

**Corrigé
examen final**


INF2010

Sigle du cours

<i>Identification de l'étudiant(e)</i>		
Nom :	Prénom :	
Signature :	Matricule :	Groupe :

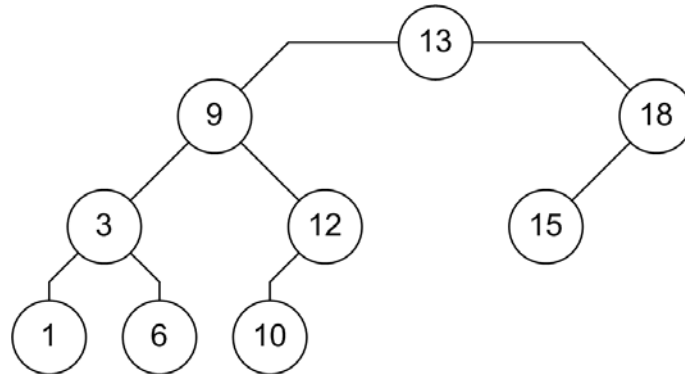
<i>Sigle et titre du cours</i>		<i>Groupe</i>	<i>Trimestre</i>
INF2010 – Structures de données et algorithmes		Tous	20101
<i>Professeur</i>		<i>Local</i>	<i>Téléphone</i>
Ettore Merlo – responsable / Tarek Ould Bachir - chargé		B-512	
<i>Jour</i>	<i>Date</i>	<i>Durée</i>	<i>Heures</i>
Jeudi	22 avril 2009	2h30	13h30-16h00

<i>Documentation</i>	<i>Calculatrice</i>	
<input checked="" type="checkbox"/> Aucune <input type="checkbox"/> Toute <input checked="" type="checkbox"/> Voir directives particulières	<input type="checkbox"/> Aucune <input type="checkbox"/> Toutes <input checked="" type="checkbox"/> Non programmable	Les cellulaires, agendas électroniques ou téléavertisseurs sont interdits.

<i>Directives particulières</i>
<p> Un cahier supplémentaire vous sera remis. Servez-vous de ce cahier comme brouillon. Nous ne récupérerons pas le cahier supplémentaire à la fin de l'examen. Pas conséquent, <u>écrivez toutes vos réponses dans ce questionnaire.</u></p> <p style="text-align: right;"><i>Bonne chance à tous!</i></p>

Important	<p>Cet examen contient <input type="text" value="5"/> questions sur un total de <input type="text" value="15"/> pages (excluant cette page)</p> <p>La pondération de cet examen est de <input type="text" value="40"/> %</p> <p>Vous devez répondre sur : <input checked="" type="checkbox"/> le questionnaire <input type="checkbox"/> le cahier <input type="checkbox"/> les deux</p> <p>Vous devez remettre le questionnaire : <input checked="" type="checkbox"/> oui <input type="checkbox"/> non</p>
------------------	--

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

Question 1 : Arbre Splay**(20 points)****1.1) (6 pts)** Considérez l'arbre suivant et répondez aux énoncés par vrai ou par faux.**1.1.1) (2 pts)** Ceci est un arbre complet :

Vrai	<input type="checkbox"/>
Faux	<input checked="" type="checkbox"/>

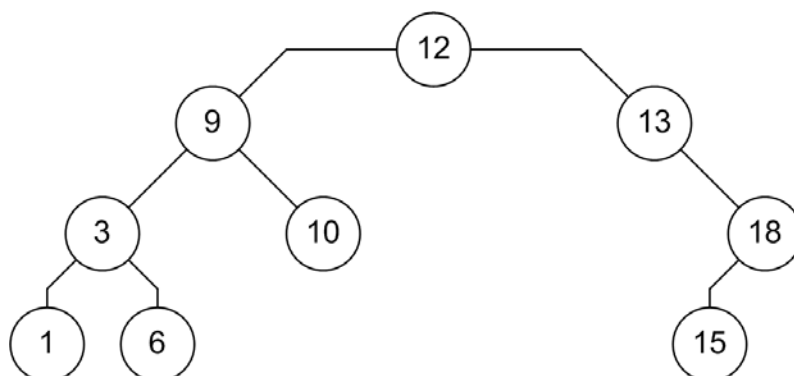
1.1.2) (2 pts) Ceci est un arbre de recherche :

