Écrit Blanc d'informatique
Préparation au CAPES de Mathématiques
20 mars 2017
Durée 5h

Instructions :

1. Les trois problèmes sont indépendants et *doivent* être traités sur des copies séparées.
2. La clarté et la précision des réponses font partie intégrante de l'évaluation.
3. Graphe sans dessin, pas de point. ...

Sources :

— Les problèmes SAT-color et SAT-stable sont des adaptations des examens de complexité et calculabilité de M1 à Grenoble, avec l'aimable autorisation de leurs auteurs.
— Le problème SAT-sudoku est repris du projet d'Emmanuel COQUERY en L3 informatique

Préambule

On rappelle les notions de base suivantes sur les graphes et la logique propositionnelle, suffisantes pour la bonne compréhension des trois problèmes à suivre.

Graphes

- Un graphe (orienté) G est un couple (S,A) où S est un ensemble fini de sommets et $A \subseteq S \times S$ est l'ensemble de ses arcs.
- Deux sommets x et y d'un graphe $G=(S,A)$ seront dits *adjacents* si et seulement si $(x, y) \in A$ ou $(y, x) \in A$ (i.e., s'il existe un arc entre les deux sommets). L'adjacence est donc une relation symétrique.
- Soit un ensemble C fini de k *couleurs*. Un *k-coloriage* d'un graphe G est une fonction qui affecte une et une seule couleur à chaque sommet. Il est "correct" si et seulement si deux sommets adjacents n'ont pas la même couleur. Un graphe est *k-coloriable* s'il existe un *k-coloriage* correct pour ce graphe.

Logique propositionnelle

- Étant donné un vocabulaire logique V constitué de v variables propositionnelles $p_1, \ldots, p_v$, une conjonction de clauses est une formule logique de la forme $E_1 \wedge \ldots \wedge E_i \ldots \wedge E_m$. Chaque clause E_i est une disjonction de littéraux (différents deux à deux) $l_{i_1} \vee \ldots \vee l_{i_{k_i}}$, et un littéral est soit une variable propositionnelle p de V soit la négation $\neg p'$ d'une variable propositionnelle p' de V.
- Une interprétation I d'un ensemble $V = \{p_1, \ldots, p_v\}$ de variables propositionnelles est une fonction qui associe à chaque variable propositionnelle p_i la valeur $Vrai$ ou $Faux$.
  - L'évaluation d'une conjonction de clauses dans une interprétation I de ses variables propositionnelles renvoie $Vrai$ si *toutes* les clauses sont évaluées à $Vrai$ dans I, et $Faux$ sinon.
  - L'évaluation d'une clause dans une interprétation I renvoie $Vrai$ si *au moins* un de ses littéraux est évalué à $Vrai$ dans I et $Faux$ sinon.
  - Un littéral positif p_i est évalué à $Vrai$ dans I si $I(p_i) = Vrai$, à $Faux$ sinon.
  - Un littéral négatif $\neg p_i$ est évalué à $Faux$ dans I si $I(p_i) = Vrai$, à $Vrai$ sinon.
- Une conjonction de clauses est satisfaisable s'il existe une interprétation de ses variables propositionnelles dans laquelle toutes les clauses sont évaluées à $Vrai$.
  Exemple : La conjonction de clauses $(\neg p_1 \vee p_3) \wedge (p_2 \vee \neg p_3) \wedge (\neg p_2)$
  - est évaluée à $Vrai$ dans l'interprétation I définie par : $I(p_1) = I(p_2) = I(p_3) = Faux$,
  - est évaluée à $Faux$ dans l'interprétation I' : $I'(p_1) = I'(p_2) = Faux, I'(p_3) = Vrai$.

  Elle est satisfaisable (I suffit pour le prouver).

NP-complétude, réduction polynomiale Un problème $\mathcal{L}_2$ est NP-complet s'il est dans NP et s'il est NP-dur, i.e, si tout problème dans NP peut être réduit à $\mathcal{L}_2$, ce qui est équivalent à montrer qu'il existe une réduction polynômiale d'un problème NP-complet $\mathcal{L}_1$ *connu* vers $\mathcal{L}_2$.

On rappelle la définition formelle d'une réduction polynomiale d'un langage vers un autre langage : Une réduction polynomiale de L_1 *vers* L_2 est une fonction $f : \Sigma^* \to \Sigma^*$ *calculable en temps polynomial* telle que $x \in \mathcal{L}_1$ ssi $f(x) \in \mathcal{L}_2$ (transfert des solutions).

Problème 1 : SAT et coloriage

On code la relation d'adjacence associée à un graphe G de n sommets par un tableau T de taille $n \times n$ tel que : $T[i,j] = 1$ si les sommets i et j sont adjacents (cf. préambule), et $T[i,j] = 0$ sinon. On code un k-coloriage de G par un tableau d'entiers C à une dimension de taille n tel que $C[i]$ vaut la couleur affectée au sommet i dans le coloriage.

