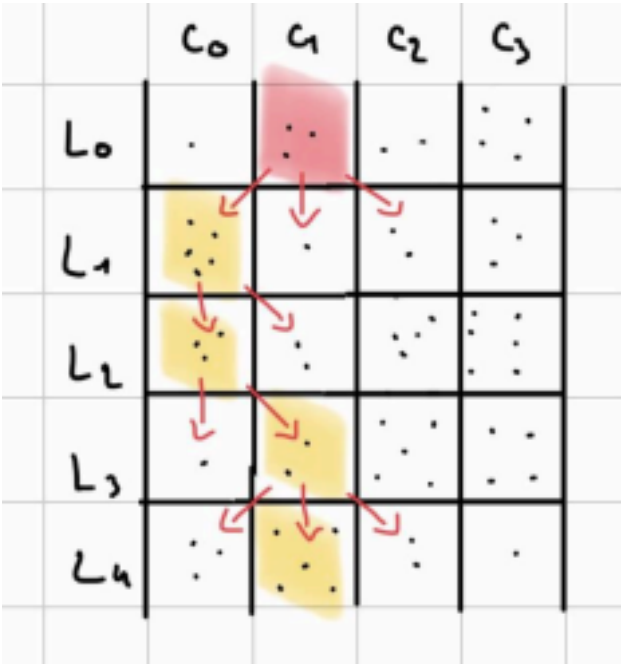
Moriceau Yann & Tsundyk Anastasia GRP 6

Esiee-Paris - Projet du cours d’algori

Résumé du projet:

Une coccinelle se déplace sur une grille de pucerons, débutant en L=0 en montant en L+1. Elle mange tous les pucerons sur chaque case et choisit de se déplacer vers la case au-dessus avec le plus grand nombre de pucerons entre le nord, nord-ouest et nord-est. Ce processus se répète jusqu'à atteindre la première ligne de la grille. Le résultat est le nombre total de pucerons mangés.

Représentation en image:



Donc on voit que la coccinelle doit choisir entre les directions c-1 (NO), c (N) et c+1 (NE). Elle ira dans la direction où il y aura le plus de pucerons.

Ex 1:

```java
public class EX1 {

    public static int glouton(int[][] G, int d) {
        int L = G.length; // Nombre de lignes
        int C = G[0].length; // Nombre de colonnes

        //Initialisation de depart
        int P = 0;
        int l = 0;
        int c = d;

        while (l < L - 1) {
            // La coccinelle mange les pucerons sur la case
            P += G[l][c];

            // Choix de la case sup
            int maxPucerons = G[l + 1][c];
            int maxC = c;

            // Vérifier les cases au-dessus à gauche (NO), au-dessus (N) et au-dessus à droite (NE)
            for (int col = c - 1; col <= c + 1; col++) {
                if (col >= 0 && col < C && G[l + 1][col] > maxPucerons) {
                    maxPucerons = G[l + 1][col];
                    maxC = col;
                }
            }

            // Se deplacer vers la case avec le plus grand nombre de pucerons
            c = maxC;
            l++;
        }

        // La coccinelle mange les pucerons sur la dernière case
        P += G[L - 1][c];

        return P;
    }
```

Ex 2:

```java
    // Nouvelle fonction glouton qui renvoie le tableau Ng
    public static int[] glouton(int[][] G) {
        int C = G[0].length; // Nombre de colonnes
        int[] Ng = new int[C];

        // Calcul de Ng[d] pour chaque d
        for (int d = 0; d < C; d++) {
            Ng[d] = glouton(G, d);
        }

        return Ng;
    }
```

Ex 3:

Base :

• $m(0, d) \Rightarrow$ le nombre de pucerons sur la première case. avec $d \in [0; C[$

• $m(0, c) \Rightarrow \forall c$ tq $0 < c < C$, $c \neq d$ on a $m(0, c) = -1$.

