

Question 1 : Monceaux**(4/20 points)**

Pour cette question, référez-vous au code Java donné à l'Annexe 1. Il s'agit de l'implémentation d'un monceau min telle que vue en classe (Weiss) et augmentée de la méthode :

```
public String printArray() {  
    StringBuilder sb = new StringBuilder();  
    for (int i=1; i<=currentSize; i++)  
        sb.append(array[i].toString() + " ");  
    return sb.toString();  
}
```

Considérez la méthode `main` de l'Annexe 1 qui manipule un monceau `h` sur des objets de type `Integer` et reproduisez les cinq (5) affichages qui y sont indiqués pour chacune des questions suivantes, tel que les exécute la méthode `main`. À titre d'exemple, les lignes 107 et 108 de l'Annexe 1 afficheront :

Exemple

1 3 2

Q1.1) (0.75 point) Donnez le résultat de l'affichage obtenu en exécutant les lignes 119 et 120 de l'Annexe 1.

1 2 3 5 5 4

Q1.2) (0.75 point) Donnez le résultat de l'affichage obtenu en exécutant les lignes 130 et 131 de l'Annexe 1.

4 5 5

Q1.3) (0.75 point) Donnez le résultat de l'affichage obtenu en exécutant les lignes 144 et 145 de l'Annexe 1.

1 3 1 6 4 5 2

Q1.4) (0.75 point) Donnez le résultat de l'affichage obtenu en exécutant les lignes 155 et 156 de l'Annexe 1.

3 4 5 6

Q1.5) (1 point) Donnez le résultat de l'affichage obtenu en exécutant les lignes 164 et 165 de l'Annexe 1.

1 2 3 6 9 4 7 8 5

Question 2 : Automate de reconnaissance de motifs (4/20 points)

2.1) **(1.5 point)** En utilisant l'algorithme de construction des automates de reconnaissance de motifs, construisez l'automate capable de reconnaître la séquence suivante:

« ababc »

en REMPLISSANT le **Tableau 1** avec la fonction de transition d'état correspondante.

Tableau 1

Symboles États	a	b	c
0	1	0	0
1	1	2	0
2	3	0	0
3	1	4	0
4	3	0	5
5	2	0	0
6			
7			

NOTE : Ajoutez ou ignorez des lignes ou des colonnes au besoin.

État initial : 0

État final : 5

2.2) **(1 point)** Exécutez l'automate en reconnaissance sur le texte suivant:

« bcababcabc »

en remplissant le **Tableau 2** avec l'état de l'automate APRÈS l'analyse des sous-chaînes (préfixes) indiquées.

Tableau 2

Sous-chaîne	État
b	0
bc	0
bca	1
bcab	2
bcaba	3
bcabab	4
bcababc	5
bcababca	1
bcababcb	2
bcababcbc	0
bcababcbcā	1

2.3) (0.5 point) Combien de fois le motif a-t-il été reconnu dans l'exécution de l'automate ?

1 fois

2.4) (0.5 point) Quelle est la complexité asymptotique de l'algorithme de CONSTRUCTION des automates de reconnaissance de motifs ?

~~$O(\log(n))$~~

2.5) (0.5 point) Quelle est la complexité asymptotique de l'algorithme de RECONNAISSANCE d'un motif en utilisant un automate Q ?

~~$O(\log(n))$~~

Question 3 : Programmation dynamique**(4/20 points)**

3.1) (1 point) REMPLISSEZ le Tableau 3 suivant avec les informations de longueur et de provenance pour retrouver la plus longue sous-séquence commune aux chaînes en entrée :

$$X = a b \underline{a} \underline{a} b \underline{b} \underline{b} a b \underline{a} b \underline{b} b b a \text{ et } Y = b \underline{a} \underline{a} b \underline{a} b a b a$$

Tableau 3

	Y	b	a	a	b	a	b	a	b	b	a
X	0	0	0	0	0	0	0	0	0	0	0
a	0	↑0*	1	1	1	1	1	1	1	1	1
b	0	1	11*	11*	2	2	2	2	2	2	2
a	0	11	2	212*	3	3	3	3	3	3	3
a	0	11	2	3	3	3	3	4	4	4	4
b	0	1	12	13	4	4	4	4	5	5	5
b	0	1	12	13	4	14*	5	5	5	5	5
a	0	11	2	3	14	5	15*	6	6	6	6
b	0	1	12	13	4	15	6	16*	7	7	7
a	0	11	2	3	14	5	16	7	17*	8	8
b	0	1	12	13	4	15	6	17	8	18*	8
b	0	1	12	13	4	15	6	17	8	18*	8
b	0	1	12	13	4	15	6	17	8	18*	8
a	0	11	2	3	14	5	16	7	18	9	9