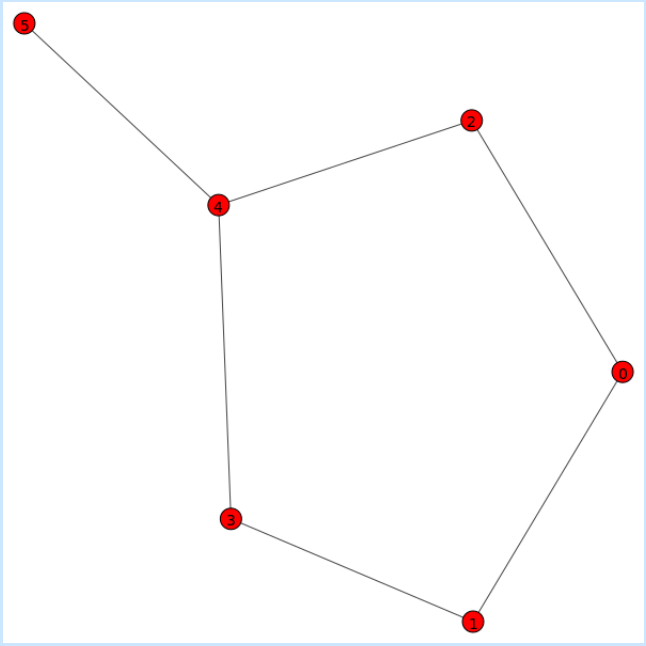
Question #1

Dessiner le graphe dont la matrice d'adjacence est déclarée par : (on numérote les sommets de 0 à 5, les indices débutent à 0) :

```
adj = np.array ([[0, 1, 1, 0, 0, 0],
  [1, 0, 0, 1, 0, 0],
  [1, 0, 0, 0, 1, 0],
  [0, 1, 0, 0, 1, 0],
  [0, 0, 1, 1, 0, 1],
  [0, 0, 0, 0, 1, 0]])
```

Dans la suite, nous nommons ce graphe G_1.

Solution: Et voici (merci le module `igraph`) :



Question #2

Écrire un programme Python prenant en entrée la matrice d'adjacence d'un graphe, un tableau représentant un coloriage, le nombre de noeuds du graphe, et qui vérifie si ce coloriage est correct :

```
coloriage=[1,1,2,2,1,0]
print is_ok (adj,coloriage,6)
#False
```

```
coloriage=[0,1,2,3,4,5]
print is_ok (adj,coloriage,6)
#True
```

Solution:

```
def is_ok (mat_adj,colortab,nbnodes):
    for i in range(0,nbnodes-1):
        colori=colortab[i]
        for j in range(0,i):
            if mat_adj[i,j]==1 and colortab[j]==colori:
                return False
    return True
```

Question #3

Quel est l'ordre de grandeur de la complexité de ce programme en fonction du nombre n de sommets du graphe ?

Solution: $O(n^2)$

Une instance du problème de $k-$*coloriabilité* d'un graphe est la donnée d'un graphe et d'un entier, ie une paire (G, k). La question associée est "existe-t-il un k-coloriage pour G ?"

Question #4

Montrer que ce problème est NP.

Solution: La solution est aussi largement inspirée de celle des collègues Grenoblois ! On montre que l'on peut tester une solution en temps polynomial. On vérifie facilement qu'un coloriage est valide avec l'algorithme précédent. Il reste à compter le nombre de couleurs différentes, ce qui est aussi polynomial.

Intéressons nous maintenant à la 3-coloriabilité : étant donné un graphe G à n sommets, on modélise par la variable propositionnelle $x_{i,c}$ ($i \in [0..n-1]$ et $c \in [0..2]$) que le sommet i est colorié avec la couleur de numéro c.

Question #5

(a) Traduire par une conjonction de clauses que chaque sommet est colorié par une et une et une seule des 3 couleurs.
Indication : on exprimera par la clause $K(i)$ que le sommet i est colorié en au moins une des 3 couleurs. On exprimera ensuite par la clause $U(i)$ (respectivement $V(i)$ et $W(i)$) que le sommet i ne peut pas être colorié à la fois par la couleur 0 et la couleur 1 (respectivement la couleur 0 et la couleur 2, la couleur 1 et la couleur 2).

Solution: $K(i) = x_{i,0} \vee x_{i,1} \vee x_{i,2}$
$U(i) = \neg x_{i,0} \vee \neg x_{i,1}$
$V(i) = \neg x_{i,0} \vee \neg x_{i,2}$
$W(i) = \neg x_{i,1} \vee \neg x_{i,2}$

(b) Écrire cette conjonction de clauses dans le cas particulier du graphe G_1 de la question 1.*Pour des raisons de longueur, vous pouvez vous restreindre aux sommets 3 et 4.*

Solution: Pour le sommet 3 :

$$(x_{3,1} \vee x_{3,2} \vee x_{3,3}) \wedge (\neg x_{3,0} \vee \neg x_{3,1}) \wedge (\neg x_{3,0} \vee \neg x_{3,2}) \wedge (\neg x_{3,2} \vee \neg x_{3,1})$$

(c) Traduire par une conjonction de clauses que deux sommets voisins n'ont pas la même couleur.

