- HAI503I: Algorithmique 4 -

Chap. 3 – Analyse amortie, analyse d'algorithmes probabilistes

L3 informatique
Université de Montpellier

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

Incrémenter un entier de 0 à $2^k - 1$

Représentation

- Tableau T de k bits (ou *mot binaire de longueur* k)
- Entier N représenté : $\sum_{i=0}^{k-1} T_{[i]} 2^i$

Incrément(T)

Entrée : Tableau T de taille k représentant un entier N

Sortie : Le même T, représentant $N + 1$ *modulo* 2^k

1. $i \leftarrow 0$
2. Tant que $i < k$ et $T_{[i]} = 1$:
3. $\quad T_{[i]} \leftarrow 0$
4. $\quad i \leftarrow i + 1$
5. Si $i < k$: $T_{[i]} \leftarrow 1$
6. Renvoyer T

Propriétés

Correction

- Si T représente N, alors après INCRÉMENT, T représente $N' = N + 1 \bmod 2^k$

Preuve

- Si $N = 2^k - 1$, $T_{[i]} = 1$ pour tout i et après incrément $T_{[i]} = 0$ pour tout i
- Sinon, soit i tel que $T_{[i]} = 0$ et $T_{[j]} = 1$ pour tout $j < i$:
  - Après : $T_{[i]} = 1$, $T_{[j]} = 0$ pour $j < i$ et $T_{[k]}$ inchangé pour $k > i$
  - Donc $N' = N + 2^i - \sum_{j<i} 2^j = N + 1$

Propriétés

Correction

- Si T représente N, alors après INCRÉMENT, T représente $N' = N + 1 \bmod 2^k$

Preuve

- Si $N = 2^k - 1$, $T_{[i]} = 1$ pour tout i et après incrément $T_{[i]} = 0$ pour tout i
- Sinon, soit i tel que $T_{[i]} = 0$ et $T_{[j]} = 1$ pour tout $j < i$:
  - Après : $T_{[i]} = 1$, $T_{[j]} = 0$ pour $j < i$ et $T_{[k]}$ inchangé pour $k > i$
  - Donc $N' = N + 2^i - \sum_{j<i} 2^j = N + 1$

Complexité

- INCRÉMENT a une complexité en $O(k)$

Preuve

- Pire cas $\rightsquigarrow$ on parcourt une fois tout le tableau T

Peut-on dire mieux ?

La complexité est-elle *vraiment* $O(k)$?

- `01...11` $\to$ `10...00` : demande effectivement k *inversions* de bits
- `10...00` $\to$ `10...01` : ne demande qu'une inversion de bit

Comment prendre en compte les variations ?

- Les déroulements de Incrément peuvent coûter 1, 2, ..., k
- Lesquels sont les plus *fréquents* ?

$\rightsquigarrow$ Cela dépend de la séquence d'appel de Incrément, mais on peut analyser le coût de cette séquence sur le même tableau T.

Suite d'Incréments

On incrémente T de 0 à $N-1$: quel est le coût *global* ?

Analyse *pire cas*

- T est de taille $k \rightarrow$ chaque Incrément coûte $O(k)$
- On effectue N Incrément $\rightarrow$ coût global $O(Nk)$
- Remarque : si $N \simeq 2^k$, chaque Incrément coûte $O(\log N) \rightarrow O(N \log N)$

Suite d'Incréments

On incrémente T de 0 à $N-1$: quel est le coût *global* ?

Analyse *pire cas*

- T est de taille $k \rightarrow$ chaque Incrément coûte $O(k)$
- On effectue N Incrément $\rightarrow$ coût global $O(Nk)$
- Remarque : si $N \simeq 2^k$, chaque Incrément coûte $O(\log N) \rightarrow O(N \log N)$

Analyse *amortie*

- $T_{[0]}$ est inversé à chaque fois
- $T_{[1]}$ est inversé une fois sur deux
- ...
- $T_{[k-1]}$ est inversé une fois sur 2^{k-1}

$\rightarrow$ Coût global : $\sum_{i=0}^{k-1} \lfloor \frac{N}{2^i} \rfloor < N \sum_{i=0}^{+\infty} \frac{1}{2^i} = 2N$

Bilan sur Incrément

Coût d'un appel à Incrément

- Pire cas : on doit parcourir tout le tableau $T \rightarrow O(k)$
- On ne peut pas dire mieux *a priori*

Coût de N appels à Incrément

- Pire cas : $N \times O(k) = O(Nk)$
- Coût global amorti : $O(N)$ car certains Incrément peu chers

Coût *amorti*

Le **coût amorti** de l'algorithme est $O(1)$ **par appel à** Incrément

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

Analyse pire cas et analyse amortie

Scénario

- Algo de complexité $C(n)$ pour une entrée de taille n, *dans le pire cas*
- Séquence de N appels à : coût $c_i \leq C(n)$ sur l'entrée n°i

Deux analyses possibles

- **Analyse pire cas :** le coût global est borné par $N \times C(n)$
- **Analyse amortie :** le coût global est $\leq \sum_{i=1}^{N} c_i$
- L'analyse pire cas reste valide ; l'analyse amortie est plus fine

Analyse pire cas et analyse amortie

Scénario

- Algo de complexité $C(n)$ pour une entrée de taille n, *dans le pire cas*
- Séquence de N appels à : coût $c_i \leq C(n)$ sur l'entrée n°i

Deux analyses possibles

- **Analyse pire cas :** le coût global est borné par $N \times C(n)$
- **Analyse amortie :** le coût global est $\leq \sum_{i=1}^{N} c_i$
- L'analyse pire cas reste valide ; l'analyse amortie est plus fine

