ÉCOLE POLYTECHNIQUE MONTRÉAL

Le génie sans frontières

Corrigé
Examen final

INF2010

Sigle du cours

Identification de l'étudiant(e)		
Nom :	**Prénom :**	
Signature :	**Matricule :**	**Groupe :**

Sigle et titre du cours	*Groupe*	*Trimestre*
INF2010 – Structures de données et algorithmes	**Tous**	**20091**
Professeur	***Local***	***Téléphone***
Ettore Merlo, responsable – Tarek Ould Bachir, chargé	**A-533**	**5758 - 5193**

Jour	*Date*	*Durée*	*Heures*
Jeudi	**25 avril 2009**	**2h30**	**9h30 – 12h00**

Documentation	*Calculatrice*	
☐ Toute ☒ Aucune ☐ Voir directives particulières	☐ Aucune ☐ Programmable ☒ Non programmable	**Les cellulaires, agendas électroniques ou téléavertisseurs sont interdits.**

Directives particulières

Bonne chance à tous!

Important

Cet examen contient **5** questions sur un total de **18** pages **(excluant cette page)**

La pondération de cet examen est de **40** **%**

Vous devez répondre sur : ☒ le questionnaire ☐ le cahier ☐ les deux

Vous devez remettre le questionnaire : ☒ oui ☐ non

L'étudiant doit honorer l'engagement pris lors de la signature du code de conduite.

Question 1 : Tri par monceau (20 points)

a) On désire étudier le comportement du tri par monceau.

a.1) **(3 pts)** Donner la complexité algorithmique du tri par monceau (fonction du nombre d'éléments *n*) pour chacun des cas énumérés ci-dessous :

i) pire cas : **O(*n* log(*n*))**

ii) cas moyen : **O(*n* log(*n*))**

iii) meilleur cas : **O(*n* log(*n*))**

a.2) **(3 pts)** Comparer les complexités du tri par monceau données dans ***a.1)*** à celles du tri par insertion et celle de QuickSort pour chacun des cas énumérés ci-dessous :

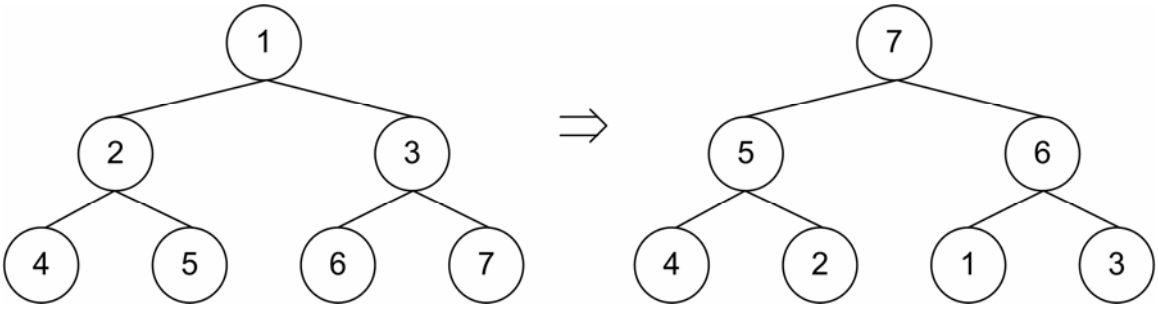
i) pire cas : **QuickSort : O(n^2), Tri par insertion O(n^2) : Le tri par monceau est meilleur en pire cas.**

ii) cas moyen : **QuickSort : O(*n* log), Tri par insertion O(n^2) : Le tri par monceau est aussi bon que QuickSort et meilleur que le Tri par insertion.**

iii) meilleur cas : **QuickSort : O(*n* log), Tri par insertion O(*n*) : Le tri par monceau est aussi bon que QuickSort mais moins bon que le Tri par insertion.**

