Série de Travaux Pratiques n° 4 (TP n° 4)

Algorithmes de complexité temporelle exponentielle $O(a^n)$ (a>1)

Note : Un minimum de connaissances en programmation est indispensable pour le suivi des séances des travaux pratiques.

L'objet de ce TP est l'étude expérimentale de l'algorithme de résolution du problème des ″tours de Hanoi″. On s'intéresse à l'algorithme récursif. C'est un problème classique en informatique. Il montre la puissance et la lisibilité des algorithmes définis de façon récursive. On utilise le langage de programmation C.

Le problème des ″tours de Hanoi″

Il est décrit dans wikipédia (encyclopédie universelle libre) comme suit :

C'est un jeu de réflexion imaginé par le mathématicien français *Édouard Lucas*, et consistant à déplacer des disques de diamètres différents d'une tour de ″départ″ à une tour d' ″arrivée″ en passant par une tour ″intermédiaire″ et ceci en un minimum de coups, tout en respectant les 2 règles suivantes :

1) on ne peut déplacer plus d'un disque à la fois,
2) on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide. On suppose que cette dernière règle est également respectée dans la configuration de départ.

Il est publié dans le tome 3 de ses *Récréations mathématiques*, parues à titre posthume en 1892. 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 mondes "[].

Rem : On note qu'un jeu à 64 disques requiert un minimum de 2^{64}-1 déplacements. En admettant qu'il faille 1 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).

1- Ecrire un algorithme récursif qui résout le problème des ″tours de Hanoi″. On suppose qu'il y a n disques à transférer (n est un entier naturel, n>=1).
2.1- Calculer la complexité temporelle de cet algorithme, notée f(n). (cette question est supposée être traitée en séances de TD, sinon la laisser pour les séances ultérieures).
2.2- Calculer la complexité spatiale de cet algorithme, notée s(n). (là aussi, on a la même remarque que celle de la question 2-1).
3- Ecrire avec le langage C le programme correspondant.
4- Mesurer les temps d'exécution T pour un échantillon de données de la variable n et représenter les résultats sous la forme d'un tableau (ci-dessous).

n	**…**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**
T(s)																				

n	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**…**	**…**	**…**	**64**
T(s)																

5- Développer un programme de mesure du temps d'exécution du programme qui a en entrée les données de l'échantillon ci-dessus et en sortie les temps d'exécution. Les données et les mesures du temps sont à enregistrer dans des tableaux notés respectivement Tab1 et Tab2.
6- Représenter par un graphe, noté $G_f(n)$, les variations de la fonction de la complexité temporelle en fonction de n ; et par un autre graphe, noté $G_T(n)$, les variations du temps d'exécution T(n) en fonction de n. Utiliser pour cela un logiciel graphique tel que excel.

7- Interprétation des résultats.
7.a- Les mesures du temps obtenues correspondent-elles au meilleur cas, au pire cas, au cas moyen ou au cas exact ?
7.b- Que remarque-t-on sur les données de l'échantillon et sur les mesures obtenues ? Peut-on déduire, même de façon approximative, une fonction T(n) reliant T et n ; c'est-à-dire une fonction T(n) permettant de déterminer directement la valeur de T à partir de n.
Ind : comparer chaque nombre n avec le suivant ; et chaque mesure du temps avec la suivante.
7.c- Comparer entre la complexités théorique et la complexité expérimentale (çàd., les mesures expérimentales). Les prédictions théoriques sont-elles compatibles avec les mesures expérimentales ?

Rem : Il est intéressant de chercher une v**ersion itérative pour cet algorithme.**

Le Corrigé du TP n°4

1- Ecrire un algorithme récursif qui résout le problème des ″tours de Hanoi″. On suppose qu'il y a n disques à transférer (n est un entier naturel, n>=1).

