

# PC1 : Diviser pour régner et algorithmes gloutons

École Centrale-Supélec — Christoph Dürr, Nguyễn Kim Thắng

Algorithmique Avancée, 2018

## 1 Cartes bancaires

Vous travaillez pour une banque dans le service des répressions des fraudes. On vous donne  $n$  cartes bancaires qui ont sont suspectés d'être frauduleuses. Chaque carte bancaire comporte une bande magnétique avec des données crypté correspondant à un compte unique. Plusieurs cartes peuvent être associées à un même compte, et dans ce cas on dit qu'elles sont *équivalentes*. Il n'est pas aisé de lire les numéros de compte sur les cartes directement, cependant vous disposez d'une technologie qui permet de tester si deux cartes données sont équivalentes. La question est de déterminer si parmi les  $n$  cartes données, il y a un ensemble d'au moins  $n/2$  cartes mutuellement équivalentes.

Montrez que cette tâche est réalisable avec seulement  $O(n \log n)$  utilisations du testeur d'équivalence.

[Algorithms Design, Chapitre 5, Exercice 3]

## 2 Placement d'antennes

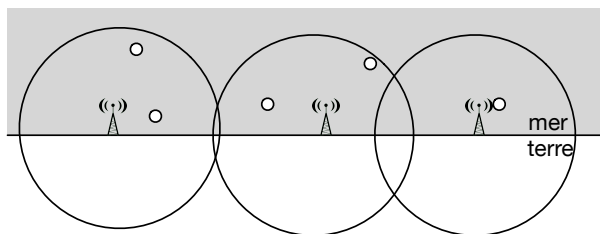


Figure 1: Exemple (non-optimal) de placement d'antennes

Soit une plage toute droite et des îles, aussi petites que des points. Toutes les îles sont à distance au plus  $r$  de la plage. On veut placer un nombre minimum d'antennes le long de la plage pour couvrir chaque île. Une antenne peut couvrir toutes les îles dans un rayon  $r$  donné, qui est le même pour toutes les antennes.

[Programmation efficace, section 5.3]

### 3 Gestion de cache

La mémoire dans les ordinateurs comporte pour plus d'efficacité un mécanisme de cache. La mémoire principale est divisée en  $n$  blocs de même taille. Ces blocs sont appelés des *pages*. La mémoire cache a des temps de réponse plus petite que la mémoire principale, mais pour des raisons de coût a une capacité plus petite que la mémoire principale. Le cache peut contenir  $k$  pages, avec  $k \ll n$ . Les accès mémoire du processeur se traduisent en des requêtes à des pages. Si la page en question est dans le cache, la requête peut être servie immédiatement. Dans le cas échéant on parle de *faute de cache*. Le mécanisme de gestion de cache doit alors choisir une page actuellement dans le cache pour l'échanger avec la page demandée. Cette opération est lente. Le but est de minimiser le nombre de fautes de cache, en prenant des bonnes décisions sur les échanges de pages.

Initialement le cache est vide, et les  $k$  première fautes de caches provoquent un chargement de la nouvelle page plutôt qu'un remplacement. Par conséquence on ne tient pas compte de ces  $k$  premières fautes de cache.

requêtes	2	3	4	2	<u>1</u>	3	<u>7</u>	<u>5</u>	4	1	<u>3</u>	<u>2</u>	4	1
cache	2	2	2		1		1	1			1	1		
	-	3	3		3		7	5			3	2		
	-	-	4		4		4	4			4	4		

Figure 2: Exemple de gestion du cache pour un capacité de  $k = 3$ . Les fautes de cache sont soulignées.

On se place dans une situation simplifiée où toutes les requêtes sont connues en avance. Considérez l'algorithme de gestion de cache suivant (proposé par László Bélády dans les années 60).

FURTHEST-IN-FUTURE (FF): En cas de faute de cache, échanger la page demandée avec une page du cache dont la prochaine requête est la plus loin dans le futur.

On va prouver l'optimalité de l'algorithme FF par induction. Montrez que si le comportement d'un algorithme A est identique avec celui de FF sur les  $i$  premières requêtes, alors il existe un algorithme B qui est identique avec FF sur les  $(i + 1)$  premières requêtes et qui ne provoque pas plus de fautes de cache que l'algorithme A.