Rapport de projet : Exponentiation rapide

Année : 2016/2017

S4 MA – IUT de Villetaneuse

Groupe : Gatto Nathan, Wong Jason et Dos Santos Jérémy

Tuteur : M. Bonino

Sommaire :

Introduction

1 – Présentation du projet

Tout d'abord nous allons commencer par expliquer ce qu'est l'exponentiation rapide. C'est une méthode mathématique utilisée pour calculer rapidement de grandes puissances entières qu'on peut éventuellement mettre à un certain modulo. On l'appelle aussi « Square and multiply ». Cette méthode est très utilisée en informatique notamment pour des méthodes de cryptage (RSA) nécessitant efficacité et rapidité.

2- Contexte du projet

Ce projet présente des algorithmes de calcul rapide de nombres de la forme m^n, où m et n sont deux entiers positifs. De tels algorithmes sont utilisés dans de nombreux domaines où l'on a besoin de manipuler de grands entiers, par exemple en cryptographie.

3-Objectif du projet

Analyser et programmer plusieurs algorithmes d'exponentiation rapide de nombres entiers (et de classes de congruence d'entiers). Le but était de concevoir plusieurs façons de calculer des exponentiations et de comparer leur rapidité et leurs complexités afin de savoir lequel est le plus optimal et donc le moins couteux.
Comprendre comment chaque algorithme tire parti des égalités mathématiques basiques pour gagner en efficacité de calcul.

Développements mathématiques

1er algorithme (Écriture de Hörner) :

On souhaite faire le calcul rapide d'une puissance de la forme m^n, pour cela, on veut tout d'abord décomposer la puissance c'est-à-dire n. On commence passer le nombre n à la base 2 (en binaire), qui correspond à un polynôme de la forme :

$$P(x) = a_d x^d + a_{d-1}x^{d-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0 x^0$$

Où : - les a_i correspondent au nombre de rang i donnés dans l'écriture de n.
- x correspond à la base, ici on utilise la base 2.
- d le nombre de chiffres dans l'écriture binaire de n.

Puis on met en facteur les x le plus possible.

$$P(x) = [a_d x^{d-1} + a_{d-1}x^{d-2} + \cdots + a_2 x^1 + a_1]x + a_0$$

Puis

$$P(x) = [a_d x^{d-2} + a_{d-1}x^{d-3} + \cdots + a_2]x + a_1]x + a_0$$

On continue jusqu'à avoir l'écriture de Hörner du polynôme P

$$P(x) = [\ldots [a_d x + a_{d-1}]x + a_{d-2}]x + \cdots + a_2]x + a_1]x + a_0$$

Enfin, on va utiliser cette écriture pour notre calcul de puissance. Pour cela, on remplace n par son écriture de Hörner et on calcule $m^{P(x)}$.

Exemple :

On veut calculer 3^9

Le nombre 9 correspond à 1001 en base 2, soit au polynôme :

$$P(x) = x^3 + 1$$

Où l'on va mettre les x en facteurs :

$$P(x) = (x^2)x + 1$$

Puis

$$P(x) = ((x)x)x + 1$$

On obtient donc $3^9 = 3^{((x)x)x+1} = \left(3^{x^{x^x}}\right) * 3^1$ où x correspond à la base utilisée, soit 2 dans notre cas.

$$3^9 = \left(3^{2^{2^2}}\right) * 3^1 = 19683$$

Pseudo Code algorithme 1 : Voir annexe 1.

Explications du code :

Si la puissance n = 0, la taille de la chaine bin qui est la longueur du nombre en base 2 sera 0 et l'algorithme retournera 1 car $x^0 = 1$.
Sinon, on remarque que quelques soit le chiffre au rang i du nombre binaire, on fait toujours le même calcul : on met à la puissance x (ici x=2) le résultat, on fait une multiplication du nombre m seulement si le chiffre au rang i est 1. Donc on crée une boucle dans laquelle on calcule res=res^x et on vérifie si le chiffre au rang i est 1, si c'est le cas, on fait également l'opération res=res*m.

