Concours Blanc d'Informatique – MPSI

Durée : **4h**. Calculatrices non autorisées.
Le soin et la clarté de la rédaction pourront faire varier la note de ±1 point. Il n'est pas attendu que vous arrivez au bout du sujet (le barème dépassera 100 points).
Les fonctions devront être commentées lorsque cela est utile, et l'indentation devra être respectée.
Le sujet fait 6 pages.

Les problèmes de la partition et de la couverture optimale

Dans ce sujet, on étudie le problème classique de la partition d'un ensemble. Étant donné $n \in \mathbb{N}^*$, on considère un ensemble E qui possède n éléments. Étant donné $p \in \mathbb{N}$, on considère également une famille $\mathcal{F} = (X_0, \cdots, X_p)$ constituée de sous-ensembles de E, i.e. pour tout $k \in \{0, \cdots, p\}$ on a $X_k \subset E$.

On dit que la famille $\mathcal{F} = (X_0, \cdots, X_p)$ est une famille **couvrante** (de E) si on a $\bigcup_{k=0}^{p} X_k = E$, ou encore si chaque élément de E est dans au moins un des sous-ensembles X_k.

Étant donné $q \in \mathbb{N}$, soit $Y_0, \cdots, Y_q$ des sous-ensembles de E. On dit que la famille $(Y_0, \cdots, Y_q)$ est une **partition** (de E) si elle est couvrante et si les sous-ensembles Y_i sont disjoints deux à deux, donc si :

$$\bigcup_{i=0}^{q} Y_i = E \qquad \text{et} \qquad \text{si } i \neq j, \text{ alors } Y_i \cap Y_j = \varnothing$$

Cela revient à dire que chaque élément de E est dans un et un seul des sous-ensembles Y_i (il est possible qu'un ou plusieurs des ensembles Y_i soit vide).

Le problème de la partition est le suivant : étant donné $\mathcal{F}$ une famille couvrante de E, peut-on en extraire (i.e. trouver une sous-famille qui est) une partition de E ?

Dans l'exemple ci-dessous, on considère un ensemble E qui contient 9 points. La famille $\mathcal{F}$ contient 5 ensembles, repérés par des cadres qui englobent les points (deux ensembles à 2 éléments, deux ensembles à 3 éléments et un ensemble à 4 éléments). Cette famille $\mathcal{F}$ est couvrante : chaque point de E appartient à au moins un ensemble de la famille $\mathcal{F}$. On remarque que la sous-famille constituée des 3 ensembles en traits épais sur le dessin est une partition de E.

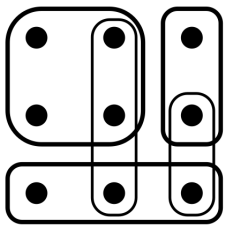

Figure 1: Exemple de famille $\mathcal{F}$ de 5 ensembles recouvrant un ensemble E de 9 éléments. La famille de 3 ensembles en gras est une partition de E.

Un rappel de quelques syntaxes utiles est donnée page 6.

Partie I

Pour simplifier, on considère $E = \{0, \cdots, n-1\}$. Un sous-ensemble X de E est représenté par une liste `L` de booléens de la manière suivante : pour tout $i \in E$, la valeur `L[i]` est égale à `True` si $i \in X$, et `False` sinon. Par exemple, l'ensemble $X = \{0, 2, 3, 5\}$ est représenté par la liste

```
L = [True, False, True, True, False, True]
```

Une telle liste sera appelé dans ce devoir une **booliste**. On remarque qu'un même ensemble X peut être représenté par plusieurs boolistes : il suffit d'ajouter un ou plusieurs `False` à la fin de toute booliste qui le représente. Par exemple la liste

```
[True, False, True, True, False, True, False, False]
```

représente également l'ensemble $\{0, 2, 3, 5\}$.

1) Écrire une fonction `cardinal` qui prend en argument une booliste `L` représentant un ensemble X, et qui renvoie le nombre d'éléments dans X.

2) Écrire une fonction `rajout`, qui prend en argument une booliste `L` représentant un ensemble X ainsi qu'un entier m, et qui :

- Si m est supérieur ou égal au nombre d'éléments de `L`, renvoie la booliste de taille m qui représente l'ensemble X, en rajoutant autant de `False` que nécessaire.
- Si m est inférieur strictement au nombre d'éléments de `L`, affiche le message "ErReUr FaTaLe" et ne renvoie rien.

