Algorithmique

Amin NAIRI <anairi@esgi.fr>

Séance 1 — Introduction à l’algorithmique

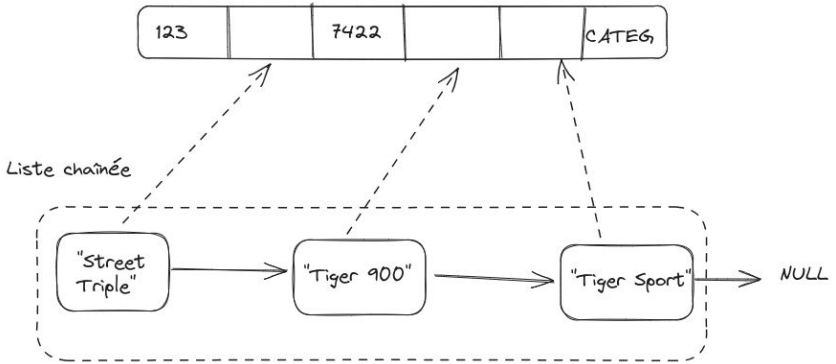
- Présentation du cours, objectifs & attentes
- Introduction aux algorithmes et à leurs importance dans la résolution de problèmes
- Notions de bases : données, variables, instructions, structures de contrôles
- Exemples simples d’algorithmes
- Projet pédagogique

Séance 2 — Structures de données

- Tableaux
- Listes chaînées
- Piles
- Files
- Avantages & inconvénients de chaque structures de données
- Manipulation de structures de données en algorithmique

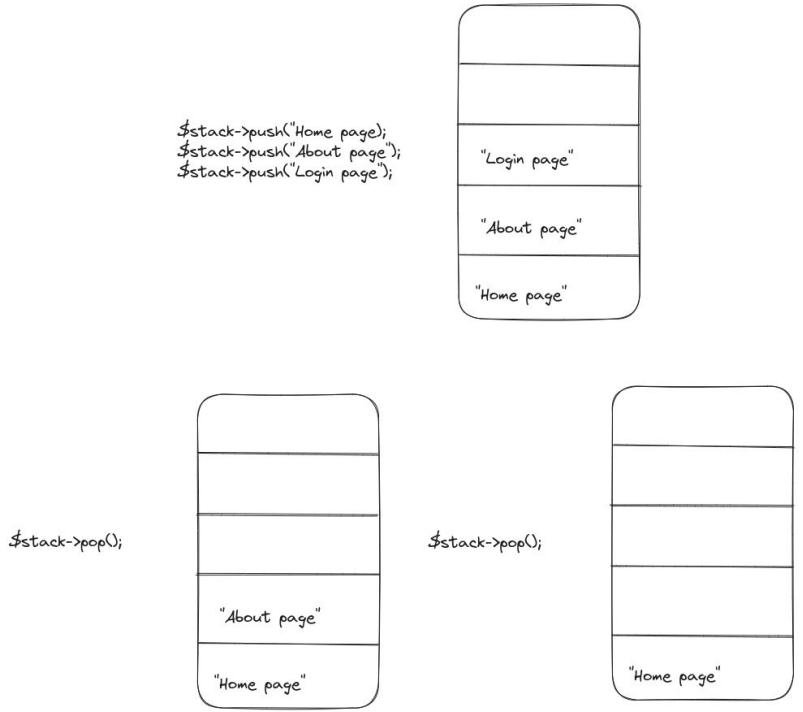
Séance 2 — Liste chaînée

- Les tableaux classiques stockent des données de manière contigüe
- Les Listes Chaînées sont utilisées pour pouvoir stocker des données de manière non-contigüe
- Idéal pour pouvoir optimiser l’espace en mémoire utilisé par une structure de données
- Obligé de se déplacer dans toute la liste pour chercher un élément à un index spécifique, contrairement à un tableau



