
# Compte Rendu

*Rédigé par*

**ZHANG Elodie et LABIAD Noriane**

*Groupe 1 encadré par*

**Nawal Benabbou**

*LU3IN025 - IA et Jeux*

*Sorbonne Université - Licence Informatique*

# PARTIE 1 : Problème et affectation

**QUESTION 2**

1) **Structure de données** : un tas
**Complexité** : **O(log n)**, où n est le nombre d'étudiants libres. La récupération du premier étudiant libre est celle avec la plus petite valeur.

2) **Structure de données** : une deque
**Complexité** : **O(1)** car on prend un élément à une extrémité

3) **Structure de données** : des listes de dictionnaires
**Complexité** : **O(1)** car on accède aux éléments directement par les indices

4) **Structure de données** : une liste de tas où l'étudiant le moins préféré est la racine du tas
**Complexité** : **O(log n)**, où n est le nombre d'étudiants affectés à la spécialité.

5) **Structure de données** : une liste où l'indice est l'étudiant et où la valeur est la spécialité
**Complexité** : **O(1)**, car on modifie l'élément directement par l'indice

| |
|---|
| **n** : nombre d'étudiants<br>**m** : nombre de parcours |

**QUESTION 3**

Le code de l'algorithme de Gale-Shapley côté étudiants est constitué de deux parties, l'initialisation des variables locales et de la boucle principale qui fait le traitement.

- Variables locales :
  - **etu_libre** : O(n) où n est le nombre d'étudiants libres
  - **affectation** qui est une liste de taille O(n)
  - **prefSpeIndices** est une liste de dictionnaires où chaque dictionnaire est de taille O(n), car pour chaque spécialité, nous avons un dictionnaire de la forme {etudiant: index}. La taille totale de cette structure est O(mxn)
  - **spe_tas** : liste de tas a une taille de O(m), chaque tas ayant une taille qui peut varier en fonction du nombre d'étudiants affectés à une spécialité
- Boucle principale :
  - La boucle principale while etu_libre parcourt tous les étudiants et est de complexité O(n)

    - À chaque itération, il y a une opération heappop sur etu_libre, suivie d'une heappush sur spe_tas ou etu_libre à chaque fois qu'un étudiant change de spécialité. Ces opérations sont O(log n), et chaque étudiant peut être ajouté ou retiré d'un tas à plusieurs reprises, mais cela ne dépasse pas O(n (log n)) dans le pire des cas.

- **Complexité** : O( n log n) car dominé par la boucle principale

**QUESTION 4**

Le code de l'algorithme de Gale-Shapley côté parcours est constitué de deux parties, l'initialisation des variables locales et de la boucle principale qui fait le traitement.

