Devoir d'informatique n° 03 (2 heures)

Ce devoir est constitué d'un probleme comportant 4 parties.
Veillez à soigner la copie tant pour l'écriture, la propreté que pour la rédaction, la rigueur et l'argumentation. On veillera également à l'indentation des scripts Python. Vous numéroterez vos copies et ferez apparaître clairement sur la première page le nombre de copies.
La calculatrice est interdite. Toute sortie est définitive.
Attention : Pour les calculs de complexité, il faudra comptabiliser l'ordre de grandeur du nombre d'opérations éléementaires (addition, soustraction, multiplication, division, affectation, comparaison de nombres, accés à un élément d'un tableau,...).
On estimera que pour les listes ou tableaux Python, (qui sont des tableaux "dynamiques"), la suppression du dernier élément d'une liste de taille N ou l'ajout d'un élément à la fin de cette liste est en $O(1)$ alors que l'ajout ou la suppression d'un élément partout ailleurs est en $O(n)$.
A part les fonctions de manipulation de listes rappelées ci-dessous, et celles écrites dans l'énoncé, vous n'aurez pas le droit d'utiliser d'autres fonctions Python sauf à les réécrire.

- ✏ On peut créer une liste de taille n remplie de la valeur x avec `li = [x] * n`. Estimation du coût : $O(n)$.
- ✏ On peut obtenir la taille d'une liste `li` avec len(`li`). Estimation du coût : $O(1)$.
- ✏ Si `li` est une liste de n éléments, on peut accéder au $k-$ème élément (pour $0 \leqslant k < len$(`li`)) avec `li[k]`. On peut alors définier (ou modifier) sa valeur avec `li[k]` = `x`. Estimation du coût : $O(1)$.
- ✏ On peut ajouter un élément `x` dans une liste `li` à l'aide de `li.append(x)`. Estimation du coût : $O(1)$.
- ✏ On peut supprimer le dernier élément d'une liste non vide `li` à l'aide de `li.pop()`. Estimation du coût : $O(1)$.
- ✏ Les matrices sont des listes de listes, chaque sous-liste est considérée comme une ligne de la matrice. Si `mat` est une matrice, elle possède len(`mat`) lignes et len(`mat`[0]) colonnes.
- ✏ On peut créer une matrice de n lignes, p colonnes, dont toutes les cases contiennent `x` avec `mat = [[x for j in range(n)] for i in range(n)]`. Estimation du coût : $O(np)$.
- ✏ On accède (resp. modifie) l'élément de `mat` dans la $i-$ème ligne et $j-$ème colonne avec `mat[i][j]` (resp. `mat[i][j] = x`). Estimation du coût : $O(1)$.

Toute autre fonction sur les listes (`in, sort, index, max, +`, copie de listes ou de sous-listes, etc) est interdite (sauf à les réecrire et estimer sa complexité...).
On rappelle enfin qu'une fonction qui s'arrête sans avoir rencontré l'instruction `return` renvoie `None`.

Requêtes simples en SQL

```
SELECT colonne(s) FROM table WHERE condition(s)
```

- ✏ SELECT colonne(s) correspond à une projection sur l'ensemble des attributs choisis. Si on veut tous les attributs, on remplace 'colonne(s)' par *
- ✏ FROM permet de choisir la table
- ✏ WHERE permet la sélection des entrées vérifiant une certaine condition

On peut compléter cette syntaxe par :

- ✏ SELECT DISTINCT : pour éviter les doublons
- ✏ ORDER BY ... pour trier dans l'ordre croissant de l'attribut (ou les attributs) qui sera choisi. Si on veut l'ordre décroissant on utilisera ORDER BY ... DESC
- ✏ LIMIT n si on ne veut retenir que les n premières entrées
- ✏ AS : pour le renommage
- ✏ UNION : pour ...
- ✏ INTERSECT : pour l'intersection
- ✏ EXCEPT : pour la différence
- ✏ rel1 JOIN rel2 ON conditions : effectue la jointure
- ✏ LIKE : pour donner le début ou la fin d'un mot dans une requête

