

COMPOSITION d INF 411

en date du 16/11/2021

INF411

2500

1. Expressions régulières

Q1: expression régulière : $a \dots * \dots b$

Q2:

```
static String toString(SRE r) {  
    String exp = " ";  
    if (r == null) {  
        return exp;  
    }  
    else {  
        SRE courant = r;  
        while (courant != null) {  
            exp += courant.c;  
            if (courant.star) {  
                exp += " * ";  
            }  
            courant = courant.next;  
        }  
        return exp;  
    }  
}
```

NOTE

N°

1

3

NE PAS ÉCRIRE DANS CE CADRE

Q3:

```
static SRE ofString(String s) {  
    if (s.charAt(0) == null) {  
        return null;  
    }  
    else {  
        SRE r = new SRE(s.charAt(0), false, null);  
        SRE courant = r;  
        for (int i = 1; i < s.length(); i++) {  
            if (s.charAt(i).equals("*")) {  
                courant.star = true;  
            }  
            else {  
                SRE next = new SRE(s.charAt(i), false, null);  
                courant = courant.next;  
            }  
        }  
        return r;  
    }  
}
```

Q4: 1^{er} appel marches("aa", r)

→ fait appel à marches("aa", 0, r)

~~On est dans le cas r.c = s.charAt(i) et r.star = true
(lignes 10 puis 11)~~

NE PAS ÉCRIRE DANS CE CADRE

* on est dans le cas $r.star = true$.

on fait donc appel à $matches(s, 0, r.next)$

→ dès lors, on a $r.c = s.charAt(0)$ et $r.star = false$

on fait donc appel à $matches("aa", 1, r.next)$

avec $r.next = null$

l'appel renvoie alors $i == s.length$. Ici, c'est false.

⇒ On renvoie donc du "ij" initial

* on a $r.c = s.charAt(0)$ et $r.star = true$

→ appel à $matches("aa", 1, r)$

$r.star = true$.

→ appel à $matches("aa", 1, r.next)$ (*)

$r.c = s.charAt(1)$ et $r.star = false$

→ appel à $matches("aa", 2, null)$

$r = null$ donc renvoie $2 == s.length()$ ⇒ renvoie true
⇒ retour au test (*)

on a les deux conditions ⇒ $matches(s, r)$ renvoie true.

Q5: Quelle que soit la condition testée, elle renvoie toujours à un appel de $matches$ sur $r.next$ (sauf le test $r.star$, mais dans ce cas $r.next$ intervient à l'appel suivant). Comme r n'est pas infini, au bout d'un moment, on tombe sur $r.next = null$.

Pour $r = \text{null}$, l'algorithme renvoie forcément true ou false. On sort donc de la récursivité (de l'étape en question). Donc l'algorithme se termine, on ne peut pas faire appel à matches à l'infini.

Q6:

Q7: Une telle chaîne est $aabbcc\dots$ avec $\frac{N}{2}$ lettres différentes. Son expression régulière est $a^*b^*c^*\dots$

Dès lors, chaque appel à matches fait rentrer dans le deuxième test if, qui rappelle à nouveau matches. On a donc un temps de calcul de 2^N .

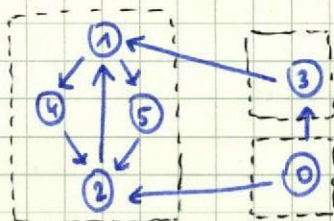
Q8:

Q9:

```
public int hashCode() {
```


2. Composantes fortement connexes

Q11:



Q12:

chaque sommet est
son propre représentant
au début
↑

On construit une structure unionfind de taille V .

on parcourt successivement chaque sommet.

Pour chaque sommet, on parcourt la liste de ses voisins:

→ on teste l'égalité des représentants ($\text{find}(i)$) du
sommet actuel et de chacun de ses voisins.

* si le représentant est le même, on ne fait rien

* sinon, on fait l'union des deux classes (celle
du sommet et de son voisin) grâce à la fonction
 $\text{union}(i, j)$

on renvoie le tableau link actualisé.