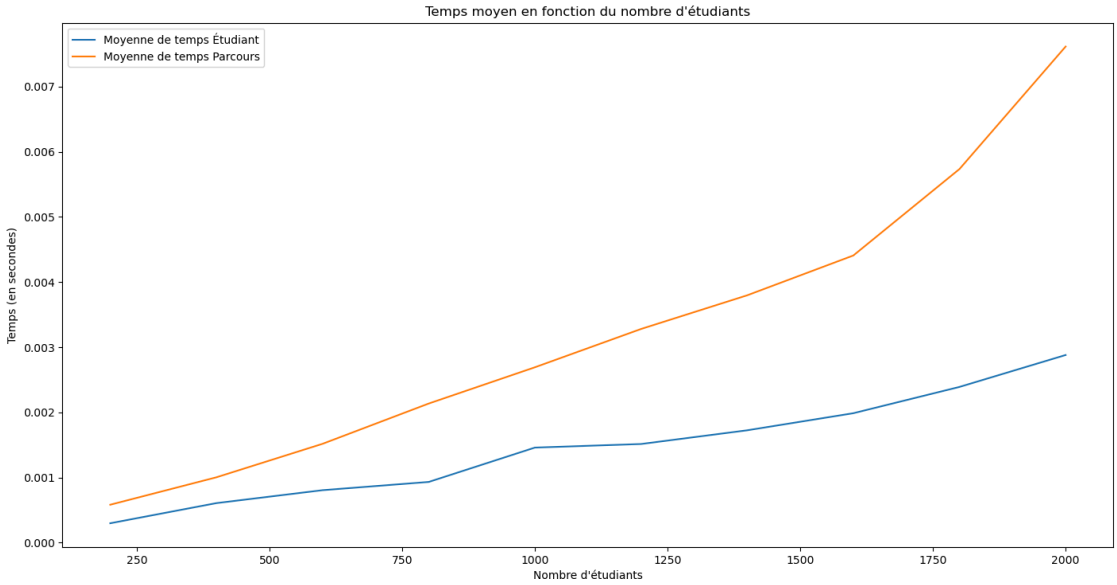
- Variables locales :
    - **spe_libre** : Il s'agit d'un tas contenant les spécialités libres. La taille de cette structure est O(m), où m est le nombre de spécialités.
    - **affectation** : C'est une liste qui enregistre l'affectation de chaque étudiant. La taille de cette liste est O(n), où n est le nombre d'étudiants.
    - **preferenceSpe** : Cette liste contient les préférences des spécialités sous forme de deque. Chaque deque a une taille maximale de n, donc la taille totale de cette structure est O(m×n).
    - **prefSpeIndices** : C'est une liste de dictionnaires où chaque dictionnaire est de taille O(m) car pour chaque spécialité, nous avons un dictionnaire de la forme {spécialité: index}. La taille totale de cette structure est O(n×m), car elle contient un dictionnaire pour chaque étudiant.
- Boucle principale :
    - La boucle principale while(spe_libre) parcourt tous les éléments dans spe_libre, ce qui a une complexité de O(m), car nous avons m spécialités.
    - À chaque itération, il y a une opération heappop sur spe_libre, suivie d'une heappush pour chaque étudiant qui est affecté ou déplacé, ce qui prend O(log m). Étant donné que chaque spécialité peut être poussée et poussée plusieurs fois dans spe_libre, chaque opération d'insertion et de suppression dans le tas est effectuée O(n) fois au maximum.
    - L'opération popleft sur les deque de preferenceSpe a une complexité de O(1), car elle supprime un élément en tête de la deque.
- **Complexité** : O( m log m) car dominé par la boucle principale

**QUESTION 5**

L'algorithme "côté étudiant" se retourne [5, 6, 8, 0, 1, 0, 8, 7, 3, 2, 4], et celui "côté parcours" retourne [6, 5, 8, 0, 1, 0, 8, 7, 3, 2, 4]. Les parcours 5 et 6 ont échangé leurs étudiants 1 et 0.

# PARTIE 2 : Evolution du temps de calcul

**QUESTION 8**




**QUESTION 9**

La complexité du “côté étudiants” est de O(n log n), et celui du “côté parcours” est de O(m log m). m est constant, tandis que que n augmente. Théoriquement, l'algorithme “côté parcours” est censé être plus rapide que celui du “côté étudiants”. Cependant d’après le graphe, l’algorithme “côté étudiants” est plus rapide.

**QUESTION 10**

Le nombre d'itérations dans les algorithmes représente combien de fois la boucle principale est exécutée.

La boucle principale de l’algorithme “côté étudiants” dépend du nombre d'étudiants libres, donc le nombre d'itérations sera au minimum **n**, car chaque étudiant doit être affecté à une spécialité donc chaque étudiant sera traité au moins une fois. Au maximum on aura **m*n itérations**.

La boucle principale de l’algorithme “côté parcours” dépend du nombre de spécialités libres, donc théoriquement, elle s'exécute au minimum **m** fois. Cependant, chacun des parcours possède une capacité d’accueil que l’on note c, donc chaque parcours va faire une proposition

à c étudiants, ce qui entraînera au minimum **C** itérations avec C la somme des capacités de tous les parcours. Au maximum on aura **m*n itérations**.
Ainsi, dans nos cas, lorsque qu'il y a autant de place disponibles que d'étudiants qui en demandent, le nombre maximum d'opérations sera égal pour les deux algorithmes, soit **n*m itérations**.
On peut conclure que GS côté parcours risque de faire beaucoup plus d'itérations car il doit faire plusieurs propositions pour compléter ses places disponibles, ce qui est donc cohérent avec le graphe.

