Titre:

. *Un problème pour la classe P = NP*

Auteur:
Sandou 1 Daffé

Affiliation:
Freelance en Développement d'application et Professeur vacataire d'algorithmique des Universités privées. Adresse: Conakry, Commune de matoto, Yimbaya tannerie

Tél:
+224 655037767

E-mail:
sandaff001@gmail.com

Date:
12/11/2022

Résumé :

La complexité algorithmique est un concept très important qui permet de ***comparer les algorithmes*** *afin de trouver celui qui est le plus efficace.*

Dans cet article, on se focalise sur l'une des questions fondamentales posée aujourd'hui à savoir si la classe de complexité P=NP ; et pour parvenir à la démontrer on fait intervenir la théorie de la logique mathématique notamment la théérie des ensembles ;

P = NP ssi P ⊆ NP et NP ⊆ P ;

*NP ⊆ P ssi au moins l'un des NP-Complets est résolu dans O (n^k) ; c'est ainsi qu'un problème de logement dans un dortoir qui est assez difficile et très fort incompatibilité a été choisi et résolu en O (n * ln(n)) et O (n^3) comme solution dont le code est écrit en java ;*

Introduction :

Une implication de P = NP concerne le problème de la décision, nommé souvent sous le terme original allemand « Entscheidungsproblem ». Ce problème, posé par le mathématicien David Hilbert en 1928, consiste à se demander s'il existe un algorithme qui, si on lui présente une question mathématique dont la réponse est « Oui » ou « Non » dans un langage formel, trouvera automatiquement et infailliblement la réponse. Un tel algorithme serait potentiellement en mesure, par exemple, de répondre à la question de savoir si la conjecture de Goldbach ou l'hypothèse de Riemann est vraie ou fausse (si ces problèmes sont décidables).

Pour résoudre ce problème, Selon le site ci-dessous :
https://interstices.info/pnp-elementaire-ma-chere-watson/
Tout problème dans la classe P est dans la classe NP, mais la réciproque est une question ouverte. Les problèmes du voyageur de commerce, du sac à dos sont des problèmes de la classe NP, ils sont même NP-complets : si on sait les résoudre en temps polynomial, alors on sait le faire pour tout problème de la classe NP et on a démontré que P=NP.

Pour donner une piste de solution, on doit avoir à l'idée qu'il y a deux façons de trouver facilement la solution d'un problème de classe NP dans O (n^k) :
1-trouver des formules le concernant ;
2-trouver une méthode réduisant considérablement la résolution par la force brute ;

Rien n'est facile, rien n'est impossible, tout doit se comprendre ; alors on peu analyser un des problèmes de NP-complets ou NP-difficiles en cherchant soit une formule le concernant, soit en réduisant le traitement par la force brute comme nous allons voir sur le problème que nous allons étudier ci-dessous;

Si l'on note le problème ci-dessous comme étant X :

A signaler que << X >> a été tiré sur le site web de technoscience : https://www.techno-science.net

<< Supposons que vous organisez des logements pour un groupe de quatre cents étudiants universitaires.

L'espace est limité et seulement une centaine d'étudiants recevront des places dans le dortoir.

Pour compliquer les choses, le doyen vous a fourni une liste de paires d'étudiants incompatibles, et a demandé qu'aucune paire de cette liste n'apparaisse dans votre choix final.

C'est un exemple de ce que les informaticiens appellent un problème NP, puisqu'il est facile de vérifier si un choix donné de cent élèves proposé par un collègue est satisfaisant (c'est-à-dire qu'aucun couple pris dans la liste de votre collègue n'apparaît également sur la liste de le bureau du doyen), mais la tâche de générer une telle liste à partir de zéro semble être si difficile qu'elle est complètement irréalisable.

En effet, le nombre total de façons de choisir cent étudiants parmi les quatre cents candidats est supérieur au nombre d'atomes dans l'univers connu ! Ainsi, aucune civilisation future ne pourrait jamais espérer construire un supercalculateur capable de résoudre le problème par la force brute ; c'est-à-dire en vérifiant toutes les combinaisons possibles de 100 étudiants.

Cependant, cette difficulté apparente ne peut que refléter le manque d'ingéniosité de votre programmeur.

En fait, l'un des problèmes en suspens en informatique est de déterminer s'il existe des questions dont la réponse peut être vérifiée rapidement, mais qui nécessitent un temps incroyablement long pour être résolues par une procédure directe.

Des problèmes comme celui énuméré ci-dessus semblent certainement être de ce genre, mais jusqu'à présent, personne n'a réussi à prouver que l'un d'entre eux est vraiment aussi difficile qu'il y paraît, c'est-à-dire, qu'il n'y a vraiment aucun moyen réalisable de générer une réponse à l'aide d'un ordinateur.

