DIU Informatique 2019
Bloc 2 : Algorithmique
— Algorithmes gloutons —

1 Qu'est-ce qu'un algorithme glouton ?

Les algorithmes gloutons sont une familles d'algorithmes qui sont utilisés pour résoudre des problèmes d'optimisation.

Rappel : un problème d'optimisation est un problème dont il existe plusieurs solutions acceptables mais parmi les solutions certaines sont jugées meilleures que d'autres. Par exemple, quand on demande à un système de navigation de nous donner un itinéraire d'un lieu à un autre, il y a typiquement plusieurs itinéraires possibles, mais on va préférer le plus rapide d'entre eux.

L'algorithme est glouton s'il construit une solution au problème étape par étape, et fait à chaque étape le ce qui semble être "localement" le meilleur choix pour résoudre le problème global.

Exemple 1 : des ponts pour l'archipel
Les habitants d'un archipel veulent construire des ponts entre leurs îles de sorte qu'on puisse aller de toute île à une autre en passant par ces ponts. Pour chaque paire d'îles (I_1, I_2), ils calculent le coût $C(I_1, I_2)$ de construire un pont entre elles. Un algorithme glouton pour les habitants est le suivant : à chaque étape, considérer l'ensemble des ponts pas encore construits qui seraient utiles (= qui relierait deux îles pas encore sur la même composante connexe), et construire le moins cher d'entre eux. Répéter jusqu'à n'avoir plus qu'une composante connexe.

Exemple 2 : le ramasseur de balle
Un ramasseur de balle de Roland Garros doit ramasser un ensemble de balles sur le terrain. Bien sûr, il veut faire cela en parcourant la plus petite distance possible. Voici un algorithme glouton pour résoudre le problème : le ramasseur de balles va jusqu'à la balle la plus proche de lui et la ramasse. Il va alors chercher la balle qui est désormais la plus proche de lui, etc, jusqu'à avoir ramassé toutes les balles.

2 Avantages et inconvénients des algorithmes gloutons

L'avantage des algorithmes gloutons est leur simplicité, qui se traduit typiquement par une bonne complexité en temps et en espace. Dans les deux exemples ci-dessus, le nombre de solutions possibles est exponentiel ($n^{(n-2)}$ façons de connecter n îles sans pont inutiles [1] et $n!$ façons de ramasser n balles) et les essayer toutes pour choisir la meilleure prendrait aussi un temps exponentiel. Ici l'algorithme des ponts a une complexité en $O(n^2)$ (on suppose que le coût de construction nous est donné) et celui du ramasseur de balles également, ce qui est vraiment, vraiment, mieux !

1. C'est le théorème de Cayley des arbres couvrants

Mais s'ils sont typiquement d'une bonne complexité en temps/espace, les algorithmes gloutons ne trouvent pas toujours une solution optimale ! Leur simplicité est aussi une forme de "naïveté" : ils font des choix à courte vue.

Supposons que notre ramasseur de balles soit au point $(0, 0)$ et ait n balles devant lui en $(0, 1)$, $(0, 2)$, ..., $(0, n)$ et une balle à sa droite en $(2, 0)$. L'algorithme glouton va le conduire à ramasser les balles devant lui d'abord, puis d'aller chercher la balle à droite, pour une distance totale de de $n + \sqrt{n^2 + 2} = 2n + O(1)$, alors qu'en ramassant la balle de droite au tout début il aurait parcouru une distance $2 + \sqrt{5} + n - 1 = n + O(1)$. L'algorithme glouton rate donc ici la solution optimale.

En revanche, l'algorithme des habitants de l'archipel permet effectivement de trouver une solution optimale au problème de la construction de ponts à moindre coût (exercice : pourquoi ?).

A noter qu'un algorithme glouton (comme beaucoup d'algorithmes d'ailleurs !) peut être non-optimal dans le cas général mais optimal dans des cas particuliers. Un exemple bien connu est le suivant.

Exemple 3 : rendu de monnaie

Etant donnés un ensemble de valeurs de pièces possibles, on cherche à payer une somme d'argent donnée N en utilisant le moins de pièces possibles. L'algorithme glouton "classique" consiste à choisir la pièce dont la valeur est la plus grande parmi celle de valeur $\leq N$, et de recommencer. Ainsi vous allez typiquement payer 2,74 euros avec 2 euros + 50 centimes + 20 centimes + 2 centimes + 2 centimes.

Est-ce optimal ? Dans le cas général, non. Si nous n'avions que des pièces de 1, 6 et 7 euros, et que nous voulions payer 12 euros, l'algorithme glouton donnerait 7 + 1 + 1 + 1 +1 + 1, beaucoup moins bien que 6 + 6. Mais pour les pièces de la vie de tous les jours (1, 2, 5, 10, 20, 50 centimes, 1 et 2 euros), l'algorithme glouton est optimal. Il l'est même toujours en incluant les billets, et le restera si l'on introduit dans le futur des billets de 1000, 2000, 5000 euros etc. [exercice : pourquoi ?]

