Séquence 10 : Algorithme et programmation

I. Etude d'algorithmes simples

I.1. Rechercher un élément dans un tableau trié par une méthode dichotomique

Si le tableau est trié par ordre croissant, on peut s'arrêter dès que la valeur rencontrée est plus grande que la valeur cherchée (ou plus petite si le tableau est trié par ordre décroissant).

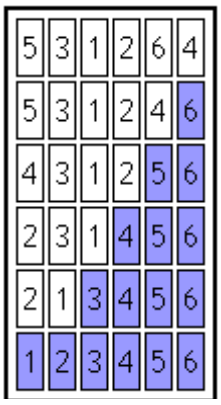
Algorithme :

```
// cette fonction renvoie vrai si x est présente dans tab, faux sinon
// le tableau tab est supposé trié par ordre croissant
        fonction avec retour booléen rechercheElement2(chaine[] tab, chaine x)
                entier i;
                début
                i <- 0;
                tantque (i < tab.longueur) faire
                        si (tab[i] = x) alors
                                retourne VRAI;
                                sinon
                                        si (tab[i] > x) alors
                                                retourne FAUX;
                                                sinon
                                                i <- i + 1;
                                        finsi
                        finsi
                fintantque
                retourne FAUX;
        fin
```

→ Programmer cette fonction en AS3.

I.2. Le tri d'un tableau par sélection

Le principe est que pour classer n valeurs, il faut rechercher la plus grande valeur et la placer en fin de liste, puis la plus grande valeur dans les valeurs restante et la placer en avant dernière position et ainsi de suite...
Considérons un tableau à n éléments. Pour effectuer le tri par sélection, il faut rechercher dans ce tableau la position du plus grand élément. Le plus grand élément est alors échangé avec le dernier élément du tableau. Ensuite, on réitère l'algorithme sur le tableau constitué par les (n-p) premiers éléments où p est le nombre de fois où l'algorithme a été itéré.
L'algorithme se termine quand p=(n-1), c'est à dire quand il n'y a plus qu'une valeur à sélectionner ; celle ci est alors la plus petite valeur du tableau.



Exemple : Grandes étapes de l'évolution du tableau au fil de l'algorithme. En bleu, les valeurs déjà traitées.

Algorithme :

```
procedure tri_selection(tableau T)
   debut
      entier longueur, maxi, i
      longueur<-taille(T)

      tantque(longueur>0) faire
         //recherche de la position du plus grand élément dans le tableau non encore trié
         maxi<-0;

         pour i=1 à (longueur-1) faire
            si T(i)>T(maxi) alors
               maxi<-i
            fin si
         fin pour

         //echange du plus grand élément avec le dernier
         echanger(T,maxi,longueur-1)

         //traitement du reste du tableau
         longueur<-longueur-1
      fin tantque
fin
```

→ Programmer cette fonction en AS3.

Remarque : Le tri sélection peut être amélioré en positionnant à chaque parcours du tableau le plus grand et le plus petit élément.

I.3. Ajouter deux entiers exprimés en binaire

Exemple : 1010
+1011

→ Ecrire l'algorithme pour résoudre cette opération :

→ Programmer cette fonction en AS3.

II.Etude d'algorithmes plus avancés

II.1. Tri d'un tableau par fusion

Principe

Le tri fusion est construit suivant la stratégie "diviser pour régner", en anglais "divide and conquer". Le principe de base de la stratégie "diviser pour régner" est que pour résoudre un gros problème, il est souvent plus facile de le diviser en petits problèmes élémentaires. Une fois chaque petit problème résolu, il n'y a plus qu'à combiner les différentes solutions pour résoudre le problème global. La méthode "diviser pour régner" est tout à fait applicable au problème de tri : plutôt que de trier le tableau complet, il est préférable de trier deux sous tableaux de taille égale, puis de fusionner les résultats.
L'algorithme proposé ici est récursif. En effet, les deux sous tableaux seront eux même triés à l'aide de l'algorithme de tri fusion. Un tableau ne comportant qu'un seul élément sera considéré comme trié : c'est la condition sine qua non sans laquelle l'algorithme n'aurais pas de conditions d'arrêt. Etapes de l'algorithme :

- Division de l'ensemble de valeurs en deux parties
- Tri de chacun des deux ensembles
- Fusion des deux ensembles

Exemple

Grandes étapes de l'évolution du tableau au fil de l'algorithme :