Solution :

L'algorithme récursif du problème des « tours de Hanoi » est basé sur les idées suivantes :

- Les 3 tours sont notées A, B et C.
- Les disques chargés sur la tour de départ A doivent être déplacés sur la tour d'arrivée B. On utilise la tour C comme une tour intermédiaire.
- Au cours des différentes étapes de déplacement des disques entre les 3 tours, ces dernières peuvent changer de rôle :
  + la tour de départ A peut devenir la tour d'arrivée ou la tour intermédiaire ;
  + la tour d'arrivée B peut devenir la tour de départ ou la tour intermédiaire ;
  + la tour intermédiaire peut devenir la tour de départ ou la tour d'arrivée.
- Pour déplacer n disques de la tour de départ A vers la tour d'arrivée B, on effectue les 3 actions suivantes :
  1- Déplacer (n-1) disques de la tour de départ A vers la tour intermédiaire C (qui devient la tour d'arrivée) ;
  2- Déplacer le disque restant de la tour de départ A vers la tour d'arrivée B ;
  3- Déplacer (n-1) disques de la tour intermédiaire C (qui devient la tour de départ) vers la tour d'arrivée B (qui est la tour d'arrivée) en utilisant la tour de départ A (qui devient la tour intermédiaire).
- Les actions 1 et 3 ne peuvent pas être réalisées immédiatement. Elles constituent des appels récursifs avec (n-1) disques de l'algorithme des « tours de Hanoi ». Chacune de ces 2 actions va générer la même suite des 3 actions ci-dessus mais avec (n-2) disques, (n-3) disques, … etc., jusqu'à (0) disques.
- L'action 2 est une instruction terminale qui ne génère pas d'appels récursifs.

L'algorithme que l'on développe, noté Hanoi, est présenté sur la figure 1 ci-dessous.

```
Algorithme Hanoi ;              //Algorithme du problème des « tours de Hanoi »
                                //avec n diques (n>=1)
Var n, nb : entier ;            //n = nombre de disques
                                //nb = nombre de déplacement des disques
 Début
   //Partie 1: Lecture des données
1  ecrire ("\nDonner le nombre de disques: n = ");
2  lire (n);

   //Partie 2: Traitement
3  nb=0;
4  f_hanoi(n, 1, 3, 2);         //appel de la fonction f_hanoi(n, 1, 3, 2)

   //Partie 2: Sertie des résultats
5  ecrire ("le nombre des déplacements nb = ", nb);

Fin. //fin de l'algorithme  Hanoi

//Définition de la fonction f_hanoi(int n, int A, int C, int B)
void f_hanoi(int n, int A, int C, int B)
Début
6  si (n >= 1)
     alors début
7                f_hanoi(n-1, A, B, C);
8                printf("\nDéplacer le disque restant de %d vers %d\n", A, B);
                                                        //La tour A devient vide
9                nb=nb+1;
10               f_hanoi(n-1, C, A, B);
           fin;
Fin ; //fin de la fonction f_hanoi(int n, int A, int C, int B)
```

Figure 1. Algorithme du problème de la tour de Hanoi avec disques (n>=1) (algorithme récursif)

2.1- Calculer la complexité temporelle de cet algorithme, notée f(n).

Solution :

Rappel : La complexité temporelle d'un algorithme désigne la mesure du temps nécessaire pour exécuter cet algorithme. Elle est donnée sous la forme d'une fonction dont les variables sont les données en entrée de l'algorithme. Cette fonction peut être une expression mathématique exacte mais cette forme est rarement utilisée. Elle peut être aussi une expression mathématique en notation asymptotique appelée aussi notation Landau. Cette deuxième forme est la forme couramment utilisée.

On montre, dans ce qui suit, **le calcul pratique de la complexité temporelle d'un algorithme récursif**. On doit ajouter la règle 10 (voir ci-dessous) par rapport au procédé de calcul **de la complexité temporelle d'un algorithme itératif** présenté aux TP n°1 et TP n°2. On utilise les règles suivantes :

1- La complexité temporelle est calculée en fonction des données en entrée de l'algorithme.
2- Le calcul de la complexité temporelle peut se ramener au calcul du nombre d'instructions exécutées par cet algorithme qu'on appelle aussi fréquence d'exécution des instructions de

l'algorithme car la fonction T(n) qui mesure le temps d'exécution d'un algorithme et la fonction F(n) qui mesure le nombre d'instructions exécutées par cet algorithme appartiennent à la même classe de complexité (voir la remarque déjà donnée dans le corrigé du TP1).

