

Exercices

Exercice 1.1

Écrire six algorithmes différents pour déterminer si un nombre entier est premier ou composé. Évaluer la complexité pour chacun des algorithmes proposés.

Exercice 1.2

Considérer 7 algorithmes A0, A1, ..., A6 conçus pour résoudre un même problème de taille n . La complexité en temps de chacun de ces algorithmes est exprimée dans la table ci-dessous.

Algorithme	Complexité temporelle	Temps		
		$n = 10$	$n = 100$	$n = 1000$
A0	$O(1)$			
A1	$O(\log_2 n)$			
A2	$O(\sqrt{n})$			
A3	$O(n)$			
A4	$O(n^2)$			
A5	$O(n^3)$			
A6	$O(2^n)$			

- 1) Calculer les ordres de grandeurs suivantes en secondes :
 - a. 1 heure
 - b. 1 jour
 - c. 1 semaine
 - d. 1 mois
 - e. 1 année
 - f. 1 siècle
 - g. 1 millénaire
- 2) En supposant qu'une unité de taille s'exécute en une milliseconde, calculer le temps nécessaire pour traiter des tailles respectivement égales à 10, 100 et 1000.

- 3) Répéter la question 2 avec une unité de temps égale à une microseconde
- 4) Que peut-on en conclure ?

Exercice 1.3

Considérer le théorème suivant sur le pgcd (plus grand commun diviseur) :

$$\text{Pgcd}(a,b) = \text{pgcd}(b, a \bmod b)$$

- 1) Utiliser le théorème pour écrire un algorithme itératif du calcul du pgcd de deux entiers a et b.
- 2) Calculer la complexité en temps de l'algorithme.
- 3) Écrire la version récursive de l'algorithme trouvé en 1) et calculer sa complexité.
- 4) Écrire un algorithme pour calculer le ppcm (plus petit multiple commun) de deux nombres a et b et calculer sa complexité.

Exercice 1.4

Considérer l'alphabet $\{0,1\}$, un palindrome est un mot qui se présente sous le format ww^{-1} , où w est un mot quelconque de l'alphabet et w^{-1} est l'image miroir de w.

- 1) Écrire un algorithme qui reconnaît des palindromes et calculer sa complexité.
- 2) Écrire un programme assembleur pour reconnaître des palindromes et calculer sa complexité.
- 3) Concevoir une machine de Turing pour reconnaître le langage des palindromes. Le résultat doit être égal à 1 si le mot en entrée est un palindrome, 0 sinon. Calculer sa complexité.

Exercice 1.5

Considérer l'alphabet $\{0,1\}$, et le langage

$$L = \{(0/1)^*, \text{ nombre de } 0 = \text{nombre de } 1\}.$$

- 1) Écrire un algorithme qui reconnaît L et calculer sa complexité.
- 2) Écrire un programme assembleur qui reconnaît L et calculer sa complexité.
- 3) Concevoir une machine de Turing pour reconnaître les mots de L . Le résultat doit être égal à 1 si le mot en entrée est correct, 0 sinon. Calculer sa complexité.

Exercice 1.6.(Machine de Turing pour additionner 2 nombres)

Dans cet exercice, il s'agit de concevoir une machine de Turing pour additionner 2 nombres binaires. Supposer que les deux nombres ont la même longueur égale à n .

- 1) Illustrer votre démarche sur l'exemple suivant :
 $1101 + 1001$
Montrer toutes les étapes du déroulement de la machine.
- 2) Développer formellement la machine de Turing proposée pour additionner deux nombres binaires de longueur égale à n . Définir clairement l'alphabet, le nombre de rubans ainsi que la fonction de transition de votre machine. Pour cette dernière, expliquer le rôle de chacun des états utilisés.
- 3) Calculer la complexité spatiale de votre machine.
- 4) Calculer la complexité temporelle de votre machine.
- 5) Écrire un programme assembleur pour additionner 2 nombres binaires et calculer sa complexité.

Exercice 1.7.(Liste des nombres premiers inférieurs à n)

Considérer un tableau constitué des n nombres entiers consécutifs suivants : 1, 2, 3, ..., n . Une méthode de détermination des nombres premiers inférieurs à n , consiste à faire le traitement suivant :

- considérer le nombre 2 qui est premier puis éliminer tous les nombres multiples de 2 en les remplaçant par 0 car ils ne sont pas premiers,

- considérer ensuite le nombre 3 qui suit 2 et qui n'a pas été éliminé et supprimer tous les multiples de 3 en les remplaçant par 0
- ...

Réitérer ce processus en continuant avec le nombre suivant non mis à 0.

- 1) Illustrer chaque itération du procédé de calcul des nombres premiers décrit ci-dessus pour $n = 10$.
- 2) Écrire 3 algorithmes de détermination des nombres premiers inférieurs à n selon la méthode décrite ci-dessus.
- 3) Calculer la complexité de chacun des algorithmes proposés.
- 4) Dresser un tableau récapitulatif pour analyser les 3 algorithmes. Prévoir 3 colonnes : une pour l'appellation de l'algorithme, une autre pour sa complexité théorique et enfin une dernière pour calculer le nombre d'itérations maximum effectuées pendant l'exécution pour $n = 10000$.
- 5) Écrire un des trois algorithmes en assembleur à l'aide du jeu d'instructions donné en section 4.
- 6) Calculer la complexité du programme assembleur

Exercice 1.8.(Machine de Turing pour la soustraction)

Il s'agit de concevoir une machine de Turing pour la soustraction de deux nombres binaires. Supposer que les deux nombres ont la même longueur égale à n .

- 1) Illustrer votre démarche sur l'exemple suivant :
 $1101 - 1001$
 Montrer toutes les étapes du déroulement de la machine.
- 2) Développer formellement une machine de Turing pour la soustraction de deux nombres binaires de longueur égale à n . Définir clairement l'alphabet, le nombre de rubans ainsi que la fonction de transition de votre machine. Pour cette dernière expliquer le rôle de chaque état utilisé.
- 3) Calculer la complexité spatiale de votre machine.
- 4) Calculer la complexité temporelle de votre machine.

