```
////////////////////////////////////////////////////////////////////////////
//////////////////////////////PageRank//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

import java.util.*;

import java.io.*;

import java.util.ArrayList;

import java.lang.*;

/**

 * Page Rank

 *

 * @author Antoine & Antonio

 * @version Avril 2016

 */

public class PageRank

{

   /**

    * Main

    */

   public static void main(String []args)

   {

      /* Initialisation de toutes les données nécéssaire*/


      String [] valeur_initial = initialisation();  //Paramètrage de la fonction

      double [][] matrice = TestReader.read_matrix(valeur_initial[0]);
//Création de la matrice du graphe dirigé

      double alpha = Double.parseDouble(valeur_initial[1]);    //coefficient
de téléportation

      double [] vector_xT = create_vector(matrice.length,1);  //vecteur xT

      double [][] matrice_proba_t = transition_matrix(matrice);   //matrice
de probabilité de transition


      double nombre_page = matrice.length;    //nombre de page de la
matrice

      double facteur = (1 - alpha);

      double [] vector_qT =
TestReader.read_vector(valeur_initial[2]);//{0.3,0,0.2,0,0.2,0,0.2,0.1,0,0,0}
;  //vecteur qT


      /* Implémentation de la formule */


      double [] terme_1;  //contenir le 1er terme

      double [] terme_2;  //contenir le 2eme terme

      double [] vector_PR = create_vector(matrice.length,0);    //contenir le
vecteur Page Rank

      int count = 0;

      while( condition(vector_xT, vector_PR) ) //Tant que la norme 2 est
supérieure à 10^-8

      {

         vector_PR = vector_xT;

         terme_1 = multiply(vector_xT , matrice_proba_t , alpha);
//Premier terme de l'algorithme

         terme_2 = multiply(vector_qT , facteur);    //Second terme de
l'algorithme

         vector_xT = sum(terme_1 , terme_2); //Résultat

         vector_xT = norme(vector_xT, sum(vector_xT));

         count++;

      }

      vector_PR = vector_xT;

      System.out.println("\nLe vecteur PageRank est : \n");

      print_a_vector(vector_PR);

      System.out.println("\nLa somme des probabilité du vecteur PageRank
est de "+sum(vector_PR));

      System.out.println("\nLe nombre d'itération est de "+count);

   }


   /**

    * METHOD NORME

    *

    * --> Calcule la norme du vecteur

    *

    * @param vect le vecteur

    * @param divise la somme des éléments du vecteur

    *

    * @return retourne la norme du vecteur

    */

   public static double [] norme (double [] vect, double divise)

   {

      for (int i = 0; i < vect.length ; i++)

      {

         vect[i] = vect[i]/divise;

      }

      return vect;

   }
```

```
/**
 * METHOD CONDITION
 *
 * --> Nous informe si nous avons passé la condition de 10^-8 pour la différence entre les deux vecteurs page rank
 *
 * @param first le nouveau vecteur
 * @param second l'ancien vecteur
 *
 * @return true si pas passée et false si oui
 */
public static boolean condition(double [] first, double [] second)
{
   double [] temp = minus(second, first);
   double acc = 0;
   for (int i = 0; i < temp.length ; i++)
   {
      acc = acc + (temp[i]*temp[i]);
   }
   acc = Math.sqrt(acc);
   return acc > 0.00000001;
}

/**
 * METHODE MINUS
 *
 * --> Calcule la différences de deux vecteurs
 *
 * @param  vector_1     Un vecteur de taille N
 * @param  vector_2     Un vecteur de taille N
 * @return     La différence entre les deux vecteurs
 */
public static double [] minus(double [] vector_1, double [] vector_2)
{
   for( int runner=0 ; runner< vector_1.length;runner++)
   {
      vector_1[runner] -= vector_2[runner];
   }
   return vector_1;
}

public static String [] initialisation()
{
   Scanner scan_1 = new Scanner( System.in );    //Nom du fichier contenant la matrice
   System.out.print("Entrez le nom du fichier contenant la matrice: ");
   String fichier = scan_1.nextLine();

