IFT2015 automne 2012 — Devoir 3

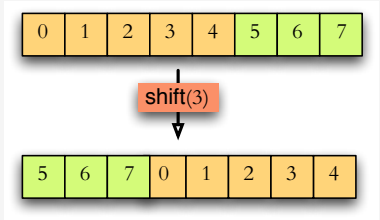
Miklós Csűrös

5 décembre 2012

À remettre avant 20 :15 le 13 décembre. Remettez un rapport écrit par courriel (à csuros@iro. . .) en format PDF. Travaillez seul.

3 Fusion bitonique en place (30+10 points)

3.1 Décalage circulaire (15 points)



Supposons qu'on veut faire des décalages circulaires dans un tableau $A[0..n-1]$. Un décalage par δ correspond à l'exécution *parallèle* des affectations $A[(i+\delta) \bmod n] \leftarrow A[i]$:

$$A[\delta] \leftarrow A[0]; \quad A[\delta+1] \leftarrow A[1]; \quad \ldots \quad A[n-1] \leftarrow A[n-1-\delta];$$
$$A[0] \leftarrow A[n-\delta]; \quad \ldots \qquad\qquad A[\delta-1] \leftarrow A[n-1]$$

Quand $\delta = 1$, il est clair qu'on peut performer le décalage en place par une boucle simple :

```
  SHIFT1(A[0..n − 1])
S1  i ← 0; navette ← A[0]
S2  do
S3      i ← (i + 1) mod n
S4      échanger navette ↔ A[i]
S5  while i ≠ 0
```

▶ Donnez l'algorithme SHIFT(A, g, d, δ) qui exécute un décalage circulaire à δ positions du sous-tableau $A[g..d-1]$ *en place*. Vous pouvez assumer que $n = d - g \geq 0$ et que $0 < \delta < n$. L'algorithme doit utiliser un espace de travail[1] de $O(1)$, et s'exécuter en $O(n)$, pour tout choix de δ (même pour p.e., $\delta = n/3$)

Indice: *Qu'est-ce qui se passe si on met $i \leftarrow (i+\delta) \bmod n$ en ligne S3 ?*

3.2 Fusion d'une séquence bitonique (15 points)

Supposons que le tableau $A[0..n-1]$ contient une **séquence bitonique** d'éléments distincts. Dans d'autres mots, il existe un $m \in \{0, 1, 2, \ldots, n-1\}$ tel que

$$A[0] < A[1] < A[2] < \cdots < A[m] \quad \text{et} \quad A[m] > A[m+1] > A[m+2] > \cdots > A[n-1]$$

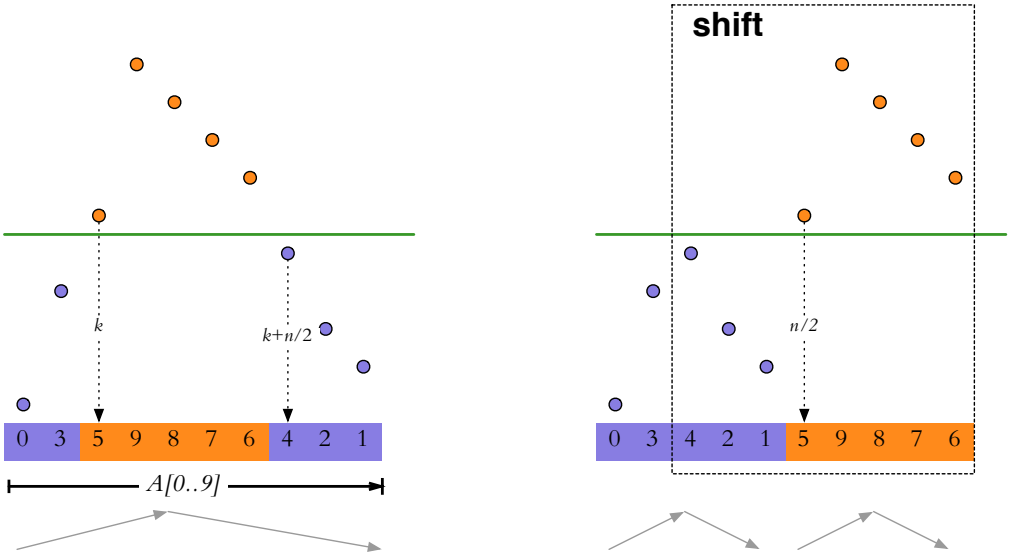
Un tableau trié en ordre croissant ($m = n-1$) ou décroissant ($m = 0$) est aussi considéré bitonique.

[1] On ne compte pas l'entrée $A[0..n-1]$ dans l'espace de travail, seulement les variables locales.

On a vu l'exemple du tri par fusion, où la fusion est très performante grâce à l'arrangement bitonique dans le tableau auxiliaire. L'exemple montre qu'on peut trier une séquence bitonique en temps linéaire, si on a un autre tableau auxiliaire. Dans cet exercice, on développe un tri (ou *fusion*) de séquences bitoniques en place. notamment, on exploite le fait que les plus grands éléments occupent des cases consécutives dans le tableau :

Théorème de la médiane bitonique. *Soit $A[0..n-1]$ une séquence bitonique, et $m = \lceil n/2 \rceil$. Il existe un indice $k \in \{0, 1, \ldots, \lfloor n/2 \rfloor\}$ tel que $A[k..k+m-1]$ contient les plus grands éléments : si $i < k$ ou $i \geq k+m$, alors $A[i] < A[j]$ pour tout $k \leq j < k+m$.*

Par le théorème, on arrive à une partition équilibrée entre $A[0..m-1]$ et $A[m..n-1]$ après le décalage circulaire de $A[k..n-1]$. Les deux sous-tableaux de la partition sont bitoniques. Le tri se continue par récurrence dans les sous-tableaux. Les cas terminaux sont des sous-tableaux de taille 1 (bitoniques par défaut).



