Algorithmique Avancée : Séance 2

Frédéric A. Hayek

Vendredi 27 septembre 2024

Recherche

Exercice 1 : Recherche Linéaire

Écrire le pseudocode d'une fonction de recherche linéaire (qui renvoie l'indice de la première occurence, ou -1 à défaut) et calculer la complexité au meilleur des cas et au pire des cas.

Recherche

Exercice 1 : Recherche Linéaire

Écrire le pseudocode d'une fonction de recherche linéaire (qui renvoie l'indice de la première occurence, ou -1 à défaut) et calculer la complexité au meilleur des cas et au pire des cas.

Solution Exercice 1

```
function recherche_lineaire(tab, element)
    for i ∈ 0, ··· , len(tab) − 1 do
        if tab[i] == element then
            return i
    return −1
```

Recherche

Exercice 1 : Recherche Linéaire

Écrire le pseudocode d'une fonction de recherche linéaire (qui renvoie l'indice de la première occurence, ou -1 à défaut) et calculer la complexité au meilleur des cas et au pire des cas.

Solution Exercice 1

function recherche_lineaire($tab, element$)
    for $i \in 0, \cdots, len(tab) - 1$ **do**
        if $tab[i] == element$ **then**
            return i
    return -1

Au meilleur des cas, on fait $\Theta(1)$ opérations.
Au pire des cas on fait $\Theta(n)$ opérations. (Pour $n = len(tab)$.)

Tableaux

Exercice 2 : Recherche dans tableau à deux dimensions

Écrire le pseudocode d'une fonction qui prend un tableau à deux dimensions ($m \times n$), un élément, les dimensions (m et n) et qui renvoie les coordonnées de la première occurence, ou -1 à défaut.

Tableaux

Exercice 2 : Recherche dans tableau à deux dimensions

Écrire le pseudocode d'une fonction qui prend un tableau à deux dimensions ($m \times n$), un élément, les dimensions (m et n) et qui renvoie les coordonnées de la première occurence, ou -1 à défaut.

Solution Exercice 2

```
function recherche_tab(tab, element, m, n)
    i ← 0
    for i ∈ 0, ··· , m − 1 do
        for j ∈ 0, ··· , n − 1 do
            if tab[i][j] == element then
                return (i, j)
    return −1
```

Tableaux

Exercice 2 : Recherche dans tableau à deux dimensions

Écrire le pseudocode d'une fonction qui prend un tableau à deux dimensions ($m \times n$), un élément, les dimensions (m et n) et qui renvoie les coordonnées de la première occurence, ou -1 à défaut.

Solution Exercice 2

function recherche_tab($tab, element, m, n$)
    $i \leftarrow 0$
    for $i \in 0, \cdots, m-1$ **do**
        for $j \in 0, \cdots, n-1$ **do**
            if $tab[i][j] == element$ **then**
                return (i, j)
    return -1

Au meilleur des cas, on fait $\Theta(1)$ opérations.
Au pire des cas on fait $\Theta(m \cdot n)$ opérations.

Tableaux

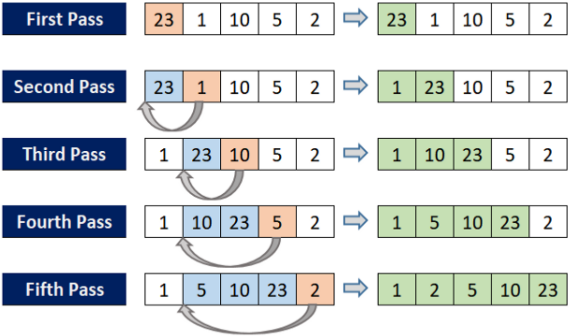
Exercice 3 : Maximum

Écrire le pseudocode d'une fonction qui prend un tableau à une dimension et qui renvoie la valeur maximum du tableau. Analyser la complexité.

Exercice 4 : Maximum 2D

Écrire le pseudocode d'une fonction qui prend un tableau à deux dimensions et qui renvoie la valeur maximum du tableau. Analyser la complexité.

