

I Expressions régulières1. $a \dots * \dots b$

2.

```

static String toString (SRE r) {
    if (r.c.equals(null)) return ;
    if (r.star) {
        return (r.c + '*' + toString(r.next))
    }
    else { return (r.c + toString(r.next)) }
}

```

3. static SRE of String (String s) {

if (s.equals(null)) return null;

int n = s.length();

int i = 0;

if (s.length == 1 || !s.charAt(1).equals('*')) {

SRE r = new SRE(s.charAt(0), false, null);

i = 1;

}

else {

SRE r = new SRE(s.charAt(0), true, null);

i = 2;

while (i < n) {

if (i+1 == n || !s.charAt(i+1).equals('*')) {

r.next = new SRE(s.charAt(i), false, null);

i++;

}

↪

NOTE

N°

1

2

NE PAS ÉCRIRE DANS CE CADRE

```
else {  
    r.next = new SRE(s.charAt(i), true, null);  
    i += 2;  
}  
r = r.next;  
}  
return r;  
}
```

4. matches("aa", 0, r): avec $r.c = 'a'$, $r.star = true$ et $r.next =$

- on a $r.c == "a"$:

- $r.star$ est vrai:

on appelle donc matches("aa", 1, r)

- ~~matches("aa", 1, r):~~ " "

- on a

- On commence par appeler matches avec pour arguments "aa", 0 et $r.c = 'a'$, $r.star = true$ et $r.next = SRE('a', false, null)$

- On appelle ensuite matches(s, i, r.next) avec "aa", 0 et $r.c = 'a'$, $r.star = false$ et $r.next = null$.

- On appelle ensuite matches(s, i+1, r.next) avec "aa",
 $i = 1$ et $r.next = null$

Elle renvoie false

- On rappelle ensuite matches(s, i+1, r.next) (mêmes arguments)
Elle renvoie false

Elle renvoie false

NE PAS ÉCRIRE DANS CE CADRE

- On appelle alors `matches(s, i+1, r)` avec "aa", 1 et le même `r` qu'au début

On appelle `matches` avec "aa", $i=1$ et `r.c = a`,
`r.star = false`, `r.next = null`

On appelle `matches` avec "aa", $i=2$ et `r = null`.

Elle renvoie true

Elle renvoie true

Elle renvoie true

Elle renvoie true

5. Avant d'accéder à `charAt(i)`, on s'est assuré que celui-ci était bien dans les bornes ligne 8.

- Avant d'accéder à `r.star` ligne 6 on a vérifié que `r` n'était pas null p.4
| `r.c` ligne 10
| `r.next` ligne 14

8 Dans l'exemple de la question 5, on a fait deux appels à `matches` avec pour arguments "aa", $i=1$ et `r.next = null`.

```
9. public int hashCode() {  
    return this.i + this.r.c.hashCode();  
}  
  
public boolean equals(Object o) {  
    MemoKey m = (MemoKey) o;  
    return (this.i == m.i && this.r.equals(m.r));  
}
```


NE PAS ÉCRIRE DANS CE CADRE

2. Composantes fortement connexes

11. Les composantes fortement connexes sont les sommets: $\{1, 2, 4 \text{ et } 5\}$, $\{3\}$ unique et $\{0\}$ unique

12. On construit une classe union-find avec tous les sommets. Initialement, chaque classe contient un unique sommet. On parcourt alors tous les arcs du graphe et pour chaque arc $x-y$, on unifie les classes des sommets x et y . Les composantes fortement connexes du graphe seront dans la même classe.

La complexité de find et union sont en $O(\log n)$
Cet algorithme sera donc de complexité $O(\log n)$

13. Premier appel par $v=0$:

On appelle dans l'ordre:

* dfs(0)

* dfs(3)

* dfs(0)

* dfs(4)

* dfs(1)

* dfs(4)

* dfs(3)

* dfs(4)

* dfs(2)

* dfs(5)

* dfs(2)

NOTE

N°
2
2

NE PAS ÉCRIRE DANS CE CADRE

Le tableau renvoyé est le suivant:

$[0; 0; 1; 0; 0; 1]$

14. Supposons par l'absurde que $C(G)$ contienne un cycle:
Prenons i et j deux sommets de $C(G)$ par lesquels
passe ce cycle. Alors il existe un chemin de i à j et de
 j à i donc i et j sont fortement connexes, ce qui est
absurde par construction de $C(G)$.

Cd: $C(G)$ ne contient pas de cycle.

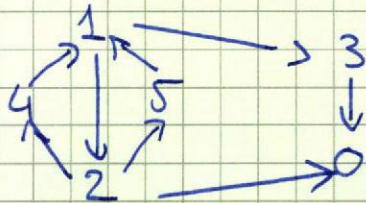
15. static Graph reverse (Graph g) {
 Graph reverse = new Graph(g.V);
 for (int v = 0, v < g.V, v++) {
 for (int w : g.adj(v)) {
 reverse.addEdge(w, v);
 }
 }
 return reverse;
}

16. G_2^R : postOrder reverse
0, 1, 2, 4, 3, 5, 7, 6

17. Considérons un graphe acyclique.
Soient u et v deux sommets tels que $u \rightarrow v$.
Alors la visite du sommet u se termine avant celle du sommet v
donc u apparaît avant v dans la liste renvoyée par postOrder.

NE PAS ÉCRIRE DANS CE CADRE

18. Commençons par dessiner reverse (G_3):



Puis par déterminer $\text{postOrder}(G^R_3)$:

3, 5, 4, 2, 1, 0

On commence donc par visiter le sommet 3 :
On appelle dans l'ordre:

$\text{dfs}(3)$
 $\text{dfs}(1)$
 $\text{dfs}(4)$
 $\text{dfs}(2)$
 $\text{dfs}(5)$
 $\text{dfs}(2)$
 $\text{dfs}(0)$
 $\text{dfs}(2)$
 $\text{dfs}(3)$

19.

NE PAS ÉCRIRE DANS CE CADRE