MINISTÈRE
DE L'ÉDUCATION
NATIONALE

Liberté
Égalité
Fraternité

EBE NSI 1

SESSION 2025

CAPES ET CAFEP/CAPES

Concours externe - Troisième concours

Section

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Épreuve écrite disciplinaire

Le sujet est constitué d'un ou plusieurs problèmes. L'épreuve consiste en leur analyse et leur résolution. Cette épreuve évalue la maîtrise des savoirs académiques.
Elle sollicite également les capacités de raisonnement et d'argumentation du candidat.

Durée : 5 heures

L'usage de tout ouvrage de référence, de tout dictionnaire et de tout matériel électronique (y compris la calculatrice) est rigoureusement interdit.

Il appartient au candidat de vérifier qu'il a reçu un sujet complet et correspondant à l'épreuve à laquelle il se présente.

Si vous repérez ce qui vous semble être une erreur d'énoncé, vous devez le signaler très lisiblement sur votre copie, en proposer la correction et poursuivre l'épreuve en conséquence. De même, si cela vous conduit à formuler une ou plusieurs hypothèses, vous devez la (ou les) mentionner explicitement.

NB : Conformément au principe d'anonymat, votre copie ne doit comporter aucun signe distinctif, tel que nom, signature, origine, etc. Si le travail qui vous est demandé consiste notamment en la rédaction d'un projet ou d'une note, vous devrez impérativement vous abstenir de la signer ou de l'identifier.
Le fait de rendre une copie blanche est éliminatoire.

A

Tournez la page S.V.P.

INFORMATION AUX CANDIDATS

Vous trouverez ci-après les codes nécessaires vous permettant de compléter les rubriques figurant en en-tête de votre copie. Ces codes doivent être reportés sur chacune des copies que vous remettrez.

CAPES EXTERNE
NUMÉRIQUE ET SCIENCES INFORMATIQUES

► Concours externe du CAPES de l'enseignement public :

Concours	Section/option	Epreuve	Matière
EBE	6200E	101	9311

► Troisième concours du CAPES de l'enseignement public :

Concours	Section/option	Epreuve	Matière
EBV	6200E	101	9311

► Concours externe du CAFEP/CAPES de l'enseignement privé :

Concours	Section/option	Epreuve	Matière
EBF	6200E	101	9311

IMPRIMERIE NATIONALE – 25 0208 – D'après documents fournis

CAPES externe et CAFEP-CAPES
Section Numérique et Sciences Informatiques
Épreuve Disciplinaire – Session 2025

Autour du problème de la couverture par ensembles

Présentation du sujet. Ce sujet traite du problème algorithmique de la couverture par ensembles. Il se décompose en 7 parties. La partie 1 présente un problème concret permettant de fixer les idées sur ce qu'est le problème de la couverture par ensembles. Dans la partie 2 on modélise le problème, les instances et leurs solutions en PYTHON. La partie 3 se penche sur deux variantes d'un algorithme par brute force pour ce problème, tandis que la partie 4 propose un algorithme glouton. La partie 5 introduit la notion de solution partielle et ne contient pas de question. La partie 6 présente la notion de graphe associé à une solution partielle. La partie 7 présente le paradigme de séparation et évaluation et utilise les résultats précédemment obtenus afin de fournir une autre solution algorithmique au problème.
Il est tout à fait possible de sauter des questions ou des parties du sujet, il est toutefois nécessaire de lire toutes les questions et définitions pour comprendre les questions suivantes.
Le sujet est complété de deux annexes (page 17). L'annexe A précise les quelques notations mathématiques utilisées dans le sujet. L'annexe B fournit une description succincte de la libraire PYTHON `typing` utilisée tout au long du sujet pour les annotations de type.

Travail attendu. Si une fonction ou un résultat est introduit dans l'énoncé, vous pouvez l'utiliser dans les questions suivantes, y compris si vous n'avez pas proposé de définition de cette fonction, de justification de ce résultat. Dans la partie 1, les questions de programmation doivent être traitées en SQL. Dans les parties suivantes les questions de programmation doivent être traitées en PYTHON, et chacune d'elle est accompagnée de la signature de la fonction à implémenter (au besoin on se référera à l'annexe B).
On rappelle que les réponses aux questions de programmation doivent être compréhensibles. Une réponse même correcte risque de ne rapporter aucun point si le style ne permet pas une compréhension aisée de la solution proposée. Pour rendre vos programmes compréhensibles vous pouvez utiliser des noms de variables pertinents, des fonctions auxiliaires et des commentaires. Chaque fonction auxiliaire (comprendre, non explicitement demandée par l'énoncé) doit être accompagnée d'une description rapide de son comportement et du sens attaché à ses arguments. Si la correction de votre programme ou de votre algorithme repose sur une idée non triviale, il faut donner cette idée en français avant le code.
Lorsqu'il vous est demandé de fournir la complexité algorithmique pire cas d'une fonction, vous devez fournir le résultat sous la forme d'un $\mathcal{O}$ et veiller à ne pas donner une majoration trop grossière (par exemple, on ne dira pas $\mathcal{O}(n^2)$ lorsque $\mathcal{O}(n)$ aurait suffit).

1. Un exemple introductif

On s'intéresse dans cette section à un problème concret d'organisation de révisions. Un groupe de personnes préparant le CAPES de NSI a recueilli des énoncés d'épreuves d'informatique de différents concours et cherche à sélectionner un ensemble d'énoncés qui couvre tout le programme du CAPES avec un minimum d'énoncés. Pour cela les différents énoncés ont été annotés à l'aide de mots-clés (par exemple *binaire*, *glouton*, *sql*, …), et les différents mots-clés ont été raccrochés aux thèmes du programme du CAPES (par exemple *Algorithmique*, *Types construits*, …). On suppose que chaque mot-clé n'est raccroché qu'à un seul thème du programme, et l'objectif est de couvrir chacun des thèmes au programme. Les données issues de ce travail documentaire ont été rassemblées dans quatre tables d'une base de données.

- La table `ProgrammeCapes` qui liste les thèmes au programme du CAPES de NSI.
  Cette table contient un seul champ `Theme`.
- La table `Enonce` qui liste les énoncés disponibles.
  Cette table contient cinq champs : `Id`, `Concours`, `Discipline`, `Annee` et `Epreuve`.
- La table `Balise` qui associe aux énoncés des mots-clés.
  Cette table a deux champs `Id` et `MotCle`. On suppose que tous les identifiants `Id` apparaissant dans cette table sont des identifiants d'énoncés de la table `Enonce`.
- La table `Ancrage` qui associe à un mot-clé le thème du programme auquel il se rapporte.
  Cette table a deux champs `MotCle` et `Theme`. On suppose que tous les thèmes apparaissant dans cette table sont des thèmes au programme figurant dans la table `ProgrammeCapes`.

On donne ci-après quelques enregistrements extraits de ces tables.

Extrait de la table `ProgrammeCapes`

Theme
Algorithmique
Types et valeurs de base
Types construits

Extrait de la table `Enonce`

Id	Concours	Discipline	Annee	Epreuve
1	CAPES	NSI	2020	écrit 1
4	CAPES	NSI	2021	écrit 2
9	CAPES	NSI	2025	écrit 1
10	Agrégation	Informatique	2023	composition

Extrait de la table `Balise`

Id	MotCle
9	SQL
9	algorithmes gloutons
9	graphe
9	dictionnaire

Extrait de la table `Ancrage`

MotCle	Theme
dictionnaire	Types construits
SQL	Bases de données
fichier csv	Traitement de données en table
arbres	Structures de données
terminaison	Algorithmique

Q. 1 Quelle requête SQL permet d'obtenir la liste des thèmes au programme du CAPES ?

Q. 2 Quelle requête donne le nombre d'années d'épreuve de CAPES présentes dans la base ? Le résultat de la requête doit être réduit à ce nombre.

Q. 3 Quelle requête SQL permet d'obtenir la liste, triée par ordre alphabétique, des mots-clés associés au thème *Algorithmique* qui apparaissent dans au moins un énoncé de la base ?

Q. 4 Quelle requête SQL permet d'obtenir la liste des thèmes couverts par les énoncés de la base de données ?

Q. 5 Quelle requête SQL permet de lister les thèmes au programme qui ne sont pas couverts par au moins un énoncé de la base de données ?