Vrai	<input checked="" type="checkbox"/>
Faux	<input type="checkbox"/>

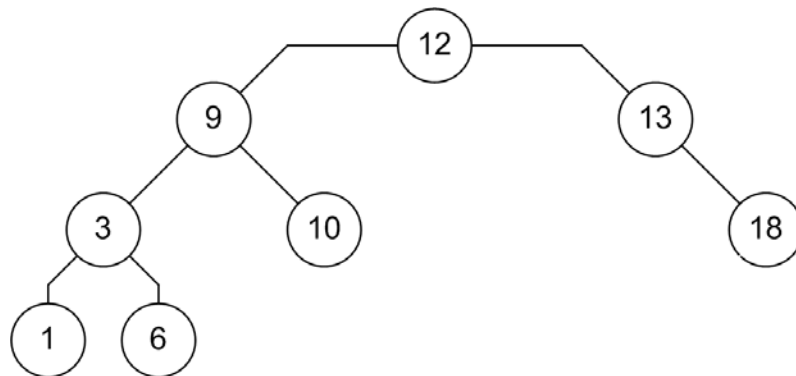
1.1.3) (2 pts) Ceci est un arbre AVL :

Vrai	<input checked="" type="checkbox"/>
Faux	<input type="checkbox"/>

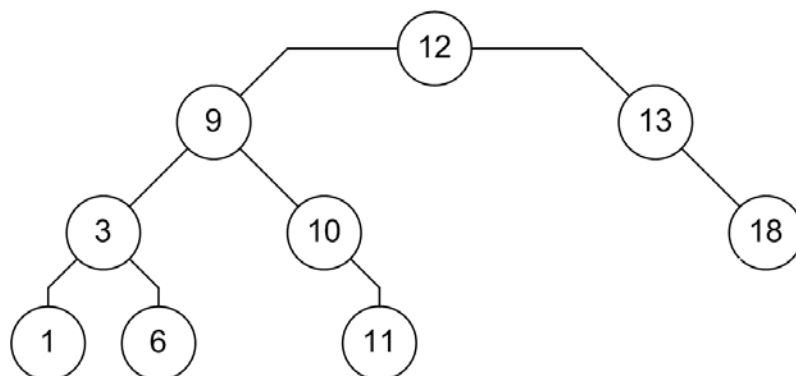
1.2) (14 pts) L'arbre donné à la question 1.1) est un arbre Splay sur lequel nous voulons exécuter (dans l'ordre) une série d'opérations. Dessinez l'état de l'arbre après chacune des opérations suivantes :

1.2.1) (2 pts) Get(12) :

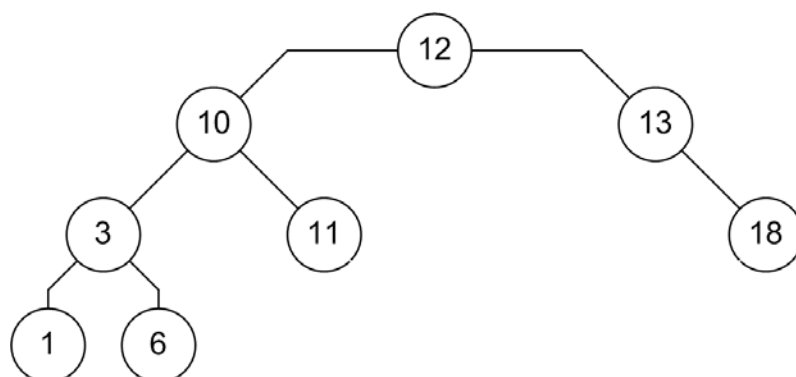
1.2.2) (2 pts) Delete(15) :