Agregation et groupage

- ✏ SELECT COUNT(A) FROM table : compte le nombre de valeurs de l'attribut A
- ✏ SELECT MAX(A) FROM table : renvoie le maximum des valeurs de l'attribut A
- ✏ SELECT MIN(A) FROM table : renvoie le minimum des valeurs de l'attribut A
- ✏ SELECT SUM(A) FROM table : renvoie la somme des valeurs de l'attribut A
- ✏ SELECT AVG(A) FROM table : renvoie la moyenne des valeurs de l'attribut A
- ✏ SELECT * FROM table GROUP BY A : regroupe les entrées selon l'attribut A.
- ✏ HAVING : permet d'exprimer une condition sur le regroupement précédent

Prologue

Dans un univers inspiré du roman *1984* de George Orwell, la police de la pensée utilise l'ordinateur Big Brother pour surveiller les citoyens et citoyennes du pays. Ses fonctionnaires souhaitent intervenir rapidement si une menace sérieuse se profile (par exemple en cas de manifestation, de critique de Big Brother ou de bavardages au travail).
Votre rôle dans ce devoir sera de programmer des fonctions supplémentaires sur Big Brother pour le rendre plus efficace.

1 Surveillance d'individus

1.1 Un peu d'action

Dans cette partie, Big Brother doit controler N individus, qui disposeront d'un numéro pris dans l'ensemble $[\![0, N-1]\!]$. La police a attribué à chaque individu une note d'insoumission (plus la note est élevée, plus l'individu est rebelle).
Ces notes sont toutes distinctes, et stockées dans une liste `notes`, de telle sorte que `notes[i] = k` signifie que l'individu numéro i a la note k. On disposera également dans cette partie d'une fonction `envoyer_en_prison`, telle que `envoyer_en_prison(i, notes)` envoie automatiquement l'individu numéro i de cette liste en prison. Cette fonction renvoie `None` et possède une complexité $O\left(\sqrt{N}\right)$. On signale également qu'un individu envoyé en prison continue de figurer dans la liste `notes`

Q. 1) Écrire une fonction `eliminer_le_chef(notes)` qui prend en argument une liste de notes et envoie en prison l'individu ayant la note la plus élevée. Cette fonction renvoie `None`

Q. 2) Écrire une fonction `trier(notes)` qui prend en entrée une liste `notes` et renvoie une liste contenant les mêmes valeurs, triées par ordre croissant. La liste `notes` ne doit pas être modifiée par cette fonction. Préciser le nom et la complexité du tri utilisé (on n'exige pas un tri efficace ici).

Q. 3) Écrire une fonction `rompre_sedition(notes)` qui envoie en prison les 10% des individus avec les notes les plus élevées. Si le nombre d'individus n'est pas un multiple de 10, on arrondira le nombre d'envoi indépendamment au nombre entier supérieur ou inférieur. Cette fonction ne doit pas modifier la liste `notes` et ne renvoie rien.
Donner et justifier la complexité de cette fonction.

1.2 Intelligence artificielle

On s'intéresse dans cette sous-partie à la manière dont Big Brother établit les notes de dangerosité des individus. On dispose pour cela d'une base de données des délits commis par les individus. Cette base est constituée de deux tables :

- ☞ La table `individus`, qui possède un enregistrement par personne, et dont les attributs sont :
  - ✔ `matricule`, le numéro de matricule de la personne, il s'agit d'une clé primaire de la table,
  - ✔ `nom`, le nom de cette personne,
  - ✔ `employeur`, l'employeur de cette personne
- ☞ La table `delits`, qui possède un enregistrement par délits commis, et dont les attributs sont :
  - ✔ `coupable`, le numéro de matricule de la personne en faute,
  - ✔ `peine`, le nombre de jours de prison associé à ce délit,
  - ✔ `indic`, numéro de matricule de l'individu qui a dénoncé de délit.

Voici un extrait de ces tables :

individus

matricule	nom	employeur
1	Winston Smith	Ministère de la vérité
2	Julia	Commissariat aux romans
...	...	...

delits

coupable	peine	indic
1	250	27
2	125	1
...	...	...

