Numéro d'anonymat :

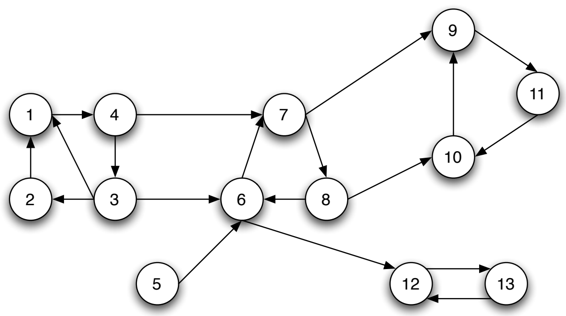
UE LU3IN003 - Algorithmique.
Licence d'informatique.

Partiel du 18 novembre 2022. Durée : 1h30

Documents non autorisés. Seule une feuille A4 portant sur les cours est autorisée.
Tous objets connectés éteints et rangés dans vos sacs.

Exercice 1 (4 points)

On considère le graphe G_1 représenté sur la figure ci-dessous.



Question 1 (1/4) — Déterminer les composantes fortement connexes de G_1 et le graphe réduit $\hat{G}_1$ de G_1.

Question 2 (1/4) — Quels sont les points de régénération du parcours générique $L = (6, 7, 9, 11, 10, 8, 12, 13, 5, 4, 3, 2, 1)$? Donner une forêt couvrante associée à L.

Question 3 (1/4) — Donner un parcours de G_1 comportant exactement deux points de régénération. Donner un parcours de G_1 comportant exactement cinq points de régénération.

Question 4 (1/4) — Dans le cas général, donner le nombre de points de régénération minimum et maximum du parcours d'un graphe orienté. On ne demande pas de justification.

Exercice 2 (7 points)

Dans cet exercice, on considère un tableau de n entiers, noté A, dont les cellules 1 à n comportent une permutation des entiers 1 à n. Une paire (i, j) est une *inversion* pour le tableau A si $i < j$ et $A[i] > A[j]$. Par exemple, le tableau $A = [6, 2, 7, 1, 3, 5, 4]$ comporte 11 inversions : $(1, 2), (1, 4), (1, 5), (1, 6), (1, 7), (2, 4), (3, 4), (3, 5), (3, 6), (3, 7), (6, 7)$. L'objet de l'exercice est de concevoir et d'étudier des algorithmes pour compter le nombre d'inversions dans un tableau A.

Question 1 (2/7) — Donner en quelques mots le principe d'un algorithme naïf permettant de compter le nombre d'inversions dans un tableau A. Quelle est la complexité de cet algorithme ?

On s'intéresse à présent à un algorithme de type diviser pour régner, appelé Trier-et-Compter, dont le pseudo-code est donné ci-après, afin de compter le nombre d'inversions dans un tableau A. Dans la suite de l'exercice, on notera $|A|$ la taille d'un tableau A.

Trier-et-Compter

Entrées : A : un tableau d'entiers
si $|A| = 1$ **alors retourner** $(0, A)$
sinon
    Diviser le tableau A en deux moitiés A_1 et A_2
    $(m_1, A'_1) \leftarrow$ Trier-et-Compter(A_1)
    $(m_2, A'_2) \leftarrow$ Trier-et-Compter(A_2)
    $(m_3, A') \leftarrow$ Fusionner-et-Compter(A'_1, A'_2)
fin
retourner $(m_1 + m_2 + m_3, A')$

Trier-et-Compter est fondé sur une partition en 3 catégories des inversions présentes dans A :

1. L'ensemble des inversions (i, j) avec $i \leq n/2$ et $j \leq n/2$.
2. L'ensemble des inversions (i, j) avec $i > n/2$ et $j > n/2$.
3. L'ensemble des inversions (i, j) avec $i \leq n/2$ et $j > n/2$.

Le nombre d'inversions du premier type, noté m_1, est obtenu par un appel récursif sur la première moitié du tableau A (notée A_1). De même, le nombre d'inversions du second type, noté m_2, est obtenu par un appel récursif sur la deuxième moitié de A (notée A_2). Pour le nombre d'inversions du troisième type, noté m_3, on fait appel à un algorithme, nommé Fusionner-et-Trier (voir pseudo-code), qui prend en entrée les sous-tableaux A_1 et A_2 préalablement triés (notés A'_1 et A'_2), et qui réalise les deux opérations suivantes simultanément :

— Le calcul de la valeur de m_3.

