ÉCOLE POLYTECHNIQUE MONTRÉAL

Le génie sans frontières

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**	**20063**
Professeur	***Local***	***Téléphone***
Ettore Merlo	**M-4021**	**5758/5193**

Jour	*Date*	*Durée*	*Heures*
Vendredi	**15 décembre 2006**	**2h30**	**9h30**

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

Directives particulières

Écrivez votre nom et votre matricule sur les pages réponses soit les pages 3 à 8, 10 et 11.

Bonne chance à tous!

Important

Cet examen contient **5** questions sur un total de **19** 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

Le plagiat, la participation au plagiat, la tentative de plagiat entraînent automatiquement l'attribution de la note **F** dans tous les cours suivis par l'étudiant durant le trimestre. L'École est libre d'imposer toute autre sanction jugée opportune, y compris l'exclusion.

Note : Remplissez les tableaux au besoin et laissez les cases non pertinentes vides.

Question 1. Arbres (15 points)

On vous demande d'insérer la séquence de nombres ci-dessous dans un arbre AVL initialement vide.

2, 10, 9, 4, 6, 8

Dessinez l'arbre après l'insertion et les rotations nécessaires des 4 derniers éléments. REMPLISSEZ le tableau 1.1 aux pages 3 et 4.

Question 2. Monceau (25 points)

Considérez le code en annexe 2.1 aux pages 12 à 16.

a) Calculez le contenu du monceau après l'insertion effectuée par l'opération avec signature "public void insert(AnyType x)" des entiers suivants :

   1, 6, 3, 10, 7, 12, 5

   REMPLISSEZ les dessins du tableau 2.2.1 et les monceaux du tableau 2.2.2 aux pages 5 et 6 avec le contenu du monceau APRÈS l'insertion des trois derniers éléments ("print" 5, 6, et 7) :

   7, 12, 5

b) Considérez le monceau suivant :

   POS: 1 VAL: 12
   POS: 2 VAL: 57
   POS: 3 VAL: 34
   POS: 4 VAL: 81
   POS: 5 VAL: 68
   POS: 6 VAL: 123
   POS: 7 VAL: 45
   POS: 8 VAL: 99

   obtenu après l'insertion des entiers suivants dans un monceau initialement vide :

   45, 68, 12, 99, 57, 123, 34, 81

   REMPLISSEZ les dessins du tableau 2.3.1 et les monceaux du tableau 2.3.2 aux pages 7 et 8 avec le contenu du monceau APRÈS l'extraction effectuée par l'opération avec signature "public AnyType deleteMin()" des trois premiers éléments ("print" 9, 10, et 11).

Question 3. Graphes (30 points)

Considérez comme ordre topologique "descendant" l'ordre topologique sur un graphe orienté sans cycle tel que les parents d'un nœud v précèdent le noeud v dans l'ordre.

Considérez comme ordre topologique "ascendant" l'ordre topologique sur un graphe orienté sans cycle tel que les fils d'un nœud v précèdent le noeud v dans l'ordre.

Considérez le graphe en Figure 3.1 à la page 9.

a) Classifiez les chemins indiqués dans le tableau 3.1.1 et REMPLISSEZ le tableau 3.1.2 à la page 10.

b) Écrivez le programme qui calcule un ordre topologique basé sur les classes en annexe 3.2 aux pages 17, 18 et 19 et sur le degré intérieur ou extérieur des nœuds qui implantent l'algorithme suggéré par M. A. Weiss.

c) Indiquez la complexité de l'algorithme au point 2b). (Il est suggéré d'utiliser une liste de travail ("*worklist*")).

Question 4. Chaînes (25 points)

Considérez la chaîne c = "ababa" et le texte t = "abababaab".

a) Construisez l'automate à états finis qui effectue la recherche de la chaîne c dans un texte arbitraire.

   REMPLISSEZ le tableau 4.1 de la page 11.

b) Identifiez le(s) décalage(s) (*shift*(s)) de concordance du texte t avec la chaîne c.