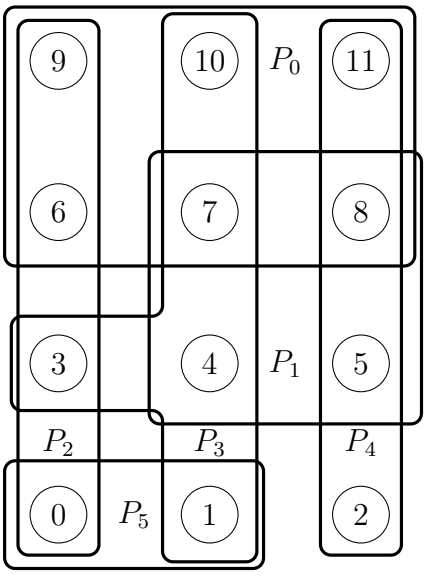
Grâce à la question précédente, on peut vérifier que la base de données contient assez d'énoncés pour couvrir tout le programme. On travaille sous cette hypothèse dans la question suivante.

Q. 6 Quelle requête SQL permet d'obtenir la liste qui associe aux identifiants d'énoncés du CAPES les thèmes qu'ils couvrent ? On veillera à éviter les doublons dans la liste résultat, par exemple si un énoncé couvre le thème *Algorithmique* parce qu'il est à la fois annoté par le mot-clé *glouton* et le mot-clé *programmation dynamique*, l'association de ce sujet au thème *Algorithmique* ne doit apparaître qu'une seule fois dans le résultat.

2. Présentation du problème

Le problème concret présenté à la section précédente consiste, connaissant un ensemble de sujets et les thèmes qu'ils abordent, à trouver un sous-ensemble de ces sujets qui couvre l'ensemble du programme du CAPES de NSI.

Un exemple. Dans cet exemple, on suppose qu'il y a 12 thèmes à couvrir, numérotés de 0 à 11. On notera donc X_e l'ensemble des thèmes à couvrir, à savoir : $X_e = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$. On dispose de 6 sujets, numérotés de 0 à 5. Pour chaque sujet on connaît les thèmes qu'il couvre, par exemple le sujet 2 couvre les thèmes $0, 3, 6, 9$. Ainsi chaque sujet donne un sous-ensemble des thèmes à couvrir, pour chaque $i \in [\![0, 5]\!]$, on note P_i le sous-ensemble de thèmes que couvre le sujet numéro i. En particulier $P_2 = \{0, 3, 6, 9\}$. L'ensemble des 6 sujets à disposition est donc un ensemble de sous-ensembles de thèmes, on le notera ici $\mathcal{F}_e$. Ainsi $\mathcal{F}_e = \{P_0, P_1, P_2, P_3, P_4, P_5\}$. La définition complète de cet exemple est fournie par la Figure 1, à la fois de manière schématique et de manière mathématique.



$$X_e = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$$
$$\mathcal{F}_e = \{P_0, P_1, P_2, P_3, P_4, P_5\}$$
$$P_0 = \{6, 7, 8, 9, 10, 11\}$$
$$P_1 = \{4, 5, 7, 8\}$$
$$P_2 = \{0, 3, 6, 9\}$$
$$P_3 = \{1, 3, 4, 7, 10\}$$
$$P_4 = \{2, 5, 8, 11\}$$
$$P_5 = \{0, 1\}$$

FIGURE 1 – L'exemple $(X_e, \mathcal{F}_e)$

La sélection des sujets P_0 et P_1 ne permet pas de couvrir tous les thèmes au programme : les thèmes 0, 1, 2 et 3 ne sont pas couverts. Au contraire, la sélection des sujets P_0, P_2, P_3 et P_4 couvre bien tous les thèmes au programme. Ainsi il est possible de couvrir tous les thèmes au programme avec 4 sujets, ce n'est pas cependant pas le minimum. En effet la sélection de sujets P_2, P_3 et P_4 couvre aussi le programme, en utilisant seulement 3 sujets.

Abstraction. Ce problème concret relève d'un problème d'optimisation connu sous le nom anglais SetCover que l'on traduit en CouvEns, pour Couverture par Ensemble. Étant donné un ensemble fini X ♣ et un ensemble $\mathcal{F}$ de sous-ensembles de X ♡, on appelle *couverture* un sous-ensemble $\mathcal{C}$ de $\mathcal{F}$ tel que $X = \bigcup_{P\in\mathcal{C}} P$ ♠. Le problème CouvEns consiste à trouver une couverture la plus petite possible, à savoir, de plus petit cardinal ♢. Ainsi une instance du problème CouvEns est la donnée d'un couple $(X, \mathcal{F})$. La Figure 1 définissait l'instance $(X_e, \mathcal{F}_e)$ qui sera utilisée en exemple tout au long du sujet.

Couvrable. Pour un instance $(X, \mathcal{F})$, il se peut qu'il n'existe aucune couverture. Toutefois, dès lors que chaque élément de X apparaît dans au moins un ensemble de $\mathcal{F}$, il existe une couverture : par exemple celle sélectionnant tous les sous-ensembles possibles, à savoir $\mathcal{F}$. Cette contrainte s'exprime par le fait que $\bigcup_{P\in\mathcal{F}} P = X$, on dira alors que X est *couvrable* par $\mathcal{F}$.

Formalisation. On formalise les trois problèmes qui nous intéresseront dans ce sujet. Le premier est un problème de recherche de solution optimale, le deuxième un problème de calcul de la valeur optimale, et le dernier un problème de décision.

CouvEns :
- **Entrée :** Un ensemble fini X, un ensemble $\mathcal{F}$ de sous-ensembles de X tel que X est couvrable par $\mathcal{F}$.
- **Sortie :** Une couverture $\mathcal{C}$ de X par $\mathcal{F}$, qui soit de cardinal minimal.

CouvEnsTaille :
- **Entrée :** Un ensemble fini X, un ensemble $\mathcal{F}$ de sous-ensembles de X tel que X est couvrable par $\mathcal{F}$.
- **Sortie :** Le cardinal minimal d'une couverture de X par $\mathcal{F}$.

CouvEnsSeuil :
- **Entrée :** Un ensemble fini X, un ensemble $\mathcal{F}$ de sous-ensembles de X tel que X est couvrable par $\mathcal{F}$, un entier $K \in \mathbb{N}$.
- **Sortie :** Existe-t-il $\mathcal{C}$ une couverture de X par $\mathcal{F}$, de cardinal $\leqslant K$?

On donne ci-dessous des exemples de relation entrée/sortie pour ces trois problèmes.
- Pour le problème CouvEns sur l'instance $(X_e, \mathcal{F}_e)$, une solution peut être l'ensemble $\{P_2, P_3, P_4\}$ qui est bien une couverture de cardinal minimal. C'est en fait la seule solution possible.
- Pour le problème CouvEnsTaille sur l'instance $(X_e, \mathcal{F}_e)$, l'unique solution est 3.
- Pour le problème CouvEnsSeuil sur l'instance $(X_e, \mathcal{F}_e, 4)$, l'unique solution est vrai (`True`) tandis que sur l'instance $(X_e, \mathcal{F}_e, 2)$, l'unique solution est faux (`False`).

♣. les thèmes dans notre exemple
♡. l'ensemble des sujets dans notre exemple, chaque sujet étant un ensemble de thèmes
♠. une sélection de sujets couvrant tous les thèmes dans notre exemple
♢. dans notre exemple, on cherche une sélection, la plus petite possible, c'est-à-dire avec le moins de sujets possible

Q. 7 On considère l'instance $X = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$, et $\mathcal{F} = \{Q_0, Q_1, Q_2, Q_3, Q_4\}$ où

- $Q_0 = \{0, 1, 3, 4, 6, 7\}$,
- $Q_1 = \{1, 2, 4, 5, 7\}$,
- $Q_2 = \{7, 8\}$,
- $Q_3 = \{6, 8\}$,
- $Q_4 = \{0, 1, 3, 4, 5\}$.

Donner des sorties pour les problèmes CouvEns et CouvEnsTaille sur l'entrée $(X, \mathcal{F})$ ainsi que pour CouvEnsSeuil sur l'entrée $(X, \mathcal{F}, 2)$.