Question de l'énoncé :

1) A) Ici on travaille en base de 2, on a donc :

$$\mathrm{n} = \mathrm{P}(2)$$

B) Sachant que dans chaque boucle dans le pire des cas on a une addition et une multiplication qui comptent chacune pour 1 opérations on a grâce à (*)

$$\mathrm{d} = \log_2(\mathrm{n})$$

Donc on a :

$$\text{Le nombre d'opération} = 2\mathrm{d}$$

2) Pour justifier l'encadrement suivant il faut comprendre la chose suivante : quand on met un nombre en base 2, il est composé de 1 et/ou de 0. Donc grâce à la formule (*) alors 2^d est inférieur ou égale à n car seul le 1er facteur est à 1. 2^{d+1} est supérieur ou égale à n car tous les facteurs sont à 1.
On peut ensuite faire :

$2^d \leq n \leq 2^{d+1}$ $\quad d \leq log_2(n)$ $\qquad n \leq 2^{d+1} - 1$ $\qquad n + 1 \leq 2^{d+1}$

$log_2(n+1) - 1 \leq d$ $\qquad et\ dans\ ce\ cas\ ci :\ 4log_2(n+1) - 1 \leq d$

Car on a 4 opérations par boucle.

2nd algorithme (Puissance consécutives de 2) :

Pour ce second algorithme, on va utiliser les puissances successives de 2. Il est possible de décomposer une écriture de la forme m^n telle que :

$$m^n = (m^{a_d})^{2^d} * (m^{a_{d-1}})^{2^{d-1}} * \dots * (m^{a_1})^{2^1} * (m^{a_0})^{2^0}$$

où : - m^{a_i} vaut 1 ou m (selon que $a_i = 0$ ou $a_i = 1$).
- d est le nombre de chiffre dans l'écriture de n en base 2.

Exemple :

On veut calculer 3^9.

Le nombre 9 correspond à 1001 en base 2. On obtient donc :

$$3^9 = (3^1)^{2^3} * (3^0)^{2^2} * (3^0)^{2^1} * (3^1)^{2^0}$$

soit :

$$3^9 = 3^8 * 3 = 3^{2^{2^2}} * 3 = 19683$$

Pseudo Code algorithme : voir annexe 2.

Explication du code :

On inverse le sens de lecture en inversant le tableau des chiffres de l'écriture binaire pour simplifier le calcul plus tard. La variable puissancecons sera celle qui va garder les puissances consécutives de 2. On crée une boucle dans laquelle on vérifie si le chiffre de rang i est un 1, dans ce cas on calcule res=res*puissancecons. Dans tous les cas, on continue à mettre puissancecons à la puissance de 2 : puissancecons=puissancecons^2

3ème algorithme (parité des puissances) :

Pseudo Code algorithme : voir annexe 3.

Explications :

Le troisième algorithme programmé de manière récursive, consiste à utiliser les égalités basiques comme $m^n = (m^2)^{n/2}$ lorsque n est pair ou $m^n = (m^2)^{n-1/2}$ lorsque n est impair. Comme l'algorithme est récursif, il fallait trouver une condition d'arrêt : on avait le choix entre n=0 ou n=1. J'ai préféré garder le second choix car calculer un nombre puissance 0 est inutile dans le sens que le résultat sera toujours 1 et ça nous évite un appel supplémentaire (sachant qu'on passe toujours à n=1 avant n=0). La condition d'arrêt de l'algorithme est donc lorsque la puissance est égale à 1, on renvoie la valeur du nombre. La partie concrète restait à programmer.

En reprenant le TP d'algorithmique avancé, j'ai écrit le code en réfléchissant à ce que je faisais. La première chose à faire est de tester si la puissance est un nombre pair ou impair. Si elle est paire, on rappelle la fonction avec comme paramètre le nombre mis au carré et la puissance divisée par 2. Dans le cas où la puissance est impaire, on fait pareil que dans le cas précédent sauf que l'on change les paramètres : on garde le nombre au carré pour le premier paramètre mais on donne l'indice de puissance – 1 divisé par 2 afin de garder une puissance entière. Cet algorithme s'enchaîne jusqu'à ce que la puissance vaille 1.

La complexité de l'algorithme est dans le pire des cas en logarithme en base 2 de la puissance. En effet, à chaque passage dans la fonction, on divise soit par 2, soit on soustrait 1 avant de diviser par 2 (ce qui est plus rapide pour terminer l'algorithme). Le pire cas de figure serait donc que la puissance donnée soit paire : on retrouverait une puissance égale à 1 après plus d'itération que si l'on avait fait -1 avant de diviser par 2. Par conséquent la complexité se déclare en combien de fois on divise par 2 successivement avant d'obtenir 1, c'est-à-dire le log en base 2.

Exemple avec 164^{74} :

On voit que l'indice de puissance est pair, donc on relance la fonction avec 164^2 et 37 (74/2).