Stephen Cook et Leonid Levin ont formulé indépendamment le problème P (c'est-à-dire facile à trouver) contre NP (c'est-à-dire facile à vérifier) en 1971 >>

Soit S_1 la solution proposée :

On part du fait que toute incompatibilité est discriminatoire et la discrimination peut être naturelle ou humaine ; on parle ainsi du critère de composition des incompatibilités ;

Discrimination naturelle : c'est une discrimination qui émane de la volonté de Dieu dont entre autres ;
Le genre masculin et féminin, la religion musulman, chrétien etc., les étrangers et nationaux, et le comportement individuel et la constations de désaccord entre deux étudiants ; ces incompatibilités sont héritées de la volonté divine dont le doyen de l'université ne peu que de se contenter de relever ses incompatibilités naturelles et les transmettre.

Discrimination humaine : c'est une discrimination qui émane de la volonté humaine dont on décide de qui est incompatible avec qui ou le doyen compose de son propre chef sa préférence qui peu amener une impossibilité de choix quand tout le monde est incompatible avec tout le monde s'il complique jusqu'à ce niveau ou qu'un ou deux étudiants sont incompatible avec tous les autres et dans ce cas ils sont d'offices hors sélection avant le commencement.

Bon si on reste dans l'énoncé du problème, on peu comprendre que les incompatibilités sont de pair ; ce qui veux dire que si un étudiant est incompatible dans cette liste, il le serra une seule fois, donc avec un seul étudiant ; pour ce faire, si un étudiant est sélectionné cella entrainera l'élimination de sa pair et les cent (100) étudiants sélectionnés entrainera la l'élimination de cent(100) autres dans les sélections suivantes ; donc parmi les quatre cent (400), au lieu de continuer la sélectionné avec trois cent (300) étudiant restants, ça sera plutôt deux cent (200) étudiants car les cent (100) sont déjà éliminés par leurs pairs choisies; Mais il n'est pas exclu de travailler dans un cadre général ou un étudiant peut être incompatible avec deux, trois et quatre autres et ainsi de suite ; la change pour être choisi sera de plus en plus réduite mais il n'aura aucun incident sur la méthodologie ;

Dans le cas de voyageur de commerce, dire qu'entre tel nombre de villes peut on avoir un chemin inferieur à une distance de longueur L ?
Ne signifie pas que la distance trouvée de longueur d<L est la plus petite distance qu'on peut trouver ;

De même ici dire que parmi 400 cent étudiants peut on trouver une liste de 100 sans l'apparition d'aucun pair d'étudiants incompatibles ?
Ne signifie pas que le choix effectué est le meilleur par rapport à toutes les possibilités de choix qu'on peut avoir ;

