Feuille de TP Python 03

Algorithmes de tri

1 Principes de tri, pourquoi trier

1.1 Pourquoi trier ?

Le tri de données et d'informations est une des opérations les plus fondamentales de l'informatique.

On effectue un tri sur un ensemble de données pour pouvoir accéder plus rapidement à certaines caractéristiques de cet ensemble, pour pouvoir sélectionner les données les plus pertinentes ou éliminer les moins pertinentes

1. Une recherche GOOGLE retourne une liste de liens triés par ordre décroissant de pertinence : les difficultés sont multiples dans ce cadre : donner un critère de pertinence, ordonner une liste gigantesque de sites plus ou moins pertinents, etc...
2. En statistiques : ayant une suite de nombres issus par exemple de mesures, on les trie pour accéder à des caractéristiques statistiques importantes de la suite : médiane, quartiles, déciles et éliminer les *outliers*, les données aberrantes
3. En algorithmique : certains algorithmes nécessitent que les données soient ordonnées en entrée ou incluent des tris à l'intérieur même de l'algorithme.

1.2 Clés de tri

Pour trier en ordre croissant ou décroissant une liste d'informations, il faut disposer d'un critère de comparaison de ces informations, *i.e.* disposer d'un ordre.

Certains types de données disposent d'un ordre naturel (les nombres entiers ou réels), pour trier une liste de nombres, on peut donc se baser sur cet ordre pour effectuer le tri. Dans ce cas, on dispose d'un vecteur (ou suite finie) T de n nombres réels et on cherche à le mettre par ordre croissant (ou décroissant).

Les chaines de caractères possèdent elles aussi un ordre naturel, l'ordre *lexicographique*. Il y a cependant toujours des embrouilles au niveau de la différence majuscule/Minuscule, caractères accentués ou autres caractères spéciaux (!, ?,...). Avant de faire confiance à un tri sur des chaines de caractères, il vaut mieux tester l'ordonnancement réel des caractères.

D'autres types de données n'ont pas d'ordre naturel ou objectif et il faut alors disposer d'un critère pour effectuer l'ordonnancement. Dans ce cas on dispose d'un tableau D ou d'une liste finie de données $D = (d_k)$

Une méthode naturelle (mais pas la seule) est d'associer à chaque donnée d_k un nombre entier ou réel t_k, sa *clé*.

Pour trier D, il suffit alors de trier $T = (t_k)$ et de garder trace de la façon dont les indices ont été déplacés.

De façon alternative, Python permettant l'accouplement des données, on dispose d'une suite TD de couples de la forme (nombre réel,donnée associée) et on cherche à trier cette suite de couples avec comme « ordre » sur les couples

$$(t_1, d_1) \leq (t_2, d_2) \Leftrightarrow t_1 \leq t_2$$

1.3 Python sait naturellement trier

Le langage Python est à base de listes (qui sont, au niveau primitif, des tableaux contigus en mémoire) et une méthode de tri [1] de ces listes est fournie le langage.

Dans toutes les applications que vous aurez à construire–votre projet final par exemple– si vous avez besoin d'un tri, utilisez celui-ci qui est très optimisé.

Exercice 1.— Utiliser le script `tri-builtin.py`.

Analyser ce script : quelles sont les commandes effectuant des tris ? Comment s'effectue un tri avec une clé ?

Exercice 2.—

1. Dans un script propre. Ecrire une fonction Python `Melange(L)` qui mélange la liste `L`. L'algorithme est le suivant

- Faire une copie `Lc` de `L` grâce à la méthode Python `L.copy()`
- Vider la liste `L` avec la méthode `L.clear()`. Attention ! `L=[]` ne fonctionne pas !
- Tend que la liste `Lc` n'est pas vide, tirer au sort un élément de `Lc` et le placer au bout de la liste `L`. La méthode de liste `L.pop(i)` permet d'extraire l'élément numéro i de la liste `L`.

