Présentation

Selon le dictionnaire, « trier » signifie « répartir en plusieurs classes selon certains critères ». Le terme de « tri » en algorithmique est très souvent attaché au processus de classement d'un ensemble d'éléments dans un ordre donné. Par exemple, trier N entiers dans l'ordre croissant, ou N noms dans l'ordre alphabétique. Tout ensemble muni d'un ordre total peut fournir une suite d'éléments à trier.

Nous allons étudier ici deux algorithmes de tri que vous devez connaître (tri par insertion et tri par sélection).

Pré requis

- Les listes (tableaux) – Première NSI
- Algorithmique – Première NSI (cf pixees.fr)
- Notion de coût – Première NSI

Rappel BO

Contenus	Capacités attendues	Commentaires
Tris par insertion, par sélection	Écrire un algorithme de tri. Décrire un invariant de boucle qui prouve la correction des tris par insertion, par sélection	La terminaison de ces algorithmes est à justifier On montre que le coût est quadratique dans le pire des cas.

Objectifs

- Reconnaître les différents types de tri et implémenter un algorithme sur ceux-ci - Vous devez être capable d'en donner le principe ainsi que d'en évaluer la complexité.

Durée

Activité d'une séance de deux heures (à compléter, individuelle)

La suite ?

Nous verrons la notion de complexité, coût. La recherche dichotomique et les algo gloutons.

Un peu d'histoire

Une méthode de tri, appellée Quicksort, inventée par Sir Charles Antony Richard **Hoare** en 1961 et fondée sur la méthode de conception « diviser pour régner ». Il peut être implémenté sur un tableau ou sur des listes ; son utilisation la plus répandue concerne tout de même les tableaux.

Concepts

Examinez les méthodes de tri par sélection et par insertion https://isnbreizh.fr/nsi/activity/algoRefTri/xSortLab/xSortLab.html (sélectionner le tri dans le menu, puis cocher « Fast » et enfin cliquer sur « start »)

➔ Noter le nombre de comparaison et de copies et commentez

Insertion	Selection
.....	.....
.....	.....
.....	.....
.....	.....

- Commentaires

.....

Réaliser la simulation également pour le temps « Timed Sort » et compléter le tableau suivant :

Insertion	Selection
.....	.....
.....	.....
.....	.....

- Commentaires

.....

Tri par sélection

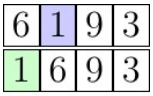
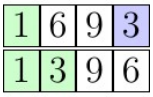
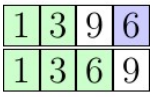
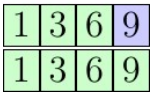
Le tri par sélection se décompose en deux étapes :

- Sélectionner un élément (d'où son nom).
- Le placer à sa bonne place.

Le facteur qui détermine si un élément est bien placé est son rang (par exemple : le *ième* plus petit élément sera forcément placé en *ième* position du tableau). Le tri par sélection va donc à chaque tour trouver le *ième* plus petit élément du tableau, pour ensuite l'insérer à sa place, en commençant par le premier plus petit, et en augmentant à chaque fois (deuxième plus petit, troisième, etc.).

Exemple

Prenons désormais comme exemple la suite de nombres suivante : 6, 1, 9, 3. Trions cette suite avec l'algorithme du tri par sélection dans l'ordre croissant :

1er tour :	6, **1**, 9, 3 -> le plus petit élément du tableau est 1, on le place donc sur la première case (en l'échangeant avec le 6).	
2ème tour :	1, 6, 9, **3** -> le deuxième plus petit élément est 3, on le place sur la deuxième case et on l'échange avec le 6.	
3ème tour :	1, 3, 9, **6** -> le troisième plus petit élément est 6, on l'échange avec 9 pour le placer sur la troisième case.	
4ème tour :	1, 3, 6, **9** -> le quatrième plus petit élément du tableau est 9, il est déjà en quatrième position on ne fait rien.	

À chaque tour, on cherche le minimum dans l'espace non trié du tableau (le minimum est représenté en bleu, et la partie non triée en blanc), ensuite on déplace cet élément à sa place définitive (représentée en vert). En faisant cela pour chaque élément du tableau, ce dernier se retrouve trié au bout de N tours maximum (N étant la taille du tableau).

Pseudo-code

Le pseudo-code du tri par sélection est simple :

```
triSelection :

   Pour chaque élément
      Pour chaque élément de la partie non triée
         Mettre à jour le minimum du tableau rencontré jusqu'ici
      Échanger l'élément actuel avec le minimum
```

Complexité

Le tri par sélection a une complexité en $O(N^2)$:

- La première boucle parcourt N tours.
- La deuxième boucle parcourt $N-i$ tours (i variant de 0 à N).

Sa complexité est donc légèrement inférieure à N^2, cependant cette différence est mineure et sa complexité est considérée comme étant en $O(N^2)$.

