Projet IPF: SUBSET-SUM-OPT

Romain PEREIRA

03/04/2018

Sommaire

Préambule

Ce projet est réalisé dans le cadre de mes études à l'ENSIIE. Rappel de l'énoncé:

> SUBSET-SUM-OPT - Etant donnée un ensemble fini E d'entiers strictements 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 $\sum_{e \in E'} e = s'$.

Ce rapport présente (en pseudo-code), les algorithmes implémentés. Le code OCaml est disponible dans le rendu, dans des fichiers de types 'approche_naive.ml', 'approche_plus_directe.ml' ...

Un Makefile est disponible pour compiler le projet et les tests:

- *make* : compile les fichiers sources avec les tests vers un executable **subset-sum-opt.out**
- *clean* : supprimes les fichiers compilés temporaires (.mlo et .cmo)
- *fclean* : supprimes tous les fichiers compilés (executable et temporaires)

1 Approche naïve

Cette approche consiste à determiner tous les sous-ensembles E' de E, d'effectuer la somme sur tous les E', et de renvoyer la somme la plus proche de s. Cette approche par 'force brute' est lourde.

Si $Card(E) = n$, alors $Card(P(E)) = 2^n$, et $\forall E' \in P(E), 0 <= Card(E') <= n$
Résoudre le problème (génerer ces ensembles et sommer dessus) se fait donc en $\boxed{O(2^n n)}$

1.1 Question 1 : somme sur un ensemble

Algorithm 1: Renvoie la somme des éléments de E

```
1: function SOMME(E')
2:     if E' = ∅ then
3:         return 0
4:     end if
5:     Soit x ∈ E'
6:     return x + Somme(E'\{x})
7: end function
```

Complexité en $\boxed{O(n)}$, où $n = Card(E')$

1.2 Question 2 : génération des parties d'un ensemble

Algorithm 2: Renvoie l'ensemble des parties de E

```
1: function SOUS-ENSEMBLES(E)
2:     if E = ∅ then
3:         return {∅}
4:     end if
5:     Soit x ∈ E
6:     Soit P ← Sous − ensembles(E\{x})
7:     return {P} ∪ {E' ∪ x | E' ∈ P}
8: end function
```

Complexité en $\boxed{O(2^n)}$, où $n = Card(E)$

1.3 Question 3 : résolution de SUBSET_SUM_OPT par force brute

Algorithm 3: Renvoie la réponse au problème SUBSET_SUM_OPT sur (E, s)

```
1: function SUBSET_SUM(E, s)
2:     Soit P ← Sous − ensembles(E)
3:     return max_{E'∈P} {s' | s' = Somme(E')  et  s' ≤ s}
4: end function
```

Complexité en $\boxed{O(2^n n)}$, où $n = Card(E)$

2 Approche plus directe

Dans cette approche, plutot que de calculer l'ensemble des parties de E, on se propose de calculer l'ensemble des sommes atteignables en sommet sur les parties de E.

2.1 Question 4 : sommes atteignables

Algorithm 4: Renvoie l'ensemble des entiers s tels qu'il existe $E' \subseteq E$ vérifiant $\sum_{e \in E'} e = s$

```
1: function GET_ALL_SUMS(E)
2:     if E = ∅ then
3:         return {0}
4:     end if
5:     Soit x ∈ E
6:     S ← Get_all_sums(E\{x})
7:     return S ∪ {x + s | s ∈ S}
8: end function
```

Si l'on suppose:

- $n = Card(E)$
- $m(n) = Card\{sommes \quad atteignables\} = \{s \mid \exists E' \subseteq E \mid \sum_{e \in E'} e = s\} \leq 2^n$
- $X \cup Y$: opération en $O(Card(X) * Card(Y))$

Alors la complexité de cet algorithme est en $\boxed{O(m(n)^2 * n)}$

- n : nombre de récursion
- $m(n)^2$: l'union

2.2 Question 5 : résolution de SUBSET_SUM_OPT

Algorithm 5: Renvoie la réponse au problème SUBSET_SUM_OPT sur (E, s)

```
1: function SUBSET_SUM(E, s)
2:     Soit S ← Get_all_sums(E)
3:     return max{s' ∈ S | 0 ≤ s' ≤ s}
4: end function
```

La complexité de cet algorithme de résolution est donc en $\boxed{O(m(n)^3 * n)}$

3 Approche avec nettoyage

4 Références

[1] 'Writting Cache-Friendly code', Gerson Robboy, Portland State University
http://web.cecs.pdx.edu/ jrb/cs201/lectures/cache.friendly.code.pdf.