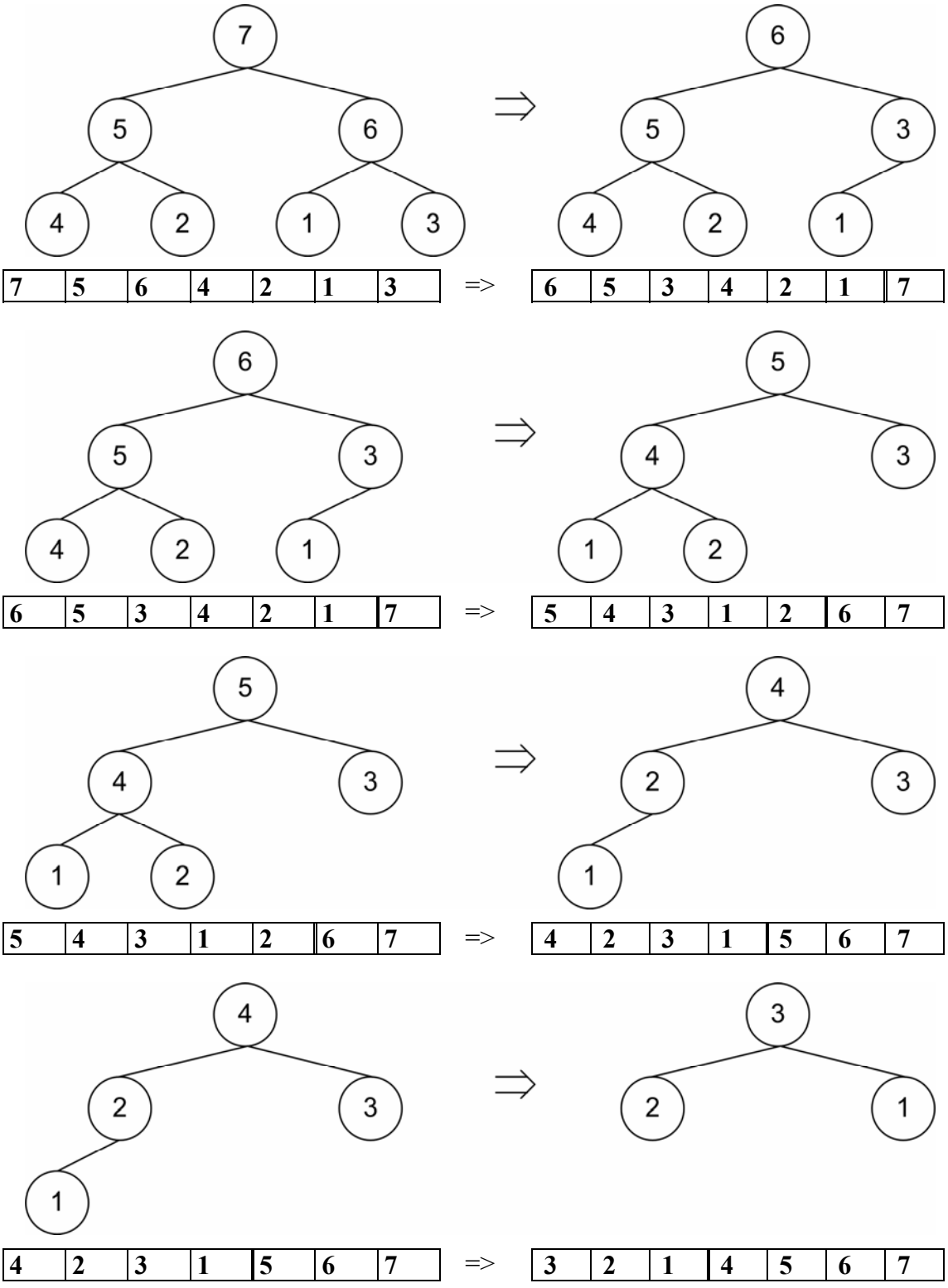
a.3) En partant du vecteur préalablement trié suivant V = {1, 2, 3, 4, 5, 6, 7}, on vous demande de :

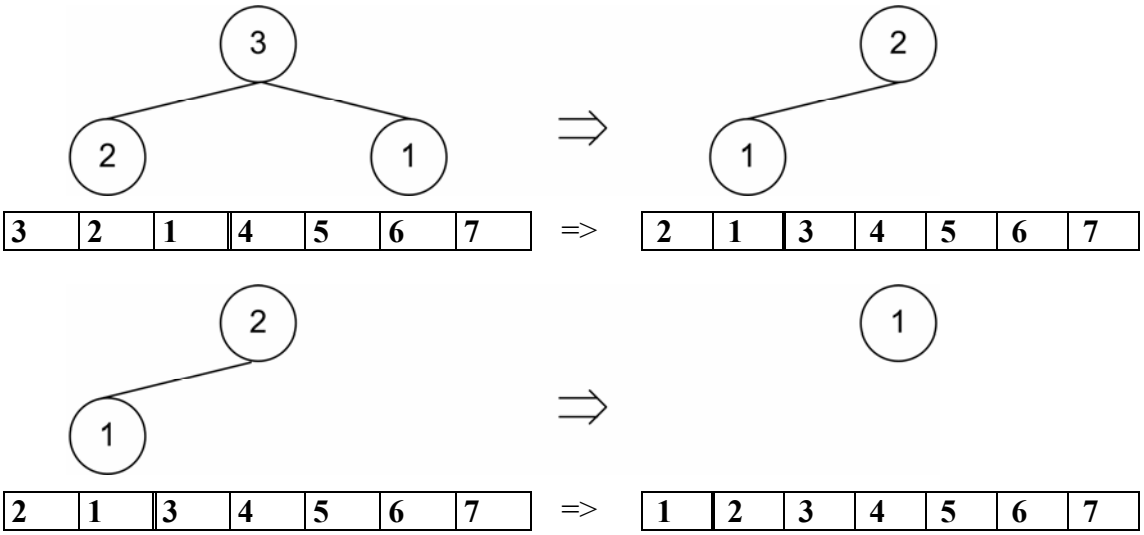
i) **(5 pts)** Construire un monceau max (indiquer l'état du vecteur après l'opération);



1	2	3	4	5	6	7	=>	**7**	**5**	**6**	**4**	**2**	**1**	**3**

ii) **(6 pts)** En partant du monceau max obtenu au point i), illustrer ci-dessous chacune des étapes du tri par monceau (indiquer l'étant du vecteur avant et après l'opération);





iii) **(3 pts)** Commenter le comportement de l'algorithme. Y a-t-il place à amélioration du tri par monceau selon vous ?

Tout comme le Merge Sort, Le tri par monceau ne fait aucune distinction entre les différents cas de figure d'entrée et maintient une complexité constante de O(*n* log(*n*)). Ainsi, si l'entrée est déjà triée ou partiellement trié, le tri par monceau aura tendance à réordonner le vecteur pour en faire un monceau sans exploiter les propriétés du vecteur d'entrée.

