Informatique - Notes de cours 7

Listes 3 : Algorithmes de tri

M. Marmorat

5 janvier 2025

1 Introduction

Le tri est un algorithme fondamental en informatique. Il permet de classer des données selon un ordre donné. Dans ce cours, nous allons voir trois algorithmes de tri différents : le tri par insertion, le tri par sélection et le tri par comptage.

2 Le tri par insertion

Le tri par insertion est un algorithme simple et efficace pour trier de petites quantités de données. Il consiste à trier les éléments de la liste un par un, en les insérant dans leur position correcte. Voici comment fonctionne l'algorithme :

1. On commence par le deuxième élément de la liste.
2. On compare cet élément avec le premier élément de la liste.
3. Si l'élément est plus petit que le premier élément, on échange leur position.
4. On prend le troisième élément de la liste et on le compare avec le deuxième élément, puis avec le premier élément. On insère l'élément dans la position appropriée.
5. On répète ce processus pour chaque élément de la liste.

Exemple 1. Supposons que l'on veuille trier la liste $[8, 3, 9, 5, 1, 2]$ en utilisant le tri par insertion. Voici la liste des itérations de l'algorithme sur cette liste.

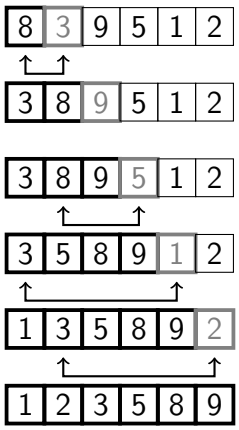

FIGURE 1 – Exemple d'application du tri par insertion

Remarque 1. C'est le tri que l'on effectue d'habitude lorsque l'on trie sa main aux cartes.

Implémentation : Voici le code Python pour réaliser le tri par insertion :

```
def tri_insertion(L):
    n = len(L)
    for k in range(1, n):
        x = L[k] #x est le k-ème élément
        j = k
        while j>=1 and L[j-1] > x :
            # on recule dans le tableau pour trouver la place de x
            L[j] = L[j-1]
            # on décale vers la droite
            j = j-1
        # on a trouvé la place de x, on l'insère à l'indice j
        L[j] = x
```

Calcul de complexité : Combien de comparaisons avons-nous effectué pour effectuer le tri de la liste ?

- On parcourt tous les indices k de la liste, il y en a n.
- Pour chaque indice k, on parcourt toute la partie du tableau qui précède k, autrement dit on effectue k comparaisons (c'est l'instruction $L[j-1] > x$).
- Finalement, comme k prend les valeurs de 1 à $n-1$, on effectue en tout

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$$

comparaisons pour trier la liste L, où n est la taille de la liste.

On dit que la complexité de l'algorithme de tri par insertion est de l'ordre de n^2, où n est la taille du tableau à trier (c'est le terme en n^2 dans la complexité de l'algorithme qui est le plus grand).

3 Le tri par sélection

Le tri par sélection est un autre algorithme simple pour trier des données. Il consiste à sélectionner l'élément le plus petit de la liste et à le placer en première position. On répète ensuite cette opération avec le deuxième élément le plus petit, et ainsi de suite, jusqu'à ce que la liste soit triée. Voici comment fonctionne l'algorithme :

1. On cherche le plus petit élément de la liste.
2. On le place en première position (en échangeant les places des éléments).
3. On cherche le deuxième plus petit élément de la liste.
4. On le place en deuxième position (en écheangeant).
5. On répète ce processus jusqu'à ce que la liste soit triée.