3.2) (0.5 point) ÉCRIVEZ la longueur de la plus longue sous-séquence commune :

~~4~~

3.3) (0.5 point) ÉCRIVEZ la plus longue sous-séquence commune :

~~ba~~

3.4) CONSIDÉREZ les différentes sous séquences en commun de la même longueur maximale:

3.4.1) (1 point) Est-ce possible de savoir de façon booléenne simplement s'il y avait plus qu'une sous séquence en commun de longueur maximale sans en savoir le nombre, à partir seulement des résultats du **Tableau 3**? Comment feriez-vous, si possible? JUSTIFIEZ brièvement votre réponse.

~~Oui, si la valeur de la dernière case est plus grande~~

3.4.2) (1 point) Est-ce possible de calculer le ~~nombre~~ de sous séquences en commun de longueur maximale à partir seulement des résultats du **Tableau 3**? COMMENT feriez-vous, si possible? JUSTIFIEZ brièvement votre réponse.

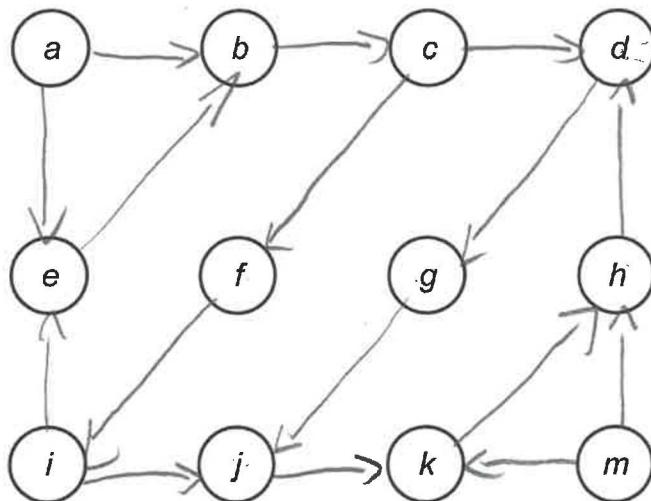
Question 4 : Composantes fortement connexes**(4/20 points)**

On veut connaître les composantes fortement connexes du graphe dirigé suivant :

$$V = \{a, b, c, d, e, f, g, h, i, j, k, m\}$$

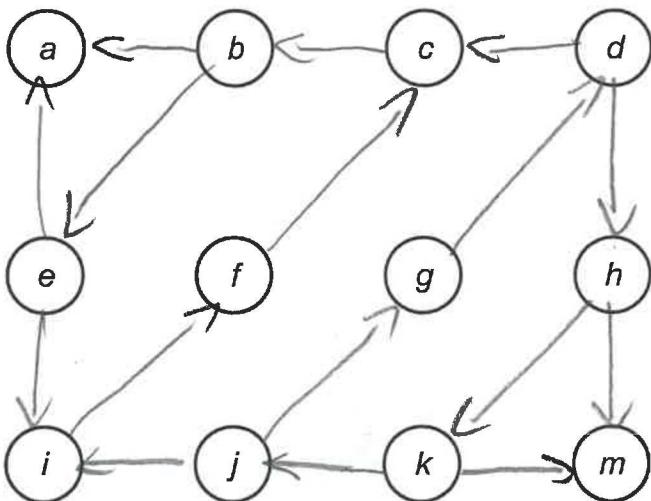
$$E = \{(a, b), (a, e), (b, c), (c, d), (c, f), (d, g), (e, b), (f, i), (g, j), (h, d), (i, e), (i, j), (j, k), (k, h), (m, h), (m, k)\}$$

- 4.1) (0.5 point) Reproduisez graphiquement le graphe $G = (V, E)$:



d g h j K
m
e b c f i
a

- 4.2) (0.5 point) Donnez G^T , le graphe transposé de G :



- 4.3) (2 points) Donnez les composantes fortement connexes (CFC) de G en associant chacun des nœuds a à m ci-après à une composante. Les composantes sont numérotées de façon incrémentale et la numérotation débute à 1. Laissez les colonnes inutilisées vides.

Nœud	Composante						
	1	2	3	4	5	6	7
a	•						
b		•					
c		•					
d			•				
e		•					
f		•					
g			•				
h			•				
i		•					
j			•				
k			•				
m				•			

- 4.4) (0.5 point) Quel est le plus grand nombre d'arcs qu'il est possible de retirer du graphe G sans affecter ses CFC ? Justifiez brièvement votre réponse.