# PARTIE 3 : Equité et PL(NE)

**QUESTION 11**

Soit E l'ensemble des paires (i, j) telles que $p_j$ (parcours j) est dans les k premiers choix de $e_i$ (étudiant i).

Variables :

$x_{ij}$ : vaut 1 si l'étudiant i est affecté au parcours j, sinon vaut 0

Fonction objective :

max $\sum_{(i,j)\in E} x_{ij}$

Contraintes :

- Chaque étudiant est lié à un seul parcours
- Un parcours est lié à moins ou autant de y étudiants (y étant la capacité du parcours)
- $x_{ij} \in \{0,1\},\ pour\ tout\ i \in \{1,...,n\}\ avec\ n\ le\ nombre\ d'étudiants$ ,
  $pour\ tout\ j \in \{1,...,k\}\ avec\ k\ le\ nombre\ de\ premiers\ choix,\ pour\ tout\ (i,j) \in E$

**QUESTION 12 :**

PLNE : QUESTION 12 (page 9)

En résolvant notre programme linéaire avec Gurobi on n'obtient pas de solution pour k = 3.

**QUESTION 13 :**

A l'aide de Gurobi, le plus petit k trouvé avec une solution est k=5.

Une solution maximisant l'utilité minimale avec k=5 trouvé à l'aide de Gurobi :

{(0-5),(1-6),(2-8),(3-7),(4-1),(5-4),(6-2),(7-0),(8-8),(9-3),(10-0)}

**QUESTION 14 :**

PLNE (général) :

Variables :

$u_i$ : utilité de l'étudiant i

$s_{ij}$ : *score de borda = n - rang,* avec n le nombre d'étudiants et rang le rang du parcours j dans les préférences de l'étudiant i

$x_{ij}$ : vaut 1 si l'étudiant i est affecté au parcours j, sinon vaut 0

Fonction objective :

$$max \sum_{i=0}^{ne} \sum_{j=0}^{np} x_{ij} * s_{ij}$$

Soit $max \sum_{i=0}^{ne} u_i$ avec $u_i = \sum_{j=0}^{np} x_{ij} * s_{ij}$

Contraintes :

- Chaque étudiant est lié à une seule spécialité
- Chaque spécialité est lié à <=y étudiants (y étant la capacité)
- $x_{ij} \in \{0, 1\},\ pour\ tout\ i \in \{1,..., ne\}\ et\ pour\ tout\ j \in \{1,..., np\}$

  $avec\ ne\ le\ nombre\ d'étudiants\ et\ np\ le\ nombre\ de\ parcours$

PLNE : QUESTION 14 (maximise la somme des utilités) (page 10)

La solution trouvée avec Gurobi pour le programme linéaire ci-dessus :

{(0-8),(1-5),(2-8),(3-6),(4-1),(5-0),(6-7),(7-0),(8-3),(9-2),(10-4)}

Valeur fonction objective : 156

Moyenne utilité = 156/11 = 14,18181818

Utilité minimale : 4

**QUESTION 15 :**

PLNE (général) :

Variables :

$se_{ij}$ : score de borda côté étudiants = *n - rang* avec *n* le nombre d'étudiants et *rang* le rang du parcours j dans les préférences de l'étudiant i

$sp_{ij}$ : score de borda côté parcours = *n - rang* avec *n* le nombre de parcours et *rang* le rang de l'étudiant i dans les préférences du parcours j

$x_{ij}$ : le mariage entre l'étudiant i et le parcours j

Fonction objective :

$$\max \sum_{i=0}^{n} \sum_{j=0}^{k} x_{ij} * (se_{ij} + sp_{ij})$$

Contraintes :

- Chaque étudiant est lié à un seul parcours
- Chaque parcours est lié à <=y étudiants (y étant la capacité du parcours)
- $x_{ij} \in \{0, 1\},\ pour\ tout\ i \in \{1,..., n\}\ avec\ n\ le\ nombre\ d'étudiants\ et\ pour\ tout\ j \in \{1,..., k\}$