3 Définir le meilleur choix local

Dans les deux premiers exemples, on remarque que l'on mesure le meilleur choix local à la même aune que la solution globale (dans le premier exemple : la solution optimale est la moins chère, le meilleur choix local est le pont le moins cher ; dans le second exemple, la meilleure solution est la plus courte distance, le meilleur choix local celui qui permet de ramasser une nouvelle balle en parcourant la plus petite distance possible). En un sens, c'est aussi le cas de l'algorithme glouton de rendu de monnaie, puisqu'on cherche à maximiser globalement la quantité (argent donné)/(nombre de pièces), et donc donner la pièce de plus grande valeur est le choix canonique pour un algorithme glouton.

Mais considérons un nouvel exemple.

Exemple 4 : le festival de cinéma

Vous allez assister à un festival de cinéma, et vous venez de recevoir le programme. Pour chaque film on vous donne l'heure de début et l'heure de fin. Votre but est de voir un maximum de films mais certains sont incompatibles entre eux car projetés dans des salles différentes à des horaires se recoupant. Vous devez planifier la liste des films que vous irez voir.

Ici, la mesure de qualité d'une d'une solution globale (nombre de films) ne nous aide pas vraiment à mesurer la qualité d'un choix local. Il faut donc se fier à notre intuition, dont

dépendra notre algorithme. Dans notre exemple du festival de cinéma, on peut imaginer trois algorithmes gloutons :

- Algo 1 : Choisir d'aller voir le film qui débute en premier. Eliminer les films incompatibles avec celui que l'on a choisi. Recommencer jusqu'à ne plus avoir de film à choisir ou éliminer. [Intuition : le meilleur choix local est celui qui commence le premier]
- Algo 2 : Choisir d'aller voir le film le plus court. Eliminer les films incompatibles avec celui que l'on a choisi. Recommencer jusqu'à ne plus avoir de film à choisir ou éliminer. [Intuition : un film plus court va sans doute avoir moins de conflits avec d'autres films]
- Algo 3 : Choisir d'aller voir le film qui termine en premier. Eliminer les films incompatibles avec celui que l'on a choisi. Recommencer jusqu'à ne plus avoir de film à choisir ou éliminer. [Intuition : aller voir le film qui se termine en premier nous donne plus de choix pour aller voir d'autres films après lui.

On voit bien que l'algorithme dépend fortement de notre définition de meilleur choix local. Et en effet, le premier algorithme est catastrophique, le second est mauvais (dans le pire des cas, il nous permettra de voir deux fois moins de films que la solution optimale) et le troisième... est optimal ! [Peut-être le prouver ?]

4 Combiner gloutonnerie avec d'autres idées

Elaborer un algorithme glouton n'exclut pas d'utiliser en même temps d'autres techniques algorithmiques. Notre dernier exemple est un mélange de stratégie gloutonne et de méthode "diviser pour régner".

Exemple 5 : économies sur le carburant

Avec votre voiture, vous devez relier deux villes à distance N kilomètres l'une de l'autre, avec une seule route entre les deux. Avec un plein, vous pouvez parcourir C kilomètres. Sur la route se trouvent des stations-service, chacune étant donnée par un couple (x, p) avec x sa position entre 0 et N, et p le prix du carburant proposé, exprimé en prix/km. Votre réservoir contient initialement une quantité $Q \leq C$ de carburant. A quelles stations acheter son carburant pour faire le parcours en payant le moins possible ? (aucune obligation de faire le plein à chaque arrêt).

Voici un algorithme optimal (preuve omise). S'il n'y a qu'une station service, acheter juste ce qu'il faut de carburant pour aller à notre point d'arrivée. Si la capacité de notre réservoir est insuffisante, l'algorithme retourne une erreur : pas de solution ! S'il y a plus d'une station-service, on trouve la station (x_k, p_k) proposant l'essence la moins chère (p_k minimal). A cette station, on doit essayer de mettre un maximum de carburant. Avec la méthode diviser pour régner, on calcule alors récursivement la façon la moins chère d'aller de 0 à x_k en utilisant les stations (x_i, p_i) avec $0 \leq x_i < x_k$, puis à la station (x_k, p_k) on fait (1) soit le plein si $(N - x_k > C)$ et on rappelle récursivement l'algorithme en partant de du point x_k avec toutes les autres stations décalées de x_k, sinon (2) on remplit juste ce qu'il faut pour finir et on finit le trajet.

5 Synthèse

- Algorithme glouton = algorithme pour un problème d'optimisation où l'on fait à chaque étape le "meilleur choix local" (charge au concepteur de l'algorithme de définir ce qu'est un meilleur choix local)
- Les algorithmes gloutons ne sont pas toujours adaptés : pour certains problèmes ils ne permettent pas de donner une solution optimale.

— Mais quand algorithme glouton permet de trouver une solution optimale à un problème, cet algorithme est souvent un excellent choix. Moralité : si l'on est confronté à un problème d'optimisation qui ne ressemble à aucun problème pour lequel on connaît déjà un bon algorithme, une bonne approche est d'abord de se demander quel pourrait être un algorithme glouton pour ce problème, et s'il est optimal (et si oui, le prouver ! !).
— (Pour aller plus loin). Pour certains problèmes d'optimisation, trouver la solution optimale est si difficile qu'on préfère faire appel à un algorithme glouton (ou autre algorithme efficace) qui donne une solution pas trop loin de l'optimal en un temps raisonnable, plutôt qu'un algorithme donnant une solution optimale en un temps extrêmement long. On parle alors d'heuristique.