Algorithmique

Algorithmes de tri naïfs

Anicet E. T. Ebou, ediman.ebou@inphb.ci

Pourquoi trier?

- La recherche d'une donnée dans un ensemble trié est plus rapide (recherche dichotomique);
- Remplir un tableau en maintenant l'ordre des éléments n'est pas toujours facile;
- Le nombre de données à trier peut être très important d’où la nécessité d'avoir un algorithme de tri efficace.

01

Tri par comptage

Counting sort en anglais

Principe et algorithme

Quand on a des nombres de 1 à n on peut les trier facilement en comptant le nombre d'occurrences de chaque nombre:

1. On crée un tableau de n valeurs.
2. On met toutes ces valeurs à 0.
3. On traverse T (le tableau à trier) et on compte le nombre de fois où T[i] est pris en incrémentant P[T[i]] (P est un tableau contenant la fréquence d'apparition des nombres de T).
4. Ensuite on balaie le tableau P et on copie autant de fois une valeur qu'elle apparaît dans P.

Illustration de l'algorithme

Tableau avant triage

T

0	1	9	1	2

Tableau de comptage

P

	0	1	2	3	4	5	6	7	8	9
frequence	1	2	1	0	0	0	0	0	0	1

Tableau après triage

T

0	1	1	2	9

Effectif d’apparition dans une liste

Cet algorithme de tri ne se base pas sur des comparaisons mais plutôt par dénombrement qui s'applique sur des valeurs entières. En d’autres mots, cet algorithme s’appuie sur la construction d’un effectif d’apparition puis le balayage de celui-ci de façon croissante, afin de reconstruire les données triées.

Implementation en pseudo-code

```
Procedure tri_comptage(T: Tableau [0..n] d'Entiers)
Var
    i,k,n: Entier
Debut
    borne_superieure <- 0
    n <- longueur(T)
    # determination de la bonne superieure: la valeur entiere maximale presente dans T
    Pour i <- 0 à n faire
        Si (T[i] > borne_superieure) alors
            borne_superieure <- T[i]
        FinSi
    FinPour
    # cree un tableau de borne_superieure elements
    P <- creer_tableau(borne_superieure)
    Pour k <- 0 à borne_superieure faire
        P[k] <- 0
    FinPour
    Pour i <- 1 à n faire
        P[T[i]] <- P[T[i]] + 1
    FinPour
    i <- 0
    compteur <- 0
    Pour k <- 0 à n faire
        Pour j <- 1 à P[k] faire
            T[compteur] <- k
            compteur <- compteur + 1
        FinPour
    FinPour
FinProcedure
```

Estimation du coût

- Coût de recherche de la borne supérieure: on recherche la borne supérieure parmi n éléments: au plus n comparaisons;
- Coût de l'initialisation du tableau de comptage: on parcourt un tableau de p éléments: au plus p initialisations;
- Coût de création du tableau de comptage: au plus n opérations;
- Coût des copies des valeurs dans T: au plus n opérations;
- Finalement, la complexité de ce tri est en O(n+p).

Travaux Pratiques

1. Implémentez en Python l'algorithme de tri par comptage en une fonction appelée tri_comptage.
2. Estimez le temps d'exécution de cette fonction en procédant de la manière suivante:

```
import timeit

import_module = "import random"

testcode = '''

Copier ici le code de votre fonction tri_comptage

'''

test = [random.randint(0,100) for i in range(10000)]

print(timeit.timeit(stmt=testcode, setup=import_module))
```

02

Tri par sélection

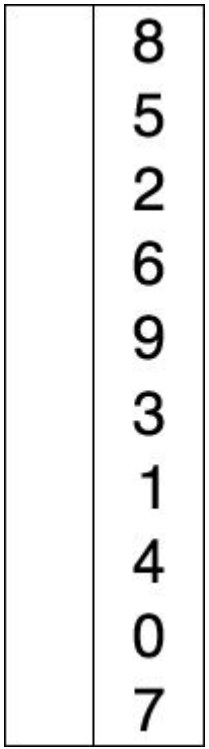
selection sort en anglais

Algorithme et principe

- L'algorithme parcourt le tableau pour rechercher l'indice de la case qui contient l'élément minimum.
- L'algorithme échange le contenu de cette case avec celui de la première case du tableau (si elle ne contient pas déjà le minimum).
- L'algorithme répète ces opérations en éliminant à chaque fois du parcours la case qui a été examinée lors du parcours précédent.
- L'algorithme s'arrête lorsque la partie de tableau sur laquelle il travaille ne contient plus qu'une seule case.