Solution: On fait la conjonction des clauses $D(e, c)$ telles que, pour chaque arête e (d'extrémités u et v), et chaque couleur c, $D(e, c) = \neg x_{u,c} \vee \neg x_{v,c}$.

(d) Écrire cette conjonction de clauses dans le cas particulier du graphe G_1 de la question 1. *Pour des raisons de longueur, vous pouvez vous restreindre au sommet 4 et à ses arêtes adjacentes.*

Solution: On énumère les arêtes qui touchent le sommet le sommet 4, 3-4,2-4,4-5, et les couleurs $0, 1, 2$: $D(3-4, 0) = \neg x_{3,0} \vee \neg x_{4,0}$, ...

Question #6

En déduire l'existence d'une réduction polynômiale du problème de 3-coloriabilité de graphes vers SAT.

Solution: Soit n le nombre de sommets, la construction et la taille de la conjonction des clauses $K(i)$, $U(i)$, $V(i)$ et $W(i)$ est : $(3 + 2 + 2 + 2)n$.
La construction et la taille de la conjonction des clauses $D(e, c)$ sont inférieures ou égales à $2 \times 3 \times n^2$. Donc la transformation a une complexité en $O(n^2)$.

Question #7

Peut-on en déduire que le problème de 3-coloriabilité de graphes est NP-complet ?

Solution: On ne peut pas déduire de la transformation précédente (bien que de coût polynômial) que le problème de 3-coloriabilité est NP-complet, car il faudrait une transformation de coût polynômial de SAT vers le problème de 3-coloriabilité, alors qu'on a une transformation de coût polynômial dans le sens inverse.

Étant donnée une instance ϕ de 3-SAT, c'est-à-dire une conjonction $E_0 \wedge \ldots \wedge E_i \ldots \wedge E_{m-1}$ de m clauses, construite à partir de v variables propositionnelles $(p_0, \ldots, p_{v-1})$, où chaque clause contient exactement 3 littéraux (différents), on construit le graphe G_ϕ à $3v+m$ sommets défini de la manière suivante :

— Ses sommets sont $p_0, \ldots, p_{v-1}$, $\neg p_0, \ldots, \neg p_{v-1}$, $y_0, \ldots, y_{v-1}$, $E_0, \ldots, E_{m-1}$.
— Chaque sommet p_i est relié au sommet $\neg p_i$.
— Chaque sommet y_i est relié :
  * à tous les sommets y_j tels que $j \neq i$,
  * à tous les sommets p_j tels que $j \neq i$,
  * et à tous les sommets $\neg p_j$ tels que $j \neq i$.
— le sommet p_i est relié au sommet E_j si p_i *n'est pas un littéral* de la clause E_j.
— le sommet $\neg p_i$ est relié au sommet E_j si $\neg p_i$ *n'est pas un littéral* de la clause E_j.

Dans la suite on considérera l'ordre des sommets du graphe de taille $3v + m$ dans l'ordre suivant : d'abord les p_i, puis les $\neg p_i$, puis les y_i, puis les E_i. p_0 est donc le sommet de numéro 0, $\neg p_0$ le sommet de numéro v, ...

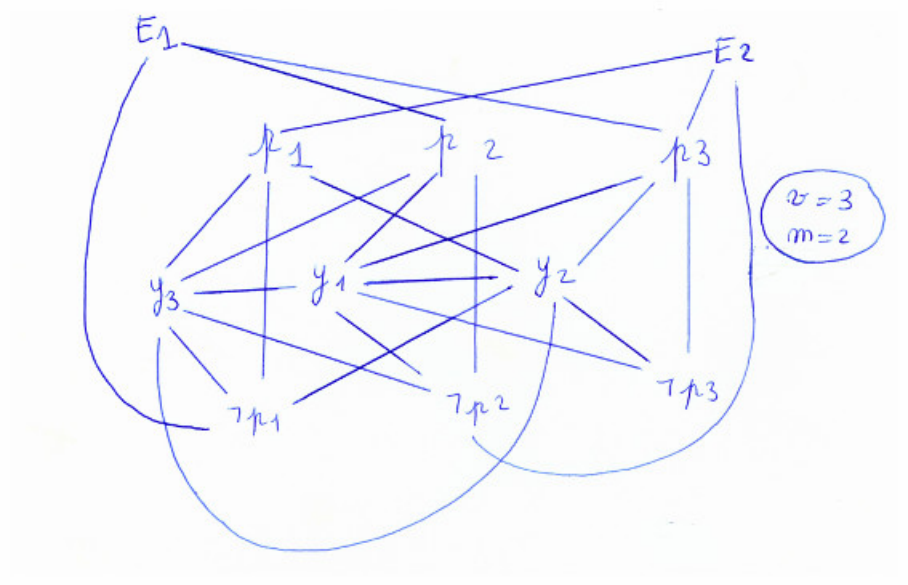
Question #8

Soit ϕ_1 l'instance de 3-SAT suivante :

