Algorithmes Gloutons D

Mano Etilé

March 2025

0.1 Algorithmes gloutons

Un algorithme a en général une structure de ce type :

- On considère une état arbitraire de notre système,

-Cette état étant relié à d'autres états, on passe vers un autre état en s'assurant de diminuer notre écart à la propriété que nous voulons montrer.

-On réitère jusqu'à arriver à une situation que l'on juge acceptable

Ainsi, une fois que nous avons exposé l'algorithme, plusieurs problèmes s'opposent à nous :

la **Terminaison :** Notre algorithme finit-il ?

la **Correction :** Notre algorithme fournit-il la bonne réponse ?

Nous devons répondre à cette question pour montrer que notre algorithme fonctionne bien.

0.1.1 Algorithme constructivistes

Voyons sur un premier exemple comment on peut résoudre à l'aide d'un algorithme un problème combinatoire.

Exemple (Putnam, coupe animath):

On dispose de n points rouges et n points bleus dans un plan. Montrer que l'on peut les relier avec n segments, chacun reliant un point rouge et un point bleu, tel que les segments ne s'intersectent jamais.

Voici un algorithme qui marcherait. On **Initialise** en traçant les segments de manière aléatoire. Ensuite, notre à chaque étape, notre algorithme effectue l'**instruction** suivante : Il considère un croisement non désiré, et il décroise les segments.

Comment montrer la terminaison et la correction ? La correction est claire si on a la terminaison. En effet, dans ce cas l'algorithme ne peux plus prendre un croisement non désiré, il n'y a donc pas de croisements.

La terminaison se montre par l'inégalité triangulaire :A chaque étape la somme des longueurs diminuait strictement, et donc à un moment on ne peux plus la diminuer (car il n'y a qu'un nombre fini d'appariement possibles donc seulement un nombre finie de somme de longueurs possibles).

Cela est le forme général d'un algorithme constructiviste : On part d'une solution globale, et on la rafistole petit à petit.

0.1.2 Algorithme de remplissage

Dans certains problèmes, On dispose de plein de cases dans lequels on veut mettre des objets, avec une condition qui nous empêche d'en mettre trop. C'est typiquement souvent le cas des problèmes de coloriage où de pavage.

Pour résoudre ce genre de problème, on peut donner un algorithme qui remplit les cases petits à petits.

exemple (C4 shortlist IMO 2001): Un ensembles de trois entiers positifs $\{x, y, z\}$ est appelé historique ssi $\{z - y, y - x\} = \{1789, 2025\}$. Montrer que l'ensemble de tous les entiers positifs peut-être décrit comme l'union disjointe d'ensembles historiques.

On initialise en ayant pris aucun entier.

L'instruction sera la suivante : On prend le plus petit nombre x qui n'a pas encore été choisi. On pose $z = x + 1789 + 2025$. Si $x + 1789$ n'a pas encore été choisi, on met $\{x, x + 1789, z\}$ comme ensemble. Sinon, on met $\{x, x + 2025, z\}$.

Il faut vérifier que notre algorithme est correct. Premièrement, z n'a pas pu être choisi avant, car les nombres choisit avant sont strictement plus petits que $x + 1789 + 2025$ (car les x précédemment choisit sont strictement plus petits que le x actuel).

Ensuite, si à la fois $x + 1789$ et $x + 2025$ avait déjà été choisit, alors aucun des deux ne peux être le premier de son ensemble. (car alors il serait plus petit que x). Si $x + 2025$ est deuxième de son ensemble, alors $x + 2025 + 2025$ ou $x + 2025 + 1789$ sont déjà dans un ensemble, absurde pour le même argument que pour z.

Donc $x + 2025$ est troisième de son ensemble. Donc cette ensemble est forcément $\{x - 1789, x - 1789 + 2025, x + 2025\}$. Mais c'est absurde, parce que dans ce cas, quand on a choisit $x - 1789$ (il y a plusieurs tours de cela), on a forcément choisi x ensuite ($x = x - 1789 + 1789$). C'est absurde.

Ici, on note que notre algorithme ne peux pas terminer (les ensembles sont infinis), mais les nombres sont bien tous choisit à un moment (à chaque étape le plus petit nombre que l'on choisit augmente forcément de 1).

0.1.3 Algorithme d'ordonnancement

Jusqu'ici, nos algorithme on très souvent pris des éléments pas encore considérés au hasard, ce qui est généralement ce que l'on fais dans un algorithme glouton. Cependant, il est parfois plus utile de considérer les éléments dans un ordre bien défini.

Prenons l'exemple de l'algorithme de tri topologique. Supposons que l'on dispose d'un graphe G simple orienté et acyclique. Il existe une numérotation du graphe tel que si une arête va de u vers v alors u est avant v dans la numérotation.

La preuve est simple : tant que le graphe n'est pas vide, on considère le plus petit sommet sans fils et on lui donne le plus petit numéro encore disponible, puis on le supprime du graphe.

ce type d'algorithme peu paraitre abstrait, mais est assez utile dès que le parcours au hasard n'est pas suffisant.

0.1.4 Problèmes d'Optimalités

Parfois on ne cherche pas seulement à trouver une solution, mais à trouver la meilleure solution. Considèrons le problème du rendu de monnaie :

Comment un boulanger peut-il rendre la monnaie de manière la plus efficace ? On entend par "efficace" que le nombre de pièces qui sont rendues est minimale.

