**Université Pierre et Marie Curie**



**UE: Algorithmique (LI311)**

**MiniProjet**

**Autour des Tours Hanoï**

**Année scolaire : 2013/2014**

**Professeur chargé de TD/TME :**

**Fanny Pascual**

**Etudiants :**

**Rémi Cadène n°3000693**

**Joël Fieux-Herrera n°3003174**

Sommaire

README ……………………………………………………...…..p2

**Readme**

Ce projet Java sous Eclipse contient quatre dossiers :

- bin contient les fichiers compilés,

- src contient les fichiers sources,

- data contient les fichiers annexes donnés en début de projet et générés par notre programme,

- doc contient le sujet et les réponses aux questions.

Afin de compiler et d'executer, deux possibilités :

- avec Eclipse :

ouvrir Eclipse

selectionner votre Workspace

importer (File / Import...)

selectionner "General / Existing Projects into Workspace", puis Next

selectionner en root directory le dossier contenant notre dossier projet (ex: Téléchargements), puis Finish

compiler et executer (Run)

- avec Terminal :

cd <directory>/LI323\_P2\_CadeneFieux

javac -cp ./src/BioInfo/\*.java -d ./bin/BioInfo/

**2. Les explorateurs et les adorateurs**

**Question 1 :**

Pour n=3 et k=2, ci dessous une des séquences de traversée valides :

((0,0,.), (1,1,I), (0,1,.), (0,3,I), (0,2,.), (2,2,I), (1,1,.), (3,1,I), (3,0,.), (3,2,I), (3,1,.), (3,3,I))

**Question 2 :**

Une configuration voisine de la configuration initiale (0,0,.) est valide, c’est à dire qu’elle appartient à l’ensemble que nous cherchons à définir, si le nombre d’explorateur sur l’île “nE” est égal au nombre total d’explorateur “n”, ou si “nE” est égal au nombre d’adorateur sur l’île “nA”, ou si il n’y a aucun explorateur sur l’île.

Alors la configuration appartient à cet ensemble si une des propositions suivantes est vérifiée :

1. k = n et nE = k,

2. 0 < nE <=⎣k/2⎦et nE = nA

3. nE = 0 et 0 < nA <= k

**Question 3 :**

En nous aidant de notre réponse à la question précédente, une configuration quelconque est valide si une des propositions suivantes est vérifiée :

1. nE = 0,

2. nE = nA,

3. nE = n

**Question 4 :**

Pour que deux configurations valides soient voisines, il faut que ces deux propriétés soient vérifiées :

1. nE - nE’ = 0 et 0 < nA - nA’ <= k

2. nE - nE’ + nA - nA’ <=⎣k/2⎦

Autrement dit 0 < |nE-nE’| + |nA-nA’| <= k en sachant que les configurations voisines (nE’,nA’,I) d’une configuration (nE,nA,.) vérifient nE’>=nE et nA’>=nA, ou inversement.

**Question 5 :**

L’algorithme en O(N+M) permettant de déterminer une plus courte chaîne dans un graphe est le parcours en largeur.

**Question 6 :**

Nombre de sommets en fonction de n : N(n) = (n+1)\*2 + (n+1)\*2 +(n+1)\*2 - 2\*2

Base : Pour n=0

N=(0+1)\*2 + (0+1)\*2 +(0+1)\*2 - 2\*2=6n+6-4

N=6-4

N=2

En effet pour n=0 on constate que les nombre de sommet seront 2 , a savoir (0,0,.) ou (0,0,I)

Induction : On suppose l'hypothèse vrai au rang n+1

Pour passer de N(n) à N(n+1) on ajoute 6 sommets :

(n+1,n+1,.) ; (n+1,n+1,I) ; (n+1,0,.) ; (n+1,0,I) ; (0,n+1,.), (0,n+1,I)

N(n+1) = N(n) + 6

N(n+1) = (n+1+1)\*2 + (n+1+1)\*2 +(n+1+1)\*2 - 2\*2

= 6n+12-4=6n+8

N(n)+6 =(n+1)\*2 + (n+1)\*2 +(n+1)\*2 - 2\*2 +6

=6n+6-4+6=6n+8

L’hypothèse est vérifiée.