Exercice 1.9

On considère un polynôme $P(x)$ d'une variable réelle x , de degré n (n entier positif ou nul) et de paramètres réels a_i , $i = 0, \dots, n$:

$$P(x) = \sum_{i=0}^n a_i x^i = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$$

Supposer que les paramètres a_i , $i = 0, \dots, n$ sont stockés dans un tableau appelé T de $[0..n]$ de réels et que les valeurs de x et de n sont des constantes.

1. Écrire deux algorithmes pour évaluer le polynôme $P(x)$, un avec une complexité spatiale en $O(1)$ et un autre avec une complexité spatiale en $O(n)$. Calculer les complexités temporelles respectives.

Le polynôme P peut également se présenter sous la forme équivalente suivante:

$$P(x) = a_0 + x (a_1 + x (a_2 + \dots + x (a_{n-1} + a_n x) \dots))$$

2. Écrire une procédure itérative pour évaluer le polynôme en s'appuyant sur cette formulation. Calculer sa complexité temporelle puis sa complexité spatiale.
3. Écrire une version récursive correspondante et calculer ses complexités temporelle et spatiale.
4. Proposer une procédure pour additionner deux polynômes $P1$ et $P2$. Calculer sa complexité temporelle puis sa complexité spatiale.
5. Proposer une procédure pour multiplier deux polynômes $P1$ et $P2$ entre eux. Calculer sa complexité temporelle puis sa complexité spatiale.

Exercice 1.10 (Machine de Turing pour le calcul de n^2)

Il s'agit de concevoir une machine de Turing capable d'imprimer une séquence de 0 de longueur n^2 lorsque sur le ruban 1 une séquence de 0 de longueur n se présente.

- 1) Illustrer votre démarche sur la séquence 000. Montrer toutes les étapes du déroulement de la machine.

- 2) Développer formellement la machine de Turing en question.
Définir clairement l'alphabet, le nombre de rubans, l'ensemble des états ainsi que la fonction de transition de votre machine.
Pour cette dernière, expliquer le rôle de chacun des états utilisés.
- 3) Calculer la complexité spatiale de votre machine.
- 4) Calculer la complexité temporelle de votre machine.

Exercice 1.11

Considérer le problème de la détermination d'un mot miroir appelé PDMM et décrit comme suit :

Instance : un alphabet $A=\{0,1\}$ et un mot $w \in A^*$

Question : déterminer le mot w^{-1} qui soit le miroir de w i.e, lorsque w^{-1} est lu de droite à gauche, on retrouve le mot w .

- 1) Concevoir une machine de Turing à deux rubans permettant de calculer le mot miroir d'un mot. Calculer sa complexité.

Corrigé des exercices

Exercice 1.1

L'algorithme le plus simple que nous dénotons A1 découle directement de la définition d'un nombre premier. Rappelons qu'un nombre premier n est un nombre entier qui n'est divisible que par 1 et par lui-même. L'algorithme va donc consister en une boucle dans laquelle on va tester si le nombre n est divisible par $2, 3, \dots, n-1$.

Algorithme A1 :

début

premier := vrai ;
i := 2 ;
tant que ($i \leq n-1$) et premier faire
 si ($n \bmod i = 0$) alors premier := faux sinon $i := i+1$;

fin.

Le pire cas qui nécessite le plus long temps, correspond au cas où n est premier car c'est dans ce cas que la boucle s'exécute avec un nombre maximum d'itérations. Dans ce cas ce nombre est égal à $n-2$. La complexité est donc $O(n)$.

Nous savons que pour améliorer l'algorithme, il est judicieux d'arrêter la boucle à $n/2$ car si n est divisible par 2, il est aussi divisible par $n/2$ et s'il est divisible par 3, il est aussi divisible par $n/3$. De manière générale, si n est divisible par i pour $i = 1 \dots \lfloor n/2 \rfloor$ où $\lfloor n/2 \rfloor$ dénote la partie entière de $n/2$, il est aussi divisible par n/i . Il n'est donc pas nécessaire de vérifier qu'il est divisible par un nombre supérieur à $n/2$. Le deuxième algorithme est donc :

Algorithme A2 :

début

premier := vrai ;
i := 2 ;
tant que($(i \leq \lfloor n/2 \rfloor)$ et premier)faire
 si ($n \bmod i = 0$) alors premier := faux sinon $i := i+1$;

fin

Le cas le plus défavorable qui nécessite le plus long temps correspond toujours au cas où n est premier et dans ce cas le nombre d'itérations est égal à $\lfloor n/2 \rfloor - 1$. La complexité est donc en $O(n)$.

Une autre amélioration possible consiste à tester si n est impair et dans ce cas dans la boucle, il ne faut tester la divisibilité de n que par les nombres impairs. L'algorithme A3 est donc comme suit :

Algorithme A3 :

début

premier := vrai ;

si((n <> 2) et (n mod 2 = 0)) alors premier := faux

sinon si (n <> 2) alors

début

i := 3 ;

tant que((i <= n-2) et premier) faire

si (n mod i = 0) alors premier := faux sinon i := i+2 ;

fin

fin

Le pire cas correspond au cas où n est premier et dans ce cas le nombre maximum d'itérations de la boucle est égal à $\lfloor n/2 \rfloor - 2$, la complexité est en $O(n)$.

L'algorithme A4 peut être obtenu en hybridant A2 et A3 et on obtient:

Algorithme A4 :

début

premier := vrai ;

si(n <> 2) et (n mod 2 = 0) alors premier := faux

sinon si (n<> 2) alors

début

i := 3 ;

tant que (i <= [n/2]) et premier faire

si (n mod i = 0) alors premier := faux sinon i := i+2 ;

fin

fin.

Le nombre d'itérations de la boucle pour un nombre premier est égal à la moitié du nombre d'itérations de A3, il est égal à $\lfloor n/4 \rfloor - 1$. La complexité est donc en $O(n)$.