2.1. Manipulation des problèmes

Q. 8 On suppose fourni, pour cette question seulement, un algorithme couv_ens_taille résolvant CouvEnsTaille, c'est-à-dire, prenant en arguments un ensemble X et un ensemble $\mathcal{F}$ de parties de X et renvoyant le cardinal d'une couverture de X de cardinal minimal. Proposer le pseudo-code d'un algorithme utilisant couv_ens_taille et résolvant CouvEnsSeuil. Donner le nombre d'appels à couv_ens_taille effectués en fonction des valeurs d'entrées X, $\mathcal{F}$ et K.

Q. 9 On suppose fourni, pour cette question seulement, un algorithme couv_ens_seuil résolvant CouvEnsSeuil, c'est-à-dire prenant en arguments un ensemble X, un ensemble $\mathcal{F}$ de parties de X et un entier K et renvoyant un booléen indiquant s'il existe une couverture de X par $\mathcal{F}$ de cardinal $\leqslant K$.

a) Proposer le pseudo-code d'un algorithme utilisant couv_ens_seuil, résolvant le problème CouvEnsTaille. Cet algorithme devra effectuer un nombre le plus faible possible d'appels à couv_ens_seuil.

b) Donner, sans justifier, en fonction de X et $\mathcal{F}$, un ordre de grandeur du nombre d'appels à couv_ens_seuil effectués par cet algorithme.

c) Proposer un invariant de boucle, qui justifierait la correction de cet algorithme.

2.2. Représentation en machine

Restriction. Dans tout le reste du sujet on se limite au cas où X est un intervalle d'entiers de la forme $\{0, 1, \ldots, n-1\} = [\![0, n-1]\!]$ pour un certain entier $n \in \mathbb{N}$.

Représentation en machine. On présente ici les différentes manières dont les objets manipulés dans ce sujet seront encodés en Python.

- Étant donné que l'on se restreint au cas où $X = [\![0, n-1]\!]$ pour un certain n, l'entrée X sera représentée par un entier `n`.
- Une partie de X sera représentée par la liste de ses éléments. Bien que dans tous les exemples les éléments d'une partie $P \in \mathcal{F}$ soient listés dans l'ordre croissant, ceci n'est pas requis. On n'utilisera pas cette hypothèse par la suite. L'ensemble $\mathcal{F}$ des parties de X autorisées pour la couverture sera alors représenté par la liste de ses éléments, de type `List[List[int]]`. Aussi dans la suite, on suppose défini le type ci-dessous.

```
Parties = List[List[int]]
```

Ainsi les données X_e et $\mathcal{F}_e$ de l'exemple seront représentées en Python par les valeurs `n_ex` et `f_ex` du code ci-dessous.

```
p0_ex = [6, 7, 8, 9, 10, 11]
p1_ex = [4, 5, 7, 8]
p2_ex = [0, 3, 6, 9]
p3_ex = [1, 3, 4, 7, 10]
p4_ex = [2, 5, 8, 11]
```

```
p5_ex = [0, 1]
f_ex  = [p0_ex, p1_ex, p2_ex, p3_ex, p4_ex, p5_ex]
n_ex  = 12
```

Dans la suite, lorsqu'une fonction prendra en arguments `n` et `f` représentant une instance $(X, \mathcal{F})$, on supposera que `n` est positif et que les listes de `f` sont bien à valeurs dans $[\![0, n-1]\!]$.

- Pour manipuler les couvertures, on s'autorisera deux encodages :
  - Un sous-ensemble $\mathcal{C}$ de `f` = `[`P_0`,` P_1`,` $\ldots$`,` P_{m-1}`]` pourra être représenté au moyen d'une liste des indices dans la liste $\mathcal{F}$ des parties présentes dans $\mathcal{C}$. Un tel objet est alors de type `List[int]`. On dira alors que $\mathcal{C}$ est représenté par *liste d'indices*. Par exemple le sous-ensemble $\mathcal{C}_1 = \{P_0, P_3, P_4, P_2\}$ de l'exemple pourra être représenté par la liste `[0, 4, 3, 2]` ou encore `[2, 3, 4, 0]`.
  - Un sous-ensemble $\mathcal{C}$ de `f` = `[`P_0`,` P_1`,` $\ldots$`,` P_{m-1}`]` pourra *aussi* être représenté par un tableau de booléens `t`, de taille $m = |\mathcal{F}|$, indiquant dans chaque case d'indice i si P_i est présent ou non dans $\mathcal{C}$. Un tel objet est alors de type `List[bool]`. On dira alors que $\mathcal{C}$ est représenté par sa *fonction indicatrice*. Par exemple le sous-ensemble $\mathcal{C}_1 = \{P_0, P_3, P_4, P_2\}$ de l'exemple sera représenté par le tableau `[True, False, True, True, True, False]`.

Q. 10 Définir une fonction `card1` prenant en argument un sous-ensemble $\mathcal{C}$, représenté par liste d'indices, et renvoyant son cardinal.

```
card1(c: List[int]) -> int
```

Q. 11 Définir une fonction `card2` prenant en argument un sous-ensemble $\mathcal{C}$, représenté par fonction indicatrice, et renvoyant son cardinal.

```
card2(c: List[bool]) -> int
```

2.3. Vérification des solutions

Q. 12 Définir une fonction `verification1` prenant en arguments `n` et `f` représentant une instance $(X, \mathcal{F})$ et `c` représentant un sous-ensemble $\mathcal{C}$ de $\mathcal{F}$ par liste d'indices, et renvoyant si $\mathcal{C}$ est une couverture de X par $\mathcal{F}$.

```
verification1(n: int, f: Parties, c: List[int]) -> bool
```

Q. 13 En utilisant les notations $X = [\![0, n-1]\!]$, $\mathcal{F} = \{P_0, P_1, ...P_{m-1}\}$, pour tout $i \in [\![0, m-1]\!]$, $n_i = |P_i|$, $l = \sum_{i=0}^{m-1} n_i$, et finalement $q = |\mathcal{C}|$, donner la complexité algorithmique pire cas de la fonction `verification1` proposée à la question précédente (**Q. 12**). On attend une justification succincte.

Q. 14 Définir une fonction `verification2` prenant en arguments `n` et `f` représentant une instance $(X, \mathcal{F})$ et `c` représentant un sous-ensemble $\mathcal{C}$ de $\mathcal{F}$ par sa fonction indicatrice, et renvoyant si $\mathcal{C}$ est une couverture de X par $\mathcal{F}$.

```
verification2(n: int, f: Parties, c: List[bool]) -> bool
```

Q. 15 En utilisant les notations de la **Q. 13** donner la complexité algorithmique pire cas de la fonction `verification2` (**Q. 14**). On attend une justification succincte.

Q. 16 Définir une fonction `est_couvrable` prenant en arguments `n` et `f` représentant une instance $(X, \mathcal{F})$ et testant si X est couvrable par $\mathcal{F}$.

```
est_couvrable(n: int, f: Parties) -> bool
```

Important : dans toute la suite on s'intéresse à des instances $(X, \mathcal{F})$ où X est couvrable par $\mathcal{F}$.

2.4. Obtenir des instances en Python

Dans cette section, on envisage l'import d'une instance du problème concret de la section 1 sous la forme d'un couple `(n, f)` où `n` est de type `int` et `f` de type `Parties` comme expliqué plus haut.

On suppose que la table de données obtenue à la question 6 a été exportée au format csv dans un fichier nommé `assoc_id_theme.csv`, dont les premières lignes sont reproduites ci-contre. Ce fichier texte contient un enregistrement par ligne, les valeurs sont séparées par des virgules.

assoc_id_theme.csv

```
9,Algorithmique
3,Algorithmique
5,Algorithmique
9,Bases de données
```

Un tel formatage permet d'extraire les données en Python grâce au module `csv`, et en particulier grâce à la fonction `csv.reader`. On donne ici un exemple d'utilisation de cette fonction très proche de celui fourni par la documentation du module `csv`. Le fichier `oeufs.csv` contient le texte ci-dessous.

oeufs.csv

```
brouillés, au plat, au plat, miroir
bénédicte, au plat, coque, parfait, mollet
```

Le code ci-dessous importe dans l'objet `filereader` les données de la table enregistrée dans le fichier `oeufs.csv`, ainsi itérer sur `filereader` revient à itérer sur les lignes du fichier, soit dans notre cas sur les enregistrements de la table.

```
>>> import csv
>>> with open('oeufs.csv', newline='') as csvfile:
...     filereader = csv.reader(csvfile, delimiter=',')
...     for row in filereader:
...             for i in range(len(row)):
...                     print(row[i], end=" / ")
...             print()
brouillés /  au plat /  au plat /  miroir /
bénédicte /  au plat /  coque /  parfait /  mollet /
```

Q. 17 À l'aide de l'exemple de code fourni ci-dessus, proposer le code d'un *programme* Python qui extrait du fichier `assoc_id_theme.csv` un dictionnaire nommé `dico_id` qui associe à chaque identifiant d'énoncé (de type `int`) la liste des thèmes (de type `List[str]`) que couvre cet énoncé.