Complexité: 2 boucles for imbriquées

complexité de $\text{find}()$: $O(\log N)$
 $\text{union}()$: $O(\log N)$

donc la complexité est en $O(N \log(N))$

NOTE

N°
2
3

NE PAS ÉCRIRE DANS CE CADRE

Q13: appel à composants (Graph g)

→ num

--	--	--	--	--	--

→ $nc = 0$

→ visited

1	1	1	1	1	1
---	---	---	---	---	---

Boucle for:

1) $v = 0$. visited[v] = false

⇒ appel à dfs (visited, g , 0)

↳ appel à dfs (visited, g , 3)

↳ appel à dfs (visited, g , ⁰~~3~~)

↳ appel à dfs (visited, g , 4)

↳ appel à dfs (visited, g , 0)

↳ appel à dfs (visited, g , 1)

↳ appel à dfs (visited, g , 4)

⇒ appel à dfs (visited, g , 2)

↳ appel à dfs (visited, g , 5)

↳ appel à (visited, g , 2)

Finalement, le tableau renvoyé est

0	1	2	3	4	5
0	0	1	0	0	1

Q14: Par l'absurde. Supposons que CCG admette un cycle. Par définition d'un cycle, il y a alors deux composantes ^{distinctes} C_i et C_j fortement liées entre elles.

Cela implique que les deux sommets par lesquels les

NE PAS ÉCRIRE DANS CE CADRE

composantes sont reliées entre elles sont fortement liés entre eux. Donc tous les sommets de C_i et C_j sont fortement liés entre eux, ce qui est absurde car C_i et C_j sont distinctes.

Donc $CC(G)$ ne peut pas contenir de cycle.

Q15:

```
static Graph reverse(Graph g) {  
    Graph R = new Graph(g.V);  
    for (int i = 0; i < g.V; i++) {  
        for (int w : g.adj(i)) {  
            R.addEdge(w, i);  
        }  
    }  
    return R;  
}
```

Q16: Pour G_2^k , la liste renvoyée est 0, 1, 2, 4, 3, 5, 7, 6

Q17:

Q18: D'une part, $\text{postOrder}(\text{reverse}(g))$ donne 1, 2, 4, 5, 3, 0
 Déroulons l'algorithme :

$v=1$ appel à $\text{dfs}(\text{visited}, g, 1)$ (noté $\text{dfs}(1)$)