Une bonne amélioration de l'algorithme serait d'arrêter la boucle non pas à $\lfloor n/2 \rfloor$ mais à \sqrt{n} car en effet si n est divisible par x , il est aussi divisible par n/x . Et donc il serait judicieux de ne pas répéter le test de la divisibilité au-delà de $x = n/x$ et dans ce cas $n = x^2$ et $x = \sqrt{n}$. L'algorithme A5 s'écrit donc comme suit :

Algorithme A5 :

```
début premier := vrai ;
      i := 2 ;
      tant que((i <=  $\lfloor \sqrt{n} \rfloor$ ) et premier) faire
          si (n mod i = 0) alors premier := faux sinon i := i+1 ;
fin
```

Le nombre maximum d'itérations est égal à $\lfloor \sqrt{n} \rfloor - 1$, la complexité est en $O(\sqrt{n})$. Enfin, on peut concevoir un algorithme en hybridant A5 et A3, on obtient l'algorithme A6 suivant:

Algorithme A6 :

```
début premier = vrai ;
      si(n <> 2) et (n mod 2 = 0) alors premier := faux
      sinon si (n <> 2) alors
          début i := 3 ;
              tant que((i <=  $\lfloor \sqrt{n} \rfloor$ ) et premier) faire
                  si (n mod i = 0) alors premier := faux sinon i :=
                      i+2 ;
              fin
fin
```

Le nombre maximum d'itérations de la boucle est égal à $\frac{\lfloor \sqrt{n} \rfloor}{2} - 1$. La complexité est donc en $O(\sqrt{n})$.

Algorithme	complexité	temps		
		n=10	n= 100	n=1000
A0	$\ln n$	0,002s	0,005s	0,009s
A1	\sqrt{n}	0,003s	0,01s	0,031s
A2	n	0,01s	0,1s	1s
A3	n^2	0,1s	10s	16mn40s
A4	n^3	1s	16 mn 40s	11j13h46mn40s
A5	n^4	10s	1j3h46mn40s	31 ans 8 mois 15j 19h 3mn 28s
A6	2^n	1,02s	$3.2 \cdot 10^{16}$ millénaires	$3.2 \cdot 10^{286}$ millénaires

3) Le temps nécessaire au traitement des tailles de problème $n=10$, $n=100$ et $n=1000$ pour une unité de temps égale à une microseconde est montré dans le tableau suivant :

Algorithme	complexité	temps		
		n=10	n= 100	n=1000
A0	$\ln n$	$2,3 \cdot 10^{-6}$ s	$4,6 \cdot 10^{-6}$ s	$9,9 \cdot 10^{-6}$ s
A1	\sqrt{n}	$3,1 \cdot 10^{-6}$ s	10^{-5} s	$3,1 \cdot 10^{-5}$ s
A2	n	10^{-5} s	10^{-4} s	10^{-3} s
A3	n^2	10^{-4} s	0,01s	1s
A4	n^3	10^{-3} s	1s	16mn40s
A5	n^4	10^{-2} s	1mn40s	11j13h46mn40s
A6	2^n	10^{-3} s	$3.2 \cdot 10^{13}$ millénaires	$3.2 \cdot 10^{283}$ millénaires

4) Nous concluons que l'augmentation de la performance de la machine de calcul apporte les effets suivants :

- a. Améliore le temps de calcul pour des complexités polynomiales.
- b. n'atténue en rien les valeurs prohibitives des complexités exponentielles des grandes tailles de problème et ne peut donc pas constituer une solution pour contourner le problème de l'explosion combinatoire.

c. Pour les petites tailles, la fonction exponentielle est plus intéressante que certaines fonctions polynomiales ($n = 10$, A6 est plus rapide que A5).

Exercice 1.3

1) La fonction suivante calcule le pgcd selon la formule donnée :

*fonction pgcd ;
entrée : a, b : entier ;
sortie : le pgcd de a et b ;*

*var
x,y,q,r : entier;
début x := a;
y := b;
q := x / y;
r := x - q * y;
tant que (r <> 0) faire
 début x := y;
 y := r;
 q := x / y;
 r := x - q * y ;
 fin;
pgcd := y;
fin;*

2) Complexité

Le pire cas correspond à la situation où à chaque itération, le quotient de la division de x par y est égal à 1 et donc au nombre maximum d'itérations. Dans ce cas le reste de la division est égal à la différence entre x et y puisque :

$$r := x - q * y = x - 1 * y = x - y$$

Si x est supérieur à y , la séquence r, y et x fait partie de la suite de Fibonacci, car les deux dernières itérations correspondent respectivement à $r=1$ et à $r=0$, qui coïncident avec l'initialisation de la suite de Fibonacci. En résumé le cas le plus défavorable correspond au

calcul du pgcd de deux nombres consécutifs de la suite de Fibonacci F_{n-1} et F_{n-2} où

$$F_n = F_{n-1} + F_{n-2}, F_1 = 1 \text{ et } F_0 = 0.$$

$$F_n > 2 * F_{n-2} > 2 * 2 * F_{n-4} > \dots > 2^k$$

Où k est égal approximativement à la moitié de la longueur de la suite de Fibonacci qui correspond au nombre d'itérations de la boucle. Le nombre d'itérations m est égal donc à 2k. Si a est supérieur à b, a=F_n et par conséquent :

$$a > 2^k$$

$$\log_2 a > \log_2 2^k = k \log_2 2$$

$$k < (1 / \log_2 2) \log_2 a$$

d'où : $m < (2 / \log_2 2) \log_2 a$, La complexité est donc $O(\log_2 a)$.

3) Version récursive

fonction pgcd ;

entrée : a,b : entier ;

sortie : le pgcd de a et b ;

var

q, r : entier ;

début

si (b = 0) alors pgcd := a sinon début q := a div b; {division entière }

r := a - q * b;

pgcd := pgcd(b,r);

fin;

fin;

La complexité de la fonction pgcd est la même que celle calculée pour la 2^{ème} question.

Exercice 1.4

1) Algorithme

Algorithme palindrome ;

entrée : *T* un mot de *L* sous forme de $T[1..max]$: tableau d'entiers;

sortie : 'mot correct' si *T* est un palindrome, 'mot incorrect' sinon ;

```
var      i,j,c : entier ;
début   i:=1,
        j:= max ;
        tant que (T[i]=T[j]) et (i<j) faire
            début   i:= i+1;
                    j := j-1;
            fin;
            si (i>j) alors signaler('mot correct') sinon signaler ('mot incorrect');
fin;
```

Le nombre d'instructions exécutées pour un mot du langage est de :

- 2 instructions d'initialisation
- 2 instructions de la 2^{ème} boucle
- 1 instruction de fin de programme

Au total et dans le pire cas lorsque le mot est correct, on aura :
 $2+2*n/2+1 = 3+n$. La complexité est donc en $O(n)$.

