Tissu_cellulaire

May 21, 2019

1 Table of Contents

2 Texte d'oral de modélisation - Agrégation Option Informatique

2.1 Préparation à l'agrégation - ENS de Rennes, 2016-17

- *Date* : 22 mai 2017
- *Auteur* : Lilian Besson
- *Texte*: Annale 2010, "Tissus cellulaires" (public2010-D2)

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
```

2.3 Question de programmation

La question de programmation pour ce texte était donnée au tout début, et occupe toute la page 2 :

2.3.1 Modélisation

On veut simuler un phénomène de contamination. L'état d'une cellule est représenté par un entier compris entre 0 et K ($K \geq 1$ est un paramètre fixé).

Une cellule est normalement saine, non contaminée, dans l'état 0 et reste dans cet état tant qu'elle n'est pas infectée. Tous les autres états représentent différents stades d'infection. Une cellule infectée commence dans l'état 1 et à chaque itération passe dans l'état suivant (c.-à-d. 2 puis 3 puis...). Quand elle atteint l'état K, elle a remporté sa lutte contre l'infection et retourne à l'état 0, saine et non infectée à l'itération suivante.

Une cellule infectée n'est pas tout de suite contagieuse (c.-à-d. capable d'infecter ses voisines). Une cellule devient infectieuse à partir du stade I ($I \leq K$, I est un autre paramètre fixé) et le reste jusqu'à sa guérison (retour à l'état 0). Durant cette période, elle peut contaminer chacune de ses deux voisines (les plus proches à droite et à gauche). Si celles-ci ne sont pas déjà infectées, elles le deviennent automatiquement, au stade 1; si elles sont déjà infectées, elles le restent.

Les cellules ne bougent pas et on les suppose rangées les unes à côté des autres sur une ligne. Elles sont toutes mises à jour en même temps, de manière synchrone. Pour simplifier, on suppose que les première et dernière cellules restent toujours dans le même état 0.

Tout ceci est schématisé sur la Fig. 1 où l'on voit en haut une configuration et en bas la configuration suivante. Chaque cellule est mise à jour en fonction de ses deux plus proches voisines, sauf pour les cellules des deux extrémités dont l'état ne change pas. Après la configuration $(0, 0, 3, 0, 0, 1, 5, \ldots, 0)$, le système passe donc à la configuration $(0, 1, 4, 1, 0, 2, 0, \ldots, 0)$ pour $K = 5$ et $I = 2$. Dans cette illustration, l'état de la troisième cellule (3) est en gras pour indiquer pour quelles cellules il intervient durant la mise à jour.

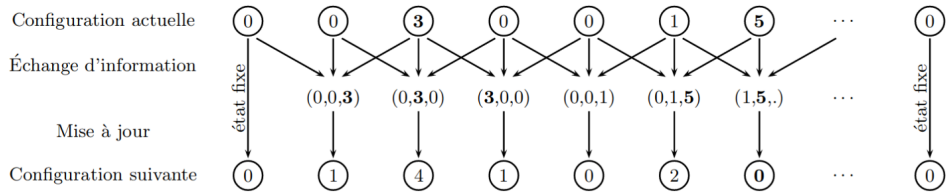

FIG. 1. Modélisation des influences entre cellules ($K = 5$ et $I = 2$).

Figure 1 : tissu cellulaire en ligne, K=5 et I=2

2.3.2 Exercice

Écrire un programme permettant d'afficher la progression de la contamination sur une vingtaine d'itérations, par exemple pour les valeurs ($K = 5$ et $I = 2$) en partant d'une seule cellule infectée sur une ligne d'une vingtaine de cellules saines.

Préciser la complexité en temps et en espace de la simulation.

2.4 Solution

On va essayer d'être rapide et de faire simple, aussi $K = 5$ et $I = 2$ seront **fixés** et constants dans tout le code suivant.

Les constantes K et I de la simulation sont fixées :

- $K = 5$ fixe le nombre d'états différents dans lequel peut être une cellule,
- $I = 2$ fixe le seuil à partir duquel une cellule devient contagieuse

