Université Assane SECK de Ziguinchor



Unité de Formation et de Recherche des Sciences et Technologies

Département d'Informatique

CONCEPTION DES ALGORITHMES

Licence 1 Math - Physique - Informatique

Janvier 2021

©Youssou DIENG

ydieng@univ-zig.sn

Il existe de nombreuses façons de concevoir un algorithme. Le tri par insertion utilise une approche incrémentale qui après avoir trié le sous-tableau A[1..j-1], on insère l'élément A[j] au bon emplacement pour produire le sous-tableau trié A[1..j].

Cette section va présenter une autre approche de conception appelée « diviser pour régner ». Grace à cette technique nous proposons un algorithme de tri avec un temps d'exécution du cas le plus défavorable très inférieur à celui du tri par insertion. L'un des avantages des algorithmes diviser-pour-régner est que leurs temps d'exécution sont souvent faciles à déterminer, via des techniques que nous présenterons.

1 - MÉTHODE DIVISER-POUR-RÉGNER

Nombre d'algorithmes utiles sont d'essence récursive : ils s'appellent eux-mêmes, de manière récursive, une ou plusieurs fois pour traiter des sous-problèmes très similaires. Ces algorithmes suivent généralement une approche diviser pour régner :

- d'abord ils séparent le problème en plusieurs sous-problèmes semblables au problème initial mais de taille moindre
- ☐ ensuite ils résolvent les sous-problèmes de façon récursive,
- enfin ils combinent toutes les solutions pour produire la solution du problème original.

1.1 - Le 3 étapes du paradigme diviser-pour-régner

- 1) **Diviser** le problème en un certain nombre de sous-problèmes.
- 2) **Régner** sur les sous-problèmes en le résolvant de manière récursive. Si la taille d'un sousproblème est suffisamment réduite, on peut le résoudre directement.
- 3) Combiner les solutions des sous-problèmes pour produire la solution du problème originel.

1.2 - Exemple du tri par fusion

Cet algorithme suit fidèlement la méthodologie diviser-pour-régner.

- 1) **Diviser** la suite de n éléments à trier en deux sous-suites de n/2 éléments chacune.
- 2) **Régner** en Triant les deux sous-suites de manière récursive en utilisant le tri par fusion.
- 3) Combiner/Fusionner les deux sous-suites triées pour produire la réponse triée.

La récursivité s'arrête quand la séquence à trier a une longueur 1: en effet il n'y a plus rien à faire puisqu'une suite de longueur 1 est déjà triée. La clé de voûte de l'algorithme est la fusion de deux séquences triées dans l'étape « combiner ». Cette fusion est faite grâce à une

procédure auxiliaire **FUSION**(A, p, q, r). A étant un tableau et p, q et r étant des indices numérotant des éléments du tableau tels que $p \le q \le r$. La procédure suppose que les soustableaux A[p . . q] et A[q + 1 . . r] sont triés. Elle les **fusionne** pour en faire un même soustableau trié, qui remplace le sous-tableau courant A[p . . r]. Notre procédure **FUSION** a une durée $\Theta(n)$, où n = r - p + 1 est le nombre d'éléments fusionnés.

Principe de fonctionnement du fusion : Supposons que l'on ait deux piles **A** et **B** de cartes posées sur la table. Supposons que chaque pile est triée de façon à ce que la carte la plus faible soit en haut. On veut fusionner les deux piles pour obtenir une pile unique triée **R**, dans laquelle les cartes seront à l'envers. Le principe de fusion est :

- Choisir la plus faible des deux cartes occupant les sommets respectifs des deux piles
 A et B:
 - Retirer la de sa pile (ce qui a va exposer une nouvelle carte), puis placer la à l'envers sur la pile résultat **R**.
- 2. Répéter cette étape jusqu'à épuisement de l'une des piles A et B.

Après quoi il suffit de prendre la pile qui reste ($\bf A$ ou $\bf B$) et de la placer à l'envers sur la pile résultat $\bf R$.

Complexité de fusion: Au niveau du calcul, chaque étape fondamentale prend un temps constant, vu que l'on se contente de comparer les deux cartes du haut. Comme on effectue au plus n étapes fondamentales, la fusion prend une durée $\Theta(n)$.

1.3 · L'algorithme du tri par fusion

Le principe est de placer en bas de chaque pile une carte *sentinelle* contenant une valeur spéciale. Cette valeur nous permettra de ne pas vérifier à chaque fois si l'une des piles est vide. (Nous utiliserons ∞ comme valeur sentinelle.) Ainsi, chaque fois qu'il y a apparition d'une carte portant la valeur ∞ , elle ne peut pas être la carte la plus faible sauf si les deux piles exposent en même temps leurs cartes sentinelle. Vu qu'on a exactement r-p+1 cartes sur la pile de sortie, nous pourrons arrêter lorsque ce nombre d'étapes sera effectué.

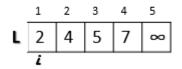
FUSION(A, p, q, r) $n_1 \leftarrow q - p + 1$ $n_2 \leftarrow r - q$ créer tableaux $L[1...n_1 + 1]$ et $R[1...n_2 + 1]$ tableau A[q + 1...r]. pour $i \leftarrow 1$ à n_1 5 faire $L[i] \leftarrow A[p+i-1]$ pour j ← 1 à n_2 6 7 faire $R[j] \leftarrow A[q+j]$ $L[n_1+1] \leftarrow \infty$ 8 $R[n_2+1] \leftarrow \infty$ 9 10 $i \leftarrow 1$ $j \leftarrow 1$ 11 12 pour $k \leftarrow p$ à r13 faire si $L[i] \leq R[j]$ 14 alors $A[k] \leftarrow L[i]$ 15 $i \leftarrow i+1$ 16 sinon $A[k] \leftarrow R[j]$ 17 $j \leftarrow j + 1$

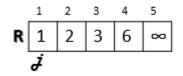
- La ligne 1 calcule la longueur n1 du sous-tableau A[p..q] et la ligne 2 calcule la longueur n2 du sous-tableau A[q+1,r]
 - On crée des tableaux L et R de longueurs n1 + 1 et n2 + 1, respectivement, en ligne 3.
- La boucle **pour** des lignes 4–5 copie le sous-tableau A[p . . q] dans L[1 . . n1] et la boucle **pour** des lignes 6–7 copie le sous-tableau A[q + 1 . . r] dans R[1 . . n2].
- Les lignes 8–9 placent les sentinelles aux extrémités des tableaux L et R.
- Les lignes 10–17, effectuent les r p + 1 étapes fondamentales en conservant l'invariant de boucle que voici :
 - Au début de chaque itération de la boucle pour des lignes 12–17, le sous-tableau A[p..k 1] contient les k p plus petits éléments de L[1..n1+1] et R[1..n2+1], en ordre trié.

i. Exemple:

1) Quel est l'élément qui doit être en position k = 9? C'est le plus petit entre L[i=1] et R[j=1].

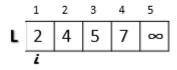
_ 8	9	10	11	12	13	14	15	16	17
Α	2	4	5	7	1	2	3	6	

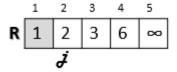




2) Quel est l'élément qui doit être en position k = 10? C'est le plus petit entre L[i=1] et R[j=2].

	8	9	10	11	12	13	14	15	16	17_
Α		2	4	5	7	1	2	3	6	
-			k							





3) Quel est l'élément qui doit être en position k = 11 ? C'est le plus petit entre L[i=2] et R[j=2].



