Exponentiation rapide

1 Préambule : parcours gauche-droite ou droite-gauche ?

Exercice 1. Parcours droite-gauche et gauche-droite sur le miroir d'un nombre (Moodle).
L'objectif de ce TD est de comparer l'efficacité de l'algorithme du miroir d'un nombre selon le sens de parcours de ses chiffres. Récupérez vos sources et répondez aux questions suivantes.

1) Supposons qu'il soit interdit de calculer le nombre de chiffres de l'entier n avant de l'inverser. Quel parcours est donc impossible à réaliser sans cette information ?

Correction. La méthode gauche-droite doit récupérer en premier lieu le chiffre de gauche. Ce dernier ne peut se récupérer qu'avec le nombre de chiffres. La méthode gauche-droite ne pouvait donc pas se faire sans cette information.

2) Rédigez l'algorithme droite-gauche du calcul du nombre miroir. Précisons que la méthode droite-gauche récupère en premier le chiffre des unités pour construire le nombre miroir.

Correction.

Algorithme 1 – `Nombre miroir droite-gauche`
Entrée. `Entier positif` n
Sortie. `Entier positif` m `miroir de` n

`Entier` $m \leftarrow 0$
`Tant que` $n > 0$ `faire`
  $m \leftarrow m \times 10 + (n \mod 10)$
  $n \leftarrow \lfloor n/10 \rfloor$
`Fin Tant que`
`Renvoyer` m

3) Rédigez un algorithme permettant de compter le nombre de chiffres d'un entier n.

Correction.

Algorithme 2 – `Nombre de chiffres (Nb_chiffres)`
Entrée. `Entier positif` n
Sortie. `Entier positif` m `nombre de chiffres de` n

`Entier` $m \leftarrow 0$
`Tant que` $n > 0$ `faire`
  $m \leftarrow m + 1$
  $n \leftarrow \lfloor n/10 \rfloor$
`Fin Tant que`
`Renvoyer` m

4) En utilisant l'algorithme précédent, rédigez un algorithme permettant d'inverser n.

Correction.

Algorithme 3 – Nombre miroir gauche-droite

Entrée. Entier positif n

Sortie. Entier positif m miroir de n

```
Entier m ← 0
Entier c ← Nb_chiffres(n)-1
Entier d ← 10^(c-1)
Entier p ← 1
Tant que d > 0 faire
    m ← m + (⌊n/d⌋ mod 10) × p
    d ← d/10
    p ← p × 10
Fin Tant que
Renvoyer m
```

5) Déroulez les deux algorithmes sur la valeur 1337 afin de vérifier l'exactitude de vos algorithmes.

Correction. À vous de vérifier !

6) Évaluez la complexité de ces deux algorithmes en fonction du nombre de chiffres de n. Le calcul a^b requiert $b-1$ opérations.

Correction. Soit c le nombre de chiffres de l'entier n Étudions la complexité de l'algorithme droite-gauche. La boucle principale dépend directement du nombre de chiffres de n. Le nombre d'opérations dans chaque tour de boucle est fixe : deux affectations, une multiplication, un modulo, une division entière : on a donc 5 opérations à réaliser dans la boucle plus le test de boucle. La complexité est donc linéaire en fonction du nombre de chiffres : $6c+1$ opérations, soit $O(c)$ ou $O(\log(n))$

Étudions maintenant la complexité de l'algorithme gauche-droite. Le calcul du nombre de chiffres de n a une complexité linéaire en fonction de c : il demande $1+4c$ opérations Le calcul de 10^{c-1} demande $c-2$ opérations La boucle principale réalise c tours avec 9 opérations plus le test de boucle. Le nombre total d'opérations est donc $1+4c+c-2+10c = 14c-1$. La complexité est là aussi linéaire en fonction du nombre de chiffres de n mais nécessite deux fois plus d'opérations.

7) Déduisez-en la stratégie la plus efficace pour inverser un entier.

Correction. La méthode droite-gauche est donc plus efficace car elle n'a pas besoin de calculer au préalable le nombre de chiffres. Cette opération est réalisée au court de l'algorithme, sans surcout.

2 Échanges de clés avec le protocole Diffie-Hellman

2.1 Définition du problème

Exercice 2. Dimensionnement.

Correction

1) Quelles sont les restrictions de taille d'entier sur g, a et b pour que les calculs soit corrects sur des entiers non signés de 32 bits (à savoir, éviter un dépassement d'entier) ?

