Filas de Prioridades

Iarley Souza

June 2023

1 Questão 1

Escreva pseudocódigos para os procedimentos decreaseKey(A,i,newKey), minMoveDown(A, i), minMoveUp(A,i), increaseKey(A,i,newKey), minHeapInsert(A,key) e extractMinimum(A), que implementem uma fila de prioridade mínima com um heap de mínimo.

```
decreaseKey(A, i, newKey)
if newKey > A[i]:
   error "invalid key"
A[i] = newKey
minMoveUp(A, i)
```

```
minMoveUp(A, i)
p = i/2
while p >= 1 and A[i] < A[p]:
   aux = A[i]
   A[i] = A[p]
   A[p] = aux
   i = p
   p = p/2
```

```
increaseKey(A, i, newKey)
if newKey < A[i]:
    error "invalid key"
A[i] = newKey
minMoveDown(A, i)
```

```
minMoveDown(A, i)
while 2i <= A.heapSize:
    l = 2i
    r = 2i + 1
    smallest = i
    if l <= A.heapSize and A[l] < A[smallest]:
        smallest = l
    if r <= A.heapSize and A[r] < A[smallest]:
        smallest = r
    if smallest != i:
        aux = A[i]
        A[i] = A[smallest]
        A[smallest] = aux
        i = smallest
    else:
        break
```

```
minHeapInsert(A, key)
if A.heapSize >= A.length:
    error "heap overflow"
A.heapSize = A.heapSize + 1
A[A.heapSize] = key
minMoveUp(A, A.heapSize)
```

```
extractMinimum(A)
if A.heapSize < 1:
    error "heap underflow"
dados = A[1]
[1] = A[A.heapSize]
A.heapSize = A.heapSize - 1
minMoveDown(A, 1)
return dados
```

2 Questão 2

Seja A um heap de máximo. A operação HEAP-DELETE(A, i) elimina o item no nó i do heap A. Dê uma implementação correta de HEAP-DELETE(A,i) que seja executada no tempo $O(lgn)$ para um heap de máximo de n elementos.

```
HEAP-DELETE(A, i)
if i < 1 or i > A.heapSize:
    error "invalid index"
if i == A.heapSize:
    A.heapSize = A.heapSize - 1
else:
    A[i] = A[A.heapSize]
    A.heapSize = A.heapSize - 1
    maxMoveDown(A, i)
    maxMoveUp(A, i)
```

3 Questão 3

3.1 Como você representaria um heap d -ário de máximo em um vetor? Justifique.

Um heap d-ário é um vetor $A[1...n]$ que satisfaz a propriedade:

$$A\left[\left(\frac{i-2}{d}\right)+1\right] \geq A[i]$$

, sendo i:

$$2 \leq i \leq n$$

3.2 Qual é a altura de um heap d -ário de n elementos em termos de n e d ? Justifique.

Um heap d-ário de n elementos pode ser representado por uma árvore d-ária completa T de altura $h = \lfloor \log_d (d-1) \cdot n) \rfloor + 1$.

Prova:

Caso base: Para $n = 1$, e $d \geq 2$.

Se $n = 1$, e $d \geq 2$, então $h = \lfloor \log_d(d-1) \cdot 1) \rfloor + 1 = 1$, verdadeiro.

H.I:

Quando $n > 1$, suponha, que o resultado da fórmula seja verdadeira para todos os heaps de até $n - 1$ elementos. Seja T' a árvore obtida de T pela remoção de todos os k nós que estão no último nível de T. Logo T' é uma árvore cheia de $n' = n - k$ nós. Pela hipótese, $h(T') = \lfloor \log_d(d-1) \cdot n') \rfloor + 1$.

Como T' é uma árvore cheia, $n' = \frac{d^{m+1}-1}{d-1}$, concluímos $h(T') = m$.

Logo, $h(T') = m$. Além disso, $1 \leq k \leq n' + 1$.

Assim,

$h(T) = h(T') + 1$

$h(T) = m + 1$

$h(T) = (\lfloor \log_d(d-1) \cdot n') \rfloor + 1) + 1$, pois $n' = \frac{d^{m+1}-1}{d-1}$

$h(T) = (\lfloor \log_d(d-1) \cdot n') \rfloor + k) + 1$, pois $1 \leq k \leq n' + 1$

$h(T) = \lfloor \log_d(d-1) \cdot n) \rfloor + 1$, pois $n' = n - k$

3.3 Dê uma implementação eficiente de EXTRACT-MAXIMUM() em um heap de máximo d -ário. Analise seu tempo de execução em termos de d e n.

```
extractMaximum(A)
if A.heapSize < 1:
   error "heap underflow
dados = A[1]
A[1] = A[A.heapSize]
A.heapSize = A.heapSize - 1
moveDown(A, 1, d)
return dados
```

A função para extrair o valor de maior prioridade se mantém inalterada, porém são necessárias modificações na função moveDown. Segue abaixo a função moveDown já modificada:

```
moveDown(A, i, d)
while ((i*d) - (d-2)) <= A.heapSize:
   largest = i
   for (c = (i*d) - (d-2), c <= ((i*d) - (d-2))+(d-1), c++):
       if c > A.heapSize:
           break
       if A[c] > A[largest]:
           largest = c
   if largest != i:
       aux = A[i]
       A[i] = A[largest]
       A[largest] = aux
       i = largest
    else:
        break
```

Note que 'd' indica o número de filhos por nó do vetor.

Para que ocorra o swap entre dois nós (pai e filho) do vetor, é necessário verificar todos os filhos do nó pai, a fim de saber qual o maior nó k dentre os nós filhos, pois se k é o maior, todos os seus irmãos serão menores ou iguais a ele. Logo, k poderá assumir o lugar de seu pai, mantendo a propriedade do heap. Esse percurso nos filhos possui complexidade de pior caso $O(n-1)$, ou seja, quando todos os nós folhas são filhos do nó raíz.