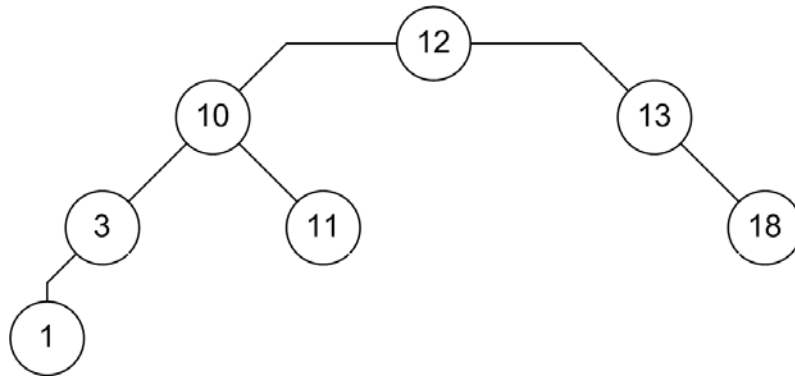
1.2.3) (2 pts) Insert(11) :



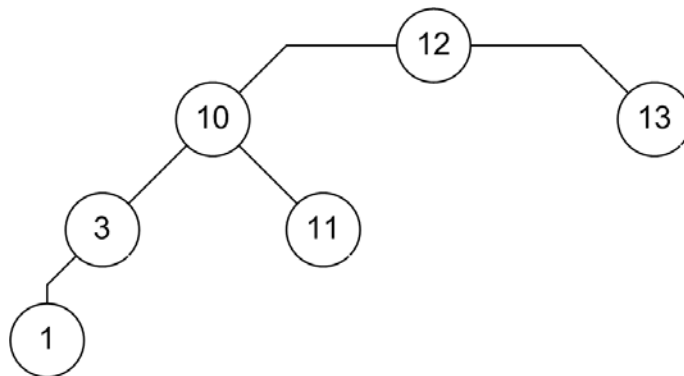
1.2.4) (2 pts) Delete(9) :



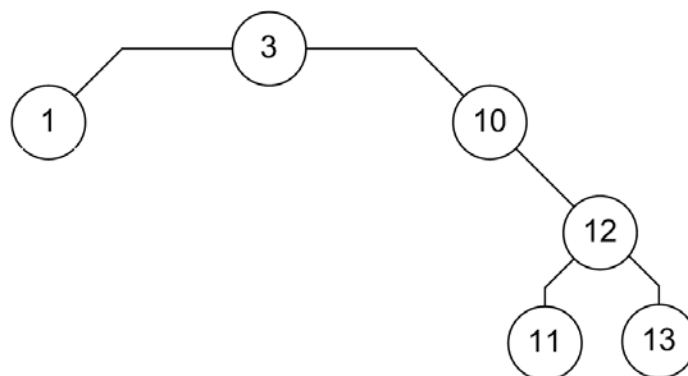
1.2.5) (2 pts) Delete(6) :



1.2.6) (2 pts) Delete(18) :

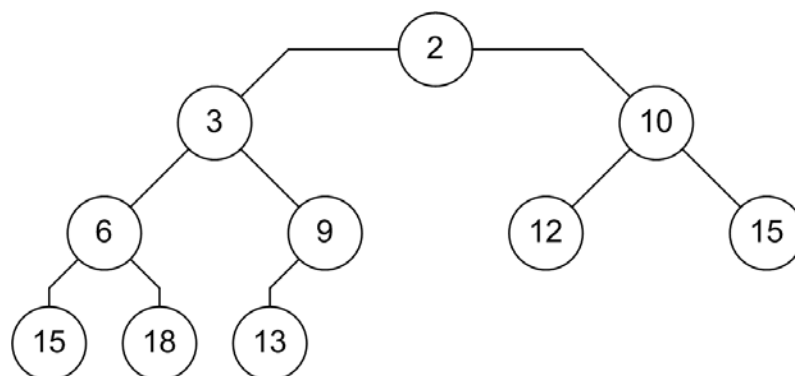


1.2.7) (2 pts) Get(3) :