PLNE : QUESTION 15 (maximise la somme des utilités (pour k=5)) (page 12)

On obtient avec Gurobi la solution suivante :

{(0-8),(1-5),(2-8),(3-6),(4-1),(5-0),(6-7),(7-0),(8-2),(9-3),(10-4)}

**QUESTION 16 :**

- Algo GS côté étudiants :

Affectation : {(0-5),(1-6),(2-8),(3-0),(4-1),(5-0),(6-8),(7-7),(8-3),(9-2),(10-4)}

Paires instables : []

Moyenne utilité : 13,45454545

Utilité minimale : 4

- Algo GS côté parcours :

Affectation : {(0-6),(1-5),(2-8),(3-0),(4-1),(5-0),(6-8),(7-7),(8-3),(9-2),(10-4)}

Paires instables : []

Moyenne utilité : 13,36363636

Utilité minimale : 4

- Question 13

Affectation : {(0-5),(1-6),(2-8),(3-7),(4-1),(5-4),(6-2),(7-0),(8-8),(9-3),(10-0)}

Paire instable : [(0, 5), (7, 5), (7, 6), (7, 7), (4, 10), (2, 9), (8, 9)] (spé,etu)

Moyenne utilité : 11,90909091

Utilité minimale : 4

- Question 14

Affectation : {(0-8),(1-5),(2-8),(3-6),(4-1),(5-0),(6-7),(7-0),(8-3),(9-2),(10-4)}

Paire instable : [(6, 0), (6, 1), (7, 7)]

Moyenne utilité : 14,18181818

Utilité minimale : 4

- Question 15

{(0-8),(1-5),(2-8),(3-6),(4-1),(5-0),(6-7),(7-0),(8-2),(9-3),(10-4)}

Paires instables : [(6, 0), (6, 1), (7, 7), (2, 9)]

Moyenne utilité : 14.0

Utilité minimale : 4

On remarque que l'utilité minimale reste la même pour tous, l'étudiant le moins satisfait a une utilité totale de 4, donc l'équité est respectée dans toutes les solutions. On peut voir que l'utilisation de l'algorithme de Gale-Shapley nous garantit une stabilité, cependant, il ne maximise pas la satisfaction globale. Tandis que les solutions aux questions 14 et 15 maximisent la satisfaction sans se soucier de la stabilité.

Si l'on souhaite une solution stable, il serait préférable d'utiliser les algorithmes de Gale-Shapley. Si l'on souhaite une solution qui maximise la satisfaction, la solution aux questions 14 et 15 serait préférable, et si l'on souhaite en plus de ceci, de la stabilité, la Q14 serait meilleure puisqu'elle possède moins de paires instables. Q13 reste la pire solution, elle est très instable et possède l'utilité moyenne la plus basse de toutes.

# QUESTION 12

Maximize
obj :x0_5 + x0_7 + x0_6 + x1_6 + x1_5 + x1_0 + x2_4 + x2_0 + x2_7 + x3_6 + x3_5 + x3_7 + x4_1 + x4_6 + x4_7 + x5_0 + x5_7 + x5_4 + x6_5 + x6_7 + x6_6 + x7_7 + x7_0 + x7_4 + x8_5 + x8_7 + x8_6 + x9_2 + x9_6 + x9_5 + x10_6 + x10_4 + x10_0

