TP n° 2 – Algorithmes gloutons et dynamiques

vie@jill-jenn.net/tp
thomas.h@ens.fr

Un algorithme est dit **glouton** s'il fait à chaque étape un choix localement optimal (la solution qui lui semble la meilleure sans chercher trop loin ni revenir sur ses choix) dans l'espoir d'obtenir une solution globalement optimale (la meilleure possible).

Un algorithme est dit de **programmation dynamique** s'il ramène un problème à plusieurs instances plus petites du même problème qu'il résout récursivement, en mémorisant les résultats pour ne pas avoir à calculer plusieurs fois la même chose.

1 Oups !

Question 1. Yorel Reivax implémente le code suivant, qui crée une matrice 4×2 remplie de 0 et initialise le premier élément à 1.

```
let v = make_vect 4 (make_vect 2 0);;
v.(0).(0) <- 1;;
```

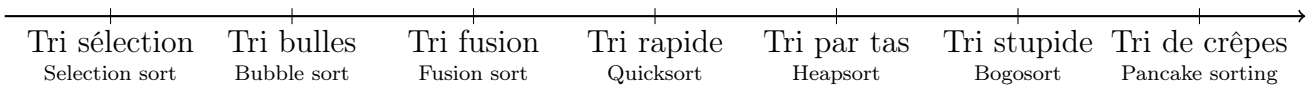
Recopier son code, et afficher `v`. Que se passe-t-il ? Comment y remédier ?

2 Cette machine (ne) rend (pas) la monnaie

L'objectif de cette partie est de programmer un distributeur automatique qui rend la monnaie **en utilisant le moins de pièces possible**. Le *système monétaire* sera représenté par une liste d'entiers distincts. Un montant m sera dit *représentable* dans un système monétaire $\{\ell_1, \ldots, \ell_n\}$ s'il existe des entiers $c_1, \ldots, c_n$ tels que $m = \sum_{i=1}^{n} c_i \ell_i$; en somme, si l'on peut rendre la monnaie.

Un premier algorithme glouton consiste à rendre autant de fois que possible la pièce ayant la plus grande valeur, puis passer à la pièce suivante, et ainsi de suite.

Question 2. Écrire une fonction `tri_selection` qui trie un système monétaire dans l'ordre décroissant, en sélectionnant à chaque étape le plus grand élément de la liste pour le placer en tête. Quelle est sa complexité ? Quelles autres fonctions de tri connaissez-vous ? Placez-vous sur l'échelle suivante.



Question 3. Écrire une fonction `rendu_glouton` telle que `rendu_glouton l m` où `l` est un système monétaire trié renvoie la liste `[c_1;...;c_n]` des nombres de pièces de chaque type à utiliser pour rendre le montant `m`[1]. Dans cette question seulement, on vous garantit que 1 appartient au système monétaire, donc que tous les montants sont représentables.

Question 4. Trouver une valeur de m pour laquelle `rendu_glouton` ne trouve pas un multiensemble de pièces de cardinal minimal dans le système monétaire de 1, 4, 6, 9 kopeks.

1. Par exemple, `rendu_glouton [200;100;50;20;10;5;2;1] 1337` devra renvoyer `[6;1;0;1;1;1;1;0]`.

Question 5. Comparer les assertions suivantes :
- N est représentable dans $\{\ell_1, \ldots, \ell_p\}$
- N est représentable dans $\{\ell_1, \ldots, \ell_{p-1}\}$
- $N - \ell_p$ est représentable dans $\{\ell_1, \ldots, \ell_p\}$

En déduire une fonction `est_representable l m` qui retourne `true` si `m` est représentable dans le système monétaire `l`.

On souhaite s'aider de cette relation de récurrence pour construire un algorithme de programmation dynamique qui manipule une matrice $m_{max} \times n$ où m_{max} est le montant maximal qu'on considère, n le nombre de pièces et $M_{i,j}$ vaut 1 si i est représentable dans $\{\ell_1, \ldots, \ell_j\}$, 0 sinon.

Question 6. Écrire une fonction `rendu_dynamique` qui rend la monnaie de manière optimale (i.e. en utilisant le moins de pièces possible) si c'est possible et retourne une erreur sinon.

Question 7. Adaptez l'une des deux fonctions de rendu pour décomposer un nombre en somme minimale de carrés.

Question 8. Adaptez l'une des deux fonctions de rendu pour décomposer un nombre en somme minimale de termes de la suite de Fibonacci.

3 Emploi du temps

Question 9. Écrire une fonction `planning_nb` qui étant donné une liste d'activités représentées par les horaires de début et de fin de l'activité renvoie une liste maximale d'activités compatibles entre elles.

Question 10. Finalement vous ne voulez plus faire un maximum d'activités mais plutôt faire des activités qui remplissent au mieux le temps dont vous disposez. Écrire une fonction `planning_duree` qui renvoie une liste d'activités de durée totale maximale.

Question 11. Vous êtes multitâche et pouvez faire jusqu'à `n` activités simultanément, modifier les fonctions de planning en conséquence.

4 Faites vos valises

Question 12. Vous partez en voyage et voulez emporter le plus d'objets possible avec vous. Écrire une fonction `remplir_sac_nb` qui étant donné le volume de votre sac et la liste des volumes de vos objets renvoie une liste maximale d'objets qui rentre dans le sac.

Question 13. Tous vos objets n'ayant pas la même importance à vos yeux, vous souhaitez maximiser non pas le nombre d'objets mais leur valeur totale. Écrire une fonction `remplir_sac_val` qui étant donné le volume de votre sac et la liste des volumes et des valeurs de vos objets renvoie une liste d'objets de valeur maximale qui rentre dans le sac.

Question 14. Calculer la complexité de `remplir_sac_val`.

Question 15. Vous disposez de plusieurs sacs et vous souhaitez adapter les fonctions précédentes pour utiliser tous vos sacs. Écrire les fonctions `remplir_sacs_nb` et `remplir_sacs_val` qui prennent en premier argument une liste de volumes de sacs au lieu du volume d'un seul sac.