Complexité amortie

Source de l'image d'en-tête XKCD #1667

I Introduction

Dans le cadre de l'étude des structures de données, il est fréquent de considérer non pas la complexité dans le pire des cas d'une opération mais celle d'une succession d'opérations divisée par le nombre d'opérations effectuées. Ainsi, on peut très bien avoir une opération ponctuellement plus coûteuse que les autres, mais en procédant ainsi on lisse le surcoût sur l'ensemble des opérations. On parle alors de **complexité amortie**.

> **Remarque** Cette notion ne masque pas le fait qu'une opération puisse prendre ponctuellement plus de temps. Dans des contextes temps réel où il est important de maitriser pleinement les complexités, il est peu judicieux d'utiliser de telles complexités. Par exemple, dans une visualisation en 3D, pour maintenir un débit constant d'images par secondes, chaque image doit prendre un temps similaire. Se reposer sur une structure de donnée ayant une faible complexité amortie mais une complexité dans le pire des cas importante, c'est risquer d'avoir des saccades avec une image qui prendrait plus de temps pour être calculée.

On va reprendre ici deux exemples simples déjà traité par ailleurs.

I.1 Implémentation d'une file avec deux piles

On considère une file réalisée avec deux piles `in` et `out` (voir le chapitre Piles et files).
Comme on l'a vu, on enfile en $O(1)$ en empilant sur la pile `in` et on défile :

- soit en $O(1)$ en dépilant sur `out` si elle est non vide
- soit en $O(n)$ en déversant `in` dans `out` puis en se ramenant au cas précédent.

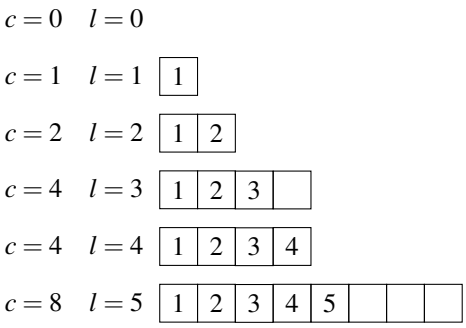
Ainsi, en pire cas, on est en $O(n)$ pour l'opération *défiler* mais la bascule permettra de faire ensuite cette même opération en $O(1)$.

I.2 Les tableaux dynamiques

Les tableaux dynamiques sont une structure de données de haut niveau pour implémenter le type abstrait des listes. La différence principale entre cette structure de données et celle des tableaux de C est qu'on peut ajouter et supprimer des éléments.

Un tableau dynamique d'entiers est un triplet `(t,c,l)` où `t` est un tableau de taille `c`, appelée la capacité du tableau dynamique, et `l` est un autre entier représentant la longueur logique du tableau. A tout moment $c \geq l$. Dans `t` il y a ainsi $c - l$ cases déjà allouées qui permettent de rajouter un élément en temps constant. Quand $c = l$, on alloue une nouvelle zone mémoire, souvent de taille $2c$, on déplace le tableau `t` dans cette zone et on a donc pour cet ajout prévu un certain nombre de cases d'avance.

La figure suivante présente une succession d'ajouts :



Remarque Une implémentation de ces opérations est proposée dans le programme suivant :

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  int *t;
  size_t c;
  size_t l;
} tableau_dynamique ;

tableau_dynamique tableau_dynamique_creer()
{
  tableau_dynamique d;

  d.t = NULL;
  d.c = 0;
  d.l = 0;

  return d;
}

void tableau_dynamique_ajout(tableau_dynamique *d, int x)
{
  if (d->c == d->l) {
    if (d->c == 0)
    {
      d->c = 1;
      d->t = malloc(sizeof(int));
    }
    else
    {
      d->c = 2 * d->c;
      d->t = realloc(d->t, d->c * sizeof(int));
    }
  }
  d->t[d->l] = x;
  d->l++;
}

void tableau_dynamique_print(tableau_dynamique d)
{
  printf("c=%d\tl=%d\t|", d.c, d.l);
  for (size_t i = 0; i < d.l; i++) {
    printf("%d|", d.t[i]);
  }
  for (size_t i = d.l; i < d.c; i++) {
    printf(" |");
  }
  printf("\n");
}

int main(void) {
  tableau_dynamique d = tableau_dynamique_creer();
  tableau_dynamique_print(d);
  for (int i = 1; i < 6; i++) {
    tableau_dynamique_ajout(&d,i);
    tableau_dynamique_print(d);
  }
}
```

C

Ce programme, une fois exécuté produit la sortie suivante qui permet de retrouver exactement le comportement attendu :

