TD #0 — Comparaisons

Deux exercices classiques des livres & polycopiés d'algorithmique, presque sans prérequis !

Exercice 1. *Histoire de boulons*

Dans une boite à outils, vous disposez de n écrous de diamètres tous différents et des n boulons correspondants. Malheureusement ceux-ci sont *en vrac*, et vous voulez apparier chaque écrou avec le boulon qui lui correspond. Les différences de diamètre entre les écrous sont tellement minimes qu'il n'est pas possible de déterminer à l'œil nu si un écrou est plus grand qu'un autre, et de même pour les boulons. Par conséquent, le seul type d'opération autorisé consiste à essayer un écrou avec un boulon, ce qui peut amener trois réponses possibles : soit l'écrou est strictement plus large que le boulon, soit il est strictement moins large, soit ils ont exactement le même diamètre (youpi !).

1. Proposez un algorithme (décrit informellement mais précisément) qui apparie chaque boulon avec son écrou, et effectue au plus n^2 comparaisons boulon-écrou dans le *pire cas*.

On cherche maintenant uniquement à trouver le plus petit boulon et l'écrou correspondant.

2. Proposez un algorithme (décrit informellement) qui résout ce problème en au plus $2n - 2$ comparaisons dans le pire cas.

Exercice 2. *Min/max simultané*

On considère un ensemble $\mathcal{S}$ de n entiers, et l'on s'intéresse au problème d'en trouver l'élément maximum et l'élément minimum. La seule opération autorisée sur les éléments de l'ensemble est la comparaison entre deux d'entre eux. Il n'est par exemple pas possible de les comparer à d'autres entiers, de les utiliser dans des expressions arithmétiques, etc. Cette opération est également la seule que l'on comptera pour mesurer le *coût* d'un algorithme, et l'on s'intéressera également uniquement au coût *pire cas*.

1. Proposez un algorithme (décrit informellement mais précisément) qui résout naïvement ce problème.
2. Combien de comparaisons (en fonction de n) sont-elles effectuées dans le pire cas par cet algorithme ?

On souhaite développer un algorithme plus efficace. Une idée pour cela est de commencer par regrouper les éléments de l'ensemble par paires ordonnées.

3. Proposez un algorithme (décrit informellement mais précisément) qui utilise cette approche pour résoudre le problème.
4. Combien de comparaisons (en fonction de n) sont-elles effectuées dans le pire cas par cet algorithme ?

On souhaite maintenant montrer que l'algorithme précédent est *optimal* dans le modèle et avec la métrique considérée. Pour cela on imagine un *adversaire* qui peut choisir le résultat des comparaisons effectuées, et qui cherche à garantir qu'il faudra toujours en effectuer au moins 'un certain nombre. Plus précisément, soit $\mathbb{A}$ un algorithme résolvant correctement le problème, et lors de toute exécution, chaque fois que l'algorithme compare deux éléments qui n'ont pas encore été comparés c'est l'adversaire qui choisit le résultat, de façon à maximiser le nombre de comparaisons effectuées.
Lors d'une exécution de $\mathbb{A}$, on note $\mathcal{N}$ (pour « nouveau ») l'ensemble des éléments n'ayant été comparé à aucun autre, $\mathcal{G}$ (pour « grand ») l'ensemble des éléments ayant été comparé à au moins un autre élément, et ayant toujours été comparé comme « plus grand », $\mathcal{P}$ (pour « petit ») l'ensemble des éléments ayant été comparé à au moins un autre élément, et ayant toujours été comparé comme « plus petit », $\mathcal{M}$ (pour « moyen ») l'ensemble des éléments ayant été comparé à au moins deux autres éléments, dont au moins l'un est plus petit et l'autre plus grand. On note le nombre d'éléments de chacun de ces ensembles (dans l'ordre) par le quadruplet (i, j, k, ℓ).

5. Que vaut $i + j + j + \ell$, à tout point de l'exécution de $\mathbb{A}$?
6. Que vaut (i, j, k, ℓ) au début de l'exécution, et à la fin d'une exécution *toujours correcte* ?
7. Développez une stratégie pour l'adversaire qui démontre l'optimalité de votre second algorithme.

N.B. Cet exercice pourrait éventuellement laisser croire qu'il est assez aisé de prouver des *bornes inférieures* pour le coût des algorithmes, et de là prouver que des algorithmes sont optimaux. Il n'en est rien : réussir à prouver des bornes inférieures est plus l'exception que la règle en informatique.

Exercice 3. *Qu'est-ce qu'un algorithme ?*
Les énoncés des exercices précédents demandaient de décrire *informellement* les algorithmes, mais que serait une description *formelle* ? Pour vous, qu'est-ce qu'un algorithme ? En quoi un algorithme se différencie-t'il d'un programme ? Qu'est-ce qui différencie l'algorithmique de la programmation ?

REMARQUE : Les réponses à ces questions sont loin de faire consensus.