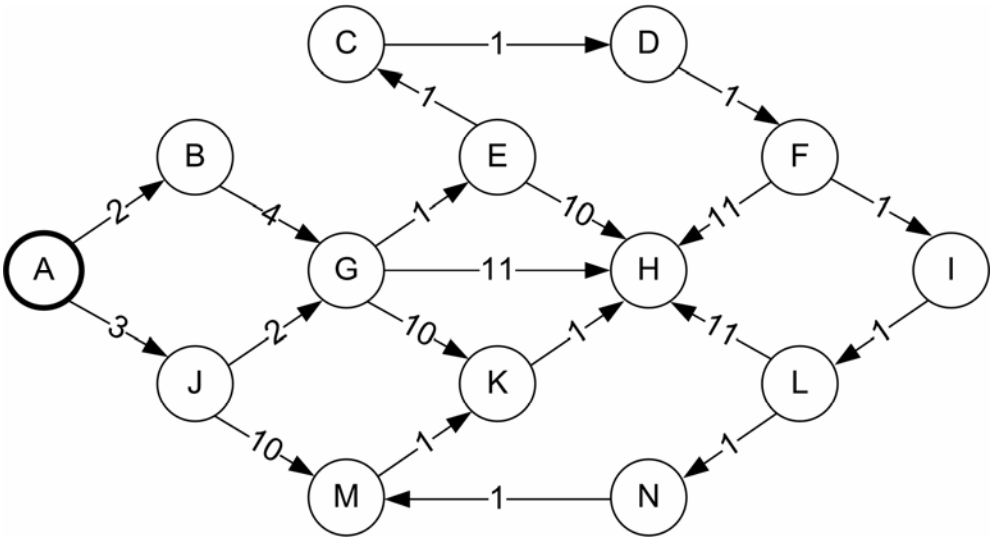
Une amélioration possible serait de vérifier si le vecteur est déjà trié et de le retourner tel quel, de sorte à avoir une complexité en O(*n*) en meilleure cas (comme le Tri par insertion). Étant donnée la faible probabilité d'un tel cas, cette amélioration serait inutile.

Une autre amélioration serait d'utiliser un monceau Min et de rajouter une contrainte sur sa construction de sorte que le vecteur soit trié (fils de droite plus grand que fils de gauche). Mais cette amélioration serait difficile à implémenter.

Question 2 : Plus court chemin d'un graphe acyclique (20 points)

a) Tri topologique

a.1) **(7 pts)** Pour le graphe dirigé suivant, numéroter les nœuds de **A** à **N** de sorte que, s'il existe un chemin du nœud **i** au nœud **j**, alors **i**<**j**.



Noeud	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
B	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
C	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
D	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
E	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
F	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
G	**2**	**2**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
H	**5**	**5**	**5**	**5**	**4**	**3**	**3**	**3**	**2**	**2**	**1**	**1**	**1**	**0**
I	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
J	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
K	**2**	**2**	**2**	**2**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**
L	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
M	**2**	**2**	**2**	**2**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**
N	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**
Entrée	**A**	**B,J**		**G**	**E**	**C**	**D**	**F**	**I**	**L**	**N**	**M**	**K**	**H**
Sortie	**A**	**B**	**J**	**G**	**E**	**C**	**D**	**F**	**I**	**L**	**N**	**M**	**K**	**H**

a.2) **(3 pts)** Peut-on dire de ce graphe qu'il est acyclique? Pourquoi?

Oui puisque la numération (le tri topologique) a été rendue possible.

b) Nous voulons trouver les plus courts chemins depuis le nœud A jusqu'à l'ensemble des nœuds B à N en apliquant l'algorithme de Dijkstra.

b.1) **(6 pts)** Continuez l'exécution de l'algorithme de Dijkstra utilisant une file de priorité (ici est partiellement réalisé), pour trouvez la longueur du plus court chemin menant à chacun des nœuds du graphe en partant de A.

Nœud	Connu	Dist min.	Parent
A	√	**0**	
B	√	~~∞~~**, 2**	**A**
C	√	∞**, 7**	**E**
D	√	∞**, 8**	**C**
E	√	∞**, 6**	**G**
F	√	∞**, 9**	**D**
G	√	∞**, 6, 5**	**B**
H	√	∞**, 16, 15**	**G, K**
I	√	∞**, 10**	**F**
J	√	∞**, 3**	**A**
K	√	∞**, 15, 14**	**G, M**
L	√	∞**, 11**	**I**
M	√	∞**, 13**	**J**
N	√	∞**, 12**	**L**

File de priorité

***b.2)* (2 pnt)** Détaillez chacun des chemins les plus courts trouvés :

Nœud	Le plus court chemin	Distance parcourue
B	**A → B**	**2**
C	**A → B → G → E → C**	**7**
D	**A → B → G → E → C → D**	**8**
E	**A → B → G → E**	**6**
F	**A → B → G → E → C → D → F**	**9**
G	**A → B → G**	**5**
H	**A → J → M → K → H**	**15**
I	**A → B → G → E → C → D → F → I**	**10**
J	**A → J**	**3**
K	**A → J → M → K**	**14**
L	**A → B → G → E → C → D → F → I → L**	**11**
M	**A → J → M**	**13**
N	**A → B → G → E → C → D → F → I → L → N**	**12**

***b.3)* (2 pts)** Comparer le chemin A → J → G → E → C → D → F → I → L → N → M avec le chemin allant de A à M que vous avez trouvé pour discuter le comportement de l'algorithme de Dijkstra.

