Méthodes de Conception d'Algorithmes

C. TRABELSI & E. MARTINS & M. FRANÇOIS

TD4-5 - - Récursivité

Avant propos

La récursivité se définie par la répétition d'un motif par lui-même. Elle s'oppose à la répétition d'un motif par une structure de contrôle (for, while, do while) que l'on appelle processus itératif. La récursivité permet la déclaration élégante de programmes complexes.

1 Inversion des éléments dans un tableau 1D

On considère un tableau d'entiers de longueur L. Le but est d'écrire une fonction récursive permettant d'inverser l'ordre des éléments entre l'indice I et l'indice $L - 1 - I$. Cette fonction sera appelée `INVERSER_TAB_REC(TAB, L, I)`.
Par exemple si le tableau est :

0	1	2	3	4	5	6	7	8	9

alors `INVERSER_TAB_REC(TAB, 10, 0)` va donner :

9	8	7	6	5	4	3	2	1	0

ou encore `INVERSER_TAB_REC(TAB, 10, 3)` va donner :

0	1	2	6	5	4	3	7	8	9

► **1.** Écrire l'algorithme permettant de réaliser ce travail.

► **2.** Supposons que le tableau initial est donné par :

23	5184	94	2004	10

Dans ce cas, quel est le résultat pour :
`INVERSER_TAB_REC(TAB, 5, 2)` ?
`INVERSER_TAB_REC(TAB, 5, 1)` ?

► **3.** Calculer la complexité en temps de cet algorithme :

	Valeur calculée	Landau	Appellation
dans le meilleur des cas			
dans le pire des cas			

► **4.** Donner la version itérative de l'algorithme précédent.

► **5.** Lequel des deux algorithmes vous semble le plus simple ?

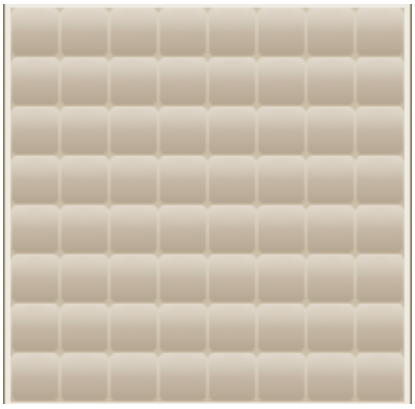
2 Problème : *les zéros en cascade du démineur*

Mines, logiciel GPL intégré dans Gnome.

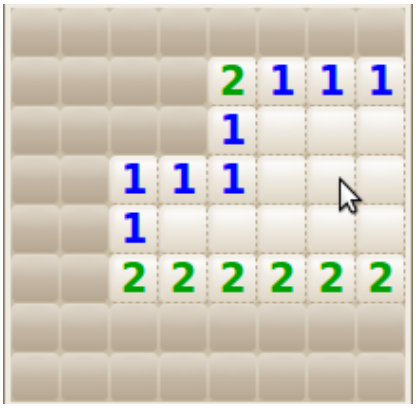
Le jeu du démineur, est un jeu de réflexion déductive où le joueur cherche à localiser en aveugle des objets (généralement des mines) à partir de l'information du nombre de mines présentes sur les cases adjacentes.

Un cas particulier de ce jeu intervient lorsque le joueur clic sur une *case Zéro*, c'est-à-dire une case n'ayant pas de mines sur les cases adjacentes. Le programme révèle alors les huit cases voisines de cette dernière.

Que se passe-t-il si les cases nouvellement révélées sont elles-aussi des *cases Zéros* ?



Avant



Après le clic : tout le voisinage des *cases Zéros* est révélé.

Quel algorithme peut-on imaginer pour réaliser cette action ?

2.1 Algorithme itératif

Il faut mémoriser l'état en plus de la valeur d'une case. Dans le démineur, on considère que l'on possède un tableau de marquage (`caché`, `révélée`, `à voir` ou `drapeau`) en plus du tableau de valeurs indiquant le nombre de mines adjacentes.

```
Marquer case cliquée comme à voir
répéter
    Affecter faux à Recommencer
    pour chaque case du tableau faire
        si (case est marquée à voir) alors
            Marquer case comme révélée
            si (case est une case Zéro) alors
                pour chaque voisin de case faire
                    si (voisin n'est pas encore révélé) alors
                        Marquer voisin comme à voir
                        Affecter vrai à Recommencer
                    fin
                fin
            fin
        fin
    fin
tant que Recommencer est vrai;
```

► **6.** On considère N comme étant le nombre total de cases du tableau. Calculer la complexité en temps de l'algorithme, (vous pouvez faire une surestimation dans le pire des cas).