Correction. On veut qu'aucune étape du calcul ne dépasse pas la taille d'un entier. Autrement dit, le résultat d'un produit doit être strictement inférieur à 2^{32}. L'exposant n'ayant pas de limite autre que celle du stockage, ; on doit avoir $e < 2^{32}$. Le premier produit réalisé est $g \times g \mod n$. On veut donc $g^2 < 2^{32}$, soit $g < \sqrt{2^{32}} = 2^{16}$.

2) Dans ces conditions, quelle est la valeur maximale possible pour n ?

Correction. Posons $r = g^2 \mod n$. Si on souhaite calculer g^e, le calcul suivant sera $r \times g \mod n$. De la même façon que précédemment, on doit donc avoir $r < 2^{16}$. Cela implique directement $n < 2^{16} = 65536$.

2.2 État de l'art

Exercice 3. Exponentiation naïve (Moodle).

1) Rédigez l'algorithme naïf d'exponentiation de g^e.

Correction.

Algorithme 4 – Exponentiation naïve
Entrée. Entiers positifs g, e
Sortie. Entier positif g^e

```
Entier r ← 1
Tant que e > 0 faire
    r ← r × g
    e ← e − 1
Fin Tant que
Renvoyer r
```

2) Évaluez sa complexité en fonction de la valeur de l'exposant e.

Correction. La complexité est linéaire en fonction de e, soit $O(e)$.

Bien entendu, l'objectif de cet exercice reste l'implémentation de l'algorithme. Il est bien évidemment impossible de retourner le résultat de g^e sans l'opération modulo si g et e sont des entiers trop grands. Notre objectif est donc d'implémenter une fonction retournant un calcul modulo n.

3) Afin d'optimiser le code, où placer le calcul modulo ?

Correction. La meilleure optimisation serait de calculer le modulo seulement à la fin de l'exponentiation.

4) Rédigez la fonction `unsigned int ExpoNaiveIter (...)` dont les paramètres sont 3 entiers positifs g, e et n et retournant la valeur $g^e \mod n$.

5) Vérifiez votre code en calculant $1337^{73} \mod 4242$. [1]

6) Afin de rendre le calcul correct, où placer le calcul modulo ? Modifiez votre code en conséquence, s'il est faux, et vérifiez à nouveau votre code.

1. Cette valeur doit être égale à 3269.

Correction

Correction. Placer le modulo à la fin donne des erreurs de calculs à cause des multiplications qui entrainent un dépassement d'entier. Il faut donc réaliser les modulos à chaque multiplication.

2.3 Propriétés mathématiques de l'exponentiation

Exercice 4. Observations.

On considère 3 entiers g, a, et b.

1) Réécrivez le calcul g^{ab} d'une autre façon en utilisant les propriétés des puissances.

Correction. On a $g^{ab} = (g^a)^b$.

2) Déduisez-en une autre écriture pour 2^4, 2^8 et enfin pour 2^{16}.

Correction. On a $2^4 = (2^2)^2$, $2^8 = (2^4)^2 = ((2^2)^2)^2$ et $2^{16} = (2^8)^2 = (((2^2)^2)^2)^2$.

Exercice 5. Exposant particulier cherche exponentiation particulière (Moodle).

Nous nous concentrons sur le cas où l'exposant e est une puissance de 2.

1) Rédigez un algorithme permettant de calculer g^e lorsque $e = 2^k$, $k \in \mathbb{N}$. Cet algorithme doit être de complexité linéaire en fonction de k.

Correction.

Algorithme 5 – Exponentiation puissance de 2

Entrée. Entiers positifs g, $e = 2^k \mid k \in \mathbb{N}^*$

Sortie. Entier positif g^e

```
Tant que e > 1 faire
    g ← g × g
    e ← e/2
Fin Tant que
Renvoyer g
```

2) Évaluez sa complexité en fonction de la valeur de l'exposant e.

Correction. A chaque tour de boucle, l'exposant est divisé par deux. Cela équivaut à enlever un bit à e. La complexité de cet algorithme est linéaire en fonction du nombre de bits de e, Soit $O(\log(e))$.

3) Rédigez la fonction `unsigned int ExpoPuiss2(...)` dont les paramètres sont 3 entiers positifs g, e et n et retournant la valeur $g^e \mod n$ lorsque $e = 2^k \mid k \in \mathbb{N}$.

2.4 Exponentiation tabulaire

Exercice 6. Observations.

Utilisons les résultats obtenus dans l'exercice précédent afin de simplifier les calculs de l'exponentiation.

1) Réécrivez le calcul g^{a+b} d'une autre façon.

Correction. On a $^{a+b} = g^a g^b$.