Tableau	Commentaire : ce qui est fait pour passer à la ligne suivante
6 3 0 9 1 7 8 2 5 4	Division du tableau en deux parties
6 3 0 9 1 7 8 2 5 4	Division de chaque sous tableau en deux parties
6 3 0 9 1 7 8 2 5 4	Division de chaque sous tableau en deux parties
6 3 0 9 1 7 8 2 5 4	Division des sous tableau bleu et rouge en deux parties, fusion des tableaux vert-rose
6 3 0 1 9 7 8 2 4 5	Fusion des sous tableaux bleu-rose et rouge-rose
3 6 0 1 9 7 8 2 4 5	Fusion des sous tableaux bleu-jaune et rouge-jaune
0 3 6 1 9 2 7 8 4 5	Fusion des sous tableaux bleu-vert et rouge-vert
0 1 3 6 9 2 4 5 7 8	Fusion des deux tableaux
0 1 2 3 4 5 6 7 8 9	Le tableau est trié, l'algorithme est terminé

Algorithme :

```
fusion(tableau T,entier premier1,entier dernier1,entier dernier2)
  debut
    tableau Tbis
    entier premier2, compteur1, compteur2, i

    premier2<-dernier1+1
    compteur1<-premier1
    compteur2<-premier2

    taille(T)<-(dernier1-premier1+1) //taille du premier sous tableau
```

```
    //on recopie dans T les éléments du premier sous tableau
    pour i=premier1 à dernier1 faire
      Tbis(i-premier1)<-T(i)
    fin pour

    //on fusionne ensuite les deux sous tableaux
    pour i=premier1 à dernier2 faire
      si compteur1=premier2 alors //tous les éléments du premier sous
tableau ont été utilisés
        arret pour //tous les éléments sont donc classés
      sinon si compteur2=(dernier2+1) alors //tous les éléments du second
sous tableau ont été utilisés
        T(i)=Tbis(compteur1-premier1) //on recopie à la fin du tableau
les éléments du premier sous tableau
        compteur1<-compteur1+1
      sinon si Tbis(compteur1-premier1)<T(compteur2) alors //l'élément du
premier sous tableau est le plus petit
        T(i)<-Tbis(compteur1-premier1) //on ajoute un élémnt du premier
sous tableau
        compteur1<-compteur1+1 //on progresse dans le premier sous
tableau
      sinon //c'est l'élément du second sous tableau qui est le plus
petit
        T(i)<-T(compteur2) //on recopie cette élément à la suite du
tableau
        compteur2<-compteur2+1 //on progresse dans le second tableau
      fin si
    fin pour
  fin

tri_fusion_bis(tableau T,entier premier,entier dernier)
  debut
    entier milieu;

    si premier<>dernier alors
      //si l'indice du premier et du dernier élément à traiter
      //est différent (condition d'arret de l'algorithme)
      milieu<-(premier+dernier)/2
      tri_fusion_bis(T,premier,milieu) //tri de la premiére moitiée du
sous tableau
      tri_fusion_bis(T,milieu+1,dernier) //tri de la seconde moitiée du
sous tableau
      fusion(T,premier,milieu,dernier) //fusion des deux sous moitiées
    fin si
  fin

tri_fusion(tableau T)
  debut
    entier longueur
    longueur<-taille(T)
```

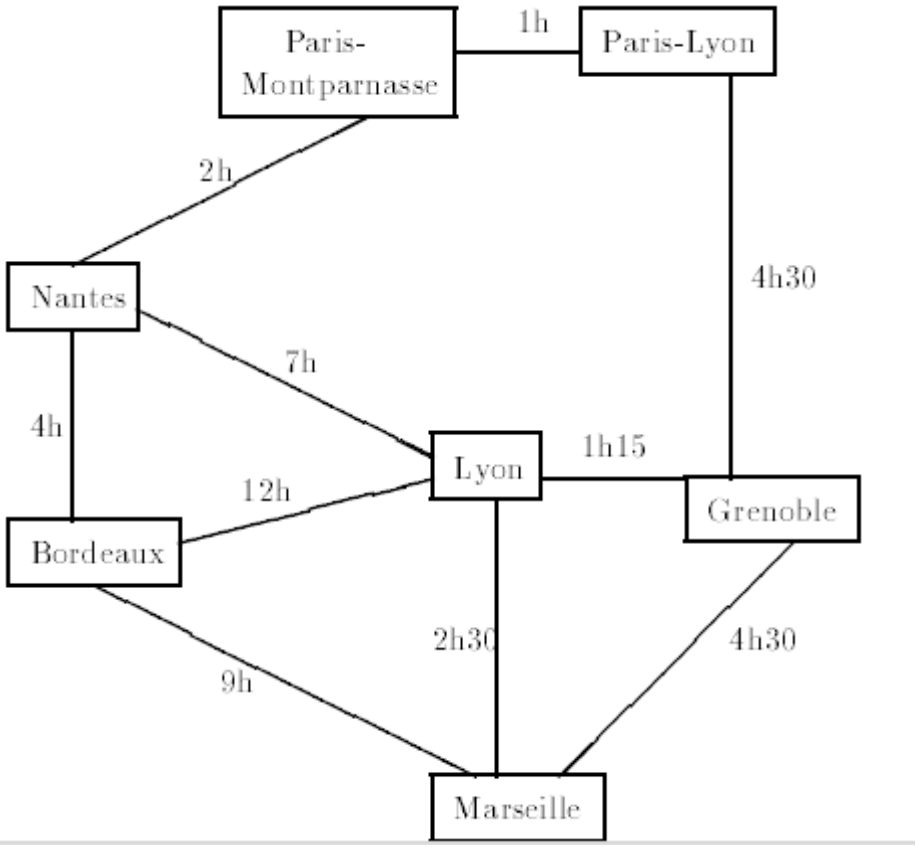
```
        si longueur>0 alors
          tri_fusion_bis(T,0,longueur-1)
        fin si
      fin
```

