Algorithmique & Programmation (INF431)
Contrôle classant CC1

28 avril 2014

Les parties I et II sont indépendantes l'une de l'autre. Elles peuvent être traitées dans l'ordre de votre choix. Elles peuvent être traitées sur la même feuille.

Première partie

Cordes

La classe `String` de Java permet de représenter des chaînes de caractères. On rappelle que cette classe est munie (en particulier) des méthodes suivantes :

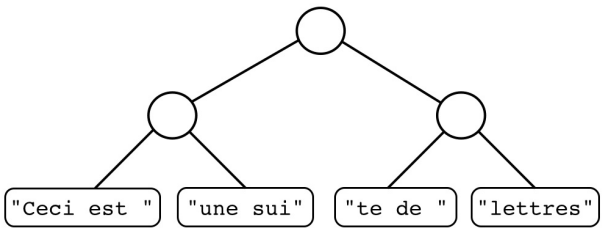
```
   int length ()
String substring (int i, int j)
```

La première méthode renvoie la longueur de la chaîne concernée. Sa complexité en temps est $O(1)$. La seconde méthode exige `0 <= i <= j <= n`, où `n` est la longueur de la chaîne concernée. Cette méthode, appliquée à une chaîne constituée des caractères $c_0, c_1, \ldots c_{n-1}$, construit et renvoie la chaîne $c_i, c_{i+1}, \ldots, c_{j-1}$. On note que `s.substring(0, s.length())` est égal à `s`. On rappelle enfin que l'opérateur `+` permet de concaténer deux chaînes de caractères.

Les *cordes*, en anglais *ropes*, sont une structure de données permettant de représenter les chaînes de caractères. Elles offrent des fonctionalités similaires à la classe `String` mais effectuent plus efficacement certaines opérations, comme l'insertion à l'intérieur d'une corde.

Dans ce qui suit, on appelle **chaîne de caractères** un objet de classe `String`.

Une **corde** est un arbre binaire dont les feuilles contiennent des chaînes de caractères. La chaîne représentée par une corde est obtenue en concaténant les feuilles de gauche à droite. Voici une représentation possible (il y en a plusieurs) de `"Ceci est une suite de lettres"` :



On se donne une classe abstraite `Rope` qui représente une corde. Pour l'implémentation, on définit deux classes concrètes `Leaf` et `Node`, qui représentent respectivement une feuille et un nœud binaire. Notez que les champs sont non mutables.

```
abstract class Rope {}

class Leaf extends Rope {
  private final String s;
  Leaf (String st) { s = st; }
}

class Node extends Rope {
  private final Rope left;
  private final Rope right;
  Node (Rope l, Rope r) { left = l; right = r; }
}
```

Dans les questions qui suivent, on va progressivement étendre ces définitions.

Question 1 Étendez ces définitions de façon à ce que, si `r` est un objet de classe `Rope`, alors `r.toString()` renvoie la chaîne représentée par `r`. Ne vous inquiétez pas de l'efficacité de votre solution. ◇

Question 2 Étendez les définitions de façon à ce que les cordes soient munies d'une méthode `int length()`. On souhaite que, si `r` est une corde, alors `r.length()` renvoie la longueur de la chaîne représentée par `r`. **On exige de plus que cette méthode ait une complexité en temps** $O(1)$ **dans le cas le pire.** Vous pouvez si besoin ajouter un champ (ou des champs) aux classes `Leaf` et `Node` et modifier les constructeurs de ces classes. Toutefois, **on exige que la complexité en temps des constructeurs reste** $O(1)$ **dans le cas le pire.** ◇

Dans les trois questions qui suivent, on refuse les solutions qui demandent la construction d'une chaîne de caractères inutilement grande. Par exemple, dans la question 3 ci-dessous, on refuse la réponse triviale `System.out.print(r.toString().substring(i, j))`.

Question 3 Munissez les cordes d'une méthode `void printSubstring(int i, int j)` de sorte que, si `r` est une corde, alors `r.printSubstring(i, j)` affiche la sous-chaîne de `r` formée par les caractères compris entre les positions `i` (inclus) et `j` (exclus). Vous pourrez supposer `0 <= i <= j <= r.length()`. Cette convention à propos des indices `i` et `j` est la même que celle de la méthode `substring` de la classe `String`, dont le comportement a été rappelé en préambule. Vous ferez en sorte que `printSubstring` s'exécute en temps constant dans le cas particulier où `i == j`. ◇

On se donne la classe suivante, qui représente une paire de cordes :

```
class PairRope {
  Rope left;
  Rope right;
  PairRope (Rope l, Rope r) { left = l; right = r; }
}
```