2) Ré-écrivez les puissances g^3, g^5 et g^{11} sous la forme d'un produit de puissances g^{2^k}.

Correction. On a $g^3 = g^2 g$, $g^5 = g^4 g$ et $g^{11} = g^8 g^2 g$.

Correction

Exercice 7. Un peu de calcul.

L'objectif de cet exercice est de calculer $4^{22} \mod 7$ à l'aide de l'exponentiation tabulaire.

1) Quelle est la taille du tableau qui stockera les puissances de 4 ?

Correction. Le tableau doit stocker toutes les valeurs de la forme $4^{(2^k)}$, avec $2^k \leq 22$. Il comprend donc 4 cases

2) Remplissez ce tableau contenant les puissances successives $4^{2^k} \mod 7$.

Correction.

e	2^1	2^2	2^3	2^4
$4^e \mod 7$	2	4	2	4

3) Décomposez 22 sous sa forme binaire.

Correction. On trouve $22 = 16 + 4 + 2 = \overline{10110}^{(2)}$.

4) À l'aide de l'algorithme précédent, calculez le résultat final.

Correction. On a $4^{22} \equiv 4^{16} \times 4^4 \times 4^2 \equiv 4 \times 4 \times 2 \equiv 4 \mod 7$

Exercice 8. Implémentation (Moodle).

L'objectif de cet exercice est d'implémenter l'algorithme d'exponentiation tabulaire. Vous devez donc créer la fonction `unsigned int ExpoTabulaire(...)` dont les paramètres sont 3 entiers positifs g, e et n et retournant la valeur $g^e \mod n$. Les questions ci-dessous vont vous aider dans cette tâche.

1) Rédigez la fonction `int NombreBits (unsigned int n)` qui retourne le nombre de bits formant l'entier `n`. Vous pouvez entre autre utiliser les opérations de décalage binaire pour vous aider.
2) Rédigez la fonction `DecompoBinaire` prenant en entrée un entier positif `n` et un tableau `T` de 32 cases de types `char`. Cette fonction remplit le tableau `T` passé en paramètres. Ce tableau doit représenter la décomposition binaire de l'entier `n` passé en paramètre. Il contient uniquement des valeurs à 0 ou à 1, et non le caractère associé. La fonction doit renvoyer le nombre de bits de l'entier n.
3) Le plus dur est fait ! Complétez la fonction `ExpoTabulaire()` en suivant l'algorithme et en utilisant la fonction précédemment codée.
4) Vérifiez votre code en évaluant les exponentiations modulaires $4^{22} \mod 7$ et $1337^{73} \mod 4242$ précédemment calculés.

3 Exponentiation rapide

Exercice 9. Premier algorithme.

En reprenant l'algorithme de l'exponentiation tabulaire et l'exemple précédent, répondez aux questions suivantes.

1) Déroulez l'exemple précédent sur le calcul de g^{38}. Pour cela, réalisez la décomposition binaire de 38 à la main.

Correction

Correction.

Variable	Valeurs					
`cpt`	0	1	2	3	4	5
`tmp`	g	g^2	g^4	g^8	g^{16}	g^{32}
`b[cpt]`	0	1	1	0	0	1
`res`	1	g^2	g^6	g^4	g^6	g^{38}

2) Déroulez l'exemple précédent sur le calcul de $4^{22} \mod 7$.

Correction. Tous les calculs sont réalisé modulo 7.

Variable	Valeurs				
`cpt`	0	1	2	3	4
`tmp`	4	2	4	2	4
`b[cpt]`	0	1	1	0	1
`res`	1	2	1	1	4

3) Déduisez-en l'algorithme d'exponentiation rapide utilisant la décomposition binaire de l'exposant.

Exercice 10. Programmation (Moodle).
Il est temps de programmer l'algorithme final d'exponentiation rapide modulaire. Utilisez l'algorithme donné dans le cours et n'oubliez pas de réaliser vos calculs modulo `n`.

4 Synthèse

5 Pour aller plus loin

5.1 Nombre de palindromes

Exercice 11. Nombre de palindromes.

1) Listez les entiers $n < 1000$ tels que $M_{10}(n) = M_2(n)$? Par exemple, $M_{10}(92) = 29$ et $M_2(92) = M_2(\overline{1011100}^{(2)}) = \overline{11101}^{(2)} = 29 = M_{10}(92)$.
2) Combien existe-t-il de nombres palindromes en base 10 comme en base 2 inférieurs à 1000 ?
3) Mêmes questions pour les entiers inférieurs à $2^{31} - 1$

5.2 Exponentiation modulaire, version arithmétique

Correction