```
In [2]: let k = 5;;

        let i = 2;;

        assert (i <= k);;
```

```
Out[2]: val k : int = 5
```

```
Out[2]: val i : int = 2
```

```
Out[2]: - : unit = ()
```

2.4.1 Pour une cellule

L'état d'une cellule est un entier :

```
In [3]: type etatcellule = int;;
```

```
Out[3]: type etatcellule = int
```

Voici quelques fonctions “triviales” pour traduire les propriétés décrites par le texte :

```
In [4]: (** Une cellule est saine si son état est [0], infectée sinon. *)
        let est_saine (etat : etatcellule) : bool =
            etat = 0
        ;;

Out[4]: val est_saine : etatcellule -> bool = <fun>


In [5]: let grossir (etat : etatcellule) : etatcellule =
            if etat >= k then 0 else (etat + 1)
        ;;

Out[5]: val grossir : etatcellule -> etatcellule = <fun>


In [6]: let est_contagieuse (etat : etatcellule) : bool =
            etat >= i
        ;;

Out[6]: val est_contagieuse : etatcellule -> bool = <fun>


In [7]: let infecte (etat : etatcellule) : etatcellule =
            if etat = 0 then 1 else etat
        ;;

Out[7]: val infecte : etatcellule -> etatcellule = <fun>
```

2.4.2 Pour une ligne de cellule (*un tissu*)

Si on voulait représenter des tissus en plusieurs dimensions (2, 3), ici on devrait utiliser `etatcellule array array` (ou `etatcellule array array array`), mais en 1D, `etatcellule array` suffit.

```
In [8]: type tissu = etatcellule array;;

Out[8]: type tissu = etatcellule array
```

Pour créer un nouvel organse, constitué uniquement de cellules saines :

```
In [9]: let nouveau_tissu (length : int) : tissu =
            Array.make length (0 : etatcellule)
        ;;
```

```
Out[9]: val nouveau_tissu : int -> tissu = <fun>
```

Par exemple, un “muscle”, long de 20 cellules, toutes saines :

```
In [10]: let muscle1 : tissu = nouveau_tissu 20;;

Out[10]: val muscle1 : tissu =
           [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0|]
```

Cette fonction sera un raccourci utile pour infecter une cellule choisie :

```
In [11]: let infecte_cible (org : tissu) (indice : int) : unit =
             org.(indice) <- (infecte org.(indice))
         ;;

Out[11]: val infecte_cible : tissu -> int -> unit = <fun>
```

Un second muscle, long de 21 cellules, avec une seule cellule malade en indice 11 :

```
In [12]: let muscle2 : tissu =
             let m = nouveau_tissu 21 in begin
                 infecte_cible m 11;
                 m
             end
         ;;

Out[12]: val muscle2 : tissu =
           [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0; 0|]
```

- Toutes ces fonctions s’exécutent en temps constant $\mathcal{O}(1)$.

2.4.3 Évolution d’une cellule qui n’est pas au bord en fonction de l’état précédent de ces deux voisins et de son état.

On représente une cellule avec ses deux voisins :

```
In [13]: type lcor = { l : etatcellule;  co : etatcellule;  r : etatcellule;  };;

Out[13]: type lcor = { l : etatcellule; co : etatcellule; r : etatcellule; }
```

On peut ensuite donner l’état de la cellule courante (précédemment dans l’état `co`) en fonction de l’état de sa voisine de gauche `l` et de droite `r` :

```
In [14]: let nouvel_etat_cellule ({l;  co;  r} : lcor) () : etatcellule =
             if co = k then
                 (* La cellule guérit, ce qu'on autoriserait plus dans le second modèle. *)
                 0
             else begin
                 if (est_saine co) then
                     (* Si elle est saine, elle peut devenir infectée : *)
                     if ( (est_contagieuse l) || (est_contagieuse r) )
                     then
                         (* Devient infectée. *)
                         1
                     else
                         co
                 else
                     (* Si elle est déjà infectée, elle grandit toute seule. *)
                     grossir co
             end
         ;;

