Projet du cours d'algorithmique

E2

Coccinelles et pucerons

ROUSTANT Jolan

QUINTARD Lohan

Q1 :

```java
static int glouton(int[][] G, int d){
        int totalPuceronsMangés = 0;
        int posX = d;

        //manger les pucerons de la case initiale
        totalPuceronsMangés += G[0][posX];
        //System.out.println("Je mange "+G[0][posX]+" pucerons de la case initiale");

        for(int i = 1; i<G.length; i++){
            //récupérer le nombre de pucerons des cases NO (nord-ouest), N (nord) et NE (nord-est)

            //System.out.println("Je suis en "+posX);

            int pNO = 0, pN = 0, pNE = 0;

            if(posX-1 >= 0){
                pNO = G[i][posX-1];
            }else{pNO = -1;}

            if(posX+1 < G[0].length){
                pNE = G[i][posX+1];
            }else{pNE = -1;}

            pN = G[i][posX];

            //choisir la case avec le plus de pucerons
            if(pNO > pN && pNO > pNE){
                posX = posX-1;
            }else if(pNE > pN && pNE > pNO){
                posX = posX+1;
            }//else : on ne bouge pas

            //manger les pucerons de la case choisie
            //System.out.println("Je mange "+G[i][posX]+" pucerons");
            totalPuceronsMangés += G[i][posX];

        }
        return totalPuceronsMangés;
    }
```

Cette fonction va tout d'abord trouver combien il y a de pucerons dans les cases nord-ouest, nord et nord-est ainsi que mettre -1 si la case n'existe pas. Puis après avoir trouver celle où il y en a le plus, elle va incrémenter le nombre total de puceron mangé ainsi qu'adapter sa position.

Q2 :

```
}
    static int[] glouton(int[][] G){
        int[] Ng = new int[G[0].length];
        for(int d = 0; d<G[0].length; d++)
{           Ng[d] = glouton(G, d);
        }
        return Ng;

    }
```

Cette fonction est assez simple et retourne juste un tableau avec le nombre de puceron mangé en fonction de la case de départ qu'on choisit. Pour chaque position de départ, elle appelle la fonction précédente et stock le résultat dans un tableau.

Q3 :

Base :

m(0,d) = G[0, d]

m(0,c) avec c ≠ d = -1

Hérédité :

$m(l,c)=G(l,c)+\max(m(l-1,c-1),m(l-1,c),m(l-1,c+1))$

Le but c'est donc de mettre à -1 les valeurs qui ne nous intéresses pas. Puis pour l'hérédité on met dans chaque case du tableau, on prend le nombre de pucerons et on lui ajoute le maximum obtenu parmi les cases voisines de la ligne précédente pour déterminer le nombre total de pucerons jusqu'à la case actuelle.

Q4 :

```java
public static int[][][] calculerMA(int[][] G, int d) {
    int[][] M = new int[G.length][G[0].length];
    int[][] A = new int[G.length][G[0].length];

    //initialisation tout à -1
    for(int i = 0; i<G.length; i++){
        for(int j = 0; j<G[0].length; j++){
            M[i][j] = -1;
            A[i][j] = -1;
        }
    }

    //Base
    M[0][d] = G[0][d];

    //Hérédité
    for (int i = 1; i < G.length; i++) {
        for (int j = 0; j < G[0].length; j++) {

            int pNO = -1, pN = -1, pNE = -1;

            if (j - 1 >= 0) {
                pNO = M[i - 1][j - 1];
            }
            pN = M[i - 1][j];
            if (j + 1 < G[0].length) {
                pNE = M[i - 1][j + 1];
            }

            //équation de récurrence uniquement si les cases existent
            if (pNO != -1 || pN != -1 || pNE != -1) {
                M[i][j] = G[i][j] + max(pNO, pN, pNE);
            }

            //trouver la case précédente et la mettre dans A
            if (pNO > pN && pNO > pNE) {
                A[i][j] = -1;
            } else if (pNE > pN && pNE > pNO) {
                A[i][j] = 1;
            } else {
                A[i][j] = 0;
            }
        }
    }
    return new int[][][] {M, A};
}
```