Les chemin le plus court allant de A à M, trouvé grâce à l'algorithme, passe uniquement par J et a une longueur de 13. Le chemin A → J → G → E → C → D → F → I → L → N → M a également une longueur de 13. La raison pour laquelle Dijsktra a choisi ce chemin et non l'autre dépend entièrement de l'ordre d'entrée dans la file de priorité.

Question 3 : Rabin-Karp (20 points)

L'algorithme Rabin Karp est un algorithme permettant de retrouver une chaîne de caractères dans un texte. La chaîne de caractères recherchée est alors remplacée par un nombre qu'il faut pré-calculer et on compare toutes les valeurs des sous-séquences du texte à la valeur pré-calculée. Les différentes implémentations (**code java**) discutées dans cette question vous sont données â l'**annexe I**.

Dans ces implémentations de Rabin-Karp, le texte et le patron ne contiennent que les caractères alphabétiques 'a'-'z', 'A'-'Z' et l'espace ' ' dans l'encodage ASCII. Les valeurs respectives de ces caractères sont données ci-dessous :

Caractère	Val. ASCII	Caractère	Val. ASCII
'A'	65	'a'	97
'B'	66	'b'	98
…		…	
'Y'	89	'y'	121
'Z'	90	'z'	122
espace	32	-	-

On vous propose la méthode de calcul suivante :

Polynôme modulaire dans la base d = 10 modulo 11

Par exemple, P = 'Anna' : Code : 65, 110, 110, 97

p = (((((65%11) * 10 + 110) % 11) * 10 + 110) % 11)*10+97)%11
= (((((10) * 10 + 110) % 11) * 10 + 110) % 11)*10+97)%11
= (((((210) % 11) * 10 + 110) % 11)*10+97)%11
= (((1 * 10 + 110) % 11)*10+97)%11
= (((120) % 11)*10+97)%11
= ((10)*10+97)%11
= (197)%11
= 10

b) **(4 pnt)** Donnez la valeur de p pour le patron
P = 'merlo' : Codes ASCII : 109, 101, 114, 108, 111

p = (109*10 000 + 101*1 000 + 114*100+108*10+111)%11
= 1 203 591 % 11
= 4

c) **(16 pnt)** Retrouvez tous les décalages donnant la présence de P dans le texte
T= « Un amerloque se divertit »
Codes ASCII :

U	n		a	m	e	r	l	o	q	u	e		s	e		d	i	v	e	r	t	i	t
85	110	32	97	109	101	114	108	111	113	117	101	32	115	101	32	100	105	118	101	114	116	105	116

Décalage (s)	Sous-chaîne	Valeur du polynôme	Égalité?	Faux positif?	Correspondance?
0	‘Un am’	**8**			
1	‘n ame’	**2**			
2	‘ amer’	**2**			
3	‘amerl’	**6**			
4	‘merlo’	**4**	√		√
5	‘erloq’	**9**			
6	‘rloqu’	**0**			
7	‘loque’	**6**			
8	‘oque ’	**2**			
9	‘que s’	**4**	√	√	
10	‘ue se’	**1**			
11	‘e se ’	**5**			
12	‘ se d’	**9**			
13	‘se di’	**7**			
14	‘e div’	**6**			
15	‘ dive’	**9**			
16	‘diver’	**5**			
17	‘ivert’	**2**			
18	‘verti’	**10**			
19	‘ertit’	**4**	√	√	

Faux positifs trouvés :

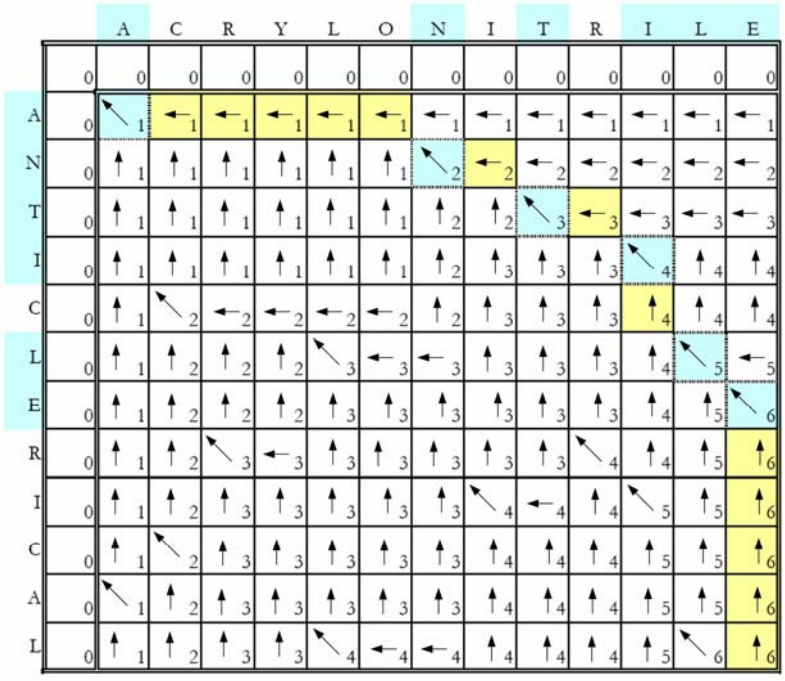
Aux décalages 9 et 19

Décalages retournés :