Question 2 : Monceau**(25 points)**

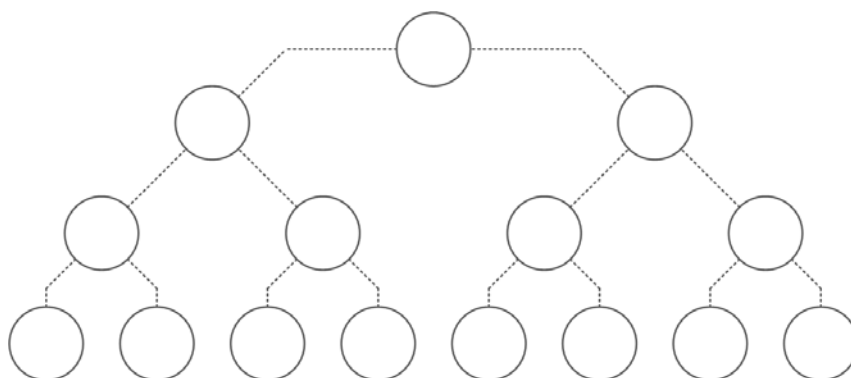
2.1) (11 pts) Considérez le monceau suivant sur lequel nous voulons exécuter (dans l'ordre) une série d'opérations. Dessinez l'état de l'arbre après chacune des opérations qui suivent. Dans chaque cas, dessinez l'état en mémoire du tableau contenant le monceau.



2.1.1) (1 pts) État du tableau de départ :

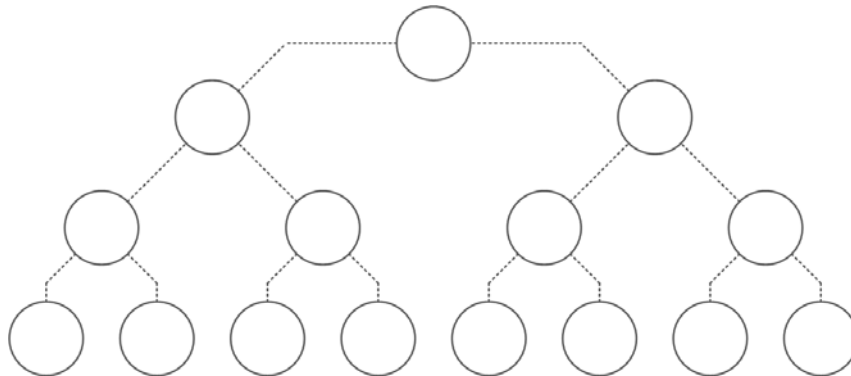
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Valeurs	X	2	3	10	6	9	12	15	15	18	13								

2.1.2) (2 pts) add(15) :



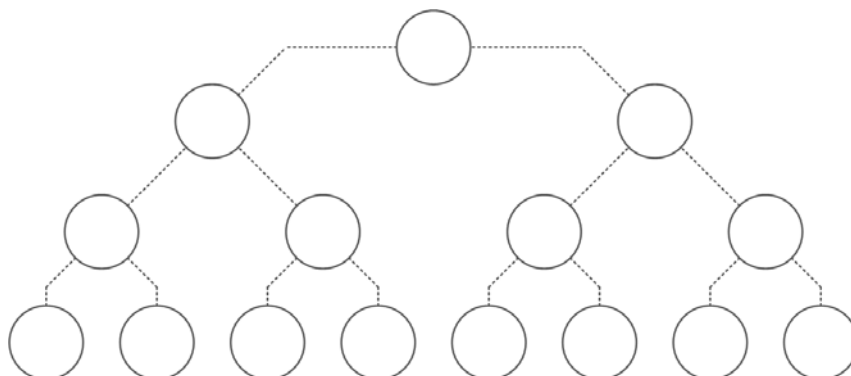
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Valeurs	X	2	3	10	6	9	12	15	15	18	13	15							

