# Esiee-Paris - Projet du cours d'algorithmique

Novembre 2023 – R. Natowicz, I. Alamé, A. Çela, X. Hilaire, T. Wu, W. Xu

Temps de travail estimé : 12 heures par binôme dont 2 en travaux dirigés. Date de remise à fixer.

**Coccinelles et pucerons.** Sur chaque case d'une grille à $L$ lignes et $C$ colonnes il y a des petits pucerons en quantités variées. Ce sont les pucerons dont les coccinelles raffolent. Et, surprise, qui voit-on sur l'une des cases de la dernière ligne ? Une coccinelle occupée à manger le dernier puceron de la case.
À bien y regarder on voit que certaines des cases de la grille n'ont plus de pucerons. La coccinelle les a mangés.
Un reporter du journal La Hulotte (Le journal le plus lu dans les terriers, `www.lahulotte.fr`) s'est rendu sur place depuis Boult-aux-bois pour en avoir le cœur net. Il s'est entretenu avec la coccinelle. Elle lui a déclaré : « Je me suis posée sur une des cases du bas de la grille et j'ai mangé tous les pucerons qui s'y trouvaient. Excellents ! Puis avec mes antennes j'ai touché les cases situées juste au-dessus (*la coccinelle parle des cases direction Nord (N), Nord-Ouest (NO) et Nord-Est (NE). En général il y a trois cases, parfois deux.*) Je me suis déplacée sur la case qui avait le plus de pucerons. Je les ai tous mangés. Et j'ai continué comme ça jusqu'à la dernière ligne. Quel festin ! »

Coccinelle occupée à manger le dernier puceron d'une case $(L-1, c)$

Le journaliste a conclu son article de façon ironique : « Mignonne, mais gloutonne. »

Q 1. Écrire une fonction `int glouton(int[][] G, int d)` qui retourne le nombre de pucerons qu'une coccinelle ayant atterri sur la case $(0, d)$ mangera sur son chemin glouton ;

Q 2. Écrire une fonction `int glouton(int[][] G)` qui prend en entrée la grille de pucerons $G$ et retourne un tableau $N_g[0:C]$ de terme général $N_g[d] = n_g(d) =$ `glouton(G, d)`.

Si la coccinelle avait suivi le cours d'algorithmique de *Sup' de Cocc* elle aurait procédé différemment.

a) Elle aurait défini la fonction $m(l, c)$, nombre total maximum de pucerons sur un chemin allant de la case de départ $(0, d)$ sur laquelle elle a choisi de se poser, jusqu'à la case $(l, c)$.

b) Elle aurait scruté la grille depuis les airs et, tout en battant des ailes, elle aurait calculé un tableau $M[0:L][0:C]$ de terme général $M[l][c] = m(l, c)$.
Le maximum de la dernière ligne du tableau $M$ est le nombre maximum de pucerons que la coccinelle pourra manger en ayant atterri sur la case $(0, d)$. De plus, la case $(L-1, c^*)$ qui contient cette valeur maximum sera la case d'arrivée de la coccinelle. Le numéro de colonne $c^*$ de cette case est $c^* = \arg\max_{0 \le c < C} m(L-1, c)$.

c) Toujours en l'air, la coccinelle aurait construit le chemin à nombre de pucerons maximum. Ce chemin va de la case $(0, d)$ sur laquelle elle se posera, jusqu'à la case $(L-1, c^*)$.

d) Puis elle se serait posée sur la case $(0, d)$ choisie. Elle aurait rangé ses ailes dans leur joli coffret rouge à points noirs et aurait parcouru le chemin maximum en se délectant des pucerons de chacune de ses cases.

Q 3. Donner l'équation de récurrence des valeurs $m(l, c)$ :
– base : valeur $m(0, d)$ et valeurs $m(0, c)$, $0 \le c < C$, $c \ne d$ ;
– hérédité : valeurs $m(l, c)$, $1 \le l < L$, $0 \le c < C$ ;

Q 4. calculer le tableau $M[0:L][0:C]$ de terme général $M[l][c] = m(l,c)$ et le tableau $A[0:L][0:C]$ dont le terme général $A[l][c] = a(l,c)$ est l'indice de la colonne qui précède la case $(l,c)$ sur le chemin maximum. La fonction `int[][][] calculerMA(int[][] G, int d)` calculera les deux tableaux $M$ et $A$ et les retournera dans un tableau `int[][][] MA = {M, A}`.

