TRAVAUX PRATIQUES XI

SubsetSum et retour sur trace

Ce TP a pour but d'introduire la méthode dite du retour sur trace. Dans tout ce TP, on manipule en théorie des ensembles d'entiers que l'on représente en pratique par des listes OCaml.

On s'intéresse au problème SubsetSum défini comme suit :

- Entrée : $E \subset \mathbb{Z}$ un ensemble fini, et t une cible (« target »)
- Tâche : déterminer s'il existe un sous-ensemble de E dont la somme est t

Par convention, la somme du sous-ensemble vide est 0.

A Recherche exhaustive

Dans un premier temps, on cherche à énumérer toutes les sommes possibles. Pour cela, on applique l'algorithme ci-dessous :

Algorithme 5 : Énumère

Entrées : E (sous forme de liste), et *total* une somme d'éléments de E

```
1 si E est vide alors
2 |   Afficher total
3 sinon
4 |   x ←un élément de E
5 |   E' ← E\{x}
  |    // On affiche la somme en ne prenant pas x puis la somme en prenant x
6 |   Énumère(E', total)
7 |   Énumère(E', total+x)
```

9. Programmer en OCaml l'algorithme Énumère. Appelez-le sur la liste `[1; 2; 3]` avec comme valeur initiale de `total` 0. Que fait-elle ? Expliquer à l'aide d'un schéma des appels récursifs :

10. (Optionnel, difficile) Modifier cette fonction pour qu'au lieu d'afficher des valeurs elle en renvoie la liste.

11. On voudrait maintenant s'inspirer de Énumère pour coder une fonction qui résout SubsetSum. L'idée est la suivante :

- Si la target t est nulle, $\emptyset \subseteq E$ convient : on renvoie donc `true`.
- Sinon, si E est vide : son seul sous-ensemble est $\emptyset$, mais la target est non-nulle, donc aucun sous-ensemble ne convient : on renvoie `false`.
- Sinon : soit $x \in E$ et $E' = E \backslash \{x\}$. Or, un sous-ensemble de E est soit un sous-ensemble de E', soit un sous-ensemble de E' auquel on rajoute x.

  Ainsi, notre code doit essayer ces deux cas : atteindre la target t avec un sous-ensemble de E', ou utiliser x et donc atteindre $t - x$ avec un sous-ensemble de E'.

Programmer une fonction `subsetSum_enumere : int list -> int -> bool` qui applique ce raisonnement. Des exemples d'entrées/sortie ainsi qu'un générateur aléatoire d'instances sont fournis.

B Retour sur trace

12. Normalement, `subsetSum_enumere` fait deux appels récursifs. Modifiez-là pour faire en sorte que si le premier appel renvoie `true` (et donc trouve un sous-ensemble ayant la bonne somme), le second appel n'ait pas lieu.

On suppose maintenant que tous les entiers de l'ensemble sont positifs.

13. Créez `subsetSum_backtrack` qui est une modification de `subsetSum_enumere` pour qu'elle renvoie `false` lorsque que la target est strictement négative.

Comparez les résultats et le temps d'exécution de cette fonction et de la précédente. Commenter.

14. Proposer une façon d'accélérer encore plus.

Félicitation, vous venez de faire votre premier algorithme de retour sur trace ! Un algorithme de retour sur trace (« backtracking » en anglais) est un algorithme qui essaye de construire pas à pas une solution à un problème. Après chaque pas, il essaye de vérifier si le début de solution déjà construit est étendable en une solution complète ou non. Lorsque la construction ne fonctionne pas, il remet en question ses choix précédents et essaye une autre construction.

15. Expliquer comment cette remise en question a lieu dans votre fonction.

16. Modifiez votre fonction pour qu'elle renvoie une `int list option` contenant, s'il existe, le sous-ensemble solution. (Allez voir dans le cours sur les Bonnes Pratiques pour en savoir plus sur le type Option).

17. Résolvez le jour 7 de l'Advent of Code 2024.