2.1.3) (2 pts) add(2) :



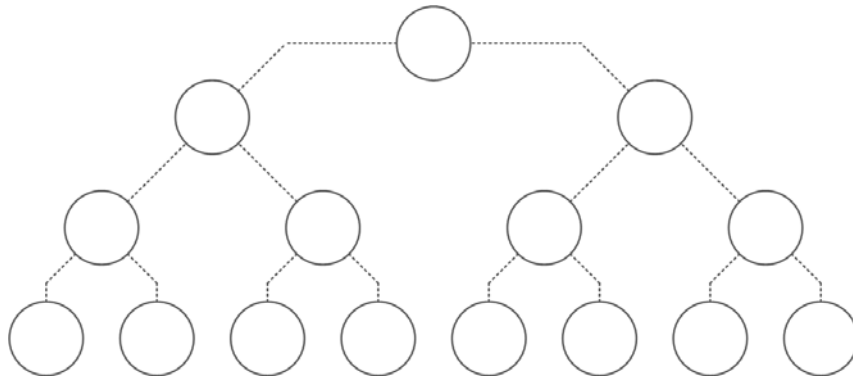
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Valeurs	X	2	3	2	6	9	10	15	15	18	13	15	12						

2.1.4) (2 pts) remove() :



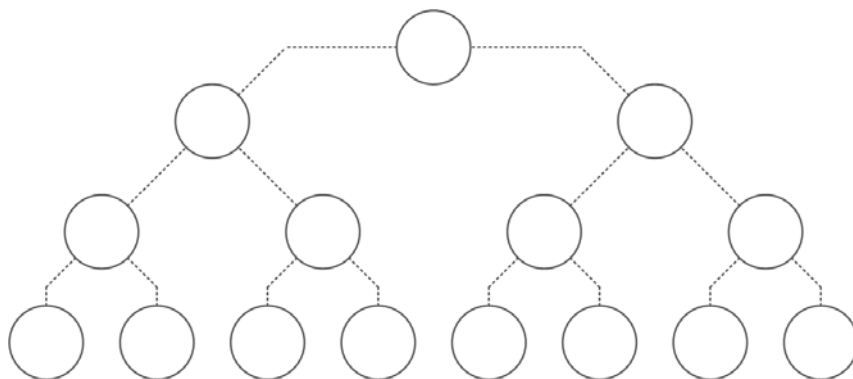
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Valeurs	X	2	3	10	6	9	12	15	15	18	13	15							

2.1.5) (2 pts) remove() :



Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Valeurs	X	3	6	10	15	9	12	15	15	18	13								

2.1.6) (2 pts) remove() :



Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Valeurs	X	6	9	10	15	13	12	15	15	18									

2.2) L'implémentation du monceau en Java vous est donnée à l'annexe I.

2.2.1) (7 pts) Écrivez le code en java d'une méthode permettant de trouver la valeur maximale du monceau.

```
int findMax( )
{
    int x = array[1];

    for( int i = currentSize/2; i <= currentSize; i++ )
    {
        if (array[i].compareTo (x) >0)
            x = array[i];
    }

    return x ;
}
```

2.2.2) (3 pts) Quelle est la complexité asymptotique de votre algorithme? Justifiez brièvement.

L'algorithme parcourt la moitié des éléments du monceau. La complexité asymptotique est donc $O(n)$.

