Jonglage

May 21, 2019

1 Table of Contents

1 Texte d'oral de modélisation - Agrégation Option Informatique
1.1 Préparation à l'agrégation - ENS de Rennes, 2018-19
1.2 À propos de ce document
1.3 Question de programmation
1.4 Réponse à l'exercice requis
1.4.1 Structures de données
1.4.2 Vérifier qu'un mot est bien une permutation de {0,...,1}{0,...,n1}
1.4.2.1 Fonction quadratique
1.4.2.2 Fonction linéaire
1.4.2.3 Comparaison expérimentale
1.4.3 Construire le mot
1.4.3.1 Transformer une chaîne de caractère "5241" en un mot
1.4.4 Test de la permutation
1.4.5 Test de la moyenne
1.5 Bonus ?
1.5.1 Complexité
1.5.2 Autres idées
1.6 Conclusion

2 Texte d'oral de modélisation - Agrégation Option Informatique

2.1 Préparation à l'agrégation - ENS de Rennes, 2018-19

- *Date* : 21 mai 2019
- *Auteur* : Lilian Besson
- *Texte*: Jonglage (public2012-D2.pdf)

2.2 À propos de ce document

- Ceci est une *proposition* de correction, partielle et probablement non-optimale, pour la partie implémentation d'un texte d'annale de l'agrégation de mathématiques, option informatique.
- Ce document est un notebook Jupyter, et est open-source sous Licence MIT sur GitHub, comme les autres solutions de textes de modélisation que j'ai écrite cette année.
- L'implémentation sera faite en OCaml, version 4+ :

```
In [1]: Sys.command "ocaml -version";;

The OCaml toplevel, version 4.04.2


Out[1]: - : int = 0
```

> Notez que certaines fonctions des modules usuels `List` et `Array` ne sont pas disponibles en OCaml 3.12. J'essaie autant que possible de ne pas les utiliser, ou alors de les redéfinir si je m'en sers.

Merci à Pierre et Clarence, élèves de la préparation à l'agrégation en 2018/2019, de m'avoir autorisé à utiliser une partie de leur implémentation pour cette (proposition de) correction.

2.3 Question de programmation

La question de programmation pour ce texte était donnée en page 5, et était assez courte :

> "Implanter dans l'un des langages de programmation autorisés l'algorithme de test de permutation."

2.4 Réponse à l'exercice requis

Dans l'ordre, on va :

1. définir une structure de donnée (mot = tableau d'entier),
2. écrire une fonction qui vérifie qu'un tel tableau est bien une permutation de $\{0, \dots, n-1\}$,
3. transformer une chaîne comme `"5340"` en un mot `[|5;4;3;0|]`,
4. calculer le mot ω depuis un tel mot, `[|1;0;2;3|]`,
5. vérifier que ce mot ω est une permutation (c'est le **test de permutation**).

En bonus, on implémente aussi le **test de moyenne**, et on illustre le bon fonctionnement de notre implémentation sur les exemples du texte.

2.4.1 Structures de données

On travaille avec des **mots** représentés comme des tableaux d'entiers.

```
In [4]: type mot = int array ;;

Out[4]: type mot = int array
```

2.4.2 Vérifier qu'un mot est bien une permutation de $\{0, \ldots, n-1\}$

On présente deux implémentations, la première n'est pas très réfléchie et se trouve être avoir une complexité temporelle en $\mathcal{O}(|mot|^2)$, tandis que la seconde est plus réfléchie et fonctionne en temps linéaire $\mathcal{O}(|mot|)$.

On va supposer que chaque valeur du mot d'entrée est bien dans $\{0, \ldots, n-1\}$, et on vérifie simplement que chaque valeur est présente une et une seule fois (si $n = |mot|$).

