Introduction à l'analyse amortie & Application aux Tables Dynamiques

Introduction à l'analyse amortie

Les opérations sur les structures de données ont une complexité en temps qui souvent dépend de l'état courant de la structure de données ce qui rend l'analyse d'une suite de n opérations souvent délicate, voire impossible. De plus, la borne supérieure consistant à faire une analyse dans le pire des cas des opérations élémentaires et à la multiplier par n donne en général des bornes trop laches, uniquement par certaines opérations couteuses ne peuvent arriver qu'à la suite d'un nombre important d'opérations peu couteuses.
Un exemple naturel est l'incrément d'un compteur binaire, où le coût d'un incrément est le nombre de digits qui doivent être modifiés pour passer de n à $n+1$. Ainsi, l'on voit que l'incrément est couteux (coût$= k+1$) quand on passe de $2^k - 1$ à 2^k. Mais cela n'arrive qu'exponentiellement rarement et de plus, à chaque fois que l'on passe d'un nombre pair à un nombre impair, le coût est uniquement de un. On montrera en exercice que le coût amorti de l'incrément est constant alors que le coût réel dans le pire cas est logarithmique en n.
La méthode du potentiel est une façon de réaliser une analyse amortie. C'est une méthode qui permet donc de majorer le coût réel des opérations exécutées sur une structure de données en "amortissant" le coût de chaque opération élémentaire. Cela signifie qu'au lieu de regarder le coût réel d'une opération, nous allons lui associer un coût amorti qui nivelle le coût réel en prenant en compte ses variations au cours de l'évolution de la structure de données.
Rentrons un peu plus précisément dans les détails techniques.
Nous commençons avec une structure de données initiale D_0 sur laquelle sont effectuées n opérations. Pour chaque $i = 1, 2, ..., n$, on note c_i le coût réel de la $i$ème opération et D_i la structure de données qui en résulte. Une fonction potentiel Φ fait correspondre à chaque structure de données D_i un nombre réel positif $\Phi(D_i)$, qui est l'énergie potentiel associée à la structure de données D_i. On assume en plus que $\Phi(D_0) = 0$.
Le coût amorti $\hat{c}_i$ de la $i$ème opération est défini par :

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

Le coût amorti de chaque opération est donc son coût réel plus l'augmentation de potentiel due à l'opération.
Le coût amorti total des n opérations est

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \Phi(D_n) - \Phi(D_0) + \sum_{i=1}^{n} c_i \\ &\geq \sum_{i=1}^{n} c_i\end{aligned}$$

Donc le coût amorti total majore le coût réel des opérations.
Cependant, cette vision ne présente aucun intéret si le coût amorti n'est pas plus simple à calculer que le coût réel. La difficulté de l'analyse amortie réside donc dans le fait de trouver une "bonne" fonction potentiel dans le sens où le coût amorti devient beaucoup

moins dépendant de l'état de la structure de données. Pour cela, il faut garder en tête que la fonction potentiel doit accumuler de l'énergie sur les opérations de faible coût réel pour pouvoir libérer cette énergie sur les structures sur lequelle le coût des opérations est couteuses.
Exercice : Analyse amortie pour l'incrément d'un compteur binaire.

— Trouver une fonction potentiel qui permette d'avoir un coût amorti constant égal à 2. (On pourra utiliser le nombre de 1 dans l'écriture en base 2 de n).

Application aux tables dynamiques

De nombreuses applications utilisent des tables pour stocker des informations sans connaître à l'avance le nombre d'objets qui doivent être stockés (et qui peut évoluer dans le temps). Afin de ne pas utiliser de l'espace inutilement, il est donc nécéssaire de ré-allouer de l'espace quand il vient à en manquer. Mais pour ce faire, tous les objets stockés dans la table initiale doivent être copiés dans la nouvelle table. Cela présente donc un coût en temps important qui va falloir contrôler.
Dans ce cours, nous étudierons ce problème d'extension et de contraction dynamique d'une table. C'est à dire le problème de maintenir une taille de table raisonnable en fonction du nombreux d'éléments à stocker, et cela en ne consumant pas trop de temps.
Commençons donc par définir et analyser le comportement d'une table dynamique dans laquelle on effectue que des insertions. Nous considérerons ensuite le cas plus général où l'on effectue à la fois des insertions et des suppressions.

0.1. **Extension d'une table.** Supposons qu'un espace de stockage soit alloué pour une table sous forme d'un tableau. Une table est remplie lorsque toutes les cases ont été utilisées. Ainsi, quand on essaie d'insérer un élément dans une table pleine, on va "étendre" cette table en allouant de l'espace pour une nouvelle table plus grande, puis copier tous les éléments de l'ancienne table vers la nouvelle.
Une solution consiste à allouer une nouvelle table contenant deux fois plus de cases que l'ancienne. Si seules des insertions sont effectuées, la table est toujours au moins remplie à moitié et donc la quantité d'espace gaspillée ne dépasse jamais la moitié de l'espace total alloué pour la table. Dans le pseudo code suivant, la variable $nom[T]$ contient le nombre d'éléments dans la table et la variable $taille[T]$ est le nombre total de cases dans la table. Au départ, la table est vide : $nom[T] = taille[T] = 0$.

