POLYTECHNIQUE MONTRÉAL

Questionnaire 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**	**20171**
Professeur	***Local***	***Téléphone***
Ettore Merlo, responsible Tarek Ould Bachir, chargé de cours	**B-638**	**5193**

Jour	*Date*	*Durée*	*Heures*
Vendredi	**05 mai 2017**	**2 h 30**	**9 h 30 – 12 h 00**

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

Directives particulières

- Un cahier supplémentaire vous sera remis. Servez-vous de ce cahier comme brouillon. Toutes vos réponses doivent être faites sur le questionnaire. Le cahier supplémentaire n'est pas à remettre à la fin de l'examen.

Important

Cet examen contient **6** questions sur un total de **17** 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 : Monceaux (20 points)

Pour cette question, référez-vous au code Java donné à l'Annexe 1.

1.1) (1 pts) Donnez la complexité asymptotique en cas moyen d'un `findMin()`.

1.2) (1 pts) Donnez la complexité asymptotique en cas moyen d'un `deleteMin()`.

1.3) (1 pts) Donnez la complexité asymptotique en cas moyen d'un `buildHeap()`.

1.4) (1 pts) Donnez la complexité asymptotique en pire cas d'un `buildHeap()`.

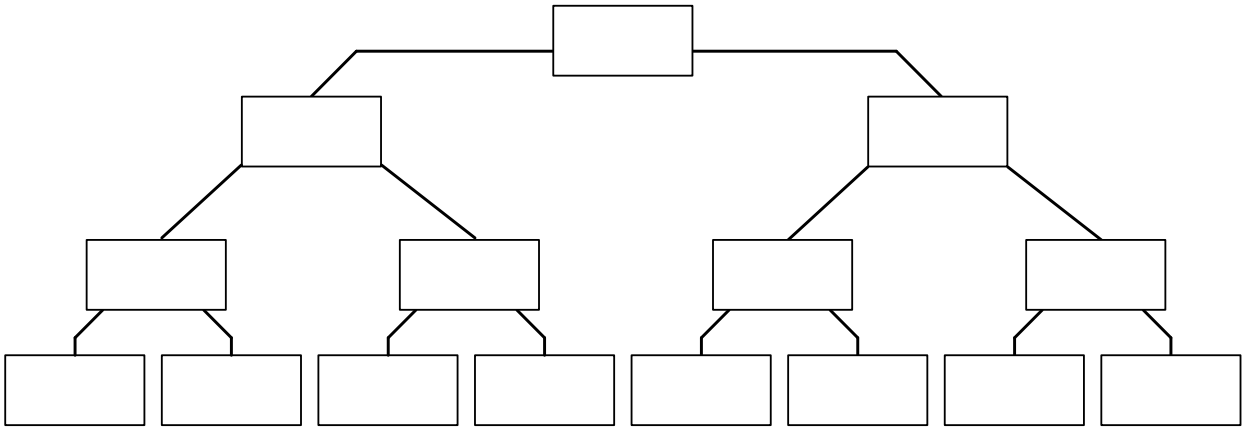
1.5) (1 pts) Donnez la complexité asymptotique en cas moyen d'un `insert(AnyType x)`.

1.6) (1 pts) Donnez la complexité asymptotique en pire cas d'un `insert(AnyType x)`.

1.7) (2 pts) Dessinez le monceau contenu en mémoire dans le tableau ci-après.