Q. 4) Indiquer comment obtenir les informations suivantes à partir de la base. On utilisera une seule requête, rédigée en SQL, par question.

Q. 4)a) Le nombre de personnes répertoriées dans la base.

Q. 4)b) Le nom de la personne qui a été condamnée à la plus grande peine au total et le nombre de jours de peine au total.

Q. 4)c) Le nombre de cas de personnes qui se sont dénoncées elles-mêmes.

Q. 4)d) La liste des personnes dénoncées au moins une fois par une personne ayant le même employeur.

2 Analyse numérique

2.1 Évolution de populations

Les modèles compartimentaux[1] sont des modèles déterministes où la population est divisée en plusieurs catégories selon leur statut selon Big Brother. On considère dans cette partie un modèle à quatre compartiments disjoints : libres (L, les personnes en liberté), prison (P, les personnes en prison), fugitifs (F, les personnes qui devraient être en prison mais qui ne le sont pas), et indics (I, les personnes qui aident la police à mettre les fugitifs en prison).
Le changement d'état des individus est gouverné par un système d'équations diffrentielles. En notant $L(t), P(t), F(t)$ et $I(t)$ la fraction de la population appartenant à chacune des quatre catégories à l'instant t, on obtient le système :

$$\begin{cases} \frac{d}{dt}L(t) = -aL(t)F(t) - bL(t) + dP(t) \\ \frac{d}{dt}P(t) = cI(t)F(t) - dP(t) \\ \frac{d}{dt}F(t) = b(I(t) + L(t)) - cI(t)F(t) \\ \frac{d}{dt}I(t) = aL(t)F(t) - bI(t) \end{cases}$$

où a est le taux de recrutement, b le taux de délit, c le taux de capture, et d le taux de libération.
On supppose qu'à l'instant initial, on a $L(0) = 0.85, F(0) = 0.15$ et $P(0) = I(0) = 0$.

Q. 5) Compléter les lignes 22 et suivantes du code suivant

```
1  import numpy as np
2
3  # Parametres
4  t_max = 25.
5  a = 2
6  b = 0.2
7  c = 1.5
8  d = 0.2
9
10 N = 150
11 dt = t_max / N
12
13 t = [0]
14 L = [0.85]
15 P = [0]
16 F = [0.15]
17 I = [0]
18
19 # Methode d'Euler
20 for i in range(N):
21     t.append(t[i]+dt)
22     L.append(...) # a completer
23     P.append(...) # a completer
24     F.append(...) # a completer
25     I.append(...) # a completer
```

1. inspirée de CCMP 2016

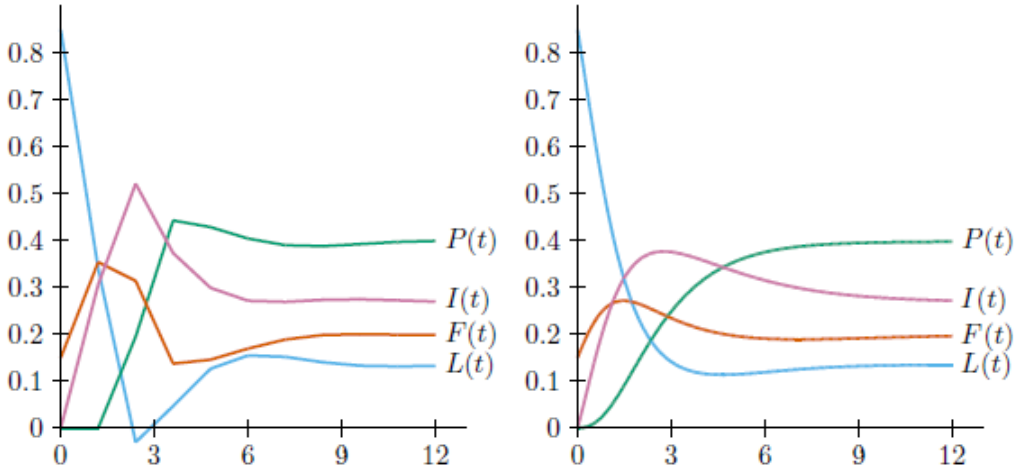

Figure 1: Tracé des résultats obtenus avec $N = 10$ et $N = 150$ avec le programme d'étude de l'évolution de la population.