Fonction quadratique Pour chaque valeur $i \in \{0, \ldots, n-1\}$, on parcourt le mot à la recherche d'une valeur $m[j] = i$.

```
In [6]: let est_permut (m : mot) : bool =
          let p = Array.length m in
          let permut_bool = ref true in
          let i = ref 0 in
          while !i<p && !permut_bool do
            permut_bool := !permut_bool && ( Array.exists (fun x -> x = !i) m );
            incr i
          done;
          !permut_bool;;
        (* Complexité temporelle au pire en O(p^2) *)
        (* Complexité spatiale en O(p) *)

Out[6]: val est_permut : mot -> bool = <fun>


In [7]: est_permut [|3; 5; 1; 2; 4; 0|];; (* true ! *)
        est_permut [|3; 5; 1; 3; 4; 0|];; (* false ! il manque le 2, il y a deux fois le 3 *)

Out[7]: - : bool = true


Out[7]: - : bool = false
```

Fonction linéaire Au lieu de parcourir les valeurs à trouver et de les chercher, on peut parcourir les valeurs directement !

```
In [8]: let est_permut_lineaire (m : mot) : bool =
          let p = Array.length m in
          let tous_vus = Array.make p false in
          Array.iter (fun mi ->
            tous_vus.(mi) <- true
          ) m;
          Array.for_all (fun b -> b) tous_vus
        ;;
        (* Complexité temporelle au pire en O(p) *)
        (* Complexité spatiale en O(p) *)
```

```
Out[8]: val est_permut_lineaire : mot -> bool = <fun>
```

```
In [9]: est_permut_lineaire [|3; 5; 1; 2; 4; 0|];; (* true ! *)
        est_permut_lineaire [|3; 5; 1; 3; 4; 0|];; (* false ! il manque le 2, il y a deux fois
```

```
Out[9]: - : bool = true
```

```
Out[9]: - : bool = false
```

Comparaison expérimentale On va faire quelques mesures empiriques du temps de calcul, entre la fonction linéaire et la fonction quadratique. Cela illustre aussi l'utilisation de `Sys.time()` pour obtenir le temps système.

```
In [10]: let echange t i j =
           let tmp = t.(i) in
           t.(i) <- t.(j);
           t.(j) <- tmp;
         ;;
```

```
Out[10]: val echange : 'a array -> int -> int -> unit = <fun>
```

```
In [11]: Random.self_init ();;
```

```
Out[11]: - : unit = ()
```

On génère une permutation aléatoire facilement, en faisant plein d'échanges aléatoires (nombre et localisation aléatoires). **Attention, on essaie pas de generer selon la loi uniforme dans Σ_p,** c'est beaucoup plus difficile !

```
In [12]: let permutation_aleatoire p =
           let m = Array.init p (fun i -> i) in
           for _ = 1 to Random.int (10*p) do
             echange m (Random.int p) (Random.int p);
           done;
           m
         ;;
```

```
Out[12]: val permutation_aleatoire : int -> int array = <fun>
```

```
In [13]: permutation_aleatoire 10;;
```

```
Out[13]: - : int array = [|1; 3; 7; 9; 8; 4; 5; 0; 2; 6|]
```

```
In [13]: permutation_aleatoire 10;;

Out[13]: - : int array = [|1; 3; 7; 9; 8; 4; 5; 0; 2; 6|]
```


```
In [13]: permutation_aleatoire 10;;

Out[13]: - : int array = [|1; 3; 7; 9; 8; 4; 5; 0; 2; 6|]
```


Cette petite fonction permet de mesurer le temps machine mis pour calculer une fonction `f ()` :

```
In [14]: let timeit f () =
           let debut = Sys.time () in
           let res = f () in
           let fin = Sys.time () in
           Printf.printf "\nTemps pour calculer f() = %f seconde(s)." (fin -. debut);
           res
         ;;

Out[14]: val timeit : (unit -> 'a) -> unit -> 'a = <fun>
```

On va s'en servir avec cette fonction, qui test `nombre_test` fois la vérification `f_est_permut` sur une permutation aléatoire de taille `taille`.

```
In [17]: let test f_est_permut nombre_test taille () =
           Printf.printf "\nDébut de %i tests de taille %i..." nombre_test taille;
           flush_all ();
           for _ = 1 to nombre_test do
             let m = permutation_aleatoire taille in
             assert (f_est_permut m);
           done
         ;;

Out[17]: val test : (int array -> bool) -> int -> int -> unit -> unit = <fun>
```