$$(p_0 \vee \neg p_1 \vee \neg p_2) \wedge (\neg p_0 \vee p_1 \vee \neg p_2).$$

Que valent v, m ? Dessiner le graphe G_{ϕ_1} et donner sa matrice d'adjacence. **Rendre la réponse sur la feuille d'accompagnement fournie avec le sujet, sur laquelle on a déjà préparé le dessin et la matrice**

Solution: Voici le graphe : attention celui qu'on veut a des indices décalés, p_1 est en fait p_0, etc (flemme du correcteur).



et sa matrice d'adjacence

$$\left(\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}0&1\\1&0\\1&1\end{pmatrix}\right.$$
$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}1&0\\0&1\\0&0\end{pmatrix}$$
$$\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}0&0&0\\0&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&0\\0&0\\0&0\end{pmatrix}$$
$$\left.\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix}\begin{pmatrix}0&0\\0&0\\0&0\end{pmatrix}\right)$$

On code en Python une instance de 3-SAT ϕ (avec les notations précédentes) sous la forme d'une matrice de F taille $m \times v$ telle que $F[i,j]$ vaut :
— 1 si p_i est un littéral de la clause E_j,
— -1 si $\neg p_i$ est un littéral de la clause E_j,
— et 0 si ni p_i ni $\neg p_i$ n'est un littéral de la clause E_j,

Question #9

(a) Que vaut F dans le cas de la formule ϕ_1 ? On nomme cette matrice F_1.

Solution:

$$F_1 = \begin{pmatrix}1&-1\\-1&1\\-1&-1\end{pmatrix}$$

(b) Quel est le numéro du noeud représentant la clause E_j ?

Solution: La clause E_0 est représentée par le noeud $3v$, donc E_j par $3v+j$.

(c) Coder dans un programme Python la transformation qui prend le codage d'une instance de 3-sat (F, matrice de taille $m \times v$) et qui construit en sortie la matrice codant le graphe correspondant :

```
def transforme_formule(F,m,v):
    T=np.zeros((3*v+m,3*v+m),dtype=np.int32) #init de T avec
        des 0
    #à compléter
    return T
```

L'appel `transforme_formule(F1,..,..)` doit retourner la matrice d'adjacence de G_{ϕ_1} obtenue à la question 8.

Solution:

```
def transforme_formule(F,m,v):
    T=np.zeros((3*v+m,3*v+m),dtype=np.int32) #T avec des
        zeros dedans
    for i in range(0,v): #-->  v-1
```

```
        print i
        T[i,i+v]=1 #pi <-> neg pi
        T[i+v,i]=1 #pi <-> neg pi
        for j in range(0,v): #liaisons ?i,?j
            if j!=i:
                T[2*v+i,2*v+j]=1 #y_j <-> y_i
                T[2*v+j, i]=1    #y_j <-> p_i
                T[i, 2*v+j]=1
                T[2*v+j, v+i]=1  #y_j <-> neg pi
                T[v+i, 2*v+j]=1
        for j in range (0,m):  #les fleches Ej<---> qq
            if F[i,j]==0: #pi,non pi ne sont pas litteraux de
                Ej
                T[i, 3*v+j]=1
                T[3*v+j, i]=1
                T[v+i, 3*v+j]=1
                T[3*v+j, v+i]=1
            elif F[i,j]==-1:
                T[i, 3*v+j]=1   # pi <-> Ej
                T[3*v+j, i]=1
            else:
                T[v+i, 3*v+j]=1 # neg pi <-> Ej
                T[3*v+j, v+i]=1
    return T
```

(d) Quelle est la taille de l'entrée, et le nombre d'instructions effectuées par votre programme en fonction de m et v ?

Solution:
La taille de l'entrée est $n = m \times v$, et le nombre d'instructions exécutées par le programme est :
$T(n) = (3v+m)^2 + v \times (2 + (v-1) \times 5 + m \times 5) = 14v^2 + 11mv + m^2 - 3v$.

Question #10

Montrer que si $k < v$, alors il n'existe pas de k-coloriage correct du graphe.

Solution: Les v sommets $y_1, \ldots, y_v$ sont tous reliés 2 à 2 et donc doivent être coloriés avec des couleurs distinctes 2 à 2.

