



2509

INF411

COMPOSITION d'INF 411

en date du 16/11/2021

Expressions régulières1) $a \dots * b$

```

2) static String toString(SRE r) {
    StringBuffer sb = new StringBuffer();
    while (r != null) {
        sb.append(r.c);
        if (r.star) sb.append("*");
        r = r.next;
    }
    return sb.toString();
}

```

```

3) static SRE ofString(String s) {
    SRE l = null;
    int n = s.length();
    int i = 0;
    while (i < n) {
        if (s[i] == '.') {
            if ((i+1 < n) && (s[i+1] == '*')) {
                l.next = new SRE('.', true, null);
                i = i + 2;
            }
            else {
                l.next = new SRE('.', false, null);
                i = i + 1;
            }
        }
        else {
            l.next = new SRE(s[i], false, null);
            i = i + 1;
        }
    }
    return l;
}

```

NOTE

N°

1/2

NE PAS ÉCRIRE DANS CE CADRE

```

    }
    else {
        char c = s.charAt(i);
        if ((i+1 < n) && (s[i+1] == '*')) {
            l.next = new SRS(c, true, null);
            i = i + 2;
        }
        else {
            l.next = new SRS(c, false, null);
            i = i + 1;
        }
    }
}
return l;

```

4) Appel	1	2	3	4
Arguments	s	s	s	s
	"aa"	"aa"	"aa"	"aa"
i	0	0	0	1
l	a*	a	a*	a*
Résultat	matches(s, 0, a) (2 ^e if)	matches(s, 1, null) <u>false</u>	matches(s, 1, a*)	match(s, 1, a) ↓ match(s, 2, .) <u>false</u>
Puis	⑦	⑧		
	match(s, 0, a) → match(s, 1, .) → true			

NE PAS ÉCRIRE DANS CE CADRE

5) A chaque parcours.

(1). soit $i = i + 1$

(2). soit $n = n \cdot m \cdot e \cdot r$

dans le nouvel appel à `matches`

Or il y a en tout $i \leq s.length$

et (2) assure qu'en sorte du 2^e if

6) ligne 3. renvoie false si $s == null$, $0 \leq i$ ou $i \leq s.length()$

7) $s = \underbrace{a a \dots a}_{n \text{ fois}}$

$n = \underbrace{a * a * \dots a * a}_{n-1 \text{ fois}}$

(Pos de preuve du 2ⁿ)

2^N

8) chaîne s : aaa et n : $a * a * a$.

On calcule : $\begin{cases} \text{matches}(s, 0, a) \\ \text{matches}(s, 1, a) \\ \text{matches}(s, 2, null) \end{cases}$

puis en sort de :

et on calcule : $\begin{cases} \text{matches}(s, 1, a^*) \\ \text{matches}(s, 1, a) \end{cases}$

⑤

⑥

NE PAS ÉCRIRE DANS CE CADRE

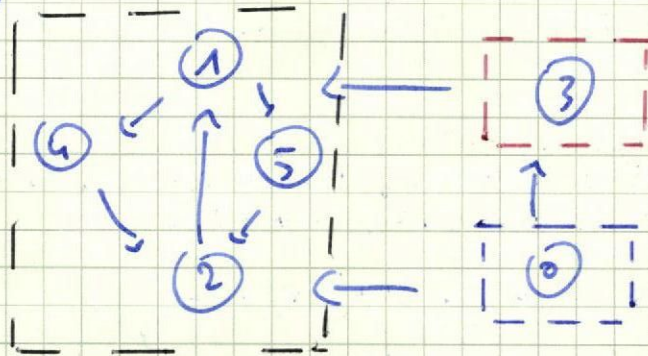
```
3) public int hashCode() {  
    return this.i + this.n.c.hashCode();  
}
```

```
public boolean equals(Object o) {  
    MemoKey m = (MemoKey) o;  
    return (this.i == m.i) && (this.n.equals(m.n));  
}
```

10)

I Composantes fortement connexes

11)



12) . Au départ, chaque sommet est son propre représentant.

. On parcourt les sommets de g .

. Pour chacun, on parcourt ses voisins, et on affecte le sommet commun représentant de ses voisins, sauf pour les sommets déjà affectés (utiliser un marqueur).

. Chaque représentant constituant alors une composante fortement connexe.

13) En notant $dfs(v)$:

$dfs(0) \rightarrow dfs(4) \rightarrow dfs(1) \rightarrow dfs(3) \rightarrow dfs(2) \rightarrow dfs(5)$

et $num = [0, 0, 1, 0, 0, 1]$

NOTE

NE PAS ÉCRIRE DANS CE CADRE

14) Si $C(G)$ contient un cycle, alors les sommets de ce cycle constituent une composante fortement connexe et peuvent être rassemblés en un seul sommet.

→ $C(G)$ ne contient pas de cycle

```
static Graph reverse (Graph g) {
    boolean[] visited = new boolean[g.V];
    for (int v = 0, v < g.V, v++) {
        for (int w : g.adj(v)) {
            if (! (v in g.adj(w)) && (! visited(w))) {
                g.adj(v).removeFirst();
                addEdge (w, v);
            }
        }
        visited[v] = true;
    }
    return g;
}
```

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

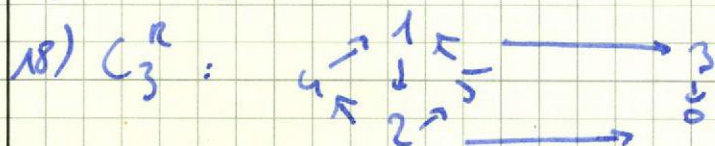
NE PAS ÉCRIRE DANS CE CADRE

16) 0 1 2 4 3 5 7 6

17) Sur un graphe acyclique :

On appelle dfs Post de manière rétroactive jusqu'à ce qu'un sommet n'ait plus de voisins. On crée ainsi une pile successive.

Le dernier appelé est le premier sommet à être ajouté à la liste. Comme on utilise addFirst, il se retrouve ensuite après son prédécesseur, etc. On retrouve l'ordre $u \rightarrow v$.



postOrder : 1 2 4 5 3 0

dfs(1) \rightarrow dfs(4) \rightarrow dfs(2) \rightarrow dfs(5) \rightarrow dfs(3) \rightarrow dfs(0)

renvoie : [2, 0, 0, 1, 0, 0]

19) Le coût de l'affectation dans le tableau est $O(1)$.

Conjecture : $O(V)$

NE PAS ÉCRIRE DANS CE CADRE