- Pour la fonction qui devrait être linéaire, on observe bien un temps de calcul qui croît linéairement avec la taille de l'entrée ($p = 1000, 2000, 4000$ ici).

```
In [26]: timeit (test est_permut_lineaire 100 1000) ();;
         timeit (test est_permut_lineaire 100 2000) ();;
         timeit (test est_permut_lineaire 100 4000) ();;

         Printf.printf "\n\n\n";;
         flush_all();;
```

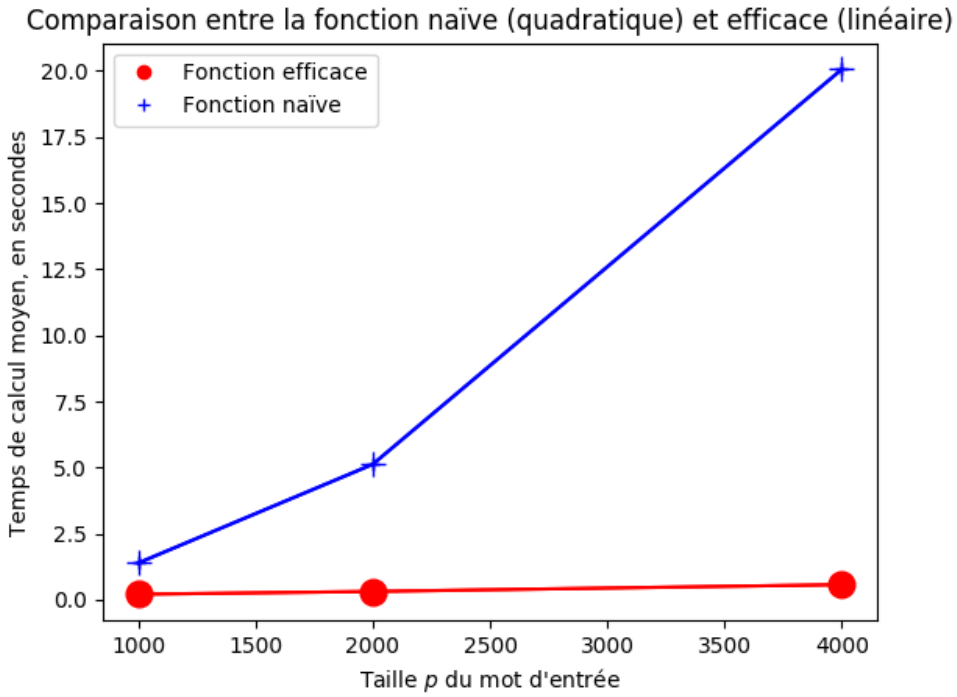
```
Out[26]: - : unit = ()


Début de 100 tests de taille 1000...
Temps pour calculer f() = 0.211866 seconde(s).


Out[26]: - : unit = ()


Début de 100 tests de taille 2000...
Temps pour calculer f() = 0.316790 seconde(s).


Out[26]: - : unit = ()


Out[26]: - : unit = ()


Début de 100 tests de taille 4000...
Temps pour calculer f() = 0.574821 seconde(s).




Out[26]: - : unit = ()
```

- Pour la fonction qui devrait être quadratique, on observe bien un temps de calcul qui croît quadratiquement avec la taille de l'entrée ($p = 1000, 2000, 4000$ ici).

```
In [27]: timeit (test est_permut 100 1000) ();;
         timeit (test est_permut 100 2000) ();;
         timeit (test est_permut 100 4000) ();;

         Printf.printf "\n\n\n";;
         flush_all();;



Out[27]: - : unit = ()


Début de 100 tests de taille 1000...
Temps pour calculer f() = 1.389953 seconde(s).


Out[27]: - : unit = ()
```

```
Début de 100 tests de taille 2000...
Temps pour calculer f() = 5.122529 seconde(s).


Out[27]: - : unit = ()


Out[27]: - : unit = ()


Début de 100 tests de taille 4000...
Temps pour calculer f() = 20.058526 seconde(s).



Out[27]: - : unit = ()
```