Algorithme 1 : Insérer

Entrées : T un tableau et x un élément à ajouter.
Sorties : Rien, mais le tableau T est modifié.

```
Insérer(T, x)
si taille[T] = 0 alors
    Créer une table avec une case mémoire allouée;
    taille[T] := 1;
si nom[T] = taille[T] alors
    Créer une nouvelle table avec 2 × taille[T] cases mémoire allouées;
    Copier tous les éléments de T dans cette nouvelle table;
    Liberer l'espace mémoire de l'ancienne table et renommer la nouvelle table T;
    taille[T] := 2 × taille[T];
Rajouter x dans T;
nom[T] := nom[T] + 1;
```

On va utiliser la méthode du potentiel pour faire une analyse amortie du coût d'une séquence de n opérations Insérer. Pour cela, il nous faut définir une fonction potentiel Φ

qui va pouvoir "amortir" le coût de l'extension. Il faut donc qu'elle permette d'accumuler de l'énergie de sorte que juste avec une extension, cette énergie soit de l'ordre du coût réel de l'extension. La fonction suivante est définie à cette effet :

$$\Phi(T) = 2 \times nom[T] - taille[T]$$

En effet, juste avant une extension, on a $nom[T] = taille[T]$, et donc $\Phi(T) = nom[T]$ comme souhaité.
Comme la table est toujours au moins à moitié pleine, $\Phi(T)$ est positif. C'est donc bien une fonction potentiel et la somme des coûts amortis de n opérations Insérer majore la somme des coûts réels.
Pour analyser le coût amorti de la $i$ème opération Insérer, on appelle nom_i le nombre d'éléments présents dans la table après la $i$ème opération, $taille_i$ la taille totale de la table après la $i$ème opération et Φ_i le potentiel après la $i$ème opération. Au départ, on a $nom_0 = 0$, $taille_0 = 0$ et $\Phi_0 = 0$.
Si la $i$ème opération Insérer ne provoque pas d'extension, alors on a $taille_i = taille_{i-1}$ et le coût amorti de l'opération est

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \times nom_i - taille_i) - (2 \times nom_{i-1} - taille_{i-1}) \\
&= 1 + (2 \times nom_i - taille_i) - (2\ (nom_i - 1) - taille_i) \\
&= 3.
\end{aligned}$$

Si la $i$ème opération déclenche une extension, alors on a $taille_i = 2 \times taille_{i-1}$ et $taille_{i-1} = nom_{i-1} = nom_i - 1$, ce qui implique que $taille_i = 2 \times (nom_i - 1)$. Le coût amorti de l'opération est donc :

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= nom_i + (2 \times nom_i - taille_i) - (2 \times nom_{i-1} - taille_{i-1}) \\
&= nom_i + (2 \times nom_i - 2\ (nom_i - 1)) - (2\ (nom_i - 1) - (nom_i - 1)) \\
&= nom_i + 2 - (nom_i - 1) = 3.
\end{aligned}$$

En conclusion le coût amorti élémentaire d'Insérer est toujours égal à 3 et donc le coût réel total de n opérations Insérer est majoré par $3n$.

0.2. **Extension et contraction d'une table.** Dès le moment où l'on s'autorise à retirer des éléments dans la table, il paraît intéressant de contracter la table lorsqu'elle ne contient plus assez d'éléments. La contraction d'une table est donc une sorte de réciproque de la notion d'extension : lorsque le nombre d'éléments de la table est trop petit, on alloue un espace mémoire pour nouvelle table plus petite et on copie les éléments de l'ancienne table vers cette nouvelle table.
On va proposer une stratégie permettant, premièrement de maintenir un coût amorti constant des opérations Insérer et Supprimer et secondo de s'assurer que la table est toujours au moins à 1/4 remplie. Pour ce faire, on continue à doubler la taille de la table quand on essaie d'inserer un élément dans une table pleine, et on va diminuer de moitié la taille de la table quand une suppression fait passer le remplissage de la table sous les 1/4.

Algorithme 2 : Supprimer

Entrées : T un tableau et x un élément de T à supprimer.
Sorties : Rien, mais le tableau T est modifié.

```
Supprimer(T, x)
si taille[T] = 1 alors
    liberer la mémoire;
    taille[T] := 0;
si nom[T] = taille[T]/4 alors
    Créer une nouvelle table avec taille[T]/2 cases mémoire allouées;
    Copier tous les éléments de T sauf x dans cette nouvelle table;
    Liberer l'espace mémoire de l'ancienne table et renommer la nouvelle table T;
    taille[T] := taille[T]/2;
sinon
    Supprimer x dans T;
nom[T] := nom[T] − 1;
```