Objectifs et techniques

- Objectif : borner le **coût amorti** par opération $\frac{1}{N} \sum_{i=1}^{N} c_i$
- Borner directement chaque c_i souvent difficile, voire impossible
- Plusieurs méthodes d'analyse :
  - méthode de l'agrégat
  - méthode comptable
  - méthode du potentiel

Méthode de l'agrégat

Idée : on calcule la somme totale $\sum_{i=1}^{N} c_i$ puis on divise par N

- Agrégat : mot compliqué mais idée simple $\rightsquigarrow$ méthode directe

Mise en œuvre

- Regarder globalement les N appels comme une seule exécution
- Regrouper des opérations venant de différents appels pour mieux compter

Méthode de l'agrégat

Idée : on calcule la somme totale $\sum_{i=1}^{N} c_i$ puis on divise par N

- Agrégat : mot compliqué mais idée simple $\rightsquigarrow$ méthode directe

Mise en œuvre

- Regarder globalement les N appels comme une seule exécution
- Regrouper des opérations venant de différents appels pour mieux compter

Exemple pour INCRÉMENT

- Compter le nombre total d'inversions du bit $T_{[0]}$, du bit $T_{[1]}$, etc.
- Coût total $2N \rightsquigarrow$ **coût amorti** 2, en $O(1)$

Méthode comptable

Idée : payer plus que le *vrai* coût à certains appels, et moins à d'autres

- Les coûts sont de l'argent, à la fin, notre **compte** doit être en positif

Mise en œuvre

- À chaque appel, on ajoute une somme a_i à un compte et on prélève c_i le vrai coup des opérations sur ce compte
- Le compte doit toujours rester positif : pour tout i, on veut : $\sum_{j \leq i} c_j \leq \sum_{j \leq i} a_j$
- Coût amorti par opération : $\frac{1}{N} \sum_i c_i \leq \frac{1}{N} \sum_i a_i$
- Remarque : plus général que l'agrégat (pour lequel, on prend $a_i = c_i$)

Méthode comptable

Idée : payer plus que le *vrai* coût à certains appels, et moins à d'autres

- Les coûts sont de l'argent, à la fin, notre **compte** doit être en positif

Mise en œuvre

- À chaque appel, on ajoute une somme a_i à un compte et on prélève c_i le vrai coup des opérations sur ce compte
- Le compte doit toujours rester positif : pour tout i, on veut : $\sum_{j \leq i} c_j \leq \sum_{j \leq i} a_j$
- Coût amorti par opération : $\frac{1}{N} \sum_i c_i \leq \frac{1}{N} \sum_i a_i$
- Remarque : plus général que l'agrégat (pour lequel, on prend $a_i = c_i$)

Exemple pour Incrément

- Intuition : quand un bit passe de 0 à 1, on paie 1, plus 1 pour l'inversion future $1 \to 0$
- $a_i = 2$ pour tout i car un seul bit passe de 0 à 1 $\to$ coût amorti 2
- Solde du compte = nombre de bits à 1 $\to$ toujours positif

Méthode du potentiel

Idée : associer à chaque appel une *modification de potentiel*

- *Technique pour calculer les a_i de la méthode comptable*, métaphore de l'**énergie potentielle** en physique

Mise en œuvre

- Définir une *fonction potentiel* Φ, qui vaut $\Phi_t \geq \Phi_0$ après t appels
- Estimer le *coût amorti* de chaque appel : $a_i = c_i + (\Phi_i - \Phi_{i-1})$
- Coût amorti par opération :
  $\frac{1}{N}\sum_{i=1}^{N} c_i = \frac{1}{N}\sum_{i=1}^{N} a_i - \frac{1}{N}(\Phi_N - \Phi_0) \leq \frac{1}{N}\sum_i a_i$

Méthode du potentiel

Idée : associer à chaque appel une *modification de potentiel*

- *Technique pour calculer les a_i de la méthode comptable*, métaphore de l'**énergie potentielle** en physique

Mise en œuvre

- Définir une *fonction potentiel* Φ, qui vaut $\Phi_t \geq \Phi_0$ après t appels
- Estimer le *coût amorti* de chaque appel : $a_i = c_i + (\Phi_i - \Phi_{i-1})$
- Coût amorti par opération :
  $\frac{1}{N}\sum_{i=1}^{N} c_i = \frac{1}{N}\sum_{i=1}^{N} a_i - \frac{1}{N}(\Phi_N - \Phi_0) \leq \frac{1}{N}\sum_i a_i$

Exemple pour Incrément

- Potentiel du tableau T : $\Phi_i =$ nombre de 1 dans T après i appels
- Avant le $i$ème appel, on écrit $T = \ldots 01 \ldots 1$ avec ℓ bits à 1
  - coût de Incrément(T) : $c_i = \ell + 1$
  - différence de potentiel : $\Phi_i - \Phi_{i-1} = -(\ell - 1)$
  - coût amorti : $a_i = (\ell + 1) - (\ell - 1) = 2$

Bilan sur les trois méthodes

Techniques plus ou moins faciles

- Méthode de l'agrégat : idée évidente... mais demande une compréhension globale
- Méthodes comptable et du potentiel : plus difficile à mettre en œuvre, mais compréhension *locale*

Idées communes aux méthodes comptable et du potentiel

- Calcul direct d'un coût amorti pour chaque appel
- Preuve globale que le coût amorti défini est *valide*
- Forme d'analyse pire cas avec une notion de coût modifiée