1. `timsort`, *c.f.* `http://en.wikipedia.org/wiki/Timsort`

2. Construire une liste triable `A` quelconque d'au moins 10 éléments et ensuite appliquer l'extrait de code suivant, après importation du module `time`.

```
instant_dep = time.time() #demarrage mesure
Melange(A)
instant_arr = time.time() #fin mesure
print("Le mélange de la liste A a mis", instant_arr-instant_dep,"s")
#print("Début de la liste mélangée", A[:10])
instant_dep = time.time() #demarrage mesure
A.sort()
instant_arr = time.time() #fin mesure
print("Le tri de la liste A a mis", instant_arr-instant_dep,"s")
#print("Début de la liste triée", A[:10])
```

Il s'agit d'une méthode pour mesurer le temps d'éxécution de l'algorithme. Regarder, dans la correction, une autre méthode de mélange demandant deux fois moins de place mémoire.

1.4 Aspects algorithmiques

L'importance de comprendre les diverses méthodes de tri n'est donc pas « pratique ». Il s'agit pour nous essentiellement

1. de toucher à de l'algorithmique élémentaire,
2. de saisir les contraintes/questions qui se posent lors de l'élaboration d'un algorithme : complexité en temps (nombre d'opérations), complexité en espace (taille mémoire nécessaire à la complétion de l'opération)

Les algorithmes de tri (un *tri* pour simplifier) peuvent avoir (ou pas) un certain nombre de caractéristiques. A titre d'exemples

— Le tri est dit *en place* s'il travaille à l'intérieur du tableau sans besoin de le dupliquer. L'espace mémoire supplémentaire pour stocker les objets reste réduit et l'algorithme est de faible *complexité spatiale*.

— Le tri est dit *stable* s'il préserve l'ordre d'apparition des objets en cas d'égalité des clés. Le tri fourni par Python est stable. On faire une cascade de tris portant sur des clés différentes en conservant l'ordre obtenu précédemment en cas d'égalité.

— Enfin, une question importante est celle de la *complexité temporelle* de l'algorithme, *i.e.* une estimation du nombre d'opérations à faire en fonction du nombre n de données. Les tris présentés en première année sont en $O\left(n^2\right)$. Il existe de meilleurs algorithmes : tri de SHELL en $O\left(n.(\ln n)^2\right)$, *quicksort*, tri par fusion, qui sont en $O\left(n.\ln n\right)$.
Malheureusement, leur implémentation naturelle s'effectue souvent à l'aide d'une notion à la limite du programme, la récursivité. Nous verrons donc d'abord deux types d'implémentations de *quicksort*, la naturelle, récursive et une autre, moins naturelle donc, itérative et ensuite l'implémentation récursive du tri par fusion.

2 Différents algorithmes

2.1 Trouver un plus grand/plus petit élément dans une liste

Cette question n'est pas à proprement parler une question de tri.

Exercice 3.—
1. On se donne un tableau T de nombres réels indicé de 1 à N. Ecrire, en pseudo-code ou en Python, une fonction `Max(T)` retournant l'élément maximal du tableau T.
2. Ecrire, en pseudo-code ou en Python, une fonction `Max2(T)` retournant l'élément maximal du tableau T et le premier indice où celui-ci est atteint.
3. On se donne un tableau T à deux dimensions de nombres réels indicé par les couples de $\{1,\dots,N\}\times\{1,\dots,P\}$. Ecrire, en pseudo-code ou en Python, une fonction `MinMax(T)` retournant le minimum des maxima de chaque ligne.

2.2 Intercaler deux listes triées.

Exercice 4.— Dans un script nommé `trifusion.py`, écrire une fonction Python `Fusion0(T,S)` qui fusionne deux tableaux d'entiers ordonnés `T` et `S` en un seul tableau ordonné et retourne ce tableau.

2.3 Tri par insertion

Le tri par insertion consiste à *insérer* les éléments de la suite les uns après les autres dans une suite triée initialement vide. Lorsque la suite est stockée dans un tableau, la suite triée en construction est stockée au début du tableau. Il s'agit de la méthode la plus communément utilisée pour trier une main lorsque l'on joue aux cartes. Voici un algorithme en pseudo-code effectuant cette opération.