Question #11

On considère un $(v+1)$-coloriage qui associe la couleur i à l'un des deux membres de chaque paire de sommets $\{p_i, \neg p_i\}$, et la couleur $v+1$ à l'autre.
Montrer que si $v > 4$, alors, dans tout $(v+1)$-coloriage correct qui étend aux sommets E_j le $(v+1)$-coloriage précédent, aucun sommet E_j n'a la couleur $v+1$.

Solution: Supposons $v > 4$. Puisque chaque clause E_i contient 3 littéraux, chaque sommet E_i est relié à p_j et $\neg p_j$ pour au moins une valeur de j. Or, p_j ou $\neg p_j$ a la couleur $v+1$. Donc,

on ne peut pas associer cette couleur à E_i.

Question #12

Montrer comment associer une couleur parmi $1, \ldots, v$ à chaque sommet E_j de sorte que G soit $(v+1)$-coloriable si et seulement si l'ensemble de toutes les clauses E_j est satisfaisable. On pourra raisonner sur un exemple.

Solution: On peut donner à chaque sommet E_j la couleur i associée à p_i ou $\neg p_i$ si p_i est une variable propositionnelle qui apparaît dans E_j.

Si l'ensemble des clauses E_j est satisfaisable, dans chaque clause E_j, il existe un littéral interprété à $Vrai$, qui correspond à p_i ou $\neg p_i$. Le $v+1$-coloriage précédent est correct et donc le graphe correspondant est $v+1$-coloriable.

Inversement, si le graphe correspondant à la conjonction de clauses est $v+1$-coloriable, tout $v+1$ coloriage correct vérifie les contraintes mises en évidence dans les questions précédentes.

Pour chaque sommet E_j, il existe un sommet p_i ou $\neg p_i$ qui est colorié avec la même couleur que E_j et qui est un littéral de la clause E_j. L'interprétation qui associe la valeur $Vrai$ à chacun de ces littéraux satisfait donc la conjonction des clauses E_j.

Question #13

Peut-on en déduire que le problème de k-coloriage d'un graphe est NP-complet ?

Solution: Oui, car on a explicité une réduction polynômiale des instances de 3-SAT à v variables propositionnelles vers des instances du problème de $(v+1)$-coloriage de graphes

Problème 2 : Sudoku et SAT

Le problème du Sodoku consiste à placer les nombres $1 \dots n^2$ sur une grille $n^2 \times n^2$ (pour le sudoku classique $n = 3$) dont certaines valeurs sont fixées, voir par exemple la figure 1.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

FIGURE 1 – Problème de Sudoku (gauche) et sa solution (droite)

Un problème de Sudoku sera représenté par un tableau représentant la grille initiale à compléter :
— ce tableau contiendra n^2 lignes non vides ;
— chaque ligne non vide représente une ligne de la grille contenant n^2 nombres ;
— les nombres strictement positifs correspondent aux cases déjà remplies ;
— les 0 correspondent aux cases à remplir.

Les régions sont des sous-grilles carrées de taille $n \times n$ découpant la grille initiale en $n \times n$ parties disjointes. On suppose que les lignes et les colonnes sont numérotées à partir de 0 en partant du haut et de la gauche. Les coordonnées du coin supérieur gauche de chaque région sont ainsi de la forme $(i \times n, j \times n)$ où $0 \leq i \leq n-1$ et $0 \leq j \leq n-1$.

Compléter une grille de Sudoku se fait en respectant les contraintes suivantes :
— Les nombres sur une même ligne sont tous différents ;
— Les nombres sur une même colonne sont tous différents ;
— Les nombres dans une même région sont tous différents ;
— Les nombres placés sur la grille sont tous compris entre 1 et n^2 inclus.

A partir de la représentation d'un problème de Sudoku, on va transformer ce problème en un problème SAT, le résoudre avec un solveur, puis convertir le modèle trouvé (quand il existe) en solution.

Afin de coder le problème de Sudoku sous forme de formule conjonctive, on introduit n^2 variables booléennes par case de la grille. On note $p^v_{i,j}$ la variable la $v$ième variable de la case située à la position (i, j) dans la grille. On utilisera la convention que la variable $p^v_{i,j}$ prend la valeur $Vrai$ si et seulement si la case (i, j) est remplie avec la valeur v.

Pour la représentation mémoire des variables, on doit trouver une bijection entre les $(n^2)^3$ variables du problème et les entiers naturels (soit 729 variables pour le Sudoku usuel). On suppose que l'on numérote les variables comme suit, où $num(p^v_{i,j})$ est le numéro de la variable $p^v_{i,j}$:
— $num(p^1_{0,0}) = 0$

— $num(p^v_{i,j}) = num(p^{v-1}_{i,j}) + 1$ si $v \geq 2$
— $num(p^1_{i,j}) = num(p^{n^2}_{i,j-1}) + 1$ si $j \geq 1$
— $num(p^1_{i,0}) = num(p^{n^2}_{i-1,n^2-1}) + 1$ si $i \geq 1$

Autrement dit : les variables d'une même case sont numérotées consécutivement. Le numéro de la première variable d'une case suit immédiatement le numéro de la dernière variable de la case précédente, les cases étant considérées lignes par ligne.