(a,b), (a,e), (m,k) et (m,h)

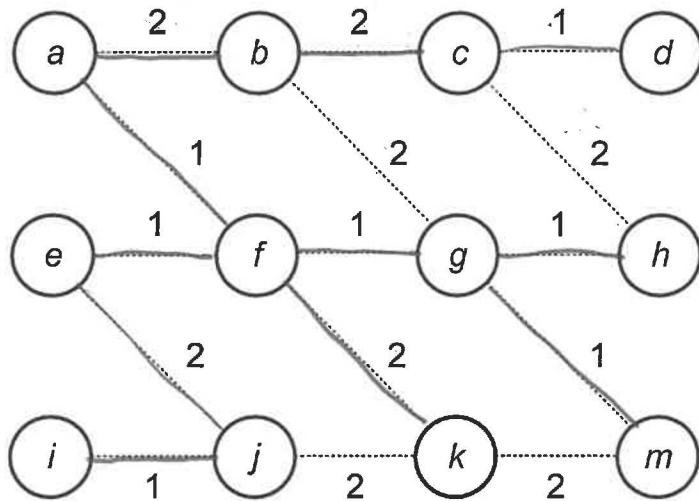
Ces arcs ne font pas partie d'un cycle et ils n'affectent pas les cycles des composantes 1 et 2

- 4.5) (0.5 point) Si on note G' le graphe obtenu en retirant de G tous les arcs n'affectant pas ses CFC tel qu'identifié en 4.4, combien $(G')^T$ admet-il de CFC ? Justifiez brièvement votre réponse.

Le même nombre que G' , car la transposition n'affecte pas les composantes fortement connexes. Donc, 4 composantes.

Question 5 : Arbre sous-tendant minimum (4/20 points + 1 point bonus)

5.1) (1.5 point) Donnez l'arbre sous-tendant minimum obtenu par l'algorithme de Kruskal en noircissant les arêtes retenues dans le graphe ci-après. DONNEZ le coût de l'arbre ainsi obtenu.



Coût:

Kruskal :

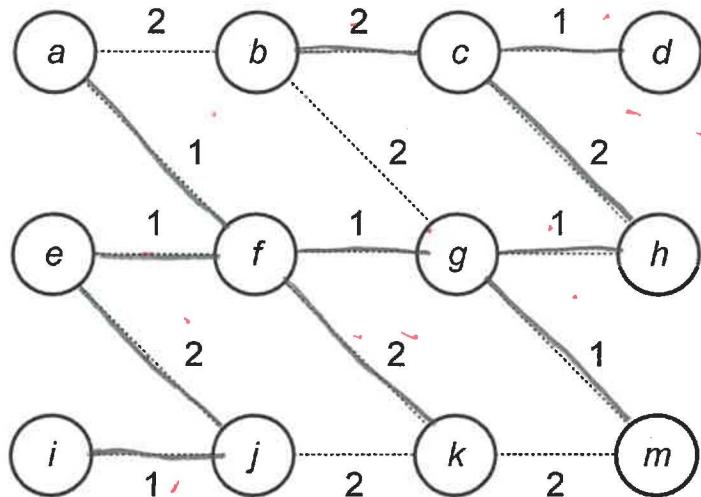
Arête	Coût	Retenue?
(a, b)	2	✓
(a, f)	1	✓
(b, c)	2	✓
(b, g)	2	✗
(c, d)	1	✓
(c, h)	2	✗
(e, f)	1	✓
(e, j)	2	✓
(f, g)	1	✓
(f, k)	2	✓
(g, h)	1	✓
(g, m)	1	✓
(i, j)	1	✓
(j, k)	2	✗
(k, m)	2	✗

5.2) (1 point) Quelle structure de données serait appropriée pour choisir l'arête de plus faible poids dans l'exécution de l'algorithme de Kruskal par ensembles disjoints ?

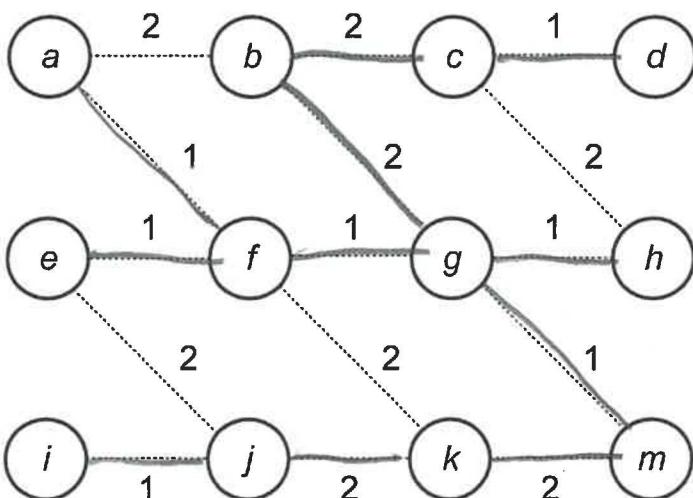
un monceau min

5.3) (1.5 point) Le graphe précédent admet d'autres arbres sous-tendant minimum. Proposez-en trois.