Question 4 Munissez les cordes d'une méthode `PairRope split (int i)` qui renvoie une paire formée par deux cordes représentant les sous-chaînes délimitées respectivement par les indices [`0`...`i`[et [`i`...`r.length()`[. Vous pourrez supposer `0 <= i <= r.length()`. ◇

Question 5 À l'aide de la méthode `split` et en modifiant uniquement la classe `Rope`, définissez une méthode `Rope substring (int i, int j)` qui renvoie une corde représentant la sous-chaîne délimitée par les indices [`i`...`j`[. Vous pourrez supposer `0 <= i <= j <= r.length()`.◇

Deuxième partie

Calcul des distances dans un graphe

1 Exponentiation rapide de matrices booléennes

On suppose qu'une opération élémentaire sur les nombres entiers (addition, soustraction, multiplication, comparaison) exige un temps $O(1)$. On suppose également qu'une lecture ou une écriture en mémoire exige un temps $O(1)$.

Si A et B sont des matrices carrées de taille n à coefficients entiers, on note $A \cdot B$ leur produit au sens de l'anneau $(\mathbb{Z}, +, \times)$. Pour calculer ce produit, on suppose que l'on dispose d'un algorithme de multiplication dont la complexité en temps est $O(n^\omega)$.

Question 6 Donnez deux entiers p et q pour lesquels on peut supposer $p \leq \omega \leq q$. Justifiez brièvement pourquoi. ◇

On identifie les booléens **faux** et **vrai** avec les entiers 0 et 1. Une matrice à coefficients booléens peut donc être considérée également comme une matrice à coefficients entiers.

On note $\mathbb{B} = \{0, 1\}$ l'ensemble des booléens. Sur cet ensemble, l'opération « ou » est appelée disjonction et notée $\vee$; l'opération « et » est appelée conjonction et notée $\wedge$. La structure $(\mathbb{B}, \vee, \wedge)$ forme un semi-anneau. (Un semi-anneau doit vérifier les mêmes propriétés qu'un anneau, à ceci près que les éléments n'ont pas forcément d'opposé. Dans $(\mathbb{B}, \vee, \wedge)$, il n'y a pas de notion d'opposé pour l'opération de disjonction.)

Si A et B sont des matrices carrées de taille n à coefficients booléens, on note $A \odot B$ leur produit au sens du semi-anneau $(\mathbb{B}, \vee, \wedge)$. On souligne que $A \cdot B$ et $A \odot B$ sont deux opérations distinctes. On note Id la matrice identité, qui est la même pour ces deux opérations.

On a donc par exemple, pour $n = 2$:

$$Id = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \odot \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} = \begin{bmatrix} (a_{00} \wedge b_{00}) \vee (a_{01} \wedge b_{10}) & (a_{00} \wedge b_{01}) \vee (a_{01} \wedge b_{11}) \\ (a_{10} \wedge b_{00}) \vee (a_{11} \wedge b_{10}) & (a_{10} \wedge b_{01}) \vee (a_{11} \wedge b_{11}) \end{bmatrix}$$

Question 7 Montrez que, pour calculer le produit $A \odot B$ de deux matrices booléennes A et B de taille n, on dispose d'un algorithme dont la complexité en temps est $O(n^\omega)$. On ne demande pas de code ni de pseudo-code. ◇

On s'intéresse maintenant à des graphes orientés dont les n sommets sont les entiers de l'intervalle $[0, n-1]$. On représente un tel graphe A par sa **matrice d'adjacence**, également notée A. Il s'agit d'une matrice carrée de taille n dont le coefficient A_{ij} vaut 1 s'il existe une arête du sommet i vers le sommet j et 0 dans le cas contraire.

Question 8 Pour cette question, on se place dans un semi-anneau quelconque, par exemple $(\mathbb{Z}, +, \times)$ ou $(\mathbb{B}, \vee, \wedge)$, et on considère des matrices à coefficients dans ce semi-anneau. Donnez (sous forme de pseudo-code précis) un algorithme efficace qui, étant donnés une matrice A et un entier $k \geq 0$, calcule la matrice A^k, c'est-à-dire « A à la puissance k ». Quelle est sa complexité en temps ? ◇

Question 9 Étant donné un graphe A, on souhaite déterminer, pour tous les sommets i et j, s'il existe dans A un chemin de i vers j. En vous appuyant sur vos réponses aux questions

précédentes, indiquez comment effectuer efficacement ce calcul. On ne demande pas de code ni de pseudo-code. Justifiez brièvement pourquoi ce calcul est correct et quelle est sa complexité en temps. ◇

2 Calcul des distances dans un graphe non orienté connexe

On s'intéresse maintenant à un graphe A **non orienté** dont les n sommets sont les entiers de l'intervalle $[0, n-1]$. Sa matrice d'adjacence A est donc symétrique : $A_{ij} = A_{ji}$. On suppose de plus que cette matrice est irréflexive : $A_{ii} = 0$. On suppose enfin que le graphe A est **connexe**.