Q. 6) En traçant les résultats, on obtient les deux figures de la figure 1 (à gauche on a utilisé $N = 10$ et $N = 150$ à droite). Expliquer la différence entre les deux courbes. Donner la complexité du calcul en fonction de la valeur de N.

Q. 7) On suppose ici qu'on dispose d'une fonction `P(t)` qui donne le nombre de prisonniers à l'instant `t`. Big Brother souhaiterait une estimation du coût de ses prisons sur un intervalle de temps $[a, b]$ ce qui revient à calculer $\displaystyle\int_a^b P(t)\,dt$. Écrire une fonction `cout_prison(a, b, N)` qui renvoie une estimation de cette valeur à l'aide de la méthodes des trapèzes, avec N substitutions.

2.2 Éléments finis

Big Brother constate que les personnes en prison sont assez uniformément malheureuses. Pour expliquer ce phénomène, on propose dans cette partie une modélisation de la propagation d ubonheur en environnement délimité.
On place un milieu uniforme de prisonniers dans un carré qu'on assimilera au carré $[0, 1] \times [0, 1]$, et on notera $b(x, y, t)$ le niveau de bonheur à l'instant t aux coordonnées (x, y).

On modélise le transfert de bonheur par une équation du même type que la diffusion thermique, à savoir $\dfrac{\partial b}{\partial t}(x, y, t) = \alpha\,\Delta b(x, y, t) - \beta(x, y)$ où :

- ✏ α est une constante, appelée *coefficient de diffusivité du bonheur*,
- ✏ β est une fonction $[0, 1] \times [0, 1] \to \mathbb{R}$ appelée *évaporation locale de bonheur*,
- ✏ Δ désigne l'opérateur Laplacien, c'est-à-dire qu'on a $\Delta b(x, y, t) = \dfrac{\partial^2 b}{\partial x^2}(x, y, t) + \dfrac{\partial^2 b}{\partial y^2}(x, y, t)$

Pour évaluer les valeurs de $b(x, y, t)$, on discrètise le modèle. Tout d'abord on ne va considérer qu'un nombre fini de valeurs de t. Pour chacune de ces valeurs, on construit des matrices carrées $B_t \in \mathcal{M}_{n+1}(\mathbb{R})$, telles que $(B_t)_{i,j}$ soit une approximation de $b(i/N, j/N, t)$, pour $0 \leqslant i, j \leqslant N$.
Attention : contrairement à l'usage mathématique, les indices des matrices sont bien numérotés à partir de 0. De plus une ligne de la matrice correspond à une droite verticale dans le plan (x constant).
On note l'approximation classique, pour une petite valeur de h : $\dfrac{\partial b}{\partial x}(x, y, t) = \dfrac{b(x + h, y, t) - b(x - h, y, t)}{2h}$ et de même pour la dérivée partielle selon y.

Q. 8) Soit i et j compris entre 1 et N. On suppose qu'on dispose de `Bt` une représentation en Python d'une matrice B_t. Compléter la ligne manquante (la n° 2...) :

```
1 def delta_b(Bt, i, j) :
2     coeff = ...
3     return coeff *  (Bt[i+1][j] + Bt[i-1][j]
4                     + Bt[i][j-1] + Bt[i][j+1]
5                     - 4*Bt[i][j])
```

Et justifier que cette fonction calcule bien une approximation de $\Delta b(i/N, j/N, t)$

En se basant sur la méthode d'Euler, on va maintenant considérer qu'on passe de la matrice Bt1 (à l'instant t_1) à la matrice Bt2 (à l'instant $t_2 = t_1 + dt$) avec l'approximation :

$$\frac{\texttt{Bt2[i][j] - Bt1[i][j]}}{dt} = \alpha\,\texttt{delta_b(Bt1, i, j)} - \beta(i/N, j/N)$$

On impose également des conditions aux bords : initialement, le niveau de bonheur de tout le monde est égal à 0, sauf pour les personnes au bord dont le niveau est à 10. Les personnes sur le bord conservent ce niveau tout le temps.