Tri par Insertion



Tri par insertion

Exercice 5 : Tri par insertion

Écrire le pseudocode du tri par insertion, et calculer le nombre d'opérations au meilleur des cas et le nombre d'opérations au pire des cas.

Tri par insertion

Exercice 5 : Tri par insertion

Écrire le pseudocode du tri par insertion, et calculer le nombre d'opérations au meilleur des cas et le nombre d'opérations au pire des cas.

Solution

function tri_insertion(tab)
    for $i \in 1, \ldots, len(tab) - 1$ **do**
        $key \leftarrow tab[i]$
        $j \leftarrow i - 1$
        while $j \geq 0$ **and** $tab[j] > key$ **do**
            $tab[j+1] \leftarrow tab[j]$
            $j \leftarrow j - 1$
        $tab[j+1] \leftarrow key$
    return tab

Tri par insertion

Solution

Au meilleur des cas, le tri par insertion prend $\Theta(n)$ opérations, et au pire des cas il prend $\Theta(n^2)$ opérations.

Tri par insertion

Solution

Au meilleur des cas, le tri par insertion prend $\Theta(n)$ opérations, et au pire des cas il prend $\Theta(n^2)$ opérations.

Problème

Comment savoir si un algorithme est bien si le nombre d'opérations est variable ?

Tri par insertion

Solution

Au meilleur des cas, le tri par insertion prend $\Theta(n)$ opérations, et au pire des cas il prend $\Theta(n^2)$ opérations.

Problème

Comment savoir si un algorithme est bien si le nombre d'opérations est variable ?

Réponse

Pas de réponse exacte. On peut calculer la complexité moyenne, mais pour certains problèmes le pire des cas reste primordial.

Complexité

Complexité moyenne – Tri par insertion

Pour insérer la prochaine valeur dans la liste triée, en moyenne sa position sera au milieu de la liste. Donc chaque itération de la boucle `for` parcourera la boucle `while` en moyenne $\frac{i}{2}$ fois (au lieu de i fois au pire des cas). La complexité du cas moyen est donc :

$$\begin{aligned}(n-1)\Theta(1) + \sum_{i=1}^{n-1} \frac{i}{2} &= \Theta(n) + \frac{\sum_{i=1}^{n} i}{2} \\ &= \Theta(n) + \frac{(n+1)n}{4} \\ &= \Theta(n^2)\end{aligned}$$

Tri par insertion : Modularité

Exercice 6 : Modularité du tri par insertion

- Écrire en pseudocode une fonction `cherche_place` qui prend en paramètre une liste triée et un nombre, et renvoie la position à laquelle le nombre devrait aller si inséré.
- Écrire en pseudocode une fonction `insert_element` qui prend en paramètre une liste, un élément et l'indice où placer cet élément. La fonction renverra la nouvelle liste.
- Écrire en pseudocode une nouvelle version de la fonction `tri_insertion` en utilisant les deux fonctions susmentionnées.
- Analyser la complexité temporelle de cette nouvelle version du tri par insertion.

Tri par insertion : Modularité

Cherche place

```
function cherche_place(tab, element)
    i ← 0
    while i < len(tab) and tab[i] < element do
        i ← i + 1
    return i
```

Tri par insertion : Modularité

Cherche place

function cherche_place($tab, element$)
    $i \leftarrow 0$
    while $i < len(tab)$ **and** $tab[i] < element$ **do**
        $i \leftarrow i + 1$
    return i

Cette fonction prend $\Theta(1)$ au meilleur des cas et $\Theta(n)$ au pire des cas.

Tri par insertion : Modularité

Insérer

function insert(tab, $element$, $index$)
    $i \leftarrow len(tab)$
    while $i > index$ **do**
        $tab[i] \leftarrow tab[i-1]$
        $i \leftarrow i - 1$
    $tab[i] \leftarrow element$
    return tab

Tri par insertion : Modularité

Insérer

```
function insert(tab, element, index)
    i ← len(tab)
    while i > index do
        tab[i] ← tab[i − 1]
        i ← i − 1
    tab[i] ← element
    return tab
```

Cette fonction prend $\Theta(1)$ au meilleur des cas et $\Theta(n)$ au pire des cas.