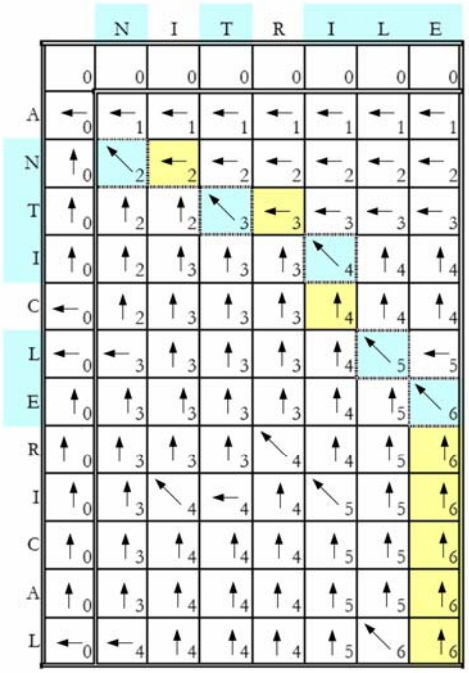
4

Question 4 : PLSC (20 points)

Partant de la table de *PLSC* pour les entrées ACRYLONITRILE et ANTICLERICAL suivante, donnant ANTILE pour solution au problème, on propose de :



a) Évaluer la *PLSC* pour les entrées NITRILE et ANTICLERICAL en tronquant la première table et de récupérer le résultat partiel NTILE :



***a.1)* (2 pts)** Est-ce que NTILE est une *PLSC* des entrées NITRILE et ANTICLERICAL ?

Oui car la *PLSC* des entrées NITRILE et ANTICLERICAL a une longue de 5.

***a.2)* (2 pts)** Discuter brièvement la validité du procédé de troncature dans ce cas de figure.

En aucun cas la troncature à l'avant des mots n'est une technique autorisée. Dans ce cas précis, elle fonctionne par un heureux hasard.

***a.3)* (6 pts)** Proposer une autre *PLSC* des entrées NITRILE et ANTICLERICAL si elle existe (vous pouvez vous servir de la table donnée à l'annexe II) :

NTRIL est une autre PLSC des entrées NITRILE et ANTICLERICAL.

b) Évaluer la *PLSC* pour les entrées ACRYLONITRILE et CLERICAL en tronquant la première table et de récupérer le résultat partiel LE :

		A	C	R	Y	L	O	N	I	T	R	I	L	E
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 4	↑ 4	↑ 4
L	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↑ 3	↑ 3	↑ 3	↑ 4	↖ 5	← 5
E	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3	↑ 3	↑ 3	↑ 4	↑ 5	↖ 6
R	0	↑ 1	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3	↑ 3	↖ 4	↑ 4	↑ 5	↑ 6
I	0	↑ 1	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3	↑ 3	↖ 4	← 4	↑ 4	↖ 5	↑ 5	↑ 6
C	0	↑ 1	↖ 2	↑ 3	↑ 3	↑ 3	↑ 3	↑ 3	↑ 4	↑ 4	↑ 4	↑ 5	↑ 5	↑ 6
A	0	↖ 1	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3	↑ 3	↑ 4	↑ 4	↑ 4	↑ 5	↑ 5	↑ 6
L	0	↑ 1	↑ 2	↑ 3	↑ 3	↖ 4	← 4	← 4	↑ 4	↑ 4	↑ 4	↑ 5	↖ 6	↑ 6

***b.1)* (2 pts)** Est-ce que LE est une *PLSC* des entrées ACRYLONITRILE et CLERICAL ?

Non car il existe une *PLSC* de longueur 5.

***b.2)* (2 pts)** Discuter brièvement la validité du procédé de troncature dans ce cas de figure.

La technique de troncature à l'avant n'est pas acceptable. Ceci est un bon exemple d'erreur potentielle qu'elle peut engendrer.

***b.3)* (6 pts)** Proposer une autre PLSC des entrées ACRYLONITRILE et CLERICAL si elle existe (vous pouvez vous servir de la table donnée à l'annexe III) :

ALRIL est une *PLSC* valide.

Question 5 : Ensemble disjoints (20 points)

Partant du théorème suivant :

> Tout ensemble S_n, défini comme l'ensemble des *n* premiers carrés entiers, admet une partition parfaite de quatre (4) ensemble disjoints (notés $S_{n,A}$, $S_{n,B}$, $S_{n,C}$, $S_{n,D}$) si et seulement si $n \geq 15$ et que $n \bmod 8 = 0$ ou $n \bmod 8 = 7$; cette partition est telle que chacune des sommes des éléments de ces ensembles disjoints sont égales.

Exemple :

> Pour n=15, $S_{15}=\{1^2, 2^2, 3^2, 4^2, 5^2, 6^2, 7^2, 8^2, 9^2, 10^2, 11^2, 12^2, 13^2, 14^2, 15^2\}$; nous avons une partition parfaite (elle est aussi unique) :
>
> $S_{15,A} = \{1^2, 7^2, 8^2, 14^2\}$
> $S_{15,B} = \{2^2, 9^2, 15^2\}$
> $S_{15,C} = \{3^2, 6^2, 11^2, 12^2\}$
> $S_{15,D} = \{4^2, 5^2, 10^2, 13^2\}$
>
> Où la somme des éléments de chaque ensemble disjoint est 310.

a) (2 pts) Peut-on trouver une partition de 4 ensembles disjoints à somme égale pour les ensembles S_n suivants :

S*n*	Partition existe	Partition n'existe pas
S_{15}	√	
S_{16}	√	
S_{23}	√	
S_{27}		√
S_{33}		√
S_{48}	√	

