

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université des Sciences et de la Technologie Houari Boumédiène

Faculté d'Informatique
Département Informatique

Master Systèmes Informatiques intelligents

Module : Conception et Complexité des Algorithmes

Rapport de projet de TP tour de Hanoï

Réalisé par :

BENBACHIR Mohamed Amir, 191932021049

BOUCHOUL Bouchra, 191931081317

KHEMISSI Maroua, 191935007943

MEDJKOUNE Roumaïssa, 191931081005

Année universitaire : 2022 / 2023

Table des matières

1	Introduction	2
2	Etude théorique du problème	3
2.1	Historique et présentation du problème.	3
2.2	Définition formelle du problème.	3
2.3	Présenter la modélisation de la solution (Structure de données de la solution). . . .	4
2.4	Présenter l'algorithme de résolution avec calcul détaillé de sa complexité théorique.	5
2.4.1	L'Algorithme de résolution :	5
2.4.2	Calcul de complexite temporel recursive	5
2.4.3	Calcul de complexite spatiale recursive	6
2.4.4	l'Algorithme de resolution Iterative	6
2.5	Présenter l'algorithme de vérification avec pseudo-code et calcul détaillé de sa complexité théorique.	9
2.5.1	L'Algorithme de vérification :	9
2.5.2	Complexité théorique	9
2.6	Présentation d'une instance du problème avec sa solution (un exemple).	9
3	Etude Expérimentale	13
3.1	Tableau Des Expérimentations	13
3.2	Simulation de la complexité temporelle et spatiale théorique de l'algorithme de résolution	14
3.2.1	Simulation de la complexité temporelle théorique	14
3.2.2	Simulation de la complexité spatiale théorique	15
3.3	Le meilleur, moyen et pire cas de l'algorithme de résolution	15
3.3.1	Le meilleur, moyen et pire cas de la complexite spatiale	15
3.3.2	Le meilleur, moyen et pire cas de la complexité temporelle	15
3.4	Analyse des résultats	16
4	Conclusion	17
5	Références	18
6	Annexe :code source	19
6.1	Repartition des taches	19
6.2	algorithme.c	20

Chapitre 1

Introduction

Le problème des tours de Hanoï est un jeu faisant parti de la catégorie des casse-têtes. Il est très exploité dans la recherche en psychologie de la résolution de problème, employé comme épreuve lors d'une évaluation neuropsychologique des fonctions exécutives, étudié en mathématiques et souvent utilisé en algorithmique pour montrer la puissance et l'intérêt de la récursivité.

Nous allons dans ce projet traiter ce problème dans deux parties une théorique où on va proposer et étudier une solution récursive des tours de Hanoï, et une partie expérimentale où on va appliquer les deux versions itératives et récursives sur des instances de ce jeu et évaluer leurs complexités.

Chapitre 2

Etude théorique du problème

2.1 Historique et présentation du problème.

Le problème mathématique des tours de Hanoï a été inventé par Édouard Lucas. Il annonce que ce problème est dû à un de ses amis, N. Claus de Siam , prétendument professeur au collège de Li-Sou-Stian .

Sous le titre « Les brahmes tombent », Lucas relate que « N. Claus de Siam a vu, dans ses voyages pour la publication des écrits de l'illustre Fer-Fer-Tam-Tam, dans le grand temple de Bénarès, au-dessous du dôme qui marque le centre du monde, trois aiguilles de diamant, plantées dans une dalle d'airain, hautes d'une coudée et grosses comme le corps d'une abeille. Sur une de ces aiguilles, Dieu enfila au commencement des siècles, 64 disques d'or pur, le plus large reposant sur l'airain, et les autres, de plus en plus étroits, superposés jusqu'au sommet. C'est la tour sacrée du Brahmâ. Nuit et jour, les prêtres se succèdent sur les marches de l'autel, occupés à transporter la tour de la première aiguille sur la troisième, sans s'écarter des règles fixes que nous venons d'indiquer, et qui ont été imposées par Brahma. Quand tout sera fini, la tour et les brahmes tomberont, et ce sera la fin des mondes2! ».