Q. 9) Écrire une fonction `init(N)` qui renvoie une matrice carrée de dimension $(N+1) \times (N+1)$ correspondant à l'état initial.

Q. 10) Écrire une fonction `evolution(Bt1, alpha, beta, dt)` qui renvoie une matrice `Bt2` correspondant à l'évolution de la matrice `Bt1` pendant un temps `dt`, et avec la constante `alpha` et la fonction `beta` définies comme ci-dessus.

Q. 11) Écrire une fonction `bonheur(N, alpha, beta, dt, k)` qui renvoie la matrice de dimension $(N+1) \times (N+1)$ correspondant à l'état après un temps $k \times dt$

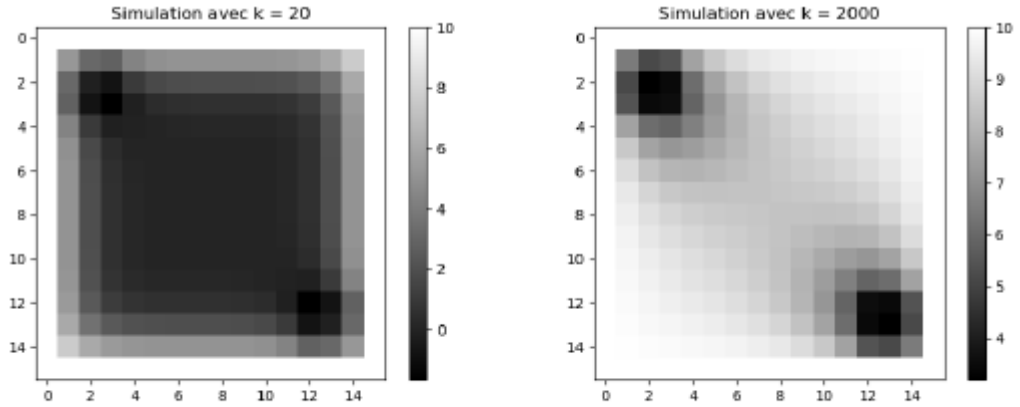

Figure 2: Exemple de résultat obtenu avec une évaporation nulle, sauf dans les coins

3 Algorithmique

Dans cette partie, Big Brother cherche à empêcher tout embryon de manifestation. Les individus surveillés sont assimilés à des points du carré $[0,1] \times [0,1]$, et l'objectif sera de retrouver efficacement les deux individus les plus proches. On supposera de plus dans toute la partie que les abscisses des points sont 2 à 2 distinctes.
On suppose qu'on dispose de trois variables globales :

- ✏ `N` est le nombre de points ,
- ✏ `coords_x` est une liste de N flottants distincts donnant les abscisses des points,
- ✏ `coords_y` est une liste de N flottants donnant les ordonnées des points.

Ainsi, les coordonnées du point numéro i sont (`coods_x[i], coords_y[i]`), pour $0 \leqslant i \leqslant N-1$.

3.1 Méthode naïve

Q. 12) Écrire une fonction `distance(i, j)` qui prend en argument deux numéros de points compris entre 0 et $N-1$ et renvoie la distance euclidienne entre les individus portant ces matricules.
On pourra calculer la racine carrée d'un nombre `x` avec `x ** 0.5`.

Q. 13) Écrire une fonction naïve `plus_proche()` qui renvoie la liste des deux numéros `i` et `j` correspondant aux deux points les plus proches. Donner la complexité de cet algorithme, en la justifiant.

3.2 Quelques outils pour s'améliorer

On souhaite maintenant obtenir la distance entre les deux individus les plus proches avec une meilleure complexité, et nous allons donc décrire un algorithme utilisant une méthode de *diviser pour régner*. Cette partie présente des fonctions utiles pour la mise en oeuvre de cet algorithme.