De manière similaire on pourrait extraire du fichier `assoc_id_theme.csv` un dictionnaire nommé `num_theme` de type `Dict[str, int]` qui associe à chaque thème un numéro unique entre 0 et $n-1$ où n est le nombre de thèmes en jeu.

Q. 18 Définir une fonction `cree_instance` prenant en arguments `dico` un dictionnaire qui associe aux identifiants d'énoncés la liste des thèmes qu'ils couvrent et `num` un dictionnaire numérotant les thèmes apparaissant dans `dico` et renvoyant le couple `(n, f)` où `n` est le nombre de thèmes à couvrir et où `f` est une liste de listes de thèmes couverts par un même énoncé. Dans `f`, les thèmes doivent être représentés par leur numéro, soit un entier entre 0 et $n-1$.

```
cree_instance (dico: Dict[int, List[str]], num: Dict[str, int]) ->
↪  Tuple[int, Parties]
```

3. Un algorithme par brute force

Dans cette section on se concentre sur la mise en place d'un algorithme de résolution du problème CouvEns par brute force : étant donné une instance $(X, \mathcal{F})$, on parcourt l'espace de tous les choix de $\mathcal{C} \subseteq \mathcal{F}$, pour trouver une couverture de cardinal minimal.

3.1. Version naïve

Pour cette première approche on choisit de représenter les couvertures $\mathcal{C} \subseteq \mathcal{F}$ par leur fonction indicatrice.

Q. 19 Définir une fonction `suivant` prenant en argument un tableau de m booléens représentant la décomposition en base 2 d'un entier e de $[\![0, 2^m - 1]\!]$ (les bits de poids faibles sont à droite), et *modifiant* le tableau pour :

- qu'il contienne la décomposition en base 2 de l'entier $e + 1$, dans le cas où $e < 2^m - 1$;
- qu'il contienne uniquement des `False` sinon.

Dans le premier cas, cette fonction renvoie `True`, dans le second elle renvoie `False`.

```
suivant(cand: List[bool]) -> bool
```

Q. 20 Définir une fonction `couv_ens_naif` prenant en arguments `n` et `f` représentant une instance $(X, \mathcal{F})$ et renvoyant un couple `(q, c)` tel que `c` est la représentation par fonction indicatrice d'une couverture de cardinal minimal de X par $\mathcal{F}$ et `q` est le cardinal de cette couverture.

```
couv_ens_naif(n: int, f: Parties) -> Tuple[int, List[bool]]
```

3.2. Énumération par cardinal croissant

Dans le problème CouvEns, on cherche un sous-ensemble de cardinal minimal, aussi on s'intéresse dans cette partie à énumérer les sous-ensembles d'un ensemble $\mathcal{F}$ fixé par cardinaux croissants, afin d'arrêter la recherche dès que l'on trouve une couverture. Pour cette seconde approche on choisit de représenter les couvertures $\mathcal{C} \subseteq \mathcal{F}$ par leur liste d'indices.
On rappelle que le nombre de sous-ensembles de cardinal k d'un ensemble de cardinal m est le coefficient binomial $\binom{m}{k}$. Les coefficients binomiaux vérifient les relations de récurrence ci-dessous.

$$\forall m \in \mathbb{N}, \forall k \in \mathbb{N}, \binom{m}{k} = \begin{cases} 1 & \text{si } k = 0 \\ \binom{m-1}{k-1} + \binom{m-1}{k} & \text{si } k \in [\![1, m]\!] \\ 0 & \text{sinon} \end{cases}$$

Afin d'énumérer, par cardinaux croissants, les sous-ensembles d'un ensemble de cardinal m, nous allons nous reposer sur la connaissance des coefficients $\binom{p}{k}$ pour tout $p \in [\![0, m]\!]$ et tout $k \in [\![0, m]\!]$. L'objectif de la question suivante est de pré-calculer efficacement ces coefficients.

Q. 21 Définir une fonction `binomiaux` prenant en argument un entier `m` et renvoyant une matrice `b`, indicée par les entiers $[\![0, m]\!] \times [\![0, m]\!]$, telle que pour tout $p \in [\![0, m]\!]$ et tout $k \in [\![0, m]\!]$, `b[p][k]` contient le coefficient $\binom{p}{k}$. On précisera, sans la justifier, la complexité algorithmique pire cas de la fonction implémentée.

```
binomiaux(m: int) -> List[List[int]]
```

Pour former un sous-ensemble de cardinal $k \neq 0$ de l'ensemble $\{0, 1, ..., m-1\}$ on peut :

$a)$ former un ensemble à $k-1$ éléments de $\{0, 1, ..., m-2\}$ et y ajouter l'élément $m-1$;

$b)$ ou bien former un ensemble à k éléments de $\{0, 1, ..., m-2\}$.

On remarque qu'on peut ainsi former $\binom{m-1}{k-1}$ ensembles de type $a)$ et $\binom{m-1}{k}$ ensembles de type $b)$.

Le but de la question suivante est d'utiliser les remarques précédentes pour construire une fonction `ensemble_num_i` qui, pour trois entiers i, k et m donnés, retourne le i-ème sous-ensemble à k éléments de l'ensemble $[\![0, m-1]\!]$, où i-ème s'entend pour une numérotation bien choisie. La Figure 2 donne la numérotation des $10 = \binom{4}{3}$ sous-ensembles à 3 éléments de $[\![0, 4]\!]$. Chaque flèche de l'arbre correspond à une décision : à gauche on choisit de former un sous-ensemble de type $a)$, à droite on choisit de former un sous-ensemble de type $b)$. Chaque chemin de la racine à une feuille correspond à une suite de décisions qui forment un ensemble de taille 3, indiqué au dessous. Les ensembles formés sont alors numérotés de gauche à droite.