Subject To
c1: x0_5 + x0_7 + x0_6 = 1
c2: x1_6 + x1_5 + x1_0 = 1
c3: x2_4 + x2_0 + x2_7 = 1
c4: x3_6 + x3_5 + x3_7 = 1
c5: x4_1 + x4_6 + x4_7 = 1
c6: x5_0 + x5_7 + x5_4 = 1
c7: x6_5 + x6_7 + x6_6 = 1
c8: x7_7 + x7_0 + x7_4 = 1
c9: x8_5 + x8_7 + x8_6 = 1
c10: x9_2 + x9_6 + x9_5 = 1
c11: x10_6 + x10_4 + x10_0 = 1
c12: x0_0 + x1_0 + x2_0 + x3_0 + x4_0 + x5_0 + x6_0 + x7_0 + x8_0 + x9_0 + x10_0 <= 2
c13: x0_1 + x1_1 + x2_1 + x3_1 + x4_1 + x5_1 + x6_1 + x7_1 + x8_1 + x9_1 + x10_1 <= 1
c14: x0_2 + x1_2 + x2_2 + x3_2 + x4_2 + x5_2 + x6_2 + x7_2 + x8_2 + x9_2 + x10_2 <= 1
c15: x0_3 + x1_3 + x2_3 + x3_3 + x4_3 + x5_3 + x6_3 + x7_3 + x8_3 + x9_3 + x10_3 <= 1
c16: x0_4 + x1_4 + x2_4 + x3_4 + x4_4 + x5_4 + x6_4 + x7_4 + x8_4 + x9_4 + x10_4 <= 1
c17: x0_5 + x1_5 + x2_5 + x3_5 + x4_5 + x5_5 + x6_5 + x7_5 + x8_5 + x9_5 + x10_5 <= 1
c18: x0_6 + x1_6 + x2_6 + x3_6 + x4_6 + x5_6 + x6_6 + x7_6 + x8_6 + x9_6 + x10_6 <= 1
c19: x0_7 + x1_7 + x2_7 + x3_7 + x4_7 + x5_7 + x6_7 + x7_7 + x8_7 + x9_7 + x10_7 <= 1
c20: x0_8 + x1_8 + x2_8 + x3_8 + x4_8 + x5_8 + x6_8 + x7_8 + x8_8 + x9_8 + x10_8 <= 2

Binary
x0_5 x0_7 x0_6 x1_6 x1_5 x1_0 x2_4 x2_0 x2_7 x3_6 x3_5 x3_7 x4_1 x4_6 x4_7 x5_0 x5_7 x5_4 x6_5 x6_7 x6_6 x7_7 x7_0 x7_4 x8_5 x8_7 x8_6 x9_2 x9_6 x9_5 x10_6 x10_4 x10_0

End

# QUESTION 14 (maximise la somme des utilités)

Maximize
obj :2 x0_0 + 1 x0_1 + 3 x0_2 + 4 x0_3 + 0 x0_4 + 8 x0_5 + 6 x0_6 + 7 x0_7 + 5 x0_8 + 6 x1_0 + 0 x1_1 + 3 x1_2 + 1 x1_3 + 5 x1_4 + 7 x1_5 + 8 x1_6 + 4 x1_7 + 2 x1_8 + 7 x2_0 + 2 x2_1 +

