Codes correcteurs d'erreurs

N'oubliez pas de charger en mémoire la biblothèque d'algèbre linéaire.

```
> with(linalg):
```

Codes de Hamming

En dimension 7: le code de Hamming H_7

Par définition, le code de Hamming H_7 est un sous-espace vectoriel de dimension 4 de l'espace $Z/2Z^7$ Pour simplifier le décodage, il est commode de définir H_7 par un système de 3 équations indépendantes. Par construction, on choisit les 3 équations de sorte que si l'on met leurs coefficients en ligne dans une matrice 3x7 V7 les 7 vecteurs colonnes de V7 sont les écritures en base 2 des nombres de 1 à 7.

Il faut commencer par créer cette matrice avec Maple.
On donne la fonction b2 suivante transforme un nombre x< 2^n en un vecteur de longueur n dont les composantes donnent la décomposition en base 2 de x.

```
> b2:=proc(x,n)
  local r;
  r:=x mod 2;
  if n=1 then [r];
  else
  [op(b2(iquo(x,2),n-1)),r]
  fi;
  end:
```

Par exemple, pour avoir la décomposition de 6=4+2:

```
> b2(6,3);
```

$$[1, 1, 0]$$

1) Créez une matrice 3x7 dont les colonnes sont (dans l'ordre) les écritures en base 2 des nombres entre 1 et 7.

Solution

```
> V7:=augment(seq(b2(i,3),i=1..7));
```

$$V7 := \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

2)Vérifiez que les trois équations sont effectivement indépendantes.

Attention: Pour demander à Maple de travailler sur le corps (Z/2Z), vous pouvez utiliser les fonctions suivantes (Noter les majuscules):

```
> ?Gausselim
> ?Nullspace
```

Solution

```
> Gausselim(V7) mod 2;
```

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Le rang de la matrice échelonnée est bien 3.

Pour réduire un vecteur ou un matrice modulo 2, vous pouvez utiliser la fonction reduit suivante.

```
> reduit:=proc(M)
  map(modp,evalm(M),2)
  end:
```

Par définition, le code H_7 est le noyau de votre matrice V7.

3) Calculez une base de H_7 et explicitez les 16 mots du code H_7.

Solution

Une base du noyau s'obtient avec la fonction Nullspace.

```
> K:=Nullspace(V7) mod 2;
```

$$K := \{[1, 1, 0, 1, 0, 0, 1], [0, 1, 0, 1, 0, 1, 0], [1, 0, 0, 1, 1, 0, 0], [1, 1, 1, 0, 0, 0, 0]\}$$

On trouve 4 vecteurs, ce qui redonne que les 3 équations étaient indépendantes.

Les $16=2^4$ mots du code s'obtiennent en prenant toutes les combinaisons linéaires (dans Z/2Z) possibles entre ces 4 vecteurs.

```
> G:=augment(op(K));
```

$$G := \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

G est donc une matrice génératrice pour le code. En multipliant cette matrice par tous les vecteurs de $Z/2Z^4$

on obtient toutes les combinaisons linéaires possibles entre ces vecteurs.

```
> seq(evalm(G &*b2(i,4)),i=0..15);
```

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

Attention, il y a des 2 et des 3 car Maple fait en fait le calcul sur les entiers. Il faut réduire modulo 2 le résultat. D'où l'intér^et de la fonction reduit.

```
> seq(reduit(G &*b2(i,4)),i=0..15);
```

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

Par définition, le **poids** d'un mot (= un vecteur) est le nombre de composantes non nulles. Par exemple, le mot

```
> v:=vector([0,1,0,1,0,0,1]);
```

$$v := [0, 1, 0, 1, 0, 0, 1]$$

est de poids 3.

4) Ecrivez une fonction poids qui prend en argument un vecteur et calcule son poids.

```
> ?vectdim
```

Solution

```
> poids:=proc(v)
  local s,i;
  s:=0;
  for i from 1 to vectdim(v) do
   if v[i]<>0 then s:=s+1;
  fi; od;
  s;
  end:
> poids(v);
```

$$3$$

Par définition, la **distance** d'un code est le minimum des poids de tous les mots du code.