Dans cette solution, nous ajoutons le menu principal pour faciliter la manipulation, une fonction de saisie de liste des étudiants, une fonction de saisie de liste des incompatibilités, deux fonctions de recherches des incompatibilités ;
Mais on se focalise ici sur la fonction de sélection qui est le travail demandé ; on considère que les listes d'étudiants et pairs d'incompatibilités sont déjà faites et fournies par le doyen ;

```
package PegaleNP;
```

```
import java.util.*;
public class PegaleNP {

//_____________________________________________________________
        public static void main(String[] args ) {
            Scanner keby = new Scanner(System.in);
            System.out.println("Veuillez saisir le nombre
d'étudiants :");
            // Liste des étudiants
            int n = keby.nextInt();
            String[][] tabEtud = new String[n][4];     // Une
structure ou class aurait été préférable
            System.out.println("Veuillez saisir le nombre pair
d'étudiant incompatible :");
            // Liste de pairs étudiants incompatibles dans le
même dortoir exemple: 2 pour 4 étudiants ou 5 pour 10
étudiants
            int n1 = keby.nextInt();
            String [][] tabEtudInc = new String[n1][2];
            System.out.println("Veuillez saisir le nombre
d'étudiants à sélectionner par pair ; càd total d'étudiant ou
2 * nombre de chambre :");
            // 2 étudiants par chambre soit 2*nombres de
chambres
            int m = keby.nextInt();
            String [][] tabEtudSelect = new String[m][2];
            String  clobalId  = "001"; //permettant de faire
la recherché dans la fonction sélection; ça récupère id1 et
fait passer dans paramètre de la fonction recherche.
            String  choix; // utile pour le menu principal.
            do {
               System.out.println("------Menu principal ------
---");
               System.out.println("   1. Liste d'étudiants
");
               System.out.println(" 2.Liste d'étudiants
incompatibles  ");
               System.out.println("  3. Sélection d'étudiants
");
               System.out.println("           4.Quitter
");
               System.out.println("      Taper votre choix :
");
                choix = keby.nextLine();
               switch(choix) {
                  case "1": saisieEtudiant (tabEtud, keby);
                     break;
                  case "2": saisieIncompatibilite (tabEtudInc,
keby);
                     break;
```

```
                    case "3": Selection(tabEtudSelect, tabEtud ,
tabEtudInc , keby, clobalId);
                       break;
                    default : System.out.println("Veuillez
respecter le menu!");
                   }
                } while(choix != "4");
               }


//_____________________________________________________________
   public static  void saisieEtudiant(String [][] tabEtud,
Scanner keby) {
                    int n = tabEtud.length;
                    int i = 0;
                    do {
                       System.out.println("Veuillez saisir l'Id
de l'étudiants :");
                       tabEtud[i][0] = keby.nextLine();
                       System.out.println("Veuillez saisir le
nom de l'étudiants :");
                       tabEtud[i][1] = keby.nextLine();
                       System.out.println("Veuillez saisir le
prénom de l'étudiants :");
                       tabEtud[i][2] = keby.nextLine();
                       System.out.println("Veuillez saisir
l'adresse de l'étudiants :");
                       tabEtud[i][3] = keby.nextLine();
                       i = i + 1;
                       } while (i < n);
                     }
//_____________________________________________________________
   public static void saisieIncompatibilite(String [][]
tabEtudInc, Scanner keby) {
                    int n = tabEtudInc.length;
                    int i = 0;
                    do {
                       System.out.println("Veuillez saisir l'Id
de l'étudiant1 incompatible:");
                       tabEtudInc[i][0] = keby.nextLine();
                       System.out.println("Veuillez saisir l'Id
de l'étudiant2 incompatible:");
                       tabEtudInc[i][1] = keby.nextLine();
                       i = i + 1;
                    } while(i < n);
                 }

//Sélection par tirage au sort _____________________________
 public static void Selection(String [][] tabEtudSelect,
String [][]tabEtud, String [][]tabEtudInc, Scanner keby,
```

```
String clobalId) {
                    int n = tabEtud.length;
                    int Indice=0; // indice de id incompatible
nécessaire pour le test de compatibilité utiliser dans if vers
la fin.
                    int  EI=tabEtudInc.length; // nombre
étudiants incompatibles
                    String id1;//id pour le premier tirage
                    String id2;//id pour le deuxième tirage
                    String idRe;//variable de sortie (id)
                    int inRe;//indice ou rang de id rechercher
                    int n1;//la taille de l'étudiant
                    //int EC=n*n-1/2-EI; // nombre étudiants
compatible
                    Boolean Ajouter = false;
// Si Ajouter = vrai sortir du while
                    String[][] tabTirage = new String[1][2];
// 2x tirés, rangés et remplacés au tirage suivant
                    // Mets la liste étudiant dans ArrayList
pour faciliter le traitement
                    ArrayList<String> tabEtudCopy = new
ArrayList< String > ();
                    // Récupère les sélections et à la sortie
recopiera dans tabEtudSelect
                    ArrayList<String> tabEtudSelectCopy = new
ArrayList< String > ();
                    int c = tabEtudSelect.length/2; // Nombre de
chambres, doit être = m / 2 et m = tabEtudSelect.length
                    for(int i = 0; i < n; i++) {
                       tabEtudCopy.add(tabEtud[i][0]); //
Transfert de la liste étudiant dans la copie
                    }

                    /*Saisie des étudiants  dans des chambres;
i=0 une chambre donc deux étudiants; i=1 une seconde
                    chambre donc deux "étudiants encore ainsi de
suite*/

                    if(EI==n*n-1/2) {
                        // si tout le monde est incompatible