Comme indiqué ci-dessous, **le problème** est ce que un jeu à 64 disques requiert un minimum de $2^{64}-1$ déplacements (soit 18 446 744 073 709 551 615 déplacements). En admettant qu'il faille une seconde pour déplacer un disque, ce qui fait 86 400 déplacements par jour, la fin du jeu aurait lieu au bout d'environ 213 000 milliards de jours, ce qui équivaut à peu près à 584,5 milliards d'années, soit 43 fois l'âge estimé de l'univers (13,7 milliards d'années selon certaines sources).

2.2 Définition formelle du problème.

Le but est de déplacer des disques de diamètres différents d'une tour de « départ » vers une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer plus d'un disque à la fois ;
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.

En suppose que cette dernière règle est également respectée dans la configuration de départ. Ce problème, peut devenir très vite complexe. En effet, pour n disques le nombre de déplacements nécessaires est au minimum de $2^n - 1$.

Le déplacement des disques nécessite donc deux fois plus de temps à chaque fois qu'un disque est ajouté à la tour initiale.

La résolution du problème des tours de Hanoï est très étudiée en algorithmique où elle sert notamment à montrer que l'utilisation de la récursivité pour résoudre de gros problèmes peut produire des codes à la fois logiques, puissants et concis. Nous allons dans ce partie theorique proposer une solution récursive des tours de Hanoï, puis évaluer sa complexité.

2.3 Présenter la modélisation de la solution (Structure de données de la solution).

Le problème de tour de Hanoi peut être aisément modélisé avec un algorithme récursif.

Fonction Hanoi(n : entier , D : tour , I : tour, A : tour)

```

1 début
2   si  $n > 0$  alors
3     Hanoi( $n-1,D,I,A$ ) ;
4     Déplacer  $D$  vers  $A$  ;
5     Hanoi( $n-1,I,A,D$ ) ;
6 fin
```

Départ, intermédiaire et arrivée représentent les 3 tours.

N est le nombre de disques à déplacer. Le disque numéro n représente le plus grand disque se trouvant en bas de l'empilement dans la tour initiale.

Le principe de l'algorithme consiste à déplacer $n-1$ disques de « départ » vers « intermédiaire », puis le disque n de « départ » vers « arrivée » pour enfin mettre les $n-1$ disques sur la tour « arrivée ». Le déplacement des $n-1$ se fait en plusieurs étapes à travers les appels récursifs de la procédure, jusqu'à ce qu'il ne reste qu'un disque à déplacer.

La structure de donnée qui serve notre problematique respectants les lois (First In Last Out , Une seule disque déplacer a la fois (Depiler)) c'est bien la structure de donnée **La pile**. donc on peut représenter nos tours avec des piles ou la sommets et le disque le plus petit a déplacer .

2.4 Présenter l'algorithme de résolution avec calcul détaillé de sa complexité théorique.

2.4.1 L'Algorithme de résolution :

La resolution de tel probleme consiste a utiliser une variable **n** : qui va représenter le nombre des disques , **3 piles A , D , I** qui vas **respectivement** représenter les pillets d'arrivee , depart et intermediaire .

Pour déplacer une tour de n disques de D vers A, on effectue ces trois étapes :

- déplacer la tour des n-1 premiers disques de D vers I (étape qui nécessite xn-1 déplacements, d'où la récurrence) ;
- déplacer le plus grand disque de D vers A (un déplacement supplémentaire) ;
- déplacer la tour des n-1 premiers disques de I vers A (à nouveau xn-1 déplacements).

Nous pouvons le représenter via le pseudo code suivant :

Fonction Hanoi(n : entier ,D : pile ,I : pile,A : pile)
--

<pre>1 début 2 si n > 0 alors 3 Hanoi(n-1,D,I,A) ; 4 Empiler(A,Depiler(D)) ; 5 Hanoi(n-1,I,A,D) ; 6 fin</pre>
--

2.4.2 Calcul de complexite temporel recursive

Soit T le nombre d'instructions de l'algorithme Hanoi (ci-dessus), on a 2 appels recursive a n-1 disques et une instruction pour le déplacement Empiler(A,Depiler(D)) qui a 2 operations Empiler la pile A et depiler de D. On suppose que le temps d'empiler/ depiler est le même et on le note f1.