5 x2_2 + 3 x2_3 + 8 x2_4 + 0 x2_5 + 1 x2_6 + 6 x2_7 + 4 x2_8 + 5 x3_0 + 1 x3_1 + 0 x3_2 + 2 x3_3 + 3 x3_4 + 7 x3_5 + 8 x3_6 + 6 x3_7 + 4 x3_8 + 4 x4_0 + 8 x4_1 + 3 x4_2 + 0 x4_3 + 2 x4_4 + 5 x4_5 + 7 x4_6 + 6 x4_7 + 1 x4_8 + 8 x5_0 + 2 x5_1 + 5 x5_2 + 3 x5_3 + 6 x5_4 + 0 x5_5 + 1 x5_6 + 7 x5_7 + 4 x5_8 + 2 x6_0 + 1 x6_1 + 5 x6_2 + 3 x6_3 + 0 x6_4 + 8 x6_5 + 6 x6_6 + 7 x6_7 + 4 x6_8 + 7 x7_0 + 2 x7_1 + 5 x7_2 + 3 x7_3 + 6 x7_4 + 0 x7_5 + 1 x7_6 + 8 x7_7 + 4 x7_8 + 2 x8_0 + 1 x8_1 + 5 x8_2 + 3 x8_3 + 0 x8_4 + 8 x8_5 + 6 x8_6 + 7 x8_7 + 4 x8_8 + 0 x9_0 + 3 x9_1 + 8 x9_2 + 4 x9_3 + 2 x9_4 + 6 x9_5 + 7 x9_6 + 1 x9_7 + 5 x9_8 + 6 x10_0 + 3 x10_1 + 1 x10_2 + 4 x10_3 + 7 x10_4 + 2 x10_5 + 8 x10_6 + 0 x10_7 + 5 x10_8 + 4 x0_0 + 5 x1_0 + 0 x2_0 + 6 x3_0 + 7 x4_0 + 8 x5_0 + 2 x6_0 + 10 x7_0 + 1 x8_0 + 9 x9_0 + 3 x10_0 + 4 x0_1 + 5 x1_1 + 0 x2_1 + 6 x3_1 + 7 x4_1 + 9 x5_1 + 1 x6_1 + 10 x7_1 + 2 x8_1 + 8 x9_1 + 3 x10_1 + 3 x0_2 + 4 x1_2 + 0 x2_2 + 10 x3_2 + 7 x4_2 + 8 x5_2 + 5 x6_2 + 6 x7_2 + 1 x8_2 + 9 x9_2 + 2 x10_2 + 4 x0_3 + 5 x1_3 + 0 x2_3 + 6 x3_3 + 7 x4_3 + 8 x5_3 + 3 x6_3 + 10 x7_3 + 1 x8_3 + 9 x9_3 + 2 x10_3 + 8 x0_4 + 1 x1_4 + 0 x2_4 + 9 x3_4 + 7 x4_4 + 6 x5_4 + 5 x6_4 + 4 x7_4 + 3 x8_4 + 2 x9_4 + 10 x10_4 + 9 x0_5 + 10 x1_5 + 3 x2_5 + 8 x3_5 + 7 x4_5 + 6 x5_5 + 5 x6_5 + 4 x7_5 + 0 x8_5 + 2 x9_5 + 1 x10_5 + 10 x0_6 + 9 x1_6 + 3 x2_6 + 8 x3_6 + 7 x4_6 + 6 x5_6 + 5 x6_6 + 4 x7_6 + 2 x8_6 + 0 x9_6 + 1 x10_6 + 3 x0_7 + 4 x1_7 + 0 x2_7 + 5 x3_7 + 6 x4_7 + 7 x5_7 + 9 x6_7 + 10 x7_7 + 1 x8_7 + 8 x9_7 + 2 x10_7 + 9 x0_8 + 10 x1_8 + 3 x2_8 + 8 x3_8 + 7 x4_8 + 6 x5_8 + 5 x6_8 + 4 x7_8 + 0 x8_8 + 2 x9_8 + 1 x10_8