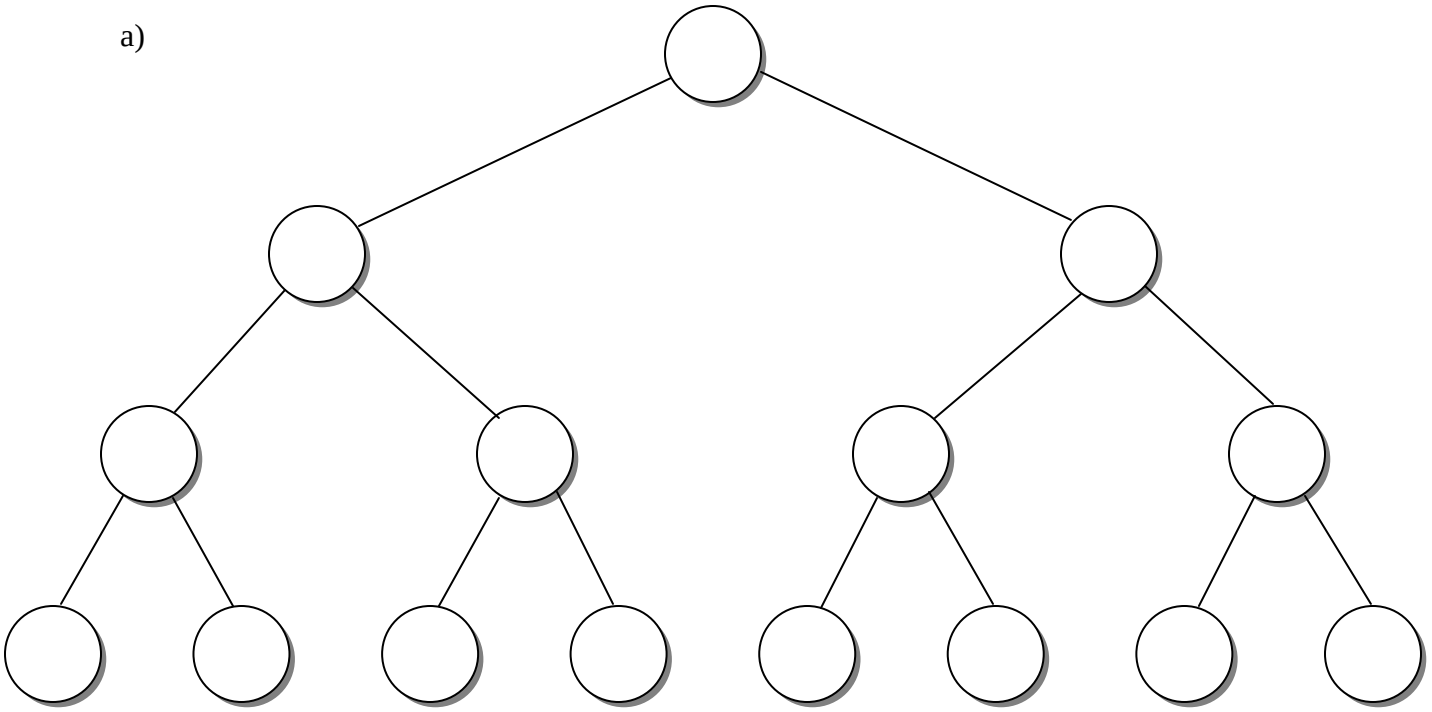
   REMPLISSEZ le tableau 4.2 de la page 11 avec le(s) décalage(s) identifié(s).

Question 5. (5 points)

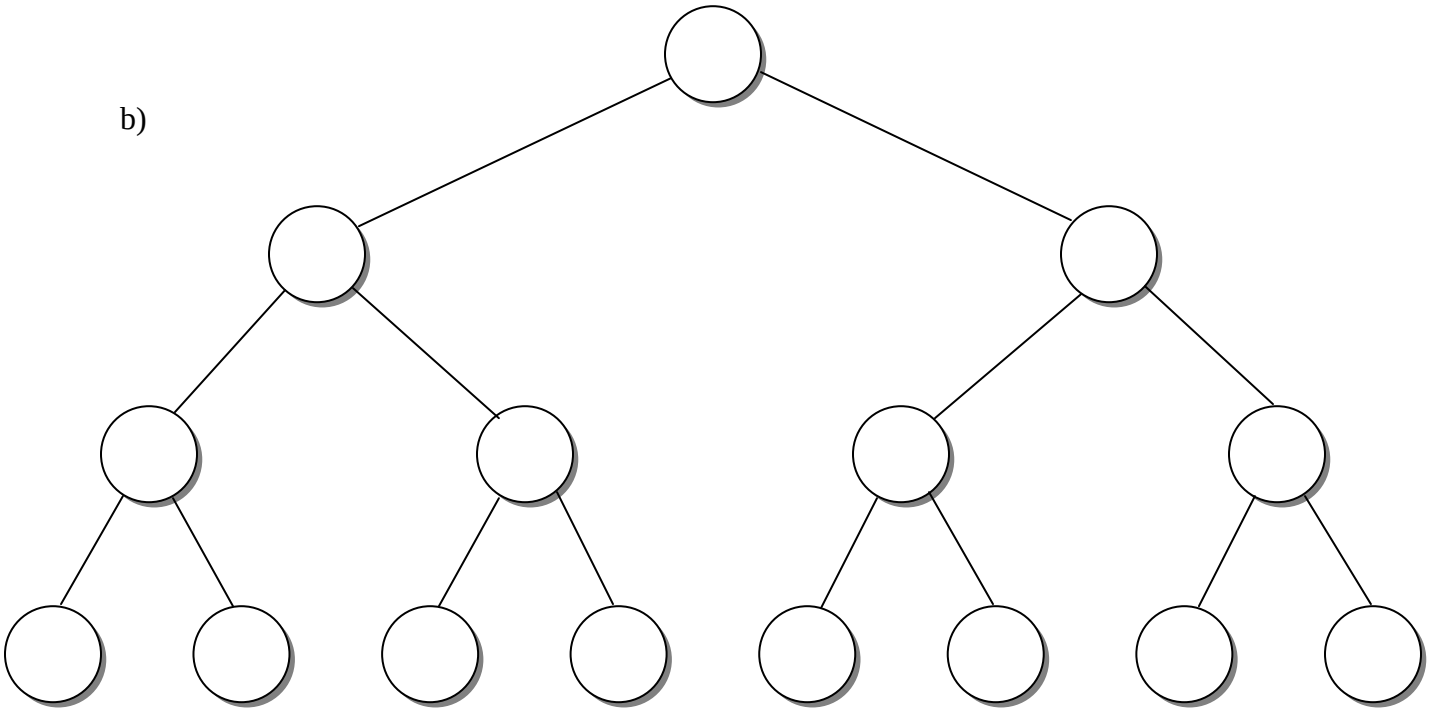
Modifiez l'algorithme de Dijkstra de façon que, s'il y avait plus qu'un chemin minimal du même poids, celui avec le plus petit nombre d'arcs (si différent entre les chemins) serait choisi.