   //Vecteur personalisation
   System.out.print("Entrez le nom du fichier contenant le vecteur de personalisation: ");
   String vector_personalisation = scan_1.nextLine();

   //Valeur de alpha
   System.out.print("Entrez la valeur du coefficient alpha: ");
   String alpha = scan_1.nextLine();
   scan_1.close();

   String [] valeurs_initial = { fichier , alpha , vector_personalisation }; //Stockage des informations

   return valeurs_initial;
}

//METHODES AVANCEE SUR MATRICES
/**
 * METHOD TRANSITION_MATRIX
 *
 *  --> Calcule la matrice de transition
 *
 * @param matrice Une matrice NxN
 * @return La matrice de transition NxN
 */
public static double [][] transition_matrix(double [][] matrice)
{
   return divise(matrice,degre(matrice)); //on divise chaque ligne de la matrice par son degré
}
```

```
//METHODES DE BASES POUR LES MATRICES

/**
 * METHODE SUM
 *
 * --> Calcule la somme de deux vecteurs
 *
 * @param  vector_1     Un vecteur de taille N
 * @param  vector_2     Un vecteur de taille N
 * @return     La somme des deux vecteurs
 */
public static double [] sum(double [] vector_1, double [] vector_2)
{
   for( int runner=0 ; runner < vector_1.length;runner++)
   {
      vector_1[runner] += vector_2[runner];
   }
   return vector_1;
}

/**
 * METHOD DIVISE
 *
 * @param matrice Une matrice NxN
 * @param vector Un vecteur de longueur N
 * @return Retourne la division entre la matrice et le vecteur sous forme de matrice NxN
 */
public static double [][] divise(double [][] matrice, double [] vector)
{
   double divide_by;
   for(int runner = 0 ; runner < matrice.length ; runner++)
   {
      divide_by = vector[runner];
      if(divide_by > 1) //Si ca vaut la peine de diviser par divide_by
      {
         for(int runner2 = 0 ; runner2 < matrice.length ; runner2++)
         {
            matrice[runner][runner2]=matrice[runner][runner2]/divide_by;
         }
      }
   }
   return matrice;
}

/**
 * METHOD TRANSPOSE
 *
 * --> Transpose une matrice NxN
 *
 * @param matrice La matrice que l'on veut transposer
 * @return La matrice transposée
 */
public static double [][] transpose(double [][] matrice)
{
   double [][] matrice_transpose = new double [matrice.length][matrice.length];

   for(int runner = 0 ; runner < matrice.length ; runner++) //Parcours
   {
      for(int runner2 = 0 ; runner2 < matrice.length ; runner2++) //Parcours
      {
         matrice_transpose[runner2][runner] = matrice[runner][runner2]; //Inversion
      }
   }

   return matrice_transpose;
}
```

```
  /**
   * METHOD MULTIPLY
   *
   * --> Multiplie une matrice avec un vecteur et un coefficient alpha
   *   -> Pour le premier terme de la formule
   *
   * @param vector Le vecteur xT
   * @param matrice La matrice de probabilité de transition
   * @param alpha Le coefficient alpha
   *
   * @return Le produit du vecteur, de la matrice et du coefficient alpha
   */
  public static double[] multiply(double[] vector, double[][] matrice,
double alpha)
  {
     double[] vector_x_matrice = new double [vector.length];
     for( int runner_1 = 0 ; runner_1 != matrice.length ; runner_1++ )
//Parcours
     {
        for( int runner_2 = 0 ; runner_2 != matrice.length ; runner_2++ )
//Parcours
        {
           vector_x_matrice[runner_1] += vector[runner_2] *
matrice[runner_2][runner_1];
        }
        if (alpha != 0)
        {
           vector_x_matrice[runner_1] =
vector_x_matrice[runner_1]*alpha;
        }
     }
     return vector_x_matrice;
  }

  /**
   * METHOD MULTIPLY
   *
   * --> Multiplie un vecteur avec un facteur
   *
   * @return Le produit du vecteur et du facteur
   */
  public static double[] multiply(double[] vector, double facteur)
  {
     for( int runner = 0 ; runner != vector.length ; runner++ ) //Parcours
     {
        vector[runner] = vector[runner]*facteur;
     }
     return vector;
  }

  /**
   * METHOD DEGRE
   *