On peut afficher ces trois valeurs, juste pour mieux les visualiser :



2.4.3 Construire le mot ω

Cette fonction est toute simple, et est linéaire en temps et en mémoire.

```
In [37]: let trouve_omega (m : mot) : mot =
           let p = Array.length m in
           let omega = Array.make p 0 in
           for i=0 to (p-1) do
             omega.(i) <- (i + m.(i)) mod p
           done;
           omega;;

Out[37]: val trouve_omega : mot -> mot = <fun>


In [36]: trouve_omega [|4;4;1|];;
         trouve_omega [|5;3;4;0|];;

Out[36]: - : mot = [|1; 2; 0|]


Out[36]: - : mot = [|1; 0; 2; 3|]
```

Note : on peut être plus rapide avec une fonction `Array.init` au lieu de `Array.make` :

```
In [32]: Array.make;;
         Array.init;;

Out[32]: - : int -> 'a -> 'a array = <fun>


Out[32]: - : int -> (int -> 'a) -> 'a array = <fun>


In [38]: let trouve_omega (m : mot) : mot =
           let p = Array.length m in
           Array.init p (fun i -> ((i + m.(i)) mod p))
         ;;

Out[38]: val trouve_omega : mot -> mot = <fun>


In [ ]: trouve_omega [|4;4;1|];;
        trouve_omega [|5;3;4;0|];;

Out[ ]: - : mot = [|1; 2; 0|]


Out[ ]: - : mot = [|1; 0; 2; 3|]
```

Transformer une chaîne de caractère "5241" en un `mot` On a d'abord besoin de convertir un caractère comme `'5'` en un entier `5`. Cela peut se faire manuellement comme ça :

```
In [ ]: let entier (s : char) : int = match s with
          | '0'-> 0
          | '1'-> 1
          | '2'-> 2
          | '3'-> 3
          | '4'-> 4
          | '5'-> 5
          | '6'-> 6
          | '7'-> 7
          | '8'-> 8
          | '9'-> 9
        ;;

File "[41]", line 1, characters 30-162:
Warning 8: this pattern-matching is not exhaustive.
Here is an example of a case that is not matched:
'a'


Out[ ]: val entier : char -> int = <fun>


In [ ]: let entier (s : char) : int =
          (int_of_char s) - (int_of_char '0')
        ;;

Out[ ]: val entier : char -> int = <fun>
```

Pour la transformation de la chaîne en un mot, on peut le faire manuellement comme ça :

```
In [ ]: let transfo_mot (m : string) : mot =
          let p = String.length m in
          let mot_tableau = Array.make p 0 in
          for i = 0 to (p - 1) do
            mot_tableau.(i) <- entier (m.[i])
          done;
          mot_tableau;;
        (* Complexité temporelle et spatiale en O(p) *)

Out[ ]: val transfo_mot : string -> mot = <fun>


In [ ]: transfo_mot "5241";;

Out[ ]: - : mot = [|5; 2; 4; 1|]
```

Mais on peut aussi accélérer un peu :

```
In [ ]: Array.init

Out[ ]: - : int -> (int -> 'a) -> 'a array = <fun>


In [ ]: let transfo_mot (m : string) : mot =
          Array.init (String.length m) (fun i -> (entier m.[i]))
        (* Complexité temporelle et spatiale en O(p) *)

Out[ ]: val transfo_mot : string -> mot = <fun>


In [47]: transfo_mot "5241";;

Out[47]: - : mot = [|5; 2; 4; 1|]
```

2.4.4 Test de la permutation

```
In [48]: let test_permut (m : string) : bool =
           est_permut (trouve_omega (transfo_mot m));;
         (* Complexité temporelle en O(p) *)

Out[48]: val test_permut : string -> bool = <fun>


In [110]: let test_permut_mot (m : mot) : bool =
            est_permut (trouve_omega m);;
          (* Complexité temporelle en O(p) *)

Out[110]: val test_permut_mot : mot -> bool = <fun>
```

