# Analyse amortie

Chapitre 17, Livre ***T. Cormen, C. Leiserson, R. Rivest et C. Stein***
“**INTRODUCTION À L’ALGORITHMIQUE**”

É. Baudry, Notes de cours INF3105 Structures de données et algorithmes, UQÀM

# Analyse de la complexité

- Quoi analyser?
- Le nombre d'opérations primitives exécutées en fonction de la taille d'entrée.
  - Cas moyen – difficile à trouver
    - Moyenne de toutes les cas possibles
- Algorithmes sont mesurés typiquement selon la complexité temporelle du pire des cas
  - Pire cas : le pire cas où l'algorithme se comporte le moins bien
- Analyse amortie
  - Le temps moyen d'une opération répétée plusieurs fois dans le cadre d'une opération de plus haut niveau

# Analyse amortie

- Temps requis pour effectuer une suite d'opérations sur une structure de données est une moyenne sur l'ensemble des opérations effectuées
- L'analyse amortie
  - permet de montrer que le coût moyen d'une opération est faible si l'on établit sa moyenne sur une suite d'opérations, même si l'opération prise individuellement est coûteuse
  - diffère de l'analyse du cas moyen au sens où on ne fait pas appel aux probabilités
  - garantit les *performances moyennes de chaque opération dans le cas le plus défavorable*.

# Analyse amortie

- Idée: s’autoriser à faire certaines opérations coûteuses sur une structure de données, si celles-ci n’arrivent pas souvent
- Le coût élevé des opérations défavorables est compensé par le fait que la plupart des autres opérations ont un coût faible

# Tableau dynamique

- Limites des tableaux natifs de Java
  - Taille fixe
- Pour contourner le problème de taille fixe, il faut :
  - allouer un nouveau tableau ;
  - copier les éléments de l'ancien vers le nouveau ;
- Un tableau est accessible via une référence (adresse sur le premier élément)
- Besoin : structure abstraite tableau
  - Faire abstraction de la gestion de mémoire
  - Modifier la taille d'un tableau (taille dynamique)

# Analyse amortie; Tableau dynamique

- Agrandir le tableau d'un élément à chaque fois permet d'éviter le gaspillage de la mémoire. Toutefois, cela est couteaux en temps.
- Il faudra recopier tous ses éléments à chaque ajout
- Pour ajouter n éléments, il en coûtera

  1 + 2 + 3 + … + n = n(n-1)/2, soit O($n^2$)
- Chaque ajout coute O($n^2$)/n => O(n) en temps amorti

# Tableau dynamique

- Taille dynamique
  - Réallocation par doublement de taille
  - Meilleure politique quant au temps d’exécution
- Si on ajoute n éléments il faudra recopier les éléments à chaque fois que le nombre d’éléments franchis une puissance de deux

# Tableau dynamique ; Analyse amortie

- Réallocation par doublement de taille
- Meilleure politique quant au temps d'exécution
- Si on ajoute n éléments il faudra recopier les éléments à chaque fois que le nombre d'éléments franchis une puissance de deux
- $n = 2^k$,k –nombre d'agrandissements
  - Ces cas: l'opération d'ajout coûtera O(n)
  - Les autres: O(1)
- Analyse amortie: Insertion => O(1)

# Analyse amortie; Tableau dynamique, détails

- Réallocation par doublement de taille
- Début : tableau à 1 élément
- À la k-ième réallocation, la taille du tableaux : $2^k$ ($n = 2^k$)
- Le coût de k agrandissements:
- 1 + 2 + 4 + 8 + 16 + … $2^k$=>

k aggrandisements

=>$\sum_{i=0}^{k} 2^i = (2^{k+1}-1)/(2-1) = 2 * 2^k - 1 = 2 * n - 1$

- 2*n – 1 opérations de copie ϵ O(n)
- Pour ajouter n éléments => O(n)
- Le temps amorti de l'opération ajout: O(n)/n =>O(1)