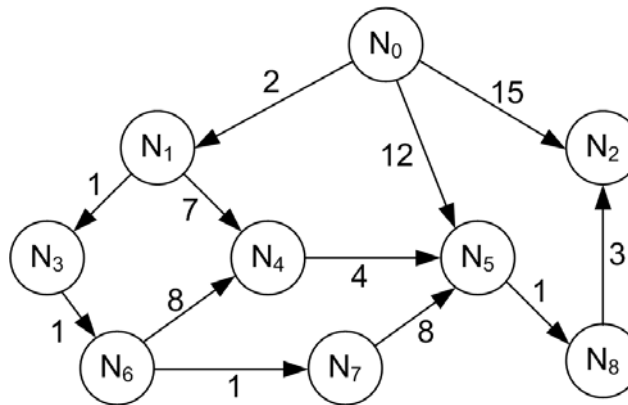
2.2.3) (4 pts) Peut-on améliorer la complexité asymptotique votre algorithme? Justifiez clairement votre réponse.

Indice : Pensez à l'algorithme du k -ième élément.

Le problème que nous avons ici est celui du k -ième élément appliqué sur les $n/2$ derniers éléments du monceau. Le k -ième élément étant résolu avec une complexité $O(n)$, on ne peut donc faire mieux que $O(n)$.

Question 3 : Graphes**(25 points)**

Soit le graphe suivant :

**3.1) (5 pts)** Répondez par vrai ou par faux à ces affirmations :**3.1.1) (1 pts)** Ce graphe n'est pas valué.

Vrai	<input type="checkbox"/>
Faux	<input checked="" type="checkbox"/>

3.1.2) (1 pts) Ce graphe n'admet pas d'ordre topologique.

Vrai	<input type="checkbox"/>
Faux	<input checked="" type="checkbox"/>

3.1.3) (1 pts) Ce graphe est dirigé.

Vrai	<input checked="" type="checkbox"/>
Faux	<input type="checkbox"/>

3.1.4) (1 pts) Ce graphe est dense.

Vrai	<input type="checkbox"/>
Faux	<input checked="" type="checkbox"/>

3.1.5) (1 pts) Ce graphe est cyclique.

Vrai	<input type="checkbox"/>
Faux	<input checked="" type="checkbox"/>

3.2.1) (7 pts) Exécutez l'algorithme de Dijkstra utilisant une file de priorité pour trouver la longueur du plus court chemin menant à chacun des nœuds du graphe en partant de N_0 .

Nœud	Connu	Dist min.	Parent
N_0	✓	0,	-
N_1	✓	∞ , 2	N_0
N_2	✓	∞ , 15	N_0
N_3	✓	∞ , 3	N_1
N_4	✓	∞ , 9	N_1
N_5	✓	∞ , 12	N_0
N_6	✓	∞ , 4	N_3
N_7	✓	∞ , 5	N_6
N_8	✓	∞ , 13	N_5

File de priorité

3.2.2) (3 pts) Détaillez chacun des chemins trouvés :

Nœud	Le plus court chemin	Distance parcourue
N_0	$N_0 \rightarrow N_0$	0
N_1	$N_0 \rightarrow N_1$	2
N_2	$N_0 \rightarrow N_2$	15
N_3	$N_0 \rightarrow N_1 \rightarrow N_3$	3
N_4	$N_0 \rightarrow N_1 \rightarrow N_4$	9
N_5	$N_0 \rightarrow N_5$	12
N_6	$N_0 \rightarrow N_1 \rightarrow N_3 \rightarrow N_6$	4
N_7	$N_0 \rightarrow N_1 \rightarrow N_3 \rightarrow N_6 \rightarrow N_7$	5
N_8	$N_0 \rightarrow N_5 \rightarrow N_8$	13