Question #1

Dessiner une grille de Sudoku à remplir pour le cas $n = 2$ en donnant son codage dans le format de tableau décrit. S'assurer que votre grille a bien une solution.

Question #2

Calculer $num(p^4_{0,0})$, $num(p^4_{0,3})$ et $num(p^4_{3,3})$ sur une grille 4×4 (cas $n = 2$).

Question #3

Donner le code Python d'une fonction `code(n, i, j, v)` qui calcule *en temps constant* le numéro de la variable codant la valeur v de la case (i, j) pour le cas d'une grille de taille $n^2 \times n^2$.

Question #4

Similairement, donner le code Python d'une fonction `decode(n, p)` qui calcule v, i et j tels que $num(n, p) = p^v_{i,j}$.

Afin de coder le problème de Sudoku sous forme d'un ensemble de clauses, on introduit une contrainte appelée $\mathsf{OneInN}(V)$ pour *exactement 1 vraie parmi l'ensemble V de littéraux donnés.* Une contrainte peut être vue comme un ensemble de clauses contraignant les valeurs des littéraux. Dans le cas de $\mathsf{OneInN}(V)$, cet ensemble de clauses va imposer que parmi l'ensemble V de littéraux, toute interprétation qui satisfait cet ensemble a *exactement un seul* littéral affecté à $Vrai$ et tous les autres à $Faux$.

Question #5

Donner le code Python d'une fonction `oneInN(V)`, avec V un tableau d'entiers de taille $|V| = q$ représentant des littéraux. Un littéral positif est représenté par sa variable et un littéral négatif est représenté par l'opposé de sa variable, par exemple, `[-2, 0, 3]` représente l'ensemble de littéraux $\{x_0, \neg x_2, x_3\}$. La fonction `oneInN(V)` génère un ensemble de clauses codant la contrainte $\mathsf{OneInN}(V)$. Un ensemble de clauses sera représenté comme dans le problème précédent. Il faudra exprimer :

— qu'au moins un littéral est vrai avec une clause ;
— que deux littéraux ne peuvent pas être à $Vrai$ simultanément avec un ensemble de $q \times (q - 1)/2$ clauses binaires.

Question #6

Calculer `oneInN([-2, 0, 3])`.

Question #7

Justifier le codage précédent en prouvant que I est un modèle du problème ϕ produit par `oneInN(V)` si et seulement si un seul des littéraux de V est $Vrai$.

Question #8

En Python, définir une fonction `oneval(n, i, j)` qui construit un tableau V et utilise `oneInN(V)` pour générer un ensemble de clauses qui assure qu'une case (i, j) est remplie et que l'on n'a pas placé 2 nombres différents dans cette case.

Question #9

En Python, définir une fonction `onerow(n, i, v)` qui construit un tableau V et utilise `oneInN(V)` pour générer un ensemble de clauses qui assure que la ligne i contient une et une seule fois la valeur v.

Question #10

En Python, définir une fonction `onecol(n, j, v)` qui construit un tableau V et utilise `oneInN(V)` pour générer un ensemble de clauses qui assure que la colonne j contient une et une seule fois la valeur v.

Question #11

En Python, définir une fonction `oneregion(n, i, j, v)` qui construit un tableau V et utilise `oneInN(V)` pour générer un ensemble de clauses qui assure que la région dont le coin supérieur gauche est la case $(i \times n, j \times n)$ contient une et une seule fois la valeur v.

Question #12

En Python, définir une fonction `constraints(n)` qui génère l'ensemble de toutes les clauses qui contraignent une grille de Sudoku pour que chaque ligne, chaque colonne et chaque région contienne toutes les valeurs de 1 à n^2.

Question #13

Pour une grille de taille $n^2 \times n^2$, combien de clauses seront produites par `constraints(n)` ?

Question #14

On suppose un tableau T désignant une grille de Sudoku tel que défini plus tôt. En Python, définir une fonction `gensat(n, T)` qui complète le codage du problème de Sudoku d'une grille donnée.

On suppose l'existence d'une fonction `solve(T)` qui prend un ensemble de clauses représenté par le tableau T et renvoie le premier modèle trouvé de T. Pour un problème à n variables représentées par les entiers de 0 à $n-1$, un modèle retourné est représenté par un tableau $[v_0, v_1, \ldots, v_{n-1}]$ où v_i vaut 1 si la variable est à $Vrai$ dans l'interprétation et -1 si la variable est à $Faux$. Si le problème n'a pas de modèle, alors le tableau vide [] est retourné.

Question #15

En Python, définir une fonction `enumsat(T)` qui énumère *tous les modèles* d'un problème T.