Pour faire ce programme j'ai d'abord initialisé les deux tableau M et A à -1. Le tableau M comporte pour chaque case le nombre max de puceron mangé en arrivant sur cette case. Pendant ce temps le tableau A prends le chemin parcourt. Chaque valeur prend 1, 0 ou -1 en

fonction de si elle vient du nord-ouest du nord ou du nord-est. Il ne faut pas oublier que la coccinelle ne peut pas sortir du tableau donc on a dû penser à ajouter des vérifications afin de ne pas avoir des erreurs de type *ArrayIndexOutOfBoundsException*.

Q5 :

```java
public static void acnpm(int[][] M, int[][] A){
        int indiceColoneFinal = argMax(M[M.length-1]) ; // colonne d'arrivee du chemin max. d'origine (0,d)
        //System.out.println("Colone d'arrivé = "+indiceColoneFinal);
        acnpm(A, M.length-1, indiceColoneFinal); // affichage du chemin maximum de (0,d) `a (L-1, cStar)
        System.out.println("  Valeur : " + M[M.length-1][indiceColoneFinal]);
}
```

Nous avons pris la décision de changer le nom de certaines variable et d'ajouter un *System.ou.println* afin de rendre le code plus facile à comprendre et d'avoir un résultat plus joli dans l'invite de command car nous avons eu des difficultés sur cette partie.

Il a tout d'abord fallu coder argMax qui prend un tableau en paramètre la dernière ligne du tableau M et va retrouver là où il y a la plus grande valeur (ce qui correspond à la case d'arrivé de la coccinelle).

```java
public static int argMax(int[] T) {
    int maxIndex = 0;
    int maxVal = T[0];

    for (int i = 1; i < T.length; i++)
{       if (T[i] > maxVal) {
            maxVal = T[i];
            maxIndex = i;
        }
    }

    return maxIndex;
}
```

Grâce à ça, on va pouvoir retrouver le chemin de la coccinelle grâce à la fonction *public static void acnpm(int[][] A, int l, int c)* codé après

```java
public static void acnpm(int[][] A, int l, int c)
{   if(l==0){
        System.out.print("("+l+", "+c+")");
    }else{
        if(A[l][c] == -1){
            acnpm(A, l-1, c-1);
        }else if(A[l][c] == 0){
            acnpm(A, l-1, c);
        }else if(A[l][c] == 1){
            acnpm(A, l-1, c+1);
        }
        System.out.print("("+l+", "+c+")");
    }
}
```

Cette fonction est récursive (elle s'appelle elle-même). Elle affiche simplement le chemin parcouru par la coccinelle en retrouvant et affichant les cases par lesquelles elle est passée grâce aux valeurs contenues dans le tableau A.

Q6 et Q7:

```java
public static int optimal(int[][] G, int d){
    int[][][] MA = calculerMA(G, d);
    //afficher le chemin
    acnpm(MA[0], MA[1]);
    return MA[0][MA[0].length-1][argMax(MA[0][MA[0].length-1])];
}

public static int[] optimal(int[][] G){
    int[] Nmax = new int[G[0].length];
    System.out.println("Chemins max. depuis toutes les cases de départ (0,d) : ");
    for(int d = 0; d<G[0].length; d++){
        System.out.print("Un chemin max pour d = "+d+ ":  ");
        Nmax[d] = optimal(G, d);
    }
    return Nmax;
}
```