Out[14]: val nouvel_etat_cellule : lcor -> unit -> etatcellule = <fun>
```

- Cette fonction s'exécute en temps constant $\mathcal{O}(1)$.

Construction du sous-tableau des triplets `lcor`, en ignorant la première et dernière cellule :

```
In [15]: let cree_lcor (org : tissu) : lcor array =
             let n = Array.length org in
             assert (n >= 2);
             Array.init (n - 2) (fun i ->
                 { l = org.(i);  co = org.(i + 1);  r = org.(i + 2) }
             )
         ;;

Out[15]: val cree_lcor : tissu -> lcor array = <fun>
```

- Cette fonction s'exécute en temps linéaire $\mathcal{O}(n)$.

Testons sur les deux exemples précédents :

```
In [16]: let lcor1 = cree_lcor muscle1;;
         let lcor2 = cree_lcor muscle2;;

Out[16]: val lcor1 : lcor array =
           [|{l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}|]
```

```
Out[16]: val lcor2 : lcor array =
           [|{l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 1}; {l = 0; co = 1; r = 0}; {l = 1; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0}; {l = 0; co = 0; r = 0};
             {l = 0; co = 0; r = 0}|]
```

Un troisième exemple plus intéressant sera le tissu de la Figure 1, qu'on supposera de petite taille.

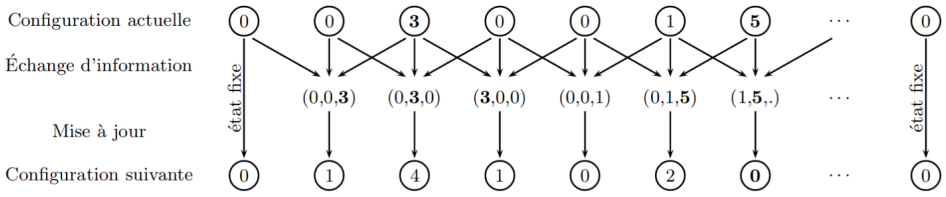

FIG. 1. Modélisation des influences entre cellules ($K = 5$ et $I = 2$).

```
In [17]: let muscle3 = [| 0; 0; 3; 0; 0; 1; 5; 0; 0 |];;

         let lcor3 = cree_lcor muscle3;;

Out[17]: val muscle3 : int array = [|0; 0; 3; 0; 0; 1; 5; 0; 0|]


Out[17]: val lcor3 : lcor array =
           [|{l = 0; co = 0; r = 3}; {l = 0; co = 3; r = 0}; {l = 3; co = 0; r = 0};
             {l = 0; co = 0; r = 1}; {l = 0; co = 1; r = 5}; {l = 1; co = 5; r = 0};
             {l = 5; co = 0; r = 0}|]
```

2.4.4 Une fonction concise pour afficher un tissu (ie. une ligne de cellules)

```
In [18]: let print = Format.printf;;

Out[18]: val print : ('a, Format.formatter, unit) format -> 'a = <fun>


In [19]: let print_tissu_x (etats : etatcellule list) : unit =
             match etats with
             | [] -> print "";
             | s0 :: [] -> print "[ %i ]" s0;
             | s0 :: l -> begin
                 print "[ %i" s0;
                 List.iter (fun (s : etatcellule) -> print "-%i" s) l;
                 print " ]";
             end
         ;;
```

```
Out[19]: val print_tissu_x : etatcellule list -> unit = <fun>
```

- Cette fonction s'exécute en temps linéaire $\mathcal{O}(n)$.

Simple fonction pour *afficher* un `tissu` (revient à afficher un tableau) :

```
In [20]: let print_tissu (o : tissu) : unit =
             print_tissu_x (Array.to_list o)
         ;;

Out[20]: val print_tissu : tissu -> unit = <fun>
```

```
In [21]: print_tissu muscle1;;
         print_endline " : muscle 1.\n";;
         flush_all ();;

         print_tissu muscle2;;
         print_endline " : muscle 2.\n";;
         flush_all ();;

         print_tissu muscle3;;
         print_endline " : muscle 3.\n";;
         flush_all ();;

Out[21]: - : unit = ()


Out[21]: - : unit = ()


Out[21]: - : unit = ()


 : muscle 1.



