LYCÉE LECONTE DE LISLE

Complexité amortie

Vincent Picard

Complexité amortie : principe général

- La complexité d'un algorithme est le plus souvent donnée sous forme de complexité **dans le pire cas**.
- Dans certains cas, lorsqu'on exécute de N opérations de pire temps d'exécution $T_{\max}$, le pire cas arrive si rarement qu'il est brutal de majorer le temps d'exécution total par $NT_{\max}$.
- Le but d'une analyse de **complexité amortie** est de donner une mesure plus réaliste du temps d'exécution réel lorsqu'on exécute une **suite d'opérations**.

Exemple classique : une file avec 2 piles

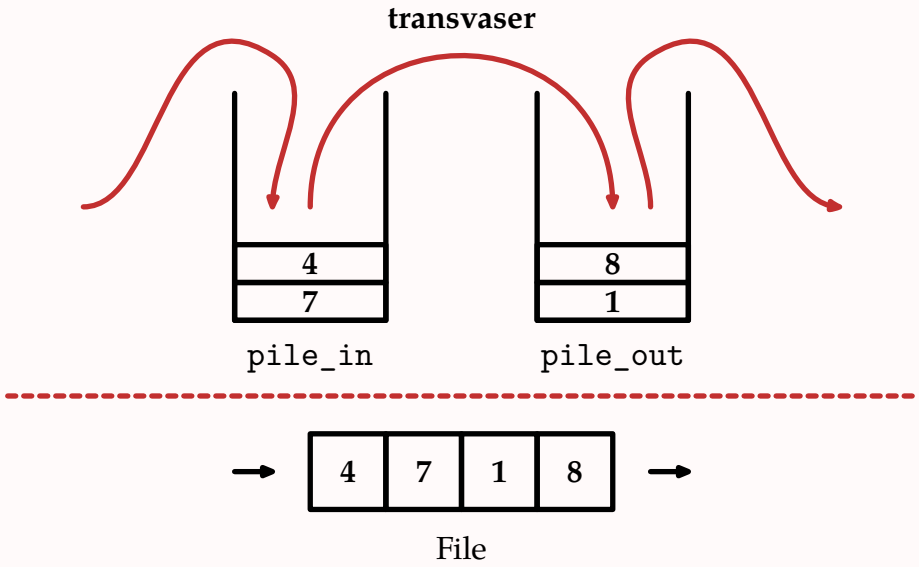
On souhaite implémenter la structure de file abstraite modifiable :

```
type 'a file;;

creer : unit -> 'a file
inserer : 'a -> 'a file ->  unit
extraire : 'a file -> 'a
est_vide : 'a file -> bool
```

On propose l'implémentation concrète suivante à l'aide de **deux piles** :

- Une pile `pile_in` dans laquelle on place les éléments insérés dans la file
- Une pile `pile_out` depuis laquelle on sort les éléments de la file
- Si `pile_out` est vide lors de `extraire` il faudra transvaser tous les éléments de `pile_in` dans `pile_out`



Voici le code correspondant à cette implémentation :

```
type 'a file = {pile_in : 'a stack; pile_out : 'a stack};;

let creer () = {pile_in = Stack.create (); pile_out = Stack.create ()};;

let inserer x file = Stack.push x file.pile_in;;

let extraire file =
    if Stack.is_empty file.pile_out then begin
        while not(Stack.is_empty file.pile_in) do
            let x = Stack.pop file.pile_in in
            Stack.push x file.pile_out
        done
    end;
    Stack.pop file.pile_out
;;

let est_vide file = Stack.is_empty file.pile_in
    && Stack.is_empty file.pile_out;;
```

Comlexité pire cas

Soit n la taille de la file, la complexité **dans le pire cas** des opérations est

Opération	Complexité pire cas
`creer`	$O(1)$
`inserer`	$O(1)$
`extraire`	$O(n)$
`est_vide`	$O(1)$

Toutefois, cette analyse est brutale car le pire cas de `extraire` n'arrive que rarement : une analyse de complexité plus réaliste peut être obtenue grâce à une **analyse amortie**.