Seconde itération, la puissance est impaire. Cette fois, on appelle la fonction avec $(164^2)^2$ et 18 ((37-1)/2)

Ensuite, la puissance étant à nouveau paire, on appelle la fonction avec $((164^2)^2)^2$ et 9 (18/2)

De même avec $(((164^2)^2)^2)^2$ et 4 ((9-1)/2) puis avec $((((164^2)^2)^2)^2)^2$ et 2 (4/2)

Enfin l'algorithme arrive à son terme car le dernier appel se fait avec les paramètres suivant $(((((164^2)^2)^2)^2)^2)^2$ et 1 (2/2). Lors du prochain passage, la fonction retournera 164 car la puissance vaut 1 et on remonte la valeur aux anciens appels récursifs et le dernier résultat correspond à ce que l'on voulait au départ.

Présentation des différents outils informatique

1- PHP

Pour réaliser ce projet nous avons décidé d'utiliser le langage de programmation PHP, PHP est un langage de programmation libre multi plateforme, principalement utilisé pour produire des pages Web dynamiques via un serveur HTTP, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale. PHP est un langage impératif orienté objet.

PHP est un langage qui a été créé en 1994 par Rasmus Lerdorf. C'est un langage extrêmement utilisé, il a permis la création de site tels que Facebook, Wikipédia et bien plus encore.

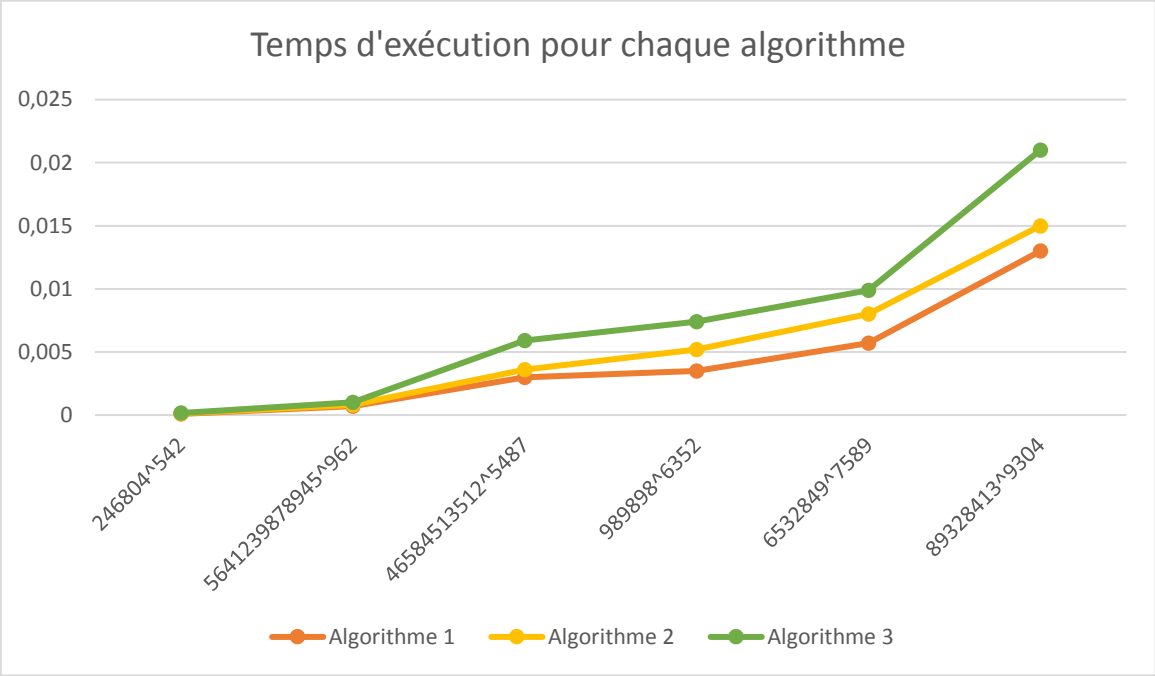
2- GMP

De plus nous avons utilisé une librairie de fonctions PHP qui se nomme GMP, *GNU Multiple Précision*, qui permet de manipuler des entiers d'une taille arbitraire. Parmi ces fonctions nous avons notamment utilisé *gmp_mod* une fonction qui permet de mettre un nombre de type GMP au modulo souhaiter. *gmp_mul* une fonction qui permet de multiplier deux nombres de type GMP.

3- WAMP

WAMP est une plate-forme de développement Web sous Windows pour des applications Web dynamiques à l'aide du serveur Apache2, du langage de scripts PHP et d'une base de données MySQL. (MAMP pour OS, LAMP pour Linux). Elle nous a permis de travailler avec GMP car cette librairie est déjà installée. Nous l'avons donc utilisé dans le but de tester et voir le rendu de notre code.

