**Université Pierre et Marie Curie**



**UE: Algorithmique (LI311)**

**MiniProjet**

**Autour des Tours Hanoï**

**Année universitaire : 2013/2014**

**Professeur chargé de TD/TME :**

**Fanny Pascual**

**Etudiants :**

**Rémi Cadène n°3000693**

**Joël Fieux-Herrera n°3003174**

Sommaires

2. Les explorateurs et les adorateurs 2

Question 1 : 2

Question 2 : 2

Question 3 : 2

Question 4 : 2

Question 5 : 3

Question 6 : 3

Question 7 : 4

3. Les Tours de Hanoï 5

3.1. Le graphe de Hanoï 5

Question 8 : 5

Question 9 : 5

3.2. Une première approche de résolution 5

Question 10 : 5

Question 11 : 6

3.3. Une deuxième approche de résolution 6

Question 12 : 6

Question 13 : 7

Question 14 : 7

Question 15 : 7

Question 16 : 8

3.4. Calcul du nombre de déplacement de disques 9

Question 17 : 9

Question 18 : 9

# 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’explorateurs sur l’île “nE” est égal au nombre total d’explorateurs “n”, ou si “nE” est égal au nombre d’adorateurs 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 :

Calculons N(n) le nombre de sommets en fonction de n :

Tout d'abord considérons les configurations valides avec la barque sur l'île. Il existe (n+1) configurations de la forme (n,0,I), (n+1) autres de la forme (0,n,I), et (n+1) autres de la forme (n,n,I) ; soustrayons les deux configurations (0,0,I) en trop ; puis multiplions la somme par 2 afin de considérer les configurations valides avec la barque sur la rive.  
Alors montrons par récurrence sur n que N(n) = (n+1)\*2 + (n+1)\*2 +(n+1)\*2 - 2\*2 = 6n+2.

Base : N(0) = 2

En effet pour n=0 on constate que le nombre de sommets est 2, à savoir (0,0,.) ou (0,0,I)

Induction : On suppose N(n) = 6n+2 vrai jusqu'au rang n, montrons N(n+1) vrai.

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)

On a alors : N(n+1) = N(n) + 6

D'après l'hypothèse de récurrence : N(n+1) = 6n+2 + 6 = 6n+8 = 6(n+1) + 2

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 sommets. 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).

En supposons que la capacité de la barque est une constante, calculons la complexité de traversee dans le pire des cas.

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 exprimé en fonction de n. Calculons M(n) le nombre maximum de sommets valides voisins en fonction de n :

Soit (x,y,.) une configuration valide quelconque. On génère toutes les configurations voisines de celle ci en supposons qu'elles sont toutes valides :

Somme(i=1 à k; (x+i,y,I)) et Somme(i=1 à k; (x,y+i,I)) et Somme(i=1 à k; (x+1,y+i,I))  
De même avec (x,y,I) en remplaçant +i par -i.

Comme k<2\*n, on peut exprimer M(n) en fonction de n tel que :

M(n) = 3\*k = 3\*(2\*n-1) = 6\*n - 6

Alors cette partie de l'algorithme est de complexité O(n + 6\*n - 6) = 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).

Nous obtenons donc une complexité en O(n+n) = O(n) pour l'algorithme traversee.

### Question 7 :

Nous avons utilisé une liste (LinkedList) pour stocké les configurations composant le plus court chemin, une liste (LinkedList) pour stocker les configurations correspondant à des sommets ouverts, et une table de hashage (HashMap) pour stocker les configurations visitées avec pour clé une chaine de caractères et comme valeur un entier représentant la distance entre le configuration finale et la configuration visitée. Les configurations sont stockées dans une instance de la classe Config possédant deux entiers nA et nE et un booléan p en attributs.

Afin de générer les voisins valides d'une configuration, nous avons implémenté une méthode *generateNext.* Celle ci commence par créer toutes les configurations voisines, puis ne sélectionne que celles qui sont valides et rend un tableau (ArrayList) de configurations (Config).

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 remarqué que 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

L'hypothèse est alors vérifiée.

## 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(n + 3^n) = 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.

La courbe générée correspond bien à une complexité exponentielle.

## 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 récursive 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 effectuons 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[] = CréationPi()

Pf[] = CréationPf()

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 CréationPi 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 complexité en O(n).

Deuxièmement, la fonction CréationPf 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, nous allons calculer le nombre d'appels à la procédure Bouger dans la procédure récursive Hanoi.

Soit f la fonction donnant le nombre d'appels de la procédure Bouger.

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

Démontrons par récurrence sur n que f(n) = 2^(n) - 1 :

Cas de base : f(1) = 2^1 - 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

C'est à dire f(n+1) = 2^(n+1) - 1.

L'hypothèse est vérifiée et on remarque que Hanoi est en O(2^n).

Comme à chaque tour de boucle quand Pi[d] != Pf[d], nous avons un appel à Bouger et un à Hanoi, alors dans le pire des cas nous avons une somme, pour d de 1 à n, de f(d) + 1 appels à Bouger.

Soit S cette somme, nous avons S(n) = Somme(d=1 à n; 2^(n) - 1 + 1).

Alors S(n) = Somme(d=1 à n; 2^(n)).

Soit la suite géométrique G(n) de raison 2 tel que G(n+1) = 2 \* G(n) et G(0) = 1. On a donc trivialement G(n) = 2^n.

Nous appliquons alors la formule contenue dans le rappel de la question 9 tel que :

S(n) = Somme(d=1 à n; G(n))) = G(0) \* (1 - 2^(n-0+1)) / (1-2)

S(n) = 2^(n+1) - 1

Alors la complexité d’Hanoi2 est donc en O(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.

La courbe générée correspond bien à une complexité exponentielle, mais Hanoi2 est bien plus efficace que Hanoi1. C'est à dire que la croissance est moins rapide. En effet, cela correspond aux complexités calculées précédemment.

## 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, puis nous avons remplacé Hanoi par la fonction f que nous avons calculé dans la question 15.

Alors l'algorithme effectue n appels à la fonction f de complexité O(1). Celui ci est donc de complexité en O(n).

### Question 18 :

Pour la configuration de l’énoncé nous obtenons 18446744073709290495 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 584942417355 années.