   * --> Calcule le vecteur de degré de la matrice
   *
   * @param matrice La matrice a analyser
   * @return Un vecteur contenant les degrés des lignes de la matrice
   */
  public static double[] degre(double[][] matrice)
  {
     double[] matrice_degre=new double [matrice.length]; //On cree la
matrice qui contiendra les degrés
     for ( int runner_1=0 ; runner_1 < matrice.length ; runner_1++)
     {
        double sum=0; //variable qui stockera le degré de la ligne runner_1

        for ( int runner_2=0 ; runner_2 < matrice.length ; runner_2++)
        {
           sum+=matrice[runner_1][runner_2]; //Additionne la ligne
        }
        matrice_degre [runner_1] = sum; //Met le degré de la ligne N à la
place N du vecteur
     }
     return matrice_degre;
  }
```

```
//METHODES DIVERSE


/**
 * METHOD CREATE_VECTOR
 *
 * --> Crée un vecteur soit initialisé à 1 (1,1,...,1), ou soit avec un 1 et que des 0 (1,0,...,0)
 *
 * @param longueur la longueur du vecteur à creer
 * @param mode  détermine le mode que l'on veut utilisé
 *
 * @post retourne le vecteur de taille longueur et initialisé selon le mode
 */
public static double [] create_vector(int longueur, int mode)
{
   double [] vector = new double [longueur];
   for( int runner=0 ; runner < longueur ; runner++ )
   {
      if( mode == 1 ) //mode (1, 0, 0, ...)
      {
         if(runner == 0)
         {
            vector [runner]=1;
         }
         else
         {
            vector [runner]=0;
         }
      }
      else    //mode ( 1, 1, 1 ...)
      {
         vector [runner] = 1;
      }
   }
   return vector;
}


/**
 * METHOD SUM
 *
 * --> Calcule la somme des éléments du vecteurs
 *
 * @param vector    Le vecteur à sommer
 * @post retourne la somme des éléments du vecteur
 */
public static double sum(double[] vector)
{
   double sum=0;
   for (int runner=0;runner<vector.length; runner++)
   {
      sum+=vector[runner];
   }
   return sum;
}


//METHODES POUR IMPRIMER
/**
 * METHOD PRINT_A_MATRIX
 *
 * --> Imprime la matrice
 *
 * @param matrice La matrice à imprimer
 */
public static void print_a_matrix(double[][] matrice)
{
   for(int runner = 0 ; runner < matrice.length ; runner++) //Parcours
   {
      for(int runner2 = 0 ; runner2 < matrice.length ; runner2++) //Parcours
      {
         System.out.print(" " + matrice[runner][runner2]);
      }
      System.out.println("");
   }
   System.out.println("\n");
}
```

```
    /**
     * METHOD PRINT_A_VECTOR
     *
     * --> Imprime le vecteur
     *
     * @param vector Le vecteur à imprimer
     */
    public static void print_a_vector(double[] vector)
    {
        for(int runner = 0 ; runner < vector.length ; runner++) //Parcours
        {
            System.out.print(vector[runner]+" ");
        }
        System.out.println(" ");
    }


}
```

```
////////////////////////////////////////////////////////////////////////////
//////////////////////////////TestReader////////////////////////////////////
////////////////////////////////////////////////////////////////////////////

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

/**

 * Classe de lectures

 *

 * @author decarvalhobo

 * @version April 2016

 */

public class TestReader

{

   public static int length; //longueur de la matrice

   /**

    * METHOD READ_MATRIX

    *

    * --> Lis le fichier et crée la matrice

    *

    * @param file fichier contenant la matrice

    */

   public static double [][] read_matrix(String file )

   {

      ArrayList <String[]> list = new ArrayList<String[]>(); //Liste dynamique qui contiendra les elements de la matrices

      BufferedReader fichier = null;

      try

      {

         fichier = new BufferedReader(new FileReader(file));  //Ouverture du flux de fichier

         String ligne = fichier.readLine();  //Lecture de la premiere ligne du fichier

         while( ligne != null )

         {

