4. Culture algorithmique

Bonjour. Dans cette partie sur l'algorithmique, nous avons vu les tâches, la séparation en petites tâches élémentaires d'un algorithme. Ensuite, nous avons vu les variables et la liste exhaustive des instructions élémentaires qui suffisent à faire tous les algorithmes. Nous allons maintenant prendre un petit peu de recul sur l'algorithmique en tant que science pour vous montrer ce qu'on peut faire d'autre, de façon un peu plus générale.

En particulier sur la structure de données, il y a beaucoup de structures de données beaucoup plus complexes qui existent en informatique, dont je vais vous donner une liste, évidemment, non exhaustive. Il y a les listes dont Brice vous a parlé dans la séquence sur l'organisation des données, les listes un peu plus compliquées qui sont doublement enchaînées, où à ce moment-là, il y a des liens de l'un vers le suivant, mais aussi vers le précédent, les piles et les files qui servent notamment à stocker des tâches à effectuer, des tables de hachage pour représenter des ensembles, des arbres et des tas qui sont des structures plus compliquées, mais avec beaucoup de structures sous-jacentes, les graphes dont je vais parler tout à l'heure et les bases de données sur lesquelles vous avez également une séquence.

Je vais vous parler des graphes. Un graphe, c'est exactement le dessin qui est sur ce transparent. Ce sont des boîtes qu'on appelle des nœuds et des arêtes, c'est-à-dire des flèches, d'un nœud vers un autre nœud. Ça sert à représenter beaucoup de choses. Vous pouvez penser à un réseau, un réseau internet, réseau électrique, réseau d'eau. Ça peut représenter des dépendances. J'ai besoin d'avoir fait C pour faire B, ou des relations, A et B se connaissent. Il y a différents types de graphes. Ce graphe que j'ai représenté est avec des flèches, mais il peut aussi y avoir des graphes sans flèches, qui sont des graphes non orientés.

Comment on représente ça par des structures de données ? Il y a plusieurs possibilités. L'une des possibilités, c'est ce qu'on appelle un graphe d'adjacence, on a un booléen qui dit si deux nœuds sont en relation. Quand on lui donne deux nœuds, par exemple E et C, il va répondre vrai parce que A est en relation avec C. Une autre façon de représenter les graphes, c'est une liste de voisins. Le graphe A a deux voisins qui sont B et C. Le graphe C n'a pas de voisin. Maintenant, je vais vous présenter un graphe un petit peu plus compliqué qui représente cette partie du cours. La partie "informatique et ses fondements" est représentée par ce graphe. Chacun des nœuds est une séquence. Ensuite, les flèches représentent les dépendances. Par exemple, I1 et I2, qui sont des séquences de la partie sur le codage binaire des informations, il faut avoir suivi la séquence I1, donc I1. 1 pour suivre la séquence I1. 2. De la même façon, tous les I1 représentent le codage binaire de l'information, les I2 c'est algorithmique, où vous êtes actuellement. Le I3, c'est la programmation et I4, c'est architecture des ordinateurs et réseaux. Le graphe de dépendance est là. Imaginons que vous avez envie de suivre une séquence donnée, comment vous allez savoir quelles sont les séquences à suivre avant ? Je peux vous donner un algorithme que je vais vous dérouler juste après. En entrée, vous avez un graphe de dépendance qui est exactement le dessin du transparent précédent où on stocke les voisins sous forme d'une liste. J'ai un nœud initial n qui est la séquence que j'ai très, très envie de voir, et la sortie, c'est la liste des dépendances de n, c'est-à-dire quelles sont toutes les séquences que je dois regarder de façon à regarder la séquence n. Vous avez un certain nombre de variables. La première variable, c'est "res" qui va être le résultat initialisé à vide. "a_traiter", c'est tout ce que je suis en train, c'est une variable temporaire qui est ce que je dois traiter. J'initialise à vide, ensuite j'ajoute n dedans parce que n, c'est effectivement

l'initialisation, qu'est-ce que j'ai à traiter ? Et bien j'ai à traiter pour l'instant n. Ensuite, je regarde la variable "a_traiter" qui peut être représentée par un certain nombre de structures de données différentes d'ailleurs. Tant que ce n'est pas vide, je sors une valeur de cette liste ou de cette structure de données. Je l'extrais complètement. Je récupère x et "a_traiter", c'est le "a_traiter" précédent privé de cette variable x. J'ajoute x dans le résultat puisque je vais dépendre de x. Ensuite, je rajoute l'ensemble des fils de x à la liste des choses "a_traiter".

Sur un exemple, qu'est-ce que ça donne ? Imaginons que je veux faire la séquence I4. 2, qu'est-ce que je dois suivre avant ? J'ai des couleurs pour représenter les différentes variables. "a_traiter" sera en rouge, le résultat sera en vert, et x qui est celui sur lequel je suis en train de travailler à un certain instant, sera en bleu. Au début, je pars de I4. 2 donc ce que j'ai à traiter, c'est I4. 2. Je regarde I4. 2, devient effectivement le x sur lequel je suis en train de travailler. Ensuite, je regarde tous ses fils et je les mets dans "a_traiter". Maintenant, le I4. 2 qui était le x est passé dans les résultats parce que j'en dépends. J'ai une liste de choses à traiter qui sont effectivement les trois nœuds en rouge. Ensuite, je prends un nœud en rouge, n'importe lequel, la structure de données va m'en sortir un, on s'en fiche. Admettons que je prenne I3. 2. C'est le nouveau x. C'est celui que je suis en train de regarder. Je prends tous ses fils. Il y en a qu'un seul qui est I3. 1, qui est rajouté à "a_traiter". J'ai I3. 2 que j'ai rajouté dans les résultats. Je vais itérer comme ça. A chaque fois, je vais rajouter les fils en rouge et à chaque fois, je vais prendre un rouge et rajouter tous ses fils. A la fin, ça fait ça. Pour suivre I4. 2, il faut que je suive l'ensemble de ces séquences.

Vous avez donc vu un exemple de parcours dans un graphe. Évidemment, l'algorithmique ne se résume pas aux graphes. Il y a beaucoup d'autres structures de données, beaucoup d'autres algorithmes qui existent. Voilà une liste non exhaustive de livres et de références que vous pouvez aller voir. Vous avez deux livres en anglais, de Sedgewick et de Knuth, et ensuite un certain nombre de références en français, une en OCaml, une en Python, une en Java.