La première fonction *optimal* affiche le chemin et retourne le nombre total de pucerons mangés, elle prend le tableau G et une position de départ, la seconde fonction *optimal* ne prend en paramètre uniquement le tableau G et retourne un tableau contenant toutes les nombres totaux de pucerons mangés. La seconde fonction se contente surtout d'appeler la

première fonction et stocker les résultats dans un tableau. Voici un exemple de ce qu'affiche la seconde fonction *optimal* :

```
Chemins max. depuis toutes les cases de départ (0,d) :
Un chemin max pour d = 0:  (0, 0)(1, 1)(2, 2)  Valeur : 17
Un chemin max pour d = 1:  (0, 1)(1, 1)(2, 2)  Valeur : 17
Un chemin max pour d = 2:  (0, 2)(1, 1)(2, 2)  Valeur : 18
Un chemin max pour d = 3:  (0, 3)(1, 3)(2, 2)  Valeur : 16
Un chemin max pour d = 4:  (0, 4)(1, 3)(2, 2)  Valeur : 14
```

Q8 :

```
public static float[] gainrelatif(int[] Nmax, int[] Ng){
    float[] GR = new float[Nmax.length];
    for(int i = 0; i<Nmax.length; i++){
        GR[i] = ( (float)Nmax[i] - (float)Ng[i ])/(float)Ng[i];
    }
    return GR;
}
```

Cette fonction revoie le tableau des gains relatif de la méthode optimal sur la méthode glouton pour chaque case de départ. C'est-à-dire à quel point la méthode optimale est plus efficace que la méthode glouton.

Q9 :

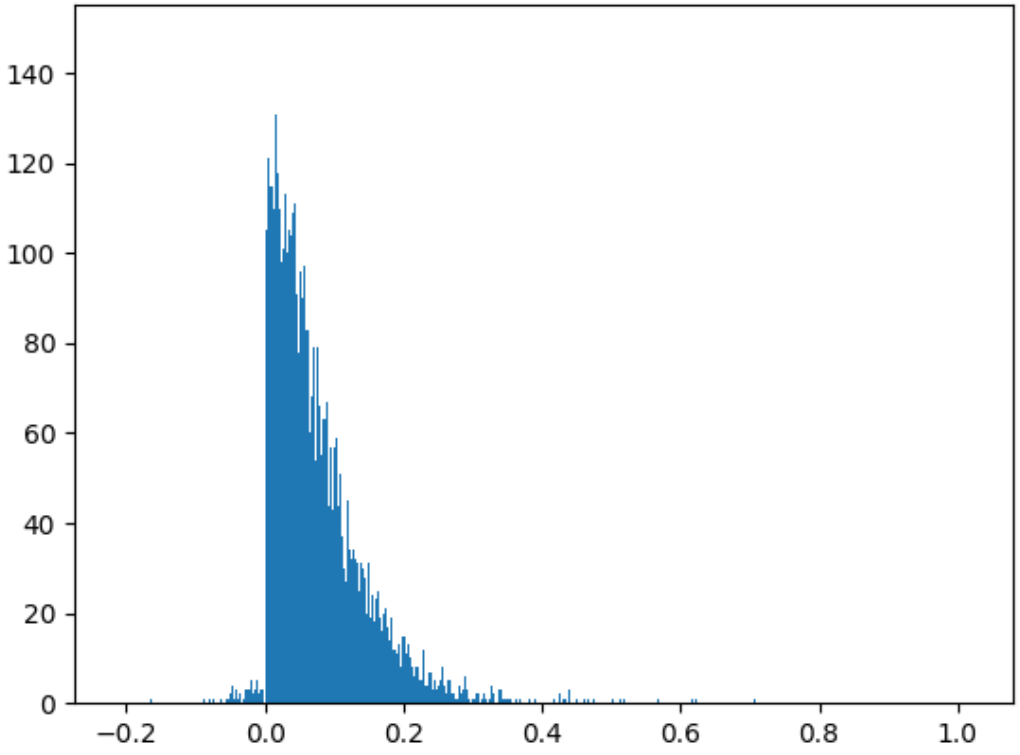
```java
public static void validationStatistique() {
    int nruns = 10000;
    Random rand = new Random();
    try (PrintWriter writer = new PrintWriter(new
FileWriter("gains_relatifs.csv"))) {

        for(int i = 0; i<nruns; i++){
            // L entre 5 et 16
            int L = rand.nextInt(12)+5;
            int C = rand.nextInt(12)+5;
            int[][] grille = generateur(L, C);
            int[] Ng = glouton(grille);
            int[] Nmax = optimal(grille);
            float[] GR = gainrelatif(Nmax, Ng);
            for(int j = 0; j<GR.length; j++){
                if(GR[j] != 0){
                    writer.println(GR[j]);
                }
            }
            //System.out.println("L = "+L+" C = "+C+" GR moyen =
"+moyenne(GR));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }


}
public static int[][] generateur(int n, int m) {
    int[][] G = new int[n][m];
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++) {
            // nombre de pucerons entre 0 et L*C
            G[i][j] = randomInt(0, n*m);
        }
    }
    return G;
}
public static int randomInt(int min, int max) {
    Random rand = new Random();
    return rand.nextInt(max-min+1)+min;
}
public static float moyenne(float[] T) {
    float somme = 0;
    for (int i = 0; i < T.length; i++) {
        somme += T[i];
    }
    return somme/T.length;
}
```