— Le tri du tableau A en fusionnant les tableaux triés A'_1 et A'_2 (le tableau trié est noté A').

Enfin, Trier-et-Compter termine en retournant le nombre d'inversions du tableau A (c'est-à-dire $m_1 + m_2 + m_3$) ainsi que le tableau trié A'.

Question 2 (1/7) — Compléter la ligne 5 de Fusionner-et-Compter avec un calcul en $O(1)$ afin que m_3 corresponde bien au nombre d'inversions du troisième type au terme de la boucle tant que.
Une indication est donnée en haut de la page suivante.

Fusionner-et-Compter

Entrées : A'_1 et A'_2 : deux tableaux triés

1 $i \leftarrow 1$; $j \leftarrow 1$; $A' \leftarrow [\,]$; $m_3 \leftarrow 0$
2 **tant que** $i \leq |A'_1|$ *et* $j \leq |A'_2|$ **faire**
3   **si** $A'_1[i] > A'_2[j]$ **alors**
4   Ajouter $A'_2[j]$ à la fin du tableau A' // instruction en $O(1)$
5         $m_3 \leftarrow$ ....
6         $j \leftarrow j + 1$
7   **fin**
8   **sinon**
9   Ajouter $A'_1[i]$ à la fin du tableau A' // instruction en $O(1)$
10         $i \leftarrow i + 1$
11   **fin**
12 **fin**
13 **tant que** $i \leq |A'_1|$ **faire**
14   Ajouter $A'_1[i]$ à la fin du tableau A' // instruction en $O(1)$
15     $i \leftarrow i + 1$
16 **fin**
17 **tant que** $j \leq |A'_2|$ **faire**
18   Ajouter $A'_2[j]$ à la fin du tableau A' // instruction en $O(1)$
19     $j \leftarrow j + 1$
20 **fin**
21 **retourner** (m_3, A')

Indication : Supposons que $A'_1 = [2, 6, 7]$ et $A'_2 = [1, 3, 4, 5]$ en entrée de Fusionner-et-Compter.
Pour $i = 1$ et $j = 1$, on détecte 3 inversions impliquant l'entier 1 car $A'_1[i] = 2 > 1 = A'_2[j]$.
Pour $i = 1$ et $j = 2$, pas d'inversion supplémentaire impliquant 2 car $A'_1[i] = 2 < 3 = A'_2[j]$.
Pour $i = 2$ et $j = 2$, on détecte 2 inversions impliquant l'entier 3 car $A'_1[i] = 6 > 3 = A'_2[j]$. Etc.

Question 3 (2/7) — Quelle est la complexité de Fusionner-et-Compter ? Justifier brièvement.

Question 4 (2/7) — Soit $T(n)$ le nombre d'opérations effectuées par l'algorithme Trier-et-Compter. Donner la formule de récurrence vérifiée par T. A l'aide d'un théorème vu en cours, en déduire la complexité de l'algorithme.

Exercice 3 (9 points)

Dans cet exercice, on considère une chaîne s de bits (0 ou 1), indicée de 1 à n. Initialement, la chaîne s est constituée uniquement de 1. On vise à transformer s en une chaîne constituée uniquement de 0 en ne s'autorisant que les deux types suivants de transformations de s (que l'on peut réaliser autant de fois que l'on souhaite), appelées *permutations autorisées* dans la suite :

1. on peut toujours permuter la valeur du bit d'indice 1 (changer un 0 en 1, ou un 1 en 0) ;
2. si s débute par une séquence d'exactement i bits à 0 (le bit d'indice $i+1$ est 1), alors on peut permuter la valeur du bit d'indice $i+2$ (changer un 0 en 1, ou un 1 en 0).