Subject To
c1: x0_0 + x0_1 + x0_2 + x0_3 + x0_4 + x0_5 + x0_6 + x0_7 + x0_8 = 1
c2: x1_0 + x1_1 + x1_2 + x1_3 + x1_4 + x1_5 + x1_6 + x1_7 + x1_8 = 1
c3: x2_0 + x2_1 + x2_2 + x2_3 + x2_4 + x2_5 + x2_6 + x2_7 + x2_8 = 1
c4: x3_0 + x3_1 + x3_2 + x3_3 + x3_4 + x3_5 + x3_6 + x3_7 + x3_8 = 1
c5: x4_0 + x4_1 + x4_2 + x4_3 + x4_4 + x4_5 + x4_6 + x4_7 + x4_8 = 1
c6: x5_0 + x5_1 + x5_2 + x5_3 + x5_4 + x5_5 + x5_6 + x5_7 + x5_8 = 1
c7: x6_0 + x6_1 + x6_2 + x6_3 + x6_4 + x6_5 + x6_6 + x6_7 + x6_8 = 1
c8: x7_0 + x7_1 + x7_2 + x7_3 + x7_4 + x7_5 + x7_6 + x7_7 + x7_8 = 1
c9: x8_0 + x8_1 + x8_2 + x8_3 + x8_4 + x8_5 + x8_6 + x8_7 + x8_8 = 1
c10: x9_0 + x9_1 + x9_2 + x9_3 + x9_4 + x9_5 + x9_6 + x9_7 + x9_8 = 1
c11: x10_0 + x10_1 + x10_2 + x10_3 + x10_4 + x10_5 + x10_6 + x10_7 + x10_8 = 1
c12: x0_0 + x1_0 + x2_0 + x3_0 + x4_0 + x5_0 + x6_0 + x7_0 + x8_0 + x9_0 + x10_0 <= 2
c13: x0_1 + x1_1 + x2_1 + x3_1 + x4_1 + x5_1 + x6_1 + x7_1 + x8_1 + x9_1 + x10_1 <= 1
c14: x0_2 + x1_2 + x2_2 + x3_2 + x4_2 + x5_2 + x6_2 + x7_2 + x8_2 + x9_2 + x10_2 <= 1
c15: x0_3 + x1_3 + x2_3 + x3_3 + x4_3 + x5_3 + x6_3 + x7_3 + x8_3 + x9_3 + x10_3 <= 1
c16: x0_4 + x1_4 + x2_4 + x3_4 + x4_4 + x5_4 + x6_4 + x7_4 + x8_4 + x9_4 + x10_4 <= 1
c17: x0_5 + x1_5 + x2_5 + x3_5 + x4_5 + x5_5 + x6_5 + x7_5 + x8_5 + x9_5 + x10_5 <= 1
c18: x0_6 + x1_6 + x2_6 + x3_6 + x4_6 + x5_6 + x6_6 + x7_6 + x8_6 + x9_6 + x10_6 <= 1
c19: x0_7 + x1_7 + x2_7 + x3_7 + x4_7 + x5_7 + x6_7 + x7_7 + x8_7 + x9_7 + x10_7 <= 1
c20: x0_8 + x1_8 + x2_8 + x3_8 + x4_8 + x5_8 + x6_8 + x7_8 + x8_8 + x9_8 + x10_8 <= 2

Binary
x0_0 x0_1 x0_2 x0_3 x0_4 x0_5 x0_6 x0_7 x0_8 x1_0 x1_1 x1_2 x1_3 x1_4 x1_5 x1_6 x1_7 x1_8 x2_0 x2_1 x2_2 x2_3 x2_4 x2_5 x2_6 x2_7 x2_8 x3_0 x3_1 x3_2 x3_3 x3_4 x3_5 x3_6

x3_7 x3_8 x4_0 x4_1 x4_2 x4_3 x4_4 x4_5 x4_6 x4_7 x4_8 x5_0 x5_1 x5_2 x5_3 x5_4 x5_5 x5_6 x5_7 x5_8 x6_0 x6_1 x6_2 x6_3 x6_4 x6_5 x6_6 x6_7 x6_8 x7_0 x7_1 x7_2 x7_3 x7_4 x7_5 x7_6 x7_7 x7_8 x8_0 x8_1 x8_2 x8_3 x8_4 x8_5 x8_6 x8_7 x8_8 x9_0 x9_1 x9_2 x9_3 x9_4 x9_5 x9_6 x9_7 x9_8 x10_0 x10_1 x10_2 x10_3 x10_4 x10_5 x10_6 x10_7 x10_8

End

# QUESTION 15 (maximise la somme des utilités (pour k=5)

Maximize
obj :17 x0_5 + 10 x0_7 + 16 x0_6 + 14 x0_8 + 8 x0_3 + 17 x1_6 + 17 x1_5 + 11 x1_0 + 6 x1_4 + 8 x1_7 + 8 x2_4 + 7 x2_0 + 6 x2_7 + 5 x2_2 + 7 x2_8 + 16 x3_6 + 15 x3_5 + 11 x3_7 + 11 x3_0 + 12 x3_8 + 15 x4_1 + 14 x4_6 + 12 x4_7 + 12 x4_5 + 11 x4_0 + 16 x5_0 + 14 x5_7 + 12

x5_4 + 13 x5_2 + 10 x5_8 + 13 x6_5 + 16 x6_7 + 11 x6_6 + 10 x6_2 + 9 x6_8 + 18 x7_7 + 17 x7_0 + 10 x7_4 + 11 x7_2 + 8 x7_8 + 8 x8_5 + 8 x8_7 + 8 x8_6 + 6 x8_2 + 4 x8_8 + 17 x9_2 + 7 x9_6 + 8 x9_5 + 7 x9_8 + 13 x9_3 + 9 x10_6 + 17 x10_4 + 9 x10_0 + 6 x10_8 + 6 x10_3