2) Le programme assembleur est le suivant :

Boucle1:	MOV	T, R0
	INPUT	R1
	SUB	#2, R1
	JZ	L1
	ADD	#2, R1
	MOV	R1, (R0)
	ADD	#1, R0
	JMP	Boucle1
L1:	MOV	T, R1
Boucle2 :	SUB	R1, R0
	JGT	R0, Succès
	SUB	(R1), (R0)
	JZ	(R0), L2
	JMP	Échec
L2:	ADD	#1, R1

	SUB	#1, R0
	JMP	Boucle2
Échec:	OUTPUT	#0
	JMP	Fin
Succès:	OUTPUT	#1
Fin	STOP	
T:		

La boucle *Boucle1* consiste à copier la donnée en mémoire dans le tableau T. La deuxième boucle consiste à vérifier si le mot présenté en entrée est un palindrome ou pas. Le nombre d'instructions exécutées pour reconnaître un mot est égal à :

- 1 instruction d'initialisation
- 7 instructions de la 1^{ère} boucle
- 1 instruction d'initialisation de la 2^{ème} boucle
- 8 instructions de la 2^{ème} boucle
- 4 instructions de fin de programme

Au pire cas et si la longueur d'un mot est égale à n, ce nombre est égal au total à : $1 + 7*n + 1 + 8*n/2 + 4 = 6 + 7*n + 4*n = 6 + 11*n$

La complexité temporelle est donc en $O(n)$. L'espace occupé est égal à $2 + n$, pour l'utilisation des 2 registres R0, R1 et de la zone T de taille n. La complexité spatiale est donc en $O(n)$.

3) Machine de Turing :

Première solution : Machine à un seul ruban :

On suppose que le mot se trouve initialement sur le premier ruban. Nous aurons besoin de définir 8 états avec les rôles suivants :

- q0 : état initial, met à blanc le 1^{er} bit rencontré et transite vers q1 si le bit est égal à 1, à q2 s'il est égal à 0.
- q1 : sert à comparer le 1^{er} 1 de la chaîne avec le dernier 1.
- q2 : sert à comparer le 1^{er} 0 de la chaîne avec le dernier 0.
- q3 : sert à mettre à blanc le dernier 1 de la chaîne.
- q4 : sert à mettre à blanc le dernier 0 de la chaîne.
- q5 : sert à se déplacer vers la gauche et positionner la tête de lecture vers le 1^{er} bit de la chaîne restante.
- q6 signale un succès.
- q7 signale un échec.

La fonction de transition est alors la suivante :

- (q0,1) → (q1, b, R)
- (q0,0) → (q2, b, R)
- (q1,1) → (q1, 1, R)
- (q1,0) → (q1, 0, R)
- (q1,b) → (q3, b, L)
- (q3,1) → (q5, b, L)
- (q3,0) → (q7, b, L) échec
- (q3,b) → (q6, 1, L) succès
- (q7,1) → (q7, b, L)
- (q7,0) → (q7, b, L)
- (q7,b) → (q7, 0, S)
- (q5,1) → (q5, 1, L)
- (q5,0) → (q5, 0, L)
- (q5,b) → (q0, b, R)
- (q2,1) → (q2, 1, R)
- (q2,0) → (q2, 0, R)
- (q2,b) → (q4, b, L)
- (q4,0) → (q5, b, L)
- (q4,1) → (q7, b, L) échec
- (q4,b) → (q6, 1, L) succès

La taille du problème est celle du palindrome et est égale à n . Le pire cas correspond au cas où la donnée est un palindrome car c'est dans ce cas que la machine fait le maximum de transitions. Le nombre de transitions effectuées pour reconnaître un palindrome est de :
 $2(n+1)+2n+2(n-1)+\dots+2 =$

$$\sum_1^{n+1} 2i = 2 \sum_1^{n+1} i = 2(n+1)(n+2)/2 = (n^2+3n+2).$$

La complexité temporelle est donc en $O(n^2)$ et la complexité spatiale est en $O(n)$.

Deuxième solution : Machine à 2 rubans

Une autre machine peut être conçue avec 2 rubans. Dans un des rubans, on stocke le palindrome puis on le copie sur l'autre ruban et enfin on parcourt les deux rubans dans les sens opposés pour les comparer.

- (q0,0,b) → (q1, (0,S), (b,R))
- (q0,1,b) → (q1, (1,S), (b,R))
- (q1,0,b) → (q1, (0,R), (0,R))
- (q1,1,b) → (q1, (1,R), (1,R))
- (q2,b,0) → (q2, (b,S), (0,L))

$(q_2, b, l) \rightarrow (q_2, (b, S), (1, L))$	
$(q_2, b, b) \rightarrow (q_3, (b, L), (b, R))$	
$(q_3, 0, 0) \rightarrow (q_4, (0, S), (b, R))$	
$(q_4, 0, 0) \rightarrow (q_5, (0, S), (b, L))$	
$(q_5, 0, 0) \rightarrow (q_6, (1, S), (b, S))$	succès
$(q_5, 0, 1) \rightarrow (q_7, (0, S), (b, S))$	échec
$(q_4, 0, 1) \rightarrow (q_3, (b, L), (1, S))$	
$(q_4, 0, 0) \rightarrow (q_3, (b, L), (0, S))$	
$(q_3, 1, 1) \rightarrow (q_4, (1, S), (b, R))$	
$(q_4, 1, b) \rightarrow (q_5, (1, S), (b, L))$	
$(q_5, 1, 1) \rightarrow (q_6, (1, S), (b, S))$	succès
$(q_5, 1, 0) \rightarrow (q_7, (0, S), (b, S))$	échec
$(q_4, 1, 0) \rightarrow (q_3, (b, L), (0, S))$	
$(q_4, 1, 1) \rightarrow (q_3, (b, L), (1, S))$	

Le nombre de transitions effectuées est égal à :

1 pour insérer un blanc au début du deuxième ruban,

n pour copier le mot dans le deuxième ruban,

$n+1$ pour déplacer la tête de lecture du deuxième ruban vers l'extrémité gauche,

1 pour démarrer la comparaison,

$2n$ pour la comparaison.