Tableau 1.1

a)



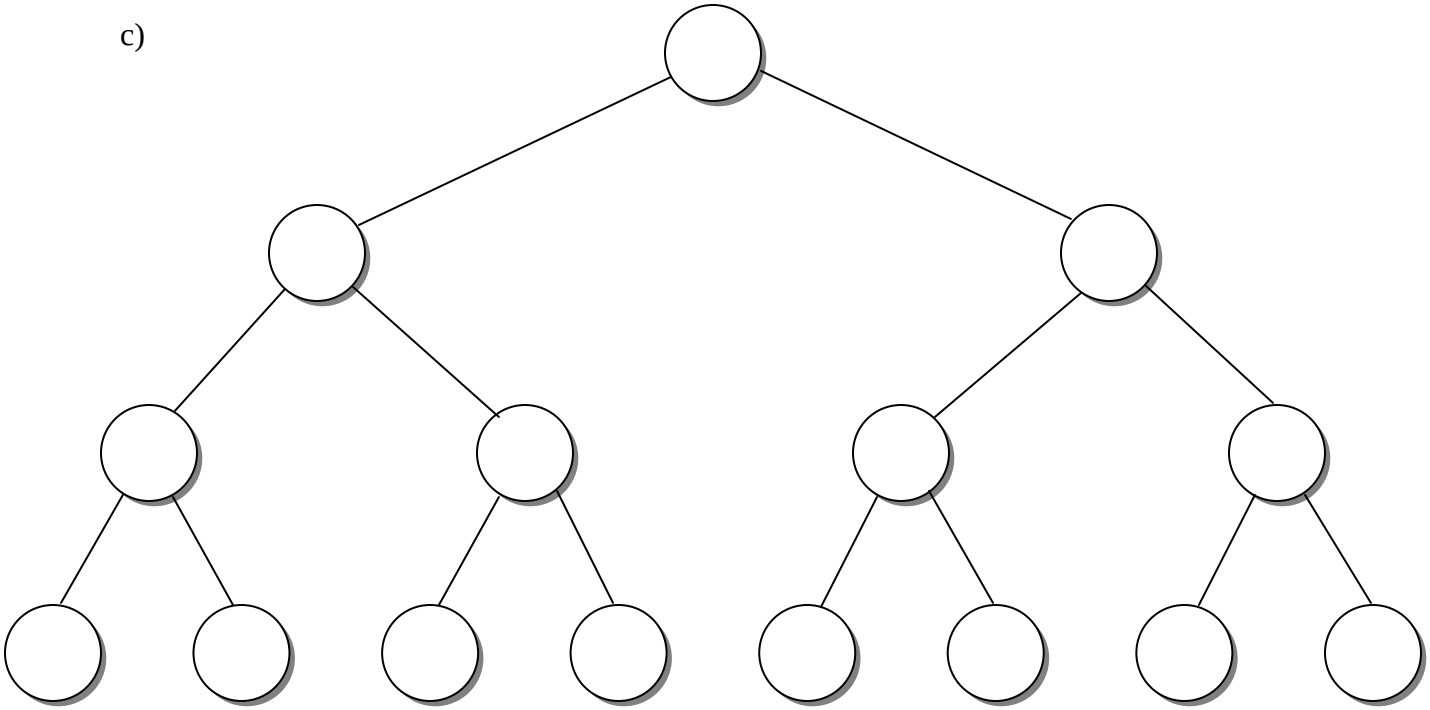
b)