Donc $T(n)=2T(n-1)+2f1$

on a :

$T(0)=f2$, f2 constante

$T(1)=2f2+ 2f1$

$T(2) =2T(1)+ 2f1 = 4f2 + 6 f1$

$T(3) =2T(2)+ 2f1 = 8f2 + 14f1$

la coefficient de f2 : 2^n

la coefficient de f1 : $2*2^n - 2$

et donc $T(n) = 2^n f_2 + (2 * 2^n - 2) f_1$

On conclut là aussi que l'Algorithme de problème des « tours de Hanoi » a une complexité temporelle exponentielle.

En conséquence, la complexité temporelle $O(n)$ de l'algorithme Hanoi est :

$$\left\{ \begin{array}{ll} T(n) = T(n) = 2^n f_2 + (2 * 2^n - 2) f_1 & \text{notation - exacte} \\ T(n) = O(n) = O(2^n) & \text{notation - asymptotique} \end{array} \right.$$

2.4.3 Calcul de complexite spatiale recursive

Soit T le nombre d'instructions de l'algorithme Hanoi (ci-dessus),
on a 2 appels recursive a n-1 disques la deuxieme se lance apres qu'on finit la premiere et donc on peut utiliser l'espace d'appel recursive 1 pour faire le traitement de la deuxieme , on a 3 piles ou la somme de contenu de toutes les piles ne depasse pas le nombre de disque n (en total on a n disque soit initialement dans le disque de depart ou bien difusee entre les 3 disque)

Donc $T(n) = T(n-1) + n$

on a :

$T(0) = k$, k est constant (peut etre 0, NIL)

$T(1) = k + 1$

$T(2) = k + 2$

$T(n) = k + n$; d'ou la complexite spatiale est de $O(n)$.

2.4.4 l'Algorithme de resolution Iterative

la solution itérative de tour hanoi qui on a implémenté ce un result de un remarque dans le pattern des mouvements des disc fait par l'algorithm recursive

```
{
- start -> end ( ou end -> start)
- start -> inter (ou inter -> start)
- inter -> end (ou end -> inter)
}repete jusqu a le nombre des mouvements (deplacement) des disc est  $2^N - 1$ 
```

Fonction move(from : pile,to : pile)

```
1  début
2      si pilevide(to) alors
3          |  empiler(to,depiler(from)) ;
4      sinon si pilevide(from) alors
5          |  empiler(from,depiler(to)) ;
6      fin si
7      sinon si tetepile(from) > tetepile(to) alors
8          |  empiler(from,depiler(to)) ;
9      fin si
10     sinon
11         |  empiler(to,depiler(from)) ;
12     fin si
13 fin
```


Fonction HanoiIter(n : entier ,start : pile ,end : pile,inter : pile)

```
1 Variables :
2 numOfMoves : entier ;
3 début
4    $numOfMoves \leftarrow pow(2, n) - 1$ ;
5   si  $n \bmod 2 \neq 0$  alors
6       pour  $i \leftarrow 1$  à  $numOfMoves$  faire
7           si  $n \bmod 3 == 1$  alors
8               move(start,end) ;
9           sinon si  $n \bmod 3 == 2$  alors
10              move(start,inter) ;
11          fin si
12          sinon si  $n \bmod 3 == 0$  alors
13              move(inter,end) ;
14          fin si
15      fin pour
16  fin si
17  sinon
18      pour  $i \leftarrow 0$  à  $numOfMoves-1$  faire
19          si  $n \bmod 3 == 1$  alors
20              move(start,end) ;
21          sinon si  $n \bmod 3 == 2$  alors
22              move(end,inter) ;
23          fin si
24          sinon si  $n \bmod 3 == 0$  alors
25              move(inter,start) ;
26          fin si
27      fin pour
28  fin si
29 fin
```

2.5 Présenter l'algorithme de vérification avec pseudo-code et calcul détaillé de sa complexité théorique.

2.5.1 L'Algorithme de vérification :

L'algorithme de vérification consiste à vérifier la règle suivante : "le disque déplacé ne doit jamais être placé au-dessus d'un disque plus petit que lui". L'algorithme fonctionne suivant les étapes suivantes :

- On compare l'élément qui se situe à l'indice i de la pile et son successeur $i+1$.
- Si $p.array[i] > p.array[i+1]$ alors l'algorithme retourne faux
- Si on atteint la taille de la pile l'algorithme retourne vrai
- Sinon on avance dans la pile et on répète l'étape 1