Question #16

Dans le cas général, donner une grille de Sudoku qui a plusieurs modèles, c'est-à-dire plusieurs solutions. En pratique, par exemple pour une grille de Sudoku trouvée dans un magazine, est-ce aussi le cas ? Justifier.

Question #17

En Python, définir une fonction `sudoku(n, T)` qui prend une grille de Sudoku, la code en SAT, résout le problème associée et décode le modèle pour retourner la grille correctement remplie.

La propagation unitaire consiste à déduire des valeurs $Vrai$ ou $Faux$ à partir de l'affectation courante de valeurs aux variables dans la résolution d'un problème SAT selon le principe suivant : si tous les littéraux d'une clause sauf un, disons ℓ, sont faux, alors pour que la clause puisse s'évaluer à vrai il est nécessaire que ℓ prenne la valeur $Vrai$. Une telle clause est dite *unitaire*.

On peut remarquer que les littéraux déduits (i.e., ceux que l'on a affecté à $Vrai$ suite à la déduction précédente) peuvent eux-même déclencher de nouvelles propagations, c'est-à-dire permettre de déduire d'autre propagations.

Question #18

Expliquer pourquoi l'implantation de la propagation unitaire dans `solve(T)` permettra de radicalement augmenter la performance de la résolution d'un problème de Sudoku.

Problème 3 : SAT et STABLE

Un ensemble de sommets $V' \subseteq V$ est un ensemble *stable* si deux sommets quelconques de V' ne sont pas joints par une arête : pour tout $u \in V'$ et pour tout $v \in V'$, $(u, v) \notin E$.

Le problème STABLE se définit de la façon suivante : étant donnés un graphe non orienté $G = (V, E)$ et un entier $k \leq |V|$, déterminer si le graphe contient un *ensemble stable* de taille au moins égale à k.

Question #1

Donner un exemple de graphe à 4 sommets contenant un ensemble stable de taille 3. Donner ensuite un exemple d'un graphe à 4 sommets ne contenant pas d'ensemble stable de taille 3.

Solution: Toute cette solution est largement copiée des collègues Grenoblois, au python près.
L'ensemble {1,2,3} est stable pour le graphe dont les arêtes sont : {(1,4), (2,4, (3,4)}.
Le graphe suivant ne contient pas d'ensemble stable de taille 3 : {(1,4), (2,4), (3,4), (1,2)}.

Question #2

Écrire un programme Python qui prend en entrée le codage d'un graphe G à n sommets (sous forme d'une matrice d'adjacence comme dans le problème 1) et un sous-ensemble $\{i_1, \ldots, i_k\}$ de $[0 \ldots n-1]$ et qui vérifie si $\{i_1, \ldots, i_k\}$ est un ensemble stable de G. Ce sous-ensemble est lui même codé par un tableau nommé `sousEns`, de taille k (indexé de 0 à $k-1$) :

```
def is_stable(G,n,sousEns,k):
  #à vous !
```

Solution: Dès que l'on trouve une arête entre éléments de `sousEns`, on peut conclure que ce n'est pas un stable :

```
def is_stable(G,n,sousEns,k):
    for j in range(0,k-1):
        for m in range(j,k-1):
            if E[j,m]==1:
                return False
    return True
```

Question #3

Donner la complexité de l'algorithme précédent et en déduire que STABLE est dans NP.

Solution: L'algorithme précédent teste si un sous-ensemble donné de taille k est un ensemble stable en $\Theta(k^2)$ donc en $O(n^2)$. Il est donc de complexité polynomiale, ce qui prouve l'appartenance du problème à NP.

Question #4

Rappelons qu'une instance de 3-SAT est une conjonction de clauses telle que chaque clause est une disjonction d'exactement 3 littéraux, tous distincts 2 à 2.

On considère la transformation suivante d'une instance de 3-SAT à k clauses en un graphe à 3k sommets (chaque littéral d'une clause fournit donc un sommet, une variable ou sa négation peut donc apparaître plusieurs fois dans le graphe)

— toute clause $C_i = x_1 \vee x_2 \vee x_3$ est transformée en un triangle (graphe complet à 3 sommets) dont les sommets sont étiquetés x_1^i, x_2^i et x_3^i ;

— une arête supplémentaire est ajoutée entre chaque paire de sommets x_j^i et $\neg x_j^{i'}$.

Question #5

Dessiner le graphe résultant de la transformation de la conjonction de clauses

$$C_1 \wedge C_2 = (\neg p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee \neg p_3)$$

Solution: Triangle entre $p11, p21, \neg p31$, idem $\neg p12, p22, \neg p23$, et une arête entre $p11$ et $p12$.

Question #6

Donner l'ordre de grandeur de la complexité de la construction précédente en fonction de la taille n de l'instance 3-SAT à transformer.