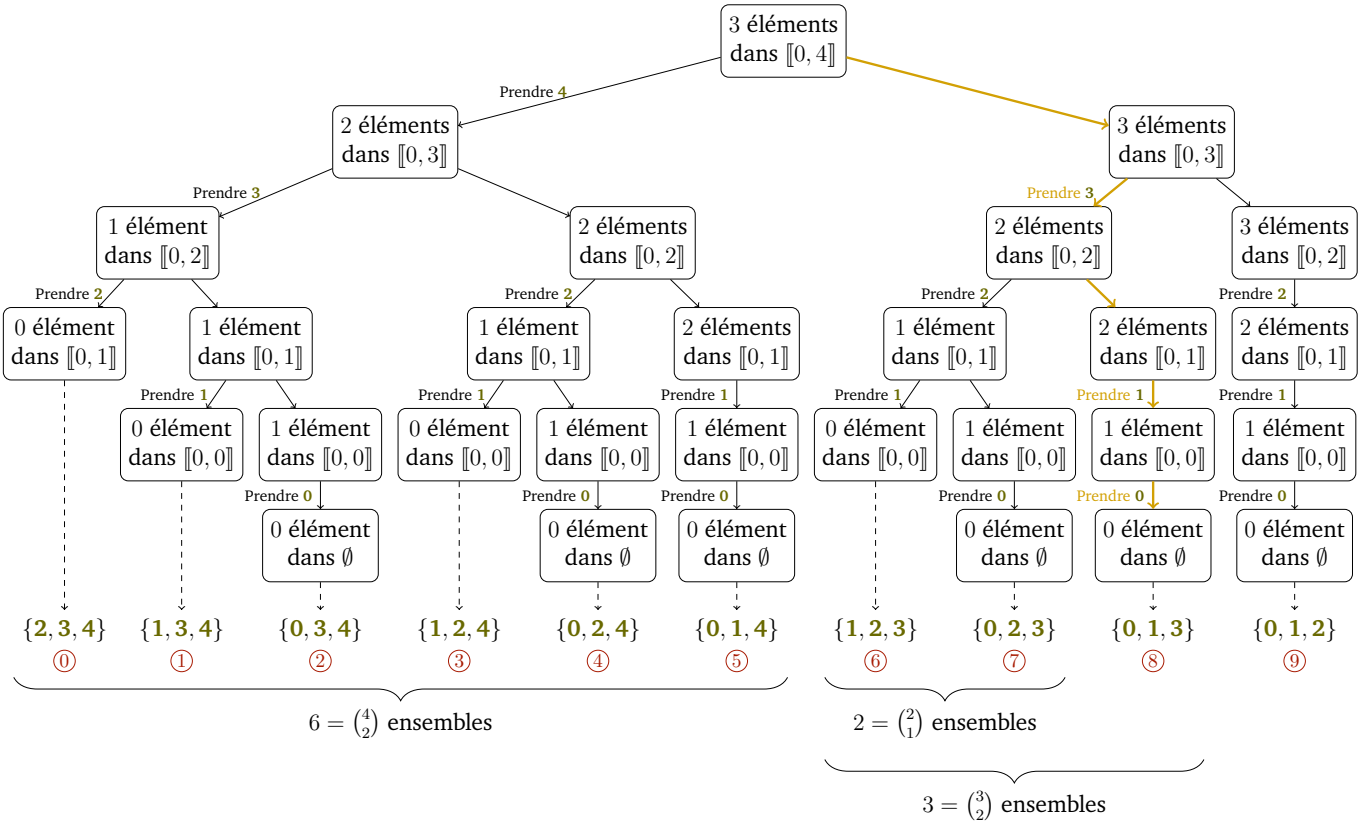


FIGURE 2 – Numérotation des sous-ensembles à 3 éléments de $[\![0, 4]\!]$.

Exemple. On cherche à construire l'ensemble de numéro ⑧ parmi les ensembles à 3 éléments de $[\![0, 4]\!]$.

- Comme l'indique la Figure 2, il y a $\binom{4}{2} = 6$ ensembles à 3 éléments de $[\![0, 4]\!]$ qui contiennent le nombre **4**. Autrement dit, en choisissant de construire un ensemble de type $a)$, on obtiendrait un ensemble de numéro entre ⓪ et ⑤. Aussi choisit-on de construire un ensemble de type $b)$, c'est-à-dire ne contenant pas **4**. On cherche donc l'ensemble de numéro $8 - 6 = 2$ parmi les ensembles à 3 éléments de $[\![0, 3]\!]$.
- Comme l'indique la Figure 2, il y a $\binom{3}{2} = 3$ ensembles à 3 éléments de $[\![0, 3]\!]$ qui contiennent le nombre **3**. Autrement dit, en choisissant de construire un ensemble de type $a)$, on obtiendrait un ensemble de numéro entre 0 et 2. Aussi choisit-on ici de construire un ensemble de type $a)$, c'est-à-dire contenant **3**. On cherche donc l'ensemble de numéro 2 parmi les ensembles à 2 éléments de $[\![0, 2]\!]$, sachant qu'on lui ajoutera **3**.

- Comme l'indique la Figure 2, il y a $\binom{2}{1} = 2$ ensembles à 2 éléments de $[\![0, 2]\!]$ qui contiennent le nombre **2**. Autrement dit, en choisissant de construire un ensemble de type a), on obtiendrait un ensemble de numéro entre 0 et 1. Aussi choisit-on de construire un ensemble de type b), c'est-à-dire ne contenant pas **2**. On cherche donc l'ensemble de numéro $2 - 2 = 0$ parmi les ensembles à 2 éléments de $[\![0, 1]\!]$.
- Il y a $\binom{1}{1} = 1$ ensemble à 2 éléments de $[\![0, 1]\!]$ qui contient le nombre **1**. Aussi choisit-on de construire un ensemble de type a), c'est-à-dire contenant **1**. On cherche donc l'ensemble de numéro 0 parmi les ensembles à 1 éléments de $[\![0, 0]\!]$, sachant qu'on lui ajoutera **1**.
- Il y a $\binom{1}{1} = 1$ ensemble à 1 élément de $[\![0, 0]\!]$ qui contient le nombre **0**. Aussi choisit-on de construire un ensemble de type a), c'est-à-dire contenant **0**. Il reste donc à trouver l'ensemble de numéro 0 parmi les ensembles à 0 élément de $[\![0, 0]\!]$, sachant qu'on lui ajoutera **0**. Un tel ensemble est l'ensemble vide.

Finalement on ajoute **3**, **1** et **0** à l'ensemble vide, construisant ainsi l'ensemble $\{\mathbf{0}, \mathbf{1}, \mathbf{3}\}$.

Q. 22 Définir une fonction `ensemble_num_i` prenant en arguments :
- trois entiers naturels i, k et m vérifiant $k \leqslant m$ et $i \in [\![0, \binom{m}{k} - 1]\!]$ et
- une matrice `b` suffisamment grande pour contenir les coefficients binomiaux $\binom{r}{s}$ pour tout couple (r, s) de $[\![0, m]\!]^2$.

et renvoyant l'ensemble de numéro i parmi les ensembles à k éléments de $[\![0, m-1]\!]$.

```
ensemble_num_i(i: int, k: int, m: int, b: List[List[int]]) -> List[int]
```

Q. 23 En déduire une fonction `couv_ens_binom` prenant en arguments `n` et `f` représentant une instance $(X, \mathcal{F})$ et renvoyant un couple `(q, c)` tel que `c` est la représentation par liste d'indices d'une couverture de cardinal minimal de X par $\mathcal{F}$ et `q` est le cardinal de cet ensemble.

```
couv_ens_binom(n: int, f: Parties) -> Tuple[int, List[int]]
```

4. Un algorithme glouton

Éléments couverts. Lorsque $(X, \mathcal{F})$ est une instance d'un problème de couverture, et que $\mathcal{C} \subseteq \mathcal{F}$ est un sous-ensemble de $\mathcal{F}$, on dit d'un élément $x \in X$ qu'il est *couvert par* $\mathcal{C}$ dès lors qu'il existe une partie $P \in \mathcal{C}$ telle que $x \in P$. Naturellement, $\mathcal{C}$ est une couverture de X, si et seulement si tout élément de X est couvert par $\mathcal{C}$.

Les algorithmes de résolution du problème CouvEns considérés jusqu'ici ont des complexités algorithmiques exponentielles. On se propose ici d'étudier l'algorithme glouton suivant.

- On initialise l'ensemble $\mathcal{C}$ à l'ensemble vide.
- Tant qu'il reste des éléments de X qui ne sont pas couverts par $\mathcal{C}$:
  - on choisit dans $\mathcal{F}$ l'ensemble P ayant le plus d'éléments non encore couverts par $\mathcal{C}$,
  - on ajoute P à $\mathcal{C}$.

On donne ci-dessous le pseudo-code de cet algorithme.

Algorithme 1 : Glouton

Entrée : Une instance $(X, \mathcal{F})$.
Sortie : Calcule une couverture de X par $\mathcal{F}$

```
1 C ← ∅ ;                                  // La solution en cours de construction
2 R ← X ;                                  // R pour Reste à couvrir
3 tant que R ≠ ∅ faire
4 |   Choisir P ∈ F tel que |P ∩ R| soit maximal ;
5 |   C ← {P} ∪ C ;
6 |   R ← R \ P ;
7 renvoyer C
```

4.1. Non optimalité de l'algorithme glouton

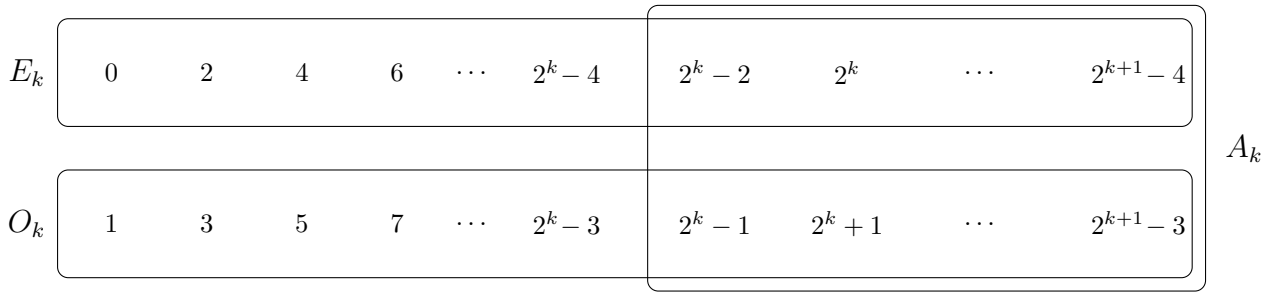
Q. 24 Donner une instance $(X, \mathcal{F})$ sur laquelle l'algorithme glouton renvoie une couverture de X qui n'est pas une solution optimale (de cardinal non minimal donc). Détailler l'exécution de l'algorithme glouton sur cette instance, justifier que la solution obtenue n'est pas optimale.