Out[21]: - : unit = ()


[ 0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0 ][ 0-0-0-0-0-0-0-0-0-0-0-1-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0 ] : musc



Out[21]: - : unit = ()


Out[21]: - : unit = ()
```

```
Out[21]: - : unit = ()


[ 0-0-3-0-0-1-5-0-0 ] : muscle 3.



Out[21]: - : unit = ()


Out[21]: - : unit = ()
```

2.4.5 La fonction demandée pour faire évoluer un tissu, étape par étape.

Attention, elle agit par effet de bords ! (i.e., en modifiant *en place* le tissu donné, c'est pour ça qu'on utilise des tableaux et non des listes).

Cette fonction renvoie la valeur du tissu, pour éventuellement faire des sauvegardes ou l'afficher, mais le tissu est bien modifié en place !

```
In [22]: let une_etape_infection (ti : tissu) : tissu =
             let n = Array.length ti in
             let lcor_ti = cree_lcor ti in
             for j = 1 to n - 2 do
                 ti.(j) <- nouvel_etat_cellule lcor_ti.(j-1) ()
             done;
             ti
         ;;

Out[22]: val une_etape_infection : tissu -> tissu = <fun>
```

- Cette fonction s'exécute en temps linéaire $\mathcal{O}(n)$, puisque `cree_lcor` l'est, et `nouvel_etat_cellule` est en temps constant.

Un exemple sur le tissu sain, le tissu infecté au milieu et le tissu de la Figure 1.

```
In [23]: une_etape_infection muscle1;;

Out[23]: - : tissu = [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0|]


In [24]: une_etape_infection muscle2;;

Out[24]: - : tissu = [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 2; 0; 0; 0; 0; 0; 0; 0; 0|]


In [25]: une_etape_infection muscle3;;

Out[25]: - : tissu = [|0; 1; 4; 1; 0; 2; 0; 1; 0|]
```

2.4.6 Faire évoluer sur plusieurs étapes, en affichant les étapes intermédiaires

```
In [26]: let m_etapes_infection (ti : tissu) (m : int) : tissu =
             print "\n\nSTART : %i étapes de propagation de l'infection." m;
             print "\nÉtape  0 : ";
             print_tissu ti;
             for j = 1 to m do
                 print "\nÉtape %.2i : " j;
                 print_tissu (une_etape_infection ti);
             done;
             flush_all ();
             ti
         ;;

Out[26]: val m_etapes_infection : tissu -> int -> tissu = <fun>
```

- Cette fonction s'exécute en temps $\mathcal{O}(n \times m)$, puisqu'elle appelle m fois `une_etape_infection` qui est en temps linéaire $\mathcal{O}(n)$.

On relance les exemples depuis le début :

```
In [27]: let muscle1 = nouveau_tissu 20;;
         m_etapes_infection muscle1 3;;
         (* Il n'évolue pas *)

Out[27]: val muscle1 : tissu =
           [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0|]



START : 3 étapes de propagation de l'infection.
Étape  0 : [ 0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0 ]
Étape 01 : [ 0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0 ]
Étape 02 : [ 0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0 ]


Out[27]: - : tissu = [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0|]


In [28]: let muscle2 = let m = nouveau_tissu 21 in begin infecte_cible m 11;  m end;;
         m_etapes_infection muscle2 40;;
         (* tout le tissu est infecté ! Dès la 20ème étape *)

Out[28]: val muscle2 : tissu =
           [|0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 1; 0; 0; 0; 0; 0; 0; 0; 0|]
```

```
Étape 03 : [ 0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0 ]