3- les principales classes de complexité sont :

$$O(n), O(n^2), O(n^3), O(\log(n)), O(nlog(n)), O(a^n)$$

4- Pour calculer la complexité temporelle d'un algorithme, on doit :

a) calculer les fréquences d'exécutions de chacune des instructions de l'algorithme ;
b) Dans le cas d'une instruction conditionnelle complète ayant les 2 branches alors et sinon, on doit calculer la somme de la fréquence de la condition de l'instruction conditionnelle et le maximum des fréquences entre ses 2 branches dans le pire cas ou le minimum des fréquences dans le meilleur cas (car une seule branche est exécutée).
c) Dans le cas d'une instruction de répétition (pour, tant que ou répéter), on doit calculer la somme de la fréquence de la condition de l'instruction de la répétition et des fréquences des instructions contenues dans le corps (ou le bloc) de l'instruction de la répétition ;
d) Calculer la somme des fréquences des instructions de l'algorithme.

5- Généralement, le calcul de la complexité temporelle exacte n'est pas possible car cela dépend des données en entrée de l'algorithme, et on se restreint alors au calcul des complexités au pire cas, au moyen cas et au meilleur cas. La complexité au pire cas est celle qui est la plus utilisée.

6- Les instructions exécutables correspondent aux opérations exécutables par le processeur de manière indivisible :

- les opérations arithmétiques (+, -, *, /) ;
- les opérations de condition (==, !=, <, <=, >, >=) ;
- les opérations d'entrée/sortie (lire(), ecrire()).

7- Les fonctions prédéfinies comme sin, cos, log, modulo, etc. sont supposées se comporter comme des opérations exécutables par le processeur de manière indivisible (cette hypothèse ne change pas la complexité de l'algorithme).

8- On suppose que toutes les instructions prennent la même durée de temps d'exécution, ce qui permet de faire l'addition de leurs fréquences d'exécutions (là aussi, cette hypothèse ne change pas la complexité de l'algorithme).

9- Le meilleur cas d'un algorithme correspond au nombre minimal du nombre d'instructions exécutées par cet algorithme alors que le pire cas correspond au nombre maximal du nombre d'instructions exécutées.

10- Avec les algorithmes récursifs, on doit tenir compte de chaque appel récursif d'une fonction qui génère de façon dynamique et donc de façon transparente au programmeur l'exécution de la même fonction avec de nouveaux paramètres d'appels et de nouvelles variables locales de la fonction. Dans la pratique, on calcule le nombre d'appels récursifs en résolvant une équation de récurrence.

Soit L le nombre d'instructions de l'algorithme Hanoi (ci-dessus), f_i (i=1…L) la fréquence d'exécution de l'instruction I_i (i=1…L), et F la somme de ces fréquences.
On a: L = 10 instructions.

1- On a pour l'algorithme principal Hanoi :

$f_1 = 1$;
$f_2 = 1$;
$f_3 = 1$;
$f_4 = 1 + H(n)$; //1 opération pour l'instruction d'appel de la fonction f_Hanoi(n, 1, 3, 2)
//H(n) mesure la complexité temporelle de la fonction f_Hanoi(n, 1, 3, 2)
$f_5 = 1$;

Donc, F est :

$$F(n)=\sum_{1}^{6} f_i=f_1+f_2+f_3+f_4+f_5$$

$$\Rightarrow F(n)=1+1+1+(1+H(n))+1$$

$$\Rightarrow F(n)=H(n)+5 \qquad (1)$$

2- Pour le calcul de $H(n)$, on procède comme suit :
2.1- On calcule les fréquences des instructions de la fonction f_Hanoi(…) (instructions 6 à 10) :