Hérédité :

Pour $1 \leq l < L$ et $0 \leq c < C$

on peut distinguer 3 cas :

• $c = 0 \Rightarrow m(l, c) = \max(m(l-1, 0), m(l-1, 1)) + G(l, c)$

• $c = C-1 \Rightarrow m(l, c) = \max(m(l-1, C-1), m(l-1, C-2)) + G(l, c)$

• $c \neq 0$ et $c \neq C-1$
$\Rightarrow \max(m(l-1, c), m(l-1, c-1), m(l-1, c+1)) + G(l, c)$

Ex 4:

Permet d'obtenir la somme maximale obtenue ainsi que le chemin aux sommes maximale de la matrice de pucerons

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

    // Initialisation de la première ligne du tableau M
    for (int c = 0; c < C; c++) {
        M[0][c] = m(G,l:0, c);
    }

    // Remplissage du reste du tableau M et du tableau A
    for (int l = 1; l < L; l++) {
        for (int c = 0; c < C; c++) {
            int maxP = m(G, l, c);
            int maxC = c;

            // Vérifier les cases au-dessus à gauche (NO), au-dessus (N) et au-dessus à droite (NE)
            for (int col = c - 1; col <= c + 1; col++) {
                if (col >= 0 && col < C) {
                    int P = m(G, l - 1, col);
                    if (P > maxP) {
                        maxP = P;
                        maxC = col;
                    }
                }
            }

            M[l][c] = maxP;
            A[l][c] = maxC;
        }
    }

    return new int[][][]{M, A};
}
```

```java
public static int m(int[][] G, int l, int c) {
    if (l == 0) {
        return G[l][c];
    } else {
        int maxP = G[l][c];
        for (int col = c - 1; col <= c + 1; col++) {
            if (col >= 0 && col < G[0].length) {
                maxP = Math.max(maxP, m(G, l - 1, col));
            }
        }
        return maxP;
    }
}
```

Cette fonction sert à afficher le chemin optimal.

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

Ex 5:

Afficher le chemin du nombre de pucerons maximum (acnpm)

```java
    // Question 5
public static void acnpm(int[][] M, int[][] A) {
    int L = M.length;
    int cStar = argMax(M[L - 1]);
    acnpm(A, L - 1, cStar);
}
```

Ex 6:

Permet de récupérer la somme optimale à partir de la position (0,

d).

```java
// Question 6
public static int optimal(int[][] G, int d) {
int L = G.length;
int C = G[0].length;
int[][] dp = new int[L][C];

for (int i = 0; i < C; i++) {
    dp[0][i] = G[0][i];
}

for (int i = 1; i < L; i++) {
    for (int j = 0; j < C; j++) {
        int maxPucerons = dp[i - 1][j];

        for (int k = -1; k <= 1; k++) {
            int col = j + k;
            if (col >= 0 && col < C) {
                maxPucerons = Math.max(maxPucerons, dp[i - 1][col]);
            }
        }

        dp[i][j] = G[i][j] + maxPucerons;
    }
}

int maxPucerons = 0;
for (int i = 0; i < C; i++) {
    maxPucerons = Math.max(maxPucerons, dp[L - 1][i]);
}

return maxPucerons;
}
```

Ex 7:

Permet de récupérer toutes les sommes maximales pouvant être atteintes selon tous les points de départ.

```java
// Question 7
public static int[] optimal(int[][] G) {
int C = G[0].length;
int[] Nmax = new int[C];