Au total, il est égal à $1+n+(n+1)+1+2n = 4n + 3$. La complexité est donc $O(n)$.

Exercice 1.5

1)

Algorithme L ;

entrée : w un mot de L ;

sortie : 'mot correct' si w appartient à L, 'mot incorrect' sinon ;

var

écart, c : entier ;

début

lire(c);

écart := 0;

tant que ((c=0) ou (c=1)) faire

début

si (c=0) alors écart := écart + 1

sinon si (c=1) alors écart := écart - 1;

fin;

si (écart = 0) alors signaler ('mot correct')

sinon signaler ('mot incorrect')

fin.

Le nombre d'instructions exécutées pour reconnaître un mot du langage est égal à $2 + 3n + 1$. La complexité de l'algorithme est donc $O(n)$.

2) Le programme assembleur est le suivant :

Boucle :	MOV	#0, R0
	INPUT	R1
	JZ	R1, L0
	SUB	#1, R1
	JZ	R1, L1
L0:	JMP	Fin Boucle
	ADD	#1, R0
	JMP	Boucle
L1 :	SUB	#1, R0
	JMP	Boucle
Fin Boucle :	JZ	R0, succès
	OUTPUT	#0
	JMP	Fin
succès :	OUTPUT	#1
Fin :	STOP	

Le nombre d'instructions exécutées pour reconnaître un mot du langage est égal à :

$$1 + 4n + 4(\text{au maximum}) = 5 + 4n$$

La complexité est donc $O(n)$.

- 3) La machine de Turing que nous proposons utilise 3 rubans. En lisant le mot stocké sur le 1^{er} ruban de gauche à droite, la machine place les 0 rencontrés sur le 2^{ème} ruban et les 1 sur le 3^{ème}. En revenant en arrière, elle vérifie que le nombre de 0 du 2^{ème} ruban est égal au nombre de 1 du 3^{ème} ruban d'où la fonction de transition suivante :

$(q_0, 0, b, b) \rightarrow (q_1, (0, S), (b, R), (b, R))$
 $(q_0, 1, b, b) \rightarrow (q_1, (1, S), (b, R), (b, R))$
 $(q_0, b, b, b) \rightarrow (q_3, (1, S), (b, R), (b, R))$ succès: chaîne vide
 $(q_1, 0, b, b) \rightarrow (q_1, (b, R), (0, R), (b, S))$
 $(q_1, 1, b, b) \rightarrow (q_1, (b, R), (b, S), (1, R))$
 $(q_1, b, b, b) \rightarrow (q_2, (b, L), (b, L), (b, L))$
 $(q_2, b, 0, 1) \rightarrow (q_2, (b, L), (b, L), (b, L))$
 $(q_2, b, b, b) \rightarrow (q_3, (1, S), (b, S), (b, S))$ succès
 $(q_2, b, 0, b) \rightarrow (q_4, (0, S), (b, L), (b, L))$ échec
 $(q_2, b, b, 1) \rightarrow (q_4, (0, S), (b, L), (b, L))$ échec

La complexité temporelle est $O(n)$, de même est la complexité spatiale.

Exercice 1.6

- 1) La machine la plus simple qu'on peut concevoir possède 4 rubans :
- 1^{er} ruban contenant le premier nombre à additionner
 - 2^{ème} ruban contenant le deuxième nombre à additionner
 - 3^{ème} ruban contenant la retenue
 - 4^{ème} ruban contenant le résultat de l'addition

Illustration de l'addition de 1101+1001 :

1101b...
1001b...
bb...
bbbbbb

1101b...

1001b...

0bbbb ...

bbbbbb...

1101b...

1001b...

0bbbb

b0000b

110bb...

100bb...

1bbbb

b0000b

11bbb...

10bbb...

0bbbb

b0010b

1bbbb...

1bbbb...

0bbbb

b0110b

bbbbbb...

bbbbbb...

0bbbb

10110b

2) La fonction de transition de la machine de Turing est la suivante:

$(q_0, (0,0,b,b)) \xrightarrow{} (q_1, (0,0,0,b), (R,R,S,R))$

$(q_0, (0,1,b,b)) \xrightarrow{} (q_1, (0,1,0,b), (R,R,S,R))$

$(q_0, (1,0,b,b)) \xrightarrow{} (q_1, (1,0,0,b), (R,R,S,R))$

$(q_0, (1,1,b,b)) \xrightarrow{} (q_1, (1,1,0,b), (R,R,S,R))$

$(q_1, (0,0,0,b)) \xrightarrow{} (q_1, (0,0,0,0), (R,R,S,R))$

$(q_1, (0,1,0,b)) \xrightarrow{} (q_1, (0,1,0,0), (R,R,S,R))$

$(q_1, (1, 0, 0, b)) \rightarrow (q_1, (1, 0, 0, 0), (R, R, S, R))$
 $(q_1, (1, 1, 0, b)) \rightarrow (q_1, (1, 1, 0, 0), (R, R, S, R))$
 $(q_1, (b, b, 0, b)) \rightarrow (q_2, (b, b, 0, 0), (L, L, S, S))$

$(q_2, (0, 0, 0, 0)) \rightarrow (q_3, (b, b, 0, 0), (S, S, S, L))$
 $(q_2, (0, 0, 1, 0)) \rightarrow (q_3, (b, b, 0, 1), (S, S, S, L))$
 $(q_2, (0, 1, 0, 0)) \rightarrow (q_3, (b, b, 0, 1), (S, S, S, L))$
 $(q_2, (0, 1, 1, 0)) \rightarrow (q_3, (b, b, 1, 0), (S, S, S, L))$
 $(q_2, (1, 0, 0, 0)) \rightarrow (q_3, (b, b, 0, 1), (S, S, S, L))$
 $(q_2, (1, 0, 1, 0)) \rightarrow (q_3, (b, b, 1, 0), (S, S, S, L))$
 $(q_2, (1, 1, 0, 0)) \rightarrow (q_3, (b, b, 1, 0), (S, S, S, L))$
 $(q_2, (1, 1, 1, 0)) \rightarrow (q_3, (b, b, 1, 1), (S, S, S, L))$