On admettra dans l'analyse que si le nombre d'éléments de la table retombe à 0, l'espace de stockage de la table est libéré. Autrement dit, si $nom[T] = 0$, alors $taille[T] = 0$. Soit $\alpha(T) = nom[T]/taille[T]$ le facteur de remplissage d'une table non vide T. Pour une table vide, on pose $\alpha[T] = 1$, de sorte que l'on a toujours $nom[T] = \alpha(T) \times taille[T]$, que la table soit vide ou non.
On va utiliser la méthode du potentiel pour analyser le coût d'une séquence de n opérations Inserer et Supprimer. Nous prenons comme fonction potentiel :

$$\Phi(T) = \begin{cases} 2 \times nom[T] - taille[T] & \text{si } \alpha(T) \geq 1/2 \\ taille[T]/2 - nom[T] & \text{si } \alpha(T) < 1/2 \end{cases}$$

Observez que le potentiel d'une table vide est 0 et que le potentiel n'est jamais négatif. Donc, le coût amorti total d'une séquence d'opération majore le coût réel de la séquence. Pour analyser une suite de n opérations Insérer ou Supprimer, on note comme précédement c_i le coût réel de la $i$ème opération et $\hat{c}_i$ son coût amorti, et $\alpha_i = nom_i/taille_i$ le facteur de remplissage de la table après la $i$ème opération. Au départ, $nom_0 = 0, taille_0 = 0, \alpha_0 = 1$ et $\Phi_0 = 0$
Commençons par le cas où la $i$ème opération est Insérer. Quand $\alpha_{i-1} \geq 1/2$, l'analyse est la même que celle concernant l'extension de table vue précédement. Le coût amorti $\hat{c}_i$ de l'opération faut 3. Si $\alpha_{i-1} < 1/2$, l'opération ne peut pas provoquer l'extension de la table puisque l'extension n'a lieu que lorsque $\alpha_{i-1} = 1$. Si l'on a aussi $\alpha_i < 1/2$, le coût amorti de la $i$ème opération est

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (taille_i/2 - nom_i) - (taille_{i-1}/2 - nom_{i-1}) \\ &= 1 + (taille_i/2 - nom_i) - (taille_i/2 - nom_i + 1) \\ &= 0. \end{aligned}$$

Si $\alpha_{i-1} < 1/2$ mais $\alpha_i \geq 1/2$, alors

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \times nom_i - taille_i) - (taille_{i-1}/2 - nom_{i-1}) \\
&= 3 \times nom_{i-1} - \frac{3}{2} taille_{i-1} + 3 \\
&= 3\alpha_{i-1} taille_{i-1} - \frac{3}{2} taille_{i-1} + 3 \\
&< \frac{3}{2} taille_{i-1} - \frac{3}{2} taille_{i-1} + 3 \\
&= 3.
\end{aligned}
$$

Le coût amorti d'une opération Insérer est donc au plus égal à 3.
Considérons maintenant le cas où la $i$ème opération est Supprimer. Dans ce cas, $nom_i = nom_{i-1} - 1$. Si $\alpha_{i-1} < 1/2$, il faut savoir si l'opération provoque ou non une contraction. Si ce n'est pas le cas, $taille_i = taille_{i-1}$ et le coût amorti de l'opération est

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (taille_i/2 - nom_i) - (taille_{i-1}/2 - nom_i - 1) \\
&= 1 + (taille_i/2 - nom_i) - (taille_i/2 - (nom_i + 1)) \\
&= 2.
\end{aligned}
$$

Si $\alpha_{i-1} < 1/2$ et que la $i$ème opération provoque une contraction, le coût réel de l'opération est $c_i = nom_i - 1$, car on déplace $nom_i - 1$ éléments (Notez que l'on ne prend pas en compte dans le coût la possible recherche de x dans la table. Mais si celui-ci est supposé constant, cela n'affecte pas fondamentalement notre analyse dans le sens où le coût amorti reste constant.
On a $taille_i/2 = taille_{i-1}/4 = nom_{i-1} = nom_i + 1$ et le coût amorti de l'opération est

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (nom_i - 1) + (taille_i/2 - nom_i) - (taille_{i-1}/2 - nom_{i-1}) \\
&= (nom_i - 1) + ((nom_i + 1) - nom_i) - ((2 \times nom_i + 2) - (nom_i + 1)) \\
&= -1.
\end{aligned}
$$

Quand la $i$ème opération est Supprimer et que $\alpha_{i-1} > 1/2$, le coût amorti est

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \times nom_i - taille_i) - (2 \times nom_{i-1} - taille_{i-1}) \\
&= 1 + (2 \times (nom_{i-1} - 1)) - taille_{i-1} - (2 \times nom_{i-1} - taille_{i-1}) \\
&= -1.
\end{aligned}
$$

Enfin, quand la $i$ème opération est Supprimer et que $\alpha_{i-1} = 1/2$, le coût amorti est

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (taille_i/2 - nom_i) - (2 \times nom_{i-1} - taille_{i-1}) \\
&= 1 + (2nom_{i-1}/2 - nom_{i-1} - 1) \\
&= 2.
\end{aligned}
$$

Nous avons donc montrer que dans tous les cas le coût amorti des deux opérations plus petit que 3.
En conclusion, le coût réel total de n opérations Insérer et Supprimer est majoré par $3n$.