Utilisation principale : structures de données

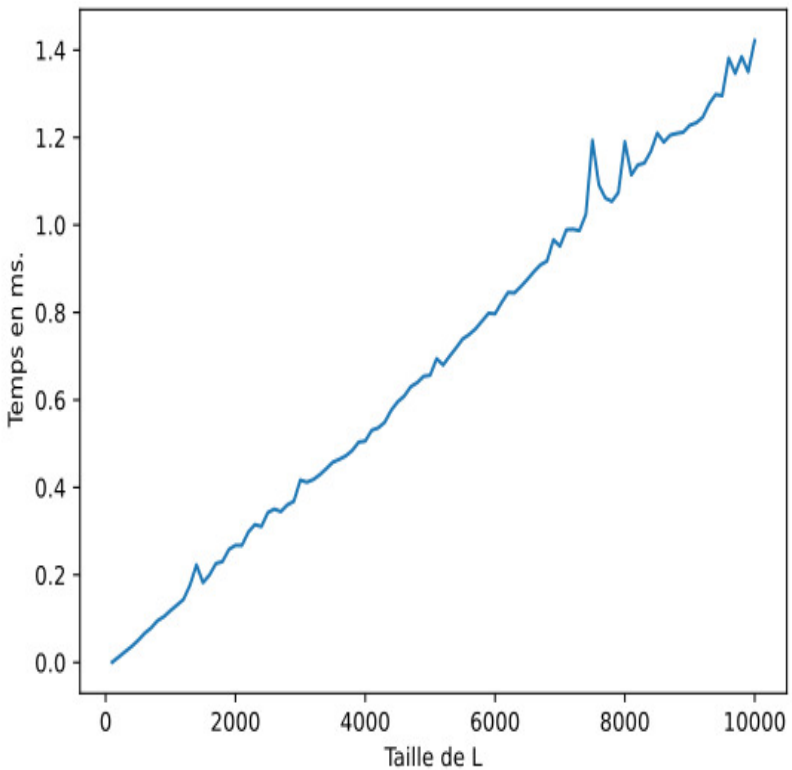
- Ensemble d'algorithmes de manipulation de la structure (ajout, suppression, etc.)
- Coûts variables $\rightsquigarrow$ analyse amortie pour avoir un *coût amorti par opération*

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

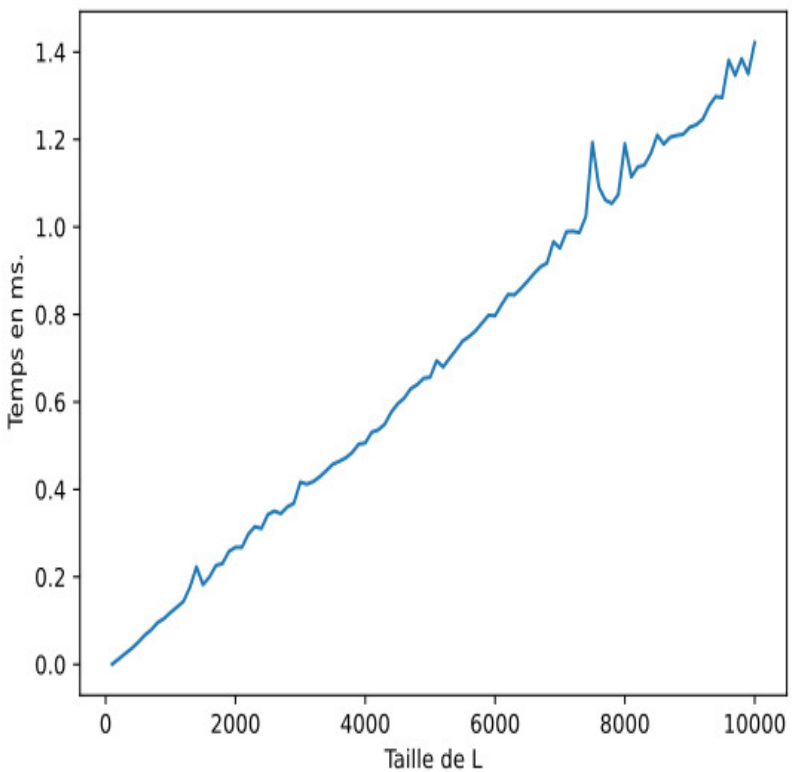
Exemple des `list` en Python

```
1  def test(n):
2          L = []
3          for i in range(n): L.append(i)
4          for i in range(n//2):
5                  L[i], L[n-i-1] = L[n-i-1], L[i]
```



Exemple des `list` en Python

```
1  def test(n):
2          L = []
3          for i in range(n): L.append(i)
4          for i in range(n//2):
5                  L[i], L[n-i-1] = L[n-i-1], L[i]
```