START : 40 étapes de propagation de l'infection.
Étape  0 : [ 0-0-0-0-0-0-0-0-0-0-0-1-0-0-0-0-0-0-0-0-0-0-0 ]
Étape 01 : [ 0-0-0-0-0-0-0-0-0-0-0-2-0-0-0-0-0-0-0-0-0-0-0 ]
Étape 02 : [ 0-0-0-0-0-0-0-0-0-0-1-3-1-0-0-0-0-0-0-0-0-0-0 ]
Étape 03 : [ 0-0-0-0-0-0-0-0-0-0-2-4-2-0-0-0-0-0-0-0-0-0-0 ]
Étape 04 : [ 0-0-0-0-0-0-0-0-0-1-3-5-3-1-0-0-0-0-0-0-0-0-0 ]
Étape 05 : [ 0-0-0-0-0-0-0-0-0-2-4-0-4-2-0-0-0-0-0-0-0-0-0 ]
Étape 06 : [ 0-0-0-0-0-0-0-0-1-3-5-1-5-3-1-0-0-0-0-0-0-0-0 ]
Étape 07 : [ 0-0-0-0-0-0-0-0-2-4-0-2-0-4-2-0-0-0-0-0-0-0-0 ]
Étape 08 : [ 0-0-0-0-0-0-0-1-3-5-1-3-1-5-3-1-0-0-0-0-0-0-0 ]
Étape 09 : [ 0-0-0-0-0-0-0-2-4-0-2-4-2-0-4-2-0-0-0-0-0-0-0 ]
Étape 10 : [ 0-0-0-0-0-0-1-3-5-1-3-5-3-1-5-3-1-0-0-0-0-0-0 ]
Étape 11 : [ 0-0-0-0-0-0-2-4-0-2-4-0-4-2-0-4-2-0-0-0-0-0-0 ]
Étape 12 : [ 0-0-0-0-0-1-3-5-1-3-5-1-5-3-1-5-3-1-0-0-0-0-0 ]
Étape 13 : [ 0-0-0-0-0-2-4-0-2-4-0-2-0-4-2-0-4-2-0-0-0-0-0 ]
Étape 14 : [ 0-0-0-0-1-3-5-1-3-5-1-3-1-5-3-1-5-3-1-0-0-0-0 ]
Étape 15 : [ 0-0-0-0-2-4-0-2-4-0-2-4-2-0-4-2-0-4-2-0-0-0-0 ]
Étape 16 : [ 0-0-0-1-3-5-1-3-5-1-3-5-3-1-5-3-1-5-3-1-0-0-0 ]
Étape 17 : [ 0-0-0-2-4-0-2-4-0-2-4-0-4-2-0-4-2-0-4-2-0-0-0 ]
Étape 18 : [ 0-0-1-3-5-1-3-5-1-3-5-1-5-3-1-5-3-1-5-3-0-0-0 ]
Étape 19 : [ 0-0-2-4-0-2-4-0-2-4-0-2-0-4-2-0-4-2-0-4-0-0-0 ]
Étape 20 : [ 0-1-3-5-1-3-5-1-3-5-1-3-1-5-3-1-5-3-1-5-0-0-0 ]
Étape 21 : [ 0-2-4-0-2-4-0-2-4-0-2-4-2-0-4-2-0-4-2-0-0-0-0 ]
Étape 22 : [ 0-3-5-1-3-5-1-3-5-1-3-5-3-1-5-3-1-5-3-1-0-0-0 ]
Étape 23 : [ 0-4-0-2-4-0-2-4-0-2-4-0-4-2-0-4-2-0-4-2-0-0-0 ]
Étape 24 : [ 0-5-1-3-5-1-3-5-1-3-5-1-5-3-1-5-3-1-5-3-0-0-0 ]
Étape 25 : [ 0-0-2-4-0-2-4-0-2-4-0-2-0-4-2-0-4-2-0-4-0-0-0 ]
Étape 26 : [ 0-1-3-5-1-3-5-1-3-5-1-3-1-5-3-1-5-3-1-5-0-0-0 ]
Étape 27 : [ 0-2-4-0-2-4-0-2-4-0-2-4-2-0-4-2-0-4-2-0-0-0-0 ]
Étape 28 : [ 0-3-5-1-3-5-1-3-5-1-3-5-3-1-5-3-1-5-3-1-0-0-0 ]
Étape 29 : [ 0-4-0-2-4-0-2-4-0-2-4-0-4-2-0-4-2-0-4-2-0-0-0 ]
Étape 30 : [ 0-5-1-3-5-1-3-5-1-3-5-1-5-3-1-5-3-1-5-3-0-0-0 ]
Étape 31 : [ 0-0-2-4-0-2-4-0-2-4-0-2-0-4-2-0-4-2-0-4-0-0-0 ]
Étape 32 : [ 0-1-3-5-1-3-5-1-3-5-1-3-1-5-3-1-5-3-1-5-0-0-0 ]
Étape 33 : [ 0-2-4-0-2-4-0-2-4-0-2-4-2-0-4-2-0-4-2-0-0-0-0 ]
Étape 34 : [ 0-3-5-1-3-5-1-3-5-1-3-5-3-1-5-3-1-5-3-1-0-0-0 ]
Étape 35 : [ 0-4-0-2-4-0-2-4-0-2-4-0-4-2-0-4-2-0-4-2-0-0-0 ]
Étape 36 : [ 0-5-1-3-5-1-3-5-1-3-5-1-5-3-1-5-3-1-5-3-0-0-0 ]
Étape 37 : [ 0-0-2-4-0-2-4-0-2-4-0-2-0-4-2-0-4-2-0-4-0-0-0 ]
Étape 38 : [ 0-1-3-5-1-3-5-1-3-5-1-3-1-5-3-1-5-3-1-5-0-0-0 ]
Étape 39 : [ 0-2-4-0-2-4-0-2-4-0-2-4-2-0-4-2-0-4-2-0-0-0-0 ]