3) On souhaite écrire une fonction `union`, qui prend en argument deux boolistes `L1` et `L2` représentant des ensembles X_1 et X_2, et qui renvoie une booliste qui représente l'ensemble $X_1 \cup X_2$. On donne la fonction suivante :

```
1  def union(L1,L2):
2      m = max(len(L1),len(L2))
3      L1 = rajout(L1,m)
4      L2 = rajout(L2,m)
5      for i in range(m):
6          if L2[i]==True:
7              L1[i]=True
8      return L1
```

a) Expliquer pourquoi le programme ne fonctionnerait pas correctement sans les lignes 3 et 4.

b) On considère la série d'instructions suivante :

```
1  M = [True, False, True]
2  N = [False, False, True, True]
3  P = union(M,N)
```

En supposant avoir exécuté les deux blocs de code ci-dessus, que contiennent les variables `M`, `N`, `P`, `L1` et `L2` ?

c) La fonction `union` possède-t-elle un effet de bord ? Écrire une fonction `union2` qui réalise la même opération que `union` mais avec le comportement inverse en termes d'effet de bord.

Dans la suite du devoir, on pourra utiliser ***union*** *ou* ***union2*** *et on ne se souciera plus des effets de bords éventuels.*

4) Écrire une fonction `diff`, qui prend en argument deux boolistes représentant des ensembles X_1 et X_2, et qui renvoie une booliste qui représente l'ensemble $X_1 \setminus X_2$ (c'est-à-dire l'ensemble des éléments de X_1 qui ne sont pas dans X_2).

Partie II

On souhaite maintenant représenter un sous-ensemble X de E par un simple entier. Pour cela, on associe à X l'entier $s = \sum_{i \in X} 2^i$. Par exemple, l'ensemble $\{0, 1, 3, 6\}$ est représenté par l'entier $2^0+2^1+2^3+2^6 = 75$. Par convention, l'entier 0 représente l'ensemble vide. On admet que cette représentation est unique et on prendra garde à ne pas confondre l'entier s qui représente X et les entiers contenus dans X.

5) Écrire 75 en base 2. Quel lien peut-on faire avec l'ensemble $\{0, 1, 3, 6\}$?

6) Donner les ensembles représentés par les entiers 13 et 151.

7) On rappelle que $E = \{0, \cdots, n-1\}$. Quel est le plus grand entier qui représente un sous-ensemble de E ?

8) Écrire une fonction `set2int` qui prend en argument une booliste `L` qui représente un ensemble X, et qui renvoie l'entier s qui représente X.

9) Quelle est la complexité de `set2int` ? On la déterminera en fonction de la longueur de `L`, notée m.

10) Écrire une fonction `int2set` qui prend en argument un entier naturel s qui représente un ensemble X, et qui renvoie une booliste `L` qui représente X. On vérifiera dans le programme que s est bien un entier naturel ; dans le cas contraire, le programme affichera un message d'erreur et ne renverra rien.

On peut désormais manipuler des sous-ensembles de E à partir de leur représentation par un entier. Grâce aux fonctions `set2int` et `int2set` ci-dessus, il est possible d'étendre les fonctions de la partie I aux entiers représentant des ensembles. *Dans le reste du sujet, on supposera que les fonctions de la partie I ont été réécrites afin de pouvoir opérer directement sur les entiers (arguments et résultats).*

Intermède

On rappelle que $E = \{0, \cdots, n-1\}$. On considère à présent une famille $\mathcal{F} = (X_0, \cdots, X_p)$ de sous-ensembles de E dont on cherche à savoir si une sous-famille est une partition de E. Sans perte de généralité, on peut supposer que les ensembles $X_0, \cdots, X_p$ de $\mathcal{F}$ sont tous distincts deux à deux. On représente la famille $\mathcal{F}$ par une liste d'entiers `F` : pour tout $q \in \{0, \cdots, p\}$, `F[q]` correspond à l'entier qui représente X_q au sens de la partie II. Par exemple, la liste `F=[17,3,8,22]` correspond à la famille $\mathcal{F} = (X_0, X_1, X_2, X_3)$ où :

- $X_0 = \{0, 4\}$ car `F[0]` $= 17 = 2^0 + 2^4$
- $X_1 = \{0, 1\}$ car `F[1]` $= 3 = 2^0 + 2^1$
- $X_2 = \{3\}$ car `F[2]` $= 8 = 2^3$
- $X_3 = \{1, 2, 4\}$ car `F[3]` $= 22 = 2^1 + 2^2 + 2^4$

En particulier, comme les ensembles $X_0, \cdots, X_p$ sont supposés distincts, la liste `F` contient des entiers distincts. Une sous-famille $\mathcal{G}$ de $\mathcal{F}$ est constituée de certains des ensembles de la famille $\mathcal{F}$, et peut donc être vue comme une sous-liste `G` de la liste `F`. Par exemple, la sous-liste `G=[22,3]` représente la sous-famille (X_3, X_1), ou de manière équivalente la sous-famille (X_1, X_3), l'ordre ici ne jouant aucun rôle.