            list.add(ligne.split(",")); //On prend chaque élément séparé par une virgule

            ligne = fichier.readLine(); //On lit la ligne suivante du fichier

         }

      }

      catch (FileNotFoundException exception) //Gestion de l'erreur du au fait de ne pas trouver le fichier

      {

         System.err.println("Le fichier " + file +" n'a pas été trouvé :" + exception.getMessage());

      }

      catch (IOException exception)   //Gestion de l'erreur du à la lecture du fichier

      {

         System.err.println ("Erreur lors de la lecture : " + exception.getMessage());

      }

      finally //Exécuté dans tout les cas

      {

         try

         {

            fichier.close();    //Fermeture du flux de fichier

         }

         catch(Exception exception)    //Gestion de l'erreur du à la fermeture du flux de fichier

         {

            System.err.println ("Erreur lors de la fermeture du flux de fichier : " + exception.getMessage());

         }

      }

      String [][] temporaire = list.toArray(new String[list.size()][]); //Resultat de la lecture dans une matrice temporaire

      if (temporaire.length != temporaire[0].length)  //Gestion erreur matrice non NxN

      {

         System.err.println("La largeur de la matrice n'est pas égale à la hauteur. Nous ne garantissons pas le bon fonctionement du programme.\n ATTENTION : Une matrice pageRank est de type NxN");

      }



      /* Construction de la matrice */

      double [][] matrice = new double[temporaire.length][temporaire.length];    //Création de la matrice

      for (int runner_1 = 0 ; runner_1 < temporaire.length ; runner_1++) //Parcours du tableau

      {

         for (int runner_2 = 0 ; runner_2 < temporaire.length ; runner_2++)

         {

            matrice[runner_1][runner_2] = Double.parseDouble(temporaire[runner_1][runner_2]); //Conversion et stockage

         }

      }

      length = matrice.length;

      System.out.println("\nLe fichier " + file + " contient la matrice suivante : \n");
```

```
    PageRank.print_a_matrix(matrice);
    return matrice;
  }

  /**
   * METHOD READ_VECTOR
   *
   * --> Lis le fichier et crée le vecteur
   *
   * @param file fichier contenant le vecteur
   */
  public static double [] read_vector(String file )
  {
    String temp [] =new String [length];
    BufferedReader fichier = null;

    try
    {
      fichier = new BufferedReader(new FileReader(file));   //Ouverture du flux de fichier
      String ligne = fichier.readLine();  //Lecture de la premiere ligne du fichier
      while( ligne != null )
      {
        temp = ligne.split(","); //On prend chaque élément séparé par une virgule
        ligne = fichier.readLine(); //On lit la ligne suivante du fichier
      }
    }
    catch (FileNotFoundException exception) //Gestion de l'erreur du au fait de ne pas trouver le fichier
    {
      System.err.println("Le fichier " + file +" n'a pas été trouvé :" + exception.getMessage());
    }
    catch (IOException exception)   //Gestion de l'erreur du à la lecture du fichier
    {
      System.err.println ("Erreur lors de la lecture : " + exception.getMessage());
    }
    finally //Exécuté dans tout les cas
    {
      try
      {
        fichier.close();    //Fermeture du flux de fichier
      }
      catch(IOException exception)    //Gestion de l'erreur du à la fermeture du flux de fichier
      {
        System.err.println ("Erreur lors de la fermeture du flux de fichier : " + exception.getMessage());
      }
    }

    double vector [] = new double[temp.length];

    for (int runner_1 = 0 ; runner_1 < length ; runner_1++)
    {
      vector[runner_1] = Double.parseDouble(temp[runner_1]); //Conversion et stockage
    }
    System.out.println("\nLe fichier " + file + " contient le vecteur de personnalisation suivant : \n");
    PageRank.print_a_vector(vector);
    return vector;
  }

}
```