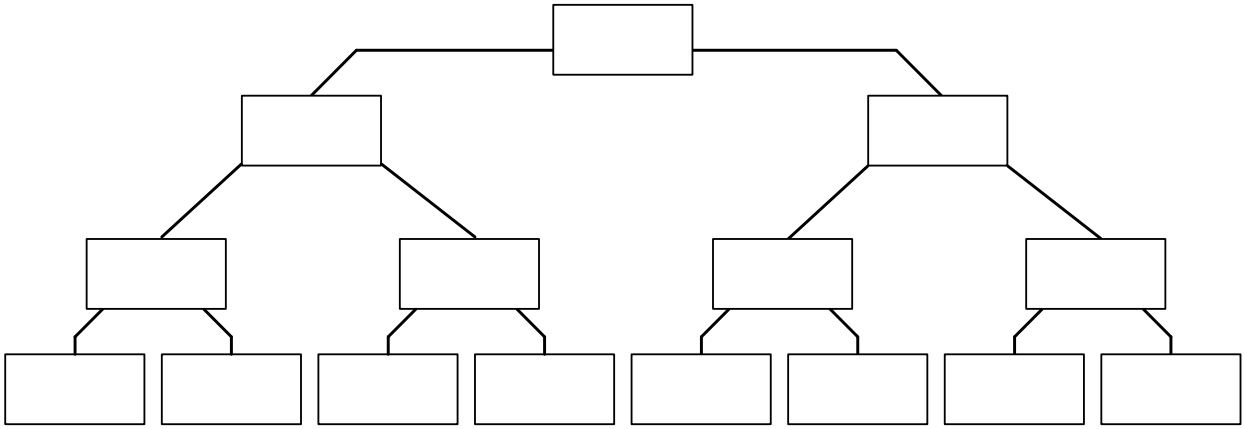
Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Contenu	-	8	8	8	8	12	12	8	13	13	14	15	13	14	15	15

Votre réponse :



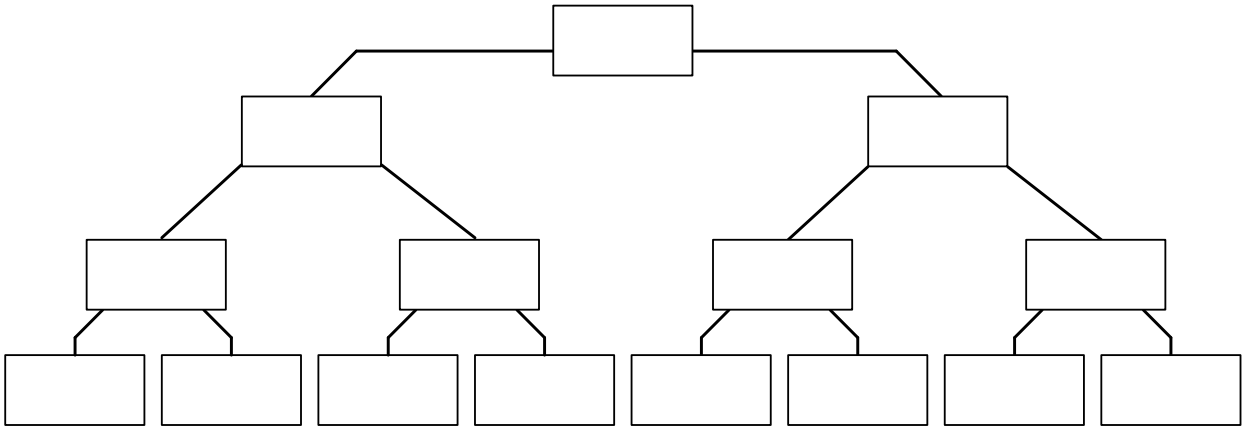
1.8) (2 pts) En partant du monceau de 1.7), effectuez un `deleteMin()`.

