IFT2015 hiver 2010 — Notes de cours

Miklós Csűrös

10 mars 2010

12 Tas d-aire et tri par tas

12.1 Tas d-aire

Tas d-aire : on utilise un arbre complet d-aire avec une arité $d \geq 2$. L'implantation utilise un tableau $A[1..n]$: Parent de l'indice i est $\lceil (i-1)/d \rceil$, enfants sont à $d(i-1)+2..di+1$. Ordre de tas :

$$A[i] \geq A\left[\left\lceil \frac{i-1}{d} \right\rceil\right] \qquad \text{pour tout } i > 1$$

- ⋆ deleteMin : $O(d \log_d n)$ dans un tas d-aire sur n éléments
- ⋆ insert : $O(\log_d n)$ dans un tas d-aire sur n éléments
- ⋆ findMin : $O(1)$
- ⋆ NAGER et COULER : $O(\log_d n)$ et $O(d \log_d n)$

⇒ Permet de balancer le coût de l'insertion et de la suppression si on a une bonne idée de leur fréquence.

12.2 Files à priorité

	liste triée	liste non-triée	binaire	binomial
deleteMin	O(1)	O(n)	O(log n)	O(log n)
insert	O(n)	O(1)	O(log n)	O(log n)
merge	O(n)	O(1)	O(n)	O(log n)
decreaseKey	O(n)	O(1)	O(log n)	O(log n)

D'autres implantations existent (nécessaires pour un merge efficace) : binomial heap, skew heap, Fibonacci heap.
decreaseKey est important dans quelques algorithmes fondamentaux sur des graphes (plus court chemin, arbre couvrant minimal)

12.3 Heapisation

Opération heapify (A) met les éléments de la vecteur $A[1..n]$ dans l'ordre de tas. Triviale ?

$H \leftarrow \emptyset$; **pour** $i \leftarrow 1, \ldots, n$ **faire** INSERT$(A[i], H)$; $A \leftarrow H$

$\Rightarrow$ prend $O(n \log_d n)$

Meilleure solution :

```
HEAPIFY(A) // vecteur arbitraire A[1..n]
pour i ← n, ..., 1 faire COULER(A[i], i, A)
```

$\Rightarrow$ prend $O(n)$

Preuve du temps de calcul : si i est à la hauteur j, alors il prend $O(j)$ de faire COULER$(\cdot, i, \cdot)$. Il y a $\leq n/d^j$ nœuds à la hauteur j. Donc temps est de

$$\sum_j \frac{n}{d^j} O(j) = O\Big(n \sum_j \frac{j}{d^j}\Big) = O(n).$$

□

«Évidemment», $O(n)$ est optimal pour construire le tas.

Preuve formelle :

- Trouver le minimum des élements dans une vecteur de taille n prend $n - 1$ comparaisons, donc un temps de $\Omega(n)$ est nécessaire pour trouver le minimum.
- Avec n'importe quelle implantation de heapify, on peut appeler findMin après pour retrouver le minimum en $O(1)$.
- Donc le temps de heapify doit être $\Omega(n)$, sinon on pourrait trouver le minimum en utilisant heapify+findMin en un temps $o(n) + O(1) = o(n)$. □

12.4 Tri par tas

```
  HEAPSORT(A) // vecteur non-trié A[1..n]
H1 heapify(A)
H2 pour i ← |A|, ... 2 faire
H3     échanger A[1] ↔ A[i]
H4     COULER(A[1], 1, A[1..i − 1])
```

$A[1..n]$ 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.