avec tout le monde alors on n'entre pas dans la boucle;
                      // en plus on a deux choix
                      //1-afficher le message d'information
                      //2-faire une seule selection et sortir
                      System.out.println("Selection impossible,
tout le monde est incompatible avec tout le monde!");
                       /* OU
                          Random random = new Random();
// Tirage au sort ou nombre aléatoire
                          n1 = random.nextInt(n);
```

```
// Choix dans l'intervalle  0 et n
                        id1 = tabEtudCopy.get(n1);
                        tabEtudSelectCopy.add(id1);
// Si id1 retenu, l'étudiant sera sélectionné

                    */
                      } else {
                            loop:
                    for(int i = 0; i < c; i++)  {
                      Ajouter = false;
// Chaque ajout égal vrai pour sortir du do while
                     do {
                        Random random = new Random();
// Tirage au sort ou nombre aléatoire
                        int nb;
                        n1  = tabEtudCopy.size();
// Détermine le nombre éléments restants dans la copie
étudiantes; chaque sélection est supprimée dedans
                         if(n1 == 1){
                                id1 = tabEtudCopy.get(n1);
// il est le seul à être logé   dans une chambre
                                tabEtudSelectCopy.add(id1);
// Il est rangé dans le tirage de façon définitive
                         tabEtudCopy.remove(n1);           //
Oté du table copie
                                break loop;
//fin de selection
                    } else {
                                // la sélection continu même si
le nombre de chambre > aux nombres compatibilités, car la
sélection s'arrête quand n1=0
                        nb = random.nextInt(n1);             //
Choix dans l'intervalle  0 et n
                        id1= tabEtudCopy.get(nb);           //
Si id1 retenu, l'étudiant sera sélectionné
                        clobalId = id1;
//clobalId est un string car c'est id1 déclaré comme variable
globale
                     idRe = recherche(tabEtudInc,clobalId); //
id1 dans la liste d'incompatibilité ?
                    if(idRe == "-1") {
                      // Si OUI alors il est ajouté
                      tabTirage[0][0] = id1;  // Il est rangé
dans le tirage
                      tabEtudCopy.remove(nb); // Oté du
tableau pour ne pas retomber sur le même id
                       n1   = tabEtudCopy.size();
                       nb  = random.nextInt(n1);
                       id2 = tabEtudCopy.get(nb);
                       tabTirage[0][1] = id2;  // Son binôme
```

```
est aussi tiré
                        tabEtudCopy.remove(nb);
                        Ajouter = true;  // Pour sortir du
while, aller dans la chambre suivante
                       }
                         else {

                          // Sinon alors on cherche le second
et on vérifie s'ils sont incompatibles
                       inRe = recherche2(tabEtudInc,clobalId);
// avant on cherche son indice  d'incompatibilié ?
                       Indice =inRe;
                       tabEtudCopy.remove(nb);// Evidemment il
faut supprimer id1 avant
                      n1   = tabEtudCopy.size();
                      nb  = random.nextInt(n1);
                      id2 = tabEtudCopy.get(nb);

if(((id1.equals(tabEtudInc[Indice][0])) &&
(!id2.equals(tabEtudInc[Indice][1])))
                                  ||
((id1.equals(tabEtudInc[Indice][1])) &&
(!id2.equals(tabEtudInc[Indice][0])))) {
                                          // Si le couplage est
possible alors on les tire tous les deux
                                  tabTirage[0][0] = id1;
                                  tabTirage[0][1] = id2;
                                  tabEtudCopy.remove(nb);
                                  Ajouter = true;
                                 }
                                else
                     tabEtudCopy.add(id1);
                   /* Si le duo n'est pas possible, on remet
dans la table tabEtudCopy l'id1 qui a été supprimé pour
recommencer sans changer de chambre. Ajouter reste à false */
                     }
                                  }
                } while(Ajouter == false);
                // Les deux tirés sont rangés dans la selection
                n1 = tabEtudSelectCopy.size();
                tabEtudSelectCopy.add(tabTirage[0][0]);
                tabEtudSelectCopy.add(tabTirage[0][1]);
                   }
                   /* récupére la selection en tableau static
si le travail est fini et tabEtudSelectCopy aura une taille =
2*c   */
                int j = 0;
                for(String elem:tabEtudSelectCopy) {// Tiens on
repasse de c à m !
                tabEtudSelect[j][0] = elem;// Récupération
```

```
                System.out.println("tabEtudSelect: "+elem);
                      j++;
                   }
                }

             }

//_____________________________________________________________________
   public static String recherche(String [][]tabEtudInc,
String clobalId) {
                 int n = tabEtudInc.length;
                 String sortietest="-1"; // cherche les
incompatibilités, sort avec "-1" si id n'existe pas
                  loop:
                 for(int i = 0; i < n ; i++) {
                 for(int j = 0; j < 2; j++ ) {
                 if(clobalId.equals(tabEtudInc[i][j])) {
                  sortietest = tabEtudInc[i][j];
                   break loop;
                    }
                    }
                   }
                    return sortietest   ;
                }
 public static int recherche2(String [][]tabEtudInc, String
clobalId) {
             int n = tabEtudInc.length;
             int Indice=0; // cherche son indice dans la table
incompatibilité
                loop:
              for(int i = 0; i < n ; i++) {
               for(int j = 0; j < 2; j++ ) {
                if(clobalId.equals(tabEtudInc[i][j])) {
                 Indice = i;
                 break loop;
                  }
                  }
                 }
                return Indice   ;
                }
          }
```