Votre réponse :



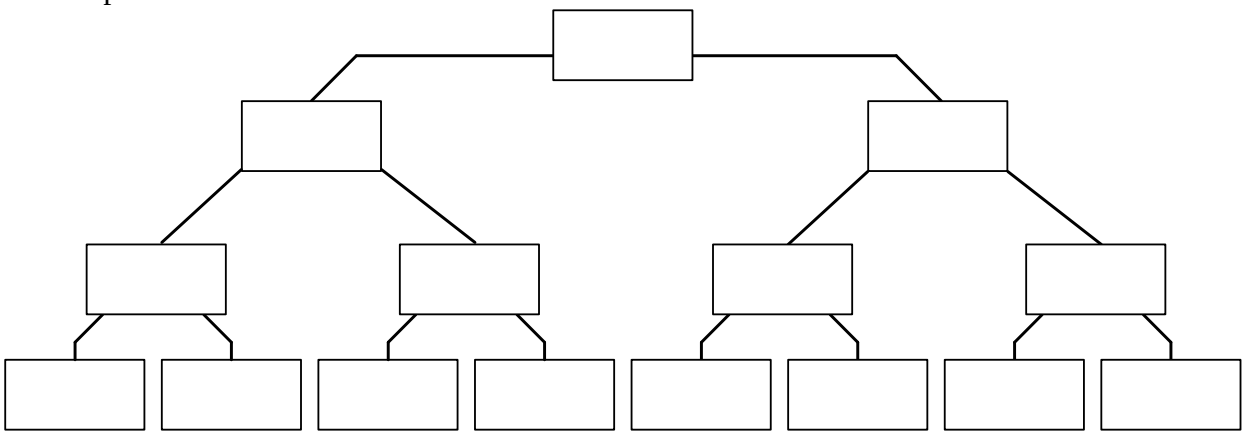
1.9) (2 pts) En partant du monceau de 1.8), effectuez un `deleteMin()`.

Votre réponse :



1.10) (2 pts) En partant du monceau de 1.9), effectuez un `deleteMin()`.

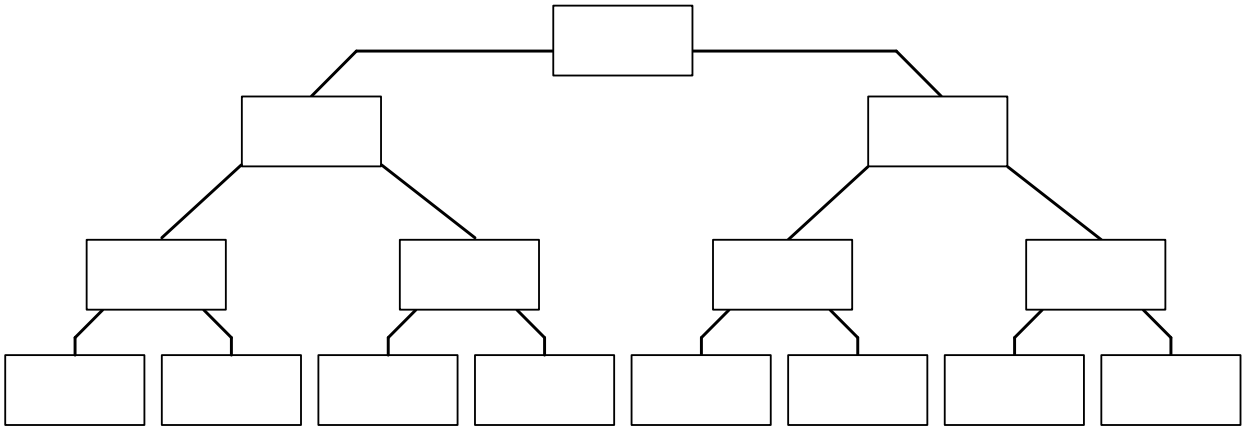
Votre réponse :



1.11) (2 pts) Dessinez le monceau résultant d'appel à `BinaryHeap(AnyType[ ] items)` où `items` est le tableau suivant :

Indice	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Contenu	13	14	12	13	8	9	10	9	10	8	14	8	9	10	12	

Votre réponse :



