Epreuve Finale : Algorithmique avancée et Complexité

(1 heure)

<u>Exercice 1 :</u> (08 pts) (2.5, 2.5, 3)

Déterminer l'invariant des différentes boucles et en déduire l'ordre de complexité des algorithmes suivants :

Algo_A1()
Début
Pour i := 1 à N
    Faire Pour j := i+1 à N
        Faire k := N ;
        Tant que k >= i
        Faire k = k-1 ; Fait;
        Fait ;
    Fait ;
Fait ;
Fin.

Algo_A3()
Début
C=0 ;
Pour i = n à 1 pas de (-1)
Faire j=1 ;
Tant que (j < n*n*n)
    Faire c= c+1 ; j=j*4 ; fait ;
Fait ;
Retourner (c) ;
Fin ;

Algo_A2()
Début
x:= N;
Tant que (*x* > 1)
Faire $x := \sqrt{x}$ Fait ;
Fin.

- L'algorithme A1 :

Comporte 3 boucles imbriquées dépendantes, dont les deux boucles externes « Pour » avec borne inférieure, supérieure et pas clairement défini.
La boucle externe s'exécute en N itérations faisant varier l'indice i de 1 à N. **(0,5)**
Pour chaque valeur de i la seconde boucle s'exécute en N-i en faisant varier l'indice j de i+1 à N. **(0,5)**
Pour chaque valeur de j la boucle la plus interne fait varier un indice k de N à i. **(0,5)**
Autrement dit,
Pour i =1, la seconde boucle devra faire N-1 itération et à chaque itération de la seconde boucle, la boucle la plus interne devra itérer N-i+1 fois.
Donc, pour chaque itération de la boucle externe les deux boucles internes coutent (N-i)*(N-i+1).
Donc le nombre d'itérations totale = $\sum_{i=1}^{N}(\mathrm{N}-\mathrm{i})*(\mathrm{N}-\mathrm{i}+1) = \sum_{i=1}^{N}(N-i)^2+N-i$
$= \sum_{i=1}^{N}(N-i)^2 + \sum_{i=1}^{N} N-i$
$= \sum_{i=1}^{N-1} i^2 + \sum_{i=1}^{N-1} i = \mathrm{O}(\mathrm{N}^3) + \mathrm{O}(\mathrm{N}^2)$ **(1 pt)**
Ainsi l'algorithme A1 est de l'ordre de $\mathrm{O}(\mathrm{N}^3)$.

- L'algorithme A2 :

Le nombre d'itération de l'Algorithme A2 correspond au nombre de fois que la racine est calculé pour atteindre en partie entière 1. **(1 pt)**

Bon courage !

Donc en réalité la valeur de X à l'issu de la dernière itération doit être comprise entre 2 et 1 pour que la partie entière soit égale à 1 et que la terminaison de la boucle soit garantie
Autrement dit :

$$1 < (((N^{\frac{1}{2}})^{\frac{1}{2}}) \ldots\ldots)^{\frac{1}{2}} < 2 \Rightarrow 1 < N^{\frac{1}{2^k}} < 2 \Rightarrow \log 1 < \log N^{\frac{1}{2^k}} < \log 2$$

$$\Rightarrow 0 < \frac{1}{2^k} \log N < \log 2$$

$$\Rightarrow 0 < \frac{1}{2^k} < \frac{\log 2}{\log N}$$

$$\Rightarrow K < \frac{1}{log2} \log \log N - loglog2$$

$$\Rightarrow La\ complexité\ est\ en\ O(loglog\ N).$$ **(1,5 pt)**

La limite peut également être calculée autrement comme suit :

$$1 < N^{\frac{1}{2^k}} < 2 \Rightarrow 1 < N < 2^{2^k} \Rightarrow K < \log \log(N)$$

- L'algorithme A3 :

Pour la boucle interne on a :

J=1 →4^0

J=4

J=4^2

……

J=4^k = n^3, (n^3 la borne supérieur de la boucle interne). **(1 pt)**

Log 4^k = log n^3, donc k log 4= 3 log n, ainsi k = 3/log4 *log n **(1 pt)**

La boucle externe est une boucle « pour » de n à 1 avec pas de -1. Le nombre d'itération est égal à N. **(0,5)**

Les deux boucles sont indépendantes, donc le nombre d'itérations au totale dépend de n logn. Donc l'ordre de complexité est de O(n log n). **(0,5)**

Exercice 2 : (12 pts) (3, 4, 2, 1, 2)

Soit en entrée deux matrices A (N, M) et B(P, Q), tel que P << N et Q <<M. On s'intéresse au problème de recherche de la matrice B dans la matrice A. En d'autres termes, on cherche à déterminer si la matrice B est une sous matrice de la matrice A (Voir exemple sur le verso de la feuille).

1. Illustrer le principe de recherche de la sous matrice B dans la matrice A sur l'exemple.
2. Proposer un algorithme itératif (ou récursif) permettant de répondre au problème posé.

Bon courage !

3. Calculer la complexité de l'algorithme proposé.
4. Est-ce qu'il est nécessaire de spécifier s'il s'agit d'une complexité au pire cas ou au meilleur cas ? Justifier votre réponse.
5. Supposons maintenant, que les matrices A et B en entrée sont triées (sur les lignes et sur les colonnes). Comment cette nouvelle distribution des données va influer sur le processus de recherche et sur l'ordre de complexité ? (Avec l'exemple uniquement sans donner un nouvel algorithme).