Algorithme et illustration

- On cherche le minimum dans la liste.
- On échange ce minimum avec le premier élément de la liste.
- On recommence avec le reste de la liste, jusqu'au dernier élément.



Implémentation en pseudo-code

```
Procedure tri_selection(L: tableau [0..n] de reels)
   n ← taille de L
   Pour i ← de 1 à n - 1 faire
      min ← i
      Pour j ← de i + 1 à n faire
         Si L[j] < L[min] alors
            min ← j
      FinPour
      Si min ≠ i alors
         échanger L[i] et L[min]
   FinPour
FinProcédure
```

Estimation du coût

Combien de comparaisons?

Au cours d'une itération, toutes les cases sont comparées au minimum courant

- (n-1) comparaisons à la première itération : on recherche le minimum parmi (n) éléments.
- (n-2) comparaisons à la seconde itération : on recherche le minimum parmi (n -1) éléments.
- ...

Au total : n x (n-1)/2 comparaisons

Estimation du coût

Combien d'affectations ?

Le placement définitif d'un élément s'effectue avec un seul échange au plus (n-1) échanges, 3 fois plus d'affectations.

Pour n = 10 000 éléments, 45 000 000 comparaisons et 10 000

échanges.

Travaux Pratiques

1. Implémentez en Python l'algorithme de tri par sélection en une fonction appelée tri_selection.
2. Estimez le temps d'exécution de cette fonction en procédant de la manière suivante:

```
import timeit

import_module = "import random"

testcode = '''

Copier ici le code de votre fonction tri_selection

'''

test = [random.randint(0,100) for i in range(10000)]

print(timeit.timeit(stmt=testcode, setup=import_module))
```

03

Tri par insertion

insertion sort en anglais

Algorithme et principe

Le tri par insertion d'un tableau à n éléments se fait comme suit :

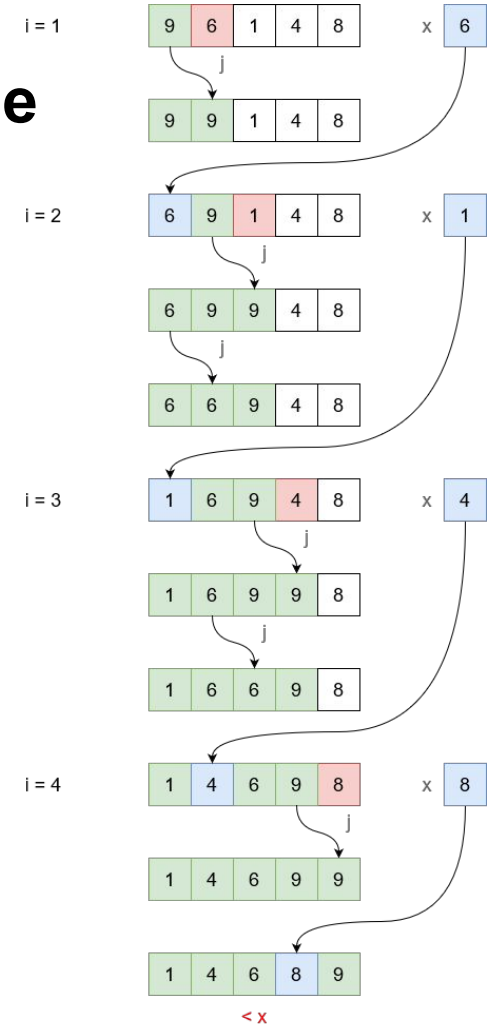
- à l'étape numéro i, (i variant de 0 à n - 2), on suppose que les données d'indice 0 jusqu'à i sont déjà triées et on considère alors la donnée d'indice i + 1 appelée clé;
- on la compare successivement aux données précédentes, en commençant par la donnée d'indice i puis en remontant dans la liste jusqu'à trouver la bonne place de la clé, c'est-à-dire entre deux données successives (qui sont déjà triées), l'une étant plus petite et l'autre plus grande que la clé), ou bien en tout premier si la clé est plus petite que toutes les données précédentes;

