TP Option informatique $n°2$

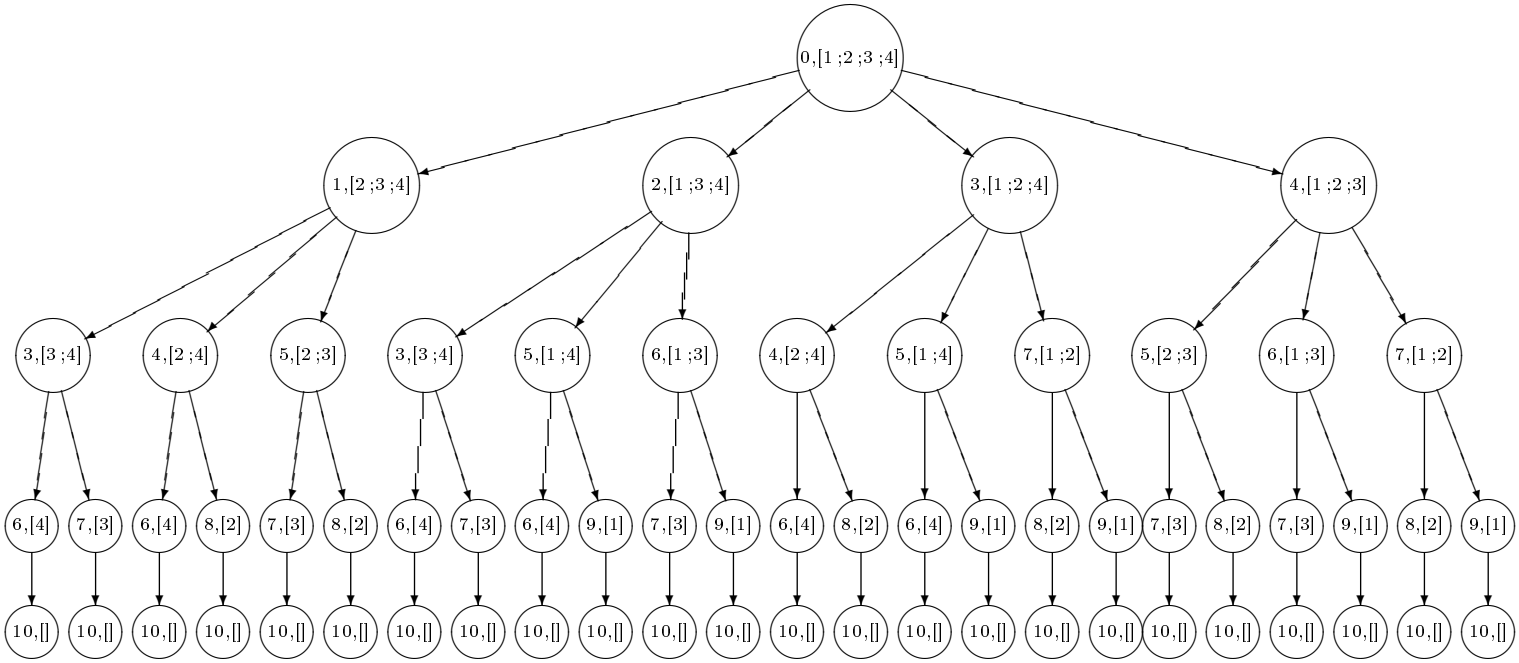
Calculs de somme, Backtracking, programmation dynamique

Le 21/12/2023

Le problème et sa résolution par backtracking

On considère le problème suivant : étant donné une liste `l` d'entiers supposés positifs, et un entier supposé positif `s`, peut-on obtenir `s` comme somme d'un sous-ensemble des éléments de `l` (chaque élément de `l` peut être choisi au plus une fois dans la somme) ?

Ce problème se présente naturellement comme une exploration en profondeur d'un arbre ayant $n!$ branches : à chaque nœud, on décide de prendre un nouvel élément dans la liste que l'on vide petit à petit, jusqu'à obtenir la somme recherchée (pour info, j'ai cru mourir en tapant cet arbre en LaTeX, alors réponds à la question 1).



Exemple pour $s = 6$ et $\ell = [1; 2; 3; 4]$. Quels bouts de branches conviennent ? Quels bouts de branches sont inutiles à explorer ?

1. Compléter la figure précédente en répondant aux questions posées dans sa légende.

2. Plein de branches sont inutiles à explorer. Lesquelles et pourquoi ? On n'essaiera pas d'exploiter la commutativité.

   Décrire un algorithme résolvant le problème par backtracking sans explorer trop de branches inutiles.

3. En utilisant l question précédente, écrire une première fonction `somme_possible : int → int list → bool` telle que `somme_possible s l` renvoie `true` s'il est possible d'obtenir `s` en sommant certains éléments de `l`, et qui renvoie `false` dans le cas contraire.

4. Adapter pour que la fonction retourne la première solution trouvée (une solution est ici une sous-liste convenable), en provoquant une exception s'il n'y en a pas.

5. Adapter pour que la fonction retourne toutes les solutions.

6. Adapter pour que la fonction retourne le nombre de solutions.

7. *Optionnel :* Eh oui, la somme est commutative, tu as bien remarqué qu'on continuait à explorer des branches inutiles parce qu'on n'exploite pas les symétries permises par la commutativité de la somme.

   Propose une optimisation.

Une autre stratégie : la programmation dynamique

Bien que ce problème ne soit pas à proprement parler un problème d'optimisation [1], on peut aussi utiliser des techniques de programmation dynamique pour le résoudre. Notons `n` la longueur de `l`. On propose les sous-problèmes suivants : pour tout $i \leqslant n$ et tout $c \leqslant s$, on note $\mathcal{B}_{i,c}$ un booléen indiquant si on peut obtenir `c` avec les `i` derniers éléments de `l`.

8. Donner les $\mathcal{B}_{i,0}$, les $\mathcal{B}_{0,c}$ pour $c \neq 0$, et une relation de récurrence sur les autres $\mathcal{B}_{i,c}$.

9. Décrire en français un algorithme permettant de résoudre le problème par programmation dynamique.

10. Le coder.

11. Adapter pour qu'il retourne pas seulement le booléen mais aussi une sous-liste permettant de réaliser la somme demandée. On pourra retourner `false,l'` où `l'` est une liste quelconque s'il n'y a pas de solution.

1. C'est un problème d'optimisation au sens où on optimise une quantité qui ne peut prendre que deux valeurs, `true` et `false`. Notons qu'on pourrait généraliser en cherchant à *approcher* s le plus près possible, soit au format *juste prix* (on ne doit pas dépasser s), soit, plus dur, au format *des chiffres et des lettres* (on peut dépasser s), tout ça c'est des exos bonus pour toi, enjoy.