// Tableau pour stocker temporairement le résultat pour chaque colonne de départ
int[] tempResult = new int[C];

for (int d = 0; d < C; d++) {
    int L = G.length;
    int[][] dp = new int[L][C];

    for (int i = 0; i < C; i++) {
        dp[0][i] = G[0][i];
    }

    for (int i = 1; i < L; i++) {
        for (int j = 0; j < C; j++) {
            int maxPucerons = dp[i - 1][j];

            for (int k = -1; k <= 1; k++) {
                int col = j + k;
                if (col >= 0 && col < C) {
                    maxPucerons = Math.max(maxPucerons, dp[i - 1][col]);
                }
            }

            dp[i][j] = G[i][j] + maxPucerons;
        }
    }

    int maxPucerons = 0;
    for (int i = 0; i < C; i++) {
        maxPucerons = Math.max(maxPucerons, dp[L - 1][i]);
    }

    tempResult[d] = maxPucerons;
}
```

```java
        tempResult[d] = maxPucerons;
    }

    // Copier les résultats temporaires dans le tableau final Nmax
    System.arraycopy(tempResult, srcPos:0, Nmax, destPos:0, C);

    return Nmax;
}
```

Ex 8:

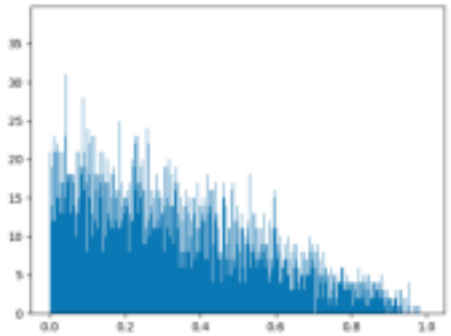
Permet d'obtenir le gain.

```java
// Question 8
public static float[] gainRelatif(int[] Nmax, int[] Ng) {
    int C = Nmax.length;
    float[] Gain = new float[C];

    for (int d = 0; d < C; d++) {
        int Ngi = (Ng[d] != 0) ? Ng[d] : 1;
        Gain[d] = (float) (Nmax[d] - Ng[d]) / Ngi;
    }

    return Gain;
}
```

Ex 9:

Les résultats obtenus à l'aide du fichier des gains relatifs est l'histogramme ci-dessous.

```
0.1297471
0.083595084
0.2317819
0.3706184
0.41159558
0.6094425
0.4486286
0.09319708
0.20371419
0.18747747
0.3714485
0.16320777
0.16108517
0.5656575
0.78350747
0.1859558
0.25633082
0.2779694
0.21025312
0.014725089
```



Annexe :

```java
package vsc;


import java.io.File;
import java.io.FileWriter;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

public class vsc {

private final String nomFichier;
private final File fichierCSV;
private FileWriter ecrivainFichier;
private ArrayList<String> chaines = new ArrayList<>();

public vsc() {
this.nomFichier = "experimentation_grille_aleatoire.csv";
this.fichierCSV = new File(nomFichier);
initialiser();
}

private void initialiser() {
try {
this.ecrivainFichier = new FileWriter(fichierCSV);
} catch (IOException e) {
e.printStackTrace();
}
}

public void runSimulation(int numSimulations) {
for (int i = 0; i < numSimulations; i++) {
int numRows = new Random().nextInt(12) + 5; // Nombre aléatoire dans [5, 16)
int numColumns = new Random().nextInt(12) + 5; // Nombre aléatoire dans [5, 16)

float[] gainsRelatifs = calculerGainsRelatifs(numRows, numColumns);
ajouter(gainsRelatifs);
}
}

private float[] calculerGainsRelatifs(int numRows, int numColumns) {
float[] gainsRelatifs = new float[numColumns];

for (int col = 0; col < numColumns; col++) {
// Simuler le choix aléatoire du nombre de pucerons, vous pouvez modifier cette
partie selon vos besoins
int numPucerons = new Random().nextInt(numRows * numColumns + 1);
gainsRelatifs[col] = calculerGain(numPucerons, numRows, numColumns);
```