2	2	2	3	5	7	8	17	24	24	54	67	76
3	4	4	5	6	6	6	8	11	12	33	81	85
12	14	23	26	26	26	31	34	44	45	52	87	90
6	6	17	24	24	54	56	61	67	81	87	90	108
2	2	2	3	5	7	8	17	24	24	54	67	76
3	4	4	5	6	6	6	8	11	12	33	81	85
12	14	23	26	26	26	31	34	44	45	52	87	90
6	6	17	24	24	54	56	61	67	81	87	90	108
12	14	23	26	26	26	31	34	44	45	52	87	90
6	6	17	24	24	54	56	61	67	81	87	90	108

Matrice A

24	54	56
5	7	8
6	6	6

Matrice B

Solution :

1. Plusieurs stratégies possibles **(3 pts)**:

- Balayage de la matrice A ligne par ligne jusqu'à la ligne (N-P) ; à la recherche du premier élément de la matrice B (B[1, 1]). Sur chaque ligne jusqu'à la position (M – Q). Une fois trouver, parcourir les deux matrices en même temps, pour identifier les éléments restants de B. Si ce n'est pas le cas reprendre à partir de la position du premier élément de B trouvé +1.

Bon courage !

- On peut également effectuée un balayage de la matrice colonne par colonne jusqu'à la M-Q colonnes et sur chaque colonnes jusqu'au N-P ieme élément.
- Il est également possible de faire un parcours à partir de la fin de la matrice de deux manières possibles : soit en considérant le dernier élément de la matrice B (B[P, Q]) ce qui signifie que la recherche va démarrer à partir des dernières positions de la matrice A jusqu'à la Nième ligne et à Mième colonne. Si vous voulez rechercher la première position de la matrice B pour commencer la recherche, alors dans ce cas-là la recherche s'arrête à la N-P ième ligne et à la M-Q ième colonne.
- Une autre manière possible, si les deux matrices sont carrées il est possible de faire une recherche à partir d'une position centrale de la matrice A [N/2, N/2], pour rechercher un élément centrale de la matrice B [P/2, P/2] et de faire un balayage sur les deux diagonales pour identifier les éléments restants de la matrice B.

2. Algorithme : **(4 pts)**

```
Début
Int i, j, k, h N, M, P, Q ;
exist = faux ;
I=1 ;
Tant que (i <= N-P) et (exist = faux)
Faire j=1 ;
        Tant que (j<= M-Q) et (exist = faux)
        Faire Si A[i, j] = B[1, 1] alors k =0 ; trouve = vrai ;
                                        Tant que (k<= P) et (trouve = vrai)
                                        Faire h=0 ;
                                                Tant que (h<= Q) et (trouve = vrai)
                                                Faire si A[i+k, j+h] <> B[k+1, h+1]
                                                                alors trouve = faux;
                                                                Sinon h=h+1 ;
                                                                Fsi ;
                                                Fait ;
                                                K=k+1 ;
                                        Fait ;
                                        Exist= trouve ;
// si trouve est resté à vrai alors la matrice B a entièrement été retrouvé dans la matrice A.
```

Bon courage !

```
                Sinon j= j+1 ;
                Fsi ;
        I=i+1 ;
Fait ;
Fin.
```

3. Complexité de l'algorithme proposé **(2 pts)**: **{1 pt calcul et 1 pt analyse}**

Nous avons 4 boucles imbriqués, nous allons donc commencer par la plus interne jusqu'à la plus externe :

Boucle 4 : Q -1 itérations

Boucle 3 : P-1 itérations

Boucle 3 et boucle 4 sont indépendantes donc le nombre d'itérations entre les deux boucles (Q-1)*(P-1).

Boucle 2 : (M – Q) itérations

Boucle 1 : (N – P) itérations.

Les deux boucles ensemble donnent (M-Q)*(N-P) itérations.

Le passage des deux premières boucles imbriquées vers les deux autres se fait de manière conditionnelle lorsque la première valeur de la matrice B est rencontrée. Donc le nombre de fois que ce passe existe dépend directement du nombre de fois où cette valeur existe dans la matrice.

Si cette valeur existe une seule fois alors le nombre d'itération au totale s'exprime ainsi :

(M-Q)*(N-P) + (Q-1)*(P-1) = O(M*N) puisque N >> P, et M >> Q.

Cette valeur peut aussi exister plusieurs fois dans la matrice A (au maximum N*M fois), dans ce cas, le nombre d'itérations devient : (M-Q)*(N-P)*(Q-1)*(P-1) = O(M*N*P*Q).

(Cas d'une matrice A qui contient que la valeur 24. Et une matrice B qui contient que la valeur 24, sauf à la dernière position B[P, Q] = 51).

Bon courage !

Puisque puisque N >> P, et M >> Q, alors même dans ce cas, O(M*N)

4. Le pire cas ou le meilleure cas ici ne dépend pas de la taille des matrices mais dépend du nombre d'occurrence de la première valeur de la matrice B dans la matrice A. Donc l'ordre de complexité susmentionné tient compte du nombre d'occurrence de cette valeur. Sachant que la dimension de la matrice B est négligeable devant la taille de la matrice A, la complexité dans le pire cas et dans le meilleure cas est égale **(1 pt)**
5. Si les deux matrices sont triées sur les lignes et sur les colonnes, il est possible d'envisager une recherche dichotomique pour localiser la première valeur de la matrices sans avoir à parcourir les (N-P)*(M-Q) positions possibles de la matrices. Il est même possible d'envisager une décomposition de la matrice en s'inspirant du principe « diviser pour régner ». On peut ainsi réduire considérablement la complexité de l'algorithme. **(2pts : 1 pt analyse, 1 pt illustration)**

Bon courage !