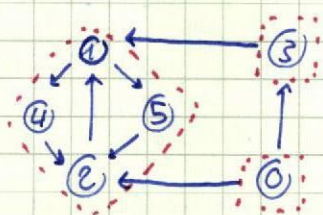


Composantes fortement connexes:Question 11:

 : composantes fortement connexes.

Question 12:

On crée une structure union-find de taille $g.V$ puis on fait $\text{union}(i, j)$ pour chaque paire de sommets (i, j) , qui leur assignera le même référent s'ils sont fortement connectés d'un à l'autre. union est de complexité $O(\log(g.V))$.

À la fin, chaque composante fortement connexe du graphe est contenue dans une classe de la structure. L'algorithme est de complexité $O(n^2)$.

Question 13:

appels

num

dfs(visited, g, 0)

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

dfs(visited, g, 3)

dfs(visited, g, 4)

dfs(visited, g, 0)

dfs(visited, g, 1)

dfs(visited, g, 4)

dfs(visited, g, 2)

dfs(visited, g, 5)

dfs(visited, g, 2)

NOTE

N°

1

NE PAS ÉCRIRE DANS CE CADRE

Question 14:

Soit un sommet d'une CFC, alors il est possible d'aller de ce sommet à tout sommet de la CFC.

Supposons qu'il existe un cycle de CFC, et x_1, x_2, x_3 sont respectivement des sommets de ces CFC.

Alors on peut aller de x_1 à toute la CFC1, et à x_2 , et de x_2 à x_1 , donc de x_2 à toute la CFC1, et donc de toute la CFC1 à x_2 .

Donc x_2 fait partie de la CFC1 et $CFC1 = CFC2$.

De même pour x_3 .

Il n'y a donc en fait pas de cycle mais juste une grande CFC.

Question 15:

```
static Graph reverse(Graph g) {
```

```
    Graph grev = new Graph(g.V);
```

```
    for (i=0; i<V; i++) {
```

```
        for (int u : g.adj(i)) { // si  $i \rightarrow u$  existe dans  $g$ 
```

```
            grev.addEdge(u, i); // on crée l'arc  $u \rightarrow i$  dans  $grev$ 
```

```
    return grev;
```

Question 16:

La liste renvoyée est: 0 1 2 4 3 5 6

NE PAS ÉCRIRE DANS CE CADRE

Question 17:

L'enchaînement des boucles for et la récursivité font que pour un arc $u \rightarrow v$, si l'on est dans $\text{dfs}(u)$, on lancera $\text{dfs}(v)$ ligne 15, avant d'ajouter u à order , ligne 16. Or, la pile d'appel fait que cette ligne 16 ne sera pas appelée tant que $\text{dfs}(v)$ n'aura pas terminé son exécution. Donc v sera ajoutée à order avant u si il y a un arc $u \rightarrow v$ et que le graphe est acyclique.

(car dans un graphe cyclique, v peut être traité avant u malgré l'existence de l'arc $u \rightarrow v$, si un autre chemin mène de v avant u)

Question 18:

Notons d'abord le résultat de postOrder si G^R : 7 3 2 5 4 0

Les appels à dfs donnent:

appels	num
$\text{dfs}(\text{visited}, g, 1)$	2 0 0 1 0 0
$\text{dfs}(\text{visited}, g, 4)$	
$\text{dfs}(\text{visited}, g, 2)$	
$\text{dfs}(\text{visited}, g, 1)$	
$\text{dfs}(\text{visited}, g, 5)$	
$\text{dfs}(\text{visited}, g, 2)$	
$\text{dfs}(\text{visited}, g, 3)$	
$\text{dfs}(\text{visited}, g, 1)$	
$\text{dfs}(\text{visited}, g, 0)$	
$\text{dfs}(\text{visited}, g, 2)$	

NE PAS ÉCRIRE DANS CE CADRE

Question 19:

Question 20:

En établissant la liste postOrder sur G^R , on a ensuite la certitude, quand on traite un sommet u avant un sommet v , qu'il existe un chemin de u à

Expression régulière:

Question 1:

L'expression régulière correspondante serait: $a \dots b$.

Question 2:

```
static String toString(SRE r) {  
    String res StringBuffer sb = new StringBuffer()  
    if while (r.next != null) {
```

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


COMPOSITION d'INF 411

en date du 16/11/2021

Question 3:

```
static SRE ofString(String s) {  
    if s.len SRE res = new SRE(); SRE act = res;  
    for (int i=0; i < s.length(); i) {  
        act.c = s.charAt(i)  
        if (s.charAt(i+1).equals("#")) {  
            act.star = True;  
            i += 2; }  
        act.next =  
        else {  
            i += 1 }  
        if i < s.length (i < s.length) { act.next = new SRE(); act = act.next; }  
    }  
    return res;  
}
```

Question 4:

matches("aa", r) $\xrightarrow{2,6}$ matches("aa", 0, r) $\xrightarrow{1,4}$ true
r \Rightarrow a*a
r.next \Rightarrow a
r.next.next \Rightarrow next
matches("aa", 0, r.next) $\xrightarrow{3,5}$ true
matches("aa", 1, r.next) $\xrightarrow{4,6}$ true
matches("aa", 2, r.next.next) $\xrightarrow{5}$ true

NOTE

N°

2

NE PAS ÉCRIRE DANS CE CADRE

Question 5:

Chaque appel fait avancer soit i soit r (remplacé par $r.next$), on attendra donc toujours soit la vérification `if(r==null)` ligne 4 qui renvoie un boolean selon l'avancée de i .