Q. 14) Pour commencer, nous avons besoin de trier les matricules, par abscisses croissantes d'une part, et par ordonnées croissantes d'autre part. Quel algorithme de tri pourrait-on utiliser pour obtenir ces listes avec la meilleure complexité possible, et quelle serait cette complexité en fonction de N ? *On ne demande ni de justification, ni de coder cet algorithme.*

On admettra qu'on dispose de deux listes de N entiers `liste_x` et `liste_y` telles ques les matricules dans `liste_x` soient rangés par abscisses croissantes, et ceux de `liste_y` par ordonnées croissantes.
Dans toute la suite, un sous-ensemble de points sera décrit avec un *cluster*. Un cluster est une matrice de deux lignes contenant chacune les mêmes numéros, qui sont les numéros des points dans le sous-ensemble considéré. Dans la première ligne, les points sont triés par abscisses croissantes ; dans la seconde, ils sont triés par ordonnées croissantes. La figure 3 donne la représentation de deux clusters.

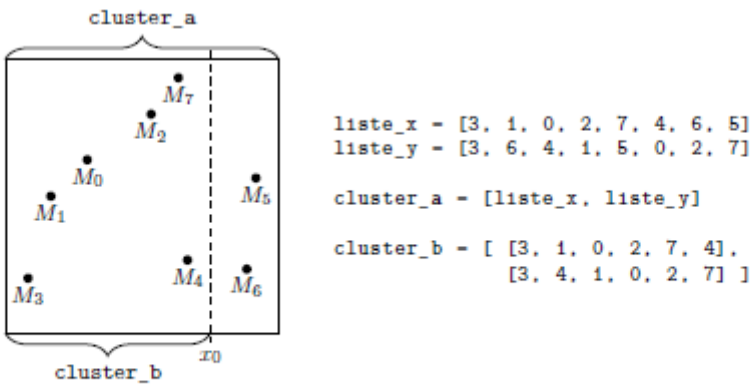

Figure 3: Représentation en Python de deux clusters

Pour être efficace, notre algorithme ne peut pas re-trier les listes de numéros de points à chaque étape. Nous allons donc définir une fonction qui permet d'extraire les numéros d'un cluster, et former ainsi un nouveau cluster plus petit.

Q. 15) Écrire une fonction `sous_cluster(cl, x_min, x_max)` qui prend en arguments un cluster `cl`, et deux flottants `x_min`, et `x_max`, et renvoie le sous-cluster des points dont l'abscisse est comprise entre `x_min`, et `x_max`(au sens large). On impose une complexité linéaire en la taille du cluster.

Q. 16) Écrire une fonction `mediane(cl)` qui prend un cluster `cl` et renvoie une abscisse médiane, c'est-à-dire que la moitié (au moins) des points a une abscisse inférieure ou égale à cette valeur, et la moitié (au moins) des points a une abscisse supérieure ou égale à cette valeur. On impose une complexité $O(1)$

3.3 Méthode sophistiquée

Le fonctionnement de l'agorithme est illustré par la figure 4 :

i) Si le cluster contient deux ou trois points, on calcule la distance minimale en calculant toutes les distances possibles.

ii) Sinon, on sépare le cluster en deux parties G et D qu'on supposera de tailles égales (à un point près) suivant la médiane des abscisses, qu'on notera x_0.

iii) Les deux points les plus proches sont soit les deux dans G, soit les deux dans D, soit un de chaque côté.

iv) On calcule récursivement le couple le plus proche dans la moitié gauche et dans la moitié droite, on note d_0 la plus petite des deux distances obtenues.

v) On cherche s'il existe une paire de points (M_1, M_2) tels que M_1 est dans G, M_2 est dans D, et $d(M_1, M_2) < d_0$.

vi) Si on en trouve une (ou plusieurs), on renvoie la plus petite. Sinon on renvoie d_0.

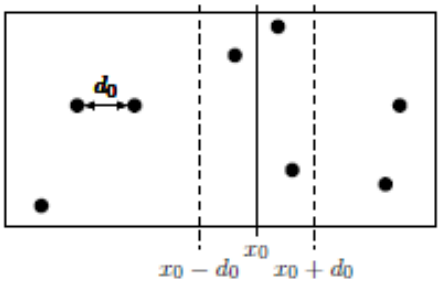

Figure 4: Illustration du diviser pour régner

Q. 17) Écrire une fonction `gauche(cl)` qui prend en argument un cluster `cl`, et renvoie le cluster constitué uniquement de la moitié (éventuellement arrondie à l'entier supérieur) des points les plus à gauche. On rappelle que toutes les abscisses sont dans $[0, 1]$.
On supposera disposer d'une fonction `droite(cl)` qui renvoie le cluster de droite.