On s'intéresse à présent à l'affichage d'un chemin à nombre de pucerons maximum, de la case d'atterissage $(0, d)$ jusqu'à la case $(L-1, c^*)$. C'est ce que réalise la fonction ci-dessous.

```
void acnpm(int[][] M, int[][] A){
   int cStar = argMax(M[L-1]) ; // colonne d'arrivée du chemin max. d'origine (0,d)
   acnpm(A, L-1, cStar); // affichage du chemin maximum de (0,d) à (L-1, cStar)
}
```

Q 5. Écrire la fonction `void acnpm(int[][] A, int l, int c)`.

À la question 1. vous avez écrit la fonction `int glouton(int[][] G, int d)` qui retournait le nombre de pucerons qu'une coccinelle ayant atterri sur la case $(0, d)$ mange sur son chemin glouton, et la fonction `int glouton(int[][] G)` qui prenait en entrée la grille de pucerons $G$ et retournait un tableau $N_g[0:C]$ de terme général $N_g[d] = n_g(d) =$ `glouton(G, d)`. Les mêmes questions se posent pour les chemins optimaux.

Q 6. Écrire la fonction `int optimal(int[][] G, int d)` qui retourne le nombre de pucerons qu'une coccinelle ayant atterri sur la case $(0, d)$ mangera sur le chemin à nombre de pucerons maximum.

Q 7. Écrire la fonction `int optimal(int[][] G)` qui prend en entrée la grille $G$ et retourne le tableau $N_{\max}[0:C]$ de terme général $N_{\max}[d] = n_{\max}(d)$, nombre de pucerons que la coccinelle qui a atteri sur case $(0,d)$ mangera sur le chemin à nombre de pucerons maximum.

Comparaison des stratégies optimale et gloutonne sur un exemple.

Q 8. Écrire une fonction `float gainRelatif(int[] Nmax, int[] Ng)` qui retourne un tableau `float Gain[0:C]` contenant, pour toute case $(0, d)$ de départ, le gain relatif de la stratégie optimale sur la stratégie gloutonne. Cette fonction prend en entrée les tableaux $N_{\max}[0:C]$ et $N_g[0:C]$ des questions précédentes. Elle retourne le tableau `Gain[0:C]` de terme général $gain(d) = \frac{n_{\max}(d) - n_{g((d)}}{n_g(d)}$.

Calculer le tableau des gains relatifs pour la grille de pucerons ci-dessous.

| | | | | | |
|---|---|---|---|---|---|
| 7 | 3 | 1 | 2 | 4 | 5 |
| 6 | 1 | 72 | 3 | 6 | 6 |
| 5 | 89 | 27 | 10 | 12 | 3 |
| 4 | 46 | 2 | 8 | 7 | 15 |
| 3 | 36 | 34 | 1 | 13 | 30 |
| 2 | 2 | 4 | 11 | 26 | 66 |
| 1 | 1 | 10 | 15 | 1 | 2 |
| $l = 0$ | 2 | 4 | 3 | 9 | 6 |
| | $c = 0$ | 1 | 2 | 3 | 4 |

TAB. 1 – *Nombres de pucerons sur les cases de la grille avant l'arrivée de la coccinelle par la voie des airs.*

**Validation statistique.**
Au-delà de cet exemple, nous voulons quantifier le gain relatif de la stratégie optimale sur la stratégie gloutonne. Pour ce faire nous mettons en œuvre une *validation statistique*. Pour le problème que nous étudions, une expérience élémentaire de validation statistique (un *run*) est le choix au hasard des nombres $L$ et $C$ de lignes et colonnes de la grille, par exemple $L$ au hasard dans l'intervalle $[5:16]$ $(0 \leq L < 16)$ et de même pour $C$ ; et pour chaque case de la grille, le choix au hasard du nombre de pucerons, par exemple au hasard dans l'intervalle $[0 : L + C]$ (beaucoup de répétitions de valeurs) ou au hasard dans l'intervalle $[0 : L \times C]$ (peu de répétitions) ou une grille contenant une permutation aléatoire des valeurs de l'intervalle $[0 : L \times C]$ (aucune répétition.)
Pour cette grille aléatoire, vous calculez le tableau `Gain[0:C]` des gains relatifs.

Cette expérience élémentaire est répétée un grand nombre de fois, par exemple $n = 10^3$ fois ou $n = 10^4$ fois ou plus. À la fin de cette validation vous disposez d'un grand nombre de valeurs de gains relatifs. Vous pouvez en afficher l'histogramme donner des valeurs statistiques caractéristiques de leur distribution : médiane des gains relatifs, moyenne, déviation standard (écart-type), ...