Fonction Verification(p : pile)

```
1 Variables :
2 i : entier;
3 début
4   si ( $p.sommet = -1$ ) alors
5       return true;
6   fin si
7   pour  $i \leftarrow 1$  à  $p.sommet$  faire
8       si  $p.array[i] < p.array[i+1]$  alors
9           return false;
10      fin si
11      return true;
12  fin pour
13 fin
```

2.5.2 Complexité théorique


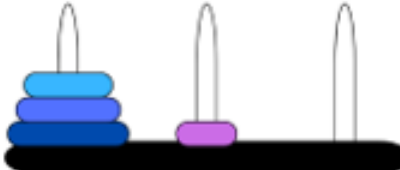
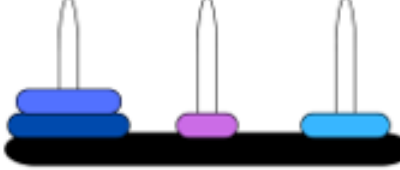

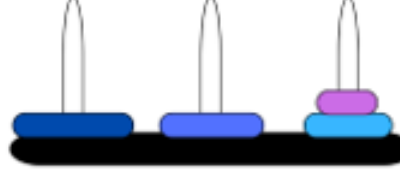
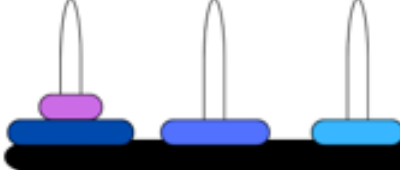
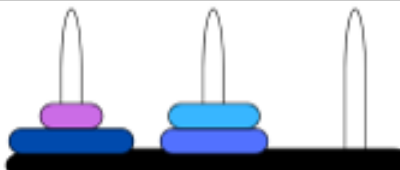

La complexité de l'algorithme de vérification est égale à la taille de la pile ça vaut dire le nombre de disques empilés (n) dans l'état finale d'où la complexité est linéaire $O(n)$

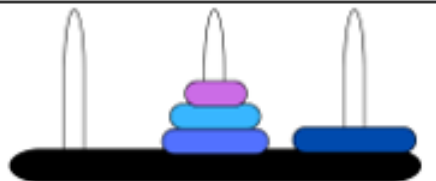
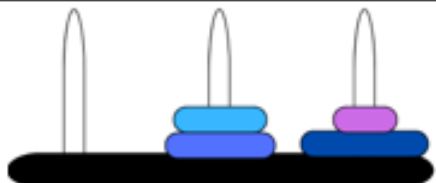
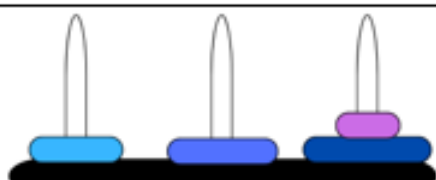
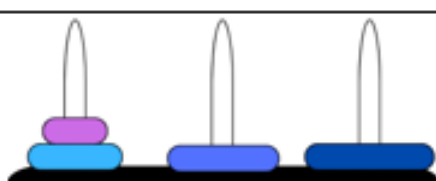
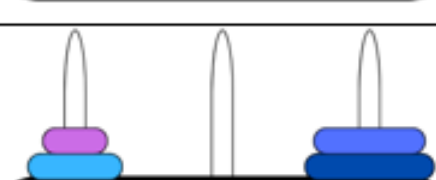
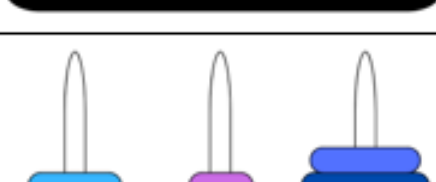
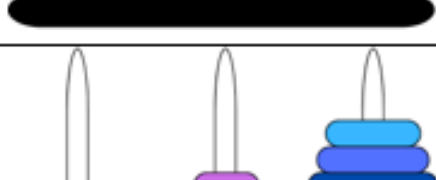
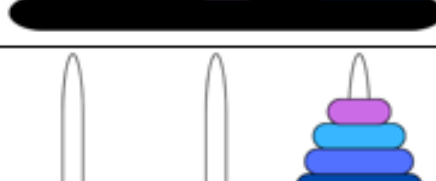
2.6 Présentation d'une instance du problème avec sa solution (un exemple).