Quelques exemples :

```
In [49]: test_permut "433";; (* pas jonglable *)

Out[49]: - : bool = false


In [52]: test_permut "432";; (* pas jonglable *)

Out[52]: - : bool = false


In [53]: test_permut "441";; (* jonglable *)

Out[53]: - : bool = true
```

```
In [50]: test_permut "5241";; (* jonglable *)

Out[50]: - : bool = true


In [51]: test_permut "9340";; (* jonglable *)

Out[51]: - : bool = true
```

2.4.5 Test de la moyenne

Cet autre test n'est pas une condition nécessaire et suffisante, mais il est rapide à implémenter :

```
In [67]: let test_moyenne_mot (m : mot) (b : int) : bool =
           let p = Array.length m in
           let s = ref 0 in
           for i = 0 to (p - 1) do
             s:= !s + m.(i)
           done;
           !s / p = b (* on teste l'égalité *)
         ;;

Out[67]: val test_moyenne_mot : mot -> int -> bool = <fun>


In [68]: test_moyenne_mot [|5;4;3;0|] 3;;

Out[68]: - : bool = true
```

On va finir par quelques exemples :

```
In [69]: let test_moyenne (m : string) (b : int) : bool =
           test_moyenne_mot (transfo_mot m) b;;

Out[69]: val test_moyenne : string -> int -> bool = <fun>


In [70]: test_moyenne "433" 3;;

Out[70]: - : bool = true


In [71]: test_moyenne "443" 3;;

Out[71]: - : bool = true


In [72]: test_moyenne "432" 3;;
```

```
Out[72]: - : bool = true
```

```
In [73]: test_moyenne "5430" 3;;
```

```
Out[73]: - : bool = true
```

2.5 Bonus ?

2.5.1 Complexité

- Sauf la fonction volontairement écrite pour être quadratique, toutes les fonctions suivantes sont linéaires, c'est à dire qu'elles ont une complexité en temps *et en mémoire* bornée par $\mathcal{O}(|mot|)$.

2.5.2 Autres idées

La fonction suivante donne la liste des mots de taille `p` commençant par `debut` (dont la somme des coefficients vaut `somme`) concaténée avec `acc`, en temps $\mathcal{O}(m^p)$.

```
In [114]: let rec partition_aux balles p debut somme acc =
            let m = balles * p in
            let manque = Array.fold_left (fun c x -> if x = (-1) then c + 1 else c) 0 debut in
            if manque = 1 then
              let t = Array.copy debut in
              t.(p - 1) <- m - somme;
              t :: acc
            else
              let nacc = ref acc in
              for k = m - somme downto 0 do
                let ndebut = Array.copy debut in
                ndebut.(p - manque) <- k;
                nacc := partition_aux balles p ndebut (somme + k) !nacc
              done;
              !nacc
          ;;
```

```
Out[114]: val partition_aux :
            int -> int -> int array -> int -> int array list -> int array list = <fun>
```

```
In [115]: let partition balles p =
            let debut = Array.make p (-1) in
            partition_aux balles p debut 0 []
          ;;
```

```
Out[115]: val partition : int -> int -> int array list = <fun>
```

```
In [116]: let bp balles p = List.filter test_permut_mot (partition balles p);;

Out[116]: val bp : int -> int -> mot list = <fun>
```