$(q_3, (b, b, 0, 0)) \rightarrow (q_2, (b, b, 0, 0), (L, L, S, S))$
 $(q_3, (b, b, 1, 0)) \rightarrow (q_2, (b, b, 1, 0), (L, L, S, S))$
 $(q_3, (b, b, 0, b)) \rightarrow (q_4, (b, b, b, 0), (S, S, S, S))$
 $(q_3, (b, b, 1, b)) \rightarrow (q_4, (b, b, b, 1), (S, S, S, S))$

q_0 : état initial

q_4 : état d'arrêt

3) Complexité spatiale :

$n + n+1 + (n+1) = 3n+2$ cases utilisées

$O(n)$

4) Complexité temporelle

$1 + n + (2n-1) = 3n$ transitions effectuées

$O(n)$

Exercice 1.7

1) Les différentes itérations sont les suivantes :

1 2 3 4 5 6 7 8 9 10

1 2 3 0 5 0 7 0 9 0

1 2 3 0 5 0 7 0 0 0

2)

1^{er} algorithme

début

pour $i := 2$ à $n-2$ *faire*

pour $j := i+1$ à n *faire*

si $T[i] <> 0$ *alors*

si ($j \bmod i$) *alors* $T[j] := 0$;

pour $i := 1$ à n *faire*

si ($T[i] = 0$) *alors* afficher ($T[i]$);

fin;

2^{ème} algorithme

début

pour $i := 2$ à $n/2 - 1$ *faire*

pour $j := i+1$ à n *faire*

si $T[i] <> 0$ *alors*

si ($j \bmod i$) *alors* $T[j] := 0$;

pour $i := 1$ à n *faire*

si ($T[i] = 0$) *alors* afficher ($T[i]$);

fin;

3^{ème} algorithme

début

pour $i := 2$ à \sqrt{n} *faire*

pour $j := i+1$ à n *faire*

si $T[i] <> 0$ *alors*

si ($j \bmod i$) *alors* $T[j] := 0$;

pour $i := 1$ à n *faire*

si ($T[i] = 0$) *alors* afficher ($T[i]$);

fin;

3) Complexités :

Pour le calcul des complexités, les constantes sont données à titre indicatif, ce qui ne perturbe pas l'ordre de la complexité obtenu pour chaque algorithme.

1^{er} algorithme

Complexité des deux boucles imbriquées : La boucle externe s'exécute de 2 à n donc (n-1) fois. La boucle interne par contre s'exécute un nombre de fois qui dépend de i, qui varie de n-2 à 1. Le nombre d'itérations au total est donc égal à : $\sum_1^{n-2} i = \frac{(n-2)(n-1)}{2} = \frac{n^2 - 3n + 2}{2}$.

Le nombre d'itérations de la dernière boucle est égal à n. Au total le nombre d'itérations est égal à : $\frac{n^2 - 3n + 2}{2} + n = \frac{n^2 - n + 2}{2}$. La complexité est donc $O(n^2)$.

2^{ème} algorithme

Complexité des deux boucles imbriquées : La boucle externe s'exécute de 2 à $\frac{n}{2} - 1$ donc $\frac{n}{2} - 2$ fois. La boucle interne par contre s'exécute un nombre de fois qui dépend de i, qui varie de n-2 à $n - \left(\frac{n}{2} - 1\right) = \frac{n}{2} + 1$. Le nombre d'itérations au total est donc égal à :

$$\sum_{n/2+1}^{n-2} i = \sum_1^{n-2} i - \sum_1^{n/2} i = \frac{n^2 - 3n + 2}{2} - \frac{n^2 + 2n}{8} = \frac{3n^2 - 14n + 8}{8}.$$

Le nombre d'itérations de la dernière boucle est égal à n. Le nombre d'itérations au total est égal à :

$$\frac{3n^2 - 14n + 8}{8} + n = \frac{3n^2 - 6n + 8}{8}$$

La complexité est donc en $O(n^2)$.

3^{ème} algorithme

Complexité des deux boucles imbriquées : La boucle externe s'exécute de 2 à \sqrt{n} , donc $(\sqrt{n} - 1)$ fois. La boucle interne par contre s'exécute un nombre de fois qui dépend de i, qui varie de n-2 à $n - \sqrt{n}$. Le nombre d'itérations au total est donc égal à :

$$\sum_{n-\sqrt{n}}^{n-2} i = \sum_1^{n-2} i - \sum_1^{n-1-\sqrt{n}} i = \frac{n^2 - 3n + 2}{2} - \frac{n^2 - 2n\sqrt{n} + \sqrt{n}}{2} = \frac{2n\sqrt{n} - 3n - \sqrt{n} + 2}{2}.$$

Le nombre d'itérations de la dernière boucle est égal à n . Le nombre d'itérations au total est égal à :

$$\frac{2n\sqrt{n} - 3n - \sqrt{n} + 2}{2} + n = \frac{2n\sqrt{n} - n - \sqrt{n} + 2}{2}$$

La complexité est donc en $O(n\sqrt{n})$.

4)

Algorithme	Complexité	Nombre d'itérations	$n=10000$
1 ^{er} Algorithme	$O(n^2)$	$\frac{n^2 - n + 2}{2}$	49994999
2 ^{ème} Algorithme	$O(n^2)$	$\frac{3n^2 - 6n + 8}{8}$	37492501
3 ^{ème} Algorithme	$O(n\sqrt{n})$	$\frac{2n\sqrt{n} - n - \sqrt{n} + 2}{2}$	999451

5) Le programme assembleur :

```

MOV      #10, R4
MOV      #2, R0
L5:    SUB      R4, R0
        JZ       R0, Fin
        ADD      R4, R0
        SUB      #1, T[R0]
        JZ       T[R0], L0
        MOV      #0, T[R0]
        OUTPUT   R0
L0:    MOV      R0, R1
L3:    ADD      #1, R1
        MOV      T[R1], R2
        MOV      T[R1], R3
        DIV      R0, R2
        IDIV    R0, R3
        SUB      R2, R3
        JZ       R3, L1
        JMP      L2
L1:    MOV      #0, T[R1]
L2:    SUB      R4, R1