Le jeu des tours de Hanoi est constitué de trois piquets, placés verticalement, et de n disques de taille décroissante. Les n disques sont initialement placés par taille décroissante autour du piquet

le plus gauche. Le but du jeu consiste donc a déplacer les disques jusqu'a parvenir a la situation finale dans laquelle tous les disques se retrouvent autour du piquet le plus droit par ordre de taille décroissante.

Les figures suivante représentent le déroulement de ce jeu ou le nombre de disque est égale à 4 (le nombre de déplacements $2^4 - 1$).

Nombre de déplacement	Position	Etat de la pile		
n=0		<div>Pile 1</div> <div>D1</div> <div>D2</div> <div>D3</div> <div>D4</div>	<div>Pile 2</div> <div></div> <div></div> <div></div> <div></div>	<div>Pile 3</div> <div></div> <div></div> <div></div> <div></div>
n=1		<div>Pile 1</div> <div>D2</div> <div>D3</div> <div>D4</div>	<div>Pile 2</div> <div>D1</div> <div></div> <div></div> <div></div>	<div>Pile 3</div> <div></div> <div></div> <div></div> <div></div>
n=2		<div>Pile 1</div> <div>D3</div> <div>D4</div>	<div>Pile 2</div> <div>D1</div> <div></div> <div></div> <div></div>	<div>Pile 3</div> <div>D2</div> <div></div> <div></div> <div></div>
n=3		<div>Pile 1</div> <div>D3</div> <div>D4</div>	<div>Pile 2</div> <div></div> <div></div> <div></div> <div></div>	<div>Pile 3</div> <div>D1</div> <div>D2</div> <div></div> <div></div>
n=4		<div>Pile 1</div> <div>D4</div>	<div>Pile 2</div> <div>D3</div> <div></div> <div></div> <div></div>	<div>Pile 3</div> <div>D1</div> <div>D2</div> <div></div> <div></div>
n=5		<div>Pile 1</div> <div>D1</div> <div>D4</div>	<div>Pile 2</div> <div>D3</div> <div></div> <div></div> <div></div>	<div>Pile 3</div> <div>D2</div> <div></div> <div></div> <div></div>
n=6		<div>Pile 1</div> <div>D1</div> <div>D4</div>	<div>Pile 2</div> <div>D2</div> <div>D3</div> <div></div> <div></div>	<div>Pile 3</div> <div></div> <div></div> <div></div> <div></div>
n=7		<div>Pile 1</div> <div>D4</div>	<div>Pile 2</div> <div>D1</div> <div>D2</div> <div>D3</div>	<div>Pile 3</div> <div></div> <div></div> <div></div> <div></div>

n=8		<table><tr><th>Pile 1</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td>D1</td></tr><tr><td>D2</td></tr><tr><td>D3</td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D4</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pile 1					Pile 2	D1	D2	D3		Pile 3	D4			
Pile 1																	
Pile 2																	
D1																	
D2																	
D3																	
Pile 3																	
D4																	
n=9		<table><tr><th>Pile 1</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td>D2</td></tr><tr><td>D3</td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D1</td></tr><tr><td>D4</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pile 1					Pile 2	D2	D3			Pile 3	D1	D4		
Pile 1																	
Pile 2																	
D2																	
D3																	
Pile 3																	
D1																	
D4																	
n=10		<table><tr><th>Pile 1</th></tr><tr><td>D2</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td>D3</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D1</td></tr><tr><td>D4</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pile 1	D2				Pile 2	D3				Pile 3	D1	D4		
Pile 1																	
D2																	
Pile 2																	
D3																	
Pile 3																	
D1																	
D4																	
n=11		<table><tr><th>Pile 1</th></tr><tr><td>D1</td></tr><tr><td>D2</td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td>D3</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D4</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pile 1	D1	D2			Pile 2	D3				Pile 3	D4			
Pile 1																	
D1																	
D2																	
Pile 2																	
D3																	
Pile 3																	
D4																	
n=12		<table><tr><th>Pile 1</th></tr><tr><td>D1</td></tr><tr><td>D2</td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D3</td></tr><tr><td>D4</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pile 1	D1	D2			Pile 2					Pile 3	D3	D4		
Pile 1																	
D1																	
D2																	
Pile 2																	
Pile 3																	
D3																	
D4																	
n=13		<table><tr><th>Pile 1</th></tr><tr><td>D2</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td>D1</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D3</td></tr><tr><td>D4</td></tr><tr><td></td></tr><tr><td></td></tr></table>	Pile 1	D2				Pile 2	D1				Pile 3	D3	D4		
Pile 1																	
D2																	
Pile 2																	
D1																	
Pile 3																	
D3																	
D4																	
n=14		<table><tr><th>Pile 1</th></tr><tr><td>D1</td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D2</td></tr><tr><td>D3</td></tr><tr><td>D4</td></tr><tr><td></td></tr></table>	Pile 1	D1				Pile 2					Pile 3	D2	D3	D4	
Pile 1																	
D1																	
Pile 2																	
Pile 3																	
D2																	
D3																	
D4																	
n=15		<table><tr><th>Pile 1</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 2</th></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr><tr><td></td></tr></table> <table><tr><th>Pile 3</th></tr><tr><td>D1</td></tr><tr><td>D2</td></tr><tr><td>D3</td></tr><tr><td>D4</td></tr></table>	Pile 1					Pile 2					Pile 3	D1	D2	D3	D4
Pile 1																	
Pile 2																	
Pile 3																	
D1																	
D2																	
D3																	
D4																	