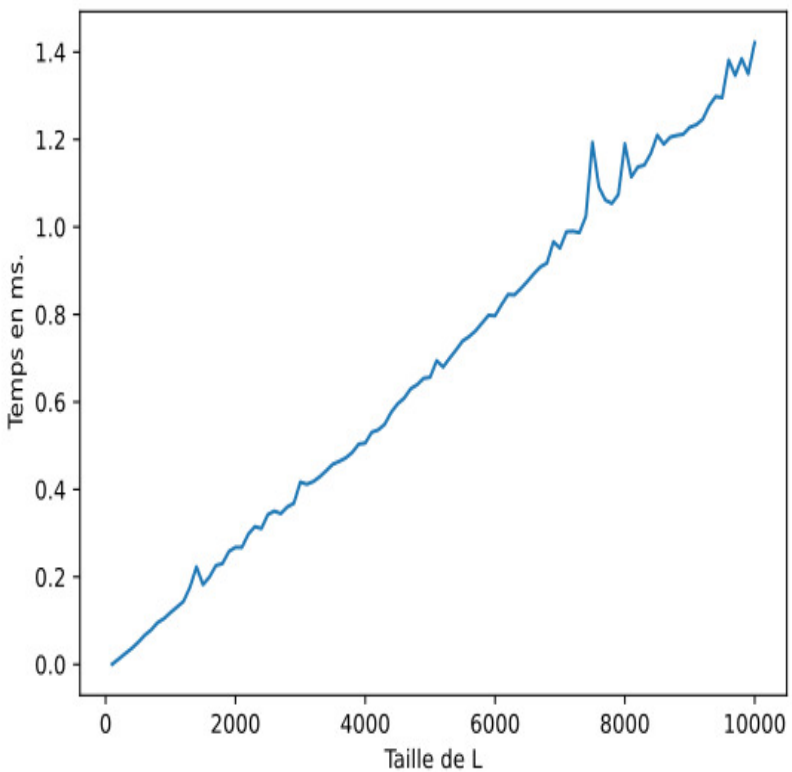


Quelle structure de données ?

- Ajout en fin de liste en $O(1)$ $\rightarrow$ liste chaînée ?
- Accès à $L_{[i]}$ en temps $O(1)$ $\rightarrow$ tableau ?

Exemple des `list` en Python

```
1  def test(n):
2          L = []
3          for i in range(n): L.append(i)
4          for i in range(n//2):
5                  L[i], L[n-i-1] = L[n-i-1], L[i]
```



Quelle structure de données ?

- Ajout en fin de liste en $O(1)$ → liste chaînée ?
- Accès à $L_{[i]}$ en temps $O(1)$ → tableau ?

`list` = **tableau dynamique**

Les tableaux dynamiques

Idée de base

- Structure de donnée sous-jacente : un tableau
- Stockage de n éléments dans un tableau de taille N

Conditions à respecter

- Il faut toujours $N \geq n$ pour avoir assez de place
- Il ne faut pas $N \gg n$: utilisation de place inutile

Objectifs

- Assurer $N \simeq n$: en pratique $n \leq N$ et $N/4 < n \rightsquigarrow n \leq N < 4n$
- Accès à un élément en temps $O(1)$: immédiat
- Ajout et suppression en fin de tableau en $O(1)$

Ajout et suppression

Ajout d'un élément x à la fin

- Si $N > n$: ajouter x dans la case libre $T_{[n]}$
- Sinon :
  - Créer un nouveau tableau U de taille $2N$
  - Recopier les N cases de T dans U
  - Ajouter x dans la case libre $U_{[N]}$
- Dans les deux cas : mettre à jour $n \leftarrow n+1$ et $N \leftarrow 2N$

Suppression d'un élément x à la fin

- Pas de difficulté : $n \leftarrow n-1$
- Pour éviter $N \gg n$, il faut (parfois) réduire la taille de T
  - Idée 1 : si $n < N/2$ on réduit de moitié $\rightarrow$ mauvaise idée
  - Idée 2 : si $n < N/4$ on réduit de moitié $\rightarrow$ bonne idée

Remarque

- On garde toujours $N \geq 1$, même si $n = 0$
- À la supression, inutile d'effacer $T[n]$, l'opération $n \leftarrow n-1$ suffit

Les algorithmes Ajout et Suppression

Ajout(T, N, n, x)

1. Si $n < N$:
2. $\quad T_{[n]} \leftarrow x$
3. $\quad$ Renvoyer $(T, N, \mathrm{n}+1)$
4. $U \leftarrow$ tableau de taille $2N$
5. Pour $i = 0$ à $N-1$:
   $U_{[i]} \leftarrow T_{[i]}$
6. $U_{[N]} \leftarrow x$
7. Renvoyer $(\mathrm{U}, 2\mathrm{N}, \mathrm{n}+1)$

Suppression(T, N, n)

1. Si $n = 1$ ou $n > N/4$:
2. $\quad$ Renvoyer $(T, N, \mathrm{n}-1)$
3. $U \leftarrow$ tableau de taille $N/2$
4. Pour $i = 0$ à $n-2$:
   $U_{[i]} \leftarrow T_{[i]}$
5. Renvoyer $(\mathrm{U}, \mathrm{N}/2, \mathrm{n}-1)$

Dans le pire cas, Ajout et Suppression effectuent chacun $O(n)$ affectations

Analyse amortie 1 : uniquement des AJOUT

Coût de m AJOUT dans un tableau initialement vide ?

Analyse pire cas

- Un AJOUT dans un tableau de taille k coûte $O(k)$
  $\rightsquigarrow$ coût total $O(m^2)$

Analyse amortie 1 : uniquement des AJOUT

Coût de m AJOUT dans un tableau initialement vide ?

Analyse pire cas

- Un AJOUT dans un tableau de taille k coûte $O(k)$
  $\rightsquigarrow$ coût total $O(m^2)$

Méthode de l'agrégat

- Les affectations initiales $T_{[n]} \leftarrow x$ coûtent 1.
- Quand les **réaffectations** $U_{[i]} \leftarrow T_{[i]}$ ont lieu, la taille de T est doublée
- N est toujours une puissance de 2, et T est doublé quand $n = N$
- Coût total des réaffectations : $\sum_{k=1}^{\lfloor \log m \rfloor} 2^k < 2^{\lfloor \log m \rfloor + 1} \leq 2m$
  $\rightsquigarrow$ coût amortie des réaffectation par AJOUT : 2

Théorème

Coût amorti par dans un tableau initialement vide : 3

Analyse amortie 2 : Ajout et Suppression

Coût de m opérations dans un tableau initialement vide ?