Données: $T = (T[k])_{1\leq k\leq n}$ est un tableau de réels, $n \geq 1$.
Résultat: Les éléments de T sont ordonnés par ordre croissant.
début
  pour $i = 2$ *à* n **faire**
    $j := i, v := T[i]$;
    tant que $j > 1$ *et* $v < T[j-1]$ **faire**
      $T[j] := T[j-1]$;
      $j := j-1$;
      #imprimer(T);
    $T[j] := v$;
    #imprimer(T);

Exercice 5.—
1. En prenant la petite liste $T = [4,2,3,1,2]$ ou autre liste de votre choix indiquer la sortie écran de cet algorithme en la calculant à la main.
2. Implémenter[2] cet algorithme en Python, dans une fonction `triinsertion` prenant en argument le tableau à trier, pour qu'il ordonne le tableau de couples (donnée,clé)

```
DT=[('spirou',4),('fantasio',2),('spip',3),('marsupilami',1),('champignac',2)]
```

3. En une ligne supplémentaire de code, imprimer à l'écran la(les) médianes de ce tableau.
4. Cet algorithme est-il *en place*. ? Quel est le nombre d'opérations effectuées dans le pire cas ? Cet algorithme est-il *stable* ?
5. Modifier le code de `triinsertion` pour qu'il compte le nombre d'opérations effectuées et l'imprime au bout du compte.

2.4 Tri à bulles.

Le tri à bulles ou tri par propagation est un algorithme de tri qui consiste à faire remonter progressivement les plus grands éléments d'un tableau, comme les bulles d'air remontent à la surface d'un liquide.

L'algorithme parcourt le tableau, et compare les éléments (des couples) successifs. Lorsque deux éléments successifs ne sont pas dans l'ordre croissant, ils sont échangés. Après chaque parcours complet du tableau, l'algorithme recommence l'opération. Lorsqu'aucun échange n'a lieu pendant un parcours, cela signifie que le tableau est trié. On arrête alors l'algorithme.

Pour les informaticiens, il s'agit de la plus mauvaise méthode de tri sur le marché, on ne la détaille donc pas.

3 Algorithmes récursifs

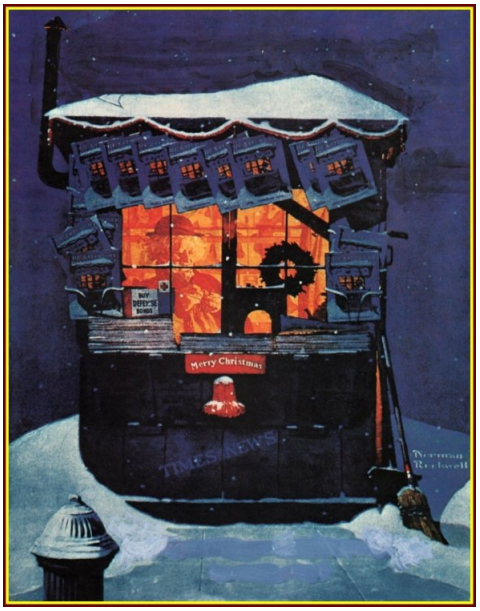
3.1 Une idée de la récursivité

Une fonction est dite *récursive* si sa définition utilise la fonction à définir elle-même, mais sur des données plus « petites ». L'exemple le plus bateau est celui de la fonction factorielle qui pourrait se définir en Python par

```
def factorielle(n) :
    if n==0 :
        return 1
    else :
        return factorielle(n-1)*n
```

Il s'agit d'une définition « mise en abyme[a] ». Il doit être clair que pour que le processus de définition soit valable, cette mise en abyme s'arrête à un moment et donc que, sur les données « minimales », la fonction donne une réponse directe.

a. c.f. `http://fr.wikipedia.org/wiki/Mise_en_abyme`



Norman Rockwell, Couverture du Saturday Evening Post, 20 décembre 1941

2. Créer un fichier `triselementaires.py`

3.2 Tri rapide `Quicksort`

3.2.1 L'algorithme

L'algorithme `Quicksort` est un algorithme qui se décrit simplement de manière récursive.
Soit T notre tableau, indicé [3] de ℓ à h, à trier.