Justifiez vos réponses :

Tous les *n* proposés sont supérieurs à 15. Aussi, nous avons :

23 mod 8 = 7 √
27 mod 8 = 3 X
33 mod 8 = 1 X
48 mod 8 = 0 √

b) Sachant que :

L'ensemble des 32 carrés consécutifs $C_i=S_{i+31}-S_{i-1}=\{i^2, (i+1)^2, \ldots, (i+31)^2\}$ ($i \geq 1$) admet une partition parfaite unique :

$C_{i,A} = \{ i^2 , (i+7)^2, (i+11)^2, (i+12)^2, (i+18)^2, (i+21)^2, (i+25)^2, (i+30)^2\}$
$C_{i,B} = \{(i+1)^2, (i+6)^2, (i+10)^2, (i+13)^2, (i+19)^2, (i+20)^2, (i+24)^2, (i+31)^2\}$
$C_{i,C} = \{(i+2)^2, (i+5)^2, (i+ 9)^2, (i+14)^2, (i+16)^2, (i+23)^2, (i+27)^2, (i+28)^2\}$
$C_{i,D} = \{(i+3)^2, (i+4)^2, (i+ 8)^2, (i+15)^2, (i+17)^2, (i+22)^2, (i+26)^2, (i+29)^2\}$

Où la somme des éléments de chaque ensemble disjoint est $8i^2+248i+2604$.

b.1) **(2 pts)** Proposer une partition parfaite pour S_{32} :

	Éléments							
$S_{32,A}$	$\mathbf{1^2}$	$\mathbf{8^2}$	$\mathbf{12^2}$	$\mathbf{13^2}$	$\mathbf{19^2}$	$\mathbf{22^2}$	$\mathbf{26^2}$	$\mathbf{31^2}$
$S_{32,B}$	$\mathbf{2^2}$	$\mathbf{7^2}$	$\mathbf{11^2}$	$\mathbf{14^2}$	$\mathbf{20^2}$	$\mathbf{21^2}$	$\mathbf{25^2}$	$\mathbf{32^2}$
$S_{32,C}$	$\mathbf{3^2}$	$\mathbf{6^2}$	$\mathbf{10^2}$	$\mathbf{15^2}$	$\mathbf{17^2}$	$\mathbf{24^2}$	$\mathbf{28^2}$	$\mathbf{29^2}$
$S_{32,D}$	$\mathbf{4^2}$	$\mathbf{5^2}$	$\mathbf{9^2}$	$\mathbf{16^2}$	$\mathbf{18^2}$	$\mathbf{23^2}$	$\mathbf{27^2}$	$\mathbf{30^2}$

b.2) **(3 pts)** Combien existe-t-il de partitions parfaites pour S_{47}? Justifiez par un calcul.

Indice : 47=15+32

Les partitions parfaites pour S47 sont obtenues en combinant les $S_{15,A}$, $S_{15,B}$, $S_{15,C}$, $S_{15,D}$ et $C_{16,A}$, $C_{16,B}$, $C_{16,C}$, $C_{16,D}$. Il existe donc 4 * 3 * 2 * 1 combinaisons possibles, pour un total de 24.

b.3) **(3 pts)** En vous servant du raisonnement utilisé pour ***b.2)***, proposer une partition parfaite pour S_{47} :

	Éléments											
$S_{47,A}$	$\mathbf{1^2}$	$\mathbf{7^2}$	$\mathbf{8^2}$	$\mathbf{14^2}$	$\mathbf{16^2}$	$\mathbf{23^2}$	$\mathbf{27^2}$	$\mathbf{28^2}$	$\mathbf{34^2}$	$\mathbf{37^2}$	$\mathbf{41^2}$	$\mathbf{46^2}$
$S_{47,B}$	$\mathbf{2^2}$	$\mathbf{9^2}$	$\mathbf{15^2}$	$\mathbf{17^2}$	$\mathbf{22^2}$	$\mathbf{26^2}$	$\mathbf{29^2}$	$\mathbf{35^2}$	$\mathbf{36^2}$	$\mathbf{40^2}$	$\mathbf{47^2}$	
$S_{47,C}$	$\mathbf{3^2}$	$\mathbf{6^2}$	$\mathbf{11^2}$	$\mathbf{12^2}$	$\mathbf{18^2}$	$\mathbf{21^2}$	$\mathbf{25^2}$	$\mathbf{30^2}$	$\mathbf{32^2}$	$\mathbf{39^2}$	$\mathbf{43^2}$	$\mathbf{44^2}$
$S_{47,D}$	$\mathbf{4^2}$	$\mathbf{5^2}$	$\mathbf{10^2}$	$\mathbf{13^2}$	$\mathbf{19^2}$	$\mathbf{20^2}$	$\mathbf{24^2}$	$\mathbf{31^2}$	$\mathbf{33^2}$	$\mathbf{38^2}$	$\mathbf{42^2}$	$\mathbf{45^2}$

c) On vous demande d'exécuter la structure d'ensembles disjoints utilisant la compression du chemin *et* la métrique des rangs[1] pour générer la partition de S_{32} (***b.1***).