↳ visited

f	f	f	f	f	f
---	---	---	---	---	---

 num

0

↳ appel à $\text{dfs}(4)$

visited

f	t	f	f	t	f
---	---	---	---	---	---

 num

0	0
---	---

↳ appel à $\text{dfs}(2)$

vis

f	t	t	f	t	f
---	---	---	---	---	---

 num

0	0	0
---	---	---

↳ appel à $\text{dfs}(1)$ —

↳ appel à $\text{dfs}(5)$

visited

f	t	t	f	t	t
---	---	---	---	---	---

 num

0	0	0	0
---	---	---	---

$v=2$ —
 $v=4$ —
 $v=5$ —

$v=3$ appel à $\text{dfs}(3)$

visited

f	t	t	t	t	t
---	---	---	---	---	---

 num

0	0	1	0	0
---	---	---	---	---

↳ appel à $\text{dfs}(1)$ —



COMPOSITION d INF 411

en date du 16/11/2021

$n = 0$

appel à $\text{djs}(0)$

visited

t	t	t	t	t	t
---	---	---	---	---	---

mem

0	0	1	0	0
---	---	---	---	---

↳ appel à $\text{djs}(2)$ —

↳ appel à $\text{djs}(3)$ —

mem

2	0	0	1	0	0
---	---	---	---	---	---

on obtient donc

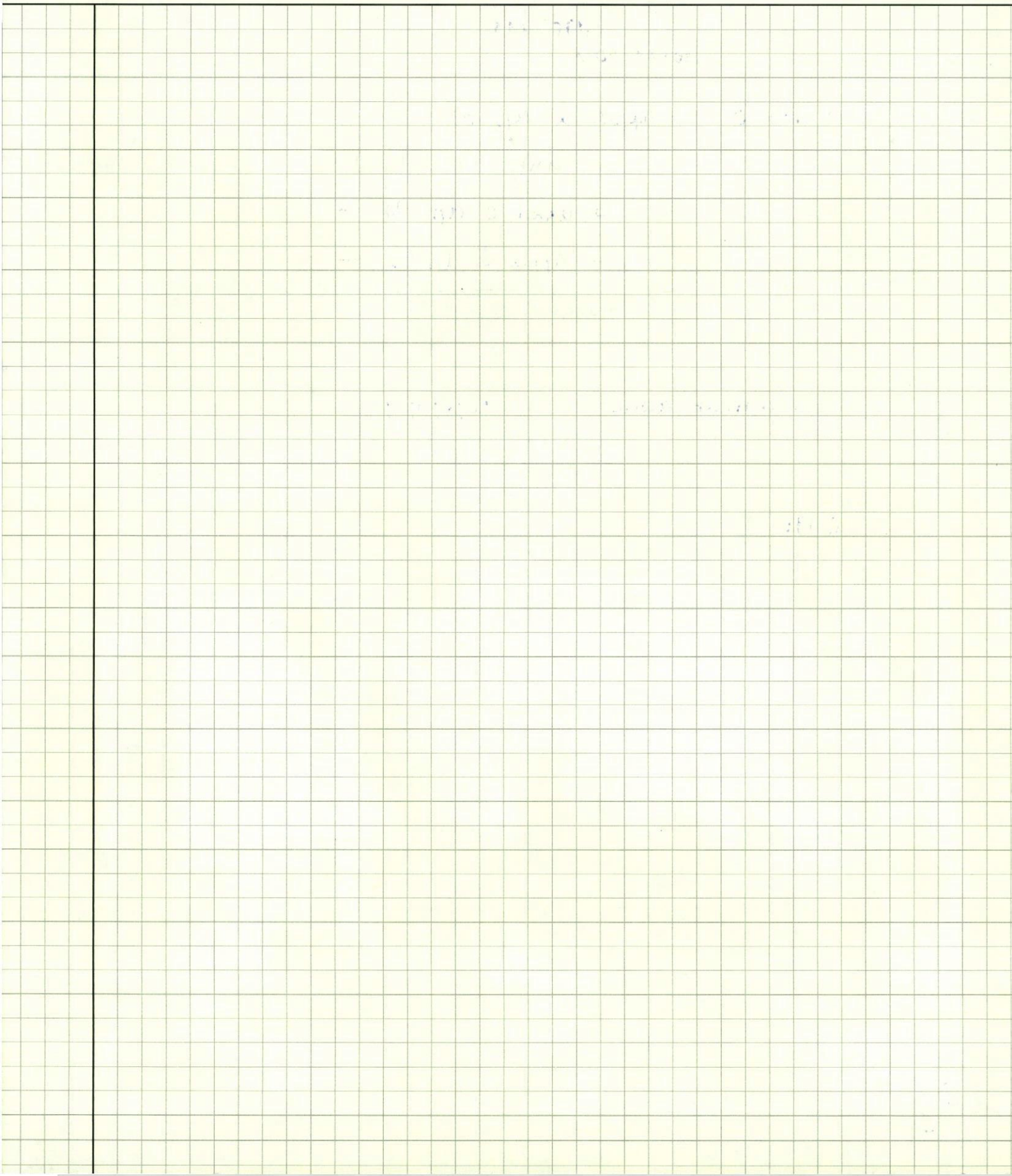
2;0;0;1;0;0

Q19:

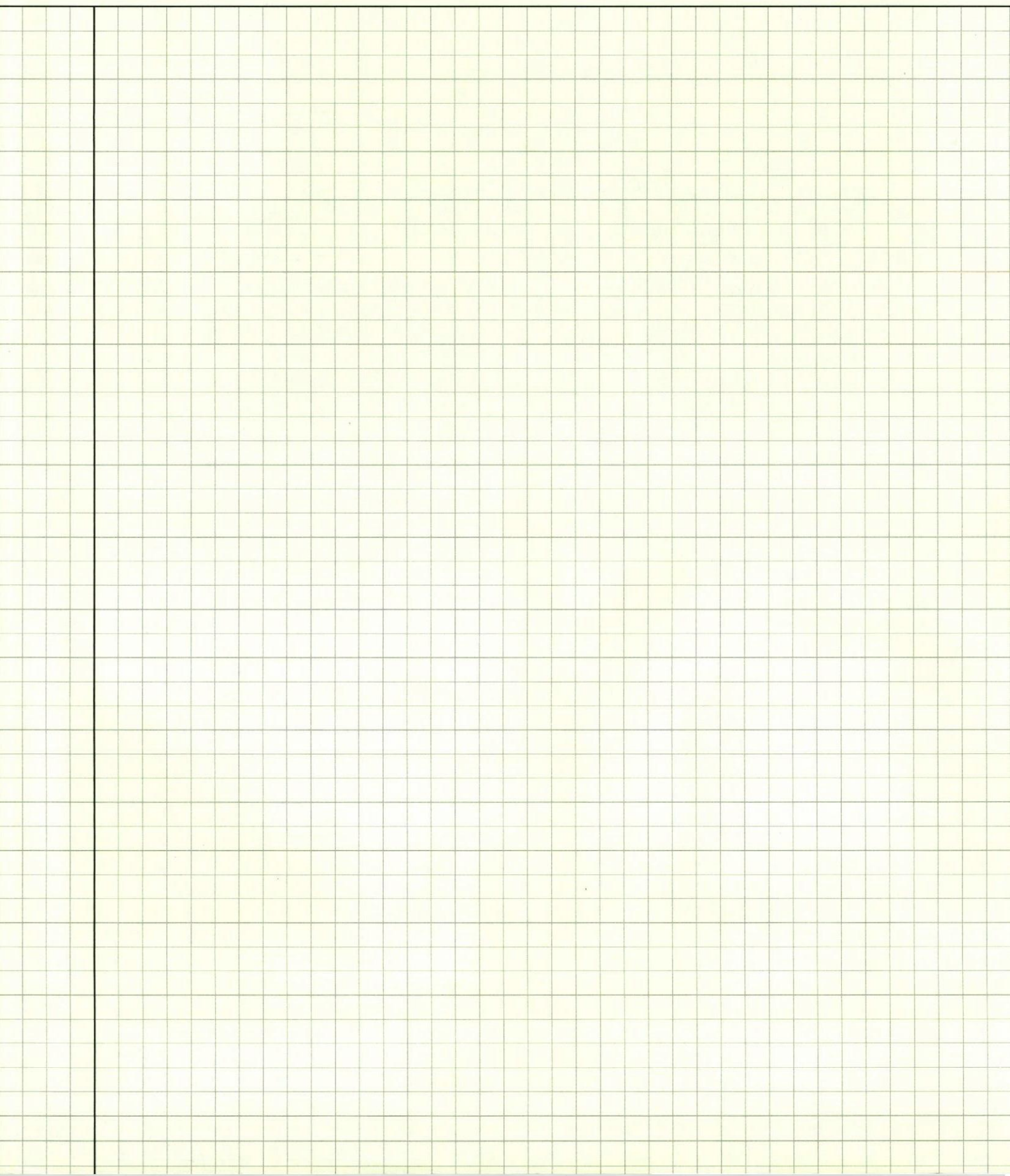
NOTE

N°
3/3

NE PAS ÉCRIRE DANS CE CADRE



NE PAS ÉCRIRE DANS CE CADRE



NE PAS ÉCRIRE DANS CE CADRE