USTHB
Faculté d'Electronique et d'Informatique
Département D'Informatique
Master SII
Algorithmique Avancée et Complexité **Bab-Ezzouar 11 Janvier 2017**

Corrigé de l'épreuve finale

Exercice 1 : (8 pts) structure de données et algorithme de tri.

Considérer la suite des nombres suivants :

11, 73, 29, 45, 6, 31, 52, 89, 93, 9

1) Construire un tas pour cette suite de nombres. Montrer clairement les étapes de l'application de l'algorithme de construction d'un tas. (2,5 pts)

Les étapes des différentes itérations et plus précisément des différentes permutations de la construction du tas sont montrées ci-dessous :

11	73	29	45	6	31	52	89	93	9
				9					6
			93					45	
		52				29			
	93		73						
			89				73		
93	11								
	89		11						
93	89	52	73	9	31	29	11	45	6

2) Insérer le nombre 90 dans le tas construit en 1). **(2 pts)**

Illustration de l'insertion de 90 dans le tas construit précédemment.

93	89	52	73	9	31	29	11	45	6	90
				90						9
93	90	52	73	89	31	29	11	45	6	9

3) Trier la suite des nombres en appliquant l'algorithme de tri par tas. Montrer clairement les étapes d'exécution de l'algorithme.

Tri de la séquence de nombres en utilisant le tri par tas : (0,35 pts par itération)

Première itération de l'algorithme tri par tas :
Le maximum de la séquence se trouve au niveau de la racine, donc 93. On le permute avec le dernier élément du tableau. Et on construit le tas de nouveau pour les éléments qui restent à ranger, ce qui donne :

9	90	52	73	89	31	29	11	45	6	93
90	9									
90	89	52	73	9	31	29	11	45	6	93

Deuxième itération de l'algorithme tri par tas :
Maintenant c'est 90 qui se trouve à la racine du tas. On le permute avec le dernier élément du tableau qui reste à trier. Et on construit le tas de nouveau pour les éléments qui restent à ranger, ce qui donne :

6	89	52	73	9	31	29	11	45	90	93
89	6									
	73		6							
89	73	52	45	9	31	29	11	6	90	93

Troisième itération de l'algorithme tri par tas :
89 se trouve maintenant à la racine. On le permute avec le dernier élément du tableau qui reste à trier et on reconstruit le tas.

6	73	52	45	9	31	29	11	89	90	93
73	6									
	45		6							
73	45	52	11	9	31	29	6	89	90	93

Quatrième itération de l'algorithme tri par tas :
On permute 73 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas.

6	45	52	11	9	31	29	73	89	90	93
52		6								
		31			6					
52	45	31	11	9	6	29	73	89	90	93

Cinquième itération de l'algorithme tri par tas :
On permute 52 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas :

29	45	31	11	9	6	52	73	89	90	93
45	29	31	11	9	6	52	73	89	90	93

Sixième itération de l'algorithme tri par tas :
On permute 45 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas :

6	29	31	11	9	45	52	73	89	90	93
31	29	6	11	9	45	52	73	89	90	93

Septième itération de l'algorithme tri par tas :
On permute 31 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas :

9	29	6	11	31	45	52	73	89	90	93
29	9									
29	11	6	9	31	45	52	73	89	90	93

Huitième itération de l'algorithme tri par tas :
On permute 29 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas :

9	11	6	29	31	45	52	73	89	90	93
11	9	6	29	31	45	52	73	89	90	93

Neuvième itération de l'algorithme tri par tas :
On permute 11 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas :

6	9	11	29	31	45	52	73	89	90	93
9	6	11	29	31	45	52	73	89	90	93

Dixième itération de l'algorithme tri par tas :
On permute 9 avec le dernier élément du tableau qui reste à trier et on reconstruit le tas :

6	9	11	29	31	45	52	73	89	90	93

Le processus s'arrête avec le tableau trié de la sorte :

6	9	11	29	31	45	52	73	89	90	93

Exercice 2 : (12 points) Robot à la recherche d'un objet.

Considérer le problème Π décrit comme suit :

Instance : *Une matrice A[n, n], un robot **R** se trouvant à la position (p, q) de la matrice A et pouvant se déplacer d'une case à une autre à la fois, à droite, à gauche, en haut ou en bas, à la recherche d'un objet **a** se trouvant à la case (r, s) comme le montre la figure ci-dessous.*

Question : *Existe-il un chemin pour le robot **R** lui permettant de trouver l'objet **a** tel que le nombre de déplacements est inférieur ou égal à k, k étant un nombre entier positif ?*

