AUCUNE DOCUMENTATION PERMISE

(80) 1. Vous devez écrire la classe Java `Astm` qui va afficher un réseau composé de noeuds situés dans le plan avec des arcs entre certains des noeuds. La valeur sur un arc représente la distance associée à cet arc. Vous allez ensuite afficher ce réseau dans une fenêtre (figure de gauche ci-dessous) avec un bouton qui va lancer le calcul d'un *arbre sous-tendant minimum* et afficher ce dernier (figure de droite ci-dessous). Un *arbre sous-tendant* est un ensemble d'arcs qui touche à tous les noeuds sans permettre plus d'un chemin entre deux noeuds. L'*arbre sous-tendant minimum* est un arbre dont le coût (la somme des distances sur les arcs) est minimum parmi tous les arbres sous-tendant possibles.

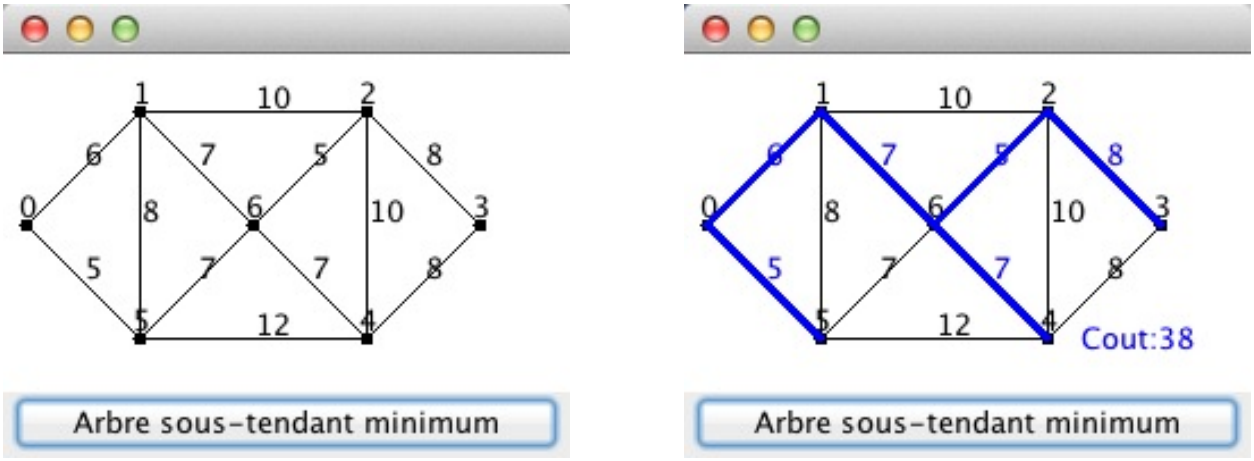


Figure 1: Affichage initial du réseau et d'un arbre sous-tendant minimum

Vous allez développer cette application en étapes. Vous pouvez répondre à une étape en supposant qu'une réponse correcte vous a été fournie pour les étapes précédentes.

Le graphe comporte N noeuds, chaque noeud est étiqueté par un entier entre 0 et $N-1$. Un noeud est représenté par une instance de la classe `Noeud` suivante:

```
public class Noeud {
    private final int RAYON=6;
    private final int R2=RAYON/2;
    private int etiq,x,y;

    Noeud(int etiq,int x, int y){
        this.etiq=etiq;
        this.x=x;
        this.y=y;
    }

    public String toString(){
        return String.format("%s@%d,%d",etiq,x,y);
```

```
    }

    public void afficher(Graphics g){
        g.fillOval(x-R2,y-R2,RAYON,RAYON);
        g.drawString(""+etiq,x-R2,y-R2);
    }

    public int getX(){return x;}
    public int getY(){return y;}
    public int getEtiq(){return etiq;}
}
```

(20) (a) Créer la classe `Arc` pour représenter un arc entre deux noeuds du réseau avec une distance. Son constructeur devrait avoir l'entête suivant:

`Arc(Noeud origine, Noeud destination, int distance)`

Dans la classe `Arc`, définissez les méthodes pour permettre la *comparaison naturelle* en ordre croissant qui devrait se faire d'abord sur la distance et, en cas d'égalité, sur l'étiquette de l'origine et finalement sur l'étiquette de destination.

Il faut aussi y définir la méthode suivante:

`public void afficher(Graphics g)`

qui dessine une ligne entre l'origine et la destination et affiche la distance au milieu de l'arc (c'est-à-dire à la moyenne des coordonnées de l'origine et de la destination).