Dans tout le reste du sujet, on suppose que $\mathcal{F}$ est une famille couvrante de E, et que `F` est une liste qui représente la famille $\mathcal{F}$. On suppose que la liste `F` et l'entier n sont accessibles dans toutes les fonctions, sans avoir à les définir ni à les passer en argument.

Les parties III et IV suivantes sont indépendantes, bien qu'elles traitent toutes deux du problème $(\mathcal{P})$ défini plus bas. La partie V est également indépendante des deux précédentes.

Partie III

11) On affirme qu'une sous-famille $\mathcal{G}$ est une partition de E si et seulement si la somme des éléments de `G` est égale à un entier σ_n. Que vaut σ_n en fonction de n ? On justifiera brièvement le résultat.

Ainsi, pour trouver une partition, on est amené à résoudre le problème suivant (où `L` joue le rôle de `F`, `M` joue le rôle de `G` et `sum` joue le rôle de σ_n) :

$(\mathcal{P})$: *Étant donné une liste* `L` *d'entiers naturels distincts et un entier naturel* `sum`*, on recherche une sous-liste* `M` *de* `L` *telle que la somme des éléments de* `M` *est égale à* `sum`*.*

Dans cette partie et la suivante, on va s'intéresser uniquement au problème $(\mathcal{P})$. On notera que, selon les valeurs de `L` et de `sum`, il est possible qu'une telle sous-liste `M` n'existe pas, auquel cas le problème $(\mathcal{P})$ est sans solution.

12) Écrire une fonction `sansDoublon` qui prend en argument une liste `L` d'entiers naturels et qui retourne `True` si la liste des valeurs contenues dans `L` sont toutes distinctes, et qui retourne `False` sinon.

13) Écrire une fonction `tri` qui prend en argument une liste `L` d'entiers naturels et qui retourne la liste triée par ordre décroissant.

Pour résoudre $(\mathcal{P})$, on utilise un algorithme glouton :

- On trie d'abord la liste `L` par ordre décroissant, et on initialise `M` à la liste vide.
- Puis on parcourt les différentes valeurs de `L` : pour chaque indice i, on ajoute `L[i]` à `M` à condition que la somme des éléments de `M` reste inférieure ou égale à `sum`. Sinon on ne fait rien.

14) Écrire une fonction `glouton` qui prend en argument `L` et `sum` et qui réalise l'algorithme ci-dessus. On testera si la liste `M` obtenue est effectivement solution de $(\mathcal{P})$. Si c'est le cas, on la retourne, sinon, on ne retournera rien.

15) En supposant que le problème $(\mathcal{P})$ admette une solution, est-ce que l'algorithme glouton retourne toujours une sous-liste `M` qui convient ?

Partie IV

Dans cette partie, on résout le problème $(\mathcal{P})$ par la force brute : on va essayer toutes les sous-listes `M` possibles au moyen d'un algorithme récursif. Pour simplifier, on va supposer que l'algorithme doit retourner `True` si une telle sous-liste `M` existe, et `False` sinon (on pourrait l'adapter pour qu'il retourne `M` comme l'algorithme glouton). On considère donc l'algorithme récursif suivant :

```python
def recuPart(L,sum):
    '''détermine si L contient une sous-liste dont la somme des
       éléments donne sum. Renvoie True si c'est le cas, False
       sinon.'''

    ### deux conditions d'arrêt
    if sum == 0:
        return True
    if L == [] :
        return False

    ### appels récursifs

    # on vérifie si on peut obtenir sum à partir de L...

    bool1 = recuPart( L[1:], sum )
    # ... ou bien en excluant le premier élément de L, ce qui revient
       à obtenir sum à partir de L[1:]

    bool2 = recuPart( L[1:], sum-L[0] )
    # ... ou bien en incluant le premier élément de L, ce qui revient
       à obtenir sum-L[0] à partir de L[1:]

    return bool1 or bool2
```

16) On lance l'instruction `recuPart([6,5,4],10)`. Cette instruction va appeler `recuPart` deux fois avec des arguments différents, qui vont eux-mêmes appeler `recuPart` avec des arguments différents, etc. Avec un arbre (similaire aux arbres de probabilités), décrire les différents appels de `recuPart` ainsi que les arguments associés à chaque appel.

17) Montrer par récurrence que l'instruction `recuPart(L, sum)` termine.

18) Complexité : on admet que le pire cas est lorsque `sum` est strictement supérieur à la somme de tous les éléments de `L`. Dans ce pire cas, on note c_n le nombre d'opérations élémentaires réalisées pour une liste `L` de taille n. Exhiber une relation de récurrence sur c_n en la justifiant.

19) En déduire la complexité de `recuPart`. L'algorithme est-il exploitable pour de grandes valeurs de n ?