Détermination de la complexité:

Quand il y a des tirages au sort, l'étude de complexité est toujours de type probabiliste.

Quelle est la probabilité que le tirage au sort soit défaillant ?

C'est u/n où u est le nombre d'étudiants incompatibles (pas pour le tirage de id1, avec id1 pour le tirage de id2) ou déjà affectés à une chambre.

Pour une chambre

Dans tous les cas il y a 1 tirage.
Pour qu'il y ait un tirage en plus il faut que le précédent ait échoué soit u/n
Pour qu'il y ait un tirage en plus il faut que les précédents aient échoué soit $(u/n)^2$
...
Le nombre de tirage moyen est donc : $1 + u/n + (u/n)^2 + (u/n)^3 + (u/n)^4 + ...$ C'est une série géométrique qui converge vers 1/(1-u/n) soit n/(n-u).

Pour toutes les chambres

*Supposons un taux moyen d'incompatibilité de 0 <= a < 1. Pour la première étape, u = a*n et la formule précédente devient 1/(1-a) .*
Pour la seconde étape on aura n/(n-2 -a(n-2)) puisqu'il y a 2 étudiants qui sont retirés du jeu (mais pas du tirage aléatoire qui est toujours sur n.*
Pour l'étape i (en supposant que i commence à 0 et se termine à c-1)) on aura n/(n-2i -a(n-2i)) soit n/((n-2i)(1-a)) (en toute rigueur, c'est vrai pour id1 (avec a=0) mais il faudrait écrire n/((n-2i-1)(1-a)) pour id2 puisque id1 doit être retiré du jeu).*

Au total nous aurons :

$$\frac{1}{1-a}\sum_{i=0}^{c-1}\frac{n}{n-2i} = \frac{n/2}{1-a}\sum_{i=0}^{c-1}\frac{1}{n/2-i} \approx \frac{n/2}{1-a}\left(\ln\left(\frac{n}{2}\right)-\ln\left(\frac{n}{2}-c+1\right)\right)$$

*D'où, sauf erreur, la complexité en O(n * ln(n)). On néglige ici le choix du premier des deux occupants d'une chambre car la contrainte est plus faible (il n'a pas de coefficient a d'incompatibilité) et cela ne change pas l'ordre de complexité.*

Si l'on considère un dortoir comme un couvant, la sélection serrait:
Si le dortoir est une grande salle ou un couvant ou tous les sélectionnés sont ensemble alors:

On conçoit la fonction sélection comme suit:

On créer une copie du tableau étudiant dont on pioche les sélectionnés;

Cette fois-ci la sélection se fait un à un au lieu de deux à deux comme la précédente;

On tire un et on le supprime dans la copie ainsi que tous ceux qui sont incompatibles avec lui;

Il y a donc une recherche comme d'habitude et après tout, on obtient une nouvelle liste;

-Si on a 400 étudiants, on tire un, ça reste 399; plus ses incompatibilités, admettons 5 qui sont ensuite supprimées, on a 394 restants;
-Si je tire un deuxième, on a 393, plus ses incompatibilités admettons 13, on a 380 restants qui sera la nouvelle liste;
ainsi de suite, ...

on arrête avec trois conditions:

-Tout le monde est incompatible avec tout le monde et un seul est tiré, la copie est vide donc on arrête; n1=0 qui est la taille du tableau;

-L'étudiant sélectionné est incompatible avec un autre, deux autres, trois autres, etc..
1- on arrête à n1 égale à 0 donc la copie est vide et le nombre sélectionnés < 100 demandés, donc compteur<100;

2-on arrête avec le nombre sélectionnés égale à 100 qui est demandés, donc compteur égale à 100; on ne se souci plus de n1;

3- on arrête avec le nombre sélectionnés égale à 100 qui est demandés, donc compteur égale à 100; ainsi si n1 est égale à zéro avant que le compteur n'atteigne 100 alors on toute la sélection est annulée et on recommence à zéro ;

-En prenant les deux premier cas, aucun tirage n'échoue et le nombre d'opération peut être inferieur (<) à n;

-Le troisième cas nous ramène dans le cadre d'un tirage au sort relatif d'une seule chambre avec une forte incompatibilité dont l'échec d'un tirage en cours peut annuler tous les tirages précédents;

Les deux algorithmes sont les suivants :

On part du constat suivant:
Dans le cas de voyageur de commerce, dire qu'entre tel nombre de villes peut on avoir un chemin inférieur à une distance de longueur L ?

Ne signifie pas que la distance trouvée de longueur d<L est la plus petite distance qu'on peut trouver ;

De même, ici dire que parmi 400 cent étudiants peut on trouver une liste de 100 sans l'apparition d'aucun pair d'étudiants incompatibles ?