Notations

Après la $i^{\text{ème}}$ opération,

- n_i : nombre d'élément dans le tableau
- N_i : taille du tableau
- $\alpha_i = n_i/N_i$: *coefficient de remplissage*
- c_i : coût de la $i^{\text{ème}}$ opération (nombre d'affectations)

Fonction potentiel

$$\Phi_i = \begin{cases} 2n_i - N_i & \text{si } \alpha_i \geq \frac{1}{2} \\ N_i/2 - n_i & \text{si } 0 < \alpha_i \leq \frac{1}{2} \\ 0 & \text{si } \alpha_i = 0 \end{cases}$$

Analyse amortie 2 : Ajout et Suppression

Coût de m opérations dans un tableau initialement vide ?

Notations

Après la $i^{\text{ème}}$ opération,

- n_i : nombre d'élément dans le tableau
- N_i : taille du tableau
- $\alpha_i = n_i/N_i$: *coefficient de remplissage*
- c_i : coût de la $i^{\text{ème}}$ opération (nombre d'affectations)

Fonction potentiel

$$\Phi_i = \begin{cases} 2n_i - N_i & \text{si } \alpha_i \geq \frac{1}{2} \\ N_i/2 - n_i & \text{si } 0 < \alpha_i \leq \frac{1}{2} \\ 0 & \text{si } \alpha_i = 0 \end{cases}$$

Le coût amorti $a_i = c_i + \Phi_i - \Phi_{i-1}$ de la $i^{\text{ème}}$ opération est ≤ 3 pour tout i

Bilan sur les tableaux dynamiques

Principes

- Tableau de taille variable
  - Mémoire *allouée* supérieure à celle utilisée
  - Remplissage : $\frac{1}{4} \leq \alpha \leq \frac{1}{2}$
  - Taille doublée ou divisée par deux quand nécessaire
- Accès direct et *en fin de tableau* en temps constant

Bilan sur les tableaux dynamiques

Principes

- Tableau de taille variable
  - Mémoire *allouée* supérieure à celle utilisée
  - Remplissage : $\frac{1}{4} \leq \alpha \leq \frac{1}{2}$
  - Taille doublée ou divisée par deux quand nécessaire
- Accès direct et *en fin de tableau* en temps constant

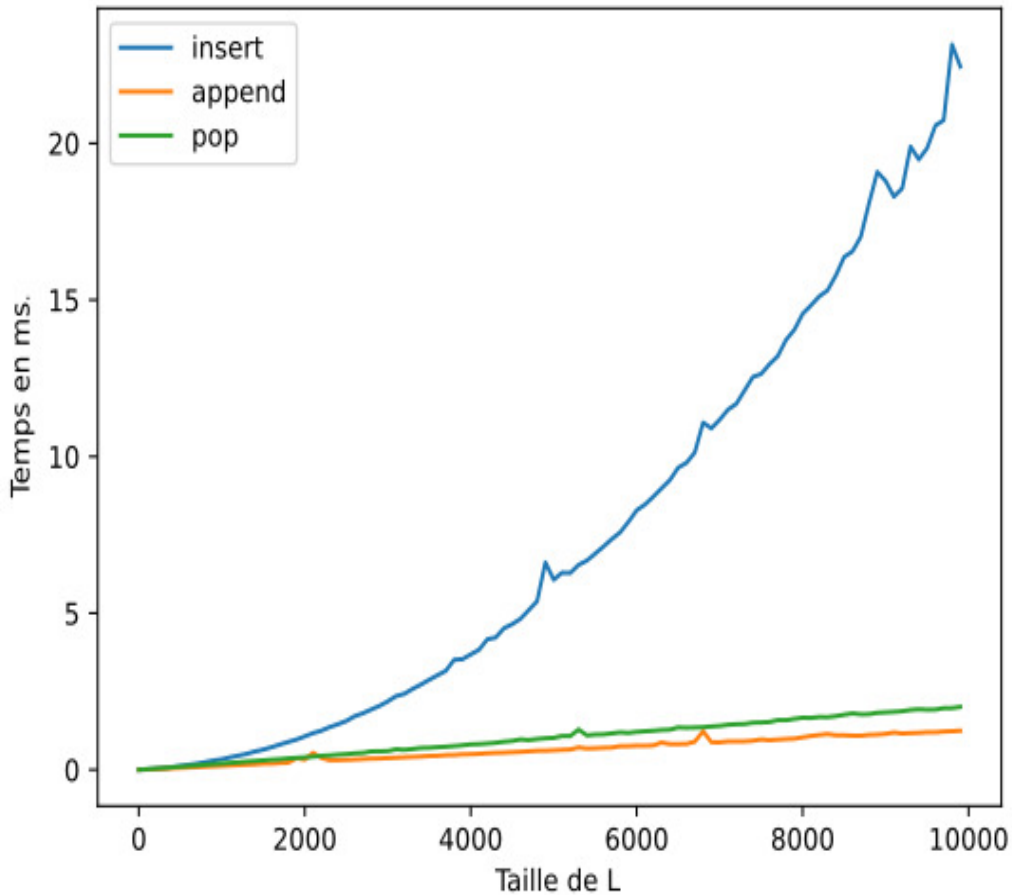
Complexité amortie

- Chaque opération coûte ≤ 3 affectations $\rightsquigarrow$ coût constant par opération
- Mais tout de même : si on connaît à l'avance la taille, coût triplé