II.2.Notions élémentaires sur les graphes

Exemple 1 : Choix d'un itinéraire
Sachant qu'une manifestation d'etudiants bloque la gare de Poitiers, et connaissant la duree des trajets suivants :

Bordeaux ! Nantes 4 h
Bordeaux ! Marseille 9 h
Bordeaux ! Lyon 12 h
Nantes ! Paris-Montparnasse 2 h
Nantes ! Lyon 7 h
Paris-Montparnasse ! Paris-Lyon 1 h (en autobus)
Paris-Lyon ! Grenoble 4 h 30
Marseille ! Lyon 2 h 30
Marseille ! Grenoble 4 h 30
Lyon ! Grenoble 1 h 15

comment faire pour aller le plus rapidement possible de Bordeaux a Grenoble ?
Les donnees du probleme sont faciles a representer par un graphe dont les arêtes sont etiquetees par les durees des trajets :



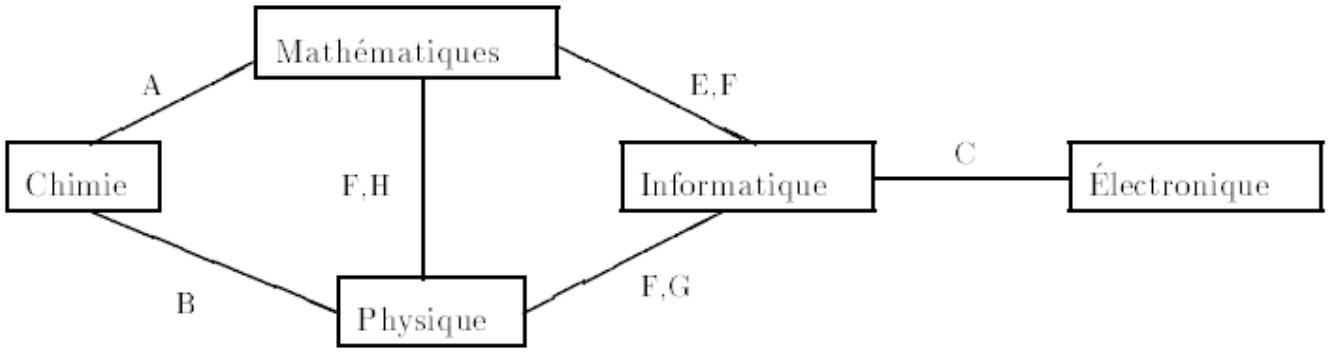
Il s'agit de determiner, dans ce graphe, le plus court chemin (ou l'un des plus courts chemins, s'il existe plusieurs solutions) entre Bordeaux et Grenoble.

Exemple 2 : Organisation d'une session d'examens
Des étudiants A, B, C, D, E et F doivent passer des examens dans différentes disciplines, chaque examen occupant une demi-journée :

Chimie : etudiants A et B
Electronique : etudiants C et D
Informatique : etudiants C, E, F et G
Mathematiques : etudiants A, E, F et H
Physique : etudiants B, F, G et H

On cherche a organiser la session d'examens la plus courte possible.

On peut representer chacune des disciplines par un sommet, et relier par des ar^etes les sommets correspondant aux examens incompatibles (ayant des etudiants en commun) :



Il s'agit alors de colorier chacun des sommets du graphe en utilisant le moins de couleurs possible, des sommets voisins (relies par une arête) etant necessairement de couleurs différentes.

II.3. Recherche d'un chemin dans un graphe par un parcours en profondeur (DFS)

Algorithme :

parcourir un graphe en profondeur :
    visiter tous les sommets
        visiter un sommet s :
        SI s n'a pas encore #et#e visit#e ALORS
            f pr#e-traitement de sg
            visiter tous les successeurs de s
            f post-traitement de sg
        FIN SI

II.4. Recherche d'un plus court chemin par un parcours en largeur (BFS).

Algorithme :

parcourir une arborescence en profondeur :
    visiter la racine
        visiter un noeud n :

Source : http://phamdavid.free.fr/cours/maths_graphes_algorithmes.pdf (81 pages)

III.Conclusion