5) Calculez la distance du code H_7. En déduire que le code H_7 permet de corriger au plus une erreur.

```
> ?min
```

Solution

On calcule le minimum de la suite des poids des vecteurs non nuls.

```
> l:=seq(poids(reduit(G
  &*b2(i,4))),i=1..15);
```

$$l := 3, 3, 4, 3, 4, 4, 3, 4, 3, 3, 4, 3, 4, 4, 7$$

```
> min(l);
```

3

6) Si c est un mot du code H_7, que vaut le produit matriciel V7 &* c?
Si v est un mot du code qui a subi une erreur au cours de la transmission, on peut l'écrire v=c+e, où e est un mot de poids 1. Que vaut le produit matriciel V7 &* v?
En déduire un algorithme pour décoder un mot reçu ayant au plus une erreur et programmez le.

Solution

Par définition le code est le noyau de V7. Donc un vecteur c appartient au code ssi V7 &* c=0.
Un vecteur ayant eu une erreur s'écrit v=c+e avec c dans le code et e un vecteur de poids 1, c'est-à-dire avec une seule coordonée non nulle. Donc le produit V7 &* v= V7 &*(c+e)= V7 &* e.
On peut retrouver e à partir du résultat V7 &* v comme suit. Puisque e est de poids 1, toutes ses coordonnées sont nulles sauf la ième et le produit V7 &* v=V7&* e est égal à la ième colonne de V7. Mais par construction la ième colonne de V7 est l'écriture de i en base 2 et il n'y aplus qu'à changer le ième bit.

```
> decodeH7:=proc(m)
  local m1,b,n;
  m1:=copy(m);
  n:=reduit(V7 &* m);
  b:=n[1]*4+n[2]*2+n[3];
  if b<> 0 then
```

```
    m1[b]:=1-m1[b];
    fi;
    evalm(m1);
    end:
```

Testez votre fonction decodeH7 grâce à la fonction test suivante. Si vous obtenez toujours true c'est bon; si vous obtenez false c'est que votre fonction ne marche pas bien.

```
>
> test:=proc()
  local c,e,K,modif,i,dec;
  K:=augment(op(Nullspace(V7)mod 2));
  c:=map(modp,evalm(augment(op(Nullspa
  ce(V7)mod 2))&*
  randvector(4,entries=rand(0..1))),2)
  ;
  e:=rand(1..7)();
  modif:=copy(c);
  modif[e]:=1-modif[e];
  dec:=decodeH7(modif);
  for i from 1 to 7 do
  if (dec[i]<>c[i]) then RETURN(false)
  fi;
  od;
  true;
  end:
> test();
```

true

7) En faisant des opérations sur les lignes de V7, montrer

que les 16 mots du code H_7 sont obtenus à partir des 16 mots de longueur 4 en leur rajoutant 3 bits de parité à des sous-mots.

```
> ?addrow
>
```

En déduire une procédure encode qui associe à tout mot m de longueur 4 un mot du code H_7 commençant par m.

Solution

On fait des opérations élémentaires sur les lignes de V7, mais en commençant par la dernière colonne à droite.

```
> N:=reduit(addrow(V7,3,1,1));
```

$$N := \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

```
> N:=reduit(addrow(N,3,2,1));
```

$$N := \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

```
> N:=reduit(addrow(N,2,1,1));
```

$$N := \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

La matrice a l'air échelonnée (en partant de la droite), mais on continue pour éliminer tous les coefficients hors de la diagonale.

```
> N:=reduit(addrow(N,1,2,1));
```

$$N := \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

```
> N:=reduit(addrow(N,1,3,1));
```

$$N := \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Les équations du code sont donc équivalentes aux suivantes:

$x_5 = x_2 + x_3 + x_4$

$x_6 = x_1 + x_3 + x_4$

$x_7 = x_1 + x_2 + x_4$

Ce qui définit bien les 3 dernières lettres en fonctions des 4 premières.

```
> encode:=proc(m)
  local v,i;
  v:=vector(7);
  for i from 1 to 4 do
   v[i]:=m[i];
  od;
  v[5]:= modp(v[2]+v[3]+v[4],2);
  v[6]:= modp(v[1]+v[3]+v[4],2);
  v[7]:= modp(v[1]+v[2]+v[4],2);
  evalm(v);
  end:
> encode([1,0,1,0]);
```

$$[1, 0, 1, 0, 1, 0, 1]$$

Variante cyclique de H_7

La matrice V7bis suivante a 7 colonnes qui sont les écritures en base 2 des nombres de 1 à 7 mais dans un

ordre différent. Son noyau définit donc un code H_7bis qui est comme H_7 1-correcteur parfait.

```
> V7bis:=matrix([[1, 0, 0, 1, 0, 1, 1], [0, 1, 0, 1, 1, 1, 0], [0, 0, 1, 0, 1, 1, 1]]);
```

$$V7bis := \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Vérifier que ce code H_7bis est **cyclique**, c'est à dire qu'il contient un vecteur v=(v_1, v_2, v_3, v_4, v_5, v_6, v_7) si et seulement s'il contient son permuté circulaire (v_2, v_3, v_4, v_5, v_6, v_7, v_1).

Solution

On peut remarquer que la deuxième ligne de V7bis s'obtient en permutant circulairement la troisième et que de même la troisième s'obtient en permutant circulairement la première. Ceci suggère l'énoncé. Pour vérifier complètement l'énoncé, il faut montrer que l'espace vectoriel H_7bis est égal au sous-espace vectoriel E de Z/2Z^7 dont les équations sont les permutées circulaires de celles de H_7bis, c'es-à-dire du noyau de la matrice V7ter suivante:

```
> C:=col(V7bis,1);
```

$$C := [1, 0, 0]$$

```
> M:=delcols(V7bis,1..1);
```

$$M := \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

```
> V7ter:=augment(M,C);
```

$$V7ter := \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Le noyau H7ter de cette matrice est clairement toujours de dimension 4 (on a juste permuté les colonnes et donc pas changé le rang). Pour vérifier l'égalité de H7bis et H7ter, il suffit de vérifier une inclusion, ce que l'on peut faire en calculant la réduction de Gauss de la juxtaposée de V7bis et V7ter.

```
> V:=stack(V7bis,V7ter);
```

$$V := \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

```
> Gausselim(V) mod 2;
```

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Ouf, le rang est bien 3 d'où l'égalité H7bis=H7ter. Noter ici la différence entre le calcul sur Z/2Z ou sur Q:

```
> gausselim(V);
```

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -2 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Il y a une méthode plus conceptuelle de prévoir le résultat démontré. Les codes cycliques forment une classe importante de codes.

En dimension 8: le code de Hamming étendu H_8

Par définition, le code de Hamming étendu H_8 est le code obtenu à partir du code H_7 précédent en rajoutant aux mots de H_7 un bit de parité.

Ecrire une matrice V8 dont les lignes sont les équations définissant H_8. Donner une base de H_8 et calculez sa distance.
Combien d'erreurs peut-on corriger avec ce code?

Solution

Le code H8 est défini par une équation supplémentaire:

$x_8 = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7$

```
> v:=vector(8,1);
```

$$v := [1, 1, 1, 1, 1, 1, 1, 1]$$

```
> M:=augment(V7,[0,0,0]);
```

$$M := \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

```
> V8:=stack(M,v);
```

$$V8 := \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

```
> K8:=op(Nullspace(V8) mod 2);
```

$$K8 := [1, 1, 1, 0, 0, 0, 0, 1], [1, 1, 0, 1, 0, 0, 1, 0], [1, 0, 1, 1, 0, 1, 0, 0], [0, 1, 1, 1, 1, 0, 0, 0]$$

```
> G8:=augment(K8):
> seq(poids(reduit(G8 &*
  b2(i,4))),i=1..15);
```

$$4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 8$$

Le code étendu H[8] est donc de poids 4. Il permet de corriger une erreur et peut détecter deux erreurs.

En dimension 15: le code de Hamming H_{15}

En mimant la construction de H_7, construire un code H_{15} de dimension 11 et de longueur 15 et qui est 1-correcteur parfait.

