Tri par séléction

Prérequis :

- Manipulations des listes en Pyhton. Bases de programmation Python.
- Algorithme de recherche dichotomique, pour la notion de complexité.

Objectifs pour les élèves :

- Savoir : Algorithme de référence : tri par sélection.
  Conformément au programme, l'objectif est une compréhension de cet algorithme et la capacité à le mettre en oeuvre. Dans le cadre de cette activé, la mise en oeuvre se fera à l'aide d'une implémentation en Python, puis d'un mini-projet.
- Les différentes capacités, ou observations, seront indiquées dans les notes en bas de page.

1 Principe

Exercice 1 : Trier à la main, par ordre croissant, la liste de nombres : [39, 77, 30, 43, 29, 58, 94, 67, 74, 39, 74, 65, 36, 47, 60, 100, 95, 5, 78, 59]. Décrire l'algorithme utilisé. [1]

Nous sommes souvent amené à trier des données : des notes, des noms, des photos par date,...

Il existe plusieurs algorithmes de tris utilisés en informatique, chacun présentant des avantages et inconvénients, et plus ou moins adaptés suivant le type de données.

Exercice 2 : En faisant une recherche sur Internet, citer 5 algorithmes de tri[2].

Parmi tous les algorithmes de tris existants, le *tri par sélection*, en plus d'être un des algorithmes de référence au programme, est l'un des plus facile à implémenter. Il fonctionne avec toutes les quantités comparables (caractères, flottants, structures, etc...), mais pour cette activité nous allons l'utiliser avec des nombres.

Algorithme de tri par sélection : Considérons une liste de nombres : liste = [6, 2, 8, 1, 4, 3, 7, 9, 5, 0]

1. Étape 1 :
   - On recherche le plus petit élément de cette liste. Ici c'est 0, en position d'indice 9.
   - On échange cet élément avec le premier élément de la liste.
     On obtient : liste=[**0**, 2, 8, 1, 4, 3, 7, 9, 5, **6**]
   - Le premier élément est donc le plus petit de la liste. On continue alors avec la même liste, en ignorant son premier élément.
     C'est-à-dire : liste = [0, 2, 8, 1, 4, 3, 7, 9, 5, 6]

   > Toute l'astuce de l'algorithme est là :on ignore volontairement dans la suite du traitement tous les éléments que l'on a placé en début de liste.

2. Étape 2 : liste = [0, **1**, 8, **2**, 4, 3, 7, 9, 5, 6]
3. Étape 3 : liste = [0, 1, **2**, **8**, 4, 3, 7, 9, 5, 6]
4. etc... On sait que notre liste est triée lorsque le nombre d'éléments non triés est égal à 1.

Exercice 3 : Terminer l'exemple[3]. En combien d'étapes la liste est-elle triée ?

Exercice 4 : Appliquer l'algorithme de tri par sélection à la liste [10, 8, 6, 4, 2, 0], puis à la liste [0, 1, 2, 3, 4, 5] (!). Combien d'étapes pour chaque liste ? Que peut-on en penser[4] ?

2 Implémentation en Python

Maintenant que vous connaissez l'algorithme et que vous avez vu sur quelques exemples son fonctionnement, nous pouvons passer à son implémentation [5]!

Mais avant cela, on remarque qu'il est possible de décomposer l'algorithme en plusieurs " sous-fonctions ", ce qui facilitera le travail :

- La recherche du plus petit élément.
- L'échange de deux éléments.
- La réalisation du tri.

1. Capacité : Concevoir un algorithme
2. Compétence : conduire des recherches documentaires.
3. Capacité : Comprendre un algorithme
4. Capacité : S'interroger sur l'efficacité d'un algorithme
5. Capacité : Programmer un algorithme

2.1 Recherche du plus petit élément

Exercice 5 : Écrire une fonction *indice_min_liste* qui prend en entrée une liste de nombres, et qui renvoie *l'indice* de son plus petit élément[6].

