Algorithmes gloutons

Le cas du pendu

Lorsqu'on joue au Pendu, comment choisir la prochaine lettre ? On peut la choisir au hasard, mais ce n'est clairement pas la meilleure des idées. On peut également fixer à l'avance un ordre pour les lettres, en suivant, par exemple, l'ordre de fréquence des lettres en Français. Enfin, il est possible de regarder à chaque nouveau tour l'ensemble des mots encore valides et de choisir la lettre apparaissant dans le plus de ces mots.
Cette dernière approche, bien qu'imparfaite, est une des meilleures. Elle permet d'illustrer le principe de base des algorithmes gloutons.

L'exemple du voyageur de commerce

Un représentant de commerce doit partir de Nancy et visiter plusieurs villes avant de revenir à Nancy. L'objectif est de trouver le chemin le plus cours permettant de faire ce circuit. Les distances entre les villes sont donnée ci-contre.

	Nancy	Metz	Paris	Reims	Troyes
Nancy		55	303	188	183
Metz	55		306	176	203
Paris	303	306		142	153
Reims	188	176	142		123
Troyes	183	203	153	123	

Une solution à ce problème consiste à choisir l'ordre dans lequel seront visitées les 4 villes, autres que Nancy. Dans cet exemple il y en a $4 \times 3 \times 2 \times 1 = 24$. Trouver une solution est facile, mais trouver la meilleure est difficile. Ce problème fait même partie des "pires" problèmes d'optimisation. Ceux pour lesquels on ne connaît pas d'algorithme optimal ne nécessitant pas de tester toutes, ou presque, les solutions possibles.
Dans notre cas, on peut remarquer que faire un circuit dans un sens ou dans l'autre revient au même.

Circuit	Longueur
Metz – Paris – Reims – Troyes	809
Metz – Paris – Troyes – Reims	825
Metz – Troyes – Paris – Reims	741
Metz – Troyes – Reims – Paris	826
Metz – Reims – Troyes – Paris	810
Metz – Reims – Paris – Troyes	709
Troyes – Metz – Paris – Reims	1022
Troyes – Metz – Reims – Paris	1007
Troyes – Reims – Metz – Paris	1091
Reims – Metz – Paris – Troyes	1006
Reims – Metz – Troyes – Paris	1023
Reims – Troyes – Metz – Paris	1123

On peut diviser par deux le nombre de solutions à tester. Le tableau ci-dessus présente 12 circuits différents avec leur longueur. Le meilleur mesure 709 km.
Nous avons pu lister tous les circuits possibles, mais leur nombre grandit de façon considérable dès qu'il y a plus de villes. Avec 11 villes, il y a 3,6 millions de circuits et avec 21, il y en a 2,4 milliards de milliards. Même en divisant par deux le nombre de circuits, comme nous l'avons fait, il est impossible de tous les tester.

Une approche possible

Puisqu'il n'est pas raisonnable de vouloir tester toutes les solutions, il faut trouver une approche plus simple. On peut, par exemple, partir de Nancy et aller à la ville la plus proche. On recommence alors en allant à la ville non visitée la plus proche, jusqu'à toutes les avoir parcourues.

Exercice 1 :

1) Dans quel ordre seront visitées les villes en suivant cet algorithme ?
2) Quelle est la longueur du circuit ?

On remarque que la solution obtenue n'est pas la meilleure, mais elle n'est pas si mauvaise. Pour ce problème, l'algorithme glouton n'apporte aucune garantie. Par contre, il est difficile d'avoir un algorithme aussi rapide, ce qui est un vrai avantage compte tenu de la complexité du problème.

Les algorithmes gloutons

Plus formellement, lorsqu'on considère un problème d'optimisation avec un très grand nombre de solutions possibles, on parle d'**algorithme glouton** lorsqu'on procède ainsi :

- On construit la réponse petit à petit.
- À chaque étape, on prend le "meilleur" choix parmi tous ceux disponibles.
- On répète la même stratégie à chaque étape sans jamais revenir en arrière.

Selon les types de problèmes, les algorithmes gloutons peuvent garantir la meilleure des solutions, ou au contraire, en donner de très mauvaises. Ils sont surtout utilisés parce qu'ils sont simples à mettre en place et relativement efficace en temps de calcul. Les algorithmes par **force brute**, eux, consistent à tester toutes les solutions possibles. Ils garantissent la meilleure des solutions, au prix d'un temps de calcul considérable.

Le problème du rendu de monnaie

Voici un autre exemple classique pour les algorithmes gloutons. Vous devez rendre la monnaie après un achat. Comment rendre la monnaie avec le minimum de pièces et de billets ?

Exercice 2 : Vous avez des pièces de 1 et 2 euros, ainsi que des billets de 5, 10 et 20.

1) Combien de billets et pièces faut-il, au minimum, pour faire 9 € ?
2) Combien de billets et pièces faut-il, au minimum, pour faire 37 € ?
3) Décrire l'algorithme glouton permettant de toujours rendre la monnaie avec le moins de billets et pièces possibles.

Avec notre système monétaire, cet algorithme donne toujours la solution optimale. C'est parce que le système est **canonique**. Il est conçu pour que ce soit possible. Par contre, dans le cas général, il n'y a aucune garantie et l'algorithme peut même ne pas trouver de solution. Si on n'a que des pièces de 2 et de 3, et qu'on veut rendre 4, l'algorithme va vouloir rendre 1 pièce de 3 et sera bloqué. Alors qu'il suffisait de rentre 2 pièces de 2.

Le problème du sac à dos

Ce dernier problème est également un classique. Il consiste à emporter la valeur maximale avec un stockage limitée.

Exercice 3 : Avec une capacité limitée à 15 kg, quelle est la valeur maximale qui peut être emportée avec les objets ci-contre ?

Objet	Valeur	Poids
1	10 €	9 kg
2	7 €	12 kg
3	1 €	2 kg
4	3 €	7 kg
5	2 €	5 kg

Pour appliquer l'algorithme glouton il faut calculer la rentabilité de chaque objet (v/p). On parcourt alors les objets dans l'ordre décroissant de leur rentabilité et on prend tous ceux qui rentrent dans le sac.

Exercice 4 : Avec l'algorithme glouton, quelle est la valeur obtenue ?

Cet algorithme ne donne pas toujours la meilleure solution. Il peut même donner la pire. Mais dans le cadre général, la valeur obtenue est assez proche de l'optimum.