	Valeur calculée	Landau	Appellation
dans le meilleur des cas			
dans le pire des cas			

► **7.** Considérons que le tableau de marque est rempli de 1 ($i.e.$ toutes les cases sont cachées) et que le joueur à cliqué sur la case (2,2), déroulez l'algorithme :

1	-1	2	-1	1
1	1	2	1	1
0	0	0	0	0
0	0	0	1	1
0	0	0	1	-1

Tableau de valeur (-1 : mine, > 0 proximité)

Tableau de marque (1 : cachée, 2 : révélée, 3 : à voir)

Remarque : *que cherche-t-on à faire ? On peut reformuler le problème de la manière suivante : si je révèle une case zéro, je révèle ses voisins. Si l'un des voisins est une case zéro, je révèle également ses voisins. Si l'un des voisins est une case zéro, je révèle aussi ses voisins, etc. Ce problème semble éminemment auto-référent, vous ne trouvez-pas ?*

2.2 Algorithme récursif

```
fonction Reveler(case)
    si (case est marquée cachée) alors
        Marquer case comme révélée
        si (case est une case Zéro) alors
            pour chaque voisin de case faire
                Reveler(voisin)
            fin
        fin
    fin


Reveler(case cliquée)
```

► **8.** Calculer la complexité en temps de l'algorithme :

	Valeur calculée	Landau	Appellation
dans le meilleur des cas			
dans le pire des cas			

► **9.** Considérons que le tableau de marque est rempli de 1 ($i.e.$ toutes les cases sont cachées) et que le joueur à cliqué sur la case (2,2), déroulez l'algorithme :

1	-1	2	-1	1
1	1	2	1	1
0	0	0	0	0
0	0	0	1	1
0	0	0	1	-1

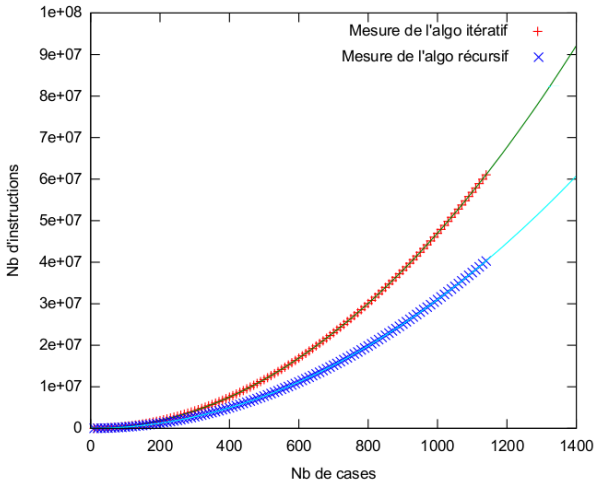
Tableau de valeur (-1 : mine, > 0 proximité)

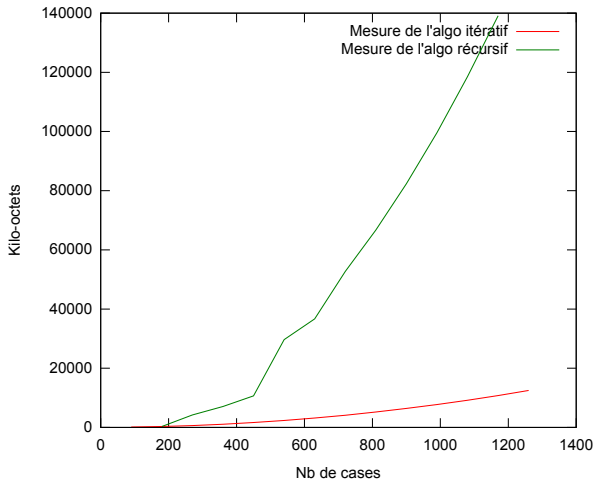
Tableau de marque (1 : cachée, 2 : révélée, 3 : à voir)

► **10.** Lequel des deux algorithmes vous semble-t-il plus lisible ?

