

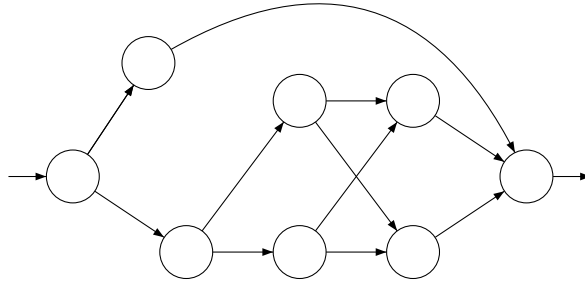
TP 4— GLOUTONNERIES

<http://www.liafa.jussieu.fr/~nath/tp4/tp4.pdf>

Les questions sont les bienvenues et peuvent être envoyées à nathanael.fijalkow@gmail.com.

Ce TP soulève une question importante : la gloutonnerie (en informatique). Un algorithme de recherche d'un objet minimal (chemin, vecteur...) peut être vu comme une méthode automatique qui effectue séquentiellement des choix qui mènent à un objet. Une preuve de correction d'un tel algorithme justifie que ces choix sont "globalement optimaux", c'est-à-dire qu'ils mènent ultimement à un objet minimal. Pourtant, dans certains cas, les choix faits par un algorithme correct ne sont pas "localement optimaux", dans le sens où au moment où le choix est fait, un autre eût été meilleur.

Par exemple, considérons le problème de chercher dans un labyrinthe le plus court chemin entre l'entrée et la sortie. À chaque intersection, on fait un choix. Un choix localement optimal est de se rapprocher (en distance euclidienne par exemple) de la sortie. Cependant, cette heuristique (c'est-à-dire méthode de choix) n'est pas correcte :



Un problème se résout par un algorithme dit "glouton" lorsqu'en effectuant des choix localement optimaux on arrive à un objet minimal. L'objectif de ce quatrième TP est d'analyser et d'implémenter des algorithmes gloutons.

1 RENDU DE MONNAIE

Pour chaque monnaie, on définit sa liste de devises, c'est l'ensemble des valeurs des pièces et billets. Par exemple, la liste de devises de l'euro (si on excepte les centimes) est $\{1, 2, 5, 10, 20, 50, 100, 200, 500\}$. Étant donné une liste de devises et un prix, on veut trouver un moyen de former cette valeur avec les devises en minimisant le nombre de devises.

Par exemple, pour payer 8 euros, la solution optimale est $\{5, 2, 1\}$, et pour payer 14 euros, la solution optimale est $\{10, 2, 2\}$.

▷ **Question 1.** Attention, on compte le nombre de pièces, pas le nombre de pièces différentes, sinon le problème est trivial : expliquer. ◀

▷ **Question 2.** Proposer un algorithme **glouton** qui donne une solution optimale pour l'euro. Cette solution est-elle unique? Écrire une fonction `rendu_monnaie_glouton` qui prend un argument une liste de devises et un prix, et retourne la solution gloutonne. ◀

▷ **Question 3.** Décrire une monnaie qui trompe l'algorithme **glouton** : il existe une valeur pour laquelle la solution trouvée par l'algorithme n'est pas optimale. Est-il possible que la solution trouvée par l'algorithme glouton soit **arbitrairement** plus coûteuse que la solution optimale? ◀

▷ **Question 4.** Donner une monnaie et un prix pour lequel il y a plusieurs solutions optimales. ◀

▷ **Question 5.** Proposer un algorithme qui donne une solution optimale pour une liste de devises quelconque. Écrire une fonction `rendu_monnaie_optimale` qui prend un argument une liste de devises et un prix, et retourne une solution optimale. Optimiser. Encore. ◀

2 ARRÊTS DE BUS

On considère une rue sur laquelle se trouve n maisons, tel que la maison i se trouve au kilomètre $k(i)$. On

veut construire une ligne de bus sur cette rue, et placer les arrêts de bus de manière à ce que pour chaque maison il y ait un arrêt à moins de d km. On cherche évidemment à minimiser le nombre d'arrêts.

On suppose pour éviter des complications techniques qu'il y a un arrêt de bus au kilomètre 0.

▷ **Question 6.** Proposer un algorithme **glouton** résolvant le problème. Écrire une fonction `de` qui résout ce problème. On peut supposer que l'entrée k est donnée par une liste triée par ordre croissant, et d un paramètre de type `int`. ◁

▷ **Question 7. Prouver** que l'algorithme est correct. ◁

3 ORDONNANCEMENT DE TÂCHES

On veut décrire un système dont le but est d'effectuer des tâches parmi n tâches données. La tâche numéro i commence à l'heure $d(i)$ et finit à l'heure $f(i)$.

On peut supposer que l'entrée est donnée par deux listes d et f triées dans l'ordre qui vous arrange.