Cette fonction me permet de faire 10000 runs sur des grilles aléatoire et d'écrire les gains dans un fichier csv afin de pouvoir tracer l'histogramme grâce au programme python.

Afin de répondre à cette question j'ai dû chercher comment utiliser les packages *Random*, *FileWriter* et *PrintWriter*, afin de pouvoir mettre les valeurs dans le fichier csv. Voilà l'histogramme généré :



Ayant mal lu le programme python je n'avais pas vu que le programme supprimait déjà les zéros, je l'avais fait avant dans le programme java.

ANNEXE – CODE

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Random;

public class Cocci {
    public static void main(String[] args) {
        //il faut lire la grille à l'envers le nord est en bas
        int[][] grille1 = {
            {2, 2, 3, 4, 2},
            {6, 5, 1, 2, 8},
            {1, 1, 10, 1, 1}
        };


        System.out.println("-------------------------");
        System.out.println("---  Méthode Glouton  ---");
        System.out.println("-------------------------");

        // Grille de test avec une position initiale donnée:
        System.out.println("");
        System.out.println("Grille de test :");
        afficheGrille(grille1);
        System.out.println("");

        int d = 1; // Exemple : position initiale (0, d)
        System.out.println("Grille de test avec une position initiale donnée
(" + d+ "):");
        int puceronsManges = glouton(grille1, d);
        System.out.println("Nombre de pucerons mangés : " + puceronsManges);

        // Grille de test avec toutes les positions initiales possibles:
        System.out.println("Grille de test avec toutes les positions initiales
possibles:");
        int[] Ng = glouton(grille1);
        System.out.print("Nombre de pucerons mangés : Ng = [");
        for (int i = 0; i < Ng.length - 1; i++) {
            System.out.print(Ng[i] + ", ");
        }
        if (Ng.length > 0) {
            System.out.print(Ng[Ng.length - 1]);
        }
        System.out.println("]");
        System.out.println("Avec la méthode glouton, le meilleur chemin est de
manger " + max(Ng) + " pucerons en partant de la position initiale (0, " +
argMax(Ng) + ")");
```

```java
        // affichage Nmax, le meilleur chemin possible pour chaque position
initiale:
        System.out.println();
        int[] Nmax = optimal(grille1);
        System.out.print("Nmax = [");
        System.out.print(" "+Nmax[0]);
        for (int i = 1; i < Nmax.length; i++) {
            System.out.print(", " + Nmax[i]);
        }
        System.out.println(" ]");


        // affichage du gain relatif pour chaque position initiale:
        System.out.println();
        float[] GR = gainrelatif(Nmax, Ng);
        System.out.print("Gain relatif = [");
        System.out.print(" "+GR[0]);
        for (int i = 1; i < GR.length; i++) {
            System.out.print(", " + GR[i]);
        }
        System.out.println(" ]");

        System.out.println();
        //validationStatistique();
    }

    ////////////////////////////////////////////////////////////
    /////////////////////////// GLOUTON ///////////////////////////
    ////////////////////////////////////////////////////////////

    static int glouton(int[][] G, int d){
        int totalPuceronsMangés = 0;
        int posX = d;