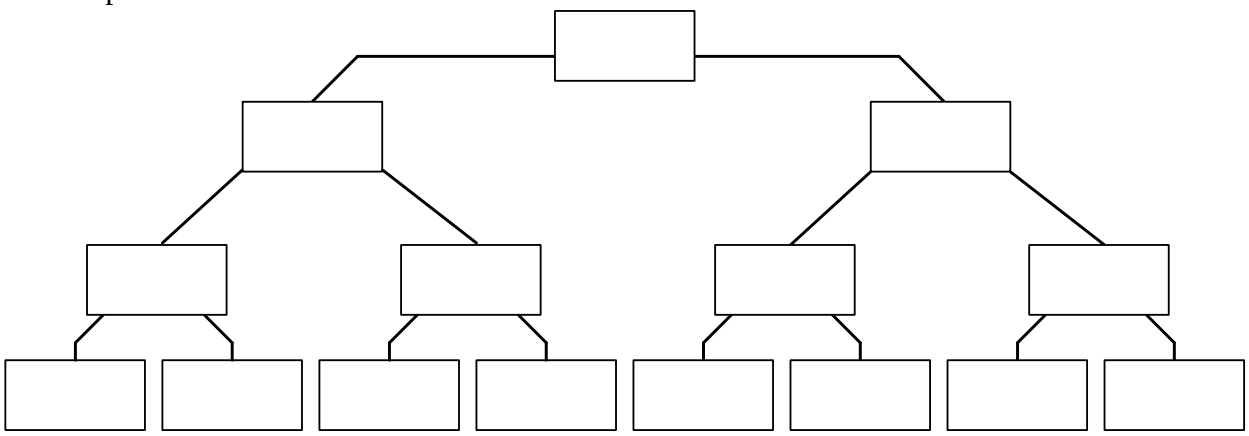
1.12) (4 pts) Dessinez le monceau résultant d'appel à `BinaryHeap(AnyType[ ] items)` où `items` est le tableau suivant :

Indice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Contenu	13	14	12	13	8	9	10	9	10	8	14	8	9	10	12	

Et où la méthode `percolateDown` a été modifiée comme suit :

```
private void percolateDown( int hole ) {
    int child;
    AnyType tmp = array[ hole ];

    for( ; hole * 2 <= currentSize; hole = child ) {

        child = hole * 2;

        if( child != currentSize &&
            array[child].compareTo( array[child+1] ) >= 0 )
            child++;

        if( array[ child ].compareTo( tmp ) < 0 )
            array[ hole ] = array[ child ];
        else
            break;
    }
    array[ hole ] = tmp;
}
```

Votre réponse :



Question 2 : Recherche de patron (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 entier via un calcul de hachage. Une implémentation de Rabin Karp vous est proposée à l'Annexe 2. On y considère l'alphabet Σ = {G, C, A, T} où d = | Σ | = 4. On admettra l'encodage suivant :

Symbole	G	C	A	T
Code	0	1	2	3

Le hachage est calculé en base d modulo q = 2^7-1 = 127. Par exemple, la séquence « GATAC », encodée 0, 2, 3, 2, 1, produira une valeur de hash :

$$\text{mod}((((\mathbf{0}\cdot d + \mathbf{2})\cdot d + \mathbf{3})\cdot d + \mathbf{2})\cdot d + \mathbf{1}, q) = 58.$$

2.1) (2 pts) Donnez le hash associé au patron « ATGCTTC » :

Afin d'accélérer les calculs lors de la recherche du patron dans le texte, Rabin Karp exploite une formule récursive : $t_{s+1} = \text{mod}((t_s - hT[s])d + T[s+m+1], q)$, où $h = \text{mod}(d^{m-1}, q)$. Sachant que $\text{mod}(2^m, 2^p-1) = 2^{\text{mod}(m, p)}$, nous aurons

$$\begin{aligned} h &= \text{mod}(d^{m-1}, q) \\ &= \text{mod}(4^{m-1}, 127) \\ &= \text{mod}(2^{2(m-1)}, 2^7-1) \\ &= 2^{\text{mod}(2(m-1), 7)} \end{aligned}$$

2.2) (2 pts) Donnez la valeur de la variable h pour le patron « ATGCTTC » :