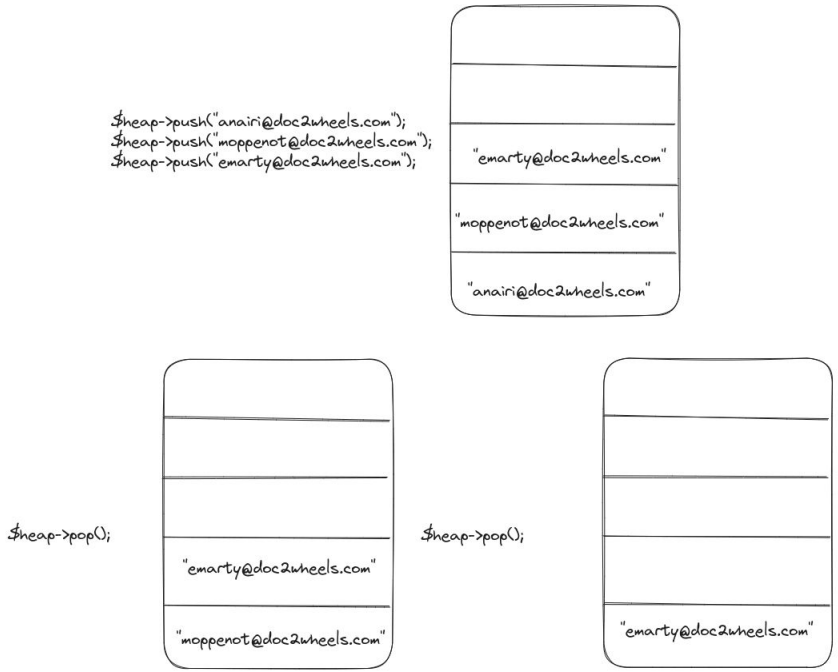
Séance 2 — Pile (Stack)

- Idéal pour stocker des choses qui peuvent être corrigées comme des changements sur un éditeur, ou un historique de navigation
- Le dernier élément inséré est le premier élément sorti (LIFO, Last In First Out)
- Repose sur une liste chaînée dans son implémentation



Séance 2 — File (Heap)

- Idéal pour stocker des queues comme pour pouvoir envoyer des emails de manière asynchrone plus tard par exemple
- Le premier élément inséré est le premier élément sorti (FIFO, First In First Out)
- Repose sur une liste chaînée dans son implémentation



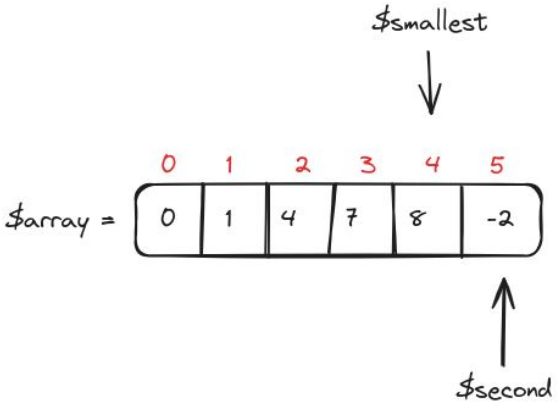
Séance 3 — Algorithmes de recherche et de tri

- Algorithmes de recherche linéaire et binaire
- Algorithmes de tri : sélection, insertion, à bulles
- Complexité des algorithmes de recherche et de tri
- Contrôle continu #1

Séance 3 — Tri par sélection

Algorithme de tri par séléction

$first = 4



Tour de boucle

Est-ce que à chaque tour de boucle $smallest est toujours plus petit que $second (index)

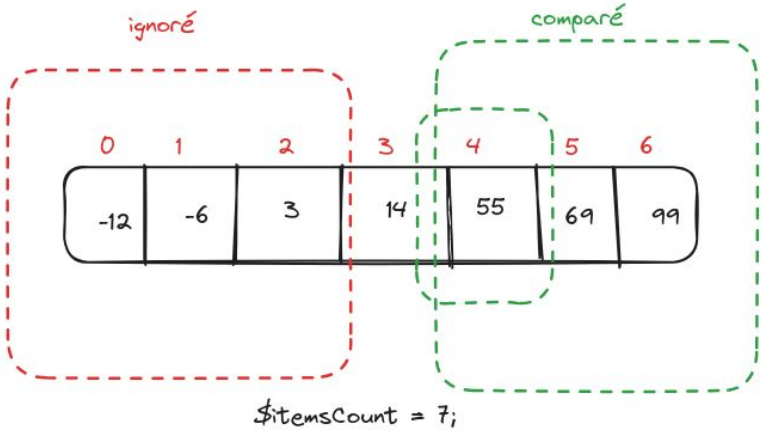
Comparaison

```
if ($array[$second] < $array[$smallest])
  $smallest = $second;
```

Echange

```
$temporary = $array[$first];
$array[$first] = $array[$smallest];
$array[$smallest] = $temporary;
```

Séance 3 — Recherche binaire




```
$itemsCount = 7;

$middleIndex = floor($itemsCount / 2);


$searchValue = 55;

if ($items[$middleIndex] === $searchValue)
    return $middleIndex;
```