Par exemple, pour $n=5$, la séquence de permutations autorisées suivante permet de transformer $s=11111$ en la chaîne 00000 (le chiffre au dessus de chaque flèche est l'indice du bit permuté) :

$$11111 \xrightarrow{1} 01111 \xrightarrow{3} 01011 \xrightarrow{1} 11011 \xrightarrow{2} 10011 \xrightarrow{1} 00011 \xrightarrow{5} 00010$$
$$\xrightarrow{1} 10010 \xrightarrow{2} 11010 \xrightarrow{1} 01010 \xrightarrow{3} 01110 \xrightarrow{1} 11110 \xrightarrow{2} 10110 \xrightarrow{1} 00110 \xrightarrow{4} 00100$$
$$\xrightarrow{1} 10100 \xrightarrow{2} 11100 \xrightarrow{1} 01100 \xrightarrow{3} 01000 \xrightarrow{1} 11000 \xrightarrow{2} 10000 \xrightarrow{1} 00000$$

Les deux procédures *mutuellement récursives* ci-dessous (c'est-à-dire qui s'appellent l'une l'autre) permettent d'effectuer une mise à zéro de s qui fait uniquement appel à des permutations autorisées :

— MiseAZero(k), pour $k \geq 0$, produit une séquence de permutations autorisées qui change les k premiers bits de s, qui doivent être tous de valeur 1 en entrée, en une séquence de k bits 0.

— MiseAUn(k), pour $k \geq 0$, produit une séquence de permutations autorisées qui change les k premiers bits de s, qui doivent être tous de valeur 0 en entrée, en une séquence de k bits 1.

Ces séquences de permutations autorisées sont appelées séquences *valides* dans la suite. L'appel initial avec une chaîne s constituée de n bits tous de valeur 1 est MiseAZero(n). L'objectif de l'exercice est de prouver la validité de l'algorithme et d'analyser sa complexité.

```
MiseAZero(k)
  si k=1
    s[1] ← 0
  sinon si k>1
    MiseAZero(k−2)
    s[k] ← 0
    MiseAUn(k−2)
    MiseAZero(k−1)
```

```
MiseAUn(k)
  si k=1
    s[1] ← 1
  sinon si k>1
    MiseAUn(k−1)
    MiseAZero(k−2)
    s[k] ← 1
    MiseAUn(k−2)
```

Question 1 (2/9) — Donner l'arbre des appels récursifs généré par l'appel MiseAZero(3). En déduire la séquence de permutations autorisées pour convertir la chaîne 111 en 000.

Tourner la page SVP

Question 2 (2/9) — Prouver par récurrence que les appels MISEAZERO(k) et MISEAUN(k) produisent chacun une séquence valide. En déduire la validité de MISEAZERO(n) pour transformer la chaîne s complète.

Indication : Il y a deux cas de base $k = 0$ et $k = 1$. L'étape inductive consistera à montrer que si MISEAZERO($k-2$), MISEAZERO($k-1$), MISEAUN($k-2$) et MISEAUN($k-1$) produisent chacun une séquence valide, alors MISEAZERO(k) et MISEAUN(k) également. On prouvera l'étape inductive uniquement pour MISEAZERO(k) (la preuve pour MISEAUN(k) étant similaire).

Question 3 (2/9) — Soit $A(n)$ le nombre de nœuds dans l'arbre des appels récursifs de MiseAZero(n) ou MiseAUn(n) (par symétrie, le nombre est le même dans les deux cas). Donner l'équation de récurrence que vérifie $A(n)$. En déduire la complexité de MiseAZero(n).

Question 4 (3/9) — Une séquence de permutations autorisées est dite *élémentaire* si elle ne comporte pas deux permutations successives dont l'une est l'inverse de la précédente (par exemple, $111 \to 011 \to 111$ n'est pas élémentaire). L'objet de cette question est de montrer que la séquence produite par MISEAZERO(n) est en fait la *seule* séquence élémentaire possible pour transformer une chaîne de n bits 1 en une chaîne de n bits 0 en utilisant uniquement des permutations autorisées. Pour cela, on va s'aider d'une modélisation par un graphe et de la propriété suivante (admise) :

Propriété.
Soit G un graphe non-orienté. Les composantes connexes de G sont uniquement des chaînes et/ou des cycles si et seulement si les degrés des sommets de G sont tous dans $\{1, 2\}$.

Indiquer quel graphe non-orienté considérer pour prouver l'unicité de la séquence élémentaire de permutations autorisées, c'est-à-dire à quoi correspond l'ensemble des sommets et sous quelle condition une arête existe entre deux sommets. Justifier que le graphe est non-orienté. Expliquer pourquoi les degrés des sommets sont tous dans $\{1, 2\}$. Identifier les deux seuls sommets de degré 1 dans ce graphe, et en déduire le résultat recherché en vous appuyant sur le résultat de la question 2.