Autres utilisations

- Création de pile $\rightsquigarrow$ idem
- Création de file $\rightsquigarrow$ travail supplémentaire, *cf* TD

Performance des `list` Python



- Insertion en début de tableau
- Uniquement insertion en fin de tableau
- Insertion et suppression en fin de tableau

Conclusion sur l'analyse amortie

Technique avancée d'analyse d'algorithmes

- Dépasser l'analyse *pire cas*
- Prendre en compte les variations de temps entre différents appels

Trois techniques

- Méthode de l'agrégat
- Méthode comptable
- Méthode du potentiel

Utilisation principale : structures de données

- Chaque opération *peut* coûter cher
- Mais peu d'opérations coûtent cher
- Si on utilise plusieurs fois la structure de donnée $\rightsquigarrow$ coût amorti faible

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

Problème de la sélection (ou $k$ième rang)

QuickSelect(T, k)

Entrées : Tableau T de taille n d'entiers tous distincts ; entier k entre 1 et n

Sortie : Le $k^{\text{ème}}$ plus petit élément $T^{(k)}$ de T

1. $p \leftarrow T_{[i]}$ avec i choisi aléatoirement entre 0 et $n-1$ *(pivot)*
2. $n_0 \leftarrow$ nombre d'élément $< p$ dans T *(boucle Pour)*
3. Si $n_0 = k-1$: Renvoyer p
4. Si $n_0 \geq k$:
5. $\quad T_0 \leftarrow$ tableau des éléments de T qui sont $< p$ *(boucle Pour)*
6. $\quad$ Renvoyer QuickSelect(T_0, k)
7. $T_1 \leftarrow$ tableau des éléments de T qui sont $> p$ *(boucle Pour)*
8. Renvoyer QuickSelect$(T_1, k - n_0 - 1)$

Correction

$$T^{(k)} = \begin{cases} p & \text{si } n_0 = k-1 \\ T_0^{(k)} & \text{si } n_0 \geq k \\ T_1^{(k-n_0-1)} & \text{sinon} \end{cases}$$

Complexité de QuickSelect

Analyse en pire cas

- Deux boucles en $O(n)$ + appel récursif sur un tableau de taille $\leq n-1$
- $C_n \leq C_{n-1} + O(n) \rightarrow C_n = O(n^2)$ $\quad$ C_n *: nombre de comparaisons*

Peut-on dire mieux ?

- Appel récursif sur un tableau de taille $n-1$ si $T_{[i]}$ est toujours le minimum ou toujours le maximum
- Est-ce que ça arrive *en pratique* ?
  - Quelle est la taille *moyenne* (espérance) du tableau pour l'appel récursif ?
  - Quelle est l'espérance du nombre de comparaisons effectuées ?

Théorème

Soit C_n le nombre de comparaisons effectuées par QuickSelect(T, k) où T est de taille n. Alors $\mathbb{E}[C_n] \leq 4n$, quelque soit k.

Bilan sur QuickSelect

Nombre de comparaisons

- Si on est **très** malchanceux, effectue $\sim \frac{1}{2}n^2$ comparaisons
- L'espérance du nombre de comparaisons effectuées est $\leq 4n = O(n)$

Que veut dire *malchanceux* ?

- Espérance valable *quelque soit l'entrée* (T et k)
  - Pas de chance ou malchance par rapport à l'entrée
  - La probabilité porte sur les choix aléatoires de l'algorithme
- On peut être *parfois* malchanceux au cours de l'algorithme
  - Pas besoin d'avoir de la chance à chaque étape...
  - ... simplement de ne pas être très malchanceux à chaque étape
- Exemple : proba. de faire toujours le pire choix $= 1/n!$
  - si $n = 10$: $1/3\,628\,800$
  - si $n = 100$: $< 1/10^{157}$

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

Coupe minimale dans un graphe

Définition

- Une **coupe** dans un graphe $G = (V, A)$ est une partition (V_1, V_2) de l'ensemble de ses sommets en deux ensembles non vides.
- La **taille** de la coupe (V_1, V_2) est le nombre d'arêtes entre V_1 et V_2 : $|\{u_1u_2 \in A : u_1 \in V_1, u_2 \in V_2\}|$

Coupe minimale dans un graphe

Définition

- Une **coupe** dans un graphe $G = (V, A)$ est une partition (V_1, V_2) de l'ensemble de ses sommets en deux ensembles non vides.
- La **taille** de la coupe (V_1, V_2) est le nombre d'arêtes entre V_1 et V_2 : $|\{u_1u_2 \in A : u_1 \in V_1, u_2 \in V_2\}|$

Problème de la coupe minimale

Entrée : Graphe $G = (V, A)$

Sortie : Une coupe (V_1, V_2) de taille minimale

Généralisation nécessaire : multigraphes

- Un *multigraphe* est un graphe qui autorise plusieurs arêtes entre 2 sommets
- Coupe et problème de la coupe minimale définis de manière équivalente

Algorithme probabiliste pour la coupe minimale

Contraction d'arête

Soit $G = (V, A)$ un (multi)graphe et uv une arête de G. Le (multi)graphe G/uv, obtenu par contraction de l'arête uv, a pour sommets $V \setminus v$ et pour ensemble d'arêtes $(A \setminus \{xv \ : \ xv \in A\}) \cup \{xu \ : \ xv \in A, x \neq u\}$

Algorithme probabiliste pour la coupe minimale

Contraction d'arête

Soit $G = (V, A)$ un (multi)graphe et uv une arête de G. Le (multi)graphe G/uv, obtenu par contraction de l'arête uv, a pour sommets $V \setminus v$ et pour ensemble d'arêtes $(A \setminus \{xv \ : \ xv \in A\}) \cup \{xu \ : \ xv \in A, x \neq u\}$

CoupeMin(G)

1. Tant que G possède au moins 3 sommets :
2. Choisir une arête uv de G, aléatoirement et uniformément
3. Contracter l'arête uv dans G
4. Renvoyer la coupe définie par les deux sommets restants

Complexité de CoupeMin

G : un multigraphe à n sommets

Lemme (admis)

Si G est représenté par listes d'adjacence avec pointeurs, la *contraction* d'une arête peut s'effectuer en temps $O(n)$, où n est le nombre de sommets de G.