2.3 Étude empirique

Voici quelques courbes tracées via `gnuplot` [1] à partir de données obtenues par nos soins. Elles correspondent à une estimation empirique du nombre d'instructions et de la place mémoire en fonction du nombre de case du démineur pour chacun des deux algorithmes précédents.





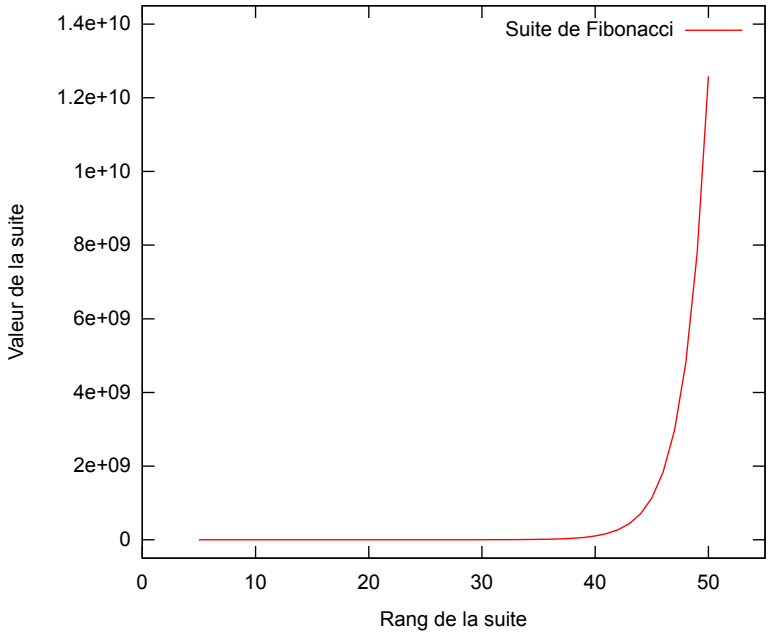
► **11.** Qu'observez-vous ?

1. GNUplot est un logiciel libre permettant des représentations graphiques 2D ou 3D de fonctions mathématiques ou de données.

► **12.** Quel semble être le problème majeur de l'approche récursive ? À quoi est-ce lié ?

3 Problème : *n-ieme terme de la suite de Fibonacci*

La suite de Fibonacci est une suite mathématique célèbre de croissance très rapide[2] et qui est probablement l'exemple le plus courant de l'utilisation de la récursivité.



Elle est définie par la formule :

$$F(n) = \begin{cases} 0 \text{ si } n = 0 \\ 1 \text{ si } n = 1 \\ F(n-1) + F(n-2) \text{ pour } n > 1 \end{cases}$$

2. `Fibonacci(1000)` = 43466557686937456435688527675040625802564660517371780402481729089536555417949051890403879840079255169295922593080322634775209689623239873322471161642996440906533187938298969649928516003704476137795166849228875

EXERCICE 1.
Écrire un programme qui :

— tant qu'il y a des nombres sur le flux d'entrée

1. récupère un nombre N positif ou nul (>=0)
2. calcule le $N^{\text{ième}}$ rang de la suite de Fibonacci par un algorithme RÉCURSIF (Pas de boucles).
3. affiche "FIBONACCI(XX) = YY" où XX est le nombre N et YY la valeur de la suite associée, suivi d'un retour à la ligne.

Le prototype de la fonction de Fibonacci devra être :

```
unsigned int FIBONACCI_REC(unsigned int N)
```

EXERCICE 2.
Écrire un programme qui :

— tant qu'il y a des nombres sur le flux d'entrée

1. récupère un nombre N positif ou nul (>=0)
2. calcule le $N^{\text{ième}}$ rang de la suite de Fibonacci par un algorithme ITÉRATIF (uniquement des boucles).
3. affiche "FIBONACCI(XX) = YY" où XX est le nombre N et YY la valeur de la suite associée, suivi d'un retour à la ligne.

Le prototype de la fonction de Fibonacci devra être :

```
unsigned int FIBONACCI_ITER(unsigned int N)
```

► **13.** Quelle est la plus grande valeur de Fibonacci (et le rang associé) représentable dans un **`unsigned int`** ?
Remarque : pour infos un `unsigned int` va de 0 à 4 294 967 295.
FIBONACCI(40) = 102 334 155 et FIBONACCCI(41) = 165 580 141.