8 Algorithmes avec un tas : tri et fusion

8.1 Fusion de listes multiples

On peut utiliser un tas pour fusionner plusieurs listes d'une manière efficace. L'algorithme suivant syncronise le parcours simultané des listes à l'aide d'une file de priorité. Le parcours maintient un tableau d'indices I ; la file stocke des paires (j, a) pour chaque $j = 0, \ldots, m-1$ qui indique que l'élément a vient de liste j. En une boucle, on retire l'entrée (j, a) avec a minimal, et avance l'indice $I[j]$ de liste j.

Algo MULTI-FUSION($A_0[0..n_0-1], \ldots, A_{m-1}[0..n_{m-1}-1]$)
// entrée : m tableaux triés
MF1 initialiser la file de priorité $H \leftarrow \emptyset$
MF2 initialiser tableau d'indices $I[0..m-1]$
MF3 **for** $j \leftarrow 0, \ldots, m-1$ **do**
MF4 $I[j] \leftarrow 0$; H.insert$\big(j, A_j\big[I[j]\big]\big)$ *// indice $I[j]$ parcourt $A_j[]$*
MF5 $n \leftarrow n_0 + n_1 + \cdots + n_{m-1}$; initialiser $C[0..n-1]$ *// résultat dans C*
MF6 **for** $k \leftarrow 0, 1, \ldots, n-1$ **do**
MF6.1 $(j, a) \leftarrow H$.deleteMin()
MF6.2 $C[k] \leftarrow a$; $I[j] \leftarrow I[j] + 1$
MF6.3 **if** $I[j] < n_j$ **then** H.insert$\big(j, A_j\big[I[j]\big]\big)$ *// prochain élément en A_j*
MF7 **return** C

Une implantation efficace est basée sur un min-tas binaire $H[0..m-1]$ sur les indices. Ici, H est une permutation des indices $0, \ldots, m-1$, et la priorité de $j = H[i]$ est $\phi(j) = A_j[I[j]]$. Pour simplifier, on définit aussi $\phi(j) = \infty$ quand $I[j] = n_j$ (liste j est épuisée). On doit maintenir l'ordre de tas $\phi(\mathsf{parent}(j)) \le \phi(j)$ pour tout $j = 1, \ldots, m-1$ où $\mathsf{parent}(j) = \big\lfloor (j-1)/2 \big\rfloor$. Il suffit d'adapter SINK car c'est seulement la priorité de l'élément $H[0]$ qui change quand on avance l'indice. Ligne MF4 est implantée par une approche analogue (initialiser $H[j] \leftarrow j$, et heapify).

MF6.1 $j \leftarrow H[0]$; $C[k] \leftarrow A_j\big[I[j]\big]$ *// le plus petit vient de la liste $j = H[0]$*
MF6.2 $I[j] \leftarrow I[j] + 1$ *// sink : la priorité $\phi(j)$ de la racine change*
MF6.3 $i \leftarrow 0$; **while** $2i + 1 < m$ **do**
MF6.4 $c \leftarrow 2i + 1$; *// indice du meilleur enfant*
MF6.5 **if** $2i + 2 < m$ && $\phi(H[2i+2]) < \phi(H[c])$ **then** $c \leftarrow 2i + 2$
MF6.6 **if** $\phi(H[c]) < \phi(j)$ **then** $H[i] \leftarrow H[c]; i \leftarrow c$ *// boucler*
MF6.7 **else break** *// sortir de la boucle : bon placement trouvé*
MF6.8 $H[i] \leftarrow j$

Théorème 8.1. *L'algorithme prend $\Theta(n \log m)$ temps et utilise $\Theta(m)$ espace de travail (à part des tableaux d'entrée et du résultat).*

Exercice 8.1. *Montrez comment implanter l'algorithme* MEILLEUR-EMTS *de §7.1 avec un tas binaire. Exploitez couler/nager pour combiner des opérations syncronisées* deleteMin *et* insert.

8.2 Tri par tas

W(fr)

Les cas extremes (fusion de n tableaux de taille 1 chacun, ou «fusion» d'un seul tableau de taille n) correspondent à un tri d'un tableau $A[0..n-1]$. Le tri par tas se fait en place : après heapify, on maintient l'ordre de tas dans le préfixe $A[0..i]$ en une boucle $i \leftarrow n-1, \ldots 1$. Le suffixe $A[i..n-1]$ est trié en ordre décroisssant. À chaque itération, après avoir échangé $A[0] \leftrightarrow A[i]$, on rétablit l'ordre de tas en $A[0..i-1]$.

```
    HEAPSORT(A[0..n − 1])                          // // tableau non-trié
H1  heapify(A)
H2  for i ← n − 1, . . . 1 do
H3      v ← A[i] ; A[i] ← A[0]                     // échange A[i] ↔ A[0]
H4      SINK(v, 0, A, i − 1)      rétablissement de l'ordre de tas en A[0..i − 1]
```

$A[0..n-1]$ est dans l'ordre décroissant à la fin — pour l'ordre croissant, utiliser un max-tas.

- ⋆ **heapsort** : temps $O(n \log n)$ dans le pire des cas, sans espace additionnelle !
- ⋆ **quicksort** : $O(n^2)$ dans le pire des cas
- ⋆ **mergesort** : $O(n \log n)$ dans le pire des cas mais utilise un espace auxiliaire de taille n.