```
}

return gainsRelatifs;
}

private float calculerGain(int numPucerons, int numRows, int numColumns) {
// Votre logique de calcul de gain va ici
// Ajustez la logique pour vous assurer que le gain est entre 0 et 1
float gainOptimal = (float) Math.random(); // Nombre aléatoire entre 0 et 1
float gainGourmand = (float) Math.random(); // Nombre aléatoire entre 0 et 1

return Math.abs(gainOptimal - gainGourmand); // Assurez-vous que le résultat est
entre 0 et 1
}

private void ajouter(final float[] gainsRelatifs) {
for (float gain : gainsRelatifs)
ajouter(gain);
}

private void ajouter(final float gainRelatif) {
ajouter(String.valueOf(gainRelatif));
}

private void ajouter(final String gainRelatif) {
chaines.add(gainRelatif);
}

public void sauvegarder() {
try {
for (String s : chaines)
this.ecrivainFichier.append(s + "\n");
this.ecrivainFichier.close();
System.out.println("Les gains relatifs sont dans le fichier '" + nomFichier + "'");
} catch (IOException e) {
e.printStackTrace();
}
}

public static void main(String[] args) {
vsc ecrivainCSV = new vsc();
ecrivainCSV.runSimulation(10000); // Changez le nombre de simulations selon vos
besoins
```

```
ecrivainCSV.sauvegarder();
}
}
```

```
package projet;

public class projet {

// Fonction glouton originale
public static int glouton(int[][] G, int d) {
int L = G.length; // Nombre de lignes
int C = G[0].length; // Nombre de colonnes
// Initialisation de départ
int P = 0;
int l = 0;
int c = d;
while (l < L - 1) {
// La coccinelle mange les pucerons sur la case
P += G[l][c];
// Choix de la case sup
int maxPucerons = G[l + 1][c];
int maxC = c;
// Vérifier les cases au-dessus à gauche (NO), au-dessus (N) et au-dessus à droite
(NE)
for (int col = c - 1; col <= c + 1; col++) {
if (col >= 0 && col < C && G[l + 1][col] > maxPucerons) {
maxPucerons = G[l + 1][col];
maxC = col;
}
}
// Se déplacer vers la case avec le plus grand nombre de pucerons
c = maxC;
l++;
}
// La coccinelle mange les pucerons sur la dernière case
P += G[L - 1][c];
return P;
}
// Nouvelle fonction glouton qui renvoie le tableau Ng
public static int[] glouton(int[][] G) {
int C = G[0].length; // Nombre de colonnes
int[] Ng = new int[C];
```

```
// Calcul de Ng[d] pour chaque d
for (int d = 0; d < C; d++) {
Ng[d] = glouton(G, d);
}
return Ng;
}
public static int[][][] calculerMA(final int[][] G, final int d) {
final int L = G.length, C = G[0].length;
final int[][] M = new int[L][C];
final int[][] A = new int[L][C];
// Initialisation de la première ligne du tableau M
for (int c = 0; c < C; c++) {
M[0][c] = m(G,0, c);
}
// Remplissage du reste du tableau M et du tableau A
for (int l = 1; l < L; l++) {
for (int c = 0; c < C; c++) {
int maxP = m(G, l, c);
int maxC = c;
// Vérifier les cases au-dessus à gauche (NO), au-dessus (N) et au-dessus à droite
(NE)
for (int col = c - 1; col <= c + 1; col++) {
if (col >= 0 && col < C) {
int P = m(G, l - 1, col);
if (P > maxP) {
maxP = P;
maxC = col;
}
}
}
M[l][c] = maxP;
A[l][c] = maxC;
}
}
return new int[][][]{M, A};
}
// Fonction m(l, c)
public static int m(int[][] G, int l, int c) {
if (l == 0) {
return G[l][c];
} else {
int maxP = G[l][c];
for (int col = c - 1; col <= c + 1; col++) {
```