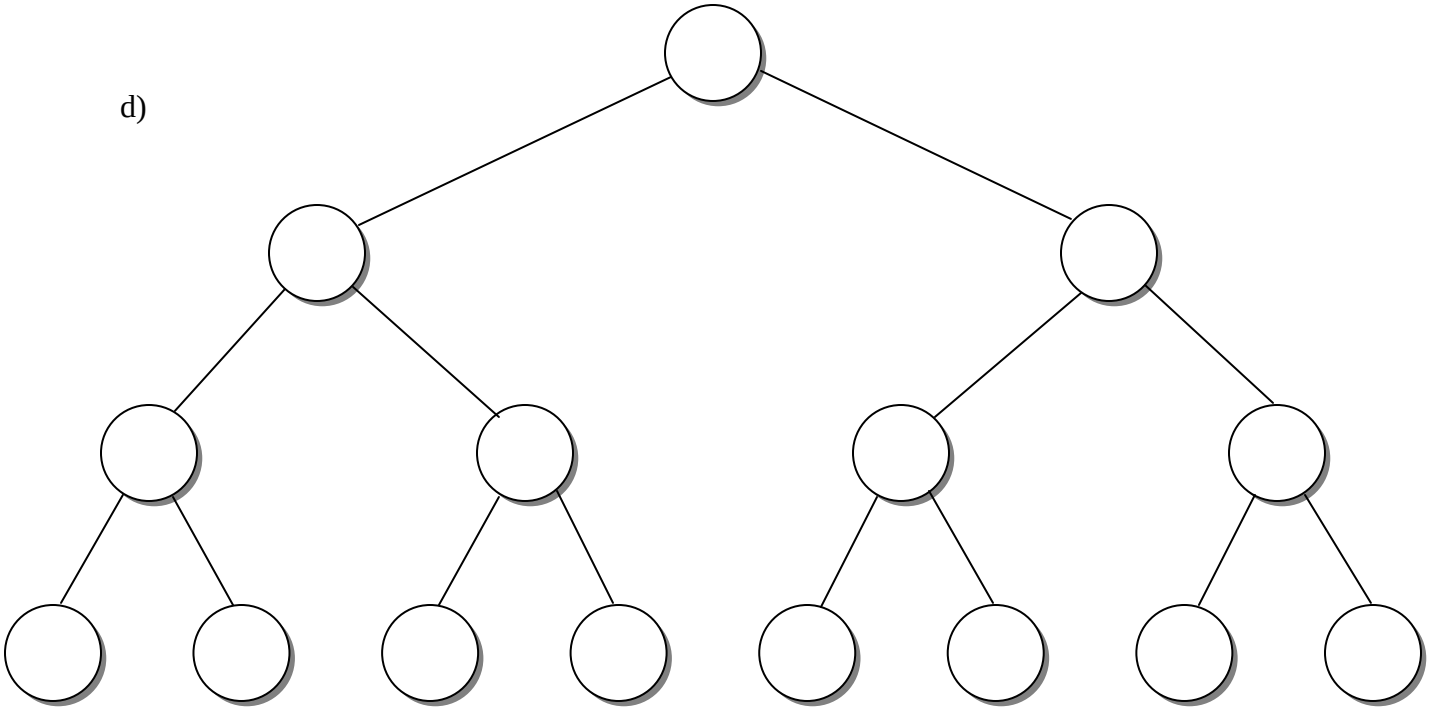
Nom : _____ Matricule : _____

Tableau 1.1 (suite)

c)



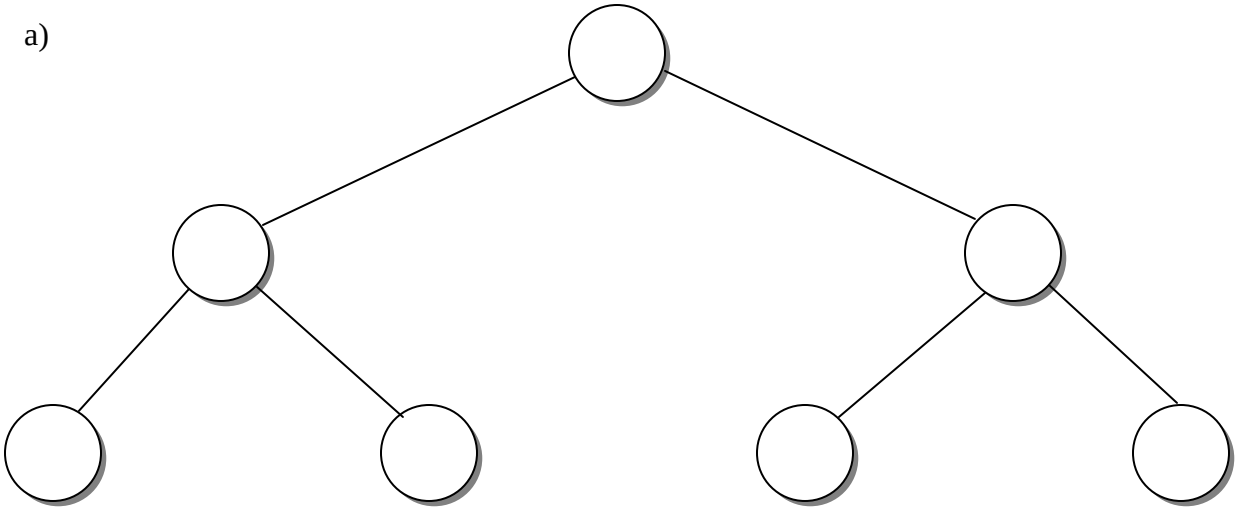
d)



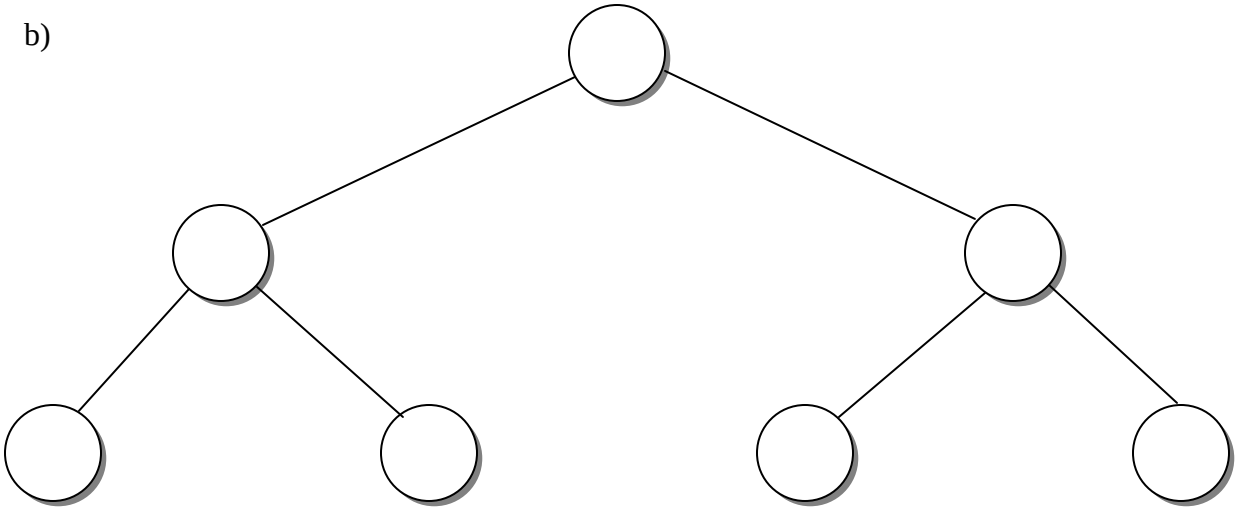
Nom : _____ Matricule : _____

Tableau 2.2.1

a)



b)