Présentation et commentaires des résultats obtenus



Voici un graphique représentant les temps d'exécution de chaque algorithme sur 6 exemples différents. Bien qu'on observe que les temps d'exécution sont plus importants pour le 3ème algorithme en moyenne, la différence n'est pas tellement significative.

Dans le premier algorithme, effectué avec les écritures binaires, on vérifie que la valeur du nombre et de la puissance sont correctes. Ensuite on a une condition et une boucle contenant une condition. La boucle compte la longueur du polynôme de Hörner, c'est à dire que la longueur dépend ici du nombre de bit à 1 et la complexité dans le pire des cas correspond à un grand nombre avec tous ses bits à 1. Le plus souvent, ce n'est pas ce qui se passe ce qui explique qu'il soit meilleur en termes de temps d'exécution.

Le second algorithme, se base aussi sur le nombre d'occurrence des puissances de 2 dans le nombre donnée, avec des puissants qui changent au lieu des coefficients. Cela revient au même dans le pire des cas, chaque terme sera comptabilisé et on retrouve une complexité en log2(n). En pratique, l'algorithme vérifie que la valeur du nombre et de la puissance sont correctes puis entre dans une boucle dont le nombre d'itération dépend de la longueur de l'écriture binaire du nombre. A l'intérieur de la boucle, on vérifie si le bit à cet endroit est actif et on vérifie si on est au début de la chaîne ou non, afin de retourner directement le nombre ou alors de multiplier le résultat par la puissance consécutive. On change cette puissance consécutive et on recommence l'itération.

Le troisième algorithme garde l'écriture standard dans ce programme et c'est peut être aussi un facteur à prendre en compte dans le calcul de son temps d'exécution. Le programme se base essentiellement sur la puissance et non sur le nombre : on va diviser par 2 la puissance et monter au carré le nombre à chaque itération, jusqu'à ce que la puissance soit égale à 1. Dans la pratique, on procède à la vérification de la valeur du nombre et de la puissance comme les deux autres algorithmes et on test si la puissance vaut 1. Si ce n'est pas le cas, on vérifie si elle est paire ou impaire et dans un cas comme dans l'autre, on rappelle la fonction avec de nouveaux paramètres.

Conclusion et commentaire personnels sur le projet

Durant ce projet, nous avons programmer un site web dynamique qui sert d'interface grâce au PHP couplé avec la librairie GMP afin de pouvoir passé outre la limite des entiers et pouvoir utilisé n'importe lequel des entiers saisis. Nous avons aussi appris à comprendre de nouvelles notations mathématiques, notamment les classes de congruences d'entier. Et enfin nous avons appris à nous familiarisés a de nouvelles notations mathématiques. En effet grâce à nos tests nous avons pu constater que l'algorithme le plus rapide est l'algorithme parité des puissances. Nous avons pu comparer toutes ces manières de calculer l'exponentiation rapide et voir leurs avantages et inconvénients. Ce travail nous a permis d'améliorer un précédent projet qui consistait à crypter et/ou décrypter et on utilisait l'exponentiation rapide pour RSA. On peut donc imaginer que des chercheurs peuvent rechercher de manières encore plus rapides de calculer l'exponentiation rapide afin de pouvoir casser des codes RSA. La plus grosse difficulté que nous avons rencontrée a été de comprendre le fonctionnement du polynôme de Hornër et de son utilisation. Après deux rendez-vous nous avons enfin réussi à nous lancer et à enfin voir des résultats.

Annexes

Annexe 1 : Pseudo code 1er algorithme (Hornër)

```
1       Variables :
2           m : Nombre a élevé à la puissance n
3           n : Puissance
4           x : base choisie
5           bin :  nombre n en base 2
6           taille : taille de la chaine bin
7           res : résultat initialisé à m
8
9       Algorithme :
10          Si taille == 0 :
11              Retourne 1
12          Pour i allant de 1 à taille :
13              res=res^x
14              Si : bin[i+1]=1 :
15                  $res=res*m
16          On retourne res
17
```

Annexe 2 : Pseudo code 2ème algorithme (puissances successives)

```
1       Variables :
2           m : Nombre a élevé à la puissance n
3           n : Puissance
4           x : base choisie
5           bin :  nombre n en base 2 mais la chaine est inversé
6           taille : taille de la chaine bin
7           res : 1 pour le moment mais contiendra le résultat
8           puissancecons : m
9
10      Algorithme :
11          Pour i allant de 0 à taille :
12              Si : bin[i]=1 :
13                  res = res*puissancecons
14              puissancecons = puissancecons^2
15
```