La question précédente justifie que cet algorithme glouton ne répond pas de manière exacte au problème CouvEns. Dans la suite on cherche à quantifier ce défaut d'optimalité. Plus précisément, on montre que le rapport entre la valeur de la solution fournie par l'algorithme glouton (le cardinal de l'ensemble des parties sélectionnées) et la valeur optimale n'est pas borné. Pour cela, il suffit d'exhiber, pour tout $k \in \mathbb{N}$, une instance $(X_k, \mathcal{F}_k)$ telle qu'en notant $\mathcal{C}^g$ la solution renvoyée par l'algorithme glouton et en notant $\mathcal{C}^\star$ une solution optimale, $\frac{|\mathcal{C}^g|}{|\mathcal{C}^\star|} \geqslant \frac{k}{2}$.

Étant donné un entier $k \in \mathbb{N}$, on considère :

- l'ensemble $X_k = \{0, 1, ..., 2^{k+1} - 3\}$;
- les parties $E_k = \{0, 2, ..., 2^{k+1} - 4\}$ et $O_k = \{1, 3, ..., 2^{k+1} - 3\}$ ♣ ;
- et finalement $A_k = \{2^k - 2, 2^k - 1, 2^k, ..., 2^{k+1} - 3\}$.

Ces parties sont représentées sur le schéma ci-dessous.



Q. 25 *a)* Donner les cardinaux des ensembles X_k, E_k, O_k et A_k.

b) Quel est le comportement de l'algorithme glouton sur l'instance $(X_k, \{E_k, O_k, A_k\})$?

Q. 26 *a)* Proposer un ensemble $\mathcal{F}_k$ contenant au moins E_k, O_k, A_k, tel que l'algorithme glouton renvoie, sur l'instance $(X_k, \mathcal{F}_k)$, une solution de cardinal k. Expliciter le déroulé de l'algorithme sur l'instance proposée.

b) Conclure.

♣. E pour Even : pair en anglais, et O pour Odd : impair en anglais

4.2. Correction de l'algorithme glouton

On admet dans cette section que la propriété "$R \subseteq X$" est un invariant de la boucle **Tant que** de l'algorithme glouton 1.

Q. 27 Démontrer, *au moyen d'un variant bien choisi*, la terminaison de l'algorithme glouton.

Q. 28 Démontrer, *au moyen d'un invariant bien choisi*, la correction de l'algorithme glouton. Il faut donc démontrer que, pour une instance $(X, \mathcal{F})$, l'ensemble $\mathcal{C}$ renvoyé par l'algorithme sur cette instance est une couverture de X.

5. Notion de solution partielle

Dans les sections qui suivent, on s'intéresse à des algorithmes de résolution du problème CouvEns qui nécessitent de représenter une suite de décisions prises lors de la fabrication d'une solution. En effet un algorithme de résolution du problème CouvEns pourrait procéder de la manière suivante.

- sélectionne-t-on P_1 dans la solution ?
  - Si oui, sélectionne-t-on P_2 dans la solution ?
    - ▷ Si oui, …
    - ▷ Si non, …
  - Si non, sélectionne-t-on P_2 dans la solution ?
    - ▷ Si oui, …
    - ▷ Si non, …

Dans un tel algorithme il est nécessaire de représenter ce que nous appellerons des solutions partielles : par exemple on a décidé que P_1 serait dans la solution, on a décidé que P_2 ne serait pas dans la solution, pour les autres éléments on n'a pas encore décidé.

Solution partielle. Étant donné une instance $(X, \mathcal{F})$ du problème CouvEns, on appelle *solution partielle*, la donnée de deux ensembles $\mathcal{O} \subseteq \mathcal{F}$ et $\mathcal{N} \subseteq \mathcal{F}$ tels que $\mathcal{O} \cap \mathcal{N} = \emptyset$.

- $\mathcal{O}$ représente l'ensemble des parties que l'on a sélectionnées dans la solution en construction,
- $\mathcal{N}$ représente l'ensemble des parties que l'on a rejetées pour cette solution.

$\mathcal{F}$ étant représenté par une liste Python $F = $ `[`P_0`,` P_1`, …,` P_{m-1}`]`, une solution partielle sera représentée en Python au moyen d'un tableau `t` à valeurs dans `True`, `False` ou `None`. $\mathcal{O}$ est alors l'ensemble des P_i tels que `t[`i`]` vaut `True`, $\mathcal{N}$ est l'ensemble des P_i tels que `t[`i`]` vaut `False`. Ainsi une solution partielle est représentée en Python par le type suivant.

```
Partiel = List[Optional[bool]]
```

Aussi, toujours avec l'exemple $(X_e, \mathcal{F}_e)$, la solution partielle dans laquelle on a sélectionné P_0, mais rejeté P_3 sera représentée par le tableau Python `[True, None, None, False, None, None]`.

Solution compatible. On dit d'une solution $\mathcal{C}$ du problème pour $(X, \mathcal{F})$ qu'elle est *compatible* avec une solution partielle $(\mathcal{O}, \mathcal{N})$ dès lors que $\mathcal{O} \subseteq \mathcal{C}$ et $\mathcal{C} \cap \mathcal{N} = \emptyset$. Par exemple, pour l'instance $(X_e, \mathcal{F}_e)$, la solution partielle $(\mathcal{O} = \{P_3\}, \mathcal{N} = \{P_0\})$ représente les choix : P_3 est sélectionnée, P_0 est rejetée. Ainsi la solution $\{P_2, P_3, P_4\}$ est compatible avec cette solution partielle, en revanche la solution $\{P_0, P_1, P_4, P_5\}$ ne l'est pas, à double titre : d'une part car P_3 n'est pas présente et d'autre part car P_0 est présente.

6. Graphe associé à une solution partielle

Un sous-ensemble $\mathcal{E}$ de parties de X induit une relation binaire sur les éléments de X : deux éléments x et y de X sont en relation dès lors qu'il existe un ensemble $P \in \mathcal{E}$ tel que x et y sont dans P. Une telle relation définit un graphe non orienté sur l'ensemble de sommets X. L'étude de tels graphes est l'objet de cette section.

Graphe associé à une solution partielle. Étant donné une instance $(X, \mathcal{F})$, on définit le graphe associé à une solution partielle $(\mathcal{O}, \mathcal{N})$, comme étant le graphe non orienté $G = (S, A)$ où

- S est l'ensemble des éléments de X non couverts par $\mathcal{O}$ (autrement dit, $S = X \setminus \bigcup_{P \in \mathcal{O}} P$) et
- A est l'ensemble des arêtes reliant deux sommets dès lors qu'ils sont contenus dans un même ensemble encore autorisé (autrement dit, $\{x, y\} \in A$ si et seulement s'il existe $P \in \mathcal{F} \setminus \mathcal{N}$ tel que $x \in P$ et $y \in P$).

En reprenant l'instance $(X_e, \mathcal{F}_e)$, le graphe associé à la solution partielle $\mathcal{O} = \{P_3\}$ et $\mathcal{N} = \{P_0\}$ est le graphe $G_e = (S, A)$ représenté en figure 3 d'ensemble de sommets $S = \{0, 2, 5, 6, 8, 9, 11\}$ et d'ensemble d'arêtes $A = \{\{0, 6\}, \{0, 9\}, \{2, 5\}, \{2, 8\}, \{2, 11\}, \{5, 8\}, \{5, 11\}, \{6, 9\}, \{8, 11\}\}$.