Nom : _____ Matricule : _____

Tableau 2.2.1 (suite)

c)

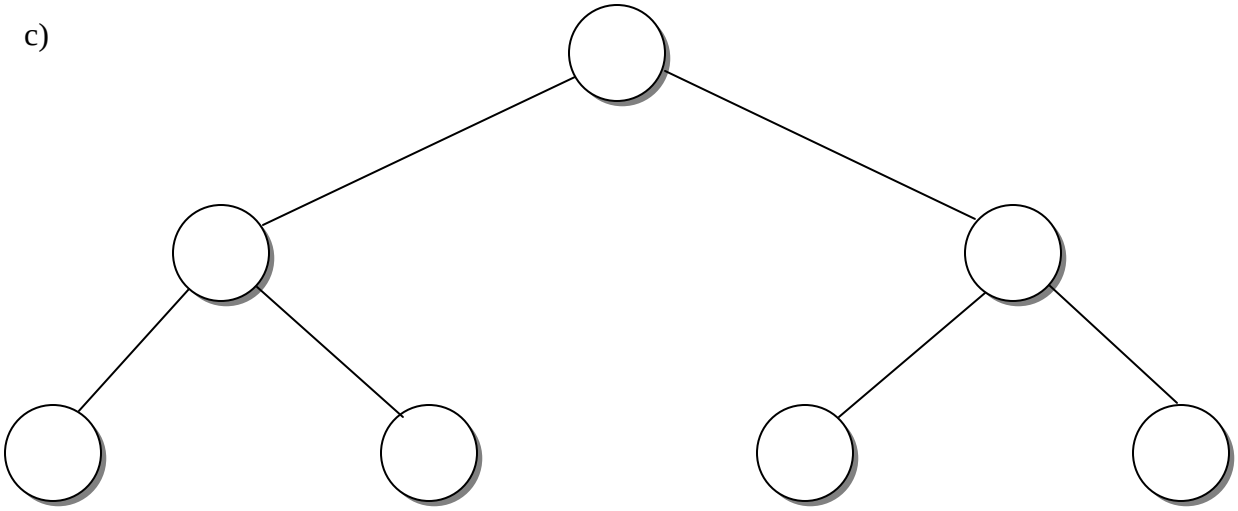


Tableau 2.2.2

a)

0	1	2	3	4	5	6	7	8

b)

0	1	2	3	4	5	6	7	8

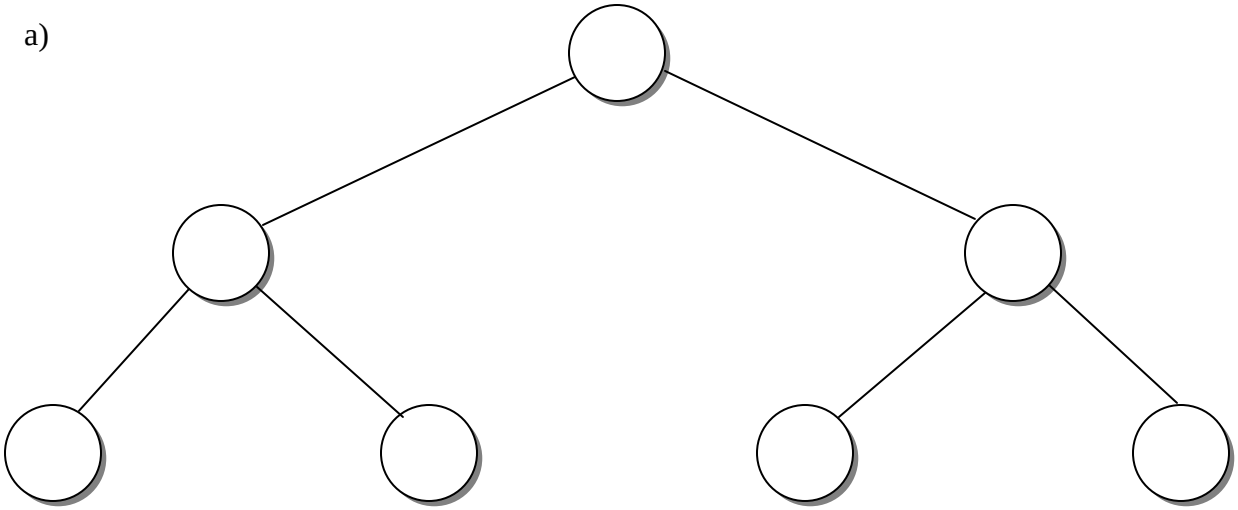
c)

