# Projet individuel d'algorithmique-programmation IPF : groupe 3 (Enseignant : C. Mouilleron)

10 mars 2018

## 1 Informations générales

### 1.1 Travail à rendre

Le projet est à réaliser en OCaml **individuellement**. Il sera accompagné d'un **dossier** contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées).

Même si le sujet est décomposé en questions, il est possible qu'une question se résolve par l'écriture d'une ou plusieurs fonctions intermédiaires. Celles-ci doivent comporter une interface également.

Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés.

Sur le site du cours figure un petit document sur ce que l'on attend dans un rapport. Consultez-le !

**Attention, le code doit être purement fonctionnel : pas de boucle, pas d'affectation, pas de tableau. La seule tolérance concerne les affichages (voir document en ligne).**

### 1.2 Calendrier et procédure de remise

Le projet est à rendre le **mardi 3 avril 2018** minuit au plus tard sur le site de dépots `exam.ensiie.fr`. Cliquez sur **IPF_S2_2018**. Vous y déposerez une archive contenant votre rapport au format pdf (impérativement) et le code de votre projet.
% newcounterquestcounter

## 2 Énoncé du projet : Étude de différentes approches pour résoudre SUBSET-SUM-OPT

On considère le problème suivant :

> SUBSET-SUM-OPT – Étant donnés un ensemble fini $E$ d'entiers strictement positifs et un entier cible $s$, trouver l'entier $s' \leq s$ le plus grand possible tel qu'il existe un sous-ensemble $E' \subseteq E$ vérifiant $\sum_{e \in E'} e = s'$.

Par exemple, sur l'ensemble $\{1, 5, 9\}$ et avec la cible $8$, la réponse attendue est $s' = 6$ et cette somme est atteinte pour le sous-ensemble $E' = \{1, 5\}$.

Ce problème est NP-difficile [1], ce qui implique en pratique qu'il n'existe pas à l'heure actuelle de solution pleinement satisfaisante, et qu'il y a peu de chances pour qu'une telle solution existe un jour.

1. Il n'est pas nécessaire de savoir ce qu'est un problème NP-difficile pour pouvoir faire ce sujet.

L'objectif de ce projet est de tester quatre approches possibles pour résoudre ce problème, et de les comparer. Les deux dernières approches sont plus difficiles à implanter et peuvent être traitées de façon complètement indépendante. Enfin, vous pouvez commencer la dernière partie dès que vous avez un code fonctionnel pour au moins une des approches demandées.

Dans toute la suite, les ensembles d'entiers seront représentés par des listes sans doublon d'entiers triés par ordre croissant. Vous êtes invités à relire le documentation du module `List` de OCaml. En particulier, vous aurez sans doute besoin d'utiliser les fonctions `List.map`, `List.fold_left`, `List.sort` et `List.rev`.

# 3 Approche naïve

*Question 1*
Écrire une fonction `sum: int list -> int` qui, sur la donnée d'une liste d'entiers $\ell$, calcule la somme des éléments de $\ell$.

*Question 2*
Écrire une fonction `powerset: 'a list -> ('a list) list` qui, sur la donnée d'une liste $\ell$, retourne la liste de toutes les sous-listes que l'on peut extraire de $\ell$.

Aucun ordre n'est imposé sur les sous-listes du résultat.

Ainsi, `powerset [1;2]` pourra par exemple retourner `[ [1]; [1;2]; []; [2] ]`.

*Question 3*
Utiliser les deux questions précédentes pour définir une fonction `subset_sum_0: int list -> int -> int` répondant au problème `SUBSET-SUM-OPT`.

La fonction `subset_sum_0` pourra être utilisée comme solution de référence lors des tests pour les parties suivantes.

# 4 Approche plus directe

Au lieu de calculer tous les sous-ensembles, puis de regarder la somme des éléments pour chacun d'entre eux, on se propose de calculer directement l'ensemble des sommes atteignables. Par exemple, pour l'ensemble $E = \{1, 3, 4\}$, on va chercher à calculer l'ensemble $\{0, 1, 3, 4, 5, 7, 8\}$ constitué de toutes les valeurs que l'on peut obtenir en faisant la somme des éléments d'un sous-ensemble de $E$.

Pour se faire, on peut appliquer l'algorithme suivant :

**Algorithme 1 :**

**Entrée :** un ensemble $E$
**Sortie :** l'ensemble des entiers $n$ tels qu'il existe un sous-ensemble $E' \subseteq E$ vérifiant $\sum_{e \in E'} e = n$.

1 $T \leftarrow \{0\}$
2 **pour tout** $e \in E$ **faire**
3 $\quad T' \leftarrow \{t + e \mid t \in T\}$ // on ajoute e à tous les éléments de T
4 $\quad T \leftarrow T \cup T'$
5 **retourner** $T$

*Question 4*

Écrire une fonction `get_all_sums: int list -> int list` reposant sur l'algorithme précédent. L'utilisation de fonctions auxiliaires est recommandé.