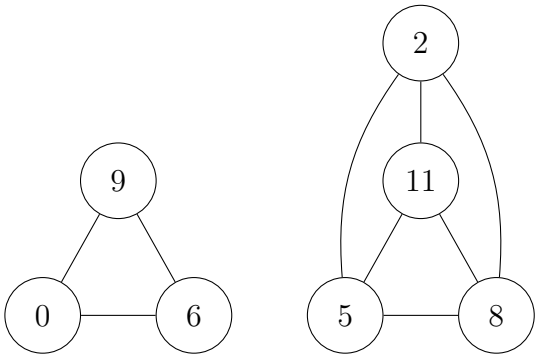


FIGURE 3 – Graphe G_e associé à la solution partielle $(\{P_3\}, \{P_0\})$ de l'instance $(X_e, \mathcal{F}_e)$

Représentation des graphes. Un graphe non orienté $G = (S, A)$, où $S = [\![0, v-1]\!]$ pour un certain $v \in \mathbb{N}$, sera représenté par listes d'adjacence. On choisit ici d'utiliser un dictionnaire PYTHON qui associe à chaque sommet du graphe l'ensemble PYTHON de ses voisins. Aussi dans la suite on suppose défini le type suivant.

```
1 Graph = Dict[int, Set[int]]
```

Le graphe de la figure 3 est donc représenté en PYTHON par le dictionnaire suivant.

```
{0: {9, 6}, 2: {8, 11, 5}, 5: {8, 2, 11},
 6: {0, 9}, 8: {2, 11, 5}, 9: {0, 6}, 11: {8, 2, 5}}
```

Q. 29 Définir une fonction `fabrique_graphe` prenant en arguments une instance $(X, \mathcal{F})$ du problème COUVENS et une solution partielle `sol_partielle` et calculant le graphe associé à cette solution partielle.

```
fabrique_graphe(n: int, f: Parties, sol_partielle: Partiel) -> Graph
```

6.1. Utilisation du graphe pour la décomposition en sous-problèmes

Q. 30 Définir une fonction PYTHON `composantes_connexes` prenant en argument un graphe et renvoyant la liste de ses composantes connexes. Une composante connexe sera représentée au moyen d'un ensemble.

```
composantes_connexes(g: Graph) -> List[Set[int]]
```

Dans le reste de cette sous-section, on fixe une instance $(X, \mathcal{F})$ du problème COUVENS. On considère le graphe $G = (S, A)$ associé à la solution partielle vide $(\emptyset, \emptyset)$. Connaissant $X = X_1 \cup X_2 \cup \ldots \cup X_p$ la décomposition en composantes connexes de G, on cherche à décomposer $\mathcal{F}$ en $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_p$, de sorte que résoudre les instances $(X_i, \mathcal{F}_i)$ permet de résoudre l'instance initiale.

Q. 31 $a)$ Proposer une définition des $\mathcal{F}_i$.

$b)$ Justifier succinctement que les $\mathcal{F}_i$ sont disjoints.

$c)$ Expliquer comment recomposer une solution $\mathcal{C}$ à l'instance $(X, \mathcal{F})$ à partir des solutions $\mathcal{C}_i$ aux instances $(X_i, \mathcal{F}_i)$.

Q. 32 On s'intéresse ici à l'algorithme qui consiste à décomposer une instance comme proposé en **Q. 31** et à résoudre chaque petite instance $(X_i, \mathcal{F}_i)$ au moyen de l'algorithme par brute force naïf étudié précédemment. Comparer le nombre de cas que doit traiter l'algorithme par brute force appliqué directement sur l'instance initiale, au nombre de cas que doit traiter l'algorithme décrit ci-avant.

6.2. Utilisation du graphe pour l'obtention d'un minorant

Lorsqu'on recherche la solution à un problème par raffinements successifs de solutions partielles, il est utile d'avoir un outil de mesure de la qualité de la solution partielle fabriquée jusqu'ici.
Supposons qu'on soit en train de considérer une solution partielle $\mathcal{C}_1$ qui ne peut être complétée que par des solutions utilisant au moins 10 ensembles alors qu'on a déjà croisé une solution complète utilisant 8 ensembles : on peut rejeter la solution partielle $\mathcal{C}_1$ sachant que la développer ne mènerait à aucune solution optimale. Aussi, afin d'évaluer la qualité d'une solution partielle, on se propose ici de mettre en place un algorithme calculant une minoration de la taille des solutions compatibles avec cette solution partielle.

Sommets non reliés. Dans le graphe $G = (S, A)$ associé à une solution partielle $(\mathcal{O}, \mathcal{N})$ d'une instance $(X, \mathcal{F})$ deux sommets x et y ne sont pas reliés (par une arête de A) si et seulement s'ils ne peuvent être couverts par un même ensemble de $\mathcal{F} \setminus \mathcal{N}$. Ainsi si deux tels sommets x et y existent, cette solution partielle ne peut être complétée qu'en ajoutant au moins deux ensembles de $\mathcal{F}$: l'un pour couvrir x, l'autre pour couvrir y.

Ensemble stable d'un graphe. Étant donné un graphe $G = (S, A)$, on dit d'un ensemble V de sommet du graphe qu'il est *stable* si les sommets de V sont deux à deux non reliés par une arête. V est donc stable si et seulement si pour tout couple (x, y) de sommets de V l'arête $\{x, y\}$ n'est pas dans A.

Q. 33 Expliquer en quoi la recherche d'un ensemble stable du graphe associé à une solution partielle fournit un minorant sur le cardinal d'une solution compatible avec cette solution partielle. Donner le minorant obtenu par la découverte d'un ensemble stable de cardinal $p \in \mathbb{N}$.

Afin de trouver un ensemble stable de sommets qui soit le plus grand possible (afin d'obtenir la meilleure minoration possible), on propose d'utiliser un algorithme glouton. L'algorithme construit un ensemble stable I en y ajoutant à chaque itération un nouveau sommet. Pour choisir un sommet à ajouter on maintient un ensemble de sommets candidats : un candidat est un sommet qu'on pourrait ajouter à I sans casser son caractère stable. Le choix glouton de l'algorithme est le suivant : à chaque étape on souhaite faire diminuer le moins possible l'ensemble des candidats, afin qu'il soit non vide le plus longtemps possible, pour construire un ensemble I le plus gros possible.

Q. 34 Définir une fonction `choix_glouton` prenant en arguments un graphe et un ensemble de sommets candidats et renvoyant un des sommets candidats qui suit le choix glouton décrit ci-avant. On devra *mettre à jour* l'ensemble des candidats, au vu du choix glouton effectué.

```
choix_glouton(g: Graph, candidats: Set[int]) -> int
```

Q. 35 En déduire une fonction `stable` prenant en argument un graphe et renvoyant un ensemble stable en itérant,tant que cela est possible, le choix glouton ci-dessus.

```
stable(g: Graph) -> Set[int]
```

Q. 36 a) Cet algorithme fournit-il un ensemble stable qui est de cardinal maximal (autrement dit, un ensemble stable tel qu'il n'existe aucun ensemble stable de cardinal strictement supérieur) ? On attend une justification succincte ou un contre-exemple.

b) Cet algorithme fournit-il un ensemble stable qui est maximal pour l'inclusion (autrement dit, un ensemble stable qui n'est contenu dans aucun ensemble stable autre que lui-même) ? On attend une justification succincte ou un contre-exemple.

Q. 37 Déduire des question précédentes une fonction `minorant_couv_ens` prenant en argument une instance du problème $(X, \mathcal{F})$ et une solution partielle et renvoyant un minorant du cardinal des solutions du problème CouvEns, compatibles avec cette solution partielle.

```
minorant_couv_ens(n: int, f: Parties, sol_partielle: Partiel) -> int
```

7. Séparation et évaluation

Une stratégie d'exploration de l'espace des solutions pour l'instance $(X, \mathcal{F})$ est de décider successivement si on sélectionne ou non P_0, P_1, ... Une telle exploration forme un arbre : la racine correspond à la solution partielle $(\emptyset, \emptyset)$ (aucune décision n'a été prise). Un nœud de profondeur i de cet arbre correspond à une solution partielle de la forme $(\mathcal{O}, \mathcal{N})$ avec $\mathcal{O} \cup \mathcal{N} = [\![0, i-1]\!]$, autrement dit on a décidé pour chaque j dans $[\![0, i-1]\!]$ si on sélectionne on non l'ensemble P_j.