Question 6:

Le premier assert ligne 3 empêche de sortir des bornes de s , et le premier `if` ligne 4 nous assure ensuite que r n'est pas nul.

Question 7:

Question 8:

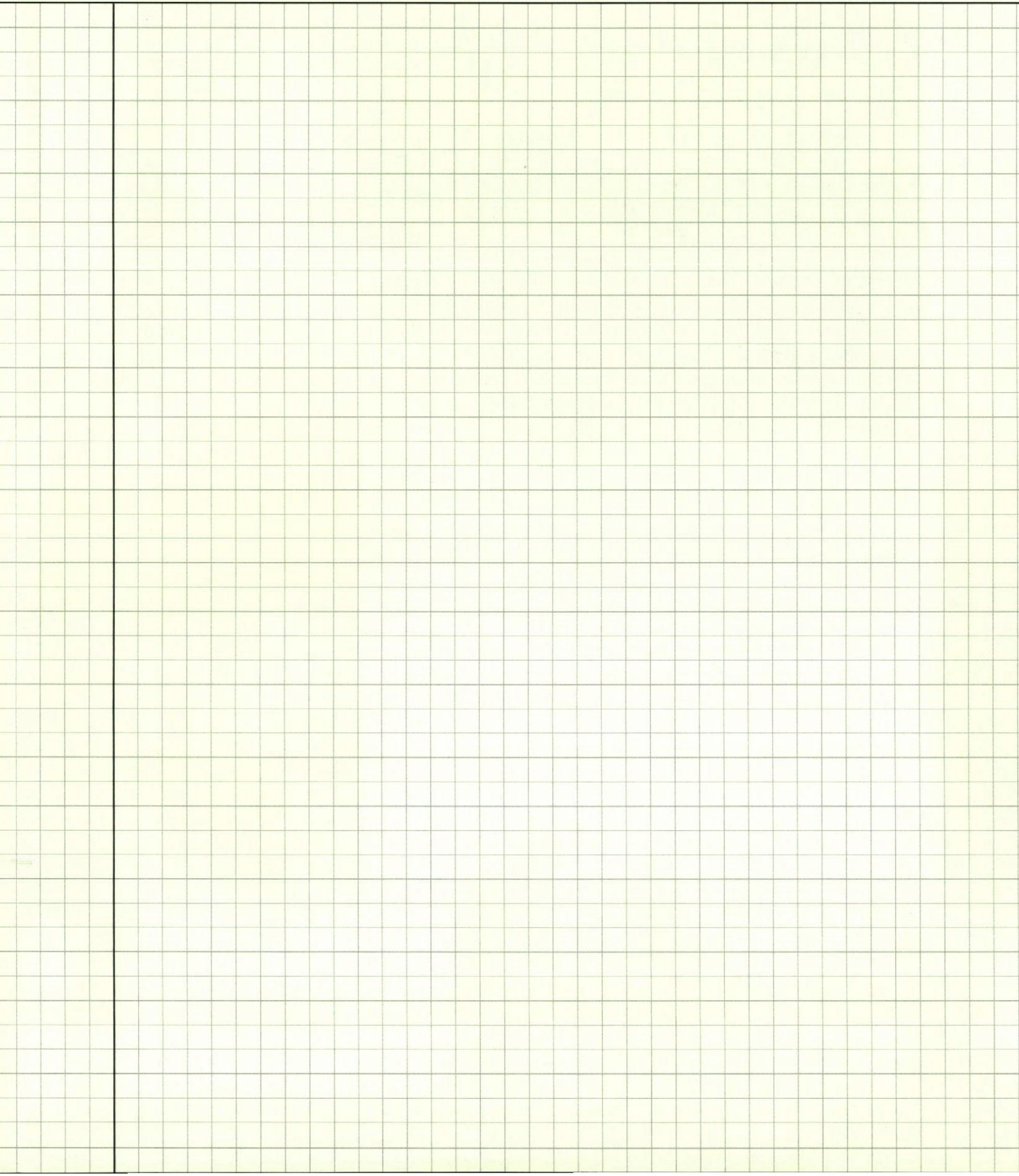
On peut penser à

Question 9:

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

On souhaite bien ici que `this.r` et `m.r` pointent vers le même SRE.

NE PAS ÉCRIRE DANS CE CADRE



NE PAS ÉCRIRE DANS CE CADRE