Subject To
c1: x0_5 + x0_7 + x0_6 + x0_8 + x0_3 = 1
c2: x1_6 + x1_5 + x1_0 + x1_4 + x1_7 = 1
c3: x2_4 + x2_0 + x2_7 + x2_2 + x2_8 = 1
c4: x3_6 + x3_5 + x3_7 + x3_0 + x3_8 = 1
c5: x4_1 + x4_6 + x4_7 + x4_5 + x4_0 = 1
c6: x5_0 + x5_7 + x5_4 + x5_2 + x5_8 = 1
c7: x6_5 + x6_7 + x6_6 + x6_2 + x6_8 = 1
c8: x7_7 + x7_0 + x7_4 + x7_2 + x7_8 = 1
c9: x8_5 + x8_7 + x8_6 + x8_2 + x8_8 = 1
c10: x9_2 + x9_6 + x9_5 + x9_8 + x9_3 = 1
c11: x10_6 + x10_4 + x10_0 + x10_8 + x10_3 = 1
c12: x0_0 + x1_0 + x2_0 + x3_0 + x4_0 + x5_0 + x6_0 + x7_0 + x8_0 + x9_0 + x10_0 <= 2
c13: x0_1 + x1_1 + x2_1 + x3_1 + x4_1 + x5_1 + x6_1 + x7_1 + x8_1 + x9_1 + x10_1 <= 1
c14: x0_2 + x1_2 + x2_2 + x3_2 + x4_2 + x5_2 + x6_2 + x7_2 + x8_2 + x9_2 + x10_2 <= 1
c15: x0_3 + x1_3 + x2_3 + x3_3 + x4_3 + x5_3 + x6_3 + x7_3 + x8_3 + x9_3 + x10_3 <= 1
c16: x0_4 + x1_4 + x2_4 + x3_4 + x4_4 + x5_4 + x6_4 + x7_4 + x8_4 + x9_4 + x10_4 <= 1
c17: x0_5 + x1_5 + x2_5 + x3_5 + x4_5 + x5_5 + x6_5 + x7_5 + x8_5 + x9_5 + x10_5 <= 1
c18: x0_6 + x1_6 + x2_6 + x3_6 + x4_6 + x5_6 + x6_6 + x7_6 + x8_6 + x9_6 + x10_6 <= 1
c19: x0_7 + x1_7 + x2_7 + x3_7 + x4_7 + x5_7 + x6_7 + x7_7 + x8_7 + x9_7 + x10_7 <= 1
c20: x0_8 + x1_8 + x2_8 + x3_8 + x4_8 + x5_8 + x6_8 + x7_8 + x8_8 + x9_8 + x10_8 <= 2

Binary
x0_0 x0_1 x0_2 x0_3 x0_4 x0_5 x0_6 x0_7 x0_8 x1_0 x1_1 x1_2 x1_3 x1_4 x1_5 x1_6 x1_7 x1_8 x2_0 x2_1 x2_2 x2_3 x2_4 x2_5 x2_6 x2_7 x2_8 x3_0 x3_1 x3_2 x3_3 x3_4 x3_5 x3_6 x3_7 x3_8 x4_0 x4_1 x4_2 x4_3 x4_4 x4_5 x4_6 x4_7 x4_8 x5_0 x5_1 x5_2 x5_3 x5_4 x5_5 x5_6 x5_7 x5_8 x6_0 x6_1 x6_2 x6_3 x6_4 x6_5 x6_6 x6_7 x6_8 x7_0 x7_1 x7_2 x7_3 x7_4 x7_5 x7_6 x7_7 x7_8 x8_0 x8_1 x8_2 x8_3 x8_4 x8_5 x8_6 x8_7 x8_8 x9_0 x9_1 x9_2 x9_3 x9_4 x9_5 x9_6 x9_7 x9_8 x10_0 x10_1 x10_2 x10_3 x10_4 x10_5 x10_6 x10_7 x10_8

End