Pour la suite, vous pouvez supposer que les déclarations et initialisations des collections suivantes sont accessibles partout:

```
List<Noeud> noeuds = new ArrayList<Noeud>();
SortedSet<Arc> arcs= new TreeSet<Arc>();
Set<Arc> astm      = new HashSet<Arc>();
```

(20) (b) Écrivez la méthode

```
void lireArcsNoeuds(String fileName)
```

qui lit le fichier désigné par `fileName` composé d'une série de lignes avec deux nombres qui correspondent aux coordonnées `x` et `y` de chaque noeud d'indice $[0, N)$, la fin de la définition des noeuds est indiquée par une ligne vide. Ensuite, viennent une série de lignes de trois nombres décrivant les arcs: les numéros des noeuds d'origine et de destination et la distance entre ces deux noeuds. Vous n'avez pas à valider les valeurs lues sur le fichier. L'appendice (page 9) donne quelques rappels sur les opérations d'entrée-sortie et indique le contenu du fichier correspondant au réseau des figures ci-dessus.

(10) (c) Écrivez la méthode

```
int cout(Set<Arc>arcs)
```

qui calcule la somme des distances des arcs d'un ensemble d'arcs reçu en paramètre.

(10) (d) Complétez la classe suivante qui affiche dans un JPanel les `noeuds`, les `arcs` (figure de gauche ci-dessus). Si l'arbre sous-tendant a été calculé dans la variable `astm`, afficher ses arcs en bleu ainsi que son coût dans le coin inférieur droit. Dans la partie droite de la figure 1, les arcs ont été dessinés en **gras**, mais il n'est pas nécessaire pour vous de le faire.

```
class STPanel extends JPanel {
... à compléter ...
}
```

(10) (e) Implantez la méthode suivante

```
Set<Arc> prim(List<Noeud> noeuds, SortedSet<Arc> arcs)
```

pour calculer l'arbre sous-tendant en utilisant l'*algorithme de Prim* qui prend en entrée un ensemble d'arcs triés en ordre croissant selon leur distance et qui retourne l'ensemble des arcs qui forme l'arbre sous-tendant minimal de la façon suivante:

- Initialiser V à l'ensemble de tous les noeuds;
- Créer T, l'ensemble de noeuds dans l'astm, initialement vide
- Créer OUT, l'ensemble d'arcs résultat, initialement vide
- Enlever un noeud de V et le placer dans T
- Tant que la cardinalité de T est inférieure à N
  - Pour chaque arc a
    - si a est un arc avec un noeud n dans V et l'autre dans T
      - enlever n de V pour le placer dans T
      - ajouter a à OUT
- retourner OUT

(10) (f) Complétez la classe `Astm` et sa méthode `main` suivante pour avoir une application Java qui permettra d'afficher le réseau et son arbre sous-tendant minimum. Vous n'avez pas à répéter le code écrit aux étapes précédentes.

```
public static void main(String[] args) {
    JFrame f = new JFrame();
    ... à compléter ...
    f.setSize(300,200);
    f.setLocation(100,100);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

(10) 2. Mathieu Rouleau dans sa conférence a présenté les défis, les satisfactions et les frustrations d'un jeune entrepreneur. Présentez ce qui vous la plus impressionné dans son parcours et ce que vous en retenez pour la suite de vos études.

(10) 3. Écrire la méthode `xref` avec l'entête suivante

```
void xref(List<String> lines)
```

qui reçoit une liste de chaînes et qui imprime la fréquence de chaque mot mais aussi les indices des chaînes de la liste dans lesquelles il apparaît. Par exemple, l'appel suivant:

```
xref(Arrays.asList("je suis le chef","et tu suis","le grand chef"));
```

imprimera les lignes suivantes:

```
chef:2 => [0, 2]
et:1 => [1]
grand:1 => [2]
je:1 => [0]
le:2 => [0, 2]
suis:2 => [0, 1]
tu:1 => [1]
```

Indice: pensez à utiliser une instance de `Scanner` (page 9) pour isoler les mots dans une chaîne.

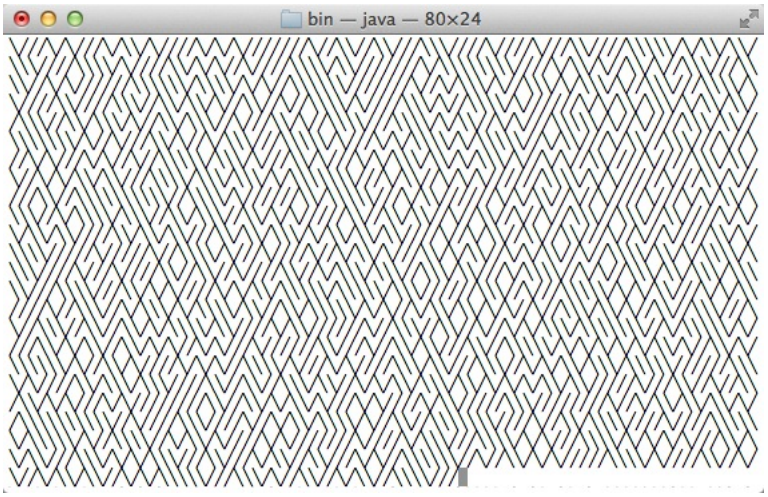
Cet examen comporte 3 questions pour un total de 100 points.

Bonne chance

(2) 4. **Question bonus**.