Partie V

Dans cette partie, on s'intéresse à un problème différent mais connexe à celui de la partition, à savoir celui de la couverture optimale. On rappelle que $\mathcal{F} = (X_0, \cdots, X_p)$ est une famille couvrante de l'ensemble E.

Le problème de la couverture optimale est le suivant : en supposant que la famille $\mathcal{F}$ est couvrante, de combien d'ensembles de $\mathcal{F}$ a-t-on besoin, au minimum, pour couvrir E ? Ou encore, quelle est la (ou une) plus petite sous-famille de $\mathcal{F}$ qui soit couvrante ? Une telle famille sera dite **couvrante optimale**.

Dans l'exemple de la Figure 1, on remarque qu'aucune sous-famille de $\mathcal{F}$ constituée de 2 ensembles n'est couvrante. En revanche, la sous-famille constituée des 3 ensembles en traits épais sur le dessin est couvrante et donc est couvrante optimale.

On introduit des notations spécifiques à cette partie. Une sous-famille $\mathcal{H}$ de $\mathcal{F}$ est constituée de certains des ensembles de la famille $\mathcal{F}$, et donc est de la forme $(X_{\ell_0}, X_{\ell_1}, \cdots, X_{\ell_r})$, où $\ell_0, \ell_1, \cdots, \ell_r$ est une suite finie strictement croissante à valeurs dans $[\![0, p]\!]$. On a donc en particulier $r \leq p$. Par conséquent, $\mathcal{H}$ est définie par l'ensemble des indices $\{\ell_0, \ell_1, \cdots, \ell_r\}$. Le nombre d'éléments de $\mathcal{H}$, appelé aussi cardinal de $\mathcal{H}$, est donc le nombre d'éléments de l'ensemble $\{\ell_0, \ell_1, \cdots, \ell_r\}$.

Au sens de la partie II, cet ensemble d'indices peut une nouvelle fois être représenté par un entier. Par exemple, si $\mathcal{F}$ est la famille $(\{0,4\}, \{0,1\}, \{3\}, \{1,2,4\})$, la sous-famille $\mathcal{H} = (\{0,4\}, \{3\})$, qui est constituée des ensembles X_0 et X_2, est de cardinal 2 et est représentée par l'entier 5, puisque $2^0 + 2^2 = 5$.

Noter que désormais trois objets distincts sont représentés par des entiers :

- Les élements de $E = \{0, 1, \cdots, n-1\}$: dans ce cas la lettre i sera de préférence utilisée.
- Les sous-ensembles de E : dans ce cas la lettre s sera de préférence utilisée, et s représente l'ensemble $X = \texttt{int2set}(s)$.
- Les sous-familles de $\mathcal{F}$: dans ce cas la lettre h sera de préférence utilisée, et h représente la sous-famille définie par les ensembles X_ℓ pour $\ell \in \texttt{int2set}(h)$. De plus, chaque ensemble X_ℓ est lui-même représenté par l'entier `F[`ℓ`]`.

Une façon de trouver une sous-famille couvrante de petit cardinal consiste à utiliser un algorithme glouton. L'idée est de construire une sous-famille étape par étape, en commençant par le sous-ensemble X_ℓ de plus grand cardinal, puis en ajoutant à chaque fois le sous-ensemble X qui permet de couvrir le plus de nouveaux éléments. On continue ainsi jusqu'à avoir couvert tous les éléments de E.

20) Proposer un exemple de famille où l'algorithme glouton ne renvoie pas une sous-famille couvrante optimale.

21) Écrire une fonction `reste`, qui prend en argument deux entiers s_1 et s_2 représentant les sous-ensembles X_1 et X_2 de E, et qui renvoie le nombre d'éléments de X_1 qui ne sont pas dans X_2.

22) Écrire une fonction `glouton`, sans argument, qui renvoie un entier h représentant une sous-famille couvrante $\mathcal{H}$ de $\mathcal{F}$ grâce à l'algorithme glouton décrit ci-dessus.

23) Écrire une fonction `couverture`, qui prend en argument un entier h représentant une sous-famille $\mathcal{H}$ de $\mathcal{F}$ et qui renvoie `True` si $\mathcal{H}$ est couvrante, et `False` sinon.

24) Écrire une fonction `optimale`, sans argument, qui parcourt toutes les sous-familles possibles afin de renvoyer une sous-famille couvrante de $\mathcal{F}$ optimale.

Quelques rappels de syntaxe :

- `len(L)` retourne le nombre d'éléments de la liste `L`.
- `L[k]` retourne l'élément d'indice `k` de `L`.
- `L[k] = x` permet de modifier l'élément d'indice `k` de `L`, et de le remplacer par `x`.
- `L1 + L2` retourne la concaténation de deux listes `L1` et `L2`.
- `L.append(x)` ajoute l'élément `x` à la fin de `L`.