DM n°1
Algorithmique 2

Hugo Salou

9 avril 2025

1 Condition nécessaire et suffisante.

Une condition nécessaire pour avoir une attribution « Noël canadien » est la suivante : en notant $j = \arg\max_{i \in [\![1,n]\!]} d_i$,

$$d_j \leq \sum_{i \in [\![1,n]\!] \setminus \{j\}} d_i.$$

Cette condition est nécessaire car, si l'on a $d_j > \sum_{i \neq j} d_i$, alors on ne peut pas construire d'attribution pour la famille (il y a au plus $\sum_{i \neq j} d_i$ échanges de cadeaux, et il en faudrait plus pour en attribuer à toute la famille j).

Pour montrer que c'est une condition suffisante, on propose un algorithme (en l'occurrence l'algorithme 1) qui maintient cette condition comme invariant et qui donne une attribution.

De manière équivalente, la condition s'écrit

$$(\star) : \quad 2 \max_{i \in [\![1,n]\!]} d_i \leq \sum_{i=1}^{n} d_i.$$

2 Algorithme pour les cycles courts.

Pour cet algorithme, on utilise une structure de données un peu particulière : un tableau de piles (où les piles sont implémentées avec des listes chaînées). La structure de donnée ressemble donc à un rideau de perles.

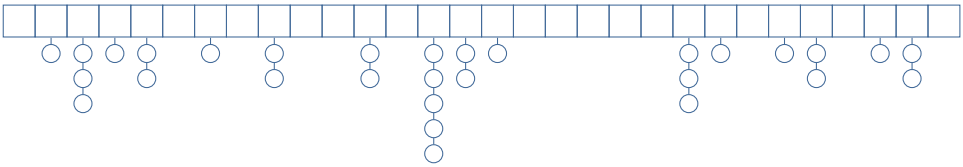

Figure 2.1 | *Structure « rideau de perles »* [1]

L'initialisation d'une telle structure (chaque pile étant initialement vide) se fait en $O(N)$ où N est la taille du tableau que l'on veut allouer. Pour simplifier, on supposera que le tableau est indicé à partir de 0.

En l'occurrence, on utilise une telle structure pour catégoriser les différentes familles en fonction du nombre de personnes respectives. J'appellerai cardinal d'une famille le nombre de personnes dans cette famille. Sur cette structure, on garde deux indices : les indices (donc ici les cardinaux) des deux plus grandes familles considérés.

1. Après le cours sur les tables de hachages, la structure « rideau de perles » est en réalité une simple table de hachage.

Algorithme 1 Attribution avec des 2- ou des 3-cycles

▷ *Initialisation et précalculs*

$N \leftarrow \sum_{i=1}^{n} d_i$
On initialise R un rideau de perles de taille $N+1$.
$m, m' \leftarrow 0, 0$
pour $i = 1$ à n **faire**
    $R[d_i]$.push(i)
    $m, m' \leftarrow$ les deux plus grands éléments de $[m, m', d_i]$ [2]

▷ *Vérification de faisabilité*

si $N < 2m$ **alors**
    retourner Impossible de trouver une attribution

▷ *Cas de N impair*

si N est impair **alors**
    $i \leftarrow R[m]$.pop()
    $j \leftarrow R[m']$.pop()
    On choisit k différent de i et j.
    On choisit p_i une personne non-attribuée de la famille i.
    On choisit p_j une personne non-attribuée de la famille j.
    On choisit p_k une personne non-attribuée de la famille k.
    On attribue $p_i \to p_j \to p_k \to p_i$.
    $R[m-1]$.push(i)
    $R[m'-1]$.push(j)
    si $R[m]$ est vide **alors** $m \leftarrow m-1$
    si $R[m']$ est vide **alors** $m' \leftarrow m'-1$

▷ *Partie gloutonne avec structure de donnée*

tant que $m > 0$ **faire**
    $i \leftarrow R[m]$.pop()
    $j \leftarrow R[m']$.pop()
    On choisit p_i une personne non-attribuée de la famille i.
    On choisit p_j une personne non-attribuée de la famille j.
    On attribue $p_i \rightleftarrows p_j$.
    $R[m-1]$.push(i)
    $R[m'-1]$.push(j)
    si $R[m]$ est vide **alors** $m \leftarrow m-1$
    si $R[m']$ est vide **alors** $m' \leftarrow m'-1$

Afin de simplifier les explications, on suppose que, lorsque quelqu'un a eu une attribution, il disparait de la famille. Ainsi, le cardinal d'une famille i, que l'on notera d'_i, correspond maintenant au nombre de personnes n'ayant pas été attribuées.

Théorème. *L'algorithme 1 calcule en* $\mathrm{O}(N)$*, où* $N = \sum_{i=1}^{n} d_i$*, une attribution « Noël canadien » maximisant le nombre de cycles, si cela est possible, ou dit « impossible » si cela n'est pas possible.*

Preuve. L'algorithme 1 maintient quelques invariants :

1. pour le rideau de perles, on a que $R[i]$ contient des familles de cardinal i ;
2. une fois la ligne 8 passée, on a toujours $\sum_{i=1}^{n} d'_i \geq 2m$;
3. une fois la ligne 18 passée, on a toujours $\sum_{i=1}^{n} d'_i$ pair ;
4. m, m' correspondent aux deux plus grands cardinaux de familles.

On montre les invariants ci-dessus.