```
c=0 l=0 |
c=1 l=1 |1|
c=2 l=2 |1|2|
c=4 l=3 |1|2|3| |
c=4 l=4 |1|2|3|4|
c=8 l=5 |1|2|3|4|5| | | |
```
c

Avec une analyse en pire cas, l'ajout d'un élément peut entrainer un doublement de taille et une recopie, donc coûter $O(c)$ opérations où c est la capacité avant doublement. Cependant, on constate que ce doublement permettra plusieurs ajouts ensuite en temps constant, ainsi, le pire cas ne semble pas le plus pertinent.

II Techniques de calcul

II.1 Cadre

On considère une structure de données, pour simplifier initialemetn vide, et une sucessions $o_1, ..., o_n$ d'opérations **valides** sur cette structure de donnée. En reprenant l'exemple précédent, cela peut être une succession d'ajouts ou de suppressions dans une file mais sans jamais essayer de retirer des éléments à une file vide.

On considère de plus que pour chaque opération o_i, on connait le coût temporel c_i en terme d'opérations élémentaires. La question que l'on se pose est d'estimer $\frac{1}{n}\sum_{i=1}^{n} c_i$, c'est-à-dire le coût moyen de chaque opération. L'objectif est de montrer que ce coût moyen est $O(f(n))$, avec assez souvent $O(1)$, c'est-à-dire que $\sum_{i=1}^{n} c_i = O(nf(n))$.

II.2 Calcul naïf

Une approche näive consiste à calculer directement la somme des coûts. Tout le problème ici est qu'on ne connait pas les opérations et l'ordre dans lequel elles ont été effectuées.

II.2.i Exemple de la file avec deux piles

On va commencer par calculer c_i pour les différentes opérations :

- S'il s'agit d'un ajout sur `in` ou d'un dépilement sur `out`, on a $c_i = 1$
- Si l'opération est un dépilement nécessitant un renversement, on a $c_i = 2p + 1$ où p est la taille de `in`.

Le problème du calcul naïf ici est de connaitre l'ordre des opérations. On peut regrouper les opérations $o_1, \dots, o_n$ par paquet : ajouts, suppression avec renversement, ajouts ou suppressions sans reversement, suppression avec renversement... On va noter p_i la taille de `in` avant le ième renversement. On a donc p_{i+1} ajouts entre le ième et le (i+1)ème renversement. Ainsi, on a effectué $p_i - 1$ suppressions sans renversements entre les deux.

Comme cela correspond au pire cas, on peut supposer que o_n est une suppression avec renversement.

On obtient alors

$$\begin{aligned}\sum_{i=1}^{n} c_i &= p_1 + (2p_1 + 1) + p_2 + (p_1 - 1) + (2p_2 + 1) \\ &\quad + \cdots + p_r + (p_{r-1} - 1) + (2p_r + 1) \\ &\leq 4(p_1 + p_2 + \cdots + p_r)\end{aligned}$$

Pour conclure, il est nécessaire de relier le nombre d'opérations aux p_i. Or, on a

$$\begin{aligned}n &= p_1 + 1 + p_2 + p_1 - 1 + \cdots + p_r + p_{r-1} - 1 + 1 \\ &= 2p_1 + 2p_2 + \cdots + 2p_{r-1} + p_r \geq p_1 + \cdots + p_r\end{aligned}$$

Ainsi $\frac{1}{n}\sum_{i=1}^{n} c_i \leq 4$.

Remarque Cette analyse est assez pénible à mettre en oeuvre car elle demande de compter avec précision ce qui se passe.

II.2.ii Exemple du tableau dynamique

Ici, les opérations sont toutes des ajouts d'éléments, mais certains provoquent une recopie tandis que d'autres sont en temps constant. Cependant, on a la même distribution des coûts, la suite des c_i est toujours $1, 2, 3, 1, 5, 1, 1, 1, 9, 1, 1$

Plus précisément, $c_i = \begin{cases} 2^p + 1 & \text{si } i = 2^p + 1 \\ 1 & \text{sinon} \end{cases}$.