Conclusion

Le tri par sélection est donc un algorithme assez simple, mais peu efficace à cause de sa complexité en $O(N^2)$. Cependant des améliorations et des variantes permettent de le rendre plus rapide, et le tri par sélection sert de base à d'autres algorithmes plus efficaces que nous étudierons plus tard.

Tri par insertion

Le tri par insertion (*insertion sort* en anglais) est un algorithme de tri par comparaison simple, et intuitif mais toujours avec une complexité en $O(N^2)$. Vous l'avez sans doute déjà utilisé sans même vous en rendre compte : lorsque vous triez des cartes par exemple. C'est un algorithme de tri stable, en place, et le plus rapide en pratique sur une entrée de petite taille.

Principe de l'algorithme

Le principe du tri par insertion est de trier les éléments du tableau comme avec des cartes :

- On prend nos cartes mélangées dans notre main.
- On crée deux ensembles de carte, l'un correspond à l'ensemble de carte triée, l'autre contient l'ensemble des cartes restantes (non triées).
- On prend au fur et à mesure, une carte dans l'ensemble non trié et on l'insère à sa bonne place dans l'ensemble de carte triée.
- On répète cette opération tant qu'il y a des cartes dans l'ensemble non trié.

Exemple

Prenons comme exemple la suite de nombre suivante : 9, 2, 7, 1 que l'on veut trier en ordre croissant avec l'algorithme du tri par insertion :

1er tour :

9 | **2**, 7, 1 -> à gauche la partie triée du tableau (le premier élément est considéré comme trié puisqu'il est seul dans cette partie), à droite la partie non triée. On prend le premier élément de la partie non triée, 2, et on l'insère à sa place dans la partie triée, c'est-à-dire à gauche de 9.

2ème tour :

2, 9 | **7**, 1 -> on prend 7, et on le place entre 2 et 9 dans la partie triée.

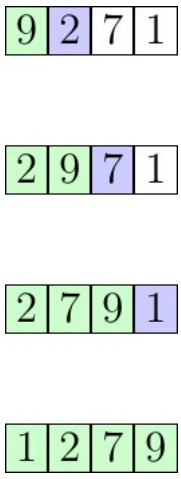
3ème tour :

2, 7, 9 | **1** -> on continue avec 1 que l'on place au début de la première partie.

1, 2, 7, 9

Pour insérer un élément dans la partie triée, on parcourt de droite à gauche tant que l'élément est plus grand que celui que l'on souhaite insérer.

Pour résumer l'idée de l'algorithme :



La partie verte du tableau est la partie triée, l'élément en bleu est le prochain élément non trié à placer et la partie blanche est la partie non triée.

Pseudo-code

```
triInsertion :

   Pour chaque élément non trié du tableau
      Décaler vers la droite dans la partie triée, les éléments supérieurs
à
      celui que l'on souhaite insérer
      Placer notre élément à sa place dans le trou ainsi créé
```

Complexité

L'algorithme du tri par insertion a une complexité de $O(N^2)$:

- La première boucle parcourt $N-1$ tours, ici on notera plutôt N tours car le -1 n'est pas très important.
- Décaler les éléments de la partie triée prend i tours (avec i variant de 0 à N).

Dans le pire des cas on parcourt N^2 tours, donc le tri par insertion a une complexité en temps de $O(N^2)$.

Conclusion

L'algorithme du tri par insertion est simple et relativement intuitif, même s'il a une complexité en temps quadratique. Cet algorithme de tri reste très utilisé à cause de ses facultés à s'exécuter en temps quasi linéaire sur des entrées déjà triées, et de manière très efficace sur de petites entrées en général.

Activité

Implémentation

Votre travail consiste à implémenter en python ces deux méthodes de tri, d'après les pseudo codes fournis, via des fonction nommées « tri_insertion » et « tri_selection ».

Vous devez déposer ces algorithmes dans votre dépôt Git (ou Repl.it) et envoyer le lien. (vous prendrez soins d'appeler ces fonctions, et d'afficher le résultat de la liste triée)

Indiquer vos performances (en temps) sur des listes de 10 et 100 éléments

	Insertion	**Selection**
10	.....	.....
100	.....	.....

Utilisez le module time

Pour aller plus loin

http://lwh.free.fr/pages/algo/tri/tri.htm https://replit.com/@jesus64bits/Illustration-tris
Nos algorithmes pourraient-ils être BEAUCOUP plus rapides ? (P=NP ?), Science étonnante, 2020

Bilan

A vous de l'écrire pour la prochaine fois

Table des matières

Sources :

- Pixees (rappels de première NSI)
- *Time complexity sur le wiki Python* (Complexité)
- DU ES NSI – UCO Angers
- http://www.apprendre-en-ligne.net
- Wikipédia : Algorithmique
- Wikipédia : Algorithme de tri
-