5.3.1)

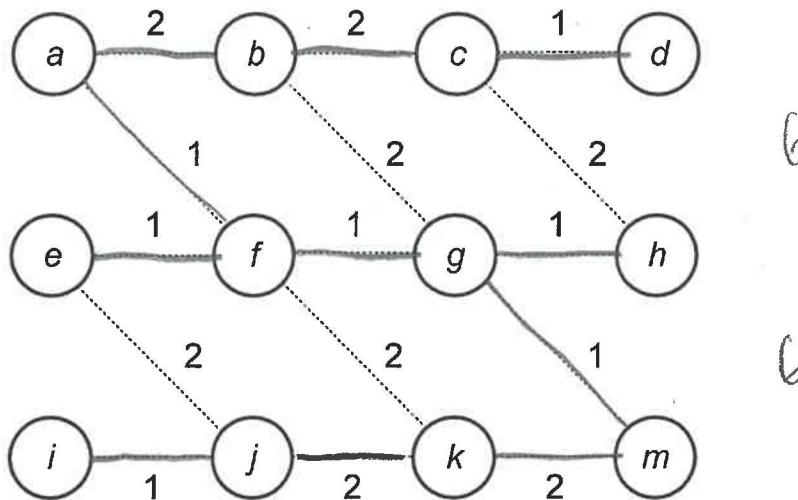


5.3.2)



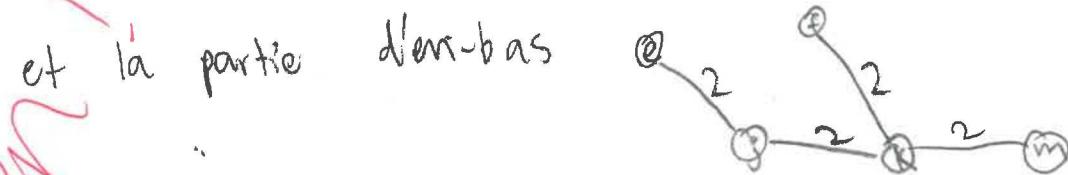
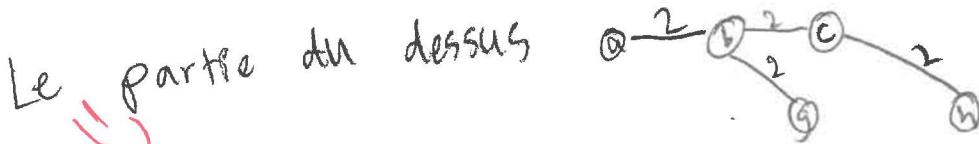
1 6

5.3.3)



5.4) (1 point bonus) Combien d'arbres sous-tendant minimum différents le graphe précédent admet-il au total ? Justifiez votre réponse par un calcul.

$$6^2 = 36$$



Bien

La partie du dessus a 6 différents pairs d'arcs qu'on peut choisir et la partie d'en-bas à 6 différents pairs d'arcs aussi. Pour combiner les différentes possibilités on fait simplement $6^2 = 36$ possibilités.

On est obligé de prendre une paire d'arcs dans chaque partie pour visiter/connecter tous les points.

