Les algorithmes **online**

Johanne Cohen

Les algorithmes Offline et Online

- **Algorithmes Offline** :

  toutes les données sont disponibles dès le début

- **Algorithmes Online** :

  il faut décider au fur et à mesure de l'arrivée des données

- **Compétitivité** : Un algorithme online est α-compétitif si

  pour toute entrée x, $online(x) \leq \alpha \cdot OPT(x) + cte$

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours :

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V,

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V, V,

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V, V, V,

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V, V, V, V,

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V, V, V, V, Stop.

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V, V, V, V, Stop.
- Algorithme on-line : louer le vélo 250 fois puis l'acheter.

L'exemple de base

Ennoncé

- Louer un vélo pour une journée coûte 1 euro.
- Acheter un vélo coûte 250 euros.

Comment ne pas trop dépenser, sachant qu'un jour on arrêtera le vélo.

Algorithme :

- Donnée : la suite des jours : V, V, V, V, Stop.
- Algorithme on-line : louer le vélo 250 fois puis l'acheter.
- Cet algorithme est 2-compétitif : l'algorithme optimal décide en fonction du nombre j de jours d'utilisation du vélo :
  1. Si $j < 250$ alors $online = OPT$
  2. Si $j \geq 250$ alors $online = 500$ et $OPT = 250$.

Retour du problème bin-packing

- **Donnée :** n objets de poids $p_1, \ldots, p_n$ et des boites de capacité C.
- **Problème :** Mettre les objets dans un nombre minimum de boites.
- **Algorithme glouton Online :** On place les objets dans l'ordre o ils arrivent en n'ouvrant une nouvelle boite que si c'est nécessaire.

algorithme Next Fit

- **Analyse :**
  - si l'algo online place un gros objet dans chaque boite, il est optimal
  - sinon Considérons la dernière boite ouverte qui n'ait pas de gros objet l'objet pour lequel elle a été ouverte néntrait dans aucune des autres boites

    toutes les boites sauf une sont au moins à moitié -pleine ;

$$\text{Glouton Online} \leq 2 \cdot OPT + 1$$

Gestion du cache mémoire

- **Donnée :**
  - mémoire centrale organisée en pages, mémoire cache de k
  - le chargement d'une page coûte 1 (et on sort une des pages du cache).
- **Problème :** être α-compétitif pour toute séquence de consultations de pages.

$\sigma =$ $<$

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1,	2,	3,	1,	2,	4,	5,	2,	3,	1
1	1	1	1	1	④	4	4	4	4
	2	2	2	2	2	2	2	③	①
		3	3	3	3	⑤	5	5	5

$>$

Différentes politiques

Lors qu'une page demandée n'est pas dans le cache, l'algorithme supprime la page. Plusieurs stratégies existent :

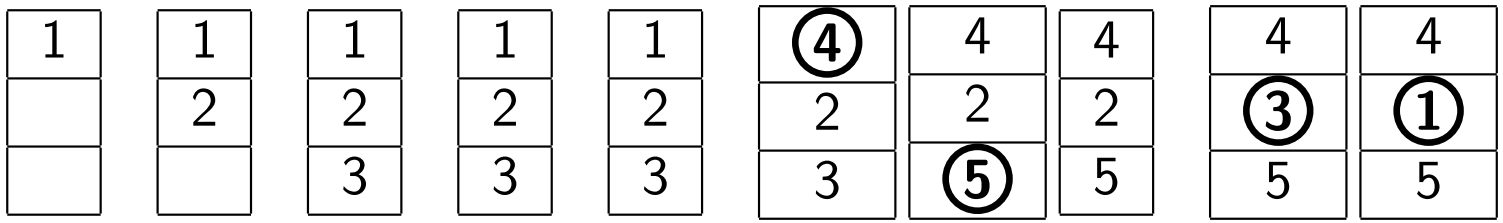
- **LIFO (Last-In/First-Out)** Supprimer la dernière page mise en cache
- **LFU (Least-Frequently-Used)** Supprimer la page la moins demandée en cache.
- **LRU (Least-Recently-Used)** Supprimer la page dont la dernière requête est la plus ancienne.
- **LFD (Longest forward distance) :** Supprimer la page dont la requête suivante est la plus éloignée dans la séquence.