Solution: Il s'agit de scanner la conjonction de clauses fournie en entrée, clause par clause, et pour chaque clause créer un triangle : on obtient un graphe de n sommets. Puis pour chaque sommet, scanner l'ensemble des sommets à la recherche d'étiquettes conjugées, et si c'est le cas de rajouter une arête. Donc $O(n^2)$.

Question #7

Soit F une instance de 3-SAT qui est une conjonction de c clauses et G le graphe résultat de la construction précédente à partir de F.

Montrer que, si F est satisfaisable, alors G contient un ensemble stable de taille c. *On considérera le sous graphe formé d'un littéral bien choisi par clause, et on montrera le résultat sur l'exemple*

Solution: Si F est satisfaisable, il existe une interprétation I telle que, pour chacune des c clauses, au moins un des littéraux est évalué à Vrai. Choisissons un de ces littéraux par clause et considérons l'ensemble des o=c sommets du graphe correspondants. Ils forment un stable : aucun de ces sommets ne fait partie d'un même triangle par construction ; il ne peut pas y avoir d'arête supplémentaire entre eux car ces arêtes ne relient que des littéraux conjugués et il ne peut y avoir deux littéraux conjugués évalués à Vrai dans une interprétation donnée.

Question #8

Montrer que, si G contient un ensemble stable de taille c, alors F est satisfaisable.

Solution: Soit S un ensemble stable de taille $\geq c$ de sommets de G. Au plus un sommet par triangle est dans S.

Donc S est de taille exactement c et contient exactement un sommet de chaque triangle. Soit I l'interprétation qui rend Vrai chacun des littéraux correspondants aux étiquettes des sommets de S : I rend Vrai F.

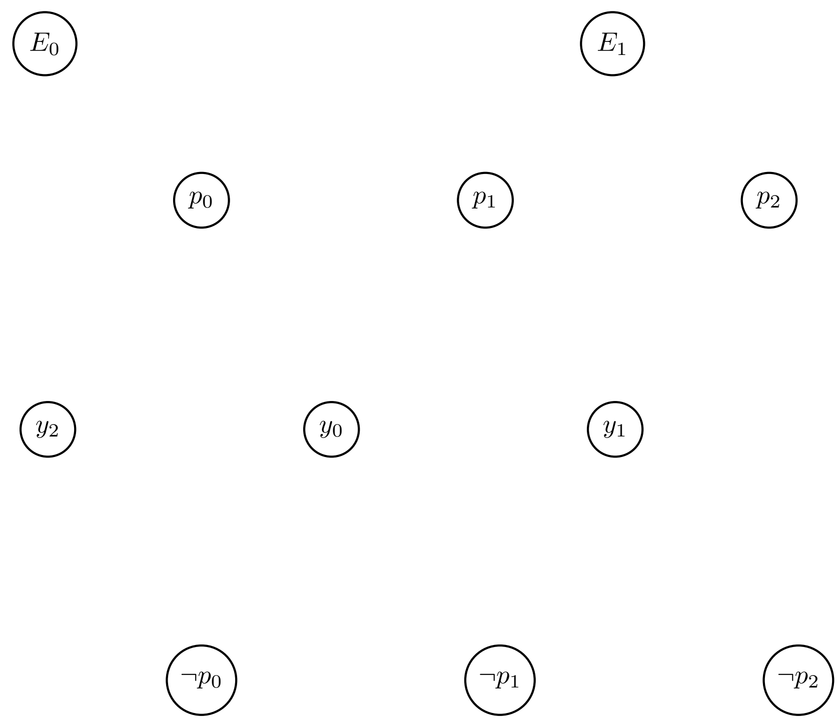
Question #9

On rappelle que le problème 3-SAT est NP-complet. En s'appuyant sur certaines des questions précédentes que vous préciserez, montrer que le problème STABLE est NP-complet.

Solution: Les questions précédentes montrent que la transformation polynomiale qui associe à chaque instance F de 3-SAT de c clauses une instance (G, c) est telle que F est satisfaisable ssi G contient un ensemble stabe de taille $\geq c$. Il s'agit donc d'une réduction de 3-SAT vers STABLE, et donc STABLE est NP-dur. Il est dans NP (cf question 3). Donc STABLE est NP-complet.

A Feuille à rendre complétée, dans votre copie

Problème 1, question 8 Le graphe G_{phi1} à compléter :



La matrice à remplir, après avoir mis en légende des lignes et les colonnes les noms des sommets du graphe (p_0, ...) :

$$\begin{pmatrix}
\begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots\\\cdots&\cdots\\\cdots&\cdots\end{pmatrix}\\
\begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots\\\cdots&\cdots\\\cdots&\cdots\end{pmatrix}\\
\begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots\\\cdots&\cdots\\\cdots&\cdots\end{pmatrix}\\
\begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots&\cdots\\\cdots&\cdots&\cdots\end{pmatrix} & \begin{pmatrix}\cdots&\cdots\\\cdots&\cdots\end{pmatrix}
\end{pmatrix}$$