0	1	2	3	4	5	6	7	8

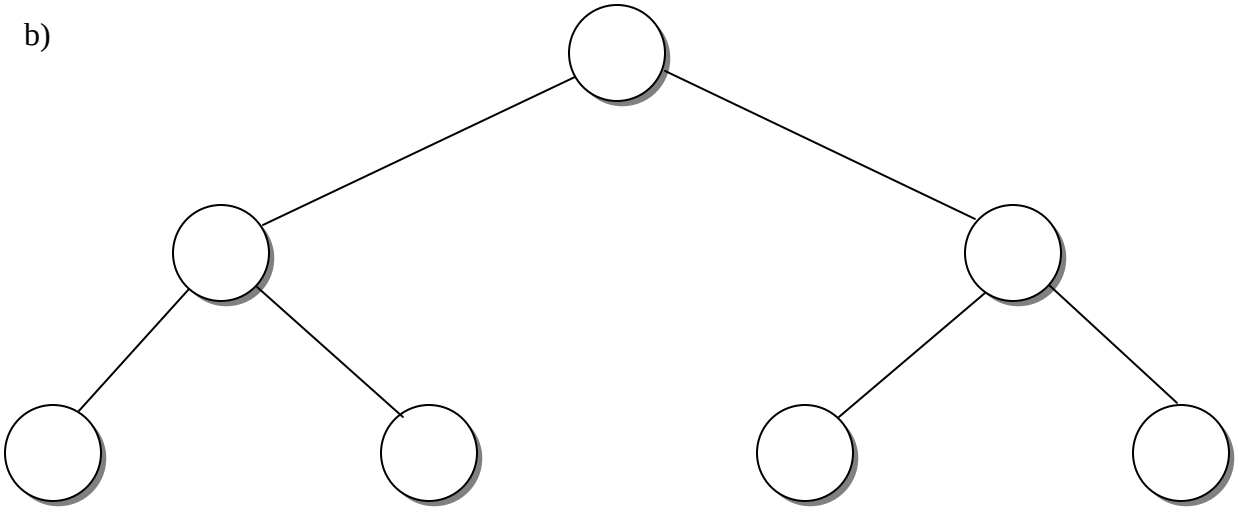
Nom : _____ Matricule : _____

Tableau 2.3.1

a)



b)



Nom : _____ Matricule : _____

Tableau 2.3.1 (suite)

c)

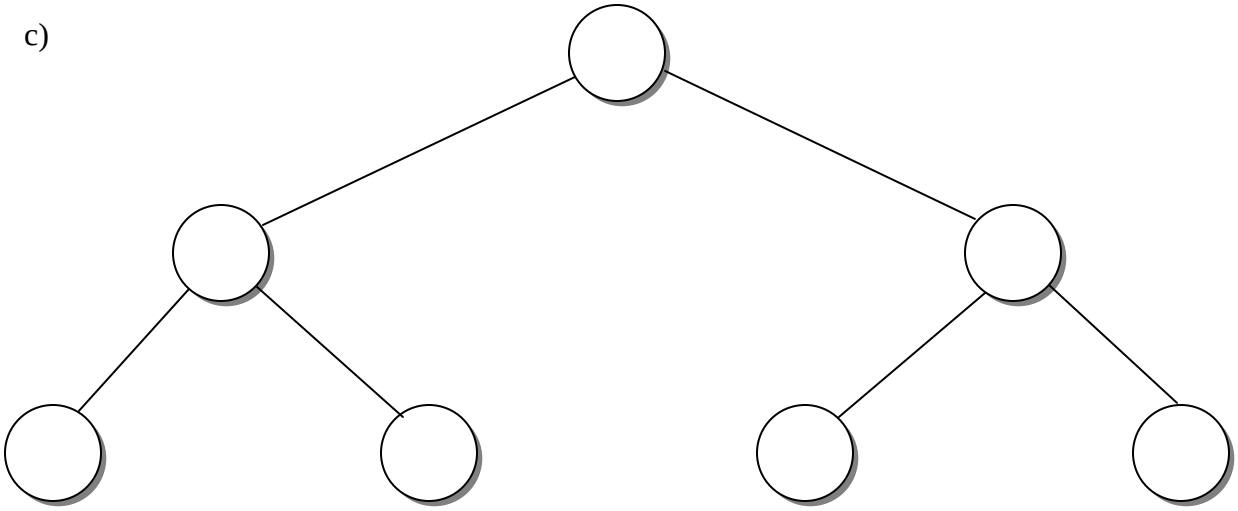


Tableau 2.3.2

a)

0	1	2	3	4	5	6	7	8

b)

0	1	2	3	4	5	6	7	8

c)