*Question 5*
Écrire une fonction `subset_sum_1: int list -> int -> int` répondant au problème SUBSET-SUM-OPT à l'aide de la fonction précédente.

# 5 Approche avec nettoyage

Une variante de l'approche précédente consiste à appliquer un nettoyage au fur et à mesure du calcul des sommes atteignables. La modification proposée est détaillée dans les algorithmes 2 et 3 ci-après.

**Algorithme 2 :** variante de l'algorithme 1 avec nettoyage

**Entrée :** un ensemble $E$, un entier cible $s$ et une précision $\delta \in \mathrm{R}_+^*$
**Sortie :** un ensemble d'entiers $n$ tels qu'il existe un sous-ensemble $E' \subseteq E$ vérifiant $\sum_{e \in E'} e = n$.

```
1 T ← {0}
2 pour tout e ∈ E faire
3 |   T' ← {t + e | t ∈ T}          // on ajoute e à tous les éléments de T
4 |   T ← nettoyage(T ∪ T', s, δ)
5 retourner T
```

**Algorithme 3 :** nettoyage

**Entrée :** un ensemble $E$ non vide, un entier $s$ et une précision $\delta \in \mathrm{R}_+^*$
**Sortie :** un ensemble $E' \subseteq E$

```
1 m ← plus petit élément de E
2 T ← {m}
3 pour tout e ∈ E \ {m} pris dans l'ordre croissant faire
4 |   si e > (1 + δ)m et e ≤ s alors
5 |   |   T ← T ∪ {e}          // on garde e
6 |   |   m ← e
7 retourner T
```

*Question 6*
Écrire une fonction `clean_up: int list -> int -> float -> int list` reposant sur l'algorithme 3. Testez la fonction avec $s = 90$, $\delta = 0.1$ et la liste des entiers de 1 à 100, et essayez d'expliquer ce qu'elle fait dans votre rapport.

*Question 7*
Écrire une fonction `subset_sum_2: float -> int list -> int -> int` qui utilise la fonction précédente pour répondre au problème `SUBSET-SUM-OPT`. L'argument de type `float` correspond à la précision $\delta$ utilisée lors du nettoyage.

Dans un premier temps, on pourra faire des tests avec $\delta = 0.01$. N'hésitez pas ensuite à essayer des valeurs de plus en plus petites pour $\delta$.

# 6 Approche de type *Diviser pour régner*

*Question 8*
Écrire une fonction `is_feasible: int -> int list -> int list -> bool` qui, sur la donnée d'un entier cible $s$, d'une liste $\ell_1$ d'entiers croissants et d'une liste $\ell_2$ d'entiers décroissants, retourne `true` si $s$ s'écrit comme la somme d'un élément de $\ell_1$ et d'un élément de $\ell_2$, et `false` sinon.

Ainsi :
- `is_feasible 12 [1;2;4;7] [10;8;2]` devra retourner `true`;
- `is_feasible 16 [1;2;4;7] [10;8;2]` devra retourner `false`.

On veillera à utiliser au maximum le fait que les listes sont triées.

*Question 9*
En s'inspirant de la question précédente, écrire une fonction `best_feasible: int -> int list -> int list -> int` qui, sur la donnée d'un entier cible $s$, d'une liste $\ell_1$ d'entiers croissants et une liste $\ell_2$ d'entiers décroissants, retourne le plus grand entier $s' \leq s$ tel que $s' = u + v$ avec $u$ dans $\ell_1$ et $v$ dans $\ell_2$.

Ainsi :
- `best_feasible 12 [1;2;4;7] [10;8;2]` devra retourner 12,
- `best_feasible 16 [1;2;4;7] [10;8;2]` devra retourner 15.

*Question 10*
Proposer une fonction `subset_sum_3: int list -> int -> int` répondant au problème SUBSET-SUM-OPT de la façon suivante :

1. répartir les éléments de l'ensemble de départ en deux sous-ensembles de tailles égales (à 1 près),
2. calculer l'ensemble des sommes atteignables pour le premier sous-ensemble en utilisant la fonction `get_all_sums` (cf partie 4),
3. calculer de même l'ensemble des sommes atteignables pour le second sous-ensemble,
4. utiliser la fonction précédente

# 7 Comparaison des approches et amélioration

*Question 11*
Écrire une fonction `gen_random: int -> int -> int list` qui, sur la donnée de deux entiers naturels $n$ et $m$, retourne un ensemble fini à $n$ éléments contenant des entiers entre 0 et $m - 1$ inclus.

On pourra faire appel à la fonction `Random.int` de OCaml.

*Question 12*
Profitez de cette fonction `gen_random` pour tester davantage vos différentes approches.

Présentez dans votre rapport les avantages et les inconvénients de chacune des approches que vous avez implantées.

*Question 13*
**[bonus]** Jusqu'à présent, les différentes approches retournent uniquement un entier $s'$. Modifiez vos codes afin de retourner aussi un sous-ensemble dont la somme des éléments vaut $s'$.

Le type de retour des fonctions passe donc de `int` à `int * int list`.