Solution

Le code H_{15} est de dimension 11 et de longueur 15. Une matrice vérificatrice est donnée par une matrice 4x15 dont les colonnes sont tous les vecteurs non nuls de 2/2Z^4:

```
> V15:=augment(seq(b2(i,4),i=1..15))
  ;
```

$$V15 :=$$

$$\begin{bmatrix} 0,0,0,0,0,0,0,1,1,1,1,1,1,1,1 \\ 0,0,0,1,1,1,1,0,0,0,0,1,1,1,1 \\ 0,1,1,0,0,1,1,0,0,1,1,0,0,1,1 \\ 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1 \end{bmatrix}$$

```
> K15:=Nullspace(V15) mod 2;
```

$$K15 := \{[0,1,0,0,0,0,0,1,0,1,0,0,0,0,0],$$
$$[0,1,0,1,0,1,0,0,0,0,0,0,0,0,0],$$
$$[1,1,1,0,0,0,0,0,0,0,0,0,0,0,0],$$
$$[1,0,0,1,0,0,0,1,0,0,0,0,1,0,0],$$
$$[0,1,0,1,0,0,0,1,0,0,0,0,0,1,0],$$
$$[0,0,0,1,0,0,0,1,0,0,0,1,0,0,0],$$
$$[1,1,0,1,0,0,0,1,0,0,0,0,0,0,1],$$
$$[1,1,0,0,0,0,0,1,0,0,1,0,0,0,0],$$
$$[1,0,0,1,1,0,0,0,0,0,0,0,0,0,0],$$
$$[1,1,0,1,0,0,1,0,0,0,0,0,0,0,0],$$
$$[1,0,0,0,0,0,0,1,1,0,0,0,0,0,0]\}$$

```
> nops(K15);
```

$$11$$

Les 4 équations sont bien indépendantes. Reste à calculer la distance du code. On peut le faire naïvement en calculant le poids de tous ($=2^{11}-1$) les vecteurs non nuls, mais on sent bien que si cette méthode va donner le résultat ici, elle va rapidement atteindre ses limites en dimensions supérieures. (On a donné une méthode en cours pour montrer que la distance est >2.

```
> G15:=augment(op(K15)):
> min(seq(poids(reduit(G15 &*
```

```
b2(i,11))),i=1..2^11-1));
```

3

On a le résultat, mais on commence à sentir le calcul!

Une variante de H_{15} pour corriger 2 erreurs

Voici comment on peut utiliser une variante du code H_{15} pour corriger 2 erreurs.
Les matrices V1 et V2 ci-dessous ont pour colonnes les écritures en base 2 des nombres de 1 à 15, mais dans un ordre différent. Soit C le code linéaire noyau de la matrice V obtenue en superposant V1 et V2.
Vérifier que C permet de corriger 2 erreurs.

```
> V1:=matrix([[1, 0, 0, 0, 1, 0, 0, 1,
  1, 0, 1, 0, 1, 1, 1], [0, 1, 0, 0,
  1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0],
  [0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1,
  1, 1, 0], [0, 0, 0, 1, 0, 0, 1, 1,
  0, 1, 0, 1, 1, 1, 1]]);
```

$$V1 := \begin{bmatrix} 1,0,0,0,1,0,0,1,1,0,1,0,1,1,1 \\ 0,1,0,0,1,1,0,1,0,1,1,1,1,0,0 \\ 0,0,1,0,0,1,1,0,1,0,1,1,1,1,0 \\ 0,0,0,1,0,0,1,1,0,1,0,1,1,1,1 \end{bmatrix}$$

```
> V2:=matrix([[1, 0, 0, 0, 1, 1, 0, 0,
  0, 1, 1, 0, 0, 0, 1], [0, 0, 0, 1,
  1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1],
  [0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0,
  1, 0, 1], [0, 1, 1, 1, 1, 0, 1, 1,
  1, 1, 0, 1, 1, 1, 1]]);
```

$$V2 := \begin{bmatrix} 1,0,0,0,1,1,0,0,0,1,1,0,0,0,1 \\ 0,0,0,1,1,0,0,0,1,1,0,0,0,1,1 \\ 0,0,1,0,1,0,0,1,0,1,0,0,1,0,1 \\ 0,1,1,1,1,0,1,1,1,1,0,1,1,1,1 \end{bmatrix}$$

```
> V:=(stack(V1,V2));
```