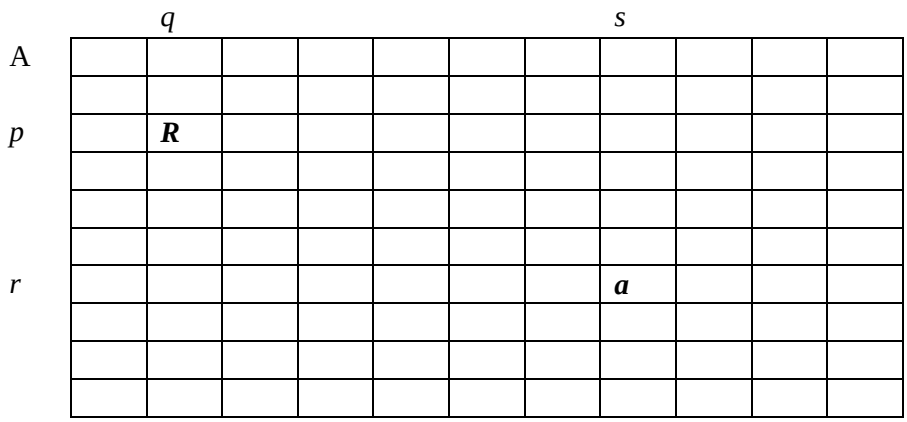

Figure. Le robot **R** à la recherche d'un objet **a**.

1) Ecrire un algorithme pour résoudre le problème Π. Calculer sa complexité. **(2 pts)**

Algorithme Π
***input** : A[n, n], p, q, r, s, k;*
***output** : oui il existe un chemin ou bien non il n'existe pas de chemin;*

début
si $(|r-p|+|s-q|) \leq k$ ***alors afficher****('oui')* ***sinon afficher*** *('non') ;*
fin

En effet, le nombre minimum de déplacements que le robot aura à faire pour atteindre l'objet **a** est égal à $(|r-p|+|s-q|)$. S'il est inférieur ou égal à k, la réponse est 'oui' sinon la réponse est 'non'.

Il est clair que la complexité de l'algorithme Π est *O(1).* ***(1 pt)***

2) Montrer que le problème Π appartient à la classe NP.

L'algorithme Π a une complexité polynomiale. Le problème Π appartient donc à la classe P. Il appartient par conséquent à la classe NP. **(2 pts)**

3) Supposer que les coordonnées de l'objet ***a*** ne sont pas connus par le robot, et soit Π' le problème associé à cette donne. Ecrire un algorithme récursif pour résoudre le problème Π'. Calculer sa complexité.

Un algorithme récursif pour résoudre le problème Π' est le suivant : **(3 pts)**

Algorithme Π'

input *: A[n, n], p, q, k;*
output *: oui il existe un chemin ou bien non il n'existe pas de chemin;*

```
procédure π'
var m : entier ;
début   si (l<k) alors
          si case-supérieure existe et non visitée
                alors début  m := l ;
                             déplacer R vers le haut;
                             si case contient a alors trouve := vrai ;
                                          sinon début m := m+1 ;
                                                      Π'(p+1,q,m) ;
                                                fin ;
                      fin ;
        si non trouve alors
          si case-inférieure existe et non visitée
                alors début  m := l ;
                             déplacer R vers le bas;
                             si case contient a alors trouve := vrai ;
                                          sinon début m := m+1 ;
                                                      Π'(p-1,q,m) ;
                                                fin ;
                      fin ;
        si non trouve alors
          si case-gauche existe et non visitée
                alors début  m := l ;
                             déplacer R vers la gauche ;
                             si case contient a alors trouve := vrai ;
                                          sinon début m := m+1 ;
                                                      Π'(p,q+1,m) ;
                                                fin ;
                      fin ;
        si non trouve alors
          si case-droite existe et non visitée
                alors début  m := l ;
                             déplacer R vers la droite ;
                             si case contient a alors trouve := vrai ;
                                          sinon début m := m+1 ;
                                                      Π'(p,q-1,m) ;
                                                fin ;
                      fin ;

fin

programme principal ;

trouve : booléen ; l : entier ;

début   l := 0 ;
        trouve := faux ;
        Π'(p ,q, l) ;
```

Si trouve ***alors afficher****(‘oui’)* ***sinon afficher*** *(‘non’) ;*

fin *;*

Le pire cas correspond à la situation où le robot ne trouve pas **a**. La complexité au pire s’exprime donc comme :

C(k) =C(k-1) + C(k-1) + C(k-1) + C(k-1) = 4 x C(k-1) **(1 pt)**

En effet, le robot commence par prendre un chemin constitué de déplacements (haut, bas, gauche ou droit). Il atteindra **a** au plus au bout de k déplacements, sinon il doit prendre un autre itinéraire pour sa recherche. k correspond à la longueur maximale au bout de laquelle le robot doit changer de chemin. A chaque fois qu’il avance d’une case, k est décrémenté.

C(k) = 4 x 4 x C(k-2)
= 4 x 4 x … 4 x C(0)
= 4^k

La complexité est donc $O(4^k)$. **(1 pt)**

4) Montrer que le problème Π’ appartient à la classe NP.

k étant une constante, l’algorithme est polynomiale et par conséquent Π’ appartient à la classe P et donc à la classe NP. **(2 pts)**