Année académique 2023-2024

Unité Pédagogique Mathématiques
UE : Complexité des algorithmes et des problèmes

Prof. DIABY Moustapha
Dr BROU Pacôme

Parcours : Master 1
Filière : INFO, RTEL, BIHAR

Important à lire :

- Les copies conformes seront automatiquement sanctionnées d'une note de 0/20.
- Tout documents est formellement interdit
- Tout appareil électronique doit être éteint (Téléphone, Ordinateur, Tablette, etc.).

DEVOIR DE NIVEAU / Durée : 02H- Novembre 2023

Exercice 1 Question de cours

1. Définir la notion de complexité algorithmique en tenant compte des différentes types et nuances.
2. En déduire la définition de la complexité d'un problème.
3. Définir la classe des problèmes NP-complet, donner un exemple.
4. Définir le problème SAT

Exercice 2 Autour de factorielle

1. Ecrire une fonction **factorielle(k)** qui reçoit un entier k positif et qui renvoie la valeur du factoriel de k. **k !=1*2*3……*k**
2. Déterminer la complexité de la fonction **factoriel(k)**et justifier votre réponse
3. Ecrire une fonction **som_fact (L)** qui reçoit une liste L de nombres entiers positifs. Exemple : la fonction **som_fact**([5, 3, 0, 6 ,1]) donne 847 (847=5 !+3 !+ 0 !+6 !+1 !)
4. Déterminer la complexité de la fonction **som_fact**(L)et justifier votre réponse

Exercice 3

1. Enoncez le théorème de Cook
2. Démontrer que le problème 3-SAT est NP-Complet.
3. Déterminer la complexité de l'algorithme suivant. (Fournir les différentes étapes du calcul)

Algorithm 1 Insertion-sort($A = [a_1, a_2, \ldots, a_n]$)

Require: An array $A = [a_1, a_2, \ldots, a_n]$ of n numbers.
Ensure: An array $A = [a'_1, a'_2, \ldots, a'_n]$ s.t. $a'_1 \leq a'_2 \leq \ldots a'_n$ and $[a'_1, a'_2, \ldots, a'_n]$ is a permutation of $[a_1, a_2, \ldots, a_n]$.

```
1: for j = 2 to n do
2:    clef ← A[j]
3:    i ← j − 1
4:    while i > 0 and A[i] > clef do
5:       A[i + 1] ← A[i]
6:       i ← i − 1
7:    end while
8:    A[i + 1] ← clef
9: end for
```

EXERCICE 4

Le problème initialement posé par K.F. Gauss en 1842 est le suivant : est-il possible de placer 8 reines sur un échiquier sans qu'aucune reine n'en menace une autre ? Gauss trouva lui-même 72 solutions à ce problème, mais il en existe 92, si l'on ne tient pas compte des symétries naturelles de l'échiquier. Pour ceux qui ne connaissent pas les règles du jeu d'échec, du moins la partie des règles qui concernent la reine, une reine menace toutes les pièces de l'échiquier qui sont situées sur la même :

- ligne ;
- colonne ;
- diagonale.

La solution du problème est codée sous la forme d'un tableau S de taille n qui représente le n-uplet solution, c'est-à-dire que S[i] est la colonne où il faut placer la i-ème reine.

L'algorithme ***EstLibre*** teste si la case à la ligne **ligne** et à la colonne **colonne** n'est pas menacée par les reines précédemment placées. Il n'est pas nécessaire de disposer d'une matrice booléenne pour savoir quelles sont les cases menacées ou non au fur et mesure de l'évolution de l'algorithme, les positions des reines sur les lignes qui précèdent suffisent et cette information est dans la table S. La vérification se fait en observant qu'aucune reine précédente n'est placée sur la même ligne, sur la même colonne ou les mêmes diagonales.

```
1  ALGORITHME EstLibre(S,ligne,colonne):booléen
2  DONNÉES
3     ligne,colonne: entiers
4     S[1,n]: tableau d'entiers
5  VARIABLES
6     l,c: entiers
7     ok: booléen
8  DEBUT
9     ok ← VRAI
10    l ← 1
11    TQ (l ≤ ligne) FAIRE
12       c ← S[l]
13       ok ← ok ET (|ligne - l| ≠ |colonne - c|) ET (colonne ≠ c)
14       l ← l + 1
15    FTQ
16    RETOURNER ok
17 FIN
```

1. Montrez que la complexité de l'algorithme ***EstLibre*** est linéaire en la taille n de l'échiquier.

2. L'algorithme ***PlacerReine*** est chargé de placer les reines les unes après les autres en parcourant implicitement l'arbre de décision. L'algorithme ne s'arrête pas dès que les n reines sont placées mais poursuit son investigation dans l'arbre jusqu'au bout. Les n alternatives de placement d'une reine sur une ligne se concrétisent dans l'algorithme par une boucle suivie d'un test pour vérifier que la case candidate n'est pas menacée. Le premier appel de l'algorithme se fait avec la valeur 1.

```
ALGORITHME PlacerReine(ligne)
DONNÉES
   ligne: entier
VARIABLES GLOBALES
   n: entier
   S: tableau de taille n
VARIABLES
   colonne: entier
DEBUT
   SI (ligne > n) ALORS
      Traiter(S)
   SINON
      colonne ← 1
      TQ (colonne ≤ n) FAIRE
         SI EstLibre(S,ligne,colonne) ALORS
            S[ligne] ← colonne
            PlacerReine(ligne + 1)
            S[ligne] ← 0
         FSI
         colonne ← colonne + 1
      FTQ
   FSI
FIN
```

Stratégie récursive consiste à : « Sachant qu'il y a des reines bien placées dans les lignes 1 à k − 1 et connaissant leur place, trouver toutes les solutions possibles en plaçant convenablement les reines de k à n. »

L'algorithme `placerRN(k)` qui tente de placer la `k`-ème reine est alors :

- Si `k=n` alors écrire la solution obtenue.
- Sinon Parcourir toutes les cases de la ligne `k` :
  - Si une case est libre (c.-à-d. non condamnée par les reines déjà en place) alors :
    - Y mettre la reine `k` puis
    - Résoudre le problème pour `k+1`.
  - Libérer la reine `k`.

Déterminer la complexité d'une approche de résolution du problème des 8 reines.

Bon courage