Tri par insertion : Modularité

Tri par insertion

```
function tri_insertion_modularite(tab)
    tab_new ← []
    for element ∈ tab do
        index ← cherche_place(tab_new, element)
        inserer(tab_new, element, index)
    return tab_new
```

Tri par insertion : Modularité

Tri par insertion

1: **function** tri_insertion_modularite(tab)
2: $tab\_new \leftarrow []$
3: **for** $element \in tab$ **do**
4: $index \leftarrow cherche\_place(tab\_new, element)$
5: $inserer(tab\_new, element, index)$
6: **return** tab_new

Le tri par insertion tel défini ici prend $\Theta(n^2)$ au meilleur des cas !
Il prend également $\Theta(n^2)$ dans le pire des cas.

Tri par insertion : Modularité

Cherche place 2

function cherche_place_2(tab, $element$)
  $i \leftarrow len(tab) - 1$
  while $i \geq 0$ **and** $tab[i] > element$ **do**
    $i \leftarrow i - 1$
  return $i + 1$

Tri par insertion : Modularité

Tri par insertion 2

function tri_insertion_modularite_2(tab)
  $tab\_new \leftarrow []$
  for $element \in tab$ **do**
    $index \leftarrow cherche\_place2(tab\_new, element)$
    $inserer(tab\_new, element, index)$
  return tab_new

Tri par insertion : Modularité

Tri par insertion 2

```
function tri_insertion_modularite_2(tab)
    tab_new ← []
    for element ∈ tab do
        index ← cherche_place2(tab_new, element)
        inserer(tab_new, element, index)
    return tab_new
```

Le tri par insertion tel défini ici prend $\Theta(n)$ au meilleur des cas !
Il prend $\Theta(n^2)$ dans le pire des cas.

Tri bulle

Exercice 7 : Tri bulle

Écrire le pseudocode d'un algorithme de tri bulle et en analyser la complexité.

TP

Pratiques

- `Python` est-il `Camel Case` ou `snake_case` ?
- Toujours faire des tests.

Sudoku

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

Sudoku

	2	4	
1			3
4			2
	1	3	

16	2	3	13	4	14	15	1	9	7	6	12	5	11	10	8
5	11	10	8	9	7	6	12	16	2	3	13	4	14	15	1
9	7	6	12	5	11	10	8	4	14	15	1	16	2	3	13
4	14	15	1	16	2	3	13	5	11	10	8	9	7	6	12
13	3	2	16	1	15	14	4	12	6	7	9	8	10	11	5
8	10	11	5	12	6	7	9	13	3	2	16	1	15	14	4
12	6	7	9	8	10	11	5	1	15	14	4	13	3	2	16
1	15	14	4	13	3	2	16	8	10	11	5	12	6	7	9
3	16	13	2	15	4	1	14	6	9	12	7	10	5	8	11
10	5	8	11	6	9	12	7	3	16	13	2	15	4	1	14
6	9	12	7	10	5	8	11	15	4	1	14	3	16	13	2
15	4	1	14	3	16	13	2	10	5	8	11	6	9	12	7
2	13	16	3	14	1	4	15	7	12	9	6	11	8	5	10
11	8	5	10	7	12	9	6	2	13	16	3	14	1	4	15
7	12	9	6	11	8	5	10	14	1	4	15	2	13	16	3
14	1	4	15	2	13	16	3	11	8	5	10	7	12	9	6

Lire et écrire en python

Aide

- `f = open("nom_fichier", "r")` pour lire en lecture
- `contenu = f.read()` pour lire le contenu de `f` dans `contenu`
- `f = open("nom_fichier", "w")` pour lire en écriture
- `f.write(contenu)` pour écrire `contenu` dans `f`