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

function SOMME($E' \subset \mathbb{N}$)
  if $E' = \emptyset$ **then**
    return 0
  end if
  Soit $x \in E'$
  return x + `Somme`($E' \backslash \{x\}$)
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

function SOUS-ENSEMBLES($E \subset \mathbb{N}$)
  if $E = \emptyset$ **then**
    return $\{\emptyset\}$
  end if
  Soit $x \in E$
  Soit $P \leftarrow$ `Sous` $-$ `ensembles`($E \backslash \{x\}$)
  return $P \cup \{E' \cup x \mid E' \in P\}$
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)

function SUBSET_SUM($E \subset \mathbb{N}, s \in \mathbb{N}$)
  Soit $P \leftarrow$ `Sous` $-$ `ensembles`(E)
  return $\max_{E' \in P}\{s' \mid s' = \texttt{Somme}(E') \quad et \quad s' \leq s\}$
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 $\sum_{e \in E'} e = s$

```
function GET_ALL_SUMS(E ⊂ ℕ)
    if E = ∅ then
        return {0}
    end if
    Soit x ∈ E
    S ← Get_all_sums(E\{x})
    return S ∪ {x + s | s ∈ S}
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)

```
function SUBSET_SUM(E ⊂ ℕ, s ∈ ℕ)
    Soit S ← Get_all_sums(E)
    return max{s' ∈ S | 0 ≤ s' ≤ s}
end function
```

La complexité de cet algorithme de résolution est donc en $\boxed{O(m(n)^2 * n)}$

3 Approche avec nettoyage

On peut réduire la complexité de l'approche précèdente en réduisant ce que l'on a noté $m(n)$ (le nombre de sommes atteignables. Dans cette approche, on se propose d'ajouter un 'filtre' sur l'algorithme qui génère les sommes.

3.1 Question 6 ; fonction clean_up

Le filtre (l'algorithme 'clean_up') est donnée dans l'énoncé. Cette fonction prends en paramètre:

- $E \subset \mathbb{N}$: l'ensemble a filtré
- $s \in \mathbb{N}$: un entier positif
- $\delta \in \mathbb{R}_+^*$: un réel positif (généralement $\ll 1$)

Cette fonction renvoie un nouvel ensemble $E' \subset E$, tel que:

- $\forall x \in E', x \leq s$
- Si l'on considère la suite croissante $(u_n)_{n \in \mathbb{N}}$ des éléments de E', $\forall n \in \mathbb{N}, u_{n+1} \geq (1+\delta)u_n$.

Autrement dit, tous les entiers strictement supérieurs à n sont supprimés, et si l'on considère 2 entiers consécutifs de l'ensemble trié x et y, tel que $x < y$, ils sont 'proches d'un rapport d'au moins $(1+\delta)$, y est supprimé. Ce filtre supprime donc les éléments de l'ensemble qui sont 'proches' l'un de l'autre (au regard de δ)

3.2 Question 7 : résolution de SUBSET_SUM_OPT

L'algorithme de résolution est également fourni dans l'énoncé. Il est le même que celui de 'l'approche plus directe' 2, sauf que lors du calcul des sommes atteignables filtrés par la fonction 'clean_up'.

Algorithm 6: Renvoie l'ensemble des entiers s tels qu'il existe $E' \subseteq E$ vérifiant $\sum_{e \in E'} e = s$, passant les tests du filtre

function GET_ALL_SUMS_2($E \subset \mathbb{N}$)
    if $E = \emptyset$ **then**
        return $\{0\}$
    end if
    Soit $x \in E$
    $S \leftarrow$ `Get_all_sums_2`$(E \backslash \{x\})$
    return `clean_up`$(S \cup \{x+s \mid s \in S\})$
end function

Algorithm 7: Renvoie la réponse au problème SUBSET_SUM_OPT sur (E, s)

function SUBSET_SUM($E \subset \mathbb{N}, s \in \mathbb{N}$)
    Soit $S \leftarrow$ `Get_all_sums_2`(E)
    return $\max\{s' \in S \mid 0 \leq s' \leq s\}$
end function

La complexité de cet algorithme de résolution est donc en $\boxed{O(m'(n)^2 * n)}$, avec

$$m'(n) = Card\{sommes \quad atteignables \quad filtrs\} < m(n)$$

. **Attention** cependant, pour δ trop grand, on perd l'optimalité du résultat.

4 Références

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