Votre réponse :

h = _____

2.3) (16 pts) Combien de faux positifs seront détectés si le patron « ATGCTTC » est recherché dans le texte « CGAATGCTTCTCAA ». Fiez-vous au code java donné à l'Annexe 2. Remarquez que dans l'implémentation donnée de la fonction `updateHash`, le calcul du modulo renvoie toujours une valeur positive.

```
private int updateHash(int oldHash, char oldChar, char newChar) {
   if( !ENCODING.containsKey( oldChar ) ) throw new IllegalArgumentException();
   if( !ENCODING.containsKey( newChar ) ) throw new IllegalArgumentException();

   int newHash = oldHash;
   newHash -= h*ENCODING.get( oldChar );
   if( newHash < 0 ) newHash += q; // valeur positive
   newHash *= d;
   newHash += ENCODING.get( newChar );
   newHash %= q;
   return newHash;
}
```

Vous pouvez vous aider de la table ci-après :

Décalage	Sous-séquence	t_s
0	CGAATGC	
1	GAATGCT	
2	AATGCTT	
3	ATGCTTC	
4	TGCTTCT	
5	GCTTCTC	
6	CTTCTCA	
7	TTCTCAA	

2.3.a) **(4 pts)** Nombre de faux positifs détectés :

2.3.b) **(12 pts)** Les sous-séquences associées aux faux positifs détectés ET les décalages correspondants :

Question 3 : Programmation dynamique (12 points)

On désire trouver le parenthésage idéal pour multiplier les matrices A_1 à A_5 permettant de minimiser le nombre de multiplications (scalaires) à effectuer. Les matrices sont dimensionnées comme suit :

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

Considérez les tables **m** et **s** obtenues par l'exécution de l'algorithme dynamique vu en cours.

m	1	2	3	4	5
1	0	9	12	11	
2		0	6	8	
3			0	6	
4				0	6
5					0

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

Complétez cette table pour répondre aux questions suivantes :

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

3.1) (3 pts) Donnez le parenthésage optimal pour multiplier A_3 à A_5. Donnez son coût.

Parenthésage optimal: A_3 A_4 A_5

Coût: _____

3.2) (4 pts) Donnez le parenthésage optimal pour multiplier A_2 à A_5. Donnez son coût.

Parenthésage optimal: A_2 A_3 A_4 A_5

Coût: _____

3.3) (5 pts) Donnez le parenthésage optimal pour multiplier A_1 à A_5. Donnez son coût.

Parenthésage optimal: A_1 A_2 A_3 A_4 A_5

Coût: _____

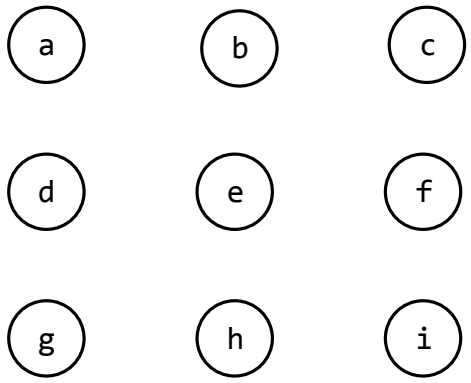
Question 4 : Ordre topologique (16 points)

On veut connaître l'ordre topologique du graphe suivant :

V = {a, b, c, d, e, f, g, h, i}

E = {(a, b), (a, d), (c, b), (c, f), (d, b), (d, g), (d, h), (e, b), (e, d), (e, f), (e, h), (f, b), (f, i), (g, h), (h, f), (h, i)}

4.1) (2 pts) Reproduisez graphiquement le graphe G = (V, E) :



4.2) (7 pts) Donnez l'ordre topologique du graphe G en appliquant l'algorithme utilisant une file vu en classe.