Chapitre 3

Etude Expérimentale

3.1 Tableau Des Expérimentations

Nombre de disque (n)	5	10	15	20	25	30	35	40	45	50	...	100
Temps d'exécution (s)	0.000	0.000	0.000	0.023	0.784	25.946	810.640	//	//	//		//
Nombre des déplacements effectués	31	1023	32767	1048575	33554431	1073741823	34359738367	//	//	//		//

3.2.2 Simulation de la complexité spatiale théorique

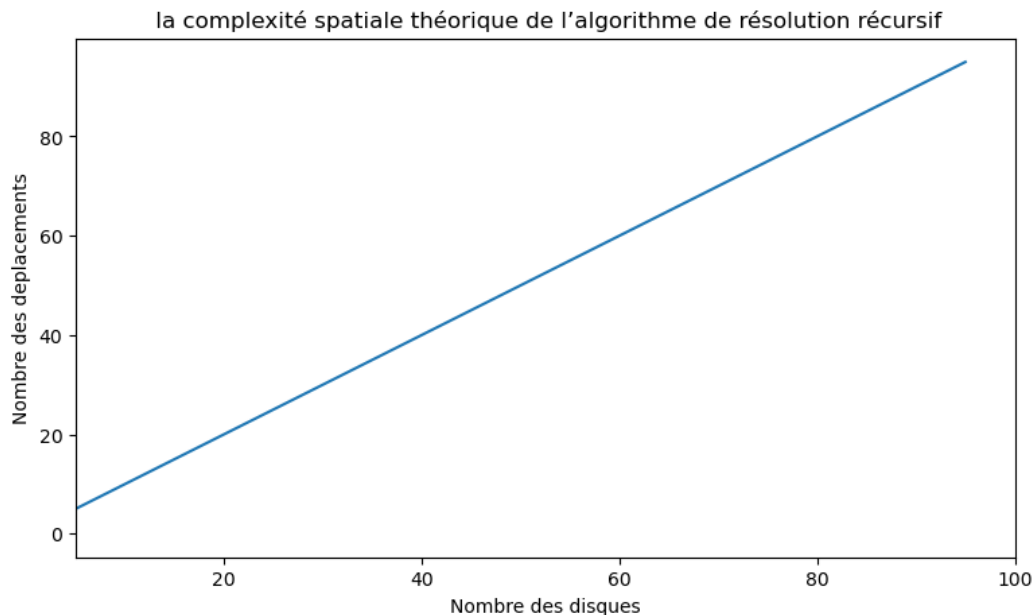


FIGURE 3.2 – Simulation de la complexité spatiale théorique de l'algorithme de résolution

3.3 Le meilleur, moyen et pire cas de l'algorithme de résolution

3.3.1 Le meilleur, moyen et pire cas de la complexité spatiale

Pour chaque appel de la fonction hanoi la solution du problème de taille k est stocké dans une pile et l'espace pri est indépendant de l'appel précédent donc on peut réutiliser l'espace de 1er appel récursif alors la complexité spatiale de tour d'hanoi est $O(n)$ dans tous les cas.