0	1	2	3	4	5	6	7	8

Nom : _____ Matricule : _____

Figure 3.1

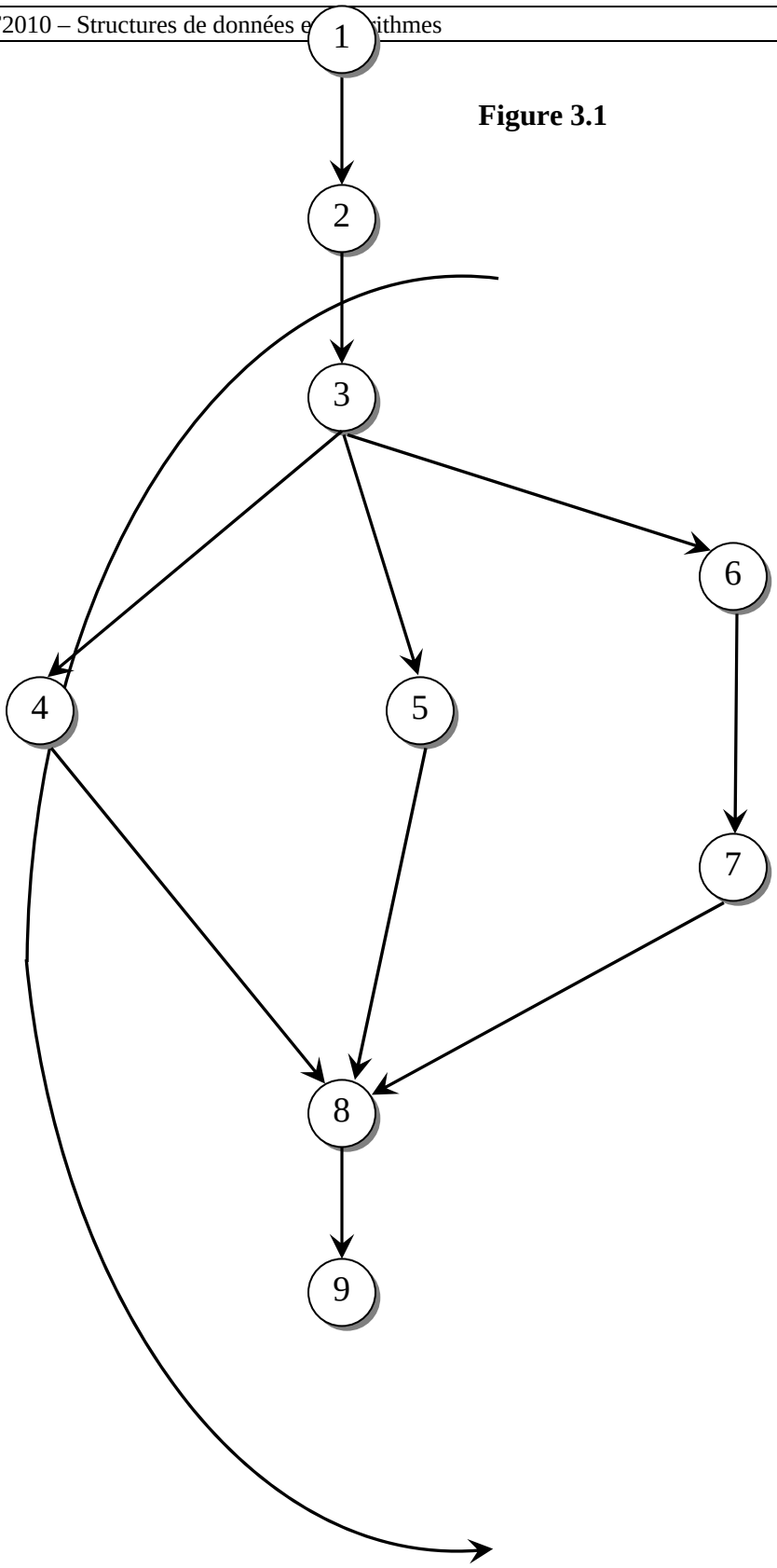


Tableau 3.1.1

ID	Chemin								
1	9	8	7	6	5	4	3	2	1
2	1	9	2	3	4	8	5	6	7
3	1	2	3	4	8	5	6	7	9
4	1	2	3	6	5	4	7	8	9
5	9	1	3	8	4	7	5	6	2
6	1	2	3	4	5	6	7	8	9
7	1	9	2	3	4	5	6	8	7
8	9	8	4	5	7	6	3	2	1

Tableau 3.1.2

ID	DFS	BFS	Topologique ascendant	Topologique descendant	Aucun
1					
2					
3					
4					
5					
6					
7					
8					