Nœud	1	2	3	4	5	6	7	8	9
a									
b									
c									
d									
e									
f									
g									
h									
i									
Entrée									
Sortie									

Ordre trouvé (débutez la numérotation à 1) :

Nœud	a	b	c	d	e	f	g	h	i
Ordre :									

4.3) (7 pts) Donnez l'ordre topologique du graphe G en appliquant l'algorithme utilisant le parcours DFS post-ordre inverse. Partez du nœud `a` et visitez les nœuds alphabétiquement.

4.3.a) (2 pts) Donnez l'affichage DFS post-ordre obtenu :

4.3.b) (2 pts) Donnez l'affichage DFS post-ordre inverse obtenu :

4.3.c) (3 pts) Donnez l'ordre topologique trouvé (débutez la numérotation à 1) :

Nœud	a	b	c	d	e	f	g	h	i
Ordre :									

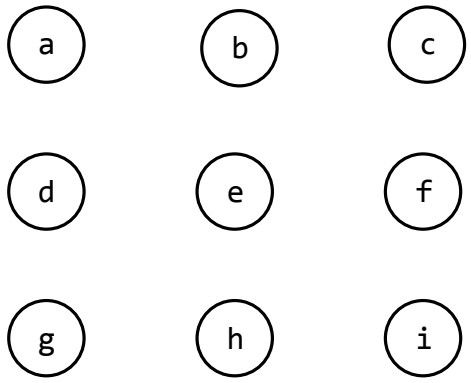
Question 5 : Composantes Fortement Connexes (18 points)

On veut connaître l'ordre topologique du graphe suivant :

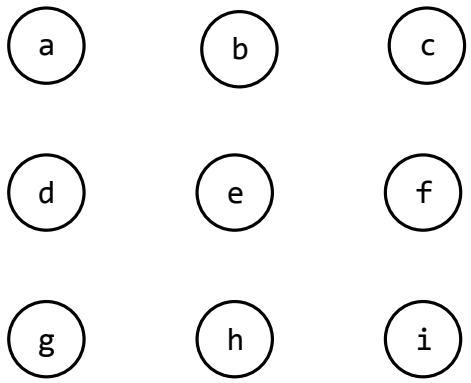
V = {a, b, c, d, e, f, g, h, i}

E = {(a, d), (b, a), (c, b), (c, f), (d, b), (d, h), (e, b), (e, d), (e, f), (e, h), (f, b), (f, i), (g, d), (g, h), (h, f), (i, h)}

5.1) (2 pts) Reproduisez graphiquement le graphe G = (V, E) :



5.2) (2 pts) Donnez le graphe G^T, c'est-à-dire le graphe transposé de G :



5.3) (4 pts) Donnez l'affichage DFS post-ordre inverse de G^T. Visitez les nœuds et les voisins dans l'ordre alphabétique.

5.4) (4 pts) Donnez l'affichage DFS post-ordre de G suivant l'ordre d'affichage obtenu en 5.3). Visitez les voisins dans l'ordre alphabétique.

5.5) (6 pts) En vous servant du parcours réalisé dans 5.4), identifiez les composantes fortement connexes du graphe G = (V, E).

Note : Le fait que la table comporte 5 lignes ne signifie pas qu'il y ait 5 composantes fortement connexes. Laissez les lignes inutilisées vides.

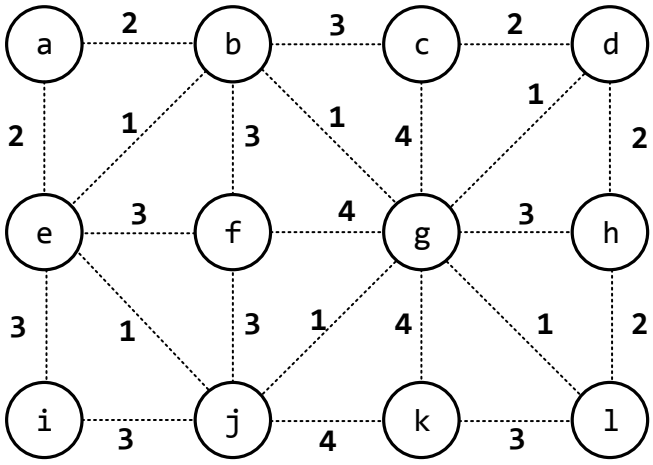
Composante	Nœuds
1	
2	
3	
4	
5	

Question 6 : Arbre sous-tendant minimum (14 points)