On a alors l'algorithme suivant esquissé :

```
FUSIONSB(A, g, d)                    // (tri du sous-tableau A[g..d-1] contenant une séquence bitonique)
B1  if d − g ≤ 1 then return
B2  PARTITIONSB(A, g, d);
B3  m ← ⌈(g + d)/2⌉; FUSIONSB(A, g, m); FUSIONSB(A, m, d)
```

m = floor((g+d)/2)

► Développez le code de PARTITIONSB en ligne B2. L'algorithme doit prendre $O(d-g)$ temps et utiliser $O(1)$ espace de travail.

Indice: *Chercher l'indice k du théorème de la médiane, et faites un décalage circulaire dans le bloc $k..d-1$. Dans la recherche de k, suivez la démarche de la fusion bitonique utilisée avec tri par fusion (v. Notes de cours, §10.5) pour compter et exclure les plus petits éléments aux deux côtés.*

► Analysez le temps de calcul de l'algorithme. Exprimez le temps de calcul $T(\ell)$ de FUSIONSB sur un sous-tableau de taille $\ell = d-g$ par une récurrence asymptotique. Donnez la solution pour la récurrence en notation asymptotique. (Il n'est pas nécessaire de donner une preuve formelle, mais justifiez votre solution en quelques mots.)

3.3 ♡ Le vrai tri bitonique (10 points boni)

Le tri bitonique classique fait la partition autrement. Au lieu de faire le décalage, la fusion se fait par échanges :

```
BitonicSplit(A, g, d) // partition du sous-tableau A[g..d − 1]
    n ← d − g ; for i ← g, g + 1, g + n/2 − 1 do if A[i] > A[i + n/2] then échanger A[i] ↔ A[i + n/2]
```

Si le sous-tableau à l'entrée était le décalage circulaire d'une séquence bitonique, alors la partition le divise en deux sous-tableaux $A[g..g + n/2 - 1]$, $A[g + n/2..g + n - 1]$ qui sont eux-mêmes des décalages de séquences bitoniques, et les éléments de la première partie sont inférieures ou égales aux éléments dans la deuxième partie. Après BitonicSplit, on appelle le tri récursivement sur les deux parties. Comment peut-on transformer un tableau quelconque en une séquence bitonique ? Par récursivité ! Il suffit de trier $[0..n/2 - 1]$ en ordre croissant et $[n/2..n - 1]$ en ordre décroissant par récursion. On obtient un tri en place.

```
BitonicSort(A, ordre, g, d)   // tri de A[g..d − 1] dans l'ordre croissant ou décroissant selon ordre =↗,↘
T1  n ← d − g ; if n = 1 then return
T2  BitonicSort(A, ordre, g, g + n/2)
T3  BitonicSort(A, inverse(ordre), g + n/2, d)   // ordre inverse : inverse(↗) =↘ et inverse(↘) =↗
T4  BitonicFusion(A, ordre, g, d)

BitonicFusion(A, ordre, g, d)        // fusion du sous-tableau A[g..d − 1] contenant une séquence bitonique
F1  n ← d − g ; if n = 1 then return
F2  if ordre =↗ then k ← g; j ← n/2 else k ← g + n/2; j ← −n/2     // préparation de BitonicSplit
F3  for i ← k, k + 1, . . . , k + n/2 − 1 do
F4      if A[i] > A[i + j] then échanger A[i] ↔ A[i + j]            // BitonicSplit selon l'ordre requis
F5  BitonicFusion(A, ordre, g, g + n/2)
F6  BitonicFusion(A, ordre, g + n/2, d)
```

L'appel initial est BitonicSort$(A, \nearrow, 0, n)$. Il est intéressant de noter qu'on combine deux variétés du principe de diviser-pour-régner. BitonicSort suit la structure du tri par fusion, où la phase de «régner» est l'appel à BitonicFusion en ligne T4. BitonicFusion suit le principe complémentaire de «combiner-pour-régner» comme le tri rapide : on assure la domination avant les appels récursifs.

Le tri bitonique est hautement parallélisable (on peux exécuter les comparaisons et échanges de ligne F4 en même temps à travers de sous-tableaux multiples), ce qu'on exploite par exemple dans des processeurs graphiques. Le tri bitonique s'exécute en $O(\log^2 n)$ temps dans un environnement parallèle.

BitonicFusion

► Supposons qu'on adapte notre code FusionSB(A, g, d) de §3.2 pour l'utiliser en place de ~~BitonicSort~~ dans la ligne T4 (si ordre $=\searrow$, décaler au début de $A[g..d-1]$ pour placer les grands éléments dans la première moitié). Analysez le temps de calcul de BitonicSort avec notre FusionSB adapté (sur un seul processeur). Montrez la récurrence et justifiez sa solution asymptotique formellement. Il suffit de considérer des tableaux de taille $n = 2^0, 2^1, 2^2, \ldots$.