1
Méthode par agrégation

Méthode par agrégation

- C'est la méthode d'analyse amortie la plus simple, mais elle est assez limitée.
- **Principe :** On estime le temps du pire cas d'exécution d'une série de N opérations bien précise et on divise le résultat par N.
- **Exemple :** Dans notre file si on effectue N insertions puis ensuite N extractions :
  - Les N insertions coûtent : $O(1)$
  - La première extraction provoque le tranvasement donc coûte $O(N)$
  - Les $N-1$ extractions suivantes sont en $O(1)$
  - La **complexité amortie** de chacune des opérations est donc :

$$\frac{N \times O(1) + O(N) + (N-1) \times O(1)}{2N} = \boxed{O(1)}$$

  - Tout se passe comme si chaque opération était de coût constant.
- **Inconvénient :** ce n'est pas général, par exemple, que se passe-t-il si on alterne des insertions et des extractions ?

2
Méthode comptable

Méthode comptable

- On suppose qu'on dispose d'une **banque** de temps intialisée à 0.
- On va attribuer à chaque opération un **coût fictif** calculé ainsi :
  - Il contient le coût réel des opérations
  - On peut ajouter du "temps" à la banque, ce qui est comptabilisé dans le coût fictif
  - On peut lui déduire du "temps" prélevé à la banque
  - On ne peut bien sûr pas prélever plus que disponible à la banque
  - Autrement dit : $c' = c + \text{placement} - \text{retrait}$
- Le coût fictif total obtenu sur une suite de N opérations quelconque est alors toujours supérieur au coût réel.

Méthode comptable : application à la file

- Dans notre cas, on va se servir de la banque pour emmagasiner à l'avance le temps que va coûter chaque élément dans son futur transvasement.
- Coûts fictifs utilisés :
  - `creer` : $c'_{\text{creer}} = O(1)$ c'est le coût réel
  - `inserer` : $c'_{\text{inserer}} = \underbrace{O(1)}_{\text{coût réel}} + \underbrace{2}_{\text{placement}} = O(1)$ on place une unité dans la banque en anticipation du transvasement de l'élément inséré.
  - `extraire` :
    - Dans le bon cas le coût réel est $c'_{\text{extraire}} = O(1)$
    - Dans le mauvais cas il faut transvaser chaque élément ce qui coûte $2n$ opérations, mais on ne les compte pas : on les prélève à la banque : $c'_{\text{extraire}} = \underbrace{O(1) + 2n}_{\text{coût réel}} - 2n = O(1)$
  - `est_vide` : $c'_{\text{est\_vide}} = O(1)$ coût réel

Méthode comptable : résultat

- On obtient les complexités amorties suivantes :

Opération	Complexité amortie
`creer`	$O(1)$
`inserer`	$O(1)$
`extraire`	$O(1)$
`est_vide`	$O(1)$

- Si on réalise N opérations sur la file, quel que soit la séquence, alors le coût total de la séquence peut être estimé en utilisant les complexités amorties plutôt que majorer chacune des opérations par son pire cas.

- **Remarque importante :** il n'y a pas **une** analyse amortie mais **des** analyses amorties possibles : un autre choix pour les coûts fictifs peut conduire à des complexités différentes (mais toujours supérieures à la complexité réelle).

3
Méthode par potentiel

Méthode par potentiel

- La méthode par potentiel est un raffinement de la méthode comptable dans lequel la *banque* est concrétisée sous forme d'une fonction potentielle associée à la structure de données manipulée.
- Une **fonction potentielle** est une fonction qui à tout état S de la structure de donnée associe une valeur $\phi(S) \in \mathbb{R}$: cela correspond à la *banque*. Ainsi, la valeur de la banque peut être connu simplement en regardant l'état de la structure de données.
- On associe ensuite des coups fictifs c' à chacune opération ainsi :

$$c' = \underbrace{c}_{\text{coût réel}} + \underbrace{\phi(S_{\text{après}}) - \phi(S_{\text{avant}})}_{\text{variation du potentiel}}$$

- Si la variation de potentielle est > 0 cela signifie qu'on a emmagasiné du temps de réserve en banque.
- Si la variation de potentielle est < 0 une partie du temps réel est amortie en prélevant dans la banque.

