Codes à barres et Nombre d'or

Michel J. Dumontier

Introduction:

On trouve le nombre d'or et la suite de Fibonacci en architecture et dans les livres qui en parlent, or, les supermarchés, ne les appelle-t-on pas les "temples de la consommation"? on devrait alors y trouver le nombre d'or: justement, on le trouve et je l'ai débusqué! (après avoir résolu un problème posé par le même ami (celui du Quinto) polytechnicien) mais où se cache-t-il?...

On le trouve tout simplement lorsqu'on passe à la caisse: on découvre dans cet article qu'il y a toujours de l'or à la caisse bien que depuis bien longtemps on ne paie plus en Louis d'or!

Mais il est bien caché, en effet, il faut être mathématicien et informaticien pour le trouver, mais quand on est APListe, on va encore plus vite pour le trouver!

La raison d'être de son apparition dans les lecteurs de codes à barres résulte du fait qu'il faut éviter les erreurs malgré la vitesse de lecture qui peut varier (dynamiquement), que ce soit l'objet (bousculé) qui passe devant le lecteur ou le lecteur manipulé par (la main tremblante de) la caissière. Et le petit bip que l'on entend après une bonne lecture, c'est le nombre d'or qui déclare: oui, l'or peut entrer dans la caisse!

Avertissement:

Malheureusement, je n'ai pas d'APL sous Windows (pas assez d'or pour l'acheter et dont je n'ose annoncer le montant comparé à J sous Windows: 500FF seulement) et comme les textes sont faits sous Word et qu'il est trop tôt encore pour écrire des fonctions en J (le cours n'est pas assez avancé) je vais écrire les fonctions en utilisant les caractères spéciaux de Word, merci à l'heureux rédacteur qui possède un APL sous Windows de les transformer en caractères APL.

Remarque:

Il y a toutes sortes de codes à barres et ils font l'objet de normes (ex les normes NF Z 63-300-X) auxquelles sont associées toutes sortes de contraintes. On s'inspire ici de la norme NF Z 63-300-2 en supprimant la contrainte "3 parmi 9".

Définitions:

Un module est soit une barre soit un espace d'épaisseur 1.

Un élément est formé d'un ou de plusieurs modules (ici 1 ou 2 modules).

Dans le code considéré, comme il n'y a que 2 sortes d'éléments, les éléments barres sont codés 0 et 1 ainsi que les éléments espaces.

La contrainte "3 parmi 9" signifie 3 éléments larges parmi 9 éléments, mais nous allons ici étudier le cas général où il n'y a aucune contrainte sur le nombre d'éléments larges: il n'y a que 2 sortes de barres et 2 sortes d'espaces.

En matière de codes barre, à un espace succède une barre et inversement.

(il ne peut y avoir 2 espaces ou 2 barres consécutifs).

Représentation:

Pour la commodité du calcul, on notera une barre fine par 1, une barre épaisse par 1 1, un espace fin par 0, un espace épais par 0 0

Cela revient à étudier des codes binaires ne présentant pas plus de 2 chiffres binaires consécutifs égaux.

Problème:

Soit la sentinelle S= 1 0 1, on veut dénombrer les suites de chiffres binaires ayant la propriété suivante :

on considère les suites commençant par S, suivi de N chiffres binaires, suivi de S.

Pb : combien y-a-t-il de suites ne comportant pas plus de 2 chiffres binaires CONSECUTIFS identiques ?

En effet, si la vitesse de lecture peut varier au cours du temps, (opérateur humain), il faut qu'il y ait le plus de transitions possibles sinon il y aura erreur sur les barres ou espaces (très) épais. (On ne saura pas si l'élément lu (barre ou espace) fait par exemple 6 ou 7 modules).

Solution exhaustive:

Propriété : Si on applique deux fois la fonction `DIF` (bien connue et listée ci-dessous) à toute chaîne numérique S, il apparaîtra au moins un 0 si celle-ci contient au moins une fois 2 marches consécutives égales (par ex. 2 4 6 ou 7 4 1, ou 6 6 6).

(*Marches d'une suite* : vecteur résultat de `DIF` appliquée à la suite: les marches respectives de l'exemple ci-dessus sont: 2 2, ¯3 ¯3 et 0 0).

Pour les chaînes binaires, les marches ne peuvent être que plates et la règle devient : il apparaîtra un 0 si la chaîne binaire contient au moins une fois plus de 2 chiffres binaires consécutifs égaux : donc dans deux cas : 0 0 0 ou 1 1 1.