Par exemples :

```
In [117]: partition 3 3;;

Out[117]: - : int array list =
          [[|0; 0; 9|]; [|0; 1; 8|]; [|0; 2; 7|]; [|0; 3; 6|]; [|0; 4; 5|];
           [|0; 5; 4|]; [|0; 6; 3|]; [|0; 7; 2|]; [|0; 8; 1|]; [|0; 9; 0|];
           [|1; 0; 8|]; [|1; 1; 7|]; [|1; 2; 6|]; [|1; 3; 5|]; [|1; 4; 4|];
           [|1; 5; 3|]; [|1; 6; 2|]; [|1; 7; 1|]; [|1; 8; 0|]; [|2; 0; 7|];
           [|2; 1; 6|]; [|2; 2; 5|]; [|2; 3; 4|]; [|2; 4; 3|]; [|2; 5; 2|];
           [|2; 6; 1|]; [|2; 7; 0|]; [|3; 0; 6|]; [|3; 1; 5|]; [|3; 2; 4|];
           [|3; 3; 3|]; [|3; 4; 2|]; [|3; 5; 1|]; [|3; 6; 0|]; [|4; 0; 5|];
           [|4; 1; 4|]; [|4; 2; 3|]; [|4; 3; 2|]; [|4; 4; 1|]; [|4; 5; 0|];
           [|5; 0; 4|]; [|5; 1; 3|]; [|5; 2; 2|]; [|5; 3; 1|]; [|5; 4; 0|];
           [|6; 0; 3|]; [|6; 1; 2|]; [|6; 2; 1|]; [|6; 3; 0|]; [|7; 0; 2|];
           [|7; 1; 1|]; [|7; 2; 0|]; [|8; 0; 1|]; [|8; 1; 0|]; [|9; 0; 0|]]


In [118]: bp 3 3;;

Out[118]: - : mot list =
          [[|0; 0; 9|]; [|0; 1; 8|]; [|0; 3; 6|]; [|0; 4; 5|]; [|0; 6; 3|];
           [|0; 7; 2|]; [|0; 9; 0|]; [|1; 1; 7|]; [|1; 2; 6|]; [|1; 4; 4|];
           [|1; 5; 3|]; [|1; 7; 1|]; [|1; 8; 0|]; [|2; 0; 7|]; [|2; 2; 5|];
           [|2; 3; 4|]; [|2; 5; 2|]; [|2; 6; 1|]; [|3; 0; 6|]; [|3; 1; 5|];
           [|3; 3; 3|]; [|3; 4; 2|]; [|3; 6; 0|]; [|4; 1; 4|]; [|4; 2; 3|];
           [|4; 4; 1|]; [|4; 5; 0|]; [|5; 0; 4|]; [|5; 2; 2|]; [|5; 3; 1|];
           [|6; 0; 3|]; [|6; 1; 2|]; [|6; 3; 0|]; [|7; 1; 1|]; [|7; 2; 0|];
           [|8; 0; 1|]; [|9; 0; 0|]]
```

On voit que l'on retrouve les mots donnés en exemples dans le texte, 441, 423, 333 par exemple :

```
In [124]: List.mem [|4; 4; 1|] (bp 3 3);;
          List.mem [|4; 2; 3|] (bp 3 3);;
          List.mem [|3; 3; 3|] (bp 3 3);;

Out[124]: - : bool = true


Out[124]: - : bool = true


Out[124]: - : bool = true
```

Et le mot 433 n'est pas dans la liste calculée :

```
In [125]: List.mem [|4; 3; 3|] (bp 3 3);;

Out[125]: - : bool = false
```