Ne signifie pas que le choix effectué est le meilleur par rapport à toutes les possibilités de choix qu'on peut avoir ;

Donc ce n'est pas un choix de solution mais plutôt un résultat qui ne comporte aucun pair d'incompatibilité ou qui respect le critère de choix;

Troisième cas :

```
//Sélectionbis par tirage au sort deuxième version
public static void Selection2(String [][] tabEtudSelect,
String [][]tabEtud, String [][]tabEtudInc, Scanner keby,
String clobalId) {
                    int n = tabEtud.length;
                    int  EI=tabEtudInc.length; // nombre
étudiants incompatibles
```

```
                    String id1; //id du tirage
                    String idRe; //variable de sortie (id)
                    int n1;
                    String idSup;
                    int cpteur;
                    //int EC=n*n-1/2-EI; // nombre étudiants
compatible
                    // Mets la liste étudiant dans ArrayList
pour faciliter le traitement
                    ArrayList<String> tabEtudCopy = new
ArrayList< String > ();
                    // Récupère les sélections et à la sortie
recopiera dans tabEtudSelect
                    ArrayList<String> tabEtudSelectCopy = new
ArrayList< String > ();
                    int c = tabEtudSelect.length; // Nombre de
place à pouvoir / qui est équivalent aux nombre de chambres
dans recherche1
                   //création de copy de la table étudiant
                    for(int i = 0; i < n; i++) {
                       tabEtudCopy.add(tabEtud[i][0]); //
Transfert de la liste étudiant dans la copie
                    }

                    /*Sélection des étudiants  dans un couvant
ou une grande salle*/

                    if(EI==n*n-1/2) {
                        // si tout le monde est incompatible
avec tout le monde alors on n'entre pas dans la boucle;
                      // en plus on a deux choix
                      //1-afficher le message d'information
                      //2-faire une seule selection et sortir
                      System.out.println("Sélection impossible,
tout le monde est incompatible avec tout le monde!");
                       /* OU
                          Random random = new Random();
// Tirage au sort ou nombre aléatoire
                          n1 = random.nextInt(n);
// Choix dans l'intervalle  0 et n
                          id1 = tabEtudCopy.get(n1);
                          tabEtudSelectCopy.add(id1);
// Si id1 retenu, l'étudiant sera sélectionné

                      */
                    } else {

                      cpteur=0;
                     Random random = new Random();        //
Tirage au sort ou nombre aléatoire
                     int nb;
```

```
                    do {
                     n1  = tabEtudCopy.size();// Détermine le
nombre éléments restants dans la copie étudiantes; chaque
sélection est supprimé dedans
                    if(n1 == 0){
//Les sélection précédentes sont annulées
                    tabEtudCopy.clear() ;
                    tabEtudSelectCopy.clear() ;
//création de la nouvelle copy de la table étudiant
                    for(int i = 0; i < n; i++) {
                    tabEtudCopy.add(tabEtud[i][0]); // Transfert
de la liste étudiant dans la copie
                                    }

                       }

                    if(n1 == 1){
                    id1 = tabEtudCopy.get(n1); // il est le seul
à être logé  dans une chambre
                    tabEtudSelectCopy.add(id1); // Il est rangé
dans le tirage de façon définitive
                    tabEtudCopy.remove(n1);    // Oté du tableau
                    n1=0;
                     cpteur++ ;
                   } else {
                    // la sélection continu même si le nombre
de chambre > aux nombres compatibilités, car la sélection
s'arrête quand n1=0
                   nb = random.nextInt(n1); // Choix dans
l'intervalle  0 et n
                   id1= tabEtudCopy.get(nb);// Si id1 retenu,
l'étudiant sera sélectionné
                   clobalId = id1; //clobalId est un string car
c'est id1 déclaré comme variable clobale
                   idRe = recherche(tabEtudInc,clobalId); // id1
dans liste d'incompatibilité ?
                   if(idRe == "-1") {
                   // Si oui alors il est ajouté
                   tabEtudSelectCopy.add(id1); // Il est rangé
dans la sélection
                   tabEtudCopy.remove(nb);// Oté du tableau pour
ne pas retomber sur le même
                   cpteur++;
                   n1  = tabEtudCopy.size();
                    }
                    else {
                     tabEtudSelectCopy.add(id1);  // Il est
rangé dans la sélection
                    tabEtudCopy.remove(nb); // Oté du tableau
pour ne pas retomber sur le même
```

```
                    // Suppression des id incompatibles avec id
sélectionné
                     idSup="-1";
                     for(int i = 0; i < tabEtudInc.length; i++)
                       { //recherche de l'id dams incompatibilité
                      for(int j = 0; j < 2; j++ ) {
                      if(id1.equals(tabEtudInc[i][j])) {
                      if(j == 0) { // correspond à id sélectionné
                      idSup = tabEtudInc[i][1];  // correspond à
id incompatible
                      }else { // j=1 correspond à id sélectionné
                      idSup = tabEtudInc[i][0];  //j=0 correspond
à id incompatible
                          }
                       // Suppression dans la liste d'étudiants
                      int h=0;
                      sortir:
                      for(String elem:tabEtudCopy) {
                        h++;
                       if(idSup==elem) { // id trouvé
                       tabEtudCopy.remove(h) ; //id supprimé
                         break sortir;
                               }
                               }
                             }
                            }
                                }

                                 cpteur++;
                               n1 = tabEtudCopy.size();

                                  }
                               }
                                } while(cpteur!=c);

          /* récupère la sélection en tableau static si le
travail est fini et tabEtudSelectCopy aura une taille = 2*c
*/
                  int j = 0;
                  for(String elem:tabEtudSelectCopy) {
                  tabEtudSelect[j][0] = elem;// Récupération
                  System.out.println("tabEtudSelect: "+elem);
                        j++;
                     }
                  }

                }
```

