Algorithmes gloutons

But

Montrer l'intérêt des algorithmes gloutons, la façon de les construire et de prouver qu'ils fonctionnent.

Algortihme glouton

Un algorithme glouton choisi 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 extrêmement important, 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 *heurstique*
- 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 données, 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 : 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 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.

exercice 3 : 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 $p_i \geq \sum_{j<i} p_j$ avec $_i$ les valeurs de pièces rangés par ordre croissant.
- donnez un exemple de système de pièces supercroissant.

exercice 4 : 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

voir un maximum de films

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

nombre minium de salles pour placer tous les films en stock

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

exercice 5 : le problème du sac à dos

Le problème du sac à dos est un exemple de problème d'optimisation. Il fait parti des problèmes les plus dur du monde car les solutions n'entretiennent pas de relations les unes avec les autres, il faut a priori toutes les regarder pour trouver la meilleure, et il y en a beaucoup.

Il est possible de modéliser beaucoup de problèmes courant par un problème de sac dos, en particuliers les:

- problème de découpe pour minimiser les chutes
- problème de remplissage (déménagement)

énoncé du problème

On dispose de :

- n objets ayant chacun un poids w_i (*weight*) et une valeur nutritionnelle p_i $(1 \leq i \leq n)$
- d'un sac à dos d'une contenance de W

On veut maximiser la valeur nutritionnelle que l'on peut emporter avec notre sac.

on essaie

Une petite variante crapuleuse du problème du sac à dos où l'on remplace la valeur nutritionnelle par un prix.

On suppose que l'on est un voleur et que l'on peut voler 3 produits :

- produit A, 2kg, 100€
- produit B, 2kg, 10€
- produit C, 3kg, 120€

Selon la valeur du sac à dos quel est la quantité maximale d'argent que le voleur peut se faire ?

sac à doc fractionnel

Si l'on peut prendre qu'une partie des objets (comme pour une poudre ou un liquide), le problème peut être résolu par un algorithme glouton.

Problème :

- entrée :
  - liste de produits décrit par leur masse et le prix total
  - une masse totale transportable
- sortie : une liste une de produit et leur masse qui maximise le prix pour une masse ne dépassant pas la masse transportable

résolvez le problème avec les données précédentes On suppose que l'on peut découper les objet pour obtenir une fraction de leurs valeurs. résolvez le problème pour une capacité de sac de 4kg et de 5kg.

algorithme glouton Proposez un algorithme glouton pour résoudre ce problème et montrer qu'il est optimal.

sac à dos non fractionnel

si on ne peut pas couper ? Donner un exemple où l'algorithme glouton ne donne pas la solution optimale pas si l'on ne peut pas prendre une partie fractionnelle d'un produit.

solution optimale On peut trouver un algorithme optimal pour le problème du sac à dos en remarquant que l'on peut construire une solution optimale avec i objets à partir de solutions optimales à $i-1$ objets.

En effet la solution optimale à i objets pour une capacité W est soit :

- une solution optimale à $i-1$ objets pour une capacité W si on ne prend pas l'objet i,
- une solution optimale à $i-1$ objets pour une capacité $W-w_i$ si on prend l'objet i.

algorithme optimal Ecrivez l'algorithme permettant de résoudre le problème.

Et explicitez pourquoi cet algorithme n'est pas glouton.

Complexité de l'algorithme Quel est la complexité de cet algorithme.

Le millions de dollars Le problème du sac à dos fait partie des problème "les plus dur de l'informatique", or on a un algorithme polynomial pour le résoudre... Soit on vient de démontrer P = NP et on a gagné 1millions de dollars, soit il y a un piège.

Quel est ce piège ?