2.2 Échanger deux éléments d'une liste

Exercice 6 : Écrire une fonction *echange_elements* qui prend en entrée une liste, deux indices i et j, et qui modifie la liste de la manière suivante : les éléments d'indices i et j sont échangés.

2.3 Le tri

Exercice 7 : En utilisant les fonctions précédentes, écrire une fonction *tri_selection* qui applique l'algorithme de tri par sélection[7].

3 Complexité du tri par sélection

Nous avons déjà abordé la notion de complexité lors du TP sur la recherche dichotomique.
Exercice 8 : Charger le script *complexite_tri_selection.py*. Ce script propose une autre implémentation du tri par sélection, mais surtout permet à chaque appel d'afficher le nombre total de comparaisons effectuées dans les boucles utilisées par la fonction triSelection.

Tester la fonction triSelection sur trois listes de longueur 5 : la première liste déjà triée, la deuxième triée par ordre décroissant, et la troisième "au hasard". Quel est le cas le plus favorable[8] ?

Exercice 9 : On va effectuer des tests sur des listes de tailles différentes. Dans le script précédent, écrire une fonction *test_tri* qui prend en entrée un entier n, puis :

- génère une liste1 d'entiers de 0 à $n-1$.
- génère une liste2 d'entiers de $n-1$ à 0. (triée dans l'ordre décroissant)
- génère une liste3 de n entiers pris au hasard entre 0 et 100. On pourra utiliser la fonction *randint* du module **random**.
- Applique la fonction *triSelection* à chacune des liste.

Effectuer quelques tests. Que peut-on penser des résultats ?
Exercice 10 : Remplir un tableau de valeur du nombre de comparaisons en fonction de n, pour n variant de 1 à 10. A l'aide de la calculatrice, ou d'un tableur, visualiser le nuage de points. Quelle semble être la nature de la courbe ?
Exercice 11 : En notant n la taille de la liste, déterminer en fonction de n le nombre de comparaisons nécessaires à l'exécution de la fonction triSelection[9].

4 Miniprojet

Dans un jeu de 32 cartes, une carte peut être modélisée par une liste de deux éléments, par exemple ["Trèfle", "As"]. Un jeu de carte mélangé est donc une liste de 32 "cartes". Par exemple jeu = [["Trèfle", "As"], ["Pique", "8"], ["Pique", "Valet"], ["Trèfle", "7"], ["Coeur", "As"], $\cdots$].

En binôme ou en trinôme[10], écrire un script qui "mélange" un jeu de cartes au hasard, puis trie le jeu en utilisant le principe du tri par sélection, et suivant l'ordre :

- Trèfle < Carreau < Coeur < Pique pour les couleurs
- 7<8<9<10<Valet<Dame<Roi<As pour les valeurs.

En complément, on pourra tenter de faire une distribution de 8 cartes à 4 joueurs, puis trier la "main" de chacun. On pourrait même imaginer utiliser les règles de la Belote : 7-8-9-V-D-R-10-As pour les couleurs normales et 7-8-D-R-10-As-9-V pour l'atout.

6. Capacités : Concevoir l'entête (ou l'interface) d'une fonction, puis la fonction elle-même. Mettre un programme au point en le testant
7. Capacités : comprendre un algorithme, puis le programmer
8. On peut conjecturer ici l'indépendance du nombre de comparaisons par rapport à la nature triée ou non de la liste. Particularité du tri par sélection : la complexité ne dépend que de la taille des données. Pas de complexité dans le "meilleur des cas" ou dans le "pire des cas"
9. On tombe sur une complexité quadratique : $\frac{n(n-1)}{2}$ donc $O(n^2)$.
10. Paragraphe 3 du programme ISN 2017 sur les projets. La principale difficulté va consister à "comparer" deux cartes. Il sera nécessaire de guider les élèves vers l'écriture d'une fonction de comparaison retournant un booléen. Ce miniprojet pourra être réutilisé pour l'algorithme de tri par fusion.