On appelle distance de i à j la longueur (c'est-à-dire le nombre d'arêtes) du plus court chemin dans A de i vers j. On note cette distance $\hat{A}_{ij}$. On souligne que, puisque le graphe A est connexe, il existe au moins un chemin entre i et j, donc $\hat{A}_{ij}$ est un entier. On souligne que la distance d'un sommet à lui-même est nulle : $\hat{A}_{ii} = 0$.

On appelle **diamètre** du graphe A, et on note $\delta(A)$, la plus grande distance entre deux sommets : $\delta(A) = \max_{i,j} \hat{A}_{ij}$.

Pour calculer toutes les distances entre sommets dans le graphe A, on souhaite se ramener à un graphe B, défini ainsi :

$$\begin{array}{ll} B_{ij} = 1 & \text{si } \hat{A}_{ij} = 1 \text{ ou } \hat{A}_{ij} = 2 \\ B_{ij} = 0 & \text{sinon} \end{array}$$

En termes imagés, à chaque fois que deux sommets i et j sont situés à distance 2 l'un de l'autre dans A, il existe dans B un « raccourci », c'est-à-dire une arête entre i et j.

Question 10 Montrez que la matrice B est irréflexive et symétrique. Montrez que le graphe B est connexe. ◇

Question 11 Indiquez comment calculer efficacement la matrice B à partir de la matrice A. Quelle est la complexité de ce calcul ? ◇

Question 12 Comment, à l'aide des matrices A et B, peut-on déterminer si le diamètre du graphe A est inférieur ou égal à 1 ? Quelle est, dans ce cas, la matrice $\hat{A}$? ◇

Pour les estimations de distance qui suivent, on travaille exclusivement avec des nombres entiers. En particulier, on note simplement $n/2$ le quotient de la division **euclidienne** de n par 2.

Donc, l'entier $n/2$ est égal à $\lfloor \frac{n}{2} \rfloor$, et l'entier $(n+1)/2$ est égal à $\lceil \frac{n}{2} \rceil$.

Question 13 Montrez que $\hat{A}_{ij} \leq 2\hat{B}_{ij}$. ◇

Question 14 Montrez que $\hat{B}_{ij} \leq (\hat{A}_{ij} + 1)/2$. ◇

Question 15 Donnez une équation qui caractérise $\hat{B}_{ij}$ en fonction de $\hat{A}_{ij}$. Inversement, peut-on déduire $\hat{A}_{ij}$ de $\hat{B}_{ij}$? ◇

Question 16 Donnez une relation qui caractérise $\delta(B)$ en fonction de $\delta(A)$. Déduisez-en que, si $\delta(A) > 1$, alors $\delta(B) < \delta(A)$. ◇

Question 17 Soient i et j deux sommets distincts. Montrez que, pour tout voisin k de j dans le graphe A, on a $\hat{A}_{ij} - 1 \leq \hat{A}_{ik} \leq \hat{A}_{ij} + 1$. Existe-t-il au moins un voisin k de j telle que l'inégalité de gauche est une égalité ? Même question pour l'inégalité de droite. ◇

Question 18 Soient i et j deux sommets distincts, et k un voisin de j dans le graphe A. Déduisez de la question précédente des inégalités reliant $\hat{B}_{ik}$ et $\hat{B}_{ij}$. Vous distinguerez le cas où $\hat{A}_{ij}$ est pair et le cas où $\hat{A}_{ij}$ est impair. ◇

On note $degré_A(j)$ le nombre de voisins du sommet j dans A.

Question 19 Soient i et j deux sommets distincts. Montrez que si $\hat{A}_{ij}$ est pair, alors on a :

$$\left(\sum_{k \text{ voisin de } j \text{ dans } A} \hat{B}_{ik} \right) \geq \hat{B}_{ij} \times degré_A(j)$$

et que si $\hat{A}_{ij}$ est impair, alors on a au contraire :

$$\left(\sum_{k \text{ voisin de } j \text{ dans } A} \hat{B}_{ik} \right) < \hat{B}_{ij} \times degré_A(j)$$

◇

Question 20 On définit la matrice S par $S_{ij} = \left(\sum_{k \text{ voisin de } j \text{ dans } A} \hat{B}_{ik} \right)$. Comment, à partir des matrices A et $\hat{B}$, peut-on calculer facilement et efficacement la matrice S ? ◇

Question 21 À l'aide des résultats accumulés lors des questions 10 à 20, définissez (sous forme de pseudo-code précis) un algorithme récursif qui, étant donnée la matrice d'adjacence d'un graphe A irréflexif, non orienté, connexe, calcule la matrice $\hat{A}$. Démontrez que l'algorithme termine, et donnez sa complexité en temps. ◇