$$V := \begin{bmatrix} 1,0,0,0,1,0,0,1,1,0,1,0,1,1,1 \\ 0,1,0,0,1,1,0,1,0,1,1,1,1,0,0 \\ 0,0,1,0,0,1,1,0,1,0,1,1,1,1,0 \\ 0,0,0,1,0,0,1,1,0,1,0,1,1,1,1 \\ 1,0,0,0,1,1,0,0,0,1,1,0,0,0,1 \\ 0,0,0,1,1,0,0,0,1,1,0,0,0,1,1 \\ 0,0,1,0,1,0,0,1,0,1,0,0,1,0,1 \\ 0,1,1,1,1,0,1,1,1,1,0,1,1,1,1 \end{bmatrix}$$

Programmez un algorithme naïf de décodage (on peut faire plus rapide avec un peu plus de mathématiques).

Solution

```
> K:=Nullspace(V) mod 2;
```

$$K := \{[1,1,0,0,1,1,1,0,0,1,0,0,0,0,0],$$
$$[1,0,0,0,1,0,1,1,1,0,0,0,0,0,0],$$
$$[1,0,1,1,1,0,0,0,0,0,0,1,0,0,0],$$
$$[0,1,1,0,0,1,1,1,0,0,1,0,0,0,0],$$
$$[0,0,1,0,1,1,1,0,0,0,0,0,0,1,0],$$
$$[0,1,0,1,1,1,0,0,0,0,0,0,1,0,0],$$
$$[0,0,0,1,0,1,1,1,0,0,0,0,0,0,1]\}$$

```
> nops(K);
```

$$7$$

```
> G:=augment(op(K)):
> min(seq(poids(reduit(G &*
  b2(i,7))),i=1..2^7-1));
```

5

Le code est de poids 5: il peut donc corriger deux erreurs.
Si l'on reçoit un mot v qui a au plus 2 erreurs, un algorithme de décodage naïf consiste à parcourir l'ensemble des 2^7 mots du code et pour chacun d'entre eux de calculer sa distance à v. On s'arrête dès que l'on a trouvé un mot du code à distance moins que 2.

Un décodage un peu meilleur, bien que toujours très naïf, consiste à calculer V &*m. On compare tout d'abord avec les colonnes de V : si c'est l'une d'elles, c'est qu'il y a eu une erreur à l'indice de cette colonne. Sinon, on continue et l'on compare avec le produit de V avec tous les vecteurs possibles de poids 2.
Il est commode pour le code de définir une fonction qui teste l'égalité de deux vecteurs.

```
> egal:=proc(v1,v2)
  local i,d;
  d:=vectdim(v1);
  if d<>vectdim(v2) then
  RETURN(false);
   else
   for i from 1 to d do
  if v1[i]<>v2[i] then
  RETURN(false);
  fi;
  od;
  fi;
  true;
  end:
```

```
> v:=vector(4,1);w:=vector(4,1);
```

$$v := [1, 1, 1, 1]$$

$$w := [1, 1, 1, 1]$$

```
> evalb(v=w); egal(v,w);
```

$$false$$

$$true$$

On introduit également pour nos tests une fonction p2(i,j) qui crée un vecteur de poids 2 (aux indices i et j).

```
> p2:=proc(i,j)
  local v;
  v:=vector(15,0);
  v[i]:=1;
  v[j]:=1;
  evalm(v);
  end:
> decode:=proc(m)
  local m1,s,i,j;
  m1:=copy(evalm(m));
  s:=reduit(V&*m);
  if egal(s, vector(8,0)) then
  RETURN(evalm(m1)) fi;
  for i from 1 to 15 do
   if egal(s,col(V,i)) then
  m1[i]:=1-m1[i];
  RETURN(evalm(m1));
  fi;
  od;
  for i from 1 to 14 do
   for j from i+1 to 15 do
```

```
    if egal(s,reduit(V&*p2(i,j))) then

    m1[i]:=1-m1[i]; m1[j]:=1-m1[j];
    RETURN(evalm(m1));
    fi;
    od;
    od;
     end:
```