$f_6=1;$
$f_7=1+H(n-1);$ //1 opération pour l'instruction d'appel de la fonction f_Hanoi(n, A, B, C)
//H(n-1) mesure la complexité temporelle de la fonction f_Hanoi(n-1, A, B, C)
$f_8=1;$
$f_9=2;$ //2 opérations : = et +
$f_{10}=1+H(n-1);$ //1 opération pour l'instruction d'appel de la fonction f_Hanoi(n, C, A, B)
//H(n-1) mesure la complexité temporelle de la fonction f_Hanoi(n-1, C, A, B)

2.2- On calcule la somme de ces fréquences :

$$H(n)=\sum_{6}^{10} f_i=f_6+f_7+f_8+f_9+f_{10}$$

$$\Rightarrow H(n)=1+(1+H(n-1))+1+2+(1+H(n-1))$$

$$\Rightarrow H(n)=2H(n-1)+6 \qquad (2)$$

Et $H(0)=1$ //équation de récurrence
//si n=0 alors la fonction f_Hanoi(…) exécute juste le
//test (n>=1)

2.3- On résout l'équation de récurrence (2) avec la méthode de substitution comme suit :

$$H(n)=2H(n-1)+6$$

$$\Rightarrow H(n)=2(2H(n-2)+6)+6$$

$$\Rightarrow H(n)=2^2H(n-2)+2^1*6+6$$

$$\Rightarrow H(n)=2^2(2H(n-3)+6)+2^1*6+6$$

$$\Rightarrow H(n)=2^3H(n-3)+2^2*6+2^1*6+6$$

$$\Rightarrow \ldots$$

$$\Rightarrow H(n)=2^nH(n-n)+2^{n-1}*6+\ldots+2^1*6+2^0*6$$

$$\Rightarrow H(n)=2^nH(0)+6(2¿¿n-1+\ldots+2^1+2^0)¿$$

$$\Rightarrow H(n)=2^n*1+6\left(\frac{2^n-1}{2-1}\right)=2^n+6(2^n-1)=7*2^n-6 \qquad (3)$$

La relation (3) représente la solution en notation exacte de l'équation de récurrence (2).

On montre aisément que :

$$H(n)=7*2^n-6$$

$$\Rightarrow H(n)\leq 7*2^n-6+6=7*2^n$$

$$\Rightarrow H(n)=O(2^n) \qquad (4)$$

//H(n) en notation asymptotique

On conclut alors que la fonction f_Hanoi(…) a une complexité temporelle exponentielle.

3- On calcule enfin la fonction $F(n)$ en y remplaçant la solution $H(n)$ dans la relation (1) :

$F(n)=H(n)+5$

$\Rightarrow$

❑ $F(n)=(7*2¿¿n-6)+5=7*2^n-1¿$ (5)

//F(n) en notation exacte

On montre aisément aussi que :

$F(n)=7*2^n-1$

$\Rightarrow$

❑ $F(n)\leq 7*2^n-6+1=7*2^n$

$\Rightarrow$

❑ $F(n)=O(2^n)$ (6)

//F(n) en notation asymptotique

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

En conséquence, la complexité temporelle CT de l'algorithme Hanoi est:

$$\begin{cases} CT(n)=F(n)=7*2^n-1 \; en\; notation\; exacte \\ CT(n)=F(n)=O(2^n) en\; notation\; asymptotique \end{cases}$$

3- Ecrire avec le langage C le programme correspondant.