3.3) (10 pts) Numérotez de 1 à 9 les nœuds N_0 à N_8 du graphe 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
N_0	0	0	0	0	0	0	0	0	0
N_1	1	0	0	0	0	0	0	0	0
N_2	2	1	1	1	1	1	1	1	0
N_3	1	1	0	0	0	0	0	0	0
N_4	2	2	1	1	0	0	0	0	0
N_5	3	2	2	2	2	1	0	0	0
N_6	1	1	1	0	0	0	0	0	0
N_7	1	1	1	1	0	0	0	0	0
N_8	1	1	1	1	1	1	1	0	0
Entrée	N_0	N_1	N_3	N_6	N_4, N_7	-	N_5	N_8	N_2
Sortie	N_0	N_1	N_3	N_6	N_4	N_7	N_5	N_8	N_2

Question 4 : Rabin-Karp**(15 points)**

L'algorithme Rabin Karp est un algorithme permettant de retrouver une chaîne de caractères dans un texte. Supposons que les chaînes de caractères soient constituées des chiffres 0 à 9 auxquels on associe la valeur numérale de ces derniers (le chiffre 9 vaut 9).

4.1) (9 pts) Quel est le nombre de faux-positifs que produit l'exécution de l'algorithme Rabin-Karp sur la chaîne de caractères 32111125881 si l'on recherche le patron 111 en travaillant modulo $q=7$? Donnez le détail de vos calculs.

$321 \bmod 7 = 6, \surd \rightarrow$ faux positif ; $211 \bmod 7 = 1, \times$;

$111 \bmod 7 = 6, \surd \rightarrow$ vrai positif ; $111 \bmod 7 = 6, \surd \rightarrow$ vrai positif ;

$112 \bmod 7 = 0, \times$; $125 \bmod 7 = 6, \surd \rightarrow$ faux positif ;

$258 \bmod 7 = 6, \surd \rightarrow$ faux positif ; $588 \bmod 7 = 0, \times$;

$881 \bmod 7 = 6, \surd \rightarrow$ faux positif ;

Il y a donc quatre (4) faux positifs.

4.2) (3 pts) Donnez la valeur des sous-séquences correspondant aux faux positifs que vous avez trouvés dans 4.1).

321, 125, 258, 881

4.3) (3 pts) Rabin-Karp produit parfois des faux-positifs (candidats retenus ne remplissant pas le critère de correspondance). Expliquez pourquoi il ne génère pas de faux-négatifs (candidats rejetés remplissant le critère de correspondance).

Si un candidat remplit le critère de correspondance, la valeur qui lui sera associée est forcément celle recherché et il ne sera pas rejeté. Ainsi, il ne peut pas y avoir de faux négatifs.

Question 5 : Programmation dynamique**(15 points)**

On désire trouver le parenthésage idéal pour multiplier les matrices A_1 à A_6 permettant de minimiser le nombre de multiplications (scalaires) à effectuer. Les matrices sont dimensionnées comme suit :

$A_1 : 3 \times 6$; $A_2 : 6 \times 2$; $A_3 : 2 \times 8$; $A_4 : 8 \times 6$; $A_5 : 6 \times 1$; $A_6 : 1 \times 6$

5.1) (6 pts) Considérez les tables **m** et **s** obtenue par l'exécution de l'algorithme dynamique vu en cours.

m	1	2	3	4	5	6
1	0	36	84	168	94	112
2		0	96	168	76	112
3			0	96	64	76
4				0	48	96
5					0	36
6						0

s	1	2	3	4	5	6
1		1	2	2	1	5
2			2	2	2	5
3				3	3	5
4					4	5
5						5
6						

5.1.1) (2 pts) Donnez le parenthésage optimal pour multiplier A_1 à A_6 . Donnez son coût.

$A_1(A_2(A_3(A_4A_5)))A_6$ au coût de 112

5.1.2) (2 pts) Donnez le parenthésage optimal pour multiplier A_2 à A_4 . Donnez son coût.