        //manger les pucerons de la case initiale
        totalPuceronsMangés += G[0][posX];
        //System.out.println("Je mange "+G[0][posX]+" pucerons de la case
initiale");

        for(int i = 1; i<G.length; i++){
            //récupérer le nombre de pucerons des cases NO (nord-ouest), N
(nord) et NE (nord-est)

            int pNO = 0, pN = 0, pNE = 0;

            if(posX-1 >= 0){
                pNO = G[i][posX-1];
```

```
            }else{pNO = -1;}

            if(posX+1 < G[0].length){
                pNE = G[i][posX+1];
            }else{pNE = -1;}

            pN = G[i][posX];

            //choisir la case avec le plus de pucerons
            if(pNO > pN && pNO > pNE){
                posX = posX-1;
            }else if(pNE > pN && pNE > pNO){
                posX = posX+1;
            }//else : on ne bouge pas

            //manger les pucerons de la case choisie
            //System.out.println("Je mange "+G[i][posX]+" pucerons");
            totalPuceronsMangés += G[i][posX];

        }
        return totalPuceronsMangés;
    }
    static int[] glouton(int[][] G){
        int[] Ng = new int[G[0].length];
        for(int d = 0; d<G[0].length; d++){
            Ng[d] = glouton(G, d);
        }
        return Ng;

    }

    ////////////////////////////////////////////////////////////////
    /////////////////////////// OPTIMAL ////////////////////////////
    ////////////////////////////////////////////////////////////////

    public static int[][][] calculerMA(int[][] G, int d) {
        int[][] M = new int[G.length][G[0].length];
        int[][] A = new int[G.length][G[0].length];

        //initialisation tout à -1
        for(int i = 0; i<G.length; i++){
            for(int j = 0; j<G[0].length; j++){
                M[i][j] = -1;
                A[i][j] = -1;
            }
        }

        //Base
        M[0][d] = G[0][d];
```

```java
        //Hérédité
        for (int i = 1; i < G.length; i++) {
            for (int j = 0; j < G[0].length; j++) {

                int pNO = -1, pN = -1, pNE = -1;

                if (j - 1 >= 0) {
                    pNO = M[i - 1][j - 1];
                }
                pN = M[i - 1][j];
                if (j + 1 < G[0].length) {
                    pNE = M[i - 1][j + 1];
                }

                //équation de récurrence uniquement si les cases existent
                if (pNO != -1 || pN != -1 || pNE != -1) {
                    M[i][j] = G[i][j] + max(pNO, pN, pNE);
                }

                //trouver la case précédente et la mettre dans A
                if (pNO > pN && pNO > pNE) {
                    A[i][j] = -1;
                } else if (pNE > pN && pNE > pNO) {
                    A[i][j] = 1;
                } else {
                    A[i][j] = 0;
                }
            }
        }
        return new int[][][] {M, A};
    }

    public static void acnpm(int[][] M, int[][] A){
        int indiceColoneFinal = argMax(M[M.length-1]) ; // colonne d'arrivee
du chemin max. d'origine (0,d)
        System.out.println("Colone d'arrivé = "+indiceColoneFinal);
        acnpm(A, M.length-1, indiceColoneFinal); // affichage du chemin
maximum de (0,d) `a (L-1, cStar)
        System.out.println("  Valeur : " + M[M.length-1][indiceColoneFinal]);
    }
    public static void acnpm(int[][] A, int l, int c){
        if(l==0){
            System.out.print("("+l+", "+c+")");
        }else{
            if(A[l][c] == -1){
                acnpm(A, l-1, c-1);
            }else if(A[l][c] == 0){
                acnpm(A, l-1, c);
```

