Algorithmes de tri

Premiere SPE NSI

Bruno DARID

CC BY NC SA

Table of contents

1. Tri par sélection 3
1.1 Principe 3
1.2 Algorithme et exemple d'implémentation en python 3
1.3 Validité de l'algorithme 3
1.4 Complexité en temps 4

CC BY NC SA

1. Tri par sélection

1.1 Principe

On commence par rechercher le plus petit élement du tableau puis on l'échange avec le premier élement. Ensuite, on cherche le deuxième plus petit élement et on l'échange avec le deuxième élément du tableau et ainsi de suite jusqu'à ce que le tableau soit entièrement trié.

Voir une l'animation proposée.

1.2 Algorithme et exemple d'implémentation en python

On peut formaliser l'algorithme du tri par sélection avec le pseudo-code suivant:

```
Tri_selection(t)
t: tableau de n éléments (t[0..n-1)
Pour i allant de 0 à n-2:
    idxmini = i
    Pour j allant de i+1 à n-1:
        Si t[j] < t[idxmini]:
            idxmini = j
    Echanger t[i] et t[idxmini]
```

Travail

- Appliquer cet algorithme à la main sur le tableau `t = [3, 4, 1, 7, 2]`.
- donner une implémentation possible en python de cet algorithme et tester.

```
def echange(t, i, j):
    """
    Permute les éléments situés aux index i et j du tableau t
    t: tableau non vide
    i, j: entiers dans l'intervalle [0, len(t)-1]
    """
    tmp = t[i]
    t[i] = t[j]
    t[j] = tmp

def tri_selection(t):
    """
    trie par ordre croissant les éléments de t
    """
    n = len(t)
    #Compléter le code
```

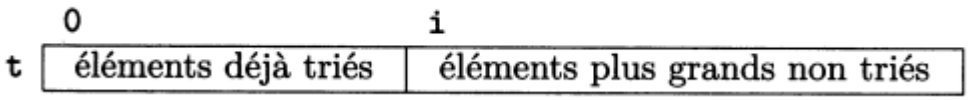
```
# Test
t = [5, 6, 1, 1, 15, 0, 4]
tri_selection(t)
assert t == [0, 1, 1, 4, 5, 6, 15]
```

1.3 Validité de l'algorithme

La **terminaison** est assurée car l'algorithme fait intervenir deux boucles bornées (boucle `for`).

CC BY NC SA

Par ailleurs, la situation au $i^{ème}$ tour de boucle peut être représentée de la manière suivante:



Tous les éléments d'indice compris entre 0 et $i-1$ inclus sont triés **et** ils sont tous inférieurs ou égaux aux éléments de la partie non triée, se trouvant entre i et $n-1$.
La preuve de cette proposition logique peut être délicate à établir en classe de 1re. Cette proposition est un **invariant** pour l'algorithme `Tri_selection`.

Définition

Un invariant de boucle est un *prédicat* (proposition logique) qui est:

- initialement vrai;
- vrai à l'entrée d'une itération ainsi qu'à la sortie de celle-ci

Vocabulaire

Le terme *correction* est à prendre ici au sens *correct*.

Trouver le *bon* invariant garantit que l'algorithme renvoie un résultat conforme aux spécifications et assure ainsi sa **correction partielle**. La combinaison de la correction partielle et de la terminaison permet de conclure à la **correction totale** de l'algorithme.

1.4 Complexité en temps

Le contenu de la boucle interne prend un temps d'exécution constant. Evaluons le nombre de fois qu'elle est exécutée.

Pour $i=0$, elle est exécutée $(n-1)-(0+1)+1=n-1$ fois.
Pour $i=1$, elle est exécutée $(n-1)-(1+1)+1=n-2$ fois.
Si on généralise, le nombre d'exécutions de la boucle interne est:

$$(n-1)+(n-2)+\cdots+2+1$$

Cette somme correspond à la *somme des termes consécutifs d'une suite arithmétique*, dont la valeur pour $n>1$ est donnée par:

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

Pour une taille n très grande de l'entrée, le terme en n^2 devient prépondérant. Autrement dit, le nombre d'opérations effectuées, donc le temps d'exécution, est proportionnel à n^2.
La complexité du tri par sélection est quadratique.

Résumé

Le tri par sélection (du minimum) consiste à chercher le plus petit élément de la partie de tableau non triée et à le mettre à sa place définitive.
Ce problème est résolu habituellement par un algorithme faisant intervenir deux boucles bornées. La **terminaison** est donc assurée.
Un **invariant de boucle** permet de conclure à sa correction partielle. La conjugaison de ces deux propriétés assure la **correction totale** de l'algorithme proposé.
Cet algorithme a une **complexité temporelle quadratique**.

CC BY NC SA

Application directe

En supposant que le tri par sélection prenne un temps directement proportionnel à n^2 et qu'un tri de 16000 valeurs nécessite 6.8 s. Calculer le temps nécessaire pour le tri d'un million de valeurs avec cet algorithme.

Exercice: temps d'exécution

Pour mesurer le temps d'exécution d'un programme, on importe la fonction `time` du module `time`. Cette fonction renvoie le temps en secondes écoulé depuis le 1^{er} janvier 1970.
Le code qui suit permet par exemple d'afficher le temps pris par l'exécution du tri d'un tableau.

```
from time import time
top = time()
tri_selection(t)
print(time() - top)
```

On souhaite comparer les temps d'exécution des tri sélection et insertion sur deux types de tableau: un tableau de nombre au hasard et un tableau de nombres déjà triés. On reprend le code des fonctions de tri du cours.

1. Construire un tableau de 3000 entiers pris au hasard entre 1 et 10000, bornes comprises. Mesurer le temps d'exécution du programme de tri sélection et de tri insertion pour trier ce tableau. *Attention: il faut reconstruire le tableau entre les deux tris*. Quel commentaire peut-on faire concernant les deux résultats ?
2. Construire un tableau de 3000 entiers de 0 à 2999, bornes comprises. Mesurer le temps d'exécution du programme de tri sélection et de tri insertion pour trier ce tableau. Quel commentaire peut-on faire concernant les deux résultats ?
3. Mesurer sur un tableau de 100000 entiers, choisis de manière aléatoire entre 1 et 100000, le temps d'exécution de la méthode `sort()` de python. Syntaxe: `t.sort()`. Commentez.

CC BY NC SA