On notera qu’un parcours en largeur d’un graphe généré avec la fonction traversee ne possèdera pas un tel nombre de sommet. En effet, pour un graphe de configurations n=3;k=2, les configurations (0,3,.) et (3,0,I) ou même (n,n,.) et (0,0,I) n’apparaitrons pas dans le parcours, car elles ne possèdent pas de voisins valides (le graphe n’est pas connexe).

Complexité de Traversee en O(n)

Dans un premier temps l’algorithme parcours le graphe des configurations valides avec l’algorithme de parcours en largeur (O(n+m)) en attribuant à chaque sommet rencontré un entier correspondant à la distance entre le sommet de départ et ces derniers.

Par ailleurs m peut être exprimer en fonction de n. En effet dans le pire des cas un sommet possède k\*3 voisins

Cette partie est alors en O(n).

Dans un second temps on parcours le graphe de la même manière en partant de la configuration finale, c’est à dire en O(n).

Alors la complexité de Traversee est en O(n).

**Question 7 :**

Nous avons utilisé une liste (LinkedList) pour stocké les configurations composant le plus court chemin, et une table de hashage (HashMap) pour stocker les configurations visités avec pour clé une chaine de caracteres et comme valeur un entier représentant la distance entre le configuration finale et la configuration visitée.

Courbe du temps CPU pour des graphes de configurations n<1500;k=4

GRAPHE

La croissance de cette courbe est linéaire ce qui correspond à la complexité théorique O(n) obtenue lors de l’exercice précédent.

On remarquera tout de même qu’une complexité en O(1) des opérations “get” et “put” de la structure de donnée HashMap n’est pas garantie.

**3. Les Tours de Hanoï**

**3.1. Le graphe de Hanoï**

**Question 8 :**

Le graphe des configurations Gn possède 3\*n sommets.

En effet chaque disque peut être positionné sur un des 3 piquets. Il existe donc 3 possibilités de placement par disque.

**Question 9 :**

Soit la fonction a donnant le nombre d’arêtes en fonction de n.

On a a(1) = 3 et a(n) = 3 \* a(n-1) + 3

Alors a(n) = 3 \* (3\* a(n-2) +3) +3 = 3^2 \* a(n-2) + 3^2 + 3

Alors a(n) = 3^2 \* (3\* a(n-3) +3) +3^2 + 3 = 3^3 \* a(n-3) + 3^3 + 3^2 + 3

Supposons a(n) = 3^(n-1) \* a(1) + Somme(i=1;n-1; 3^i)

= 3^(n-1) \* 3 + 3\*(1-3^(n-1) / (1-3)

= 3^n - 3 \* (1-3^(n-1)) / 2

Prouvons par récurrence sur n que le nombre d’arêtes est 3^n - 3 \* (1-3^(n-1)) / 2.

Base : a(1) = 3^1 - 3 \* (1-3^(1-1)) / 2 = 3

Induction :

On a a(n+1) = 3 \* a(n) + 3

D’après l’hypothèse de récurrence :

a(n+1) = 3 \* ( 3^n + Somme(i=1;n-1; 3^i) ) + 3

a(n+1) = 3^(n+1) + Somme(i=1;n; 3^i)

a(n+1) = 3^n+1 - 3 \* (1-3^(n)) / 2

CQFD

**3.2. Une première approche de résolution**

**Question 10 :**

De même que pour la question 6, on parcourt le graphe à l’aide du parcours en largeur en O(n+m). Cette fois ci, m = 3^n+1 - 3 \* (1-3^(n)) / 2, c’est à dire que m = O(3^n).  
L’algorithme Hanoi1 est donc de complexité en O(3^n), car dans le pire des cas le graphe généré possèdera 3\*n sommets et 3^n+1 - 3 \* (1-3^(n)) / 2 arrêtes.