3.3.2 Le meilleur, moyen et pire cas de la complexité temporelle

Le temps pris par un algorithme pour les Tours de Hanoi sera proportionnel au nombre de fois où nous déplaçons un disque car dans ce cas la un mouvement élémentaire consiste à déplacer un disque d'une tige à une autre.

Soit $T(n)$ le nombre minimum de déplacements de disques nécessaires pour résoudre une instance Tours de Hanoi avec n disques. Si $n > 1$, $T(n)$ est égal à $2 T(n-1) + 1$ car nous pouvons d'abord déplacer récursivement les $n-1$ premiers disques vers la tige auxiliaire, déplacer le plus grand disque vers la tige de destination, puis déplacer récursivement la pile de la tige auxiliaire vers la tige de destination.

Le nombre minimum de déplacements de disques dans tous les cas sera toujours le résultat de cette formule récursive et uniquement dépendant du paramètre 'n' le nombre des disques en entrée car il n'y a qu'un seul "cas" pour chaque taille d'entrée avec une seule contrainte sur les disques d'être trié suivant un ordre croissant donc la complexité temporelle dans le meilleur, moyen et pire cas est $O(2^n)$ qui est obtenue par l'élimination de la récursivité comme nous l'avons déjà montré dans la partie II.

3.4 Analyse des résultats

En observant les graphes nous remarquons que pour un nombre de disques modéré inférieur à 35 le problème de tour d'hanoi peut être résolu dans un temps d'exécution acceptable qui dépend de la performance de la machine utilisée mais en dépassant le seuil des 45 le temps d'exécution croît très rapidement lorsque n augmente car le nombre minimum de déplacements requis pour une instance des Tours de Hanoi avec n disques est de $2^n - 1$ qui correspond à la complexité exponentielle $O(2^n)$.

En outre, l'espace requis par l'algorithme de résolution est linéaire ce qui fait que la complexité temporelle est le seul obstacle majeur empêchant d'appliquer cette solution.

Pour approximer l'importance de cette complexité temporelle, supposons qu'il nous faille une seconde pour déplacer un disque d'une tige à une autre tige. Alors, pour résoudre une instance avec 64 disques, il nous faudra environ 585 milliards d'années avec les machines les plus performantes d'aujourd'hui !

Chapitre 4

Conclusion

Le jeu de Tour d'Hanoi est l'un des problèmes mathématiques classiques qui illustre la puissance et le pouvoir de la récursion ainsi que les limites de l'informatique actuellement.

Dans ce rapport nous avons proposé deux solutions itérative et récursive à ce problème et nous les avons étudié et analysé théoriquement aussi démontré empiriquement leurs complexité, caractéristiques, et limitations sur nos machines.

Chapitre 5

Références

[1] Wikipedia .

[2] Cours Algorithmique et Complexite - Prof DERIAS.H .

Chapitre 6

Annexe :code source

6.1 Repartition des taches

Member	Tache
BEBNBACHIR Mohamed Amir	la conception de la solution iterative de hanoi + l'implementation en langage C + etude experimentale et analyse des resultats de la version iterative
BOUCHOUL Bouchra	L'algorithme de verification avec calcul de sa complexite + Presentation d'une instance du probleme avec la solution (exemple)
KHEMISSI Maroua	Historique et presentation du probleme + Definition formelle du probleme+ Presentation de l'algorithme de resolution avec complexite theorique + representation de la modelisation de la solution
MEDJKOUNE Roumaissa	le programme de l'algorithme de resolution recursive en langage c + Son étude expérimentale + Analyse des résultats

6.2 algorithme.c

Le programme c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #define MAX 100
5  #include <time.h>
6  //pile
7  struct discs {
8      int size;
9  };
10 typedef discs *disc;
11 //les piles
12 typedef struct pile
13 {
14     disc items[MAX];
15     int top;
16     int taille;
17     int num;
18 } pile;
19
20 typedef struct Tour{
21     int sizej;
22     pile *A;
23     pile *B;
24     pile *C;
25 }Tour;
26 typedef Tour *TourH;
27
28
29 //les fonctions des piles
30 void initPile(pile *p,int num)
31 {
32     p->top = -1;
33     p->taille = 0;
34     p->num=num;
35 }
36 int isempty(pile *p) {
37     if (p->top == -1)
```