```

	JZ	R1, L4
	ADD	R4, R1
	JMP	L3
L4:	ADD	#1, R0
	JMP	L5
Fin :	STOP	
T :	0	
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	

- 6) La complexité du programme est identique à celle de l'algorithme de la question 2, car le programme est constitué de deux boucles imbriquées fonctionnant de la même manière que les boucles de l'algorithme. Elle est par conséquent en $O(n^2)$

Exercice 1.8

1) On utilise une machine de Turing à 4 rubans :

Le 1^{er} ruban contient le premier nombre

Le 2^è ruban contient le deuxième nombre

Le 3^è ruban sert à calculer le complément à 2 du 2^è nombre

Le 4^è ruban sert à stocker le résultat de la soustraction

L'idée est d'utiliser le complément à 2 pour effectuer la soustraction de 2 nombres binaires. Sur l'exemple donné :

$$1101 - 1001$$

Le complément à 2 du nombre 1001 est 0111, quand on l'additionne à 1101 on obtient

$$1101 + 0111 = 0100 \text{ avec une retenue de } 1$$

Le résultat est donc 0100

Si on doit faire

$$1001 - 1101$$

Le complément à 2 du nombre 1101 est égal à : 0011, ce qui donne :
 $1001+0011=1100$ sans retenue. Le résultat s'obtient en complémentant à 2 1100 et est donc égal à -0100

2) Machine de Turing

Alphabet = {0,1,2, b}

Etats = {q0,q1, q2, q3, q4, q5, q6, q7, q8, q9, q10}

Etats-finaux = {q6}

Les transitions sont données ci-dessous.

a) Initialement, la machine a la configuration suivante :

Le 1^{er} ruban contient le premier nombre

Le 2^è ruban contient le deuxième nombre

Le 3^è ruban contient des blancs

Le 4^è ruban contient des blancs

b) Se déplacer d'un cran vers la droite en inversant le premier bit du 2^è nombre et en marquant le début de l'opération en mettant le caractère 2 au niveau du 3^è ruban :

(q0, (0,0,b,b)) → (q1, (0,R), (1, R), (2, R), (b,R))

(q0, (0,1,b,b)) → (q1, (0,R), (0, R), (2, R), (b,R))

(q0, (1,0,b,b)) → (q1, (1,R), (1, R), (2, R), (b,R))

(q0, (1,1,b,b)) → (q1, (1,R), (0, R), (2, R), (b,R))

c) Avancer à droite en inversant le 2^è nombre :

(q1, (0,0,b,b)) → (q1, (0,R), (1, R), (0, R), (b,R))

(q1, (0,1,b,b)) → (q1, (0,R), (0, R), (0, R), (b,R))

(q1, (1,0,b,b)) → (q1, (1,R), (1, R), (0, R), (b,R))

(q1, (1,1,b,b)) → (q1, (1,R), (0, R), (0, R), (b,R))

(q1, (b,b,b,b)) → (q2, (b,S), (b, S), (b, L), (b,S))

d) Mettre 1 sur le 3^è ruban pour le complément à 2 :

(q2, (b,b,0,b)) → (q3, (b,L), (b, L), (1, S), (b,L))

e) Commencer l'addition avec le complément à 2, rester dans q³ si il n'y a pas de retenue et passer à q4 dans le cas contraire !

(q3, (0,0,0,b)) → (q3, (b,L), (b, L), (b, L), (0,L))

(q3, (0,0,1,b)) → (q3, (b,L), (b, L), (b, L), (1,L))

(q3, (0,1,0,b)) → (q3, (b,L), (b, L), (b, L), (1,L))

$(q3, (0,1,1,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (0,L))$
 $(q3, (1,0,0,b)) \rightarrow (q3, (b,L), (b, L), (b, L), (1,L))$
 $(q3, (1,0,1,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (0,L))$
 $(q3, (1,1,0,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (0,L))$
 $(q3, (1,1,1,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (1,L))$

- f) A la rencontre du caractère 2 sur le 3^e ruban en étant dans q3, on continue à additionner, on marque un temps d'arrêt, on passe à q5 s'il n'y a pas de retenue pour complémenter à 2 le résultat, à q6 s'il y a une retenue pour stopper :

$(q3, (0,0,2,b)) \rightarrow (q5, (b,S), (b, S), (b, S), (0,S))$
 $(q3, (0,1,2,b)) \rightarrow (q5, (b,S), (b, S), (b, S), (1,S))$
 $(q3, (1,0,2,b)) \rightarrow (q5, (b,S), (b, S), (b, S), (1,S))$
 $(q3, (1,1,2,b)) \rightarrow (q6, (b,S), (b, S), (b, S), (0,L))$

- g) On continue l'addition avec une retenue exprimée par l'état q4 mais uniquement avec 0 au niveau du 3^e ruban car on est à une position inférieure à n :

$(q4, (0,0,0,b)) \rightarrow (q3, (b,L), (b, L), (b, L), (1,L))$
 $(q4, (0,1,0,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (0,L))$
 $(q4, (1,0,0,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (0,L))$
 $(q4, (1,1,0,b)) \rightarrow (q4, (b,L), (b, L), (b, L), (1,L))$

- h) A la rencontre du caractère 2 sur le 3^e ruban en étant dans q4, on continue à additionner, on marque un temps d'arrêt, on passe à q5 s'il n'y a pas de retenue pour complémenter à 2 le résultat, à q6 s'il y a une retenue pour stopper :

$(q4, (0,0,2,b)) \rightarrow (q5, (b,S), (b, S), (b, S), (1,S))$
 $(q4, (0,1,2,b)) \rightarrow (q6, (b,S), (b, S), (b, S), (0,S))$
 $(q4, (1,0,2,b)) \rightarrow (q6, (b,S), (b, S), (b, S), (0,S))$
 $(q4, (1,1,2,b)) \rightarrow (q6, (b,S), (b, S), (b, S), (1,L))$

- i) Dans le cas où il n'y a pas de retenue, on commence à complémenter à 2 le résultat, on met 2 sur le 3^e ruban pour indiquer l'extrême gauche du nombre lors du retour pour additionner 1 :