Q. 18) Justifier qu'on peut se contenter de chercher les points M_1, M_2 de l'étape **v)** dans les points dont l'abscisse appartient à $I_0 = [x_0 - d_0, x_0 + d_0]$.

Q. 19) Écrire une fonction `bande_centrale(cl, d0)` qui prend en argument un cluster `cl` et un réel `d0`, et renvoie le cluster des individus dont l'abscisse est dans I_0. On impose encore une fois une complexité linéaire en la taille du cluster.

Q. 20) Montrer que deux points M_1 et M_2 de la bande centrale situés à une distance inférieure à d_0 l'un de l'autre sont au plus à 7 cases d'intervalle dans la deuxième ligne du cluster (c'est-à-dire la ligne triée par ordonnées croissantes).
On pourra montrer qu'un rectangle à préciser de dimension $2d_0 \times d_0$ contient au plus 8 points.

Q. 21) En déduire une fonction `fusion(cl, d0)` qui prend en entrée un cluster de points dont toutes les abscisses sont dans un intervalle $[x_0 - d_0, x_0 + d_0]$, et renvoie la distance minimale entre deux points du cluster si elle est inférieure à d_0, ou d_0 sinon. On impose une complexité linéaire en la taille du cluster `cl`, et on justifiera cette complexité.

Q. 22) Écrire une fonction récursive `distance_minimale(cl)` qui prend en argument un cluster et utilise l'algorithme défini plus haut pour renvoyer la distance minimale entre deux points du cluster.

Q. 23) Si on note n la taille du cluster `cl`, justifier que la complexité $C(n)$ de la fonction `distance_minimale(cl)` vérifie : $C(n) \leqslant 2C(n/2) + O(n)$

Q. 24) En déduire la complexité $C(n)$. On pourra se limiter au cas où n est une puissance de 2.

4 Propagation

Pour Big Brother, les quartiers de la ville sont assimilés à une grille rectangulaire de dimension $n \times m$. Certains quartiers sont tranquilles alors que d'autres sont en révolte. On attribue la valeur -1 aux quartiers tranquilles, et on attribue aux quartiers en révolte une valeur entière correspondant au jour où ils passent en révolte.
De plus, la révolte se répand : un quartier se révolte au jour $k+1$ si au moins deux quartiers adjacents (les diagonales ne comptent pas) sont en révolte au jour k. Un quartier en révolte le reste ensuite indéfiniement.
Un certain nombre de quartiers sont en révolte initialement et ont donc une valeur de 0. L'objectif est donc de prédire quels quartiers seront finalement en révolte et à partir de quel moment. On montre sur la figure 5 comment évolue la contamination dans un cas particulier de quartier.

$$\begin{bmatrix} 0 & -1 & -1 & -1 \\ -1 & -1 & 0 & -1 \\ 0 & -1 & -1 & -1 \end{bmatrix} \to \begin{bmatrix} 0 & -1 & -1 & -1 \\ 1 & -1 & 0 & -1 \\ 0 & -1 & -1 & -1 \end{bmatrix} \to \cdots \to \begin{bmatrix} 0 & 3 & 4 & -1 \\ 1 & 2 & 0 & -1 \\ 0 & 3 & 4 & -1 \end{bmatrix}$$

Figure 5: Un exemple de quartier représenté aux jours 0, 1, et 4.

Q. 25) Écrire une fonction `voisins(mat, i, j)` qui prend en argument une matrice `mat` décrivant l'état de révolte des quartiers, et qui renvoie le nombre de quartiers en révolte voisins du quartier de la case (i, j).

Q. 26) Écrire une fonction `contamination(mat)` qui transforme la matrice `mat` correspondant à un état initial de quartiers (avec uniquement des -1 et des 0) en une matrice finale (lorsqu'il n'y a plus d'évolution)