Annexe 3 : Pseudo code 3ème algorithme (parité des puissances)

```
1   Variables :
2       m : Nombre a élevé à la puissance n
3       n : Puissance
4           modulo : modulo choisis
5       Algorithme :
6           Si : n=1 :
7               On retourne m
8           Si : n est divisible par 2 :
9               On retourne algorithme 3( m^2 , n/2 )
10          Sinon :
11              On retourne algorithme 3( m^2 , n-1/2 )
```

Annexe 4 : Programme non modulaire :

```php
function Algo1($m,$n){
    if(  preg_match("#^[0-9]+$#", $m) and preg_match("#^[0-9]+$#", $n)){
        $bin = decbin($n);
        $x=2;
        $taille = strlen($bin);
        $res=$m;
        if($n==0) return 1;

        for($i=1;$i<$taille;$i++){
            $res=gmp_pow($res,$x);
            if($bin[$i]==1){
                $res=gmp_mul($res,$m);
            }
        }

    }
    else echo "Saisie Incorrecte";
    return $res;
}
function Algo2($m,$n){
    if(  preg_match("#^[0-9]+$#", $m) and preg_match("#^[0-9]+$#", $n)){
            $bin = decbin($n);
            $bin = strrev($bin);
            $taille = strlen($bin);
            $res=1;
            $puissancecons=$m;
            for($i=0;$i<$taille;$i++){
                if($bin[$i]==1){
                    $res=gmp_mul($res,$puissancecons);
                }
                $puissancecons=gmp_pow($puissancecons,2);
            }
    }
    else
        echo "Saisie incorrecte";
    return $res;
}
function Algo3($m,$n){
    if(  preg_match("#^[0-9]+$#", $m) and preg_match("#^[0-9]+$#", $n)){
        if ($n == 1) {
            return $m;}
        else if ($n%2 == 0){
            return Algo3(gmp_pow($m,2),$n/2);
        }
        else {
            return $m*Algo3(gmp_pow($m,2),($n-1)/2);
        }
    }
}
```

Annexe 5 : Programme modulaire :

```php
function Algo1Mod($m,$n,$modulo){
    if(  preg_match("#^[0-9]+$#", $m) and preg_match("#^[0-9]+$#", $n)){
        $bin = decbin($n);
        $x=2;
        $taille = strlen($bin);
        if($n==0) return 1;
        $res=$m;

        for($i=1;$i<$taille;$i++){
            $res=gmp_mod(gmp_pow($res,$x),intval($modulo));
            if($bin[$i]==1){
                $res=gmp_mod(gmp_mul($res,$m),intval($modulo));
            }
        }

    }
    else echo "Saisie incorrecte";
    return $res;

function Algo2Mod($m,$n,$modulo){
    if(  preg_match("#^[0-9]+$#", $m) and preg_match("#^[0-9]+$#", $n) and preg_match('#^[0-9]+$#', $modulo) ){
        $bin = decbin($n);
        $bin = strrev($bin);
        $taille = strlen($bin);
        $res=1;
        $puissancecons=$m;
        for($i=0;$i<$taille;$i++){
            if($bin[$i]==1){
                $res=gmp_mod(gmp_mul($res,$puissancecons),intval($modulo));
            }
            $puissancecons=gmp_mod(gmp_pow($puissancecons,2),intval($modulo));
        }
    }
    else
        echo "Saisie incorrecte";
    return $res;

function Algo3Mod($m,$n,$modulo){
    if(  preg_match("#^[0-9]+$#", $m) and preg_match("#^[0-9]+$#", $n)){
        if ($n == 1) {
            return gmp_mod($m,intval($modulo));}
        else if ($n%2 == 0){
            return gmp_mod(Algo3(gmp_pow($m,2),$n/2),intval($modulo));
        }
        else {
            return gmp_mod($m*Algo3(gmp_pow($m,2),($n-1)/2),intval($modulo));
        }
    }
}
```

Annexe 6 : Code HTML :

```html
<h2 class="titreDansLaFonctions">Exponentiation modulaire rapide</h2>
<form action="ExpoRapide.php" method="post" >
    <p>
        Entier : <input type="text" name="m"><br>
        Puissance : <input type="text" name="n"><br>
        Modulo : <input type="text" name="modulo"><br>
        <input type='submit' class='boutton' value="Calculer">
    </p>
</form>
```

Bibliographie

Site web consultés :

https://hal.inria.fr/inria-00540485v2/document
http://www.maths-algo.fr/index.php/informatique/problemes-d-informatique/arithmetique/exponentiation-rapide
http://php.net/manual/fr/book.gmp.php

Cours sur la cryptographie de David Hébert.

Cours d'algorithmique avancée de Frédérique Bassino.