Projeto e Análise de Algoritmos

Heapsort

Diane Castonguay
diane@inf.ufg.br

Instituto de Informática
Universidade Federal de Goiás

heap: arvore implementado em forma de vetor

tamanho-do-heap[A] ≤ comprimento[A]

PARENT(i)
    retorna $\lfloor i/2 \rfloor$

LEFT(i)
    retorna 2i

RIGHT(i)
    retorna 2i+1

Heap maximo
A[PARENT(i)] ≥ A[i]

```
MAX-HEAPIFY(A, i)
1. l ← LEFT(i)
2. r ← RIGHT(i)
3. se l ≤ tamanho-do-heap[A] e A[l] > A[i]
4.      então maior ← l
5.      senão maior ← i
6. fim-se
7. se r ≤ tamanho-do-heap[A] e A[r] > A[maior]
8.      então maior ← r
9. fim-se
10.se maior ≠ i
11.    então    A[i] ↔ A[maior]
12.             MAX-HEAPIFY(A, maior)
13.fim-se
```

BUILD-MAX-HEAP(A)

1. *tamanho-do-heap*[A] ← *comprimento*[A]
2. para i ← ⌊*comprimento*[A]/2⌋ até 1 passo -1 faça
3. MAX-HEAPIFY(A, i)
4. fim-para

```
HEAPSORT(A)
1. BUILD-MAX-HEAP(A)
2. para i ← comprimento[A] até 2 passo -1 faça
3.         A[1] ↔ A[i]
4.         tamanho-do-heap[A] ← tamanho-do-heap[A] – 1
5.         MAX-HEAPIFY(A, 1)
6. fim-para
```

HEAP-MAXIMUM(A)
1. retorna A[1]

HEAP-EXTRACT-MAX(A)
1. se *tamanho-do-heap*[A] < 1
2. então **error** “heap underflow”
3. fim-se
4. *max* ← A[1]
5. A[1] ← A[*tamanho-do-heap*[A]
6. *tamanho-do-heap*[A] ← *tamanho-do-heap*[A] – 1
7. MAX-HEAPIFY(A, 1)
8. retorna *max*

HEAP-INCREASE-KEY(A, i, *chave*)

1. se *chave* < A[i]
2. então **error** “nova chave é menor que chave atual”
3. fim-se
4. A[i] ← *chave*
5. enquanto i > 1 e A[PARENT(i)] faça
6. A[i] ↔ A[PARENT(i)]
7. i ← PARENT(i)
8. fim-enquanto

HEAP-EXTRACT-MAX(A)

1. *tamanho-do-heap*[A] ← *tamanho-do-heap*[A] + 1
2. A[*tamanho-do-heap*[A] ← $-\infty$
3. HEAP-INCREASE-KEY(A, *tamanho-do-heap*[A], *chave*)