Out[28]: - : tissu = [|0; 3; 5; 1; 3; 5; 1; 3; 5; 1; 3; 5; 3; 1; 5; 3; 1; 5; 3; 1; 0|]

In [29]: let muscle3 = [| 0; 0; 3; 0; 0; 1; 5; 0; 0 |];;
```

```
        m_etapes_infection muscle3 8;;
        (* 3 étapes suffisent à tout infecter *)

Out[29]: val muscle3 : int array = [|0; 0; 3; 0; 0; 1; 5; 0; 0|]


Out[29]: - : tissu = [|0; 2; 5; 2; 1; 3; 1; 2; 0|]


Étape 40 : [ 0-3-5-1-3-5-1-3-5-1-3-5-3-1-5-3-1-5-3-1-0 ]

START : 8 étapes de propagation de l'infection.
Étape  0 : [ 0-0-3-0-0-1-5-0-0 ]
Étape 01 : [ 0-1-4-1-0-2-0-1-0 ]
Étape 02 : [ 0-2-5-2-1-3-1-2-0 ]
Étape 03 : [ 0-3-0-3-2-4-2-3-0 ]
Étape 04 : [ 0-4-1-4-3-5-3-4-0 ]
Étape 05 : [ 0-5-2-5-4-0-4-5-0 ]
Étape 06 : [ 0-0-3-0-5-1-5-0-0 ]
Étape 07 : [ 0-1-4-1-0-2-0-1-0 ]
```

Voilà, ce sera tout pour cette solution.

2.5 Complexités en temps et espace (bonus)

Il est toujours utile de préciser, rapidement à l'oral et/ou dans le code (un commentaire suffit) les complexité (ou ordre de grandeur) des fonctions exigées par l'énoncé.

2.5.1 En temps

- Une étape d'infection, i.e., `une_etape_infection`, est en temps linéaire $\mathcal{O}(n)$ (et c'est optimal),
- Donc m étapes, i.e., `m_etapes_infection`, est en temps $\mathcal{O}(n \times m)$ (et c'est optimal).

2.5.2 En espace

Tout est bien évidemment linéaire en n la taille du tissu.

2.6 Conclusion

Voilà pour la question obligatoire de programmation.

2.6.1 Qualités

- On a décomposé le problème en 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 de petits exemples et sur un exemple de taille réelle (venant du texte).

2.6.2 Défauts

- Par contre, on a testé avec uniquement une valeur pour I et K (resp., 2 et 5), et les fonctions écrites ne sont pas paramétriques en I et K. (notez que ce serait assez trivial de les rendre des paramètres)

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

  Vous auriez pu choisir de modéliser le problème avec une autre approche, ou vous auriez pu expérimenter des extensions de cette approche (e.g., des tissus en 2D ou 3D).