```

38         return 1;
39     else
40         return 0;
41 }
42 int pilePleine(pile *s)
43 {
44     if (s->top == MAX - 1)
45         return 1;
46     else
47         return 0;
48 }
49 void push(pile *p, disc bt) {
50
51     if (pilePleine(p))
52     {
53         printf("la pile est pleine");
54     }
55     else
56     {
57         p->top++;
58         p->items[p->top] = bt;
59     }
60     p->taille++;
61
62 }
63 disc pop(pile *p) {
64     if (isempty(p)){
65         return NULL;
66     }
67     p->taille--;
68     disc val = p->items[p->top];
69     p->top--;
70     return val;
71 }
72
73 ///Initialiser le jeux en precisant le nombre initial des disques
74 TourH initHanoi(int size){
75     TourH Tr= (TourH)malloc(sizeof(TourH));
76     pile *A = (pile *)malloc(sizeof(pile));
77     initPile(A,1);

```

```

78     pile *B = (pile *)malloc(sizeof(pile));
79     initPile(B,2);
80     pile *C = (pile *)malloc(sizeof(pile));
81     initPile(C,3);
82     int i;
83     for (i=size;i>0;i--){
84         disc d = (disc)malloc(sizeof(disc));
85         d->size=i;
86         push(A,d);
87     }
88     Tr->sizej=size;
89     Tr->A=A;
90     Tr->B=B;
91     Tr->C=C;
92     return Tr;
93 }
94
95 ///fonction pou deplacer les disque entre les piles
96
97 void move(pile *p, pile *k,int* dep){
98     disc d;
99     d=pop(p);
100    push(k,d);
101    (*dep) += 1;
102
103 }
104
105 ///fonctions tour d'hanoi
106 void Hanoi(int n,pile *A,pile *B,pile *C,int* dep){
107     if (n>0){
108         Hanoi(n-1,A,C,B,dep);
109         move(A,C,dep);
110         Hanoi(n-1,B,A,C,dep);
111
112     }
113 }
114
115 void moveI(pile * from ,pile * to){
116
117     if (isempty(to))

```

```

118     {
119         push(to,pop(from));
120     }
121
122     else if (isempty(from))
123     {
124         push(from,pop(to));
125     }
126
127     else if (from->items[from->top]>to->items[to->top])
128     {
129         push(from,pop(to));
130     }
131
132     else
133     {
134         push(to,pop(from));
135     }
136 }
137 void iterativeHanoi(int n,pile* start,pile* end,pile* inter){
138
139     int total_num_of_moves = pow(2, n) - 1;
140
141
142     if (n % 2 != 0)
143     {
144
145         for (int i = 1; i <= total_num_of_moves; i++)
146         {
147             if (i % 3 == 1)
148                 moveI(start,end);
149
150             else if (i % 3 == 2)
151                 moveI(start,inter);
152
153             else if (i % 3 == 0)
154                 moveI(inter,end);
155         }
156     }else{
157         for (int i = 0; i < total_num_of_moves; i++)

```



```

158     {
159         if (i % 3 == 1)
160             moveI(start,end);
161
162         else if (i % 3 == 2)
163             moveI(end,inter);
164
165         else if (i % 3 == 0)
166             moveI(inter,start);
167     }
168 }
169 }
170 void TourHanoi(TourH Tr,int* dep){
171     Hanoi(Tr->sizej,Tr->A,Tr->B,Tr->C,dep);
172 }
173
174 int main(){
175     //enregistrement des executions
176
177     clock_t t1,t2;
178     double time;
179     int j=5;
180     FILE *f;
181     f = fopen("nbDepHanoiRec.txt", "a");
182     if (f != NULL)
183         { printf(" le fichier existe \n");}
184     else
185         { printf("Impossible d'ouvrir le fichier \n");}
186
187     while(j<100){
188         int d=0;
189         TourH jeu= initHanoi(j);
190         t1=clock();
191         TourHanoi(jeu,&d);
192         t2=clock();
193         time=(float)(t2 - t1) / CLOCKS_PER_SEC;
194         fprintf(f,"%d %d \n",j,d);
195         j+=5;
196     }
197

```

```
198
199  return 0;
200 }
201
```