**Question 11 :**

De même que pour la question 7, nous avons choisi une liste et une table de hashage.

GRAPHE

**3.3. Une deuxième approche de résolution**

**Question 12 :**

Nous avons utilisé une liste (LinkedList) afin de stocker les configurations composant la solution.

Dans un premier temps, nous créons une configuration initiale qui va se modifier jusqu’à atteindre la configuration finale. Et à chaque modification (Bouger), nous clonons celle ci et l’ajoutons à la liste des solutions (O(1)).

Dans un second temps, nous exécutons la procédure HanoiProc qui effectuera les mêmes actions que dans l’énoncé.

**Question 13 :**

Afin de déterminer Pf[i] avec i de 1 à n, nous effectuons la procédure suivante :

Pf[n] = “numéro du piquet final”

Pour d de n-1 à 1, faire

si Pi[d+1] = Pf[d+1],

alors Pf[d] = Pf[d+1]

sinon Pf[d] = 6 - Pi[d+1] - Pf[d+1]

**Question 14 :**

Afin de passer de la configuration finale de P(d-1) à la configuration finale de P(d), nous effectons la procédure suivante :

Si Pi[d] != Pf[d],

Bouger(Pi[d],Pf[d])

Hanoi(d-1,3-Pi[d]-Pf[d], Pf[d])

**Question 15 :**

Soit la procédure Hanoi2 suivante :

Pi[] = CreationPi()

Pf[] = CreationPf()

Pour d de 1 à n faire

Si Pi[d] != Pf[d]

Bouger(Pi[d],Pf[d])

Hanoi(d-1,3-Pi[d]-Pf[d], Pf[d])

Premièrement, la fonction CreationPi transforme la configuration donnée en tableau en parcourant chaque piquet. C’est à dire qu’on parcourt au total n disques en copiant l’identifiant de leur piquet dans le tableau. Celle ci est donc de compléxité en O(n).  
Deuxièmement, la fonction CreationPf est l’application de la procédure déterminée lors de la question 13. Elle est composée d’une boucle Pour, de n-1 à 1, d’opérations de complexité en O(1). Celle ci est donc en O(n).

Troisièmement, la procédure Bouger est en O(1).

Quatrièmement, la procédure récursive Hanoi est en O(2^n) (vu en cours et démontré dans la question 17).

On effectue dans le pire des cas n fois la procédure Hanoi. La complexité d’Hanoi2 est donc en O(n\*2^n)

**Question 16 :**

Nous avons implémenté Hanoi2 à l’aide d’une liste (LinkedList) contenant les configurations formant la solution, de deux tableaux d’entier (ArrayList) représentant Pi et Pf.

GRAPHE

**3.4. Calcul du nombre de déplacement de disques**

**Question 17 :**

Afin d’adapter l’algorithme Hanoi2 à Hanoi3, nous avons incrémenté un entier à la place de la procédure Bouger.

Nous avons par ailleurs remarqué que la procédure récursive Hanoi(n) exécute (2^n-1) fois la procédure Bouger.

En effet, soit f la fonction donnant le nombre de fois que la procédure Bouger est appelée.  
On a f(1) = 1 et f(n) = 2\*f(n-1)+1.

Démontrons par récurrence sur n que ceci est égal à f(n) = 2^n-1 :

Cas de base : f(1) = 2\*0+1 = 1.

Induction :

On a f(n+1) = 2\*f(n)+1.

En supposant que f(n) = 2^n-1, on a f(n+1) = 2\*(2^n-1)+1 = 2^(n+1)-1.

CQFD

Alors nous pouvons remplacer Hanoi par la fonction f.

**Question 18 :**

Pour la configuration de l’énoncé nous obtenons jhsiuhdf déplacements de disques. Alors, si la légende dit vraie et en supposant que les prêtres sont des maîtres Jedi, il restera avant la fin du monde pojsfoij secondes, soit osijfoisjd années.