TD d'Algo n° 3

EPITA ING1 2012; A. DURET-LUTZ

12 novembre 2009

1 Manipulation des tas

Rappelons des différents algorithmes sur les tas "max" (c'est-à-dire avec la plus grande valeur en racine) :

```
HEAPIFY(A, i, m)
 1  l ←LEFTCHILD(i)
 2  r ←RIGHTCHILD(i)
 3  if l ≤ m and A[l] > A[i]
 4     then largest ← l
 5     else largest ← i
 6  if r ≤ m and A[r] > A[largest]
 7     then largest ← r
 8  if largest ≠ i then
 9     A[i] ↔ A[largest]
10     HEAPIFY(A, largest, m)
```

```
HEAP-SORT(A)
1  BUILD-HEAP(A)
2  for i ← length(A) down to 2 do
3     A[1] ↔ A[i]
4     HEAPIFY(A, 1, i − 1)
```

```
BUILD-HEAP(A)
1  for i ← ⌊length(A)/2⌋ down to 1 do
2     HEAPIFY(A, i, length(A))
```

```
HEAP-INSERT(A, m, v)
1  i ← m + 1
2  A[i] ← v
3  while i > 1 and A[Parent(i)] < A[i] do
4     A[Parent(i)] ↔ A[i]
5     i ← Parent(i)
```

```
HEAP-REMOVE-MAX(A, v)
1  v ← A[1]
2  A[1] ← A[length(A)]
3  length(A) ← length(A) − 1
4  HEAPIFY(A, i, length(A))
5  return v
```

1. On considère le tas représenté par le tableau | 10 | 7 | 9 | 4 | 3 | 6 | 2 | 1 |. Donnez l'état du tas après les insertions successives des valeurs 8 et 12, puis après les suppressions successives des 3 plus grandes valeurs.
2. Triez le tableau | 2 | 4 | 6 | 8 | 1 | 3 | 5 | 7 | (avec un tri par tas, bien sûr)
3. Rappelez les complexités établies en cours pour chacun des algorithmes ci-dessus lorsque $n = length(A)$.
4. Quelle est la complexité de trier (par tas) un tableau strictement décroissant ?
5. Quelle est la complexité de trier (par tas) un tableau dont tous les éléments sont identiques ?
6. Quelles sont les modifications à apporter aux algorithmes ci-dessus pour réaliser un tas "min".

2 File de priorité

On considère la gestion d'un file d'attente d'imprimante, dans laquelle chaque tâche d'impression est soumise avec une priorité. Lorsque l'imprimante est libre, la tâche la plus prioritaire est retirée de la liste pour être traitée par l'imprimante. Une telle file d'attente sera réalisée efficacement avec un tas "max" :

- la soumission d'une tâche correspond à HEAP-INSERT,
- le retrait de la tâche prioritaire se fait avec HEAP-REMOVE-MAX.

On souhaiterait d'autre part être capable d'augmenter la priorité d'une tâche existante dans la file. Les opérations écrites jusqu'à présent ne le permettent pas.

1. Écrivez une procédure HEAP-INCREASE-KEY(A, i, v) qui donne à la tâche $A[i]$ la valeur v (supposée supérieure à l'ancienne valeur) et corrige la structure du tas en conséquence.
2. Quelle est la complexité de HEAP-INCREASE-KEY ?
3. Parmi les cinq algorithmes présentés au début du TD, lequel peut maintenant être simplifié avec un appel à HEAP-INCREASE-KEY ?
4. Une file de priorité pourrait-elle être implémentée par une *skip list* décroissante ? Quelles seraient les complexités des opérations d'insertion, de suppression du max, et d'augmentation d'une priorité ?
5. Une file de priorité pourrait-elle être implémentée par un arbre rouge et noir ? Quelles seraient les complexités des trois opérations ?

3 Arbres rouge et noirs

Dessinez les arbres rouge et noir résultants des insertions successives des valeurs 8, 17, et 19 dans l'arbre suivant :