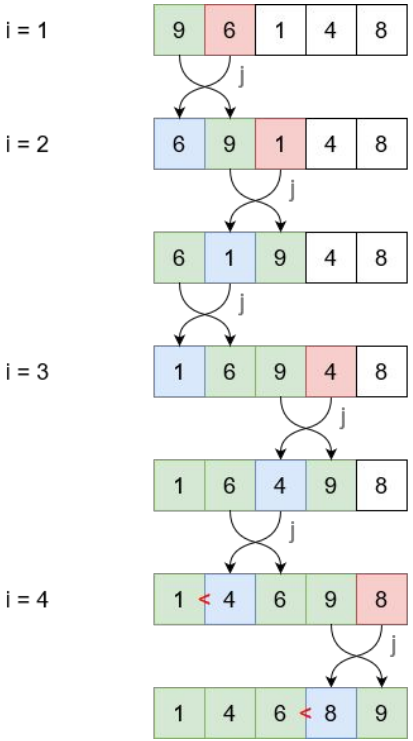
Algorithme et principe

- Au fur et à mesure de ces comparaisons, on décale d'une place vers la droite les données plus grandes que la clé;
- On met la clé à la bonne place et à l'issue de cette étape, les données d'indice 0 à i + 1 sont donc triées.

Illustrations de l'algorithme

6 5 3 1 8 7 2 4





Implémentation en pseudo-code

```
Procedure tri_insertion(L: tableau[0..n] de reels)
    n ← taille de L
    Pour i ← de 1 à  n-1 faire
        x ← L[i]
        j ← i
        Tant que j > 0 et L[j-1] > x faire
            L[j] ← L[j-1]
            j ← j - 1
        FinTantQue
        L[j] ← x
    FinPour
FinProcedure
```

Estimation du coût

- Coût d'exécution de la boucle principale: on fait n - 1 itérations;
- Coût du décalage (étape 3): le cas le plus défavorable est celui où les nombres de la liste initiale sont en ordre décroissant. Dans ce cas, la boucle de décalage est toujours exécutée, mais avec un nombre de décalages croissant. Pour la première clé, il y a 1 décalage, pour la clé suivante 2 décalages, jusqu'à la clé N-1, qui nécessite N-1 décalages. Le temps d'exécution est donc de la forme: $1 + 2 + ... + N - 1 = N (N - 1) / 2 = ½ \times N^2 - N$ d'où la complexité est $O(n^2)$.

Travaux Pratiques

1. Implémentez en Python l'algorithme de tri par insertion en une fonction appelée tri_insertion.
2. Estimez le temps d'exécution de cette fonction en procédant de la manière suivante:

```
import timeit

import_module = "import random"

testcode = '''

Copier ici le code de votre fonction tri_insertion

'''

test = [random.randint(0,100) for i in range(10000)]

print(timeit.timeit(stmt=testcode, setup=import_module))
```

Travaux Pratiques

Exercice

Soit la série statistique suivante décrivant le diamètre à la hauteur de poitrine des arbres d'une parcelle de reboisement: 32, 66, 39, 34, 38, 11, 96, 1, 28, 45, 89, 81, 9, 67, 36, 38, 98, 10, 64, 67, 91, 83, 12, 98, 44, 7, 35, 38, 67, 88, 56, 40, 23.

1. Trier cette série statistique avec l'algorithme de votre choix.
2. Écrire des fonctions permettant de trouver la médiane, le premier quartile et le troisième quartile.
3. Appliquer votre fonction pour retrouver la médiane, le premier quartile et le troisième quartile de cette série.