Donc si $2^{p-1} < n \leq 2^p$ on a $\sum_{i=1}^{n} c_i = n + \sum_{k=1}^{p-1} 2^k = n + 2^p \leq 3n$.

On en déduit que $\frac{1}{n}\sum_{i=1}^{n} c_i \leq 3$ c'est-à-dire que c'est un $O(1)$.

II.2.iii Améliorer la méthode naïve

Une amélioration de la méthode naïve va consister à considérer non pas le coût réel c_i de l'opération o_i mais un coût amorti c'_i tel qu'à la fin des opérations on ait $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} c'_i$ où cette dernière somme est facile à majorer.

On fera souvent en sorte que la majoration soit valide tout au long des opérations, c'est-à-dire que

$$\forall k \in [\![1, n]\!], \sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} c'_i$$

II.3 Méthode du banquier

On va imaginer maintenant que les opérations coûtent un certain nombre de crédits, comme si l'ordinateur marchait avec des pièces de monnaie.

Des opérations vont alors accumuler des crédits dans une réserve et d'autres opérations vont consommer ces crédits. L'idée est de payer un peu plus pour chaque opération peu coûteues afin de couvrir les coûts des opérations coûteuses.

Précisément, on considère pour chaque opération o_i :

- une quantité a_i de crédits qui seront accumulés dans la réserve
- et une quantité d_i de crédits qu'il faudra dépenser.

On pose alors $c'_i = c_i + a_i - d_i$. On s'arrange pour faire en sorte que $c'_i \leq Af(n)$ où A est une constante. Dans le cas d'une complexité constante amortie, on s'assurera que $c'_i \leq A$.

On va maintenir un invariant sur les crédits qui est que

$$\forall k \in [\![1, n]\!], \sum_{i=1}^{k} a_i \geq \sum_{i=1}^{k} d_i$$

Cela signifie qu'à tout moment la réserve de crédit est positive ou nulle : on ne dépense pas de crédits n'ayant pas été accumulés.

On a alors directement la majoration voulu de $\sum_i c_i$: soit $k \in [\![1, n]\!]$,

$$\sum_{i=1}^{k} c'_i = \sum_{i=1}^{k} c_i + \left(\sum_{i=1}^{k} a_i - \sum_{i=1}^{k} d_i\right) \geq \sum_{i=1}^{k} c_i$$

II.3.i Exemple de la file avec deux piles

Pour chaque élément enfilé on va allouer deux crédits correspondant au futur dépilement de `in` puis empilement sur `out` qui auront lieu lors du renversement.

On pose donc pour chaque opération o_i, $c'_i = c_i + a_i - d_i$ où :

- $a_i = 2$ et $d_i = 0$ si l'opération est un ajout
- $a_i = d_i = 0$ si l'opération est un défilement ne nécessitant pas de renversement
- $a_i = 0$ et $d_i = 2p$ où p est la taille de `in` dans le cas d'un défilement provoquant un renversement.

On a donc $\sum_{i=1}^{k} a_i = 2A$ où A est le nombre d'ajouts effectués par les k premières opérations. Si on a effectué

des renversements successifs de $p_1, p_2, \ldots, p_r$ éléments dans `in`, on a

$$\begin{aligned}\sum_{i=1}^{k} d_i &= 2p_1 + \cdots + 2p_r \\ &= 2(p_1 + \cdots + p_r) \\ &\leq 2A \\ &\leq \sum_{i=1}^{k} a_i\end{aligned}$$

La majoration venant du fait qu'un élément n'est ajouté qu'une seule fois dans `in` et donc que la somme $p_1 + \cdots + p_r$ des longueurs de `in` au cours des opérations ne peut dépasser le nombre A d'éléments empilés.
On a ainsi vérifié l'invariant sur les crédits et on a donc la majoration $\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} c'_i$.