$A_2(A_3A_4)$ au coût de 168

5.1.3) (2 pts) Donnez le parenthésage optimal pour multiplier A_1 à A_4 . Donnez son coût.

$(A_1A_2)(A_3A_4)$ au coût de 168

5.2) (9 pts) Considérez les tables **m** et **s** modifiées du résultat précédent et complétez-les de sorte à trouver le coût optimal de la multiplication des matrices A_1 à A_7 , sachant que A_7 a une taille de 6×12 . Donnez ce parenthésage optimal et son coût.

m	1	2	3	4	5	6	7
1	0	36	84	168	94	112	202
2		0	96	168	76	112	220
3			0	96	64	76	160
4				0	48	96	122
5					0	36	144
6						0	72
7							0

s	1	2	3	4	5	6	7
1		1	2	2	1	5	5
2			2	2	2	5	5
3				3	3	5	5
4					4	5	6
5						5	5
6							6

Rappel : $m[i, j] = \min \{m[i, k] + m[k+1, j] + p_{i-1}p_kp_j\}$ pour $k = i$ à $j-1$, sachant que la matrice A_i a une dimension $p_{i-1} \times p_i$.

Annexe I

```
public class BinaryHeap <AnyType> extends AbstractCollection<AnyType>
{
    /**
     * Crée un BinaryHeap vide
     */
    public BinaryHeap( )
    {
        currentSize = 0;
        cmp = null;
        array = (AnyType[]) new Object[ DEFAULT_CAPACITY + 1 ];
    }

    /**
     * Effectue la comparaison de deux objets de type AnyType
     */
    private int compare( AnyType lhs, AnyType rhs )
    {
        return ((Comparable)lhs).compareTo( rhs );
    }

    /**
     * Insère x
     */
    public boolean add( AnyType x )
    {
        if( currentSize + 1 == array.length )
            doubleArray( );

        // Percolate up
        int hole = ++currentSize;
        array[ hole ] = x;

        for( ; compare( x, array[ hole / 2 ] ) < 0; hole /= 2 )
            array[ hole ] = array[ hole / 2 ];
        array[ hole ] = x;

        return true;
    }

    /**
     * Donne le nombre d'éléments présents
     */
    public int size( )
    {
        return currentSize;
    }

    /**
     * Vide le monceau
     */
    public void clear( )
    {
        currentSize = 0;
    }

    /**
     * Retourne le plus petit element du monceau
     */
    public AnyType element( )
    {
        if( isEmpty( ) )
            throw new NoSuchElementException( );
        return array[ 1 ];
    }
}
```

```
/**
 * Retire le plus petit élément
 */
public AnyType remove( )
{
    AnyType minItem = element( );
    array[ 1 ] = array[ currentSize-- ];
    percolateDown( 1 );

    return minItem;
}

/**
 * Construit un monceau à partir des éléments présents
 */
private void buildHeap( )
{
    for( int i = currentSize / 2; i > 0; i-- )
        percolateDown( i );
}

private static final int DEFAULT_CAPACITY = 100;
private int currentSize;
private AnyType [ ] array;

/**
 * Méthode privée pour percoler vers le bas
 */
private void percolateDown( int hole )
{
    int child;
    AnyType tmp = array[ hole ];

    for( ; hole * 2 <= currentSize; hole = child )
    {
        child = hole * 2;
        if( child != currentSize &&
            compare( array[ child + 1 ], array[ child ] ) < 0 )
            child++;
        if( compare( array[ child ], tmp ) < 0 )
            array[ hole ] = array[ child ];
        else
            break;
    }
    array[ hole ] = tmp;
}

/**
 * Méthode privée pour doubler la taille de la mémoire
 */
private void doubleArray( )
{
    AnyType [ ] newArray;

    newArray = (AnyType [ ]) new Object[ array.length * 2 ];
    for( int i = 0; i < array.length; i++ )
        newArray[ i ] = array[ i ];
    array = newArray;
}
```