1. La première étape est l'étape de *partition* : On se donne un élément π du tableau : le *pivot* de la méthode et on déplace les éléments du tableau T de sorte que à gauche du pivot se trouvent les éléments inférieurs (ou égaux) à π et qu'à droite se trouvent les éléments supérieurs (strictement) à π. Dans cette nouvelle configuration, l'élement π reçoit l'indice p.
2. La deuxième étape est l'étape de récursion. On trie (par la même méthode) d'une part la partie gauche, située entre les indices ℓ et $p-1$, du tableau T et d'autre part, la partie droite, située entre les indices $p+1$ et h. Si l'une de ces parties est vide ou ne contient qu'un élément, elle n'a pas besoin d'être triée.

3.2.2 Travail à faire

1. Charger le script `quicksort.py`. Effacer les corps des fonctions `partition0` et `quickSort`
2. Ecrire, dans le script, la fonction Python `partition0(T,l,h)` où T est un tableau qui implémente la première étape entre les indices `l` (attention ! cette lettre est le ℓ !) et `h-1` en prenant comme pivot l'élément `T[h-1]`. Cette fonction doit
   (a) modifier le tableau `T`
   (b) retourner l'indice de pivot `p`
3. Ecrire, en suivant le principe de la deuxième étape, une fonction Python `quickSort(T,l,h)` où `T` est le tableau à trier, entre les indices (au sens large) `l` et `h-1`. Par défaut `l` vaut 0 et `h` vaut `len(T)` de sorte que l'appel `quickSort(T)` trie tout le tableau `T`.
4. Tester
5. Observer le corps de la fonction Python `partition(T,l,h)`. Quel élément est choisi [4] comme pivot ?.

3.2.3 Eliminer la récursivité ?

La notion de récursivité, *i.e.* le fait qu'une fonction puisse s'appeler elle-même sur des données plus « petites », est en marge du programme. On peut toujours écrire une version non-récursive d'une fonction en s'appuyant sur le concept de *pile*.
Imaginons que vous devez faire un travail de bibliographie sur un article de biologie et donc « ficher » ou résumer l'article lui-même ainsi que les articles auxquels il fait référence.
Une méthode pour organiser son travail est la suivante.

1. Vous disposez d'un emplacement où vous allez déposer des fiches décrivant le travail restant à faire. Cet emplacement contient initialement la fiche « lire l'article A ainsi que les articles y étant référencés ». Les règles sont les suivantes :
   (a) lorsque vous lisez un article, à chaque référence bibliographique A' rencontrée, si vous ne l'avez pas déjà traitée, vous déposez une fiche sur votre pile décrivant le travail à faire à propos de cette nouvelle référence : « lire l'article A' ainsi que... »..
   (b) lorsque vous avez fini de lire un article, vous prenez la fiche sur le dessus de la pile et vous la traitez.
2. Le travail sera fini lorsque votre pile de fiches sera vide.

3.3 Tri par fusion (`mergesort`)

3.3.1 Algorithme

Le *tri fusion*, que l'on peut pratiquer à la main lorsque l'on doit classer des fiches par ordre alphabétique est l'algorithme suivant

1. Couper le tas en deux tas de tailles comparables
2. Trier chacun des tas (Etape récursive, on utilise l'algo. lui même pour faire cette étape, sauf si le tas est réduit à une fiche)
3. Fusionner les deux tas ordonnés en les intercalant de manière à obtenir un tas ordonné.

Exercice 6.—Dans le script `trifusion.py` commencé lors de l'exercice 4, érire une fonction Python `TriFusion0(T)` qui retourne une version triée du tableau `T` en appliquant l'algorithme de tri fusion.
Tester sur une liste de 1000 nombres tirés au sort.
Une correction est disponible : Utiliser le script `trifusion.py`.

3. ℓ comme *low*, h comme *high*
4. il s'agit de la médiane des trois éléments `T[l]`, `T[c]` `T[h-1]` où `c` est un indice à mi-chemin entre les deux extrémités. Cette correction combat le mauvais comportement de la première version de l'algorithme sur des données couramment rencontrées, par exemple lorsque les données sont déjà triées mais en ordre décroissant