Politique optimale en Offline

LFD (Longest forward distance) :

Supprimer la page dont la requête suivante est la plus éloignée dans la séquence.

$\sigma =$ $<$ 1, 2, 3, 1, 2, 4, 5, 2, 3, 1 $>$



Idée de la preuve : il existe un algorithme de gestion de base $\mathcal{B}$ de gestion de pages basé sur l'algorithme $\mathcal{A}$ tel que

1. à l'étape i, l'algorithme $\mathcal{B}$ utilise la politique *LFD* pour remplacer la page supprimée.
2. $cout_B(\sigma) \leq k \cdot cout_A(\sigma)$ pour toute séquence de pages.

Politiques en online

- Pour LIFO (Last-In/First-Out) :

$\sigma =$ $<$ **1,** **2,** **3,** **3,** **4,** **3,** **4,** **3,** **4** $>$

cache :									
	1	1	1	1	1	1	1	1	4
		2	2	2	2	2	2	2	2
			3	3	④	③	④	③	④

- Pour LRU (Least-Frequently-Used) :

$\sigma =$ $<$ **1,** **2,** **3,** **3,** **4,** **3,** **4,** **3,** **4** $>$

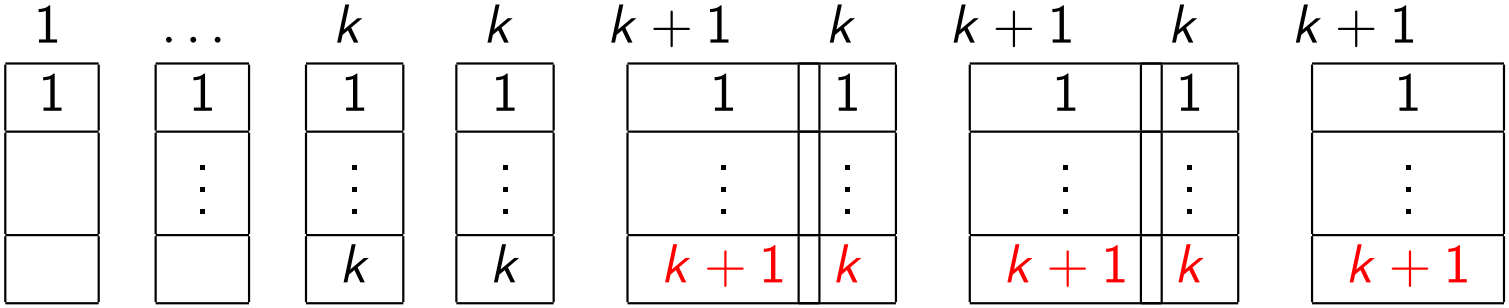
cache :									
	1	1	1	1	④	4	4	4	4
		2	2	2	2	2	2	2	2
			3	3	3	3	3	3	3

Performance de LIFO (Last-In/First-Out)

Lemme

Il existe une séquence de pages demandées σ tel que, pour tout entier pair ℓ, on a $LIFO(\sigma) = \ell \cdot OPT(\sigma)$.

Considérons $\sigma = 1, 2, \ldots, k, (k, k+1)^{\ell/2}$.