[Détermination de la complexité:]

La complexité est tout à fait similaire à la précédente ;

Premier et deuxième cas dans un seul algorithme :

```
// Sélectionbis par tirage au sort deuxième version_____
public static void Selection2(String [][] tabEtudSelect,
String [][]tabEtud, String [][]tabEtudInc, Scanner keby,
String clobalId) {
                    int n = tabEtud.length;
                    int  EI=tabEtudInc.length; // nombre
étudiants incompatibles
                    String id1; //id du tirage
                    String idRe; //variable de sortie (id)
                    int n1;
                    String idSup;
                    int cpteur;
                    //int EC=n*n-1/2-EI; // nombre étudiants
compatible
                    // Mets la liste étudiant dans ArrayList
pour faciliter le traitement
                    ArrayList<String> tabEtudCopy = new
ArrayList< String > ();
                    // Récupère les sélections et à la sortie
recopiera dans tabEtudSelect
                    ArrayList<String> tabEtudSelectCopy = new
ArrayList< String > ();
                    int c = tabEtudSelect.length; // Nombre de
place à pouvoir / qui est équivalent aux nombre de chambres
dans recherche1
                   //création de copy de la table étudiant
                    for(int i = 0; i < n; i++) {
                    tabEtudCopy.add(tabEtud[i][0]); // Transfert
de la liste étudiant dans la copie
                    }

                    /*Sélection des étudiants  dans un couvant
ou une grande salle*/

                    if(EI==n*n-1/2) {
                        // Si tout le monde est incompatible
avec tout le monde alors on n'entre pas dans la boucle;
                      // En plus on a deux choix
                      //1-afficher le message d'information
                      //2-faire une seule sélection et sortir
                      System.out.println("Sélection impossible,
tout le monde est incompatible avec tout le monde!");
                       /* OU
                          Random random = new Random();
// Tirage au sort ou nombre aléatoire
                          n1 = random.nextInt(n);
// Choix dans l'intervalle  0 et n
```

```
                        id1 = tabEtudCopy.get(n1);
                        tabEtudSelectCopy.add(id1);
// Si id1 retenu, l'étudiant sera sélectionné

                        */
                       } else {
                       cpteur=0;
                       Random random = new Random(); // Tirage au
sort ou nombre aléatoire
                      int nb;
                       do {
                       n1  = tabEtudCopy.size(); // Détermine le
nombre éléments restants dans la copie étudiantes; chaque
sélection est supprimé dedans
                        if(n1 == 1){
                       id1 = tabEtudCopy.get(n1);// il est le seul
à être logé  dans une chambre
                        tabEtudSelectCopy.add(id1); // Il est rangé
dans le tirage de façon définitive
                      tabEtudCopy.remove(n1); // Oté du tableau
                       n1=0;
                     cpteur++ ;
                     } else {
                     // la sélection continu même si le nombre de
chambre > aux nombres compatibilités, car la sélection
s'arrête quand n1=0
                    nb = random.nextInt(n1); // Choix dans
l'intervalle  0 et n
                   id1= tabEtudCopy.get(nb);// Si id1 retenu,
l'étudiant sera sélectionné
                 clobalId = id1; //clobalId est un string car
c'est id1 déclaré comme variable clobale
                 idRe = recherche(tabEtudInc,clobalId); // id1
dans liste d'incompatibilité ?
                if(idRe == "-1") {
                // Si oui alors il est ajouté
                tabEtudSelectCopy.add(id1);   // Il est rangé dans
la sélection
                tabEtudCopy.remove(nb); // Oté du tableau pour ne
pas retomber sur le même
               cpteur++;
               n1  = tabEtudCopy.size();
                }
                 else {
                tabEtudSelectCopy.add(id1); // Il est rangé dans
la sélection
               tabEtudCopy.remove(nb); // Oté du tableau pour ne
pas retomber sur le même
          // Suppression des id incompatibles avec id
sélectionné
               idSup="-1";
```

```
            for(int i = 0; i < tabEtudInc.length; i++) {//
recherche de l'id dams incompatibilité
            for(int j = 0; j < 2; j++ ) {
            if(id1.equals(tabEtudInc[i][j])) {
            if(j == 0) { // correspond à id sélectionné
            idSup = tabEtudInc[i][1];  // correspond à id
incompatible
             }else { // j=1 correspond à id sélectionné
              idSup = tabEtudInc[i][0];  //j=0 correspond à id
incompatible
              }
                // Suppression dans la liste d'étudiants
              int h=0;
              sortir:
            for(String elem:tabEtudCopy) {
                h++;
            if(idSup==elem) { // id trouvé
             tabEtudCopy.remove(h) ; //id supprimé
             break sortir;
                 }
                 }
                }
               }
            cpteur++;
            n1=tabEtudCopy.size();
                }
               }
       } while((cpteur!=c) && (n1!=0));

          /* récupère la sélection en tableau static si le
travail est fini et tabEtudSelectCopy aura une taille = 2*c
*/
            int j = 0;
            for(String elem:tabEtudSelectCopy) {
            tabEtudSelect[j][0] = elem; // Récupération
            System.out.println("tabEtudSelect: "+elem);
              j++;
                 }
                }
          }
```