Pour conclure, il suffit de calculer c'_i selon les cas :

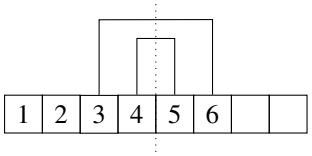
- Si l'opération est un ajout, on a $c'_i = 1 + 2 - 0 = 3$.
- Si l'opération est une suppression sans renversement, on a $c'_i = 1$
- Si l'opération est une suppression avec renversement, on a $c'_i = 2p + 1 + 0 - 2p = 1$

Dans tous les cas, on a donc $c'_i \leq 3$.
On en déduit la majoration $\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} c'_i \leq 3n$ qui est plus fine que la majoration obtenue précédemment.

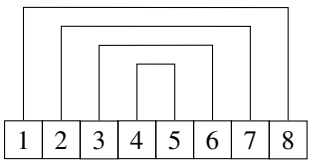
II.3.ii Exemple du tableau dynamique

On va considérer qu'on alloue 2 crédits pour chaque opération d'ajout d'un élément :

- Un crédit pour la première recopie de cet élément
- Un crédit pour copier l'élément qui lui est symétrique par rapport à la demi-capacité :



Ainsi, juste avant un doublement de capacité, on dispose d'un crédit pour chaque recopie :



Plus précisément, on définit pour chaque opération o_i : $c'_i = c_i + a_i - d_i$ où $a_i = \begin{cases} 1 & \text{si } i = 1 \\ 2 & \text{sinon} \end{cases}$ et $d_i = \begin{cases} 2^p & \text{si } i = 2^p + 1 \\ 0 & \text{sinon} \end{cases}$.
On a donc $c'_i = 3$ dans tous les cas.

Théorème II.1 $\forall n \in \mathbb{N}^*, \sum_{i=1}^{n} a_i \geq \sum_{i=1}^{n} d_i$.

■ **Preuve**
On a $\sum_{i=1}^{n} a_i = 2n - 1$ et si $2^p < n \leq 2^{p+1}$,

$$\begin{aligned}\sum_{i=1}^{n} d_i &= d_2 + d_3 + d_5 + \cdots + d_{2^p+1} \\ &= 1 + 2 + 2^2 + \cdots + 2^p \\ &= 2^{p+1} - 1 \leq 2n - 1\end{aligned}$$

L'inégalité est bien vérifiée.

■

Ainsi,

$$3n = \sum_{i=1}^{n} c'_i = \sum_{i=1}^{n} c_i + \left(\sum_{i=1}^{n} a_i - \sum_{i=1}^{n} d_i \right) \geq \sum_{i=1}^{n} c_i$$

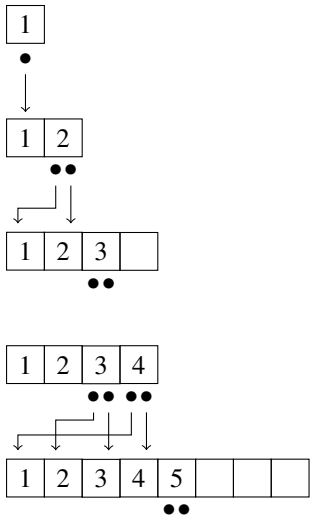
On retombe sur une complexité amortie en $O(1)$.

II.3.iii Présentation de la méthode

La présentation qui vient d'être faiter de cette méthode est assez lourde bien que précise. En général, on se contente de présenter informellement la distribution de crédits en s'assurant qu'on a couvert les opérations coûteuses.

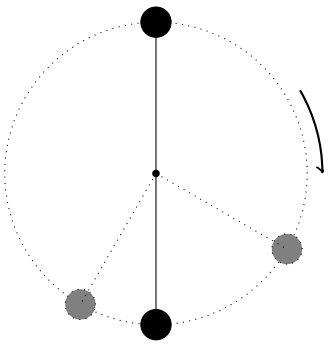
Une manière de bien s'en assurer est de *décorer* les structures de données en rendant les crédits apparents.

Si on reprend les tableaux dynamiques, on va rajouter sous une case le nombre de crédits dont elle dispose. Si on reprend la séquence d'ajouts précédente on pourra la représenter ainsi :



II.4 Méthode du potentiel

L'idée de la méthode du potentiel est de définir une fonction sur Φ qui associe à chaque valeur de la structure de données un nombre réel. Ce réel correspond à la notion physique de potentiel : si on considère un pendule simple dans sa trajectoire son énergie potentielle est maximale au sommet quand sa vitesse, et donc son énergie cinétique est nulle. À l'inverse son énergie potentielle est nulle au point le plus bas et sa vitesse maximale.



L'idée ici est que le potentiel va correspondre aux gains de temps obtenus par des opérations peu coûteuses. Chaque opération va ainsi faire évoluer le potentiel. On peut prendre l'image d'un empilement de tuiles de bois, comme des Kapla, chaque ajout d'une tuile de bois augmente le potentiel jusqu'à un éventuel effondrement. On retrouve ici la dualité entre les opérations rapides et les opérations coûteuses liées à un basculement.

