que el nombre primer p, el programa s'haurà d'executar interpretant aquells nombres mòdul p, però mostrant un avís (*warning*) per `stderr`.

```
./CognomNom descodifica missatge.dat
```

Si escrivim aquesta instrucció, el programa ha de llegir el fitxer (que suposadament està codificat amb l'algorisme de Reed-Solomon), descodificar-lo segons l'algorisme de Berlekamp-Welch i escriure el missatge codificat en un nou fitxer, amb el nom `missatge.BW.dat`.

Com en el cas de la codificació, el programa haurà d'identificar l'extensió en el nom del fitxer donat com a segon argument, processar-lo d'acord amb el format, i escriure el fitxer descodificat amb el nom acabat `BW.dat`, `BW.bin` o `BW.txt` (aquest últim només si el fitxer d'entrada és `.txt.bin`). Igualment, haurà de donar un missatge d'error si l'extensió és desconeguda o un avís si els nombres no estan en el rang adequat.

Si tot va bé, executant successivament

```
./CognomNom codifica missatge.dat
./CognomNom descodifica missatge.RS.dat
```

s'haurà de crear un fitxer anomenat `missatge.RS.BW.dat` que hauria de ser idèntic al fitxer original `missatge.dat`.

Selecció de paràmetres En els programes de terminal és habitual poder indicar opcions amb el signe "-" en els paràmetres. Aquí seguirem aquest conveni si volem usar valors de p i k diferents als establerts per defecte. Vegem-ne exemples.

```
./CognomNom -p=401 codifica missatge.dat
```

Si el paràmetre `argv[1]` comença per un guió `-` cal interpretar-lo com una opció. Si aquesta té la forma `-p=` seguit d'un nombre, vol dir que caldrà usar aquell nombre primer per a la operació a realitzar (ja sigui `codifica` o `descodifica`).

El programa ha de comprovar que efectivament el nombre sigui primer, i que $p >> k+3$ (per poder corregir almenys un error); en aquest cas, 401 és primer i és més gran que el nombre $k+3=203$ que s'aplica per defecte. Igualment, si el fitxer fos de tipus `.txt`, caldrà que el primer p sigui més gran que 256. Si no es compleix alguna d'aquestes condicions, cal aturar el programa amb un missatge d'error.

També, com hem dit abans, si algun dels nombres en el fitxer és més gran que el primer escollit, cal donar un avís per `stderr`.

```
./CognomNom -p=401 -k=352 codifica missatge.dat
```

Si el paràmetre `argv[1]` comença per un guió `-` (és una opció), caldrà mirar també el paràmetre següent, i si també comença per un guió, interpretar-lo també com una opció.

Si trobem una opció de la forma `-k=` seguit d'un nombre, caldrà usar aquell valor de k com a mida dels vectors paraula. Cal que es compleixi $1 < k \leq p - 3$.

Hauria de ser possible introduir només el nombre primer, només el paràmetre k, o ambdós paràmetres, en l'ordre que es vulgui.

Fitxer de configuració Una altra manera d'establir els paràmetres d'un programa de terminal és mitjançant un fitxer de configuració. De fet, probablement si volem treballar amb una codificació de tipus $(p = 401, k = 352)$ no serà per a un únic fitxer sinó que ho mantindrem durant un temps.

El fitxer de configuració s'anomenarà `RS-BW.cfg` i constarà de dues línies, per exemple:

```
p=401
k=352
```

En iniciar-se el programa, aquest haurà de comprovar si existeix un fitxer amb aquest nom, i en cas que existeixi llegir els valors de p i k del fitxer; en cas contrari es mantindran els valors per defecte $p = 257, k = 200$. Si existeix el fitxer `RS-BW.cfg` i també s'introdueixen opcions `-p` o `-k`, les opcions introduïdes a la terminal han de prevaldre sobre les que estan establertes al fitxer de configuració.

El fitxer de configuració es pot crear amb un editor de text (en trobareu d'exemple al Campus Virtual). També es pot afegir l'operació de configurar a través del programa:

```
./CognomNom -p=401 -k=352 configura
```

En aquest cas no cal donar el nom de cap fitxer; el programa crearà (o sobreescriurà) un fitxer anomenat `RS-BW.cfg` amb les opcions donades.

Notes finals

Sobre l'avaluació del treball

El programa final a entregar fins el 7 de juny de 2022 tindrà una base formada per les funcions bàsiques (secció 1) completada amb l'algorisme de codificació i descodificació de la secció 2. És admissible entregar un programa que només consti d'aquestes parts (seccions 1 i 2), en el qual calgui introduir les dades per teclat. En aquest cas la nota màxima que es podrà assolir és un 6,9. Per aspirar a un Notable cal implementar la manipulació de fitxers tal com s'explica a la secció 3, com a mínim incloent la manipulació de fitxers `.dat` i les opcions `-p` i `-k`. Per aspirar a un Excellent cal implementar també la manipulació de fitxers `.bin` i `.txt`, i els fitxers de configuració.

Qualsevol programa que no compleixi algun dels requisits demanats o que no implementi correctament l'algorisme tindrà penalització. Podreu trobar al Campus Virtual fitxers d'entrada i de configuració de mostra per comprovar el funcionament del vostre programa.

Un programa que generi *warnings* en compilar-se amb la instrucció

```
gcc -std=c11 -Wall -pedantic -o CognomNom CognomNom.c
```

també tindrà penalització. Un programa que tingui errors sintàctics i no generi un executable serà suspès.

Canvis en l'algorisme

El programa entregat ha de fer la feina de codificació i descodificació de codis de Reed-Solomon seguint l'algorisme indicat. Inventar (o descobrir a internet) un algorisme diferent per fer la mateixa feina pot ser molt interessant, però no és el que es demana, i serà penalitzat.

Ara bé, sí que seria admissible *afegir* funcionalitats al programa més enllà de les demanades. Per exemple, es podria fer que el fitxer de configuració contingui també la matriu de codificació $RS_{p,k}$. O, donat que per descodificar (en absència d'errors) cal resoldre molts sistemes lineals amb la mateixa matriu del sistema, $RS_{p,k}$, podeu sistematitzar la solució d'aquests sistemes. Sabem de l'assignatura d'Àlgebra Lineal que el seguit de canvis de files del procés d'esglaonament de Gauss es pot codificar en una matriu invertible $P_{p,k}$. Aquesta compleix que, per a tota paraula-codi correcta $v = (y_1, \ldots, y_n)$ el resultat de multiplicar-la per $P_{p,k}$ és la paraula corresponent seguida de zeros:

$$P_{p,k} \cdot \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

mentre que per a una paraula-codi incorrecta w el vector producte té algun valor no nul en els últims $n-k$ coeficients. Podria ser interessant fer que el programa calculi aquesta matriu, i potser que resolgui els sistemes fent-la servir.