Donnez les arbres sous-tendant minimum obtenus par les algorithmes de Prim (le nœud de départ étant `a`) et Kruskal en reliant les arêtes retenues dans les graphes données ci-après.

Respectez l'ordre alphabétique pour visiter les nœuds voisins ou les arêtes. Utilisez les tables fournies pour ce faire (le remplissage des tables ne compte pas dans l'attribution des points pour la question).

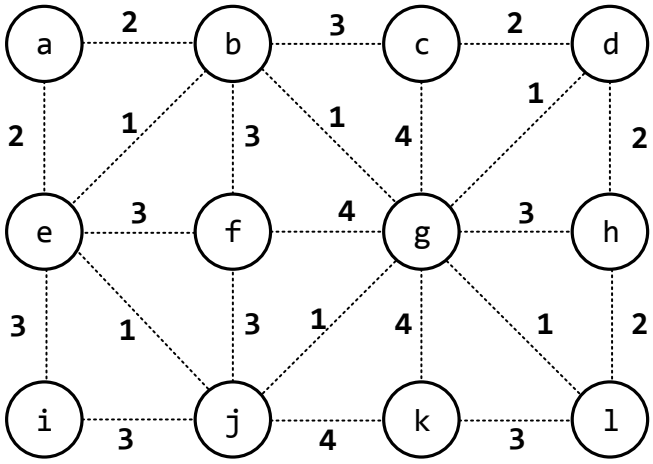
Par Prim (7 pts) :



Prim:

Nœud	Distance	Parent	Connu?
a			
b			
c			
d			
e			
f			
g			
h			
i			
j			
k			
l			

Par Kruskal (7 pts) :



Kruskal :

Arête	Poids	Retenue?
(a, b)	2	
(a, e)	2	
(b, c)	3	
(b, e)	1	
(b, f)	3	
(b, g)	1	
(c, d)	2	
(c, g)	4	
(d, g)	1	
(d, h)	2	
(e, f)	3	
(e, i)	3	
(e, j)	1	
(f, g)	4	
(f, j)	3	
(g, h)	3	
(g, j)	1	
(g, k)	4	
(g, l)	1	
(h, l)	2	
(i, j)	3	
(j, k)	4	
(k, l)	3	