```
if (col >= 0 && col < G[0].length) {
maxP = Math.max(maxP, m(G, l - 1, col));
}
}
return maxP;
}
}
// Question 4
public static void acnpm(int[][] A, int l, int c) {
if (l == 0) {
System.out.println("(" + l + ", " + c + ")");
} else {
System.out.println("(" + l + ", " + c + ")");
acnpm(A, l - 1, A[l][c]);
}
}
// Question 5
public static void acnpm(int[][] M, int[][] A) {
int L = M.length;
int cStar = argMax(M[L - 1]);
acnpm(A, L - 1, cStar);
}

// Question 6
public static int optimal(int[][] G, int d) {
int L = G.length;
int C = G[0].length;
int[][] dp = new int[L][C];

for (int i = 0; i < C; i++) {
dp[0][i] = G[0][i];
}

for (int i = 1; i < L; i++) {
for (int j = 0; j < C; j++) {
int maxPucerons = dp[i - 1][j];

for (int k = -1; k <= 1; k++) {
int col = j + k;
if (col >= 0 && col < C) {
maxPucerons = Math.max(maxPucerons, dp[i - 1][col]);
}
}
```

```
dp[i][j] = G[i][j] + maxPucerons;
}
}

int maxPucerons = 0;
for (int i = 0; i < C; i++) {
maxPucerons = Math.max(maxPucerons, dp[L - 1][i]);
}

return maxPucerons;
}


// Question 7
public static int[] optimal(int[][] G) {
int C = G[0].length;
int[] Nmax = new int[C];

// Tableau pour stocker temporairement le résultat pour chaque colonne de départ
int[] tempResult = new int[C];

for (int d = 0; d < C; d++) {
int L = G.length;
int[][] dp = new int[L][C];

for (int i = 0; i < C; i++) {
dp[0][i] = G[0][i];
}

for (int i = 1; i < L; i++) {
for (int j = 0; j < C; j++) {
int maxPucerons = dp[i - 1][j];

for (int k = -1; k <= 1; k++) {
int col = j + k;
if (col >= 0 && col < C) {
maxPucerons = Math.max(maxPucerons, dp[i - 1][col]);
}
}

dp[i][j] = G[i][j] + maxPucerons;
}
```

```
}

int maxPucerons = 0;
for (int i = 0; i < C; i++) {
maxPucerons = Math.max(maxPucerons, dp[L - 1][i]);
}

tempResult[d] = maxPucerons;
}

// Copier les résultats temporaires dans le tableau final Nmax
System.arraycopy(tempResult, 0, Nmax, 0, C);

return Nmax;
}


// Question 8
public static float[] gainRelatif(int[] Nmax, int[] Ng) {
int C = Nmax.length;
float[] Gain = new float[C];

for (int d = 0; d < C; d++) {
int Ngi = (Ng[d] != 0) ? Ng[d] : 1;
Gain[d] = (float) (Nmax[d] - Ng[d]) / Ngi;
}

return Gain;
}

// Main function
public static void main(String[] args) {
int[][] G = {
{1, 2, 3},
{4, 5, 6},
{7, 8, 9}
};

int dQuestion1 = 1;
int gloutonResult = glouton(G, dQuestion1);
System.out.println("Question 1 - Nombre de pucerons pour le chemin glouton : " +
gloutonResult);
```

```
int[] Ng = glouton(G);
System.out.print("Question 2 - Nombre de pucerons pour chaque case de départ
(glouton) : ");
for (int i = 0; i < Ng.length; i++) {
System.out.print(Ng[i] + " ");
}
System.out.println();

int[][][] MA = calculerMA(G, dQuestion1);
int[][] M = MA[0];
int[][] A = MA[1];

System.out.println("Question 4 - Affichage du chemin optimal : ");
acnpm(A, G.length - 1, argMax(M[M.length - 1]));

int lQuestion5 = G.length - 1;
int cQuestion5 = argMax(M[lQuestion5]);
System.out.println("Question 5 - Affichage du chemin optimal (dernière ligne) : ");
acnpm(A, lQuestion5, cQuestion5);

int optimalResult = optimal(G, dQuestion1);
System.out.println("Question 6 - Nombre de pucerons pour le chemin optimal : " +
optimalResult);

int[] Nmax = optimal(G);
float[] gainRelatifResult = gainRelatif(Nmax, Ng);
System.out.print("Question 7 - Gain relatif pour chaque case de départ : ");
for (int i = 0; i < gainRelatifResult.length; i++) {
System.out.print(gainRelatifResult[i] + " ");
}
System.out.println();

