TRAVAUX PRATIQUES XI

SubsetSum et retour sur trace

Ce TP a pour but d'introduire la méthode dite du retour sur trace. Vous pouvez le faire en OCaml (conseillé) ou en C (même difficulté).

Dans tout ce TP, on manipule en théorie des ensembles d'entiers que l'on représente en pratique par des listes OCaml ou des tableaux C.

On s'intéresse au problème SubsetSum défini comme suit :

- Entrée : $E \subset \mathbb{Z}$ un ensemble fini, et t une cible (« **t**arget »)
- 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. Implémenter ce pseudo-code comme une fonction `enumere` qui prend en argument l'ensemble d'entiers (et sa longueur ainsi que l'indice actuel si vous programmez en C) et `total`. Appelez-le sur l'ensemble `[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. Modifiez Énumère pour qu'elle résolve SubsetSum. Pour cela, ajoutez un argument : la target `t` , et modifiez légèrement le code là où il faut.

Bravo ! C'est votre première exploration exhaustive codée par vous-même !

12. Re-modifiez la fonction de la question précédente pour enlever l'argument `total`. Appelez la fonction obtenue `subset_enumere` . *Demandez-moi un indice si vous ne voyez pas comment faire*

B Retour sur trace

13. Normalement, `subsetSum_enumere` fait deux appels récursifs. Modifiez-la 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.

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

15. Pour mesurer le temps d'exécution d'un programme en terminal, vous pouvez utiliser `time prgm` . Par exemple, `time ./a.out` . Comparez les résultats et le temps d'exécution de `subsetSum_backtrack` et de `subsetSum_enumere` . Commenter.

Félicitation, vous venez de faire votre premier algorithme de retour sur trace ! 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.

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

17. Allez lire la documentation de `List.sort` (resp. `qsort` en C, n'hésitez pas à me demander de l'aide !) , et triez la liste d'entiers (dans l'ordre croissant ou décroissant, à vous de voir !) avant le retour sur trace. Cela améliore-t-il les performances ? Pourquoi ?

18.
- (OCaml) 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).
- (C) Modifiez votre fonction pour qu'elle recopie une solution à une adresse passée en argument. Le format précis de la solution n'est pas spécifié : à vous de prendre une bonne initiative. (Vous devez pouvoir afficher votre solution une fois le retour sur trace terminé.)

19. (Bonus) Résolvez le jour 7 de l'Advent of Code 2024.