▷ **Question 8.** Proposer un algorithme **glouton** pour résoudre le problème suivant : on veut maximiser le temps passé à effectuer les tâches (au lieu du nombre de tâches effectuées). Est-il correct? ◁

À partir de maintenant, on veut planifier le nombre maximal de tâches parmi les n .

▷ **Question 9.** Proposer un algorithme **glouton** résolvant le problème. Écrire une fonction `ordonnement` qui résout ce problème. ◁

▷ **Question 10.** Prouver que l'algorithme proposé est correct. Pour cela, considérer F retourné par l'algorithme, et G une solution optimale, puis montrer que F et G ont le même nombre d'éléments. ◁

▷ **Question 11.** Y a-t-il unicité de la solution? Proposer un algorithme **glouton dual**, également optimal. ◁

4 ARBRE COUVRANT MINIMAL

On considère un graphe non-orienté $G = (V, E)$ muni d'une valuation des arêtes $v : E \mapsto \mathbb{R}^+$. On suppose le graphe connexe, c'est-à-dire que pour tout couple de sommets il y a un chemin de l'un vers l'autre.

Le problème de l'arbre couvrant minimal est de trouver un arbre couvrant $T = (V, E')$ où $E' \subseteq E$, de poids minimal, où le poids d'un arbre est la somme des arêtes qu'il contient.

▷ **Question 12.** Considérons $V = V_1 \cup V_2$ une partition des sommets, et $E_{1,2}$ l'ensemble des arêtes (u_1, u_2) du graphe telles que $u_1 \in V_1$ et $u_2 \in V_2$. Considérons ensuite une arête (u_1, u_2) de $E_{1,2}$ de poids minimal : montrer qu'il existe un arbre couvrant minimal de G qui contient cette arête. ◁

▷ **Question 13.** Proposer un algorithme **glouton** pour résoudre le problème, en construisant l'arbre de proche en proche, depuis l'arbre vide. Écrire une fonction `glouton_p` qui résout ce problème. Quelle est sa complexité? ◁

▷ **Question 14.** Proposer un algorithme **glouton** pour résoudre le problème, en construisant l'arbre de proche en proche, mais cette fois en maintenant une forêt couvrante minimale, puis en la fusionnant. Écrire une fonction `glouton_k` qui résout ce problème. Quelle est sa complexité? ◁

▷ **Question 15.** Qui sont P et K ? ◁

5 QUESTIONS DIFFICILES

▷ **Question 16.** Revenons sur le tri rapide. On vous a dit que sa complexité est $O(n \log(n))$ en moyenne, et $O(n^2)$ dans le pire cas. Ces résultats concernent le nombre de comparaisons effectuées, c'est une bonne mesure

de l'efficacité de l'algorithme. Voici une autre mesure pertinente pour quantifier l'efficacité du tri rapide : la hauteur de la pile de récursion.

Rappelons le principe du tri rapide : pour trier $a :: l$, on construit $l_{\leq a}$ et $l_{> a}$, puis on trie récursivement $l_{\leq a}$ et $l_{> a}$, c'est-à-dire que l'on ajoute ces deux listes sur la pile de récursivité. Quelle est la hauteur de pile maximale que l'on peut atteindre ? Donner un exemple de liste où on atteint cette hauteur.

Peut-on modifier un peu l'implémentation du tri rapide pour avoir une pile moins haute? \triangleleft

La hauteur de la pile de récursivité est une mesure de la complexité en **espace** de l'algorithme.

▷ **Question 17.** On s'intéresse à une suite d'entiers définie par u_0 et une fonction $f : \mathbb{N} \rightarrow \mathbb{N}$, où $u_{n+1} = f(u_n)$. De deux choses l'une :

- soit tous les termes de la suite sont distincts,
- soit il y a deux termes égaux.

Dans le second cas, la suite est ultimement périodique. On note T la période, et L l'indice du premier terme qui apparaît deux fois. Donner un algorithme de calcul de L et de T , faisant un nombre linéaire de calculs de f . Donner un algorithme de calcul de u_n pour n quelconque. \triangleleft

▷ **Question 18.** On considère $G = (V, E)$ un graphe. Pour chacun de ces problèmes, décrire un algorithme résolvant le problème, et étudier sa complexité.

- (G est non-orienté) existe-t-il un cycle qui passe par toutes les arêtes exactement une fois (un tel cycle est dit eulérien)?
- (G est orienté) existe-t-il un cycle qui passe par toutes les arêtes exactement une fois?
- (G est non-orienté) existe-t-il un cycle qui passe par tous les sommets exactement une fois (un tel cycle est dit hamiltonien)?
- (G est non-orienté) existe-t-il un chemin qui passe par tous les sommets exactement une fois?

\triangleleft

6 POUET

▷ **Question 19.** Écrire un objet de type `pouet pouet`. \triangleleft