Correction de la méthode

La correction de la méthode du potentiel est donnée par le théorème suivant :

Si la fonction de potentiel vérifie :

i. $\phi(S_0) = 0$ où S_0 est l'état de la structure à la création

ii. $\forall S, \quad \phi(S) \geq 0$

alors pour toute suite d'opérations $(op_i)_{i \in [\![1,N]\!]}$ de coûts réels (c_i) et de coûts fictifs (c'_i), on a :

$$\sum_{i=1}^{N} c'_i \quad \geq \quad \sum_{i=1}^{N} c_i$$

Preuve :

Cette proposition découle d'un calcul de somme télescopique :

$$\begin{aligned}\sum_{i=1}^{N} c'_i &= \sum_{i=1}^{N} (c_i + \phi(S_i) - \phi(S_{i-1})) \\ &= \sum_{i=1}^{N} c_i + \sum_{i=1}^{N} (\phi(S_i) - \phi(S_{i-1})) \\ &= \sum_{i=1}^{N} c_i + \underbrace{\phi(S_N)}_{\geq 0} - \underbrace{\phi(S_0)}_{=0}\end{aligned}$$

dans ce calcul S_i est l'état de la structure après l'opération op_i.

Méthode du potentiel : application à la file

- On définit le potentiel par : $\phi(\{\texttt{pile\_in}, \texttt{pile\_out}\}) = |\texttt{pile\_in}|$
- Le potentiel vérifie bien les deux propriétés : il est initialement nul et il est toujours positif.
- On calcule les coûts fictifs avec le potentiel
  - `creer` : $c'_{\text{creer}} = O(1)$ (initialisation effectuée une fois seulement)
  - `inserer` : $c'_{\text{inserer}} = \underbrace{O(1)}_{\text{coût réel}} + \underbrace{1}_{\text{variation potentiel}} = O(1)$ la longueur de `pile_in` augmente de 1.
  - `extraire` :
    - ⋆ Dans le bon cas le coût réel est $c'_{\text{extraire}} = O(1) + 0$ car `pile_in` ne change pas.

- ⋆ Dans le mauvais cas il faut transvaser chaque élément ce qui coûte n opérations supplémentaires : $c'_{\text{extraire}} = \underbrace{O(1) + n}_{\text{coût réel}} \underbrace{-n}_{\text{variation potentiel}} = O(1)$ car `file_in` passe de n éléments à 0.

- `est_vide` : $c'_{\text{est\_vide}} = O(1) + 0 = O(1)$ car `pile_in` ne change pas.

Méthode du potentiel : résultat

- On obtient les complexités amorties suivantes :

Opération	Complexité amortie
`creer`	$O(1)$
`inserer`	$O(1)$
`extraire`	$O(1)$
`est_vide`	$O(1)$

- Si on réalise N opérations sur la file, quel que soit la séquence, alors le coût total de la séquence peut être estimé en utilisant les complexités amorties plutôt que majorer chacune des opérations par son pire cas.

- **Remarque importante :** encore une fois il n'y a pas **une** analyse amortie mais **des** analyses amorties possibles : un autre choix de fonction potentielle conduit à des coûts fictifs différents

4
Exemple : un compteur binaire

Un compteur binaire

On considère un compteur binaire de p bits, cette structure de donnée possède l'opération `incr` qui incremente sa valeur d'une unité. Cette opération changera un certain nombre de bits du compteur. Par exemple :

1001000100111	4647
1001000101000	4648

On peut représenter ce compteur par un tableau de bits de longueur p.

```
void incr(int n[P]) {
    int i = P - 1;
    while (i >= 0 && n[i] == 1) {
        n[i] = 0;
        i -= 1;
    }
    if (i >= 0) { n[i] = 1; }
}
```

Compteur binaire : questions

1. Estimer la complexité pire cas de la fonction `incr` en fonction de `P`.
2. En déduire la complexité de N appels à `incr` depuis $n = 0$, cette mesure est-elle réaliste ?
3. On pose $\phi(n) = \alpha|n|_1$ où α est une constante positive et $|n|_1$ est le nombre de bits à 1 de n. Vérifier que ϕ est une fonction potentielle ayant les bonnes propriétés.
4. Calculer le coût fictif de l'opération `incr`.
5. Choisir α afin d'obtenir une complexité amortie intéressante pour `incr`.