Précisement, on va noter t_i la structure obtenue à l'issue de l'opération o_i. On considère que t_0 est la structure vide et $\Phi(t_0) = 0$. On pose $c'_i = c_i + \Phi(t_i) - \Phi(t_{i-1})$ ce qui correspond à considérer la différence de potentiel.

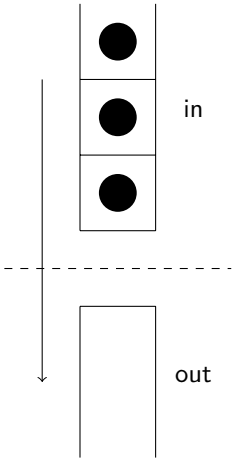
On a donc

$$\sum_{i=1}^{k} c'_i = \sum_{i=1}^{k} c_i + \sum_{i=1}^{k} \Phi(t_i) - \Phi(t_{i-1}) = \sum_{i=1}^{k} c_i + \Phi(t_k)$$

Pour conclure par une majoration de $\sum_{i=1}^{k} c_i$, il suffit de s'assurer que le potentiel ne soit jamais négatif.

II.4.i Exemple de la file avec deux piles

On va poser $\Phi(t) = 2|\texttt{in}|$ où $|\texttt{in}|$ est la taille de la pile d'entrée. Pour reprendre le parallèle physique, on peut s'imaginer que la pile `in` est au dessus de la pile `out` et que les éléments menacent de s'y deverser. Ce qui permet d'avoir l'image du potentiel comme pour le pendule au sommet de sa trajectoire.



On a directement la majoration car $\Phi(t) \geq 0$.

Pour conclure, il faut calculer c'_i dans chaque cas :

- Si l'opération est un ajout, `in` augmente d'un élément et donc $c'_i = c_i + 2 = 3$
- Si l'opération est une suppression sans renversement, le potentiel ne change pas et $c'_i = 1$
- Si l'opération est une suppression avec renversement et que `in` contient p éléments, on a $c'_i = 2p+1+0-2p = 1$.

On retombe exactement sur le calcul précédent avec la méthodde du banquier.

II.4.ii Exemple du tableau dynamique

On pose ici $\Phi(t) = 2l(t) - c(t)$ où $l(t)$ est la longueur de t et $c(t)$ est sa capacité.

Ici, ce potentiel se comprend mieux en écrivant $2(l(t) - c(t)/2)$ car $l(t) - c(t)/2$ correspond aux éléments ajoutés après doublement.

Comme $c(t) \geq l(t) \geq \frac{c(t)}{2}$ on a $\Phi(t) \geq 0$ ce qui valide la majoration.

On calcule c'_i dans les deux cas d'ajout :

- Si c'est un ajout sans doublement, alors $l(t_i) = 1 + l(t_{i-1}), c(t_i) = c(t_{i-1})$ et ainsi

$$\begin{aligned} c'_i &= c_i + \Phi(t_i) - \Phi(t_{i-1}) \\ &= 1 + 2l(t_{i-1}) - c(t_{i-1}) - 2l(t_{i-1}) + c(t_{i-1}) = 1 \end{aligned}$$

- Si c'est un ajout avec doublement de capacité, alors $l(t_{i-1}) = c(t_{i-1}) = A$, $l(t_i) = 1 + l(t_{i-1}) = 1 + A$, $c(t_i) = 2c(t_{i-1}) = 2A$ et $c_i = A + 1$. On a donc :

$$\begin{aligned} c'_i &= c_i + \Phi(t_i) - \Phi(t_{i-1}) \\ &= A + 1 + 2(A + 1) - 2A - 2A + A \\ &= 3 \end{aligned}$$

Ainsi, $3n \geq \sum_{i=1}^{n} c'_i \geq \sum_{i=1}^{n} c_i$. On retombe sur la majoration voulue.

II.4.iii Comparatif des méthodes

On remarque que la méthode du potentiel et la méthode du banquier sont très proches. En fait, on peut montrer qu'elles sont équivalentes. La différence entre les deux est que la méthode du banquier demande de regarder avec précision les opérations pour lesquelles il va falloir allouer des crédits alors que pour la méthode du potentiel il suffit de trouver son expression. Cependant, trouver l'expression du potentiel est parfois plus complexe que de définir les crédits.

On cherchera donc la méthode la mieux adaptée aux problèmes considérés. Dans l'exemple de la file, le potentiel est très facilement défini, dans le cas du tableau dynamique, la méthode du banquier peut paraitre plus naturelle.