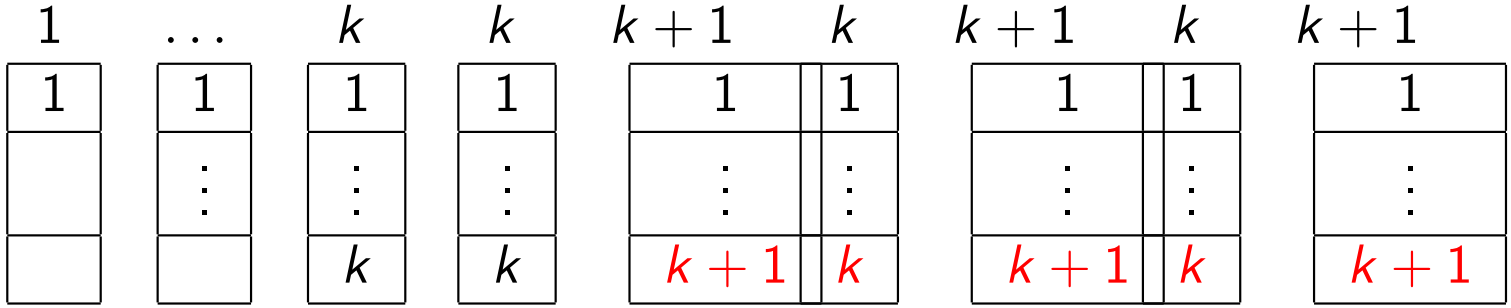


- Observons $OPT(\sigma) = LFD(\sigma) = 1$
  - comme la séquence contient $k+1$ pages différentes il faut au moins qu'une page doit être supprimée.
  - En appliquant LFD, uniquement la page 1 est supprimée lors de l'étape o la page $k+1$ est pour la première fois demandée.

Performance de LIFO (Last-In/First-Out)

Lemme

Il existe une séquence de pages demandées σ tel que, pour tout entier pair ℓ, on a $LIFO(\sigma) = \ell \cdot OPT(\sigma)$.

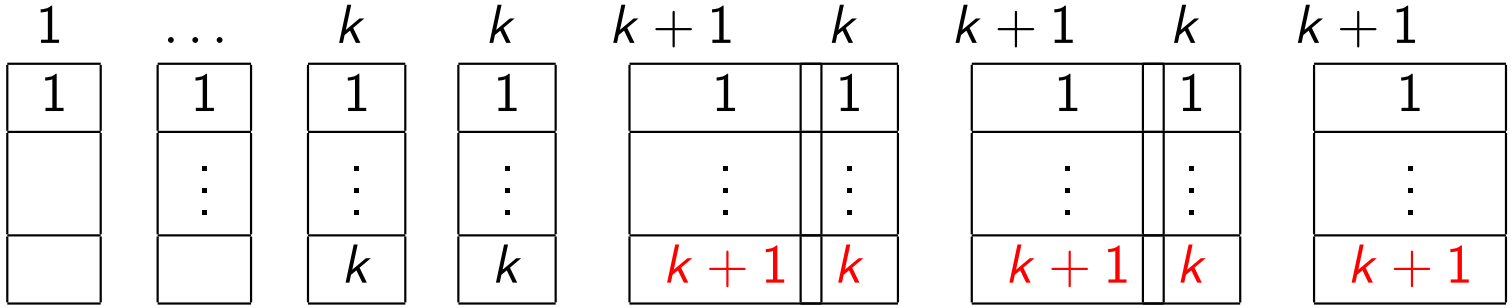


- Observons $OPT(\sigma) = LFD(\sigma) = 1$
- Regardons le comportement de l'algorithme LIFO.
  - Durant les k premières étapes, les pages $1, 2, \ldots, k$ sont chargées dans le cache.

Performance de LIFO (Last-In/First-Out)

Lemme

Il existe une séquence de pages demandées σ tel que, pour tout entier pair ℓ, on a $LIFO(\sigma) = \ell \cdot OPT(\sigma)$.