En visitant tous les nœuds de cet arbre on réaliserait en fait un algorithme par brute force. Le schéma algorithmique de séparation et évaluation permet de limiter le nombre de nœuds visités en maintenant, tout au long de l'exploration, la meilleure solution rencontrée jusque lors. Notons $\mathcal{C}^b$ cette meilleure solution. Si une solution partielle $(\mathcal{O}, \mathcal{N})$ ne peut être complétée qu'en des solutions de cardinal strictement supérieur à $|\mathcal{C}^b|$, il n'est pas intéressant d'explorer le sous-arbre issu de ce nœud $(\mathcal{O}, \mathcal{N})$. Il s'agit donc de trouver un minorant μ des valeurs des solutions complétant $(\mathcal{O}, \mathcal{N})$ afin de tester si $\mu > |\mathcal{C}^b|$ et ce sans explorer lesdites solutions. Pour cela on utilisera les résultats de la section 6.2.

L'algorithme glouton de la section 4 permet de calculer une solution pour $(X, \mathcal{F})$ *non nécessairement optimale*, donnant ainsi une première valeur pour C^b. Afin d'améliorer cette valeur, on va calculer pour chaque nœud $(\mathcal{O}, \mathcal{N})$, une solution pour $(X, \mathcal{F})$ *non nécessairement optimale* complétant $(\mathcal{O}, \mathcal{N})$.

Q. 38 En s'inspirant de l'algorithme glouton de la section 4 (Algorithme 1, page 10), écrire le pseudo-code d'un algorithme fournissant une solution compatible avec une solution partielle donnée si cela est possible. Dans le cas contraire l'algorithme renverra `None`.

On suppose définie dans la suite une fonction `glouton_compatible` implémentant cet algorithme et dont la signature est la suivante.

```
glouton_compatible(n: int, f: Parties, partiel: Partiel) -> Optional[Tuple[int,
↪  List[bool]]]
```

Pour une instance $(X, \mathcal{F})$ et une solution partielle $(\mathcal{O}, \mathcal{N})$ cette fonction renvoie **`None`** si $(\mathcal{O}, \mathcal{N})$ ne peut être complétée en une solution. En pareil cas, il n'est pas intéressant d'explorer le sous-arbre issu du nœud $(\mathcal{O}, \mathcal{N})$. Dans le cas contraire, elle renvoie un couple `(v, s)` où `s` représente par sa fonction indicatrice une solution $\mathcal{C}$ pour $(X, \mathcal{F})$, non nécessairement optimale, et où `v` est le nombre d'ensembles utilisés par cette solution $\mathcal{C}$ (*i.e.* le nombre de **`True`** dans le tableau `s`).

Q. 39 Définir une fonction `separation_et_evaluation` implémentant l'algorithme décrit ci-avant.

```
separation_et_evaluation(n: int, f: Parties) -> Tuple[int, List[bool]]
```

∴ Fin du sujet ∴

A Notations mathématiques utilisées dans le sujet

- $\mathbb{N}$ est l'ensemble des entiers naturels $\{0, 1, 2, \ldots\}$.
- $\mathbb{Z}$ est l'ensemble des entiers relatifs $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
- Lorsque $a \in \mathbb{Z}$ et $b \in \mathbb{Z}$, $[\![a, b]\!]$ désigne l'ensemble des entiers z vérifiant $a \leqslant z \leqslant b$, à savoir $\{a, a+1, a+2, \ldots, b-1, b\}$ si $a \leqslant b$, et l'ensemble vide sinon.

- $\emptyset$ désigne l'ensemble vide.
- Si X et Y sont deux ensembles, alors
  - $X \subseteq Y$ désigne le fait que l'ensemble X est inclus ou égal à l'ensemble Y ;
  - $X \cup Y$ désigne l'ensemble union de X et Y ;
  - $X \cap Y$ désigne l'intersection de X et Y ;
  - $X \setminus Y$ désigne l'ensemble des éléments de X qui ne sont pas dans Y.
- Si X est un ensemble fini, alors $|X|$ désigne son cardinal, c'est-à-dire son nombre d'éléments.

- Si $\mathcal{C}$ est un ensemble d'ensembles $\bigcup_{X \in \mathcal{C}} X$ désigne l'union de tous les ensembles X de $\mathcal{C}$. Ainsi si $\mathcal{C} = \{X_1, X_2, \ldots, X_p\}$ est un ensemble fini de cardinal $p > 0$, alors $\bigcup_{X \in \mathcal{C}} X = X_1 \cup X_2 \cup \ldots \cup X_p$. Dans le cas particulier où $\mathcal{C} = \emptyset$, $\bigcup_{X \in \mathcal{C}} X = \emptyset$.

- Si $(u_1, u_2, \ldots, u_n)$ est une suite finie de nombres, $\sum_{i=1}^{n} u_i$ désigne la somme $u_1 + u_2 + \ldots + u_n$.

B Typage Python

Dans tout l'énoncé, et dans toutes les réponses on supposera la librairie `typing` déjà importée au moyen de la ligne ci-dessous.

```
1 from typing import Dict, Set, List, Tuple, Optional
```

B.1 Signature de fonction

Cette librairie nous permet d'intégrer des annotations de type aux fonctions Python définies dans l'énoncé, et d'expliciter la signature attendue pour les fonctions qu'il vous est demandé de coder. L'énoncé peut par exemple demander de coder une fonction f ayant la signature suivante, où x_1, x_2, …, x_n sont des noms de variables, t_1, t_2, …, t_n et t sont des types.

```
f(x1: t1, x2: t2, ..., xn: tn) -> t
```

Vous devez alors fournir une fonction Python nommée f dont les arguments sont nommés x_1, x_2, …, x_n, que vous pouvez supposer être respectivement de type t_1, t_2, …, t_n au moment de l'appel. Vous devez assurer en retour que votre fonction renvoie bien un objet de type t dans tous les cas. On dit alors que t est le type de sortie de la fonction `f`.

B.2 Types prédéfinis

La librairie `typing` permet la manipulation des types de base suivants.

- `bool` le type des booléens.
- `int` le type des entiers relatifs. Par exemple `0`, `1`, `-3` sont de type `int`.
- `float` le type des nombres flottants.
- `str` le type des chaînes de caractères. Par exemple `"toto"`, `""`.

De plus elle permet aussi de manipuler les types composés suivants.

`List[`t**`]`** , où t est un type, est le type des listes d'éléments de type t. Par exemple `[1, 2, 3]` est un objet de type `List[int]`, `[1, 2, "toto"]` n'en est pas un.

`Tuple[`t_1**`, `**t_2**`, …, `**t_n**`]`** , où $t_1, t_2, \ldots, t_n$ sont des types, représente le type des n-uplets dont la i-ème coordonnée est de type t_i. Par exemple `(1, 2, "a")` est de type `Tuple[int, int, str]` et `(1, ["toto", "abc"])` est de type `Tuple[int, List[str]]`.

`Set[`t**`]`** , où t est un type, est le type des ensembles d'éléments de type t. Par exemple `{1, 2, 3}` est un objet de type `Set[int]`.

`Dict[`t_k**`, `**t_v**`]`** , où t_k et t_v sont des types, est le type des dictionnaires dont les clés sont de type t_k et les valeurs de type t_v. Par exemple `{"abc" : 2.0, "toto" : 3.5}` est un objet de type `Dict[str, float]`.

`Optional[`t**`]`** , où t est un type, est le type des objets qui sont de type t ou qui valent `None`. Ce type est particulièrement utile pour les fonctions qui ne renvoient pas un résultat dans tous les cas. Par exemple la fonction Python `mal_typee` ci-dessous à gauche est mal typée car appelée sur `y` valant 0 cette fonction renvoie `None` qui n'est pas de type `int`. En revanche la fonction `bien_typee` ci-dessous à droite est bien typée : elle précise, dans son type de sortie, que la fonction peut renvoyer `None` ou un `int`.

```python
def mal_typee(y: int) -> int:
  if y != 0:
    return 2025 // y
```

```python
def bien_typee(y: int) -> Optional[int]:
  if y != 0:
    return 2025 // y
```

B.3 Définition de nouveaux types

Enfin la librairie `typing` permet de définir des alias de type (*i.e.* des nouveaux noms de type) en écrivant simplement :

```
NouveauType = t
```

où t est un type déjà défini. Par exemple la ligne ci-dessous définit le nouveau type `Partiel` comme étant le type des listes dont les éléments sont des booléens ou `None`.

```
Partiel = List[Optional[bool]]
```

Après cette définition, on peut par exemple définir une fonction Python `contientNone(p: Partiel) -> bool` permettant de tester si l'élément `p` de type `Partiel` contient un `None` ou non.