Remarque : sur des petites grilles, les valeurs gloutonne et optimales sont très souvent égales. On recommande alors d'afficher l'histogramme des seules valeurs non nulles de grain relatif car l'histogramme de toutes les valeurs, nulles et non nulles, est difficile à lire.

Q 9. *Just do it!*

Vous pourriez avoir besoin d'une fonction qui calcule une permutation aléatoire des valeurs d'un tableau. En voici une.

```
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;
}
```

Quatre binômes de coccinelles élaborant des stratégies

On peut remarquer que pour toute case de départ $(0, d)$ la plus grande partie du tableau $M[0:L][0:C]$ est inutilisée car le nombre de cases atteignables depuis la case $(0,d)$ est faible comparé au nombre de cases de la grille.

La structure de données Arbres ternaires permet d'obtenir un programme plus efficace en taille mémoire et temps de calcul. En voici définition :

```
class AT{ int c; // numéro de colonne de la présente case
   int mlc; // mlc = m(l, c), nb de pucerons sur ch. max. "(0,d) -----> présente case"
   AT ne, n, no; // arbres ternaires des cases situées au Nord Est, Nord et Nord Ouest
   AT cp; // case qui précède la présente case sur le chemin maximum
```

La racine de cet arbre a les valeurs $c = d$, $mlc = G[0][d]$, et $cp =$ `null` car le chemin de valeur maxium commence en case $(0,d)$.
Les arbres ternaires $ne$, $n$ et $no$ des cases situées hors de la grille sont des arbres vides.
Les numéros de colonne des arbres ternaires non vides $ne$, $n$ et $no$ sont $c+1$, $c$, et $c-1$ respectivement.

Vous avez toutes les connaissances nécessaires pour écrire les algorithmes précédents avec cette structure de données d'arbres ternaires mais, bonne nouvelle, ça n'est pas demandé dans le projet.

## Annexe: trace d'une exécution du programme:

```
% java Projet_2024
grille G:
G[2] : [1, 1, 10, 1, 1]
G[1] : [6, 5, 1, 2, 8]
G[0] : [2, 2, 3, 4, 2]
Valeurs des chemins gloutons depuis les cases (0,d) : Ng = [9, 9, 18, 13, 11]
Programmation dynamique, case de départ (0, 0)
M :
M[2] : [9, 9, 17, -1, -1]
M[1] : [8, 7, -1, -1, -1]
M[0] : [2, -1, -1, -1, -1]
un chemin maximum : (0,0)(1,1)(2,2) Valeur : 17
Programmation dynamique, case de départ (0, 1)
M :
M[2] : [9, 9, 17, 4, -1]
M[1] : [8, 7, 3, -1, -1]
M[0] : [-1, 2, -1, -1, -1]
un chemin maximum : (0,1)(1,1)(2,2) Valeur : 17

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

Ng = [9, 9, 18, 13, 11]
Nmax = [17, 17, 18, 16, 14]
Gains relatifs = [0.888, 0.888, 0.0, 0.23, 0.272]

VALIDATION STATISTIQUE
nruns=10000
L au hasard dans [5:16]
C au hasard dans [5:16]
Nb. de pucerons / case au hasard dans [0:L+C]
run 1/10000, (L,C) = (13,9)
run 101/10000, (L,C) = (6,9)
run 201/10000, (L,C) = (14,7)
run 301/10000, (L,C) = (13,13)
run 401/10000, (L,C) = (15,11)
run 501/10000, (L,C) = (6,15)
[...]
run 9501/10000, (L,C) = (12,5)
run 9601/10000, (L,C) = (5,5)
run 9701/10000, (L,C) = (7,8)
run 9801/10000, (L,C) = (10,12)
run 9901/10000, (L,C) = (15,8)
GAINS.length=100653, min=0,000000, max=1,500000, mean=0,051199, med=0,025641
Les gains relatifs sont dans le fichier gainsRelatifs_nruns=10000_Linf=5_Lsup=16_Cinf=5_Csup=16.csv
%
```

## Génération de l'histogramme des gains relatifs

```
% python3 histogramme.py gainsRelatifs_nruns=10000_Linf=5_Lsup=16_Cinf=5_Csup=16
100653 valeurs dans le fichier des gains relatifs
35921 valeurs 0.0 dans le fichier des gains relatifs
64732 valeurs non nulles dans le fichier des gains relatifs
valeur non nulle minimum : 0.0026455026
valeur non nulle maximum : 1.5
creation de l'histogramme des valeurs non nulles de gain relatif
l'histogramme est dans le fichier gainsRelatifs_nruns=10000_Linf=5_Lsup=16_Cinf=5_Csup=16.png
%
```