Voici comment fonctionne l'algorithme de tri par sélection :

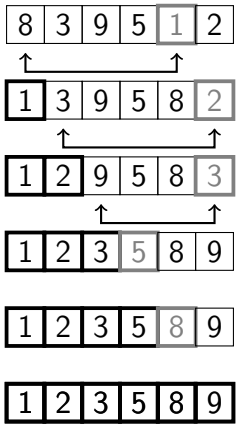

FIGURE 2 – Exemple d'application du tri par sélection

Implémentation : Voici le code Python pour réaliser le tri par sélection.
Nous avons besoin d'une fonction nous permettant de trouver l'indice du minimum d'une liste. On a déjà vu cette fonction :

```
def ind_min(L):
    mini = L[0]
    ind = 0
    for k in range(len(L)):
        if L[k] < mini:
            mini = L[k]
            ind = k
    return ind
```

Equipés de celle-ci nous pouvons coder le tri par sélection.

```
def tri_selection(L):
    for i in range(len(L)):
        sous_liste = L[i:] # fabrique la sous-liste des éléments non
            encore triés
        k_min = ind_min(sous_liste) # cherche l'indice du plus petit élé
            ment
        L[i], L[i+k_min] = L[i+k_min], L[i] # échange l'élément courant
            et le plus petit élément de la sous-liste
```

Calcul de complexité : Combien de comparaisons effectuons-nous ?

- Pour calculer l'indice minimum d'une liste L (avec la fonction `ind_min`) de longueur k, on effectue k comparaisons (on parcourt toute la liste et on vérifie si l'élément courant est plus petit que le minimum).
- On appelle `ind_min` avec la liste L toute entière puis avec la liste privée de son premier élément, puis de ses deux premiers éléments, etc. En tout, nous effectuons

$$n + (n-1) + (n-2) + \cdots + 1 = \sum_{k=1}^{n} = \frac{n(n+1)}{2}.$$

A nouveau, la complexité de l'algorithme est de l'ordre de n^2, où n est la taille de la liste à trier.

4 Le tri par comptage

Le tri par comptage est un algorithme efficace pour trier des données lorsque la plage de valeurs est petite par rapport au nombre d'éléments de la liste. Il consiste à compter le nombre d'éléments de chaque valeur, puis à les placer dans la liste triée. Voici comment fonctionne l'algorithme :

1. On détermine la plage de valeurs dans la liste.
2. On compte le nombre d'éléments de chaque valeur.
3. On calcule la position de chaque élément dans la liste triée.
4. On place chaque élément à sa position dans la liste triée.

Voici comment fonctionne le tri par comptage sur l'exemple $L = [0, 1, 2, 0, 0, 1, 2, 1, 3]$.

- On détermine la plage de valeurs de la liste, ici les entiers de 0 à 3.
- On construit l'histogramme de valeurs de la liste, ici :

Valeur	0	1	2	3
Effectif				

- On reconstruit la liste triée en parcourant l'histogramme :

Remarque 2. Notez que l'algorithme ne fonctionne que si la liste à trier contient des valeurs entières positives.

Implémentation : Voici le code Python pour réaliser le tri par comptage.

```
def tri_comptage(L):
    maxi = max(L) #on détermine le maximum
    histogramme = [0]*(maxi+1) # on crée l'histogramme vide
    for x in L:
        histogramme[x] += 1 #on remplit l'histogramme
    L_triee = [] #création de la liste vide à retourner
    for k in range(len(histogramme)):
        L_triee += [k]*histogramme([k]) #on remplit la liste vide grâce
           à l'histogramme
    return L_triee
```

Le tri par comptage a une complexité temporelle en $O(n + k)$, où n est le nombre d'éléments de la liste et k est la plage de valeurs. Il est donc efficace lorsque k est beaucoup plus petit que n.

5 Conclusion

Dans ce cours, nous avons vu trois algorithmes de tri différents : le tri par insertion, le tri par sélection et le tri par comptage. Chacun de ces algorithmes a ses avantages et ses inconvénients, et il est important de choisir l'algorithme le mieux adapté à la situation. Le tri par insertion est simple et efficace pour de petites quantités de données, le tri par sélection est également simple mais peut être inefficace pour de grandes quantités de données, tandis que le tri par comptage est efficace pour des plages de valeurs restreintes.

Il existe d'autres algorithmes de tri, plus sophistiqués et beaucoup plus efficaces (tri rapide, tri fusion, tri par tas, etc), hors programme en BCPST.