Annexe 1

```

001  public class BinaryHeap<AnyType extends Comparable<? super AnyType>> {
002
003      private static final int DEFAULT_CAPACITY = 11;
004
005      private int currentSize;           // Number of elements in heap
006      private AnyType [] array;         // The heap array
007
008      public BinaryHeap( ) { this( DEFAULT_CAPACITY ); }
009
010     @SuppressWarnings("unchecked")
011     public BinaryHeap( int capacity ) {
012         currentSize = 0;
013         array = (AnyType[]) new Comparable[ capacity + 1 ];
014     }
015
016     @SuppressWarnings("unchecked")
017     public BinaryHeap( AnyType [ ] items ) {
018         currentSize = items.length;
019         array = (AnyType[]) new Comparable[ ( currentSize + 2 ) * 11 / 10 ];
020
021         int i = 1;
022         for( AnyType item : items )
023             array[ i++ ] = item;
024         buildHeap( );
025     }
026
027     public void insert( AnyType x ) {
028         if( currentSize == array.length - 1 )
029             enlargeArray( array.length * 2 + 1 );
030
031         int hole = ++currentSize;
032         for( ; hole > 1 && x.compareTo( array[ hole / 2 ] ) < 0; hole /= 2 )
033             array[ hole ] = array[ hole / 2 ];
034         array[ hole ] = x;
035     }
036
037     @SuppressWarnings("unchecked")
038     private void enlargeArray( int newSize ) {
039         AnyType [] old = array;
040         array = (AnyType[]) new Comparable[ newSize ];
041         for( int i = 0; i < old.length; i++ )
042             array[ i ] = old[ i ];
043     }
044
045     public AnyType findMin( ) throws Exception {
046         if( isEmpty( ) ) throw new Exception( );
047         return array[ 1 ];
048     }
049
050     public AnyType deleteMin( ) throws Exception {
051         if( isEmpty( ) ) throw new Exception( );
052
053         AnyType minItem = findMin( );
054         array[ 1 ] = array[ currentSize-- ];
055         percolateDown( 1 );
056
057         return minItem;
058     }

```

```
059  private void buildHeap( ) {
060      for( int i = currentSize / 2; i > 0; i-- )
061          percolateDown( i );
062  }
063
064  public boolean isEmpty( ) {
065      return currentSize == 0;
066  }
067
068  public void makeEmpty( ) {
069      currentSize = 0;
070  }
071
072  private void percolateDown( int hole ) {
073      int child;
074      AnyType tmp = array[ hole ];
075
076      for( ; hole * 2 <= currentSize; hole = child ) {
077          child = hole * 2;
078          if( child != currentSize &&
079              array[ child + 1 ].compareTo( array[ child ] ) < 0 )
080              child++;
081          if( array[ child ].compareTo( tmp ) < 0 )
082              array[ hole ] = array[ child ];
083          else
084              break;
085      }
086      array[ hole ] = tmp;
087  }
088
089  public String printArray() {
090      StringBuilder sb = new StringBuilder();
091      for (int i=1; i<currentSize; i++ )
092          sb.append(array[i].toString() + " ");
093      return sb.toString();
094  }
095
096  public static void main( String [ ] args ) {
097
098      BinaryHeap<Integer> h;
099
100      // EXEMPLE
101      h = new BinaryHeap<Integer>( );
102      h.insert( 3 );
103      h.insert( 2 );
104      h.insert( 1 );
105
106      // Affichage donné pour exemple
107      System.out.println( "Exemple" );
108      System.out.println( h.printArray() );
```

```
109      // QUESTION 1.1
110      h = new BinaryHeap<Integer>();
111      h.insert( 5 );
112      h.insert( 3 );
113      h.insert( 2 );
114      h.insert( 1 );
115      h.insert( 5 );
116      h.insert( 4 );
117
118      // Affichage demandé pour Q 1.1
119      System.out.println( "Q 1.1" );
120      System.out.println( h.printArray() );
121
122      // QUESTION 1.2
123      try {
124          h.deleteMin();
125          h.deleteMin();
126          h.deleteMin();
127      } catch (Exception e) {e.printStackTrace();}
128
129      // Affichage demandé pour Q 1.2
130      System.out.println( "Q 1.2" );
131      System.out.println( h.printArray() );
132
133      // QUESTION 1.3
134      h = new BinaryHeap<Integer>();
135      h.insert( 5 );
136      h.insert( 6 );
137      h.insert( 4 );
138      h.insert( 3 );
139      h.insert( 1 );
140      h.insert( 1 );
141      h.insert( 2 );
142
143      // Affichage demandé pour Q 1.3
144      System.out.println( "Q 1.3" );
145      System.out.println( h.printArray() );
146
147      // QUESTION 1.4
148      try {
149          h.deleteMin();
150          h.deleteMin();
151          h.deleteMin();
152      } catch (Exception e) {e.printStackTrace();}
153
154      // Affichage demandé pour Q 1.4
155      System.out.println( "Q 1.4" );
156      System.out.println( h.printArray() );
157
158      // QUESTION 1.5
159      Integer[] cs = {9, 8, 7, 6, 5, 4, 3, 2, 1};
160
161      h = new BinaryHeap<Integer>( cs );
162
163      // Affichage demandé pour Q 1.5
164      System.out.println( "Q 1.5" );
165      System.out.println( h.printArray() );
166  }
167 }
```