Nom : _____ Matricule : _____

Tableau 4.1

Q	
Q_o	
F	

Fonction de transition δ

Symbole de l'alphabet

État

0				
1				
2				
3				
4				
5				
6				
7				
8				

Tableau 4.2

0	1	2	3	4	5	6	7	8

Décalages identifiées

Nom : _____ Matricule : _____

Annexe 2.1

```
import java.io.*;
import java.util.*;
import java.lang.*;

// BinaryHeap class
//
// CONSTRUCTION: with optional capacity (that defaults to 100)
//               or an array containing initial items
//
// ******************PUBLIC OPERATIONS*********************
// void insert( x )       --> Insert x
// Comparable deleteMin( )--> Return and remove smallest item
// Comparable findMin( )  --> Return smallest item
// boolean isEmpty( )     --> Return true if empty; else false
// void makeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// Throws UnderflowException as appropriate

/**
 * Implements a binary heap.
 * Note that all "matching" is based on the compareTo method.
 * @author Mark Allen Weiss
 */
public class BinaryHeap<AnyType extends Comparable<? super AnyType>>
{
    /**
     * Construct the binary heap.
     */
    public BinaryHeap( )
    {
        this( DEFAULT_CAPACITY );
    }

    /**
     * Construct the binary heap.
     * @param capacity the capacity of the binary heap.
     */
    public BinaryHeap( int capacity )
    {
        currentSize = 0;
        array = (AnyType[]) new Comparable[ capacity + 1 ];
    }

    /**
     * Construct the binary heap given an array of items.
     */
    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( );
}

/**
 * Insert into the priority queue, maintaining heap order.
 * Duplicates are allowed.
 * @param x the item to insert.
 */
public void insert( AnyType x )
{
    if( currentSize == array.length - 1 )
        enlargeArray( array.length * 2 + 1 );

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

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 ];
}

/**
 * Find the smallest item in the priority queue.
 * @return the smallest item, or throw an UnderflowException if empty.
 */
public AnyType findMin( )
{
    if( isEmpty( ) )
        //throw new UnderflowException( );
        System.exit(1);
    return array[ 1 ];
}

/**
 * Remove the smallest item from the priority queue.
 * @return the smallest item, or throw an UnderflowException if empty.
 */
public AnyType deleteMin( )
```

```
{
    if( isEmpty( ) )
        //throw new UnderflowException( );
        System.exit(1);

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

    return minItem;
}

/**
 * Establish heap order property from an arbitrary
 * arrangement of items. Runs in linear time.
 */
private void buildHeap( )
{
    for( int i = currentSize / 2; i > 0; i-- )
        percolateDown( i );
}

/**
 * Test if the priority queue is logically empty.
 * @return true if empty, false otherwise.
 */
public boolean isEmpty( )
{
    return currentSize == 0;
}

/**
 * Make the priority queue logically empty.
 */
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

/**
 * Internal method to percolate down in the heap.
 * @param hole the index at which the percolate begins.
 */
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;
}

public void print() {

    int i = 0;

    for (i=1; i <= currentSize; i++) {
        if (array[i] != null)
            System.out.println("POS: " + i + " VAL: " + array[i]);
    }
    System.out.println();
}

// Test program
public static void main( String [ ] args )
{
    int iVal = 0;

    BinaryHeap<Integer> h = new BinaryHeap<Integer>( );

    iVal = 5;
    h.insert(iVal);
    h.print(); // print 1

    iVal = 7;
    h.insert(iVal);
    h.print(); // print 2

    iVal = 1;
    h.insert(iVal);
    h.print(); // print 3

    iVal = 10;
    h.insert(iVal);
    h.print(); // print 4

    iVal = 6;
    h.insert(iVal);
    h.print(); // print 5
```

```
        iVal = 12;
        h.insert(iVal);
        h.print(); // print 6

        iVal = 3;
        h.insert(iVal);
        h.print(); // print 7

        Integer [] v = { 45, 68, 12, 99, 57, 123, 34, 81};

        h = new BinaryHeap<Integer>( v );
        h.print(); // print 8

        System.out.println("MIN: " + h.deleteMin());
        h.print(); // print 9

        System.out.println("MIN: " + h.deleteMin());
        h.print(); // print 10

        System.out.println("MIN: " + h.deleteMin());
        h.print(); // print 11
    }
}
```

Annexe 3.2

```
import java.io.*;
import java.util.*;

class adj {
    static final int UNDEF_VAL = -9999;

    ArrayList<graphNode> adjList = new ArrayList<graphNode>();
    //ArrayIterator<graphNode> listIt = null;
    //graphNode curItem = null;
    int curIndex = UNDEF_VAL;

    void first() {
        curIndex = 0;
    };

    boolean currentIsValid() {
        return((curIndex >= 0) && (curIndex < adjList.size()));
    };

    graphNode getCurrent() {
        if ((curIndex >= 0) && (curIndex < adjList.size()))
            return(adjList.get(curIndex));
        else
            return(null);
    };

    void next() {
        if ((curIndex >= 0) && (curIndex < (adjList.size() - 1)))
            curIndex++;
        else
            curIndex = UNDEF_VAL;
    };

    void add(graphNode node) {
        adjList.add(node);
    }
};
```

```
import java.io.*;
import java.util.*;

class graph {

    static final int UNDEF_VAL = -9999;

    ArrayList<graphNode> nodeArr = new ArrayList<graphNode>();
    int curIndex = UNDEF_VAL;

    void initNodes() {
        int i = UNDEF_VAL;

        for (i = 0; i < nodeArr.size(); i++) {
            nodeArr.get(i).init();
        }
    }

    void first() {
        curIndex = 0;
    };

    boolean currentIsValid() {
        return((curIndex >= 0) && (curIndex < nodeArr.size()));
    };

    graphNode getCurrent() {
        if ((curIndex >= 0) && (curIndex < nodeArr.size()))
            return(nodeArr.get(curIndex));
        else
            return(null);
    };

    graphNode getNode(int pos) {
        if ((pos >= 0) && (pos < nodeArr.size()))
            return(nodeArr.get(pos));
        else
            return(null);
    };

    void next() {
        if (curIndex < (nodeArr.size() - 1))
            curIndex++;
        else
            curIndex = UNDEF_VAL;
    };


    int size() {
        return(nodeArr.size());
    }
}
```

```
class graphNode {

    static final int UNDEF_VAL = -9999;

    int nodeId;
    int color;
    int dtime;
    int ftime;
    int componentId;
    graphNode pred;
    adj kids = new adj();
    adj parents = new adj();

    graphNode() {
        nodeId = UNDEF_VAL;
        color = UNDEF_VAL;
        dtime = UNDEF_VAL;
        ftime = UNDEF_VAL;
        componentId = UNDEF_VAL;
        pred = null;
    }

    void init() {
        color = UNDEF_VAL;
        dtime = UNDEF_VAL;
        ftime = UNDEF_VAL;
        componentId = UNDEF_VAL;
        pred = null;
    }

    void addKid (graphNode node) {
        kids.add(node);
    }

    void addParent (graphNode node) {
        parents.add(node);
    }
};
```