Algorithmes gloutons : comme solution exacte

But

Montrer l'intérêt des algorithmes gloutons, la façon de les construire et de prouver qu'ils fonctionnent. On s'attachera dans cette séance tableau à prouver qu'ils rende une solution optimale à un problème donné.

Algortihme glouton

Un algorithme glouton choisit à chaque étape la meilleure possibilité localement. Ce type d'algorithme est très utilisé pour résoudre des problèmes où l'on veut une réponse rapide, mais pas forcément une réponse optimale. D'un point de vue théorique, ces algorithmes sont extrêmement importants. Il sont, par exemple, en bijection avec la structure de matroïde.

Intérêt :

- donne toujours un résultat
- souvent de complexité faible

Problème :

- ne donne pas forcément le meilleur résultat : une *heuristique*
- pas forcément de solution unique

Pour beaucoup de problèmes d'optimisation, un algorithme glouton est optimal pour une version simplifiée du problème. Comme l'algorithme va vite, on peut recommencer plusieurs fois pour trouver une meilleure solution.

optimalité et glouton

Les problèmes d'optimalité demandent de trouver, parmi un ensemble de solutions possible, une solution minimisant (ou maximisant) un critère. Par exemple :

- pour un ensemble de coûts de constructions possibles d'une voiture, trouver celle qui minimise le coûts tout en maximisant la qualité totale des pièces,
- parmi tous les parcours passant par un ensemble de villes donné, choisir celui qui minimise le nombre de kilomètres parcourus
- maximiser le nombre de films projetés dans un multiplexe de cinéma
- ...

La difficulté de ces problèmes vient du fait que l'on ne peut a priori pas trouver la meilleure solution sans les examiner toutes. Et s'il y a beaucoup de solutions ça peut prendre vraiment beaucoup de temps.

Certains problèmes cependant permettent d'être résolus en construisant petit à petit une solution, sans jamais remettre en cause ses choix. On peut alors souvent trouver très rapidement la meilleure solution possible. On peut également utiliser cette solution construite petit à petit pour trouver une solution approchée à

un problème plus général. Cette classe d'algorithme qui construit itérativement d'une solution est appelée *algorithmes gloutons*.

condition nécessaire et suffisante d'optimalité.

Pour qu'un algorithme glouton **trouve une solution optimale** il faut :

- **initialisation** : montrer qu'il existe une solution optimale contenant le 1er choix de l'algorithme
- **récurrence** : montrer que la première différence entre une solution optimale et la solution de l'algorithme ne peut résulter en une meilleure solution. Pour cela on cherchera une solution optimale dont les choix coïncident le plus longtemps possible avec la solution donnée par notre algorithme et on prouvera qu'elles coïncident jusqu'à la fin.

exercice 1 : le gradient

On suppose que l'on veuille trouver le minimum d'une fonction f dérivable.

1. Décrivez l'algorithme du gradient sous la forme d'un algorithme glouton
2. montrer qu'il peut converger vers la solution
3. montrer qu'il peut ne pas converger vers la solution

exercice 2 : le rendu de pièces

Un système de pièce particulier

Proposez un algorithme glouton permettant de rendre la monnaie d'un achat en un nombre minimum de pièces valant 5, 3 et 1 pokédollar.

Démontrez que votre algorithme est bien optimal.

système de pièces quelconque ?

- donnez une version générale de votre algorithme de rendu de pièce, c'est à dire où l'on a n pièces valant $p_1 < p_2 < \cdots < p_n$.
- Cet algorithme glouton de va pas donner de solution optimale quelque soit le système de pièces, donnez un exemple pour lequel ça ne fonctionne pas.
- montrer que l'algorithme glouton fonctionne pour un système de pièces supercroissant, c'est à dire où $p_i > \sum_{j<i} p_j$ avec $_i$ les valeurs de pièces rangés par ordre croissant.
- donnez un exemple de système de pièces supercroissant.

exercice 3 : allocation de salles de cinéma

Un gérant de cinéma a en sa possession m films caractérisés chacun par un couple (d_i, f_i) où d_i est l'heure de début du film et f_i l'heure de fin. Il se pose 2 problèmes :

- Quel est le nombre maximum de films que je peux voir en une journée ?
- Quel est le nombre minimum de salles à avoir pour visionner tous les films en stock.

  Nota Bene : Si vous êtes d'humeur programmatrice, codez les algorithmes de cet exercice. Si vous séchez, une solution possible vous est donnée dans le corrigé.

voir un maximum de films

Proposez (et prouvez) un algorithme permettant de rendre une liste maximale de films à voir en une journée.

nombre minimum de salles pour placer tous les films en stock

Proposez (et prouvez) un algorithme permettant de rendre le nombre minimum de salles et son organisation permettant de projeter tous les films.

exercice 4 : ordonnancement

Les problèmes d'ordonnancement sont multiples. Certains sont durs d'autres faciles. Mais un algorithme glouton permet de trouver souvent une solution acceptable pour beaucoup d'entres eux et même parfois optimale pour certains problèmes.

Le problème suivant est résoluble par un algorithme glouton : On considère m produits de durée 1 à fabriquer. Si le produit i est réalisée avant la date d_i on peut le vendre pour un prix p_i, sinon il est invendable. Proposez un algorithme permettant de maximiser les profits en considérant que l'on a qu'un seul ouvrier.

ensemble compatible

Un ensemble de produits est dit *compatible* s'il existe un ordonnancement de leur production permettant de tous les vendre (chaque produit est fabriqué avant sa date de péremption).

Montrer qu'un ensemble de produits est compatible si et seulement si l'ordonnancement par date d_i croissante permet de tous les vendre.

algorithme

Montrer que l'algorithme glouton suivant est optimal :

1. on trie les produits par prix décroissant
2. ensemble = {}
3. pour chaque produit x dans cet ordre : on ajoute x à ensemble s'il reste compatible
4. rendre ensemble (qui est un ensemble de profit maximal)

   Nota bene : pour la preuve, on pourra comparer une solution optimale et la solution donnée par notre algorithme en regardant la première différence.