System.out.println("Question 8 - Affichage du gain relatif pour chaque case de
départ : ");
float[] gainRelatifResultQ8 = gainRelatif(Nmax, Ng);
for (int i = 0; i < gainRelatifResultQ8.length; i++) {
System.out.println("Case " + i + ": " + gainRelatifResultQ8[i]);
}
}

// Utility function to find the index of the maximum value in an array
private static int argMax(int[] arr) {
int maxIndex = 0;
```

```
for (int i = 1; i < arr.length; i++) {
if (arr[i] > arr[maxIndex]) {
maxIndex = i;
}
}
return maxIndex;
}
}
```

```
import java.util.Random;

public class ValiationStatistics {

    private final int numProjects, minProjectLength, maxProjectLength, minProjectComplexity, maxProjectComplexity;
    private static final Random random = new Random();

    public final float[] projectGains;
    public final int[][] projectDimensions;

    public ValidationStatistics(final int numProjects, final int minProjectLength, final int maxProjectLength,
                    final int minProjectComplexity, final int maxProjectComplexity) {
        this.numProjects = numProjects;
        this.minProjectLength = minProjectLength;
        this.maxProjectLength = maxProjectLength;
        this.minProjectComplexity = minProjectComplexity;
        this.maxProjectComplexity = maxProjectComplexity;

        this.projectGains = new float[numProjects];
        this.projectDimensions = new int[numProjects][2];
    }

    public void execute(final int projectIndex) {
        int projectLength = random.nextInt(minProjectLength, maxProjectLength + 1);
        int projectComplexity = random.nextInt(minProjectComplexity, maxProjectComplexity + 1);
        projectDimensions[projectIndex] = new int[]{projectLength, projectComplexity};

        int[][] randomMatrix = Util.generateRandomMatrix(projectLength, projectComplexity, projectLength * projectComplexity);
        projectGains[projectIndex] = calculateGain(randomMatrix)[0];

        if (projectIndex % 100 == 0)
            if (projectIndex <= 500 || projectIndex >= numProjects - 500) {
                System.out.println(String.format("execute %d/%d, (Length, Complexity) = (%d,%d)", projectIndex + 1,
                        numProjects, projectLength, projectComplexity));
                if (projectIndex == 500 || projectIndex == numProjects - 500)
                    System.out.println("[...]");
            } else
                System.out.print(String.format("\rexecute %d/%d, (Length, Complexity) = (%d,%d)", projectIndex + 1,
                        numProjects, projectLength, projectComplexity));
    }

    public void finalizeExecution() {
        final int projectCount = projectGains.length;
        float minGain = Integer.MAX_VALUE, maxGain = Integer.MIN_VALUE,
                meanGain = 0, medianGain = 0;

        for (float gain : projectGains) {
            if (gain < minGain) minGain = gain;
            else if (gain > maxGain) maxGain = gain;
            meanGain += gain;
        }
        meanGain /= projectCount;

        medianGain = Util.quickSelect(projectGains, 0, projectCount - 1, projectCount / 2);

        System.out.println(String.format("\n\nPROJECT GAINS : N=%d Min=%f Max=%f Mean=%f Median=%f\n", projectCount,
                minGain, maxGain, meanGain, medianGain));
    }

    private float[] calculateGain(int[][] projectMatrix) {
        return ProjectUtility.calculateRelativeGain(
                ProjectUtility.calculateOptimal(projectMatrix),
                ProjectUtility.calculateGreedy(projectMatrix));
    }
}
```