***c.1)* (4 pts)** Dessiner la forêt d'arbres (indiquer le rang de chaque arbre) obtenue à la fin des appels suivants (pseudo-code java) :

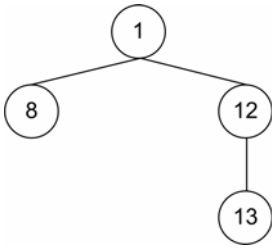
```
// A la fin de cet appel, nous avons 32 arbres
for(int i=1; i ≤ 32; i++)
      Create(i²);

// A la fin de ces appels, nous avons 16 arbres
Union( 1², 8²); Union(12²,13²); Union(19²,22²); Union(26²,31²);
Union( 2², 7²); Union(11²,14²); Union(20²,21²); Union(25²,32²);
Union( 3², 6²); Union(10²,15²); Union(17²,24²); Union(28²,29²);
Union( 4², 5²); Union( 9²,16²); Union(18²,23²); Union(27²,30²);

// A la fin de ces appels, nous avons 8 arbres
Union( 1²,12²); Union(19²,26²); Union(11²,20²); Union(20²,25²);
Union( 3²,17²); Union( 3²,28²); Union( 4², 9²); Union(18²,27²);
```

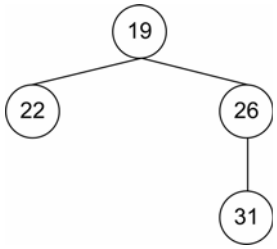
N.B.: Vous pouvez omettre le carré dans les nœuds et représenter le nœud i^2 par i

ARBRE 1



Rang : **3**

ARBRE 2

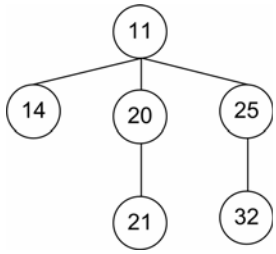


Rang : **3**

ARBRE 3

Rang : **2**

ARBRE 4

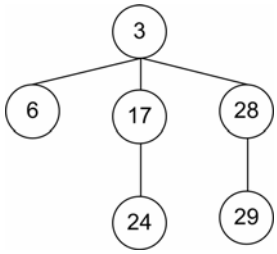


Rang : **3**

ARBRE 5

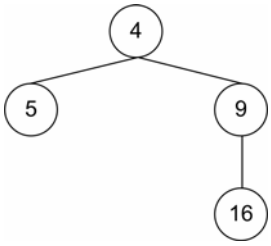
Rang : **2**

ARBRE 6



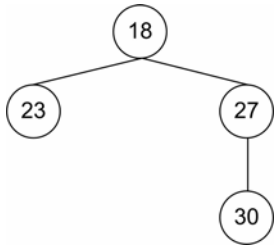
Rang : **3**

ARBRE 7



Rang : **3**

ARBRE 8



Rang : **3**

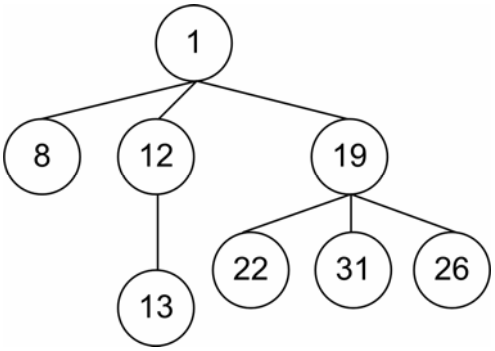
[1] **Rappel :** i) Lors de l'union de deux arbres de rangs différents, l'arbre de rang inférieur devient l'enfant de l'arbre de rang supérieur. Le nouveau rang est le plus grand des deux rangs.
ii) La racine de l'arbre issu de l'union de deux arbres de même rang est la racine ayant la plus petite valeur parmi les deux arbres. Le nouveau rang est celui des deux arbres incrémenté de 1.

c.2)* (4 pts)** En partant des résultats de la question ***c.1), dessiner la forêt d'arbres (indiquer le rang de chaque arbre) obtenue à la fin des appels :

```
// A la fin de ces appels, nous avons 4 arbres
Union( 1^2,31^2); Union(7^2, 32^2); Union(15^2,24^2); Union( 4^2,30^2);
```

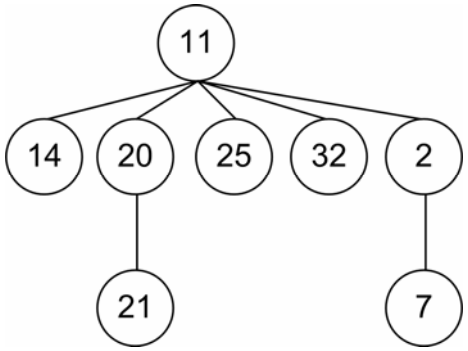
N.B.: Vous pouvez omettre le carré dans les nœuds et représenter le nœud i^2 par i