1. Initialement, c'est vrai, puis après les ajouts de la partie « initialisation », on a toujours que la famille i est de cardinal d_i, et enfin, on attribue a exactement un élément, donc le cardinal décroit de 1.
2. Après 8, on a bien l'inégalité, et elle est conservée car, on décroit la somme de 2 (car deux attributions), et la valeur de m de un, sauf dans le cas où m reste fixée, ce qui correspond à $m = m'$, et dans ce cas, on a bien $\sum_{i=1}^{n} d_i \geq 2m + 2$ (on a au moins 2 de leste dans l'inégalité), d'où la conservation.
3. On décroît à chaque fois de 2, et c'est vrai après la ligne 18.
4. On remplit bien R en maintenant cette propriété, et on la conserve bien à chaque tour de boucle.

On n'attribue jamais un cadeau à deux personnes de la même famille car on retire la famille du rideau de perles avant de prendre la famille suivante. Les attributions en 2-cycles et un éventuel 3-cycle sont optimales (dans le but de minimiser la taille des cycles), en effet

2. Ce n'est pas un ensemble ici mais bien une liste de 3 éléments. De plus, on choisit bien $m \geq m'$

on devrait avoir des 1-cycles si l'on voulait faire plus court, mais cela n'est pas autorisé.

Et, il n'y a pas de personne n'ayant pas été attribuée, car, à la fin de l'algorithme, $m = \max_{i \in [\![1,n]\!]} d'_i = 0$, donc toutes les familles ont été complètement attribuée.

On en conclut que l'algorithme fournit une attribution lorsque la condition $(\star)$ est vérifiée. Ceci justifie également que la condition $(\star)$ est suffisante.

Avant la boucle de la ligne 21, la complexité est en $\mathrm{O}(N)$. Et, la boucle réalise au plus $N/2$ tours de boucles, car on réalise au plus $(N/2) \times 2$ attributions (on néglige le 3-cycle, car il n'a lieu qu'une fois). D'où une complexité en $\mathrm{O}(N)$. □

3 Algorithme pour l'unique cycle long.

On part de la solution en 2- et 3-cycles donné par l'algorithme précédent, et on se donne un algorithme permettant d'agrandir un cycle en ajoutant un 2-cycle dedans. Ainsi, on commence, soit par un 2-cycle choisi au hasard, soit par l'éventuel 3-cycle créé (si N est impair).

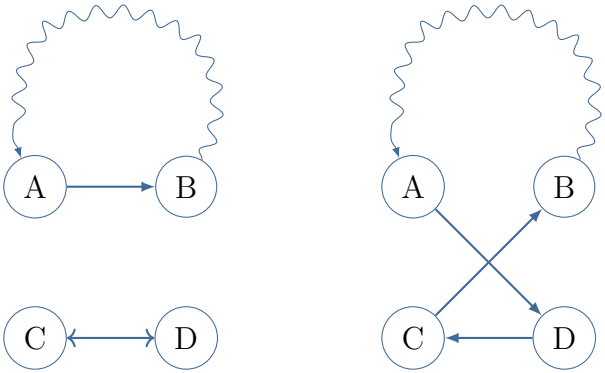

Figure 3.1 | *Agrandir un cycle*

Dans la figure ci-dessus, on distingue les attributions $\longrightarrow$, qui sont celles donné par l'algorithme précédent, et les attributions $\rightarrow$, qui sont celles générées par l'algorithme de construction de cycles.

Il est important de justifier la construction. Comme on considère des 2-cycles, on peut inverser les deux personnes comme on le veut sans casser l'attribution. Ainsi, quitte à permuter C et D, on peut s'assurer que A n'est pas dans la même famille que D, et que C et B ne sont pas dans la même famille non plus. En effet, si avec les deux permutations

on ne peut pas garantir cette condition, c'est que A, B, C et D sont dans la même famille. Mais, c'est impossible car notre algorithme maintient l'invariant que l'on conserve une attribution valide.

Algorithme 2 Calcul d'un cycle de taille maximale

1 : On exécute l'algorithme 1.
2 : On considère C le premier cycle produit par l'algorithme 1. C'est un 2- ou un 3-cycle.

3 : **tant que** il existe un 2-cycle non considéré **faire**
4 : Soit $p_i \rightleftarrows p_j$ un tel 2-cycle.
5 : On l'ajoute dans C comme montré sur la figure précédente (en mettant à jour C par la suite).
▷ *On doit donc faire la disjonction de cas avec les familles de* A, B, C *et* D. *Mais, ça peut se faire en* O(1) *si l'on représente une personne comme un couple numéro de famille, numéro de personne dans la famille.*

Théorème. *L'algorithme 2 renvoie un cycle de longueur maximale (en supposant la condition nécessaire et suffisante précédente) avec une complexité en* $O(N)$ *où* N *est le nombre total de personnes.*

Preuve. On maintient comme invariant que l'on a une attribution valide. Le nombre de cycles dans le graphe diminue strictement, jusqu'à atteindre 1 : on a donc un unique cycle à la fin de l'algorithme. En effet, ceci est assuré car on produit toujours $N/2$ 2-cycles, ou un 3-cycle et $(N-3)/2$ 2-cycles, avec l'algorithme 1. On a donc moins de $N/2$ tours de boucle, et on manipule des cycles comme des listes chaînées, donc le traitement dans la boucle se fait en O(1). (On suppose avoir une file de cycles pour récupérer les cycles produits par l'algorithme 1.) D'où la complexité en $O(N)$. □