Quelques tests. On tire un mot du code au hasard.

```
> c:=reduit(G &*
  b2(rand(1..2^7-1)(),7));
```

$$c := [1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0]$$

Pas d'erreur:

```
> decode(c);
```

$$[1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0]$$

Une erreur:

```
> e:=vector(15,0):
  e[rand(1..15)()]:=1:decode(reduit(
  c+e));
```

$$[1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0]$$

Deux erreurs:

```
> e:=p2(rand(1..15)(),rand(1..15)())
  :decode(reduit(c+e));
```

$$[1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0]$$

Codes tirés au hasard

Tirer quelques codes au hasard (pour n=15 et k=11 par exemple) et calculez leur distance.

Peut-on espérer tirer un bon code au hasard?

Solution

```
> V:=randmatrix(4,15,entries=rand(0..1));
```

$$V := \begin{bmatrix} 0,0,0,0,1,0,1,1,1,1,0,0,0,1,0 \\ 0,0,1,1,1,0,0,0,1,0,1,1,1,0,1 \\ 1,1,1,1,1,1,1,1,1,1,1,0,0,0,1 \\ 0,0,1,0,0,1,0,1,1,1,0,0,0,1,1 \end{bmatrix}$$

```
> G:=augment(op(Nullspace(V)mod 2));
```

$$G := \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

```
> coldim(G);
```

$$11$$

Les équations tirées au hasard sont indépendantes; logique.
Dans cet exemple, l'avant dernière colonne est de poids 2, donc le poids du code est 1 ou 2 et il ne

corrige pas d'erreur. On s'aperçoit ainsi qu'un code tiré au hasard a en général une petite distance (1 ou 2).

Le code de Golay G_{23}

On présente ici un autre code parfait, découvert par le physicien suisse Marcel Golay en 1949. On peut montrer que c'est le seul code correcteur binaire parfait permettant de corriger au moins deux erreurs. Ce code a eu une grande importance dans un domaine plus abstrait des mathématiques: la classification des groupes finis simples.

Le code de Golay G_{23} et par définition donné par les équations suivantes:

```
> V23:=matrix([[1, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
  1, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1],
  [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
  0, 0, 0, 1, 1, 1, 0, 1, 1, 0], [0, 0,
  0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
  0, 0, 1, 1, 1, 0, 1, 1], [0, 0, 0, 0,
  1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
  0, 0, 1, 1, 1, 1], [0, 0, 0, 0, 0, 1,
  0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1,
  0, 1, 0, 1], [0, 0, 0, 0, 0, 0, 1, 0,
  0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0,
  0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
```

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

V23 :=

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

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

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

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

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

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

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

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

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

[0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0

```
, 1 , 1 , 1 , 1 ]
[0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 1 , 1 , 1 , 1 , 1 , 0 , 0 , 1 , 0
, 0 , 1 , 0 , 1 ]
>
```

Calculer la dimension du code G_{23} ainsi que sa distance.
Combien d'erreurs peut-il corriger?
Vérifier que ce code est parfait.

Solution

Vu la forme de la matrice, les équations sont clairement indépendantes (V23 commence par un bloc identité). Le code est donc de dimension

```
> coldim(V23)-rowdim(V23);
```

$$12$$

Faute de mieux, on calcule encore sa distance "à la main".

```
> G:=augment(op(Nullspace(V23) mod
  2)):
> min(seq(poids(reduit(G&*b2(i,12))),i
  =1..2^12-1));
```

$$7$$

```
>
```

Le calcul est un peu long, mais on s'en sort. Ce code est de distance 7: il peut donc corriger 3 erreurs. Pour vérifier qu'il est parfait, il faut vérifier que le nombre d'éléments dans les 2^11 boules de rayon 3 représentent exactement l'ensemble des 2^23 mots.

```
> ifactor(1+23+binomial(23,2)+binomial
  (23,3));
```

$$(2)^{11}$$

```
> 2^23-2^12*(1+23+binomial(23,2)+binom
```

```
ial(23,3));
```

```
0
```

```
>
>
>
```