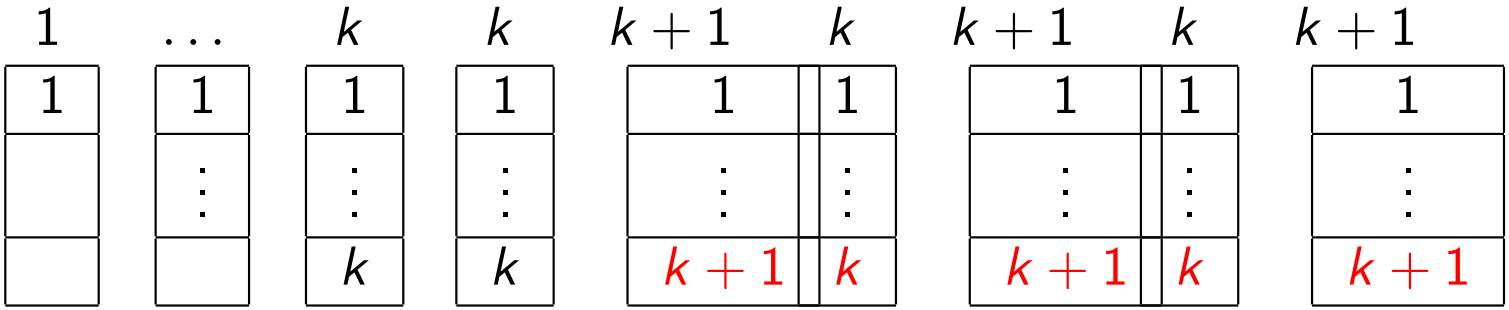


- Observons $OPT(\sigma) = LFD(\sigma) = 1$
- Regardons le comportement de l'algorithme LIFO.
  - Durant les k premières étapes, les pages $1, 2, \ldots, k$ sont chargées dans le cache.
  - A l' étape $k + 1$, l'algorithme doit supprimer une page : k.
  - A l' étape $k + 2$, l'algorithme doit supprimer une page : $k + 1$.

Performance de LIFO (Last-In/First-Out)

Lemme

Il existe une séquence de pages demandées σ tel que, pour tout entier pair ℓ, on a $LIFO(\sigma) = \ell \cdot OPT(\sigma)$.



- Observons $OPT(\sigma) = LFD(\sigma) = 1$
- Regardons le comportement de l'algorithme LIFO.
  - Durant les k premières étapes, les pages $1, 2, \ldots, k$ sont chargées dans le cache.
  - A l' étape $k + 1$, l'algorithme doit supprimer une page : k.
  - A l' étape $k + 2$, l'algorithme doit supprimer une page : $k + 1$.
  - Donc, pour tout $\ell/2 \geq i \geq 1$
    - à l' étape $k + 2i$, l'algorithme doit supprimer une page : $k + 1$.
    - à l' étape $k + 2i - 1$, l'algorithme doit supprimer une page : k.

$$LFD(\sigma) = \ell$$

Performance de LRU (Least-Recently-Used)

Soit une sous-séquence de σ débutant à i et terminant à t contient exactement k différentes pages et que la page demandée à l'étape $t+1$ n'est pas demandée entre les étapes s et t.

étape	...	**i**	**i+1**							**t**
σ :	...	1	1	2	3	2	...	k	...	$k+1$

- l'algorithme optimal doit supprimer au moins une page pour gérer les demandes entre l'étape s et l'étape $t+1$.
  - le cache ne peut contenir que k pages,
  - l'algorithme optimal offline doit avoir au moins une suppression d'une pages après avoir utilisées k pages distinctives.

Performance de LRU (Least-Recently-Used)

Soit une sous-séquence de σ débutant à i et terminant à t contient exactement k différentes pages et que la page demandée à l'étape $t+1$ n'est pas demandée entre les étapes s et t.

étape	...	**i**	**i+1**							**t**
σ :	...	1	1	2	3	2	...	k	...	$k+1$

- l'algorithme optimal doit supprimer au moins une page pour gérer les demandes entre l'étape s et l'étape $t+1$.
- On a $LRU(\sigma) \leq k \cdot OPT(\sigma)$.
  - diviser une séquence de requêtes en M sous-séquences de k.
  - le nombre de suppression par l'algo optimal $\geq M$.
  - au maximum k chargement de pages peuvent avoir lieu dans chaque sous séquence.

    le nombre de suppression par LRU $\leq Mk \leq OPTk$.

Récapitulatif : Algorithmes Online

- **Algorithmes Offline** :

  toutes les données sont disponibles dès le début

- **Algorithmes Online** :

  il faut décider au fur et à mesure de l'arrivée des données

- **Compétitivité** : Un algorithme online est α-compétitif si

  pour toute entrée x, $online(x) \leq \alpha \cdot OPT(x) + cte$