Nous pouvons écrire donc la fonction `COMPTE3B` qui calculera le nombre de suites possibles pour toute valeur de N.

```
  ∇ Z←DIF V
[1]   Z←(1↓V)-¯1↓V
    ∇
    ∇ Z←COMPTE3B N;V;M;I;S;T
[1]   V←Nρ2 ◇ M←2*N ◇ Z←0 ◇ I←¯1 ◇ S←1 0 1
[2]  BO:→(M=I←I+1)/0 ◇ T←S,(V[I]),S ◇ →(0∈DIF DIF T)/BO
[3]   Z←Z+1◇→BO
    ∇
```

Voici les valeurs du nombre de codes possibles pour les valeurs de N de 1 à 20

```
N:       1   2   3   4    5    6    7    8    9    10
codes    1   3   4   6   11   17   27   45   72   116

N:      11  12  13  14   15   16   17   18   19    20
codes 189 305 493 799 1292 2090 3383 5473 8855 14329
```

Comme le calcul est un peu long, cherchons une formule plus rapide :

Introduisons les fonctions `A`, `B`, `C` `et` `D` récursives suivantes :

```
     ∇ Z←A N
[1]    →(N=1)/B1
[2]    →(N=2)/B2
[3]    Z←(B N−1)+C N−1 ◇ →0
[4]    B1:Z←0 ◇ →0
[5]    B2:Z←1
    ∇
     ∇ Z←B N
[1]    →(N=1)/B1
[2]    →(N=2)/B2
[3]    Z←D N−1 ◇ →0
[4]    B1:Z←1 ◇ →0
[5]    B2:Z←0
    ∇
     ∇ Z←C N
[1]    →(N=1)/B1
```

```
[2]    →(N=2)/B2
[3]    Z←D N-2 ◇ →0
[4]    B1:Z←0 ◇ →0
[5]    B2:Z←1
    ∇
     ∇ Z←D N
[1]    →(N=1)/B1
[2]    →(N=2)/B2
[3]    Z←(A N-1)+(B N-1)+C N-1 ◇ →0
[4]    B1:Z←0 ◇ →0
[5]    B2:Z←1
    ∇
```

A N étant le nombre de configurations commençant par 1 0 0 1 à l'ordre N,

B N étant le nombre de configurations commençant par 1 0 1 0 à l'ordre N,

C N étant le nombre de configurations commençant par 1 0 1 1 à l'ordre N,

D N étant le nombre de configurations commençant par 1 1 0 0 ou 1 1 0 1 à l'ordre N.

Et voici la fonction qui compte :

```
    ▽ Z←CO N
[1]   Z←(A N)+(B N)+(C N)+D N
    ▽
```

Une fonction pour filou ! devinez qui ?

Soit TAB20 une table de dimension 5 20 dont la première ligne contient A (pour N=1 à 20);

la seconde ligne contient les valeurs de la fonction B pour les mêmes valeurs;

la troisième : idem mais avec la fonction C, la quatrième: idem avec la fonction D.

la dernière ligne contient la somme des lignes précédentes : elle contient donc les valeurs de CO N pour N=1 à 20

Voici cette fonction COD , plus rapide encore! (et pour cause) :

```
    ▽ Z←COD N;⎕IO
[1]   ⎕IO←1 ◇ →(N>20)/FIN
[2]   Z←TAB20[5;N] ◇ →0
[3]   FIN:Z←0 ◇'N EST TROP GRAND'
    ▽ Voici la table :
0 1 1 1  3  4  6 11 17  27  45  72 116 189  305  493  799 1292 2090  3383
1 0 1 2  2  4  7 10 17  28  44  72 117 188  305  494  798 1292 2091  3382
0 1 0 1  2  2  4  7 10  17  28  44  72 117  188  305  494  798 1292  2091
0 1 2 2  4  7 10 17 28  44  72 117 188 305  494  798 1292 2091 3382  5473
1 3 4 6 11 17 27 45 72 116 189 305 493 799 1292 2090 3383 5473 8855 14329
```

Et le nombre d'or dans tout cela ? Nous y arrivons, mais patience encore :

Définitions: Algorithme de Fibonacci : suite de nombres : un terme de la suite est égal à la somme des deux précédents.

La suite de Fibonacci : celle obtenue par l'algorithme de Fibonacci, les 2 premiers termes de la suite étant 1 1.

Si cela peut faire plaisir à quelqu'un, on obtient la même suite en partant de 0 1 ou 1 0.

Suite fibonaccienne: suite obtenue à partir de termes générateurs autres que 1 1.

Exemple : `FIBX` génère une suite fibonaccienne de longueur `N` à partir des nombres situés dans l'argument gauche `A`:

```
    ∇ Z←A FIBX N;⎕IO
[1]   ⎕IO←1 ⋄ Z←N⍴0 ⋄ Z[1]←A[1] ⋄ Z[2]←A[2] ⋄ I←3
[2]  BO:Z[I]←Z[I−1]+Z[I−2] ⋄ →(N≥I←I+1)/BO
    ∇
      1 1 FIBX 15
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
      1 3 FIBX 15
1 3 4 7 11 18 29 47 76 123 199 322 521 843 1364
```

Propriété : Toute suite fibonaccienne F est telle que F[N] / F[N-1] tend vers le nombre d'or lorsque N croît.