```
            }else if(A[l][c] == 1){
                acnpm(A, l-1, c+1);
            }
            System.out.print("("+l+", "+c+")");
        }
    }
    public static int argMax(int[] T) {
        int maxIndex = 0;
        int maxVal = T[0];

        for (int i = 1; i < T.length; i++) {
            if (T[i] > maxVal) {
                maxVal = T[i];
                maxIndex = i;
            }
        }

        return maxIndex;
    }

    public static int optimal(int[][] G, int d){
        int[][][] MA = calculerMA(G, d);
        //afficher le chemin
        acnpm(MA[0], MA[1]);
        return MA[0][MA[0].length-1][argMax(MA[0][MA[0].length-1])];
    }

    public static int[] optimal(int[][] G){
        int[] Nmax = new int[G[0].length];
        System.out.println("Chemins max. depuis toutes les cases de départ
(0,d) : ");
        for(int d = 0; d<G[0].length; d++){
            System.out.print("Un chemin max pour d = "+d+ ":  ");
            Nmax[d] = optimal(G, d);
        }
        return Nmax;
    }

    ////////////////////////////////////////////////////////////////
    /////////////////////////// GAIN ///////////////////////////////
    ////////////////////////////////////////////////////////////////

    public static float[] gainrelatif(int[] Nmax, int[] Ng){
        float[] GR = new float[Nmax.length];
        for(int i = 0; i<Nmax.length; i++){
            GR[i] = ( (float)Nmax[i] - (float)Ng[i ])/(float)Ng[i];
        }
        return GR;
    }
```

```java
////////////////////////////////////////////////////////////
//////////////////////////// OUTILS ////////////////////////
////////////////////////////////////////////////////////////

static void afficheGrille(int[][] G){
    for(int i = G.length-1; i>=0; i--){
        for(int j = 0; j<G[0].length; j++){
            System.out.print(G[i][j]+" ");
        }
        System.out.println();
    }
}

public static int max(int a, int b, int c){
    if(a>b && a>c){
        return a;
    }else if(b>a && b>c){
        return b;
    }else{
        return c;
    }
}
public static int max(int[] T){
    int max = T[0];
    for(int i = 1; i<T.length; i++){
        if(T[i] > max){
            max = T[i];
        }
    }
    return max;
}
public static int min(int a, int b){
    if(a<b){
        return a;
    }else{
        return b;
    }
}
static int[] permutationAleatoire(int[] T){ int n = T.length;
// Calcule dans T une permutation aléatoire de T et retourne T
    Random rand = new Random(); // bibliothèque java.util.Random
    for (int i = n; i > 0; i--){
        int r = rand.nextInt(i); // r est au hasard dans [0:i]
        permuter(T,r,i-1);
    }
    return T;
}
static void permuter(int[] T, int i, int j){
```

```
        int ti = T[i];
        T[i] = T[j];
        T[j] = ti;
    }

// validation statistique
    public static void validationStatistique() {
        int nruns = 10000;
        Random rand = new Random();
        try (PrintWriter writer = new PrintWriter(new
FileWriter("gains_relatifs.csv"))) {

            for(int i = 0; i<nruns; i++){
                // L entre 5 et 16
                int L = rand.nextInt(12)+5;
                int C = rand.nextInt(12)+5;
                int[][] grille = generateur(L, C);
                int[] Ng = glouton(grille);
                int[] Nmax = optimal(grille);
                float[] GR = gainrelatif(Nmax, Ng);
                for(int j = 0; j<GR.length; j++){
                    if(GR[j] != 0){
                        writer.println(GR[j]);
                    }
                }
                //System.out.println("L = "+L+" C = "+C+" GR moyen =
"+moyenne(GR));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }


    }
    public static int[][] generateur(int n, int m) {
        int[][] G = new int[n][m];
        for (int i = 0; i < n; i++){
            for (int j = 0; j < m; j++) {
                // nombre de pucerons entre 0 et L*C
                G[i][j] = randomInt(0, n*m);
            }
        }
        return G;
    }
    public static int randomInt(int min, int max) {
        Random rand = new Random();
        return rand.nextInt(max-min+1)+min;
    }
    public static float moyenne(float[] T) {
```

```
        float somme = 0;
        for (int i = 0; i < T.length; i++) {
            somme += T[i];
        }
        return somme/T.length;
    }
}
```