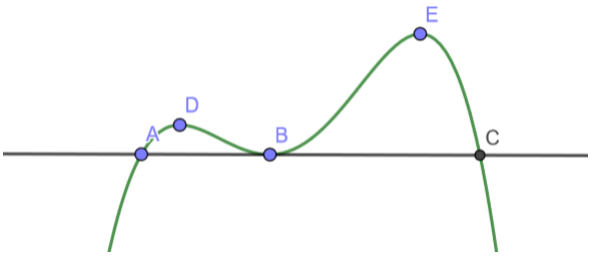
Ainsi, par exemple pour rendre 7 euros, la manière la plus efficace est de rendre 5 et 2. Mais, on pourrait aussi rendre 7 pièces de 1.

Maintenant, comment faire cela de manière algorithmique ? On pourrait dire "bien, à chaque fois que je dois rendre de l'argent, je rends la plus grande pièce possible sans rendre trop, et je recommence".

Pour certain cas, ce rendu n'est pas optimal. Par exemple, supposons que je dispose de pièces de type $(1, 3, 4)$, et que je veux rendre 6. L'algorithme précédent fournit comme rendu $6 = 4 + 1 + 1$ ce qui n'est pas optimal car $6 = 3 + 3$.

Le réel problème des algorithmes gloutons peut être visualisé comme ceci : Supposons que nous cherchons le plus haut sommet du graphe suivant. L'algorithme glouton naturel serait de dire "peut-importe où je suis, si je peux monter, je monte".

Evidemment, si on part de A, et que l'on monte vers D, on n'arrivera jamais au point le plus haut qui est le E



Les algorithmes gloutons sont des algorithmes locaux : on ne se déplacent que si près de nous on peut améliorer notre situation.

On va voir cela avec le problème suivant (JBMO 2019 P4) :

On dispose d'une grille de 5×100 colorié en rouge et en noir. Deux cases sont dites adjacentes si elles ont un côté en commun. On sait que peut importe

la case, il n'y a pas plus de deux cases noirs qui lui sont adjacentes. Trouver le nombre maximal de cases noires.

On pourrait imaginer colorier une bande sur deux, comme ceci :

et appliquer l'algorithme glouton suivant :

Je part d'une certaine situation, et tant que je peux colorier un case noire en plus je le fais.

On peut voir sans problème que le coloriage d'au dessus est final pour notre algorithme glouton. Cependant, Il n'est pas optimal. On colorie ainsi seulement 250 cases, alors que le nombre maximal réel est de 302 (cf Base de combinatoire des structures M)

0.1.5 Exercices

***exercice 1:**

On reprend le problème du rendu de monnaie expliqué dans la dernière section. On suppose que les pièces peuvent prendre toutes les valeurs comprisent entre 1 et r pour un certain r. Montrer que l'algorithme glouton naif fonctionne.

***exercice 2:**

Soit G un graphe, de degré maximal d. Montrer que le nombre chromatique du graphe est au plus $d + 1$.

****exercice 3:**

Pour chaque entier positif n, une banque produit des pièces de valeur $\frac{1}{n}$. Etant donné une collection finie de ces pièces de valeur totale au plus $k - \frac{1}{2}$, prouvez qu'il est plus possible de diviser cette collection en au plus k groupes de pièces, chacun de valeur totale au plus 1. Une valeur peut apparaitre plusieurs fois dans une collection donnée.

****exercice 4:**

Montrer le Théorème de Dirac : Soit G un graphe à au moins 3 sommets, où chaque sommet à un degré d'au moins $\frac{|V|}{2}$. Montrer que G admet un cycle passant par chaque sommet au moins une fois.

*****exercice 5:**

On définit le Polynome chromatique d'un graphe G simple à n sommets comme un polynome P de degré n tel que pour tout k, le nombre de k-coloration de G soit $P(k)$.

Montrer l'existence d'un tel polynome et déterminer son expression pour un arbre.

****exercice 6:**

Montrer que tout entier n s'écrit de manière unique sous la forme $n = \sum_{i=t}^{k} a_i i$ où les a_i sont une suite strictement croissantes avec $a_t \geq t$.

*****exercice 7:**

Soit G un graphe avec strictement plus que $2(k-1)^2$ arêtes. Une k-étoile est un ensemble de k arêtes avec une extrémité commune, et un k-couplage est un ensemble de k arêtes ayant toutes des extrémités différentes.

Montrer que G contient une k-étoile ou un k-couplage.

********exercice 8:**

Un carré nordique est un carré $n \times n$ contenant tout les nombres positifs entre 1 et n^2. Deux cases sont dites adjacentes si elles partagent une arête. chaque case qui est plus petite que toutes ses adjacentes est appelé vallée. Un canyon est défini comme ceci : Un chemin de cases adjacentes, dont les nombres sont croissants et tel que la première case soit une vallée.

Trouver en fonction de n le plus petit nombre de carré montant dans une vallée.

********exercice 9:**

Montrer que pour tout n suffisamment grand, pour tout ensemble de n droites dans le plan en position générale, il est possible de colorier au moins n de ces droites en bleus tel que toute région n'ai pas sa frontière entièrement bleu.