ARBRE 1



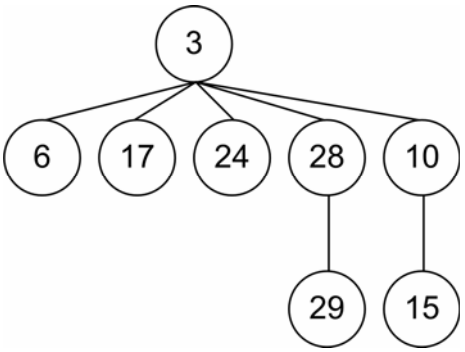
Rang : **4**

ARBRE 2



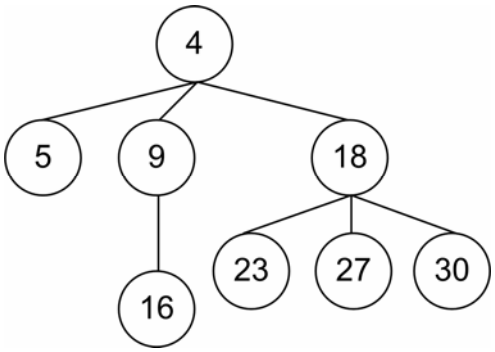
Rang : **3**

ARBRE 3



Rang : **3**

ARBRE 4



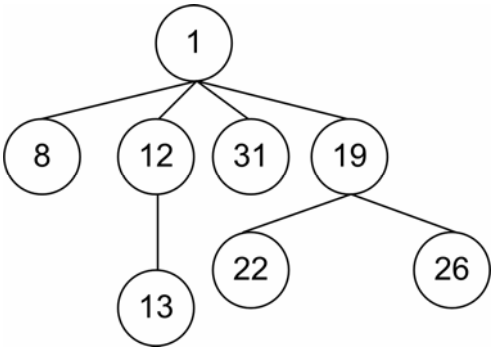
Rang : **4**

c.3)* (2 pts)** En partant des résultats de la question ***c.2), dessiner la forêt d'arbres obtenue à la fin des appels :

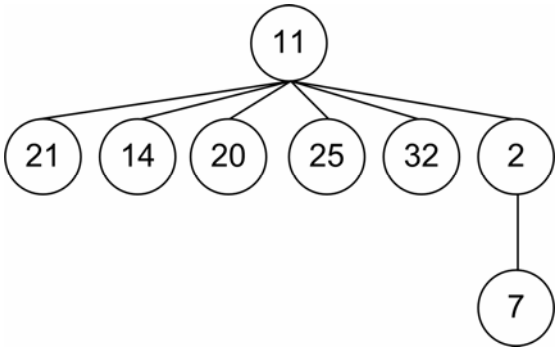
```
// A la fin de ces appels, nous avons 4 arbres
Find(31²); Find(21²); Find(14²); Find(29²), Find(30²);
```

N.B.: Vous pouvez omettre le carré dans les nœuds et représenter le nœud i^2 par i

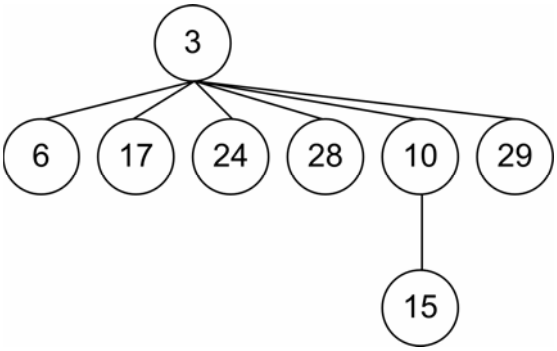
ARBRE 1



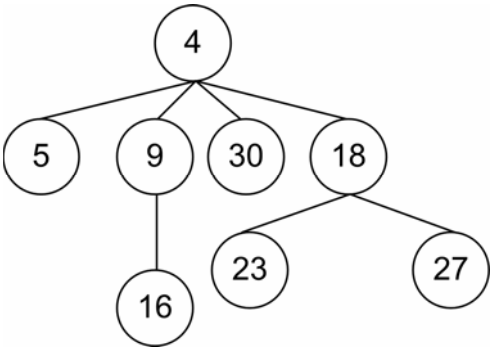
ARBRE 2



ARBRE 3



ARBRE 4



Annexe I

```
import java.util.ArrayList;

public class RabinKarp {

	public static ArrayList<Integer>
	RabinKarpFind(String Text, String Pattern)
	{
		ArrayList<Integer> decalages = new ArrayList<Integer>();

		if( Text.length() < Pattern.length() )
			return decalages;

		int p = ComputePatternValue( Pattern );

		for(int i=0; i <= Text.length() - Pattern.length(); i++)
		{
			int t = ComputePatternValue(Text.substring(i, i+Pattern.length()));

			if( t == p )
			{
				int j;
				for(j=0; j< Pattern.length(); j++)
				{
					if( Pattern.charAt( j ) != Text.charAt( i +j ) )
						break;
				}

				if(j == Pattern.length())
				{
					decalages.add( i );
					System.out.println("Correspodance à " + i);
				}
				else
					System.out.println("Faux positif à " + i);
			}
		}

		return decalages;
	}

	public static int ComputePatternValue(String Pattern)
	{
		int p = 0;

		for(int i=0; i<Pattern.length(); i++)
		{
			p *=10;
			p += (int) Pattern.charAt( i );
			p %= 11;
		}

		return p;
	}

	public static void main(String[] args)
	{
		String Pattern = "merlo";
		int p = ComputePatternValue(Pattern);
		System.out.println( p );

		String Text = "Un amerloque se divertit";
		System.out.println( Text.length() );
		ArrayList<Integer> decalages = RabinKarpFind(Text, Pattern);

		for(int s : decalages )
			System.out.print( s + "\t" );
	}
}
```

Annexe II

a) *PLSC* de NITRILE et ANTICLERICAL

		N	I	T	R	I	L	E
	0	0	0	0	0	0	0	0
A	0							
N	0							
T	0							
I	0							
C	0							
L	0							
E	0							
R	0							
I	0							
C	0							
A	0							
L	0							

Annexe III

b) *PLSC* de ACRYLONITRILE et CLERICAL :

		A	C	R	Y	L	O	N	I	T	R	I	L	E
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0													
L	0													
E	0													
R	0													
I	0													
C	0													
A	0													
L	0													