Suite quasi-fibonacienne : suite engendrée par l'algorithme de Fibonacci dans lequel on introduit une correction.

Suite quasi-fibonacienne à termes correcteurs constants : suite quasi-fibonacienne dans laquelle la correction est un vecteur constant.

En ce qui concerne notre code à barre, le nombre de codes est une suite quasi-fibonaccienne à termes constants, d'ordre 3, ces termes sont 0 ¯1 1 .

Voici cette fonction:

```
    ∇ Z←COFT N;M;⎕IO
[1]   ⎕IO←1 ⋄ Z←N⍴0 ⋄ Z[1]←1 ⋄ Z[2]←3 ⋄ I←3 ⋄ M←0 ¯1 1
[2]  BO:Z[I]←Z[I−1]+Z[I−2]+M[1+3|I] ⋄ →(N≥I←I+1)/BO
    ∇
      COFT 20
1 3 4 6 11 17 27 45 72 116 189 305 493 799 1292 2090 3383 5473 8855 14329
```

C'est la 5ème ligne de `TAB20`.

Propriété : alors que le rapport de 2 termes consécutifs d'une suite fibonacienne tend vers le nombre d'or, le rapport de 2 termes consécutifs d'une suite quasi-fibonacienne tend quasiment vers le nombre d'or.

Soit F une suite fibonaccienne et QF une suite quasi-fibonacienne; à partir d'une certaine valeur N>N0, F[N]/F[N-1] est plus proche du nombre d'or que QF[N]/QF[N-1].

Les 5 lignes de `TAB20` sont des suites quasi-fibonaciennes.

Et maintenant, le nombre d'or...

Quand vous disposez de dix chiffres binaires pouvant représenter 2*10=1024 codes différents et que vous n'en utilisez que 116 (voir `TAB20`), vous vous dites d'abord que vous n'en avez pas beaucoup et ensuite vous voudriez connaître le rapport entre ce que vous pourriez avoir et ce que vous avez! c'est là qu'intervient le nombre d'or.

[Digression : Rappel de la classe de troisième; partage d'un segment en moyenne extrême raison. On veut partager un segment en 2 bouts de telle sorte que le rapport du plus gros bout au plus petit bout soit égal au rapport du segment entier au gros bout: ce dernier étant le bout moyen (entre le petit bout et le segment entier)]. Ce rapport, vous l'avez deviné, c'est le nombre d'or Φ= (1+ √5)/2. C'est la racine positive de l'équation X2-X-1=0 que l'on trouve dans les deux définitions du nombre d'or Φ= 1.618033988749895.]

1) le nombre d'or Φ est tel que Φ2=Φ+1 (même équation que ci-dessus)

2) le nombre d'or Φ est tel que 1/Φ=Φ−1 (même équation que ci-dessus)

en considérant la suite fibonacienne engendrée par 1 3, les rapports des deux derniers termes pour 1 3 `FIBX 20` est 15127/9349=1.6180340143330838

plus proche du nombre d'or que pour la suite quasi-fibonacienne de notre code à barres au même ordre 20: `COFT 20` soit 14329/8855=1.618181818...

Et voici enfin où intervient le nombre d'or, le rapport du nombre de codes que nous pourrions avoir au nombre de codes disponibles dans notre code à barre serait exactement (2/Φ)*N si la suite était fibonaccienne, mais comme elle est quasi-Fibonacienne cette formule est approximative (et de moins en moins bonne lorsque N croît). (Comparer 1 3 `FIBX 50 et PHI⋆50`).

Ainsi, pour N=10, au lieu de 116 codes exactement, le calcul approximatif donne :
1024/116=8.82758 (2/Φ)*10=8.32575 .

Le nombre de codes disponibles se calule directement par Φ*N ;

exemple : pour 10, le nombre de codes disponibles est 116 ; or, Φ*10=122 . On pourra dire que cette formule donne une valeur optimiste du nombre de codes disponibles mais c'est une formule rapide.

Problème: Qui peut donner une formule exacte plus rapide que ma fonction exacte `COFT` ? ... bon amusement !

Conclusions : 1) la suite fibonacienne engendrée par les deux premiers termes 1 3 "colle" parfaitement à la formule ΦN à partir du 34ème terme.

2) En imposant une seule contrainte sur un code barre (celui des transitions non trop lentes), le nombre C(N) de codes disponibles pour N modules est très réduit et C(N) est exactement une suite quasi-fibonacienne dont les 2 premiers termes sont 1 3.