Annexe 1

```
public class BinaryHeap<AnyType extends Comparable<? super AnyType>>
{
    public BinaryHeap( ) { this( DEFAULT_CAPACITY ); }

    @SuppressWarnings("unchecked")
    public BinaryHeap( int capacity ) {
        currentSize = 0;
        array = (AnyType[]) new Comparable[ capacity + 1 ];
    }

    @SuppressWarnings("unchecked")
    public BinaryHeap( AnyType [ ] items ) {
        currentSize = items.length;
        array = (AnyType[]) new Comparable[ ( currentSize + 2 ) * 11 / 10 ];

        int i = 1;
        for( AnyType item : items )
            array[ i++ ] = item;
        buildHeap( );
    }

    public void insert( AnyType x ) {
        if( currentSize == array.length - 1 )
            enlargeArray( array.length * 2 + 1 );

        int hole = ++currentSize;
        for( ; hole > 1 && x.compareTo( array[ hole / 2 ] ) < 0; hole /= 2 )
            array[ hole ] = array[ hole / 2 ];
        array[ hole ] = x;
    }

    @SuppressWarnings("unchecked")
    private void enlargeArray( int newSize ) {
        AnyType [] old = array;
        array = (AnyType []) new Comparable[ newSize ];
        for( int i = 0; i < old.length; i++ )
            array[ i ] = old[ i ];
    }

    public AnyType findMin( ) {
        if( isEmpty( ) )
            throw new UnderflowException( );
        return array[ 1 ];
    }

    public AnyType deleteMin( ) {
        if( isEmpty( ) ) throw new UnderflowException( );

        AnyType minItem = findMin( );
        array[ 1 ] = array[ currentSize-- ];
        percolateDown( 1 );

        return minItem;
    }
```

```
    private void buildHeap( ) {
        for( int i = currentSize / 2; i > 0; i-- )
            percolateDown( i );
    }

    public boolean isEmpty( ) {
        return currentSize == 0;
    }

    public void makeEmpty( ) {
        currentSize = 0;
    }

    private static final int DEFAULT_CAPACITY = 10;

    private int currentSize;      // Number of elements in heap
    private AnyType [ ] array; // The heap array

    private void percolateDown( int hole ) {
        int child;
        AnyType tmp = array[ hole ];

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

Annexe 2

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class PatternMatch {

   private static final Map<Character, Integer> ENCODING;
    static {
        ENCODING = new HashMap<Character, Integer>();
        ENCODING.put('G', 0); ENCODING.put('C', 1);
        ENCODING.put('A', 2); ENCODING.put('T', 3);
    }
    private static final int d = 4;
    private static final int v = 7;
    private static final int q = (1 << v) - 1; // 2^v - 1 = 2^7 - 1 =127
    private int p;
    private int m;
    private int h;
    private String pattern;

   PatternMatch(String pattern){
      if(pattern == null) throw new IllegalArgumentException();
      if(pattern.length() == 0) throw new IllegalArgumentException();

      this.pattern = pattern;
      m = pattern.length();
      p = computeHash(pattern);
      h = 1 << ((2*(m-1)) % v); // 2^mod(2(m-1), v)
   }
   private int computeHash(String pattern) {
      int p = 0;
      for(int i=0; i<pattern.length(); i++){
         char c = pattern.charAt(i);

         if( !ENCODING.containsKey( c ) ) throw new IllegalArgumentException();

         p *= d; p += ENCODING.get( c );
         p %= q;
      }
      return p;
   }
   private int updateHash(int oldHash, char oldChar, char newChar) {
      if( !ENCODING.containsKey( oldChar ) ) throw new IllegalArgumentException();
      if( !ENCODING.containsKey( newChar ) ) throw new IllegalArgumentException();

      int newHash = oldHash;
      newHash -= h*ENCODING.get( oldChar );
      if( newHash < 0 ) newHash += q;
      newHash *= d;
      newHash += ENCODING.get( newChar );
      newHash %= q;
      return newHash;
   }
```

```
   public ArrayList<Integer> findPattern(String text) {

      if(text== null) throw new IllegalArgumentException();
      if(text.length() == 0) throw new IllegalArgumentException();

      ArrayList<Integer> shifts = new ArrayList<Integer>();
      int n = text.length();

      if( n < m ) return shifts;

      int i = 0;
      String substring = text.substring(0, m);
      int t = computeHash(substring);

      if( t == p && (substring.equals(pattern)) )
         shifts.add( i );

      while( ++i <= (n-m) ){
         char oldChar = text.charAt(i - 1);
         char newChar = text.charAt(i + m - 1);
         t = updateHash(t, oldChar, newChar);
         if( t == p && (text.substring(i, i+m).equals(pattern)) )
             shifts.add( i );
      }

      return shifts;
   }

   public static void main(String[] args) {
      String pattern  = "ATGCTTC";
      String text     = "CGAATGCTTCTCAA";
      PatternMatch pm = new PatternMatch(pattern);

      ArrayList<Integer> shifts = pm.findPattern(text);

      for(int s : shifts )
         System.out.println( s + "\t" +  text.substring(s, s+pattern.length()));
   }
}
```