```
10 PRINT CHR$(205.5+RND(1)); :GOTO 10
```

est un programme BASIC d'une seule ligne qui, sur un Commodore 64 au début des années 80, produisait un motif de labyrinthe semblable à celui de la figure suivante.



Ce programme produit une boucle infinie, il faut donc forcer son arrêt avec `Control-C` par exemple.

Donnez une version Java de ce programme en utilisant les caractères Unicode de diagonales dont les codes sont les entiers 9585 et 9586 qui correspondent aux caractères de codes 205 et 206 sur le Commodore 64. Le corps de la méthode **main** ne devrait comporter qu'une seule ligne.

Pour votre information: ce programme BASIC est aussi le titre d'un livre de 309 pages, une collaboration de 10 auteurs sous la direction de Nick Montfort, publié en 2013 par MIT Press qui l'annnonce comme suit:

> *This book takes a single line of code – the extremely concise BASIC program for the commodore 64 inscribed in the title – and uses it as a lens through which to consider the phenomenon of creative computing and the way computer programs exists in culture.*

Rappel d'éléments du *Collection Framework* de Java

```
public interface Collection<E> extends Iterable<E>{
    // Opérations de base
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);
    boolean remove(Object element);
    Iterator<E> iterator();

    // Opérations de groupe
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();
}

public interface Iterator<E> {
    boolean hasNext();
    <E> next();
    void remove();
}

public interface Comparable<T> {
    public int compareTo(T o);
}

public interface Comparator<T> {
    public int compare(T o1, T o2);
}

public interface List<E> extends Collection<E> {
    // accès par position
    E get(int index);
    E set(int index, E element);
    void add(int index, E element);
    E remove(int index);
    boolean addAll(int index, Collection <? extends <E>>);
    // recherche
    int indexOf(Object o);
    int lastIndexOf(Object o);
    //Exemples de création
    //List<String> l = new ArrayList<String>();
    //List<Integer> l = new LinkedList<Integer>();
}
```

```
public interface Set<E> extends Collection<E>{
    // aucune nouvelle méthode
    // mais s'assure que tous les éléments sont distincts
    // Exemple de création
    // Set<String> s = new HashSet<String>();
}

public interface Map<K,V> {
    // Opérations de base
    V put(K key, V value);
    V get(K key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();
    //  Opérations de groupe
    void putAll(Map<? Extends K, ? extends V> m);
    void clear();
    //  Vues comme des Collections
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();
    // Interface pour les entrées de la table (entrySet)
    //      on la désigne par Map.Entry
    public interface Entry<K,V> {
        K getKey();
        V getValue();
        V setValue(V value);
    }
    //Création
    //Map<String,Integer> m = new HashMap<String,Integer>();
    //Map<String,Integer> m = new LinkedHashMap<String,Integer>();
}

public interface SortedSet<E> extends Set<E> {
    // vue comme sous-ensemble
    SortedSet<E> subSet(Object fromElement, Object toElement);
    SortedSet<E> headSet(Object toElement);
    SortedSet<E> tailSet(Object fromElement);

    // points extrêmes
    E first();
    E last();
```

```
    // accès au comparateur
    Comparator<? super E> comparator();
    //Création
    //SortedSet<String> ss = new TreeSet<String>();
}

public interface SortedMap<K,V> extends Map<K,V> {

    // vue comme sous-table
    SortedMap<K,V> subMap(K fromKey, K toKey);
    SortedMap<K,V> headMap(K toKey);
    SortedMap<K,V> tailMap(K fromKey);

    // accès aux clés extrêmes
    K firstKey();
    K lastKey();

    // accès au comparateur
    Comparator<? super K> comparator();
    //Création
    //SortedMap<String,Integer> sm = new TreeMap<String,Integer>();
}
```

Rappel sur la lecture de fichiers

Entrées-sorties `BufferedReader in = new BufferedReader(new FileReader(fileName));`
`BufferedReader` contient la méthode `readLine()` qui retourne une ligne comme un `String` ou `null` lorsqu'on est à la fin du fichier.

Tokenization `Scanner` constructeur `Scanner(String ligne)`
`Scanner` comprend les méthodes suivantes:

`hasNext()` qui retourne `true` s'il reste des tokens à traiter sur `ligne`

`next()` qui retourne le prochain token de `ligne` comme une `String`

`nextInt()` qui retourne le prochain token de `ligne` comme un `int`

Exceptions `FileReader` peut lever `FileNotFoundException` et `BufferedReader` peut lever `IOException` qui doivent être traitées en indiquant le nom du fichier où s'est produit l'exception sur `System.err`.

Contenu du fichier décrivant le réseau
Le contenu du fichier correspondant à la Figure 1 est le suivant:

```
10 75
60 25
160 25
210 75
160 125
60 125
110 75

0 1 6
0 5 5
1 2 10
1 6 7
1 5 8
2 6 5
2 3 8
2 4 10
3 4 8
4 6 7
4 5 12
5 6 7
```