Solution :

Le programme est le suivant (figure 2=) :

```
//USTHB, Année 2014/2015
//1ère année Master  Informatique, Semestre 1
//Module : Conception et Complexité des Algorithmes
//Série Travaux Pratiques n°5 (TP n°5)
//Archivage: D:\BC45\M1_RSD_Complexité_2015\TP5.cpp)
```

```
//Ce programme résout le problème des tours de Hanoi.
//Ce programme est récursif

//développé par Mr. AMANI

#include <stdlib.h>
#include <stdio.h>

//Déclaration de la fonction hanoi(int, int, int, int)
void hanoi(int, int, int, int);

int nb;                         //nb est une variable globale

int main(void)
{
  int n;                        //n = nombre de disques
  //int nb;                     //nb = nombre de déplacement des disques
  clock_t t1, t2;               //clock_t désigne le type temps
  double delta;                 //delta mesure la durée d'exécution du programme entre
                                //les points t1 et t2
  //Partie 1: Lecture des données
  printf("\n\n\n Série de Travaux Pratiques n°5 (TP n°5) \n");
  printf("\nDonner le nombre de disques: n = ");
  scanf("%d", &n);              //le format "%d" permet de lire le type entier en
                                //en simple précision sur 2 octets = 16 bits
  //Partie 2: Traitement
  nb=0;
  t1=clock();                   //La variable t1 reçoit la valeur du temps fournie par la
                                //la fonction clock(). C'est le début de la mesure du temps.
  hanoi(n, 1, 3, 2);            //appel de la fonction hanoi(n, 1, 3, 2)
                                //On suppose : A=1, B=2 et C=3
  t2=clock();                   //La variable t2 reçoit la valeur du temps fournie par la
                                //la fonction clock(). C'est la fin de la mesure du temps.
```

```
  delta=(double)(t2-t1)/CLOCKS_PER_SEC;          //formule permettant de calculer la
                                                 //durée d'exécution du programme entre les points t1 et t2

  printf("\n\n nb=%d déplacements", nb);
  printf("\n\nEntrer une touche quelconque pour terminer le programme: ");
  getchar(); getchar();
  return(0);
}//fin du programme

//Définition de la fonction hanoi(int n, int A, int C, int B)
void hanoi(int n, int A, int C, int B)
{
  if (n >= 1)
    {
       hanoi(n-1, A, B, C);
       printf("\nDéplacer le disque restant de %d vers %d\n", A, B);
                                                 //La tour A devient vide
       nb=nb+1;                  //Lors des mesures du temps, ces 2 instructions doivent être
                                 //bloquées (par exemple mises en commentaires)
       hanoi(n-1, C, A, B);
    }
}//fin de la fonction hanoi(int n, int A, int C, int B)
```

Figure 2. Programme du problème de la tour de Hanoi avec disques (n>=1) (programme récursif)

4- Mesurer les temps d'exécution T pour un échantillon de données de la variable n et représenter les résultats sous la forme d'un tableau (ci-dessous).

n	…	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
T(s)																				

n	29	30	31	32	33	34	35	36	37	38	39	40	…	…	…	64
T(s)																

Solution :

A faire par les étudiants.

5- Développer un programme de mesure du temps d'exécution du programme qui a en entrée les données de l'échantillon ci-dessus et en sortie les temps d'exécution. Les données et les mesures du temps sont à enregistrer dans des tableaux notés respectivement Tab1 et Tab2.

Solution :

A faire par les étudiants.

6- Représenter par un graphe, noté $G_f(n)$, les variations de la fonction de la complexité temporelle en fonction de n ; et par un autre graphe, noté $G_T(n)$, les variations du temps d'exécution T(n) en fonction de n. Utiliser pour cela un logiciel graphique tel que excel.

Solution :

A faire par les étudiants.

7- Interprétation des résultats.
7.a- Les mesures du temps obtenues correspondent-elles au meilleur cas, au pire cas, au cas moyen ou au cas exact ?
7.b- Que remarque-t-on sur les données de l'échantillon et sur les mesures obtenues ? Peut-on déduire, même de façon approximative, une fonction T(n) reliant T et n ; c'est-à-dire une fonction T(n) permettant de déterminer directement la valeur de T à partir de n.
Ind : comparer chaque nombre n avec le suivant ; et chaque mesure du temps avec la suivante.
7.c- Comparer entre la complexités théorique et la complexité expérimentale (çàd., les mesures expérimentales). Les prédictions théoriques sont-elles compatibles avec les mesures expérimentales ?

Solution :

A faire par les étudiants.