Pour $m = 4$, on commence à avoir beaucoup plus de réponses :

```
In [121]: partition 4 4;;

          List.length (partition 4 4);;

Out[121]: - : int array list =
          [[|0; 0; 0; 16|]; [|0; 0; 1; 15|]; [|0; 0; 2; 14|]; [|0; 0; 3; 13|];
           [|0; 0; 4; 12|]; [|0; 0; 5; 11|]; [|0; 0; 6; 10|]; [|0; 0; 7; 9|];
           [|0; 0; 8; 8|]; [|0; 0; 9; 7|]; [|0; 0; 10; 6|]; [|0; 0; 11; 5|];
           [|0; 0; 12; 4|]; [|0; 0; 13; 3|]; [|0; 0; 14; 2|]; [|0; 0; 15; 1|];
           [|0; 0; 16; 0|]; [|0; 1; 0; 15|]; [|0; 1; 1; 14|]; [|0; 1; 2; 13|];
           [|0; 1; 3; 12|]; [|0; 1; 4; 11|]; [|0; 1; 5; 10|]; [|0; 1; 6; 9|];
           [|0; 1; 7; 8|]; [|0; 1; 8; 7|]; [|0; 1; 9; 6|]; [|0; 1; 10; 5|];
           [|0; 1; 11; 4|]; [|0; 1; 12; 3|]; [|0; 1; 13; 2|]; [|0; 1; 14; 1|];
           [|0; 1; 15; 0|]; [|0; 2; 0; 14|]; [|0; 2; 1; 13|]; [|0; 2; 2; 12|];
           [|0; 2; 3; 11|]; [|0; 2; 4; 10|]; [|0; 2; 5; 9|]; [|0; 2; 6; 8|];
           [|0; 2; 7; 7|]; [|0; 2; 8; 6|]; [|0; 2; 9; 5|]; [|0; 2; 10; 4|];
           [|0; 2; 11; 3|]; [|0; 2; 12; 2|]; [|0; 2; 13; 1|]; [|0; 2; 14; 0|];
           [|0; 3; 0; 13|]; [|0; 3; 1; 12|]; [|0; 3; 2; 11|]; [|0; 3; 3; 10|];
           [|0; 3; 4; 9|]; [|0; 3; 5; 8|]; [|0; 3; 6; 7|]; [|0; 3; 7; 6|];
           [|0; 3; 8; 5|]; [|0; 3; 9; 4|]; [|0; 3; 10; 3|]; [|0; 3; 11; ...|]; ...]


Out[121]: - : int = 969


In [123]: bp 4 4;;

          List.length (bp 4 4);;

Out[123]: - : mot list =
          [[|0; 0; 0; 16|]; [|0; 0; 1; 15|]; [|0; 0; 4; 12|]; [|0; 0; 5; 11|];
           [|0; 0; 8; 8|]; [|0; 0; 9; 7|]; [|0; 0; 12; 4|]; [|0; 0; 13; 3|];
           [|0; 0; 16; 0|]; [|0; 1; 1; 14|]; [|0; 1; 3; 12|]; [|0; 1; 5; 10|];
           [|0; 1; 7; 8|]; [|0; 1; 9; 6|]; [|0; 1; 11; 4|]; [|0; 1; 13; 2|];
           [|0; 1; 15; 0|]; [|0; 2; 0; 14|]; [|0; 2; 3; 11|]; [|0; 2; 4; 10|];
           [|0; 2; 7; 7|]; [|0; 2; 8; 6|]; [|0; 2; 11; 3|]; [|0; 2; 12; 2|];
           [|0; 4; 0; 12|]; [|0; 4; 1; 11|]; [|0; 4; 4; 8|]; [|0; 4; 5; 7|];
           [|0; 4; 8; 4|]; [|0; 4; 9; 3|]; [|0; 4; 12; 0|]; [|0; 5; 1; 10|];
           [|0; 5; 3; 8|]; [|0; 5; 5; 6|]; [|0; 5; 7; 4|]; [|0; 5; 9; 2|];
           [|0; 5; 11; 0|]; [|0; 6; 0; 10|]; [|0; 6; 3; 7|]; [|0; 6; 4; 6|];
```

```
[|0; 6; 7; 3|]; [|0; 6; 8; 2|]; [|0; 8; 0; 8|]; [|0; 8; 1; 7|];
[|0; 8; 4; 4|]; [|0; 8; 5; 3|]; [|0; 8; 8; 0|]; [|0; 9; 1; 6|];
[|0; 9; 3; 4|]; [|0; 9; 5; ...|]; ...]
```

```
Out[123]: - : int = 369
```

2.6 Conclusion

Voilà pour les deux questions obligatoires de programmation :

- on a décomposé le problème en sous-fonctions,
- on a essayé d'être fainéant, en réutilisant les sous-fonctions,
- on a fait des exemples et *on les garde* dans ce qu'on présente au jury,
- on a testé la fonction exigée sur un exemple venant du texte,
- et on a essayé d'en faire un peu plus (au début).

  Bien-sûr, ce petit notebook ne se prétend pas être une solution optimale, ni exhaustive.