Complexité de CoupeMin

G : un multigraphe à n sommets

Lemme (admis)

Si G est représenté par listes d'adjacence avec pointeurs, la *contraction* d'une arête peut s'effectuer en temps $O(n)$, où n est le nombre de sommets de G.

Théorème

L'algorithme renvoie une coupe de G en temps $O(n^2)$

Preuve

- À chaque itération, on contracte une arête $\rightarrow$ le nombre de sommets diminue de 1
- Le nombre d'itérations est donc $\leq n-2$
- La complexité totale est $O(n^2)$

La complexité ne dépend pas des choix probabilistes

Correction de CoupeMin

Lemme de correction

L'algorithme appliqué à un multigraphe à n sommets renvoie une coupe *minimale* avec probabilité $\geq \frac{2}{n(n-1)}$

Correction de CoupeMin

Lemme de correction

L'algorithme appliqué à un multigraphe à n sommets renvoie une coupe *minimale* avec probabilité $\geq \frac{2}{n(n-1)}$

Remarque

- Cette probabilité est très petite
- Exemple pour $n = 100$, $\frac{2}{n(n-1)} \simeq 0,02\%\ldots$

Répétitions de CoupeMin

Probabilité de *succès* très faible $\rightsquigarrow$ on répète l'algorithme pour améliorer cette probabilité
Technique très classique avec des algorithmes probabilistes !

Lemme de répétition

Si on répète N fois CoupeMin et qu'on garde la plus petite coupe renvoyée, cette coupe est minimale avec probabilité $\geq 1 - e^{-2N/n(n-1)}$

Remarques

- Si on répète $N = 2n^2$ fois l'algorithme, on obtient
  - une complexité $O(n^4)$
  - une coupe minimale avec probabilité $\geq 1 - e^{-4} \simeq 98\%$

Bilan sur CoupeMin

Complexité

- Un appel à coûte toujours $O(n^2)$
- On a besoin de $O(n^2)$ *répétitions* $\rightarrow$ $O(n^4)$

Correction

- Un appel à renvoie une coupe minimale avec proba. $\geq \frac{2}{n(n-1)}$
- N appels à renvoient une coupe minimale avec proba. $\geq 1 - e^{-2N/n(n-1)}$
- cn^2 appels à renvoient une coupe minimale avec proba. $\geq 1 - e^{-2c}$

Bilan sur CoupeMin

Complexité

- Un appel à coûte toujours $O(n^2)$
- On a besoin de $O(n^2)$ *répétitions* $\rightarrow O(n^4)$

Correction

- Un appel à renvoie une coupe minimale avec proba. $\geq \frac{2}{n(n-1)}$
- N appels à renvoient une coupe minimale avec proba. $\geq 1 - e^{-2N/n(n-1)}$
- cn^2 appels à renvoient une coupe minimale avec proba. $\geq 1 - e^{-2c}$

Théorème, Coupe minimale probilisée

CoupeMin répété $O(n^2)$ fois demande un temps $O(n^4)$, et retourne une coupe minimale avec très forte probabilité.

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

Définition

Algorithme probabiliste

Un **algorithme probabiliste** est un algorithme qui effectue des **choix aléatoires** au cours de son exécution.

Remarque

- Choix aléatoires : accès à un générateur de bits aléatoires, ou d'entiers aléatoires, etc.

Analyse d'un algorithme probabiliste

- Le comportement d'un algorithme probabiliste est une *expérience probabiliste*
- Le résultat renvoyé (*correction*) ou le nombre d'opérations effectuées (*complexité*) peuvent dépendre des choix aléatoires $\rightsquigarrow$ **deux familles d'algorithmes**

Les algorithmes de type *Las Vegas*

Algorithme de type Las Vegas

Un algorithme probabiliste est de **type Las Vegas** si

- son résultat ne dépend pas des choix aléatoires
- sa complexité dépend des choix aléatoires

- Le nombre d'opérations élémentaires est modélisé par une variable aléatoire, son espérance donne la complexité attendue
- Exemple : QuickSelect

Signification de l'espérance de la complexité

"L'espérance de la complexité est $O(n^2)$"

- On *s'attend* à avoir une exécution en temps $O(n^2)$
- Si on exécute l'algorithme de nombreuses fois (sur la même entrée), le temps de calcul moyen sera $O(n^2)$

Les algorithmes de type *Las Vegas*

Algorithme de type Las Vegas

Un algorithme probabiliste est de **type Las Vegas** si

- son résultat ne dépend pas des choix aléatoires
- sa complexité dépend des choix aléatoires

- Le nombre d'opérations élémentaires est modélisé par une variable aléatoire, son espérance donne la complexité attendue
- Exemple : QuickSelect

Signification de l'espérance de la complexité

"L'espérance de la complexité est $O(n^2)$"