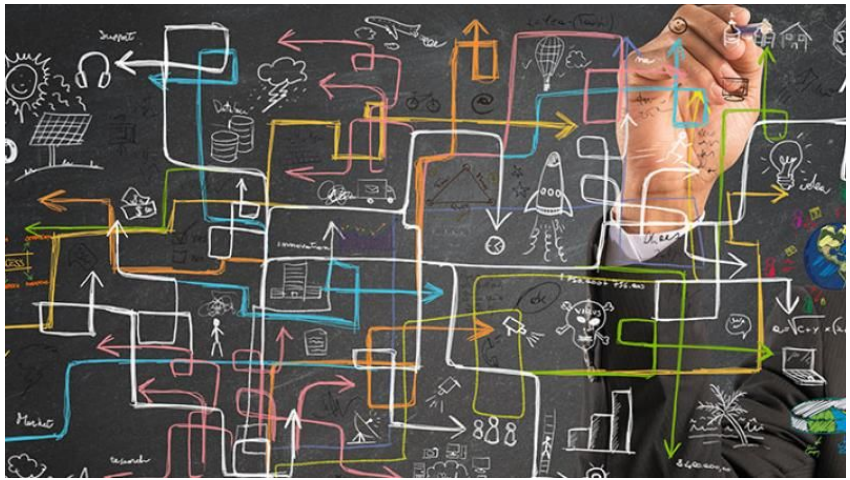
Recherche binaire
« diviser pour mieux régner »

VS

Recherche linéaire
boucle sur tous les éléments

Séance 4 — Complexité algorithmique

- Mesure du temps que prends un algorithme à s’exécuter
- Mesure en terme de temps (de CPU)
- Mesure en terme d’espace (occupée en mémoire-vive)
- Permet d’évaluer l’efficacité de toute ou partie de son application, d’un algorithme, d’une fonction etc…
- Ce n’est ni une mesure absolue, ni une mesure précise car différents facteurs influence le résultat (latence, puissance du CPU, rapidité de la RAM, etc…)
- Notation en O(n), n étant le nombre d’éléments d’un tableau par exemple



Séance 4 — Complexité algorithmique : constante

- Se note en O(1)
- Quantité de mémoire ou temps de CPU constant
- Pour un tableau de 1M d’éléments, il faudra parcourir le tableau 1 seule fois
- Par exemple, accéder à un index précis d’un tableau

Séance 4 — Complexité algorithmique : linéaire

- Se note en O(n)
- Quantité de mémoire ou temps de CPU qui évolue de paire avec le nombre d’éléments d’un tableau à parcourir
- Pour un tableau de 1M d’éléments, il faudra parcourir le tableau 1M de fois
- Par exemple, parcourir avec une boucle for…of un tableau

Séance 4 — Complexité algorithmique : logarithmique

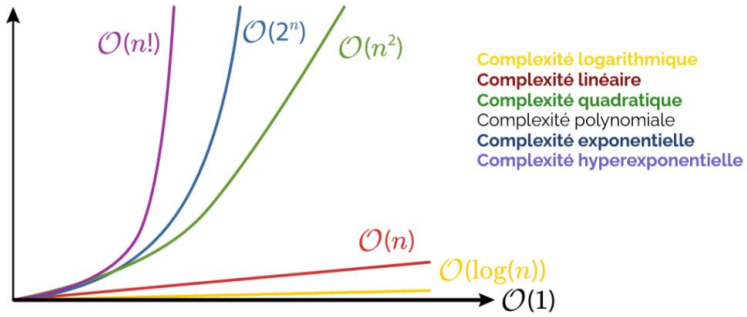
- Se note en O(log n)
- Quantité de mémoire ou temps de CPU qui n’évolue un peu moins qu’avec le nombre d’éléments d’un tableau à parcourir
- Pour un tableau de 1M d’éléments, il faudra parcourir environ 7 fois (valeur arbitraire)
- Par exemple, utiliser un algorithme de recherche binaire pour trouver un élément dans un tableau

Séance 4 — Complexité algorithmique : quadratique

- Se note en $O(n^2)$
- Quantité de mémoire ou temps de CPU qui évolue avec le carré du nombre d’éléments d’un tableau à parcourir
- Pour un tableau de 1M d’éléments, il faudra parcourir environ $1M^2$
- Par exemple, utiliser un algorithme de tri à bulle qui va utiliser une première boucle pour parcourir tous les éléments, et une deuxième boucle pour parcourir de nouveau tous les éléments afin d’échanger le plus petit et le plus grand

Séance 4 — Complexité algorithmique : autres

- Il existe beaucoup d’autres types de complexité algorithmique
- L’idée n’est pas de tout savoir
- Ni de maîtriser des concepts mathématiques
- L’idée est de savoir donner un ordre de grandeur de la complexité d’un algorithme



Séance 4 — Exercice

- Créer une classe ArrayUtils
- Créer une méthode map qui permet de transformer chaque élément d’un tableau à partir d’une fonction de transformation donnée en argument
- Créer une méthode filter qui permet de retirer des éléments d’un tableau en fonction d’une fonction de filtre
- Donner pour chacune des méthodes la complexité algorithmique