Détermination de la complexité:

Comme aucun choix n'est défaillant, le calcule de la complexité est ramenée à un calcule ordinaire ;
Ainsi les trois boucles while, for et for sont imbriquées, donc la complexité est en $O(n^3)$*;*
Une complexité cubique.

Classe P

Un problème de décision est dans P s'il peut être décidé sur une machine déterministe en temps polynomial par rapport à la taille de la donnée. On qualifie alors le problème de polynomial, c'est un problème de complexité $O(n^k)$ pour un certain k.

Classe NP

La classe NP des problèmes ***Non-déterministes Polynomiaux*** *réunit les problèmes de décision qui peuvent être décidés sur une machine non déterministe en temps polynomial. De façon équivalente, c'est la classe des problèmes qui admettent un algorithme polynomial capable de tester la validité d'une solution du problème, on dit aussi capable de construire un certificat. Intuitivement, les problèmes dans NP sont les problèmes qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial.*
Par exemple, la recherche de cycle hamiltonien dans un graphe peut se faire à l'aide de deux algorithmes :
-le premier engendre l'ensemble des cycles (en temps exponentiel, classe EXPTIME, voir ci-dessous) ;

-le second teste les solutions (en temps polynomial).

Ce problème est donc de la classe NP.

Démonstration de P ⊆ NP :

NP = classe des problèmes qu'on peut résoudre

P = classe des problèmes qu'on peut résoudre en temps polynomial

Si un problème est dans P, on peut le résoudre ; Donc il est aussi dans NP, ce qui signifie en conséquence que $P \subseteq NP$.

On a clairement $P \subseteq NP$ car un algorithme déterministe est un algorithme non déterministe particulier, ce qui, dit en mots plus simples, signifie que si une solution peut être calculée en temps polynomial, alors elle peut être vérifiée en temps polynomial.

Démonstration de NP ⊆ P :

Soit << X >> le problème évoqué ci-dessus et S_1 sa solution ;

Comme ici, NP= {X} et sa solution $S_1 \in O(n^k)$ alors NP est aussi la classe des problèmes qu'on peut résoudre en temps polynomial, ce qui signifie en conséquence que $NP \subseteq P$.

On a clairement $NP \subseteq P$ car un algorithme non déterministe particulier est un algorithme déterministe, ce qui, dit en mots plus simples, signifie que si une solution quelle conque peut être calculée en temps polynomial, alors elle peut être vérifiée en temps polynomial.

Démonstration de NP = P :

$P \subseteq NP$ et $NP \subseteq P \Rightarrow P = NP$

Conclusion:

La solution proposée ici n'est probablement pas une bonne nouvelle pour l'informatique, pour des solutions déjà en place en matière de cryptographie ; ainsi, même si pour son évolution notamment dans des échanges de données, on ne serrait plus trop sûr de la fiabilité de nos clés de transfert, quand je fais référence de ma théorie publiée sur le forum de developpez.net qui a pour titre << les noyaux des nombres entiers composés impairs >> Une théorie qui réduit considérablement la solution par la force brute et qui, si on a la chance de trouver une bonne formule pour le noyau, la décomposition d'un nombre en produit de facteurs premiers serrait un souvenir au lieu d'un problème ;mais de nombreux problèmes seront résolus et il aura aussi de nouvelles perspectives pour des solutions plus dynamiques pour la sécurisation des données.

Remerciement :

Tous mes remerciements aux experts qui m'ont aidé à la correction et à la détermination de la complexité de l'algorithme ; avant de recevoir leurs noms, nous avons entre autre les pseudonymes suivants : **Monsieur tbc92, Monsieur Guesset, Monsieur WhiteCrow** qui sont tous des intervenants sur developpez.net;

Les contributions de l'auteur :

Il est à signalé que j'ai eux l'intervention sur le forum des experts mentionnés dans le remerciement après mes étude et analyse ; j'ai écrit l'article puis j'ai procédé à la traduction de l'article en anglais ;

Informations sur le financement :

Cette recherche n'a reçu aucune subvention spécifique d'un organisme de financement, d'un secteur commercial ou à but non lucratif.