- On *s'attend* à avoir une exécution en temps $O(n^2)$
- Si on exécute l'algorithme de nombreuses fois (sur la même entrée), le temps de calcul moyen sera $O(n^2)$

Las Vegas : " toujours correct, souvent rapide "

Les algorithmes de type *Monte Carlo*

Algorithme de type Monte Carlo

Un algorithme probabiliste est de **type Monte Carlo** si

- son résultat dépend des choix aléatoires
- sa complexité ne dépend pas des choix aléatoires

- L'étude de la correction se fait avec une *probabilité de succès* : probabilité que l'algorithme soit correct
- Exemple : CoupeMin

Améliorer la probabilité de succès par répétition

- On note p la probabilité de succès et on répète N fois l'algorithme
- La probabilité qu'une (au moins) des répétitions soit correcte est $1-(1-p)^N \geq 1-e^{-pN}$
- Il faut alors multiplier la complexité par N...

Monte Carlo : " toujours rapide, souvent correct "

Bilan sur les algorithmes probabilistes

Pourquoi des algorithmes probabilistes ?

- Algorithmes souvent simples et efficaces
- Parfois, meilleure complexité que les algorithmes déterministes. En pratique, ils fonctionnent très bien !

Analyse des algorithmes probabilistes

- Modélisation probabiliste, avec variable aléatoire
- **Las Vegas** : étude de l'espérance de la complexité
- **Monte Carlo** : étude de la probabilité de succès

Et ensuite ?

- **Atlantic City** : " souvent correct, souvent rapide "
- Algorithmes quantiques : généralisation des algorithmes probabilistes

1. Analyse amortie
1.1 Exemple 1 : le compteur binaire
1.2 L'analyse amortie
1.3 Exemple 2 : les tableaux dynamiques

2. Analyse d'algorithmes probabilistes
2.1 Exemple 1 : QuickSelect
2.2 Exemple 2 : coupe minimale
2.3 Algorithmes probabilistes
2.4 Exemple 3 : analyse probabiliste du tri rapide

Le tri rapide

TriRapide(T)

1. Si taille(T) $= 1$: renvoyer T
2. $p \leftarrow T_{[i]}$ avec i choisi aléatoirement entre 0 et $n-1$ *(pivot)*
3. $n_p \leftarrow$ nombre d'indices i tels que $T_{[i]} = p$ *(boucle Pour)*
4. $T_0 \leftarrow$ tableau des éléments de T qui sont $< p$ *(boucle Pour)*
5. $T_1 \leftarrow$ tableau des éléments de T qui sont $> p$ *(boucle Pour)*
6. $T_0 \leftarrow$ TriRapide(T_0)
7. $T_1 \leftarrow$ TriRapide(T_1)
8. Renvoyer la concaténation T_0, n_p fois p, et T_1

Correction (rapide...)

- Par récurrence sur la taille n de T ($n = 1$: ok)
- T_0 et T_1 sont de taille $< n \rightsquigarrow T_0$ et T_1 sont correctement triés
- donc le tableau renvoyé est correctement trié

Espérance du nombre de comparaisons

Théorème

L'espérance du nombre de comparaisons effectuées par TriRapide est $O(n \log n)$

Notations

- $T^{(i)}$: $i^{\text{ème}}$ plus petit élément de T
- $X_{ij} = 1$ si $T^{(i)}$ est comparé à $T^{(j)}$ au cours de l'algo, 0 sinon
- X : nombre total de comparaisons $\rightsquigarrow X = \sum_{i<j} X_{ij}$

Lemme

Pour $1 \leq i < j \leq n$, $\mathbb{E}[X_{ij}] = \Pr[X_{ij} = 1] = 2/(j - i + 1)$

Preuve du théorème

- $\mathbb{E}[X] = \sum_{i<j} \mathbb{E}[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k} \leq \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{2}{k}$
- On admet que $\sum_{k=1}^{n} 1/k = O(\log n)$
- Donc $\mathbb{E}[X] = O(n \log n)$

Bilan sur le TriRapide

Propriétés du TriRapide

- L'algorithme est toujours correct
- L'espérance de sa complexité est $O(n \log n)$ (dans le pire des cas, complexité en $O(n^2)$)
- Type d'algorithme probabiliste : *Las Vegas*

Comportement pratique

- Le TriRapide est efficace en pratique, s'il est implanté *en place*
- En pratique, c'est un des algorithmes de tri les plus rapides, donc un des plus utilisés.

Conclusion générale

Analyse amortie

- Analyse de plusieurs exécutions consécutives d'un même algorithme
- Complexité amortie = temps *moyen* pris par les exécutions successives
- Trois techniques de preuve : **agrégat**, **comptable** et **potentiel**

Analyse d'algorithmes probabilistes

- Analyse d'algorithmes qui font des choix aléatoires
- Étude de l'espérance de la complexité ou de la probabilité de succès
- Comportement *moyen* vis-à-vis des choix aléatoires

Conclusion générale

Analyse amortie

- Analyse de plusieurs exécutions consécutives d'un même algorithme
- Complexité amortie = temps *moyen* pris par les exécutions successives
- Trois techniques de preuve : **agrégat**, **comptable** et **potentiel**

Analyse d'algorithmes probabilistes

- Analyse d'algorithmes qui font des choix aléatoires
- Étude de l'espérance de la complexité ou de la probabilité de succès
- Comportement *moyen* vis-à-vis des choix aléatoires

Analyse en moyenne

- Analyse du comportement d'algorithmes sur des *entrées aléatoires*
- Calcul de l'espérance de la complexité sur une entrée aléatoire
- Question à considérer : quelle distribution sur les entrées ?