$(q5, (b,b,b,0)) \rightarrow (q7, (b,S), (b, S), (2, R), (1,R))$
 $(q5, (b,b,b,1)) \rightarrow (q7, (b,S), (b, S), (2, R), (0,R))$

j) On continuer à inverser le résultat à mettre 0 au niveau du 3^e ruban :

$$(q7, (b,b,b,0)) \rightarrow (q7, (b,S), (b, S), (0, R), (1,R))$$

$$(q7, (b,b,b,1)) \rightarrow (q7, (b,S), (b, S), (0, R), (0,R))$$

$$(q7, (b,b,b,b)) \rightarrow (q8, (b,S), (b, S), (b, L), (b,S))$$

k) Mettre 1 sur le 3^e ruban pour poursuivre le calcul du complément à 2 du résultat :

$$(q8, (b,b,b,0)) \rightarrow (q9, (b,S), (b, S), (1, S), (b,L))$$

l) Ajouter 1 à l'inverse du résultat à partir de q9 sans retenue en entrée: passer à 9 s'il n'y a pas de retenue à 10 dans le cas contraire :

$$(q9, (b,b,0,0)) \rightarrow (q9, (b,S), (b, S), (b, L), (0,L))$$

$$(q9, (b,b,0,1)) \rightarrow (q9, (b,S), (b, S), (b, L), (1,L))$$

$$(q9, (b,b,1,0)) \rightarrow (q10, (b,S), (b, S), (b, L), (0,L))$$

$$(q9, (b,b,1,1)) \rightarrow (q10, (b,S), (b, S), (b, L), (1,L))$$

m) Ajouter 1 à l'inverse du résultat à partir de q10 avec retenue en entrée: passer à 9 s'il n'y a pas de retenue à 10 dans le cas contraire. Dans ce cas le 3^e ruban contient 0 :

$$(q10, (b,b,0,0)) \rightarrow (q9, (b,S), (b, S), (b, L), (1,L))$$

$$(q10, (b,b,1,0)) \rightarrow (q10, (b,S), (b, S), (b, L), (0,L))$$

n) A la rencontre du symbole 2, on continue l'addition et on s'arrête. Le résultat est un nombre négatif dans ce cas :

$$(q9, (b,b,2,0)) \rightarrow (q6, (b,S), (b, S), (b, S), (1,S))$$

$$(q9, (b,b,2,1)) \rightarrow (q6, (b,S), (b, S), (b, S), (0,S))$$

$$(q10, (b,b,2,0)) \rightarrow (q6, (b,S), (b, S), (b, S), (1,S))$$

3) Complexité spatiale

La machine utilise 4 rubans et $(n+1)$ cases par ruban au maximum, donc $4n+4$ au total. La complexité spatiale est $O(n)$.

4) Complexité temporelle

La complexité temporelle s'exprime en termes de nombre de transitions au maximum. Le pire cas correspond au cas où le premier nombre est inférieur au second et dans ce cas le résultat est un nombre

négatif. Dans ce cas-là, la machine passe par les étapes consécutives suivantes:

- se déplace de gauche à droite pour atteindre l'extrémité droite pour démarrer l'opération d'addition du premier nombre avec le complément à 2 du 2^e nombre. Cette étape nécessite **n transitions**.
- se déplace d'un cran vers la gauche pour mettre 1 sur le 3^e ruban. Cette étape nécessite **1 transition**.
- se déplace de droite à gauche pour effectuer l'addition. Cette étape nécessite **n transitions**.
- se déplace de gauche à droite en inversant le résultat qui se trouve sur le 4^e ruban. Cette étape nécessite **n transitions**.
- se déplace d'un cran vers la gauche pour mettre 1 sur le 3^e ruban. Cette étape nécessite **1 transition**.
- se déplace de droite à gauche pour effectuer l'addition. Cette étape nécessite **n transitions**.

Au total, la machine effectue un nombre de transitions égal à : $(n+1+n+n+1+n) = 4n + 2$. La complexité temporelle de la machine est donc $O(n)$.

Exercice 1.11

Considérer le problème de la détermination d'un mot miroir appelé PDMM et décrit comme suit :

Instance : un alphabet $A=\{0, 1\}$, un mot $w \in A^*$

Question : déterminer le mot w^{-1} qui soit le miroir de w i.e, lorsque w^{-1} est lu de droite à gauche, on retrouve le mot w .

- 1) Concevoir une machine de Turing à deux rubans permettant de calculer le mot miroir d'un mot. Calculer sa complexité.

Machine de Turing à deux rubans :

Entrée : w sur le premier ruban et la tête de lecture/écriture pointe sur le premier élément de w

Sortie : w^{-1} sur le premier ruban et la tête de lecture/écriture pointe sur le premier élément de w^{-1}

/* copie du mot sur le deuxième ruban
 $(q_0, 0, b) \rightarrow (q_1, (0, S), (b, R))$

$(q_0, 1, b) \rightarrow (q_1, (1, S), (b, R))$
 $(q_1, 0, b) \rightarrow (q_1, (b, R), (0, R))$
 $(q_1, 1, b) \rightarrow (q_1, (b, R), (1, R))$
 $(q_1, b, b) \rightarrow (q_2, (b, L), (b, L))$

/* inversement du mot
 $(q_2, b, 0) \rightarrow (q_2, (b, S), (0, L))$
 $(q_2, b, 1) \rightarrow (q_2, (b, S), (1, L))$
 $(q_2, b, b) \rightarrow (q_3, (b, S), (b, R))$
 $(q_3, b, 0) \rightarrow (q_3, (0, L), (b, R))$
 $(q_3, b, 1) \rightarrow (q_3, (1, L), (b, R))$
 $(q_3, b, b) \rightarrow (q_4, (b, R), (b, S))$ arrêt

Complexité:

La machine parcourt le mot qui se trouve sur le premier ruban et le recopie sur le deuxième ruban. Cette opération nécessite $n+1$ transitions

La machine ensuite parcourt les deux rubans dans des sens opposés pour inverser le mot et le placer sur le premier ruban. Cette